

**GNU Awk**



# GAWK: Programmare efficacemente in AWK

---

Una Guida Utente per GNU Awk  
Edizione 4.1  
gennaio 2017

Arnold D. Robbins

“To boldly go where no man has gone before” (“Per arrivare là dove nessun uomo è mai giunto prima”) è un Marchio Registrato della Paramount Pictures Corporation.

Titolo originale:

**Gawk: Effective AWK Programming**

*A User's Guide for GNU Awk*

Published by **Free Software Foundation**

51 Franklin Street, Fifth Floor – Boston, MA 02110-1301 USA

Tel.: +1-617-542-5942 – Fax: +1-617-542-2652 – Email: [gnu@gnu.org](mailto:gnu@gnu.org)

URL: <http://www.gnu.org/>

ISBN 1-882114-28-0

Copyright © 1989, 1991, 1992, 1993, 1996–2005, 2007, 2009–2017

Free Software Foundation, Inc.

Traduzione e revisione:

Antonio Giovanni Colombo – [azc100\(chiocciola\)gmail\(punto\)com](mailto:azc100(chiocciola)gmail(punto)com)

Marco Curreli – [marcocurreli\(chiocciola\)tiscali\(punto\)it](mailto:marcocurreli(chiocciola)tiscali(punto)it)

(Italian Linux Documentation Project – <http://www.pluto.it/ildp>)

Pubblicato da: Free Software Foundation

Email: [gnu@gnu.org](mailto:gnu@gnu.org); URL: <http://www.gnu.org/>

e da: Italian Linux Documentation Project (ILDp)

Email: [ildp@pluto.it](mailto:ildp@pluto.it); URL: <http://www.pluto.it/ildp>

Copyright © 2017 – Free Software Foundation, Inc.

Questa è l'Edizione 4.1 di *GAWK: Programmare efficacemente in AWK: Una Guida Utente per GNU Awk*, per la versione 4.1.4 (o successiva) dell'implementazione GNU di AWK.

È garantito il permesso di copiare, distribuire e/o modificare questo documento seguendo i termini della Licenza per Documentazione Libera GNU, Versione 1.3 o ogni versione successiva pubblicata dalla Free Software Foundation; con le Sezioni Non Modificabili “GNU General Public License”, con i testi di copertina “Un Manuale GNU”, e con i testi di quarta di copertina come in (a) più avanti. Una copia della licenza è acclusa nella sezione intitolata “Licenza per Documentazione Libera GNU”.

- a. Il testo di quarta di copertina della FSF è: “È garantito il permesso di copiare e modificare questo manuale GNU.”

*Ai miei genitori, per il loro amore, e per lo splendido esempio che mi hanno dato.*

*A mia moglie, Miriam, per avermi reso completo. Grazie per aver costruito la tua vita insieme a me.*

*Ai nostri figli, Chana, Rivka, Nachum e Malka, per aver arricchito le nostre vite in misura incalcolabile.*



## Sommario abbreviato

Introduzione alla Terza Edizione .....	1
Introduzione alla Quarta Edizione .....	3
Prefazione .....	5

### Parte I: Il Linguaggio `awk`

1 Per iniziare con <code>awk</code> .....	17
2 Eseguire <code>awk</code> e <code>gawk</code> .....	33
3 Espressioni regolari .....	49
4 Leggere file in input .....	63
5 Stampare in output .....	95
6 Espressioni .....	115
7 Criteri di ricerca, azioni e variabili .....	145
8 Vettori in <code>awk</code> .....	177
9 Funzioni .....	195

### Parte II: Risoluzione di problemi con `awk`

10 Una libreria di funzioni <code>awk</code> .....	245
11 Programmi utili scritti in <code>awk</code> .....	281

### Parte III: Andare oltre `awk` con `gawk`

12 Funzionalità avanzate di <code>gawk</code> .....	331
13 Internazionalizzazione con <code>gawk</code> .....	349
14 Effettuare il debug dei programmi <code>awk</code> .....	361
15 Calcolo con precisione arbitraria con <code>gawk</code> .....	379
16 Scrivere estensioni per <code>gawk</code> .....	395

### Parte IV: Appendici

A L'evoluzione del linguaggio <code>awk</code> .....	461
B Installare <code>gawk</code> .....	479
C Note di implementazione .....	499
D Concetti fondamentali di programmazione .....	511
Glossario .....	515
Licenza Pubblica Generale GNU (GPL) .....	529
Licenza per Documentazione Libera GNU (FDL) .....	543
Indice analitico .....	551



# Sommario

Introduzione alla Terza Edizione .....	1
--	---

Introduzione alla Quarta Edizione .....	3
---	---

Prefazione .....	5
------------------	---

La storia di <b>gawk</b> e <b>awk</b> .....	6
Una rosa, con ogni altro nome... ..	6
Uso di questo libro .....	7
Convenzioni tipografiche .....	9
Angoli Bui .....	10
Breve storia del Progetto GNU e di questo libro .....	10
Come collaborare .....	11
Ringraziamenti .....	12

## Parte I: Il Linguaggio **awk**

1 Per iniziare con <b>awk</b> .....	17
-------------------------------------	----

1.1 Come iniziare a eseguire programmi <b>gawk</b> .....	17
1.1.1 Eseguire un breve programma <b>awk</b> <i>usa-e-getta</i> .....	17
1.1.2 Senza uso di file in input (input immesso da tastiera) .....	18
1.1.3 Eseguire programmi lunghi .....	19
1.1.4 Programmi <b>awk</b> da eseguire come <i>script</i> .....	19
1.1.5 Documentare programmi <b>gawk</b> . .....	20
1.1.6 Uso di apici nella shell. ....	21
1.1.6.1 Doppi apici in file <i>.BAT</i> Windows .....	23
1.2 File-dati per gli esempi .....	23
1.3 Alcuni esempi molto semplici .....	24
1.4 Un esempio che usa due regole .....	26
1.5 Un esempio più complesso .....	27
1.6 Istruzioni e righe in <b>awk</b> .....	28
1.7 Altre funzionalità di <b>awk</b> .....	30
1.8 Quando usare <b>gawk</b> .....	30
1.9 Sommario .....	31

2 Eseguire <b>awk</b> e <b>gawk</b> .....	33
---	----

2.1 Come eseguire <b>awk</b> .....	33
2.2 Opzioni sulla riga di comando .....	33
2.3 Altri argomenti della riga di comando .....	40
2.4 Come specificare lo standard input insieme ad altri file .....	41
2.5 Le variabili d'ambiente usate da <b>gawk</b> .....	41
2.5.1 Ricerca di programmi <b>awk</b> in una lista di directory. ....	42

2.5.2	Ricerca di librerie condivise <b>awk</b> su varie directory. ....	43
2.5.3	Le variabili d'ambiente. ....	43
2.6	Il codice di ritorno all'uscita da <b>gawk</b> ....	45
2.7	Come includere altri file nel proprio programma. ....	45
2.8	Caricare librerie condivise nel proprio programma. ....	47
2.9	Opzioni e/o funzionalità obsolete ....	47
2.10	Opzioni e funzionalità non documentate ....	47
2.11	Sommario ....	47
<b>3</b>	<b>Espressioni regolari</b> ....	<b>49</b>
3.1	Uso di espressioni regolari ....	49
3.2	Sequenze di protezione. ....	50
3.3	Operatori di espressioni regolari ....	52
3.4	Usare espressioni tra parentesi quadre. ....	55
3.5	Quanto è lungo il testo individuato? ....	57
3.6	Usare <i>regexp</i> dinamiche ....	57
3.7	Operatori <i>regexp</i> propri di <b>gawk</b> ....	59
3.8	Fare confronti ignorando maiuscolo/minuscolo ....	60
3.9	Sommario ....	62
<b>4</b>	<b>Leggere file in input</b> ....	<b>63</b>
4.1	Controllare come i dati sono suddivisi in record ....	63
4.1.1	Come <b>awk</b> standard divide i record. ....	63
4.1.2	Divisione dei record con <b>gawk</b> ....	65
4.2	Un'introduzione ai campi ....	67
4.3	Numeri di campo variabili ....	68
4.4	Cambiare il contenuto di un campo ....	69
4.5	Specificare come vengono separati i campi ....	71
4.5.1	Lo spazio vuoto normalmente separa i campi ....	72
4.5.2	Usare <i>regexp</i> come separatori di campo ....	72
4.5.3	Fare di ogni carattere un campo separato ....	73
4.5.4	Impostare FS dalla riga di comando ....	74
4.5.5	Fare di una riga intera un campo solo ....	75
4.5.6	Sommario sulla separazione dei campi ....	76
4.6	Leggere campi di larghezza costante ....	77
4.7	Definire i campi in base al contenuto. ....	79
4.8	Record su righe multiple. ....	80
4.9	Richiedere input usando <b>getline</b> ....	83
4.9.1	Usare <b>getline</b> senza argomenti. ....	83
4.9.2	Usare <b>getline</b> in una variabile ....	85
4.9.3	Usare <b>getline</b> da un file. ....	85
4.9.4	Usare <b>getline</b> in una variabile da un file ....	86
4.9.5	Usare <b>getline</b> da una <i>pipe</i> ....	86
4.9.6	Usare <b>getline</b> in una variabile da una <i>pipe</i> ....	87
4.9.7	Usare <b>getline</b> da un coprocesso ....	88
4.9.8	Usare <b>getline</b> in una variabile da un coprocesso ....	88
4.9.9	Cose importanti da sapere riguardo a <b>getline</b> ....	88

4.9.10	Sommario delle varianti di <code>getline</code> .....	89
4.10	Leggere input entro un tempo limite.....	90
4.11	Elaborare ulteriore input dopo certi errori di I/O.....	91
4.12	Directory sulla riga di comando.....	92
4.13	Sommario di Input.....	92
4.14	Esercizi.....	93
<b>5</b>	<b>Stampare in output.....</b>	<b>95</b>
5.1	L'istruzione <code>print</code> .....	95
5.2	Esempi di istruzioni <code>print</code> .....	95
5.3	I separatori di output e come modificarli.....	97
5.4	Controllare l'output di numeri con <code>print</code> .....	98
5.5	Usare l'istruzione <code>printf</code> per stampe sofisticate.....	98
5.5.1	Sintassi dell'istruzione <code>printf</code> .....	98
5.5.2	Lettere di controllo del formato.....	99
5.5.3	Modificatori per specifiche di formato <code>printf</code> .....	101
5.5.4	Esempi d'uso di <code>printf</code> .....	103
5.6	Ridirigere l'output di <code>print</code> e <code>printf</code> .....	104
5.7	File speciali per flussi standard di dati pre-aperti.....	107
5.8	Nomi-file speciali in <code>gawk</code> .....	108
5.8.1	Accedere ad altri file aperti con <code>gawk</code> .....	108
5.8.2	File speciali per comunicazioni con la rete.....	109
5.8.3	Avvertimenti speciali sui nomi-file.....	109
5.9	Chiudere ridirezioni in input e in output.....	109
5.10	Abilitare continuazione dopo errori in output.....	112
5.11	Sommario.....	113
5.12	Esercizi.....	114
<b>6</b>	<b>Espressioni.....</b>	<b>115</b>
6.1	Costanti, variabili e conversioni.....	115
6.1.1	Espressioni costanti.....	115
6.1.1.1	Costanti numeriche e stringhe.....	115
6.1.1.2	Numeri ottali ed esadecimali.....	115
6.1.1.3	Costanti fornite tramite espressioni regolari.....	117
6.1.2	Usare espressioni regolari come costanti.....	117
6.1.2.1	Costanti <i>regex</i> normali in <code>awk</code> .....	117
6.1.2.2	Costanti <i>regex</i> fortemente tipizzate.....	118
6.1.3	Variabili.....	119
6.1.3.1	Usare variabili in un programma.....	119
6.1.3.2	Assegnare una variabile dalla riga di comando.....	120
6.1.4	Conversione di stringhe e numeri.....	121
6.1.4.1	Come <code>awk</code> converte tra stringhe e numeri.....	121
6.1.4.2	Le localizzazioni possono influire sulle conversioni ...	122
6.2	Operatori: fare qualcosa coi valori.....	123
6.2.1	Operatori aritmetici.....	123
6.2.2	Concatenazione di stringhe.....	124
6.2.3	Espressioni di assegnamento.....	126

6.2.4	Operatori di incremento e di decremento .....	128
6.3	Valori e condizioni di verità .....	130
6.3.1	Vero e falso in <b>awk</b> .....	130
6.3.2	Tipi di variabile ed espressioni di confronto .....	130
6.3.2.1	Tipo stringa rispetto a tipo numero .....	131
6.3.2.2	Operatori di confronto .....	133
6.3.2.3	Confronto tra stringhe usando l'ordine di collazione locale .....	135
6.3.3	Espressioni booleane .....	136
6.3.4	Espressioni condizionali .....	137
6.4	Chiamate di funzione .....	138
6.5	Precedenza degli operatori (Come si nidificano gli operatori) ..	140
6.6	Il luogo fa la differenza .....	141
6.7	Sommario .....	142
<b>7</b>	<b>Criteri di ricerca, azioni e variabili .....</b>	<b>145</b>
7.1	Elementi di un criterio di ricerca .....	145
7.1.1	Espressioni regolari come criteri di ricerca .....	145
7.1.2	Espressioni come criteri di ricerca .....	146
7.1.3	Specificare intervalli di record con i criteri di ricerca .....	147
7.1.4	I criteri di ricerca speciali <b>BEGIN</b> ed <b>END</b> .....	148
7.1.4.1	Azioni di inizializzazione e pulizia .....	148
7.1.4.2	Input/Output dalle regole <b>BEGIN</b> ed <b>END</b> .....	149
7.1.5	I criteri di ricerca speciali <b>BEGINFILE</b> ed <b>ENDFILE</b> .....	150
7.1.6	Il criterio di ricerca vuoto .....	151
7.2	Usare variabili di shell in programmi .....	151
7.3	Azioni .....	152
7.4	Istruzioni di controllo nelle azioni .....	153
7.4.1	L'istruzione <b>if-else</b> .....	153
7.4.2	L'istruzione <b>while</b> .....	154
7.4.3	L'istruzione <b>do-while</b> .....	154
7.4.4	L'istruzione <b>for</b> .....	155
7.4.5	L'istruzione <b>switch</b> .....	156
7.4.6	L'istruzione <b>break</b> .....	157
7.4.7	L'istruzione <b>continue</b> .....	158
7.4.8	L'istruzione <b>next</b> .....	159
7.4.9	L'istruzione <b>nextfile</b> .....	160
7.4.10	L'istruzione <b>exit</b> .....	161
7.5	Variabili predefinite .....	162
7.5.1	Variabili predefinite modificabili per controllare <b>awk</b> .....	162
7.5.2	Variabili predefinite con cui <b>awk</b> fornisce informazioni ....	165
7.5.3	Usare <b>ARGC</b> e <b>ARGV</b> .....	172
7.6	Sommario .....	174

<b>8</b>	<b>Vettori in awk</b>	<b>177</b>
8.1	Informazioni di base sui vettori	177
8.1.1	Introduzione ai vettori	177
8.1.2	Come esaminare un elemento di un vettore	179
8.1.3	Assegnare un valore a elementi di un vettore	180
8.1.4	Esempio semplice di vettore	180
8.1.5	Visitare tutti gli elementi di un vettore	181
8.1.6	Visita di vettori in ordine predefinito con <b>gawk</b>	182
8.2	Usare numeri per indicizzare i vettori	185
8.3	Usare variabili non inizializzate come indici	186
8.4	L'istruzione <b>delete</b>	187
8.5	Vettori multidimensionali	188
8.5.1	Visitare vettori multidimensionali	189
8.6	Vettori di vettori	190
8.7	Sommario	192
<b>9</b>	<b>Funzioni</b>	<b>195</b>
9.1	Funzioni predefinite	195
9.1.1	Chiamare funzioni predefinite	195
9.1.2	Funzioni numeriche	196
9.1.3	Funzioni di manipolazione di stringhe	198
9.1.3.1	Ulteriori dettagli su '\ ' e '&'	
	con <b>sub()</b> , <b>gsub()</b> e <b>gensub()</b>	207
9.1.4	Funzioni di Input/Output	210
9.1.5	Funzioni per gestire marcature temporali	214
9.1.6	Funzioni per operazioni di manipolazione bit	219
9.1.7	Funzioni per conoscere il tipo di una variabile	223
9.1.8	Funzioni per tradurre stringhe	224
9.2	Funzioni definite dall'utente	224
9.2.1	Come scrivere definizioni e cosa significano	224
9.2.2	Un esempio di definizione di funzione	226
9.2.3	Chiamare funzioni definite dall'utente	228
9.2.3.1	Scrivere una chiamata di funzione	228
9.2.3.2	Variabili locali e globali	228
9.2.3.3	Passare parametri di	
	funzione per valore o per riferimento	231
9.2.4	L'istruzione <b>return</b>	232
9.2.5	Funzioni e loro effetti sul tipo di una variabile	234
9.3	Chiamate indirette di funzione	234
9.4	Sommario	240

## Parte II: Risoluzione di problemi con awk

<b>10</b>	<b>Una libreria di funzioni awk</b>	<b>245</b>
10.1	Dare un nome a variabili globali in funzioni di libreria	246
10.2	Programmazione di tipo generale	247
10.2.1	Conversione di stringhe in numeri	247
10.2.2	Asserzioni	249
10.2.3	Arrotondamento di numeri	250
10.2.4	Il generatore di numeri casuali Cliff	251
10.2.5	Tradurre tra caratteri e numeri	251
10.2.6	Trasformare un vettore in una sola stringa	253
10.2.7	Gestione dell'ora del giorno	254
10.2.8	Leggere un intero file in una sola volta	255
10.2.9	Stringhe con apici da passare alla shell	257
10.3	Gestione di file-dati	257
10.3.1	Trovare i limiti dei file-dati	258
10.3.2	Rileggere il file corrente	259
10.3.3	Controllare che i file-dati siano leggibili	261
10.3.4	Ricerca di file di lunghezza zero	261
10.3.5	Trattare assegnamenti di variabile come nomi-file	262
10.4	Elaborare opzioni specificate sulla riga di comando	263
10.5	Leggere la lista degli utenti	268
10.6	Leggere la lista dei gruppi	272
10.7	Attraversare vettori di vettori	277
10.8	Riassunto	279
10.9	Esercizi	279
<b>11</b>	<b>Programmi utili scritti in awk</b>	<b>281</b>
11.1	Come eseguire i programmi di esempio	281
11.2	Reinventare la ruota per divertimento e profitto	281
11.2.1	Ritagliare campi e colonne	282
11.2.2	Ricerca espressioni regolari nei file	286
11.2.3	Stampare informazioni sull'utente	290
11.2.4	Suddividere in pezzi un file grosso	292
11.2.5	Inviare l'output su più di un file	294
11.2.6	Stampare righe di testo non duplicate	296
11.2.7	Contare cose	300
11.3	Un paniere di programmi awk	302
11.3.1	Trovare parole duplicate in un documento	302
11.3.2	Un programma di sveglia	303
11.3.3	Rimpiazzare o eliminare caratteri	305
11.3.4	Stampare etichette per lettere	308
11.3.5	Generare statistiche sulla frequenza d'uso delle parole	309
11.3.6	Eliminare duplicati da un file non ordinato	311
11.3.7	Estrarre programmi da un file sorgente Texinfo	312
11.3.8	Un semplice editor di flusso	316
11.3.9	Una maniera facile per usare funzioni di libreria	317
11.3.10	Trovare anagrammi da una lista di parole	324
11.3.11	E ora per qualcosa di completamente differente	326

11.4	Sommario .....	326
11.5	Esercizi .....	326

## Parte III: Andare oltre awk con gawk

### 12 Funzionalità avanzate di gawk..... 331

12.1	Consentire dati di input non decimali .....	331
12.2	Controllare la visita di un vettore e il suo ordinamento .....	332
12.2.1	Controllare visita vettori .....	332
12.2.2	Ordinare valori e indici di un vettore con <b>gawk</b> .....	336
12.3	Comunicazioni bidirezionali con un altro processo .....	339
12.4	Usare <b>gawk</b> per la programmazione di rete .....	341
12.5	Profilare i propri programmi <b>awk</b> .....	343
12.6	Sommario .....	347

### 13 Internazionalizzazione con gawk..... 349

13.1	Internazionalizzazione e localizzazione .....	349
13.2	Il comando GNU <b>gettext</b> .....	349
13.3	Internazionalizzare programmi <b>awk</b> .....	352
13.4	Traduzione dei programmi <b>awk</b> .....	354
13.4.1	Estrarre stringhe marcate .....	354
13.4.2	Riordinare argomenti di <b>printf</b> .....	354
13.4.3	Problemi di portabilità a livello di <b>awk</b> .....	356
13.5	Un semplice esempio di internazionalizzazione .....	357
13.6	<b>gawk</b> stesso è internazionalizzato .....	358
13.7	Sommario .....	358

### 14 Effettuare il debug dei programmi awk..... 361

14.1	Introduzione al debugger di <b>gawk</b> .....	361
14.1.1	Generalità sul debug .....	361
14.1.2	Concetti fondamentali sul debug .....	361
14.1.3	Il debug di <b>awk</b> .....	362
14.2	Esempio di sessione di debug di <b>gawk</b> .....	363
14.2.1	Come avviare il debugger .....	363
14.2.2	Trovare il bug .....	363
14.3	I principali comandi di debug .....	366
14.3.1	Controllo dei punti d'interruzione .....	367
14.3.2	Controllo di esecuzione .....	368
14.3.3	Vedere e modificare dati .....	370
14.3.4	Lavorare con lo stack .....	371
14.3.5	Ottenere informazioni sullo stato del programma e del debugger .....	372
14.3.6	Comandi vari del debugger .....	374
14.4	Supporto per Readline .....	376
14.5	Limitazioni .....	376
14.6	Sommario .....	377

## 15 Calcolo con precisione arbitraria con gawk .. 379

15.1	Una descrizione generale dell'aritmetica del computer .....	379
15.2	Altre cose da sapere .....	380
15.3	Funzionalità per il calcolo a precisione arbitraria in gawk .....	382
15.4	Calcolo in virgola mobile: <i>Caveat Emptor!</i> .....	383
15.4.1	La matematica in virgola mobile non è esatta .....	383
15.4.1.1	Molti numeri non possono essere rappresentati esattamente .....	383
15.4.1.2	Fare attenzione quando si confrontano valori .....	384
15.4.1.3	Gli errori diventano sempre maggiori .....	384
15.4.2	Ottenere la precisione voluta .....	385
15.4.3	Tentare di aggiungere bit di precisione e arrotondare ....	386
15.4.4	Impostare la precisione .....	386
15.4.5	Impostare la modalità di arrotondamento .....	387
15.5	Aritmetica dei numeri interi a precisione arbitraria con gawk ..	389
15.6	Confronto tra standard e uso corrente .....	391
15.7	Sommario .....	393

## 16 Scrivere estensioni per gawk .. 395

16.1	Cos'è un'estensione .....	395
16.2	Tipo di licenza delle estensioni .....	395
16.3	Una panoramica sul funzionamento ad alto livello .....	396
16.4	Una descrizione completa dell'API .....	398
16.4.1	Introduzione alle funzioni dell'API .....	398
16.4.2	I tipi di dati di impiego generale .....	400
16.4.3	Funzioni per allocare memoria e macro di servizio .....	403
16.4.4	Funzioni per creare valori .....	404
16.4.5	Funzioni di registrazione .....	405
16.4.5.1	Registrare funzioni di estensione .....	406
16.4.5.2	Registrare una funzione <i>exit callback</i> .....	408
16.4.5.3	Registrare una stringa di versione per un'estensione ..	408
16.4.5.4	Analizzatori di input personalizzati .....	408
16.4.5.5	Registrare un processore di output .....	412
16.4.5.6	Registrare un processore bidirezionale .....	414
16.4.6	Stampare messaggi dalle estensioni .....	415
16.4.7	Funzioni per aggiornare <b>ERRNO</b> .....	415
16.4.8	Richiedere valori .....	416
16.4.9	Accedere ai parametri e aggiornarli .....	416
16.4.10	Accedere alla Tabella dei simboli .....	417
16.4.10.1	Accedere alle variabili per nome e aggiornarle .....	417
16.4.10.2	Accedere alle variabili per "cookie" e aggiornarle ..	417
16.4.10.3	Creare e usare valori nascosti .....	419
16.4.11	Manipolazione di vettori .....	421
16.4.11.1	Tipi di dati per i vettori .....	421
16.4.11.2	Funzioni per lavorare coi vettori .....	422
16.4.11.3	Lavorare con tutti gli elementi di un vettore .....	424
16.4.11.4	Come creare e popolare vettori .....	427

16.4.12	Accedere alle ridirezioni e modificarle .....	430
16.4.13	Variabili fornite dall'API .....	431
16.4.13.1	Costanti e variabili della versione dell'API .....	431
16.4.13.2	Variabili informative .....	432
16.4.14	Codice predefinito di interfaccia API .....	432
16.4.15	Modifiche dalla versione 1 dell'API .....	434
16.5	Come <b>gawk</b> trova le estensioni compilate .....	434
16.6	Esempio: alcune funzioni per i file .....	434
16.6.1	Usare <b>chdir()</b> e <b>stat()</b> .....	435
16.6.2	Codice C per eseguire <b>chdir()</b> e <b>stat()</b> .....	437
16.6.3	Integrare le estensioni .....	443
16.7	Le estensioni di esempio incluse nella distribuzione <b>gawk</b> ....	445
16.7.1	Funzioni relative ai file .....	445
16.7.2	Un'interfaccia a <b>fnmatch()</b> .....	448
16.7.3	Un'interfaccia a <b>fork()</b> , <b>wait()</b> , e <b>waitpid()</b> .....	449
16.7.4	Consentire la modifica in loco dei file .....	449
16.7.5	Caratteri e valori numerici: <b>ord()</b> e <b>chr()</b> .....	451
16.7.6	Leggere directory .....	451
16.7.7	Invertire la stringa in output .....	452
16.7.8	Esempio di I/O bidirezionale .....	452
16.7.9	Scaricare e ricaricare un vettore .....	453
16.7.10	Leggere un intero file in una stringa .....	454
16.7.11	Funzioni dell'estensione <b>time</b> .....	454
16.7.12	Test per la API .....	455
16.8	Il progetto <b>gawkextlib</b> .....	455
16.9	Sommario .....	456
16.10	Esercizi .....	457

## Parte IV: Appendici

### Appendice A L'evoluzione del linguaggio **awk** .. 461

A.1	Differenze importanti tra V7 e System V Release 3.1 .....	461
A.2	Differenze tra le versioni System V Release 3.1 e SVR4 .....	462
A.3	Differenze tra versione SVR4 e POSIX di <b>awk</b> .....	463
A.4	Estensioni nell' <b>awk</b> di Brian Kernighan .....	463
A.5	Estensioni di <b>gawk</b> non in POSIX <b>awk</b> .....	464
A.6	Storia delle funzionalità di <b>gawk</b> .....	466
A.7	Sommario Estensioni Comuni .....	473
A.8	Intervalli <b>regex</b> e localizzazione: una lunga e triste storia ....	474
A.9	I principali contributori a <b>gawk</b> .....	475
A.10	Sommario .....	478

## **Appendice B Installare gawk..... 479**

B.1	La distribuzione di <b>gawk</b> .....	479
B.1.1	Ottenere la distribuzione di <b>gawk</b> .....	479
B.1.2	Scompattare la distribuzione .....	479
B.1.3	Contenuti della distribuzione <b>gawk</b> .....	480
B.2	Compilare e installare <b>gawk</b> su sistemi di tipo Unix .....	483
B.2.1	Compilare <b>gawk</b> per sistemi di tipo Unix .....	483
B.2.2	File di inizializzazione della shell .....	484
B.2.3	Ulteriori opzioni di configurazione .....	484
B.2.4	Il processo di configurazione .....	485
B.3	Installazione su altri Sistemi Operativi .....	486
B.3.1	Installazione su MS-Windows .....	486
B.3.1.1	Installare una distribuzione predisposta per sistemi MS-Windows .....	486
B.3.1.2	Compilare <b>gawk</b> per sistemi operativi di PC .....	486
B.3.1.3	Usare <b>gawk</b> su sistemi operativi PC .....	487
B.3.1.4	Usare <b>gawk</b> in ambiente Cygwin .....	488
B.3.1.5	Usare <b>gawk</b> in ambiente MSYS .....	488
B.3.2	Compilare e installare <b>gawk</b> su Vax/VMS e OpenVMS ...	488
B.3.2.1	Compilare <b>gawk</b> su VMS .....	488
B.3.2.2	Compilare estensioni dinamiche di <b>gawk</b> in VMS ...	489
B.3.2.3	Installare <b>gawk</b> su VMS .....	490
B.3.2.4	Eseguire <b>gawk</b> su VMS .....	491
B.3.2.5	Il progetto VMS GNV .....	492
B.3.2.6	Vecchia versione di <b>gawk</b> su sistemi VMS .....	492
B.4	Segnalazione di problemi e bug .....	493
B.4.1	Segnalare Bug .....	493
B.4.2	Non segnalare bug a USENET! .....	494
B.4.3	Notificare problemi per versioni non-Unix .....	494
B.5	Altre implementazioni di <b>awk</b> liberamente disponibili .....	494
B.6	Sommario .....	497

## **Appendice C Note di implementazione..... 499**

C.1	Compatibilità all'indietro e debug .....	499
C.2	Fare aggiunte a <b>gawk</b> .....	499
C.2.1	Accedere al deposito dei sorgenti Git di <b>gawk</b> .....	499
C.2.2	Aggiungere nuove funzionalità .....	500
C.2.3	Portare <b>gawk</b> su un nuovo Sistema Operativo .....	502
C.2.4	Perché i file generati sono tenuti in Git .....	503
C.3	Probabili estensioni future .....	505
C.4	Alcune limitazioni dell'implementazione .....	505
C.5	Note di progetto dell'estensione API .....	506
C.5.1	Problemi con le vecchie estensioni .....	506
C.5.2	Obiettivi per un nuovo meccanismo .....	507
C.5.3	Altre scelte progettuali .....	508
C.5.4	Possibilità di sviluppo futuro .....	509
C.6	Compatibilità per le vecchie estensioni .....	509

C.7	Sommario .....	510
<b>Appendice D Concetti fondamentali di</b>		
	<b>programmazione.....</b>	<b>511</b>
D.1	Quel che fa un programma .....	511
D.2	Valore dei dati in un computer .....	513
<b>Glossario .....</b>		
<b>Licenza Pubblica Generale GNU (GPL) .....</b>		<b>529</b>
<b>Licenza per Documentazione Libera GNU (FDL) ..</b>		<b>543</b>
	ADDENDUM: Come usare questa licenza per i vostri documenti ...	550
<b>Indice analitico.....</b>		<b>551</b>



## Introduzione alla Terza Edizione

Arnold Robbins e io siamo buoni amici. Ci siamo conosciuti nel 1990 per un insieme di circostanze—e per il nostro linguaggio di programmazione preferito, AWK. Tutto era iniziato un paio d’anni prima. Avevo appena iniziato un nuovo lavoro e avevo notato un computer Unix scollegato che giaceva in un angolo. Nessuno sapeva come usarlo, tanto meno io. Comunque, qualche giorno più tardi, stava funzionando, con me come `root` e solo e unico utente. Quel giorno, iniziai la transizione da statistico a programmatore Unix.

In uno dei miei giri per biblioteche e librerie alla ricerca di libri sullo Unix, trovai il libro, dalla copertina grigia, su AWK, noto anche come Alfred V. Aho, Brian W. Kernighan e Peter J. Weinberger, *The AWK Programming Language*, (Addison-Wesley, 1988). Il semplice paradigma di programmazione di AWK —trovare un’espressione di ricerca nell’input e di conseguenza compiere un’azione—riduceva spesso complesse e tediose manipolazioni di dati a poche righe di codice. Ero entusiasta di cimentarmi nella programmazione in AWK.

Ahimè, l’`awk` sul mio computer era una versione limitata del linguaggio descritto nel libro grigio. Scoprii che il mio computer aveva il “vecchio `awk`” mentre il libro descriveva il “nuovo `awk`.” Imparai che non era un caso isolato; la vecchia versione si rifiutava di farsi da parte o di cedere il suo nome. Se un sistema aveva un nuovo `awk`, questo era chiamato invariabilmente `nawk`, e pochi sistemi lo avevano. Il miglior modo per ottenere un nuovo `awk` era quello di scaricare via `ftp` il codice sorgente di `gawk` da `prep.ai.mit.edu`. `gawk` era una versione del nuovo `awk` scritta da David Trueman e Arnold, e disponibile sotto la GNU General Public License.

Per inciso, ora non è più così difficile trovare un nuovo `awk`. `gawk` viene fornito con GNU/Linux, e si possono scaricare i binari e il codice sorgente per quasi tutti i sistemi; mia moglie usa `gawk` nella sua stazione di lavoro VMS.

Il mio sistema Unix non era inizialmente collegato a una presa di corrente; a maggior ragione non era collegato a una rete. Così, ignaro dell’esistenza di `gawk` e in generale della comunità di Unix, e desiderando un nuovo `awk`, ne scrissi uno mio, chiamato `mawk`. Prima di aver finito, scoprii l’esistenza di `gawk`, ma era troppo tardi per fermarmi, così alla fine inviai un messaggio a un newsgroup `comp.sources`.

Qualche giorno dopo ricevetti un cordiale messaggio di posta elettronica da Arnold che si presentava. Propose di scambiarsi progetti e algoritmi, e allegò una bozza dello standard POSIX, che mi permise di aggiornare `mawk` per includere le estensioni al linguaggio aggiunte dopo la pubblicazione di *The AWK Programming Language*.

Francamente, se i nostri ruoli fossero stati invertiti, io non sarei stato così disponibile e probabilmente non ci saremmo mai incontrati. Sono felice che l’incontro sia avvenuto. Lui è un vero esperto tra gli esperti di AWK e una persona squisita. Arnold mette a disposizione della Free Software Foundation parti significative della sua esperienza e del suo tempo.

Questo libro è il manuale di riferimento di `gawk`, ma sostanzialmente è un libro sulla programmazione in AWK che interesserà un vasto pubblico. È un riferimento completo al linguaggio AWK come definito dalla versione del 1987 di Bell Laboratories e codificato nelle POSIX Utilities standard del 1992.

D’altra parte, un programmatore AWK alle prime armi può studiare una quantità di programmi pratici che permettono di apprezzare la potenza dei concetti di base di AWK: flusso di controllo guidato dai dati, ricerca di corrispondenze tramite espressioni regolari e

## 2 GAWK: Programmare efficacemente in AWK

vettori associativi. Chi desidera qualcosa di nuovo può provare l'interfaccia di **gawk** verso i protocolli di rete attraverso i file speciali `/inet`.

I programmi in questo libro evidenziano come un programma AWK sia generalmente molto più piccolo e veloce da sviluppare di uno equivalente scritto in C. Di conseguenza, è spesso conveniente creare un prototipo di un algoritmo o di un progetto in AWK per arrivare a eseguirlo in breve tempo e scoprire prima i problemi che possono presentarsi. Spesso, l'efficienza di questa versione iniziale interpretata è sufficiente e il prototipo AWK diventa il prodotto finale.

Il nuovo comando **pgawk** (profiling **gawk**) produce conteggi sull'esecuzione delle istruzioni del programma. Recentemente ho fatto un tentativo con un algoritmo che, a fronte di  $n$  righe di input, produceva il risultato in un tempo  $\sim Cn^2$ , mentre in teoria avrebbe dovuto terminare in un tempo  $\sim Cn \log n$ . Dopo qualche minuto di attenta lettura del profilo in **awkprof.out**, ho ricondotto il problema a una singola riga di codice. **pgawk** è una gradita integrazione ai miei strumenti di programmatore.

Arnold ha condensato in questo libro oltre un decennio di esperienza nell'uso di programmi AWK e nello sviluppo di **gawk**. Se si vuole usare AWK o imparare ad usarlo, è consigliabile leggere questo libro.

Michael Brennan  
Autore di **mawk**  
Marzo 2001

## Introduzione alla Quarta Edizione

Ci sono cose che non cambiano. Tredici anni fa scrivevo: “Se si vuole usare AWK o imparare ad usarlo, è consigliabile leggere questo libro.” Era vero allora e rimane vero anche oggi.

Imparare a usare un linguaggio di programmazione richiede qualcosa di più che padroneggiarne la sintassi. Occorre comprendere come usare le funzionalità del linguaggio per risolvere problemi pratici di programmazione. Uno dei punti più importanti di questo libro è che fornisce molti esempi che mostrano come utilizzare AWK.

Altre cose, invece, cambiano. I nostri computer sono diventati molto più veloci e la loro memoria è molto più estesa. Per questa ragione, la velocità di esecuzione e l’uso efficiente della memoria, caratteristiche di un linguaggio di livello elevato, hanno minore rilevanza. Scrivere un programma prototipo in AWK per poi riscriverlo in C per migliorare l’utilizzo delle risorse capita sempre meno, perché sempre più spesso il prototipo è abbastanza veloce anche per essere messo in produzione.

Naturalmente, ci sono tipi di calcoli che sono effettuati più agevolmente da programmi scritti in C o C++. Con **gawk** 4.1 e successive versioni, non è necessario decidere se scrivere un programma in AWK oppure in C/C++. Si può scrivere buona parte del programma in AWK e le parti che richiedono specificamente il C/C++ possono essere scritte in C/C++ e quindi il tutto può essere eseguito come un programma unico, con il modulo **gawk** che carica dinamicamente il modulo C/C++ in fase di esecuzione. Il **Capitolo 16 [Scrivere estensioni per gawk]**, **pagina 395**, spiega la procedura in gran dettaglio, e, come prevedibile, riporta molti esempi che sono di aiuto per approfondire anche gli aspetti più complessi.

È per me un piacere programmare in AWK ed è stato divertente (ri)leggere questo libro. Penso che sarà lo stesso per voi.

Michael Brennan  
Autore di **mawk**  
Ottobre 2014



## Prefazione

Lavorando con file di testo capita di dover eseguire alcuni tipi ripetitivi di operazioni. Si potrebbe voler estrarre alcune righe e scartare il resto, o fare modifiche laddove siano verificate certe condizioni, lasciando inalterato il resto del file. Questi compiti risultano spesso più agevoli usando **awk**. Il programma di utilità **awk** interpreta un linguaggio di programmazione specializzato che rende facile eseguire semplici attività di riformattazione di dati.

L'implementazione GNU di **awk** è chiamata **gawk**; se invocato con le opzioni o con le variabili d'ambiente appropriate, (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), è pienamente compatibile con le specifiche POSIX<sup>1</sup> del linguaggio **awk** e con la versione Unix di **awk** mantenuta da Brian Kernighan. Ciò implica che tutti i programmi **awk** scritti correttamente dovrebbero funzionare con **gawk**. Perciò nella maggior parte dei casi non si distingue tra **gawk** e altre implementazioni di **awk**.

Usando **awk** potete:

- Gestire piccole basi di dati personali
- Generare rapporti
- Validare dati
- Produrre indici ed effettuare altre operazioni per la preparazione di documenti
- Sperimentare algoritmi che possono essere adattati in seguito ad altri linguaggi per computer

Inoltre, **gawk** fornisce strumenti che rendono facile:

- Estrarre frammenti di dati per l'elaborazione
- Ordinare dati
- Effettuare semplici comunicazioni di rete
- Creare il profilo di esecuzione ed effettuare il debug di programmi **awk**.
- Estendere il linguaggio con funzioni scritte in C o C++.

Questo libro spiega il linguaggio **awk** e come lo si può usare efficacemente. È richiesta una familiarità coi comandi di sistema di base, come **cat** e **ls**,<sup>2</sup> così come con le funzionalità di base della shell, come la ridirezione, l'input/output (I/O) e le *pipe*.

Implementazioni del linguaggio **awk** sono disponibili per diversi sistemi operativi di computer. Questo libro, oltre a descrivere il linguaggio **awk** in generale, descrive anche la specifica implementazione di **awk** chiamata **gawk** (che sta per "GNU **awk**"). **gawk** funziona su una vasta gamma di sistemi Unix, dai PC basati su architettura Intel fino a sistemi di potenza molto maggiore. **gawk** è stato portato anche su Mac OS X, Microsoft Windows (tutte le versioni), e OpenVMS.<sup>3</sup>

<sup>1</sup> Lo standard POSIX 2008 è accessibile in rete all'indirizzo <http://www.opengroup.org/onlinepubs/9699919799/>.

<sup>2</sup> Questi programmi di utilità sono disponibili sui sistemi conformi a POSIX, come pure sui sistemi tradizionali basati su Unix. Se si usa qualche altro sistema operativo, si deve comunque avere familiarità con i concetti di ridirezione I/O e di *pipe*.

<sup>3</sup> Qualche altro sistema operativo obsoleto su cui **gawk** era stato portato non è più mantenuto e il codice specifico per quei sistemi è stato rimosso.

## La storia di gawk e awk

### Ricetta per un linguaggio di programmazione

1 parte di **egrep**    1 parte di **snobol**  
 2 parti di **ed**        3 parti di **C**

Mescolare bene tutte le parti usando **lex** e **yacc**. Preparare una concisa documentazione e distribuire.

Dopo otto anni, aggiungere un'altra parte di **egrep** e altre due parti di **C**. Documentare molto bene e distribuire.

Il nome **awk** deriva dalle iniziali dei suoi progettisti: Alfred V. Aho, Peter J. Weinberger e Brian W. Kernighan. La versione originale di **awk** fu scritta nel 1977 negli AT&T Bell Laboratories. Nel 1985, una nuova versione rese il linguaggio di programmazione più potente, introducendo le funzioni definite dall'utente, flussi di input multipli ed espressioni regolari calcolate. Questa nuova versione ebbe larga diffusione con Unix System V Release 3.1 (1987). La versione in System V Release 4 (1989) ha aggiunto alcune nuove funzionalità e ha fatto pulizia nel comportamento di alcuni degli "punti oscuri" del linguaggio. Le specifiche per **awk** nello standard POSIX Command Language and Utilities ha in seguito reso più chiaro il linguaggio. Sia i progettisti di **gawk** che quelli dell'originale **awk** dei Bell Laboratories hanno collaborato alla formulazione delle specifiche POSIX.

Paul Rubin ha scritto **gawk**, nel 1986. Jay Fenlason l'ha completata, seguendo i consigli di Richard Stallman. Anche John Woods ha fornito parti del codice. Nel 1988 e 1989, David Trueman, col mio aiuto, ha rivisto completamente **gawk** per la compatibilità col più recente **awk**. Intorno al 1994, sono divenuto il manutentore principale. Lo sviluppo corrente è incentrato sulla correzione degli errori, sul miglioramento delle prestazioni, sulla conformità agli standard e, occasionalmente, su nuove funzionalità.

Nel maggio 1997, Jürgen Kahrs avvertì la necessità di un accesso alla rete da **awk**, e con un piccolo aiuto da parte mia, cominciò ad aggiungere funzionalità a **gawk** per fare questo. A quel tempo, lui scrisse anche il grosso di *TCP/IP Internetworking with gawk* (un documento separato, disponibile come parte della distribuzione **gawk**). Il suo codice alla fine venne integrato nella distribuzione principale di **gawk** con la versione 3.1 di **gawk**.

John Haque ha riscritto la parte interna di **gawk**, mentre metteva a punto un debugger a livello di **awk**. Questa versione divenne disponibile come **gawk** versione 4.0 nel 2011.

Si veda la [Sezione A.9 \[I principali contributori a gawk\]](#), pagina 475, per un elenco completo di quelli che hanno fornito contributi importanti a **gawk**.

## Una rosa, con ogni altro nome...

Il linguaggio **awk** si è evoluto nel corso degli anni. Tutti i dettagli si trovano in [Appendice A \[L'evoluzione del linguaggio awk\]](#), pagina 461. Il linguaggio descritto in questo libro viene spesso citato come "nuovo **awk**". Per analogia, la versione originale di **awk** è citata come "vecchio **awk**."

Su molti sistemi di uso corrente, eseguendo il programma di utilità **awk**, si invoca qualche versione del nuovo **awk**.<sup>4</sup> Se il comando **awk** nel sistema in uso è il vecchio, il risultato che vedrete per il programma di test che segue è del tipo:

```
$ awk 1 /dev/null
error awk: syntax error near line 1
error awk: bailing out near line 1
```

Se questo è il caso, dovrete cercare una versione del nuovo **awk**, o semplicemente installare **gawk**!

All'interno di questo libro, quando si fa riferimento a funzionalità del linguaggio che dovrebbe essere disponibile in ogni implementazione completa di **awk** POSIX, viene usato il termine **awk**. Quando si fa riferimento a una funzionalità specifica dell'implementazione GNU, viene usato il termine **gawk**.

## Uso di questo libro

Il termine **awk** si riferisce sia a uno specifico programma sia al linguaggio che si usa per dire al programma stesso cosa deve fare. Quando dobbiamo essere precisi, chiamiamo il linguaggio “il linguaggio **awk**,” e il programma “l'utilità **awk**.” Questo libro spiega sia come scrivere programmi nel linguaggio **awk** che come eseguire l'utilità **awk**. Il termine “programma **awk**” si riferisce a un programma scritto dall'utente nel linguaggio di programmazione **awk**.

In primo luogo, questo libro spiega le funzionalità di **awk** come definite nello standard POSIX, e lo fa nel contesto dell'implementazione **gawk**. Oltre a questo, cerca anche di descrivere le differenze significative tra **gawk** e altre implementazioni **awk**.<sup>5</sup> Infine, vien fatta rilevare ogni funzionalità di **gawk** non inclusa nello standard POSIX per **awk**.

Questo libro ha il difficile compito di essere tanto una guida introduttiva che un manuale di riferimento. I neofiti possono tranquillamente saltare i dettagli che sembrano loro troppo complessi. Possono anche ignorare i molti riferimenti incrociati, preparati avendo in mente gli utenti esperti e per le versioni Info e **HTML** del libro.

Ci sono dei riquadri sparsi in tutto il libro. Aggiungono una spiegazione più completa su punti importanti, ma che probabilmente non sono di interesse in sede di prima lettura. Si trovano tutti nell'indice analitico, alla voce “sidebar.”

La maggior parte delle volte, gli esempi usano programmi **awk** completi. Alcune delle sezioni più avanzate mostrano solo la parte del programma **awk** che illustra il concetto che si sta descrivendo.

Sebbene questo libro sia destinato soprattutto alle persone che non hanno una precedente conoscenza di **awk**, esso contiene anche tante informazioni che anche gli esperti di **awk** troveranno utili. In particolare, dovrebbero essere d'interesse la descrizione di POSIX **awk** e i programmi di esempio nel **Capitolo 10** [Una libreria di funzioni **awk**], pagina 245, e nel **Capitolo 11** [Programmi utili scritti in **awk**], pagina 281.

Questo libro è suddiviso in diverse parti, come segue:

<sup>4</sup> Solo i sistemi Solaris usano ancora un vecchio **awk** per il programma di utilità predefinito **awk**. Una versione più moderna di **awk** si trova nella directory **/usr/xpg6/bin** su questi sistemi.

<sup>5</sup> Tutte queste differenze si trovano nell'indice alla voce “differenze tra **awk** e **gawk**.”

## 8 GAWK: Programmare efficacemente in AWK

- La Parte I descrive il linguaggio `awk` e il programma `gawk` nel dettaglio. Inizia con le nozioni di base, e continua con tutte le caratteristiche di `awk`. Contiene i seguenti capitoli:
  - Capitolo 1 [Per iniziare con `awk`], pagina 17, fornisce le nozioni minime indispensabili per iniziare a usare `awk`.
  - Capitolo 2 [Eseguire `awk` e `gawk`], pagina 33, descrive come eseguire `gawk`, il significato delle sue opzioni da riga di comando e come trovare i file sorgenti del programma `awk`.
  - Capitolo 3 [Espressioni regolari], pagina 49, introduce le espressioni regolari in generale, e in particolare le varietà disponibili in `awk` POSIX e `gawk`.
  - Capitolo 4 [Leggere file in input], pagina 63, descrive come `awk` legge i dati inseriti dall'utente. Introduce i concetti di record e campi, e anche il comando `getline`. Contiene una prima descrizione della ridirezione I/O, e una breve descrizione dell'I/O di rete.
  - Capitolo 5 [Stampare in output], pagina 95, descrive come i programmi `awk` possono produrre output con `print` e `printf`.
  - Capitolo 6 [Espressioni], pagina 115, descrive le espressioni, che sono i componenti elementari di base per portare a termine la maggior parte delle operazioni in un programma.
  - Capitolo 7 [Criteri di ricerca, azioni e variabili], pagina 145, descrive come scrivere espressioni di ricerca per individuare corrispondenze nei record, le azioni da eseguire quando si è trovata una corrispondenza in un record, e le variabili predefinite di `awk` e `gawk`.
  - Capitolo 8 [Vettori in `awk`], pagina 177, tratta dell'unica struttura di dati di `awk`: il vettore associativo. Vengono trattati anche l'eliminazione di elementi del vettore e di interi vettori, e l'ordinamento dei vettori in `gawk`. Il capitolo descrive inoltre come `gawk` fornisce vettori di vettori.
  - Capitolo 9 [Funzioni], pagina 195, descrive le funzioni predefinite fornite da `awk` e `gawk`, e spiega come definire funzioni personalizzate. Viene anche spiegato come `gawk` permetta di invocare funzioni in maniera indiretta.
- La Parte II illustra come usare `awk` e `gawk` per la risoluzione di problemi. Qui ci sono molti programmi da leggere e da cui imparare. Questa parte contiene i seguenti capitoli:
  - Capitolo 10 [Una libreria di funzioni `awk`], pagina 245, fornisce diverse funzioni pensate per essere usate dai programmi scritti in `awk`.
  - Capitolo 11 [Programmi utili scritti in `awk`], pagina 281, fornisce molti programmi `awk` di esempio.

La lettura di questi due capitoli permette di capire come `awk` può risolvere problemi pratici.

- La Parte III si concentra sulle funzionalità specifiche di `gawk`. Contiene i seguenti capitoli:
  - Capitolo 12 [Funzionalità avanzate di `gawk`], pagina 331, descrive diverse funzionalità avanzate. Di particolare rilevanza sono la capacità di controllare l'ordine di visita dei vettori, quella di instaurare comunicazioni bidirezionali con altri processi, di effettuare connessioni di rete TCP/IP, e di profilare i propri programmi `awk`.

- Capitolo 13 [Internazionalizzazione con `gawk`], pagina 349, descrive funzionalità speciali per tradurre i messaggi di programma in diverse lingue in fase di esecuzione.
- Capitolo 14 [Effettuare il debug dei programmi `awk`], pagina 361, descrive il debugger di `gawk`.
- Capitolo 15 [Calcolo con precisione arbitraria con `gawk`], pagina 379, illustra le capacità di calcolo avanzate.
- Capitolo 16 [Scrivere estensioni per `gawk`], pagina 395, descrive come aggiungere nuove variabili e funzioni a `gawk` scrivendo estensioni in C o C++.
- La Parte IV contiene le appendici, il Glossario, e due licenze relative, rispettivamente, al codice sorgente di `gawk` e a questo libro. Contiene le seguenti appendici:
  - Appendice A [L’evoluzione del linguaggio `awk`], pagina 461, descrive l’evoluzione del linguaggio `awk` dalla sua prima versione fino a oggi. Descrive anche come `gawk` ha acquisito nuove funzionalità col passare del tempo.
  - Appendice B [Installare `gawk`], pagina 479, descrive come ottenere `gawk`, come compilarlo sui sistemi compatibili con POSIX, e come compilarlo e usarlo su diversi sistemi non conformi allo standard POSIX. Spiega anche come segnalare gli errori di `gawk` e dove si possono ottenere altre implementazioni di `awk` liberamente disponibili.
  - Appendice C [Note di implementazione], pagina 499, descrive come disabilitare le estensioni `gawk`, come contribuire scrivendo del nuovo codice per `gawk`, e alcune possibili direzioni per il futuro sviluppo di `gawk`.
  - Appendice D [Concetti fondamentali di programmazione], pagina 511, fornisce del materiale di riferimento a livello elementare per chi sia completamente digiuno di programmazione informatica.
- Il [Glossario], pagina 515, definisce quasi tutti i termini significativi usati all’interno di questo libro. Se si incontrano termini coi quali non si ha familiarità, questo è il posto dove cercarli.
- [Licenza Pubblica Generale GNU (GPL)], pagina 529, e [Licenza per Documentazione Libera GNU (FDL)], pagina 543, presentano le licenze che si applicano, rispettivamente, al codice sorgente di `gawk` e a questo libro.

## Convenzioni tipografiche

Questo libro è scritto in `Texinfo`, il linguaggio di formattazione della documentazione GNU. Viene usato un unico file sorgente `Texinfo` per produrre sia la versione a stampa della documentazione sia quella online. A causa di ciò, le convenzioni tipografiche sono leggermente diverse da quelle presenti in altri libri che potete aver letto.

Gli esempi da immettere sulla riga di comando sono preceduti dai comuni prompt di shell primario e secondario, ‘\$’ e ‘>’. L’input che si inserisce viene mostrato *in questo modo*. L’output del comando è preceduto dal glifo “`␣`”, che in genere rappresenta lo standard output del comando. Messaggi di errore e altri output sullo standard error del comando sono preceduti dal glifo “`error`”. Per esempio:

```
$ echo ciao su stdout
␣ ciao su stdout
$ echo salve su stderr 1>&2
```

```
error  salve su stderr
```

Nel testo, quasi tutto ciò che riguarda la programmazione, per esempio i nomi dei comandi, appare in **questo font**. I frammenti di codice appaiono nello stesso font e tra apici, ‘in questo modo’. Ciò che viene sostituito dall’utente o dal programmatore appare in *questo font*. Le opzioni sono stampate così: `-f`. I nomi-file sono indicati in questo modo: `/percorso/al/file`. Certe cose sono evidenziate *in questo modo*, e se un punto dev’essere reso in modo più marcato, viene evidenziato **in questo modo**. La prima occorrenza di un nuovo termine è usualmente la sua *definizione* e appare nello stesso font della precedente occorrenza di “definizione” in questa frase.

I caratteri che si battono sulla tastiera sono scritti come *questi*. In particolare, ci sono caratteri speciali chiamati “caratteri di controllo”. Questi sono caratteri che vengono battuti tenendo premuti il tasto *CONTROL* e un altro tasto contemporaneamente. Per esempio, *Ctrl-d* è battuto premendo e tenendo premuto il tasto *CONTROL*, poi premendo il tasto *d* e infine rilasciando entrambi i tasti.

Per amor di brevità, in questo libro, la versione di Brian Kernighan di `awk` sarà citata come “BWK `awk`.” (Si veda la [Sezione B.5 \[Altre implementazioni di `awk` liberamente disponibili\]](#), [pagina 494](#), per informazioni su questa e altre versioni.)

## Angoli Bui

*Gli angoli bui sono essenzialmente frattali—per quanto vengano illuminati, ce n’è sempre uno più piccolo e più buio.*

—Brian Kernighan

Fino allo standard POSIX (e *GAWK: Programmare efficacemente in AWK*), molte caratteristiche di `awk` erano poco documentate o non documentate affatto. Le descrizioni di queste caratteristiche (chiamate spesso “angoli bui”) sono segnalate in questo libro con il disegno di una torcia elettrica nel margine, come mostrato qui. Appaiono anche nell’indice sotto la voce “angolo buio.”



Ma come osservato nella citazione d’apertura, ogni trattazione degli angoli bui è per definizione incompleta.

Estensioni al linguaggio standard di `awk` disponibili in più di una implementazione di `awk` sono segnate “(e.c.),” ed elencate nell’indice sotto “estensioni comuni” e “comuni, estensioni”.

## Breve storia del Progetto GNU e di questo libro

La Free Software Foundation (FSF) è un’organizzazione senza scopo di lucro dedicata alla produzione e distribuzione di software liberamente distribuibile. È stata fondata da Richard M. Stallman, l’autore della prima versione dell’editor Emacs. GNU Emacs è oggi la versione di Emacs più largamente usata.

Il Progetto GNU<sup>6</sup> è un progetto della Free Software Foundation in continuo sviluppo per creare un ambiente per computer completo, liberamente distribuibile, conforme allo standard POSIX. La FSF usa la GNU General Public License (GPL) per assicurare che il codice sorgente del loro software sia sempre disponibile all’utente finale. Una copia della GPL è inclusa in questo libro per la consultazione (si veda la [Licenza Pubblica Generale](#)

<sup>6</sup> GNU sta per “GNU’s Not Unix.”

GNU (GPL)], [pagina 529](#)). La GPL si applica al codice sorgente in linguaggio C per `gawk`. Per saperne di più sulla FSF e sul Progetto GNU, si veda [la pagina principale del Progetto GNU](#). Questo libro si può leggere anche dal [sito di GNU](#).

Una shell, un editor (Emacs), compilatori ottimizzanti C, C++ e Objective-C altamente portabili, un debugger simbolico e dozzine di grandi e piccoli programmi di utilità (come `gawk`), sono stati completati e sono liberamente disponibili. Il kernel del sistema operativo GNU (noto come HURD), è stato rilasciato ma è ancora allo stato di sviluppo iniziale.

In attesa che il sistema operativo GNU venga più completamente sviluppato, si dovrebbe prendere in considerazione l'uso di GNU/Linux, un sistema operativo liberamente distribuibile e basato su Unix disponibile per Intel, Power Architecture, Sun SPARC, IBM S/390, e altri sistemi.<sup>7</sup> Molte distribuzioni GNU/Linux sono scaricabili da internet.

Il libro è realmente libero—almeno, l'informazione che contiene è libera per chiunque—. Il codice sorgente del libro, leggibile elettronicamente, viene fornito con `gawk`. (Dare un'occhiata alla Free Documentation License in [\[Licenza per Documentazione Libera GNU \(FDL\)\], pagina 543](#).)

Il libro in sé ha già avuto parecchie edizioni in passato. Paul Rubin ha scritto la prima bozza di *The GAWK Manual*, che era lunga una quarantina di pagine. Diane Close e Richard Stallman l'hanno migliorata arrivando alla versione che era lunga una novantina di pagine, e descriveva solo la versione originale “vecchia” di `awk`. Ho iniziato a lavorare con quella versione nell'autunno del 1988. Mentre ci stavo lavorando, la FSF ha pubblicato parecchie versioni preliminari, numerate 0.x). Nel 1996, l'edizione 1.0 fu rilasciata assieme a `gawk` 3.0.0. La FSF ha pubblicato le prime due edizioni col titolo *GAWK: The GNU Awk User's Guide*.

Questa edizione mantiene la struttura di base delle edizioni precedenti. Per l'edizione FSF 4.0, il contenuto era stato accuratamente rivisto e aggiornato. Tutti i riferimenti a versioni di `gawk` anteriori alla versione 4.0 sono stati eliminati. Di particolare interesse in quella edizione era l'aggiunta del [Capitolo 14 \[Effettuare il debug dei programmi `awk`\], pagina 361](#).

Per l'edizione FSF 4.1, il contenuto è stato riorganizzato in parti, e le aggiunte più importanti sono il [Capitolo 15 \[Calcolo con precisione arbitraria con `gawk`\], pagina 379](#), e il [Capitolo 16 \[Scrivere estensioni per `gawk`\], pagina 395](#).

Questo libro continuerà certamente ad evolversi. Se si trovano errori nel libro, si prega di segnalarli! Si veda la [Sezione B.4 \[Segnalazione di problemi e bug\], pagina 493](#), per informazioni su come inviare la segnalazione di problemi elettronicamente.

## Come collaborare

Come manutentore di GNU `awk`, un tempo pensai che sarei stato in grado di gestire una raccolta di programmi `awk` pubblicamente disponibili e avevo anche esortato a collaborare. Rendere disponibili le cose su Internet aiuta a contenere la distribuzione `gawk` entro dimensioni gestibili.

L'iniziale raccolta di materiale, come questo, è tuttora disponibile su <ftp://ftp.freefriends.org/arnold/Awkstuff>.

Chi fosse *seriamente* interessato a contribuire nell'implementazione di un sito Internet dedicato ad argomenti riguardanti il linguaggio `awk`, è pregato di contattarmi.

<sup>7</sup> La terminologia “GNU/Linux” è spiegata nel [\[Glossario\], pagina 515](#).

## Ringraziamenti

La bozza iniziale di *The GAWK Manual* riportava i seguenti ringraziamenti:

Molte persone devono essere ringraziate per la loro assistenza nella produzione di questo manuale. Jay Fenlason ha contribuito con molte idee e programmi di esempio. Richard Mlynarik e Robert Chassell hanno fatto utili osservazioni sulle bozze di questo manuale. Lo scritto *A Supplemental Document for AWK* di John W. Pierce, del Chemistry Department di UC San Diego, fa il punto su diverse questioni rilevanti sia per l'implementazione di `awk` che per questo manuale, che altrimenti ci sarebbero sfuggite.

Vorrei ringraziare Richard M. Stallman, per la sua visione di un mondo migliore e per il suo coraggio nel fondare la FSF e nel dare inizio al Progetto GNU.

Edizioni precedenti di questo libro riportavano i seguenti ringraziamenti:

Le seguenti persone (in ordine alfabetico) hanno inviato commenti utili riguardo alle diverse versioni di questo libro: Rick Adams, Dr. Nelson H.F. Beebe, Karl Berry, Dr. Michael Brennan, Rich Burridge, Claire Cloutier, Diane Close, Scott Deifik, Christopher (“Topher”) Eliot, Jeffrey Friedl, Dr. Darrel Hankerson, Michal Jaegermann, Dr. Richard J. LeBlanc, Michael Lijewski, Pat Rankin, Miriam Robbins, Mary Sheehan, e Chuck Toporek.

Robert J. Chassell ha dato preziosissimi consigli sull'uso di Texinfo. Merita anche un particolare ringraziamento per avermi convinto a *non* dare a questo libro il titolo *How to Gawk Politely*. [Un gioco di parole in inglese che può significare sia *Come usare Gawk educatamente* che *Come curiosare educatamente*]. Karl Berry ha aiutato in modo significativo con la parte T<sub>E</sub>X di Texinfo.

Vorrei ringraziare Marshall ed Elaine Hartholz di Seattle e il Dr. Bert e Rita Schreiber di Detroit per i lunghi periodi di vacanza trascorsi in tutta tranquillità in casa loro, che mi hanno permesso di fare importanti progressi nella scrittura di questo libro e con lo stesso `gawk`.

Phil Hughes di SSC ha contribuito in modo molto importante prestandomi il suo portatile col sistema GNU/Linux, non una volta, ma due, il che mi ha permesso di fare tantissimo lavoro mentre ero fuori casa.

David Trueman merita un riconoscimento speciale; ha fatto un lavoro da sentinella durante lo sviluppo di `gawk` affinché funzioni bene e senza errori. Sebbene non sia più impegnato con `gawk`, lavorare con lui a questo progetto è stato un vero piacere.

Gli intrepidi membri della lista GNITS, con una particolare menzione per Ulrich Drepper, hanno fornito un aiuto prezioso e commenti per il progetto delle funzionalità di internazionalizzazione.

Chuck Toporek, Mary Sheehan, e Claire Cloutier della O'Reilly & Associates hanno fornito un'assistenza editoriale rilevante per questo libro per la versione 3.1 di `gawk`.

Dr. Nelson Beebe, Andreas Buening, Dr. Manuel Collado, Antonio Colombo, Stephen Davies, Scott Deifik, Akim Demaille, Daniel Richard G., Darrel Hankerson, Michal Jaegermann, Jürgen Kahrs, Stepan Kasal, John Malmberg, Dave Pitts, Chet Ramey, Pat Rankin, Andrew Schorr, Corinna Vinschen, ed Eli Zaretskii (in ordine alfabetico) costituiscono

l'attuale "gruppo di lavoro sulla portabilità" di **gawk**. Senza il loro duro lavoro e il loro aiuto, **gawk** non sarebbe stato neanche lontanamente il buon programma che è oggi. È stato e continua a essere un piacere lavorare con questo gruppo di ottimi collaboratori.

Notevoli contributi di codice e documentazione sono arrivati da parecchie persone. Si veda la [Sezione A.9 \[I principali contributori a gawk\], pagina 475](#), per l'elenco completo.

Grazie a Michael Brennan per le Prefazioni.

Grazie a Patrice Dumas per il nuovo programma **makeinfo**. Grazie a Karl Berry, che continua a lavorare per tenere aggiornato il linguaggio di marcatura Texinfo.

Robert P.J. Day, Michael Brennan e Brian Kernighan hanno gentilmente fatto da revisori per l'edizione 2015 di questo libro. Le loro osservazioni hanno contribuito a migliorare la stesura finale.

Vorrei ringraziare Brian Kernighan per la sua preziosa assistenza durante la fase di collaudo e di debug di **gawk** e per l'aiuto in corso d'opera e i consigli nel chiarire diversi punti sul linguaggio. Non avremmo proprio fatto un così buon lavoro su **gawk** e sulla sua documentazione senza il suo aiuto.

Brian è un fuoriclasse sia come programmatore che come autore di manuali tecnici. È mio dovere ringraziarlo (una volta di più) per la sua costante amicizia e per essere stato per me un modello da seguire ormai da quasi 30 anni! Averlo come revisore è per me un privilegio eccitante, ma è stata anche un'esperienza che mi ha fatto sentire molto piccolo. . .

Devo ringraziare la mia meravigliosa moglie, Miriam, per la sua pazienza nel corso delle molte versioni di questo progetto, per la correzione delle bozze e per aver condiviso con me il computer. Vorrei ringraziare i miei genitori per il loro amore, e per la gentilezza con cui mi hanno cresciuto ed educato. Infine, devo riconoscere la mia gratitudine a D-o, per le molte opportunità che mi ha offerto, e anche per i doni che mi ha elargito, con cui trarre vantaggio da quelle opportunità.

Arnold Robbins  
Nof Ayalon  
Israel  
Febbraio 2015



# Parte I:

## Il Linguaggio awk



# 1 Per iniziare con awk

Il compito fondamentale di **awk** è quello di ricercare righe (o altre unità di testo) in file che corrispondano a certi criteri di ricerca. Quando una riga corrisponde a uno dei criteri, **awk** esegue su quella riga le azioni specificate per quel criterio. **awk** continua a elaborare righe in input in questo modo, finché non raggiunge la fine delle righe nei file in input.

I programmi scritti in **awk** sono differenti dai programmi scritti nella maggior parte degli altri linguaggi, poiché i programmi **awk** sono *guidati dai dati* (ovvero, richiedono di descrivere i dati sui quali si vuole operare, e in seguito che cosa fare una volta che tali dati siano stati individuati). La maggior parte degli altri linguaggi sono *procedurali*; si deve descrivere, in maniera molto dettagliata, ogni passo che il programma deve eseguire. Lavorando con linguaggi procedurali, solitamente è molto più difficile descrivere chiaramente i dati che il programma deve elaborare. Per questa ragione i programmi **awk** sono spesso piacevolmente facili da leggere e da scrivere.

Quando si esegue **awk**, va specificato un *programma awk* che dice ad **awk** cosa fare. Il programma consiste di una serie di *regole* (può anche contenere *definizioni di funzioni*, una funzionalità avanzata che per ora ignoreremo; si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224). Ogni regola specifica un criterio di ricerca e un'azione da effettuare una volta che viene trovato un record corrispondente.

Sintatticamente, una regola consiste in un *criterio di ricerca*, seguito da una *azione*. L'azione è racchiusa tra parentesi graffe per separarla dal criterio di ricerca. Per separare regole, basta andare a capo. Quindi un programma **awk** ha una struttura simile a questa:

```
criterio { azione }
criterio { azione }
...
```

## 1.1 Come iniziare a eseguire programmi gawk

Ci sono vari modi di eseguire un programma **awk**. Se il programma è corto, è più facile includerlo nel comando con cui si invoca **awk**, così:

```
awk 'programma' input-file1 input-file2 ...
```

Quando il programma è lungo, di solito è meglio metterlo in un file ed eseguirlo con un comando come questo:

```
awk -f file-di-programma input-file1 input-file2 ...
```

Questa sezione si occupa di entrambe queste modalità insieme a parecchie varianti di ciascuna di esse.

### 1.1.1 Eseguire un breve programma awk usa-e-getta

Una volta acquisita familiarità con **awk**, capiterà spesso di preparare semplici programmi nel momento in cui servono. In questo caso si può scrivere il programma come primo argomento del comando **awk**, così:

```
awk 'programma' input-file1 input-file2 ...
```

dove *programma* consiste in una serie di criteri di ricerca e di azioni, come descritto precedentemente.

Questo formato di comando chiede alla *shell*, ossia all'interprete dei comandi, di richiamare **awk** e di usare il *programma* per trattare record nei file in input. Il *programma* è incluso tra apici in modo che la shell non interpreti qualche carattere destinato ad **awk** come carattere speciale della shell. Gli apici fanno inoltre sì che la shell tratti tutto il *programma* come un solo argomento per **awk**, e permettono che *programma* sia più lungo di una riga.

Questo formato è utile anche per eseguire programmi **awk** di dimensioni piccole o medie da *script* di shell, perché non richiede un file separato che contenga il programma **awk**. Uno *script* di shell è più affidabile, perché non ci sono altri file che possono venirsi a trovare fuori posto.

Più avanti in questo capitolo, nella [Sezione 1.3 \[Alcuni esempi molto semplici\]](#), pagina 24, si vedranno esempi di parecchi programmi, brevi, scritti sulla riga di comando.

### 1.1.2 Senza uso di file in input (input immesso da tastiera)

Si può anche eseguire **awk** senza indicare alcun file in input. Se si immette la seguente riga di comando:

```
awk 'programma'
```

**awk** prende come input del *programma* lo *standard input*, che di solito significa qualsiasi cosa venga immesso dalla tastiera. Ciò prosegue finché non si segnala una fine-file battendo **Ctrl-d**. (In sistemi operativi non-POSIX, il carattere di fine-file può essere diverso.)

Per esempio, il seguente programma stampa un consiglio da amico (dalla *Guida galattica per gli autostoppisti* di Douglas Adams), per non lasciarsi spaventare dalle complessità della programmazione per computer:

```
$ awk 'BEGIN { print "Non v\47allarmate!" }'
→ Non v'allarmate!
```

**awk** esegue le istruzioni associate a **BEGIN** prima di leggere qualsiasi input. Se non ci sono altre istruzioni nel proprio programma, come in questo caso, **awk** si ferma, invece di tentare di leggere input che non sa come elaborare. Il '\47' è un modo straordinario (spiegato più avanti) per inserire un apice singolo nel programma, senza dover ricorrere a fastidiosi meccanismi di protezione della shell.

**NOTA:** Se si usa Bash come shell, si dovrebbe digitare il comando **'set +H'** prima eseguire questo programma interattivamente, per non avere una cronologia dei comandi nello stile della C shell, che tratta il **'!'** come un carattere speciale. Si raccomanda di inserire quel comando nel proprio file di personalizzazione della shell.

Il seguente semplice programma **awk** emula il comando **cat**; ovvero copia qualsiasi cosa si batta sulla tastiera nel suo standard output (perché succede è spiegato fra poco):

```
$ awk '{ print }'
Ora è il tempo per tutti gli uomini buoni
→ Ora è il tempo per tutti gli uomini buoni
di venire in aiuto al loro paese.
→ di venire in aiuto al loro paese.
Or sono sedici lustri e sette anni, ...
→ Or sono sedici lustri e sette anni, ...
Cosa, io preoccupato?
```

```

└─ Cosa, io preoccupato?
Ctrl-d

```

### 1.1.3 Eseguire programmi lunghi

Talora i programmi **awk** sono molto lunghi. In tali situazioni conviene mettere il programma in un file separato. Per dire ad **awk** di usare quel file come programma, digitare:

```
awk -f file-sorgente input-file1 input-file2 ...
```

L'opzione **-f** dice al comando **awk** di ottenere il programma **awk** dal file *file-sorgente* (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)). Ogni nome-file può essere usato come *file-sorgente*. Per esempio, si potrebbe mettere il programma:

```
BEGIN { print \"Non v'allarmate!\" }
```

nel file *consiglio*. Allora questo comando:

```
awk -f consiglio
```

è equivalente al comando:

```
awk 'BEGIN { print \"Non v\\47allarmate!\" }'
```

Questo è già stato spiegato prima (si veda la [Sezione 1.1.2 \[Senza uso di file in input \(input immesso da tastiera\)\]](#), [pagina 18](#)). Si noti che normalmente non serve mettere apici singoli nel nome-file che si fornisce con **-f**, perché di solito i nomi-file non contengono caratteri che sono speciali per la shell. Si noti che in *consiglio*, il programma **awk** non ha dei doppi apici che lo delimitano. I doppi apici sono necessari solo per programmi scritti direttamente sulla riga di comando di **awk**. (Inoltre, se il programma si trova in un file, è possibile usare un apice singolo all'interno del programma, invece del magico '\\47'.)

Per identificare chiaramente un file di programma **awk** come tale, si può aggiungere il suffisso **.awk** al nome-file. Ciò non cambia l'esecuzione del programma **awk** ma semplifica la “manutenzione”.

### 1.1.4 Programmi awk da eseguire come *script*

Una volta familiarizzato con **awk**, si potrebbero scrivere *script* che richiamano **awk**, usando il meccanismo di *script* **#!**. Ciò è possibile in molti sistemi operativi.<sup>1</sup> Per esempio, si potrebbe modificare il file *consiglio* e farlo divenire:

```
#!/bin/awk -f
```

```
BEGIN { print \"Non v'allarmate!\" }
```

Dopo aver reso eseguibile questo file (con il comando **chmod**), digitare semplicemente *'consiglio'* al prompt della shell e il sistema si preparerà a eseguire **awk** come se si fosse digitato *'awk -f consiglio'*:

```

$ chmod +x consiglio
$ consiglio
└─ Non v'allarmate!

```

(Si suppone che la directory corrente sia tra quelle contenute nella variabile che indica il “percorso” di ricerca [solitamente **\$PATH**]. In caso contrario si potrebbe aver bisogno di digitare *'./consiglio'* nella shell.)

<sup>1</sup> Il meccanismo **#!** funziona nei sistemi GNU/Linux, in quelli basati su BSD e nei sistemi Unix a pagamento.

*Script* `awk` autocontenuti sono utili se si vuol scrivere un programma che gli utenti possono richiamare senza dover essere informati che il programma è scritto in `awk`.

### Comprendere ‘#!’

`awk` è un linguaggio *interpretato*. Ciò significa che il comando `awk` legge il programma dell’utente e poi elabora i dati secondo le istruzioni contenute nel programma (diversamente da un linguaggio *compilato* come il C, dove il programma viene prima compilato in codice macchina che è eseguito direttamente dal processore del sistema). Il programma di utilità `awk` è perciò chiamato *interprete*. Molti linguaggi moderni sono interpretati.

La riga che inizia con ‘#!’ lista l’intero nome-file di un interprete da richiamare, con degli argomenti facoltativi che saranno passati a quell’interprete sulla riga di comando. Il sistema operativo quindi richiama l’interprete con gli argomenti dati e con l’intera lista di argomenti con cui era stato invocato il programma. Il primo argomento nella lista è l’intero nome-file del programma `awk`. Il resto della lista degli argomenti contiene opzioni per `awk`, oppure file-dati, o entrambi. (Si noti che in molti sistemi `awk` può essere trovato in `/usr/bin` invece che in `/bin`.)

Alcuni sistemi limitano la lunghezza del nome del programma interprete a 32 caratteri. Spesso, si può rimediare utilizzando un collegamento simbolico.

Non si dovrebbero mettere altri argomenti oltre al primo nella riga ‘#!’ dopo il percorso del comando `awk`. Non funziona. Il sistema operativo tratta il resto della riga come un argomento solo, e lo passa ad `awk`. Così facendo il comportamento sarà poco chiaro; con ogni probabilità un messaggio di errore di qualche tipo da `awk`.

Infine, il valore di `ARGV[0]` (si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)) può variare a seconda del sistema operativo. Alcuni sistemi ci mettono ‘`awk`’, altri il nome completo del percorso di `awk` (ad. es. `/bin/awk`), e altri ancora mettono il nome dello *script* dell’utente (‘`consiglio`’). Non bisogna fidarsi del valore di `ARGV[0]` per ottenere il nome del proprio *script*.



### 1.1.5 Documentare programmi `gawk`.

Un *commento* è del testo incluso in un programma per aiutare le persone che lo leggeranno; non è parte del programma eseguibile vero e proprio. I commenti possono spiegare cosa fa il programma e come funziona. Quasi tutti i linguaggi di programmazione possono contenere commenti, poiché i programmi sono solitamente difficili da comprendere senza di essi.

Nel linguaggio `awk`, un commento inizia con il segno del cancelletto (‘#’) e continua fino alla fine della riga. Il ‘#’ non deve necessariamente essere il primo carattere della riga. Il linguaggio `awk` ignora il resto di una riga dopo il carattere cancelletto. Per esempio, potremmo mettere quel che segue in `consiglio`:

```
# Questo programma stampa uno scherzoso consiglio amichevole.
# Aiuta a far passare la paura del computer agli utenti novelli.
BEGIN    { print "Non v'allarmate!" }
```

Si possono mettere dei commenti nei programmi `awk` usa-e-getta da digitare direttamente da tastiera, ma ciò solitamente non serve molto; il fine di un commento è di aiutare l’utente o qualcun altro a comprendere il programma, quando lo rilegge in un secondo tempo.

**ATTENZIONE:** Come detto in Sezione 1.1.1 [Eseguire un breve programma `awk` usa-e-getta], pagina 17, si possono includere programmi di dimensioni da piccole a medie tra apici singoli, per mantenere compatti i propri *script* di shell autocontenuti. Nel far questo, *non* bisogna inserire un apostrofo (ossia un apice singolo) in un commento, (o in qualsiasi altra parte del vostro programma). La shell interpreta gli apici singoli come delimitatori di chiusura dell'intero programma. Di conseguenza, solitamente la shell emette un messaggio riguardo ad apici presenti in numero dispari, e se `awk` viene comunque eseguito, è probabile che stampi strani messaggi di errori di sintassi. Per esempio, nel caso seguente:

```
$ awk 'BEGIN { print "Ciao" } # un'idea brillante'
>
```

La shell considera il secondo apice singolo come delimitatore del testo precedente, e trova che un nuovo testo tra apici ha inizio verso la fine della riga di comando. A causa di ciò emette una richiesta secondaria di input, e si mette in attesa di ulteriore input. Con il comando `awk` Unix, se si chiude l'ulteriore stringa tra apici singoli il risultato è il seguente:

```
$ awk '{ print "Ciao" } # un'idea brillante'
> '
[error] awk: fatale: non riesco ad aprire file 'brillante'
[error] in lettura (File o directory non esistente)
```

Mettere una barra inversa prima dell'apice singolo in `'un'idea'` non risolverebbe, poiché le barre inverse non sono speciali all'interno di apici singoli. La prossima sottosezione descrive le regole di protezione della shell.

### 1.1.6 Uso di apici nella shell.

Per programmi `awk` di lunghezza da corta a media spesso conviene digitare il programma sulla riga di comando `awk`. La maniera migliore per farlo è racchiudere l'intero programma tra apici singoli. Questo vale sia che si digiti il programma interattivamente su richiesta della shell, sia che lo si scriva come parte di uno *script* di shell di maggiori dimensioni:

```
awk 'testo del programma' input-file1 input-file2 ...
```

Quando si lavora con la shell, non guasta avere una conoscenza di base sulle regole per l'uso di apici nella shell. Le regole seguenti valgono solo per shell in stile Bourne (come Bash, la Bourne-Again shell). Se si usa la C shell, si avranno regole differenti.

Prima di immergerci nelle regole, introduciamo un concetto che ricorre in tutto questo libro, che è quello della stringa *null*, o vuota.

La stringa nulla è una variabile, di tipo carattere, che non ha un valore. In altre parole, è vuota. Nei programmi `awk` si scrive così: `""`. Nella shell la si può scrivere usando apici sia singoli che doppi: `""` oppure `''`. Sebbene la stringa nulla non contenga alcun carattere, essa esiste lo stesso. Si consideri questo comando:

```
$ echo ""
```

Qui, il comando `echo` riceve un solo argomento, anche se quell'argomento non contiene alcun carattere. Nel resto di questo libro, usiamo indifferentemente i termini *stringa nulla* e *stringa vuota*. Ora, proseguiamo con le regole relative agli apici:

- Elementi tra apici possono essere concatenati con elementi non tra apici. La shell converte il tutto in un singolo argomento da passare al comando.

- Mettere una barra inversa (`\`) prima di qualsiasi singolo carattere lo protegge. La shell toglie la barra inversa e passa il carattere protetto al comando.
- Gli apici singoli proteggono qualsiasi cosa sia inclusa tra un apice di apertura e uno di chiusura. La shell non interpreta il testo protetto, il quale viene passato così com'è al comando. È *impossibile* inserire un apice singolo in un testo racchiuso fra apici singoli. Potete trovare in [Sezione 1.1.5 \[Documentare programmi gawk.\]](#), pagina 20, un esempio di cosa succede se si prova a farlo.
- I doppi apici proteggono la maggior parte di quel che è racchiuso tra i doppi apici di apertura e quelli di chiusura. La shell effettua almeno la sostituzione di variabili e di comandi sul testo racchiuso tra doppi apici. Shell differenti possono fare ulteriori tipi di elaborazione sul testo racchiuso tra doppi apici.

Poiché alcuni caratteri all'interno di un testo racchiuso tra doppi apici sono interpretati dalla shell, essi devono essere *protetti* all'interno del testo stesso. Sono da tener presenti i caratteri `$`, `'`, `\`, e `"`, tutti i quali devono essere preceduti da una barra inversa quando ricorrono all'interno di un testo racchiuso tra doppi apici, per poter essere passati letteralmente al programma. (La barra inversa viene tolta prima del passaggio al programma.) Quindi, l'esempio visto precedentemente in [Sezione 1.1.2 \[Senza uso di file in input \(input immesso da tastiera\)\]](#), pagina 18:

```
awk 'BEGIN { print "Non v\47allarmate!" }'
```

si potrebbe scrivere invece così:

```
$ awk "BEGIN { print \"Non v'allarmate!\" }"
+ Non v'allarmate!
```

Va notato che l'apice singolo non è speciale all'interno di un testo racchiuso tra doppi apici.

- Le stringhe nulle sono rimosse se presenti come parte di un argomento non-nullo sulla riga di comando, mentre oggetti esplicitamente nulli sono mantenuti come tali. Per esempio, per richiedere che il separatore di campo FS sia impostato alla stringa nulla, digitare:

```
awk -F "" 'programma' file # corretto
```

Non è invece da usare:

```
awk -F"" 'programma' file # errato!
```

Nel secondo caso, `awk` tenta di usare il nome del programma come valore di `FS`, e il primo nome-file come testo del programma! Ciò come minimo genera un errore di sintassi, e un comportamento confuso nel caso peggiore.

Mischiare apici singoli e doppi è difficile. Occorre utilizzare trucchi della shell per gli apici, come questi:

```
$ awk 'BEGIN { print "Questo è un apice singolo. <'\"'>" }'
+ Questo è un apice singolo. <'>
```

Questo programma stampa tre stringhe tra apici concatenate tra loro. La prima e la terza sono rinchiuse tra apici singoli, la seconda tra apici doppi.

Quanto sopra può essere “semplificato” così:

```
$ awk 'BEGIN { print "Questo è un apice singolo <'\''>" }'
+ Questo è un apice singolo <'>
```

A voi la scelta del più leggibile dei due.

Un'altra opzione è quella di usare doppi apici, proteggendo i doppi apici inclusi, a livello `awk`:

```
$ awk 'BEGIN { print "\"Questo è un apice singolo <'>\"" }'
+ Questo è un apice singolo <'>
```

Quest'opzione è fastidiosa anche perché il doppio apice, la barra inversa e il simbolo del dollaro sono molto comuni nei programmi `awk` più avanzati.

Una terza opzione è quella di usare le sequenze ottali equivalenti (si veda la [Sezione 3.2 \[Sequenze di protezione\], pagina 50](#)) per i caratteri apice singolo e doppio, così:

```
$ awk 'BEGIN { print "\"Questo è un apice singolo <\47>\"" }'
+ Questo è un apice singolo <'>
$ awk 'BEGIN { print "\"Questo è un doppio apice <\42>\"" }'
+ Questo è un doppio apice <">
```

Questo funziona bene, ma sai dovrebbe commentare chiaramente quel che il testo protetto significa.

Una quarta possibilità è di usare assegnamenti di variabili sulla riga di comando, così:

```
$ awk -v sq="'" 'BEGIN { print "\"Questo è un apice singolo <" sq ">\"" }'
+ Questo è un apice singolo <'>
```

(Qui, le due stringhe costanti e il valore di `sq` sono concatenati in un'unica stringa che è stampata da `print`.)

Se servono veramente sia gli apici singoli che quelli doppi nel proprio programma `awk`, è probabilmente meglio tenerlo in un file separato, dove la shell non interferisce, e si potrà scrivere quello che si vuole.

### 1.1.6.1 Doppi apici in file .BAT Windows

Sebbene questo libro in generale si preoccupi solo di sistemi POSIX e della shell POSIX, il problema che stiamo per vedere emerge abbastanza spesso presso parecchi utenti, e per questo ne parliamo.

Le “shell” nei sistemi Microsoft Windows usano il carattere doppio apice per protezione, e rendono difficile o impossibile inserire un carattere doppio apice in uno *script* scritto su una riga di comando. L'esempio che segue, per il quale ringraziamo Jeroen Brink, mostra come stampare tutte le righe di un file, racchiudendole tra doppi apici:

```
gawk "{ print \"\042\" $0 \"\042\" }" file
```

## 1.2 File-dati per gli esempi

Molti degli esempi in questo libro hanno come input due file-dati di esempio. Il primo, `mail-list`, contiene una lista di nomi di persone, insieme ai loro indirizzi email e a informazioni riguardanti le persone stesse. Il secondo file-dati, di nome `inventory-shipped`, contiene informazioni riguardo a consegne mensili. In entrambi i file, ogni riga è considerata come un *record*.

Nel `mail-list`, ogni record contiene il nome di una persona, il suo numero di telefono, il suo indirizzo email, e un codice che indica la sua relazione con l'autore della lista. Le colonne sono allineate usando degli spazi. Una ‘A’ nell'ultima colonna indica che quella persona è

un conoscente [Acquaintance]. Una ‘F’ nell’ultima colonna significa che quella persona è un amico [Friend]. Una ‘R’ vuol dire che quella persona è un parente [Relative]:

Amelia	555-5553	amelia.zodiacusque@gmail.com	F
Anthony	555-3412	anthony.asserturo@hotmail.com	A
Becky	555-7685	becky.algebrarum@gmail.com	A
Bill	555-1675	bill.drowning@hotmail.com	A
Broderick	555-0542	broderick.aliquotiens@yahoo.com	R
Camilla	555-2912	camilla.infusarum@skynet.be	R
Fabius	555-1234	fabius.undevicesimus@ucb.edu	F
Julie	555-6699	julie.perscrutabor@skeeve.com	F
Martin	555-6480	martin.codicibus@hotmail.com	A
Samuel	555-3430	samuel.lanceolis@shu.edu	A
Jean-Paul	555-2127	jeanpaul.campanorum@nyu.edu	R

Il file-dati `inventory-shipped` contiene informazioni sulle consegne effettuate durante l’anno. Ogni record contiene il mese, il numero di contenitori verdi spediti, il numero di scatole rosse spedite, il numero di borse arancione spedite, e il numero di pacchetti blu spediti, in quest’ordine. Ci sono 16 record, relativi ai dodici mesi dello scorso anno e ai primi quattro mesi dell’anno in corso. Una riga vuota separa i dati relativi a ciascun anno:

Jan	13	25	15	115
Feb	15	32	24	226
Mar	15	24	34	228
Apr	31	52	63	420
May	16	34	29	208
Jun	31	42	75	492
Jul	24	34	67	436
Aug	15	34	47	316
Sep	13	55	37	277
Oct	29	54	68	525
Nov	20	87	82	577
Dec	17	35	61	401
Jan	21	36	64	620
Feb	26	58	80	652
Mar	24	75	70	495
Apr	21	70	74	514

Questi file di esempio sono inclusi nella distribuzione `gawk`, nella directory `awklib/eg/data`.

### 1.3 Alcuni esempi molto semplici

I seguenti comandi eseguono un semplice programma `awk` che cerca nel file in input `mail-list` la stringa di caratteri ‘li’ (una sequenza di caratteri è solitamente chiamato una *stringa*; il termine *stringa* è basato su un uso linguistico, del tipo “una stringa di perle” o “una stringa di luci decorative”):

```
awk '/li/ { print $0 }' mail-list
```

Quando si incontra una riga che contiene `'li'`, la si stampa, perché `'print $0'` significa "stampa la riga corrente". (Lo scrivere solo `'print'` ha lo stesso significato, quindi avremmo anche potuto limitarci a fare così).

Si sarà notato che delle barre (`'/'`) delimitano la stringa `'li'` nel programma `awk`. Le barre indicano che `'li'` è il modello da ricercare. Questo tipo di notazione è definita come *espressione regolare*, e sarà trattata più avanti in maggior dettaglio (si veda il [Capitolo 3 \[Espressioni regolari\]](#), pagina 49). Il modello può corrispondere anche solo a una parte di una parola. Ci sono apici singoli che racchiudono il programma `awk` in modo che la shell non interpreti alcuna parte di esso come un carattere speciale della shell.

Questo è quello che il programma stampa:

```
$ awk '/li/ { print $0 }' mail-list
+ Amelia          555-5553      amelia.zodiacusque@gmail.com    F
+ Broderick       555-0542      broderick.aliquotiens@yahoo.com  R
+ Julie          555-6699      julie.perscrutabor@skeeeve.com   F
+ Samuel         555-3430      samuel.lanceolis@shu.edu         A
```

In una regola `awk`, il criterio di selezione o l'azione possono essere omessi, ma non entrambi. Se il criterio è omissso, l'azione viene applicata a *ogni* riga dell'input. Se l'azione viene omessa, per default si stampano tutte le righe che sono individuate dal criterio di selezione.

Quindi, si potrebbe omettere l'azione (l'istruzione `print` e le graffe) nell'esempio precedente e il risultato sarebbe lo stesso: `awk` stampa tutte le righe che corrispondono al criterio di ricerca `'li'`. Per confronto, omettendo l'istruzione `print` ma lasciando le graffe si richiede un'azione nulla, che non fa nulla (cioè non stampa alcuna riga).

Molti programmi `awk` pratici sono lunghi solo una o due righe. Qui sotto troviamo una collezione di programmi utili e corti, per iniziare. Alcuni di questi programmi contengono elementi del linguaggio che non sono ancora stati spiegati. (La descrizione del programma fornisce una buona idea di quel che si vuole ottenere, ma occorre leggere il resto del libro per divenire esperti in `awk`!) Molti degli esempi usano un file-dati di nome `data`. Questo serve solo a indicare la posizione del nome; se questi programmi devono venir usati per se stessi, sostituire i propri nomi-file al posto di `data`. Per futura memoria, si noti che spesso c'è più di un modo per fare qualcosa in `awk`. In un altro momento, si potrebbe tornare a guardare questi esempi per vedere se si riescono a trovare modi differenti per fare le stesse cose mostrate qui appresso:

- Stampare ogni riga lunga più di 80 caratteri:

```
awk 'length($0) > 80' data
```

L'unica regola presente ha un'espressione di relazione come modello e non ha azione—quindi applica l'azione di default, stampando il record.

- Stampare la lunghezza della riga in input più lunga:

```
awk '{ if (length($0) > max) max = length($0) }
      END { print max }' data
```

Il codice associato a `END` viene eseguito dopo che tutto l'input è stato letto; è l'altra faccia della medaglia di `BEGIN`.

- Stampare la lunghezza della riga più lunga in `data`:

```
expand data | awk '{ if (x < length($0)) x = length($0) }
```

```
END { print "la lunghezza massima di una riga è" x }'
```

Questo esempio è leggermente diverso da quello precedente: l'input è l'output del comando `expand`, che cambia i TAB in spazi, in modo che le larghezze confrontate siano quelle che sarebbero qualora le si stampasse, e non il numero dei caratteri di input su ogni riga. [il carattere TAB occupa un byte nel file, ma può generare fino a otto spazi bianchi in fase di stampa.]

- Stampare ogni riga che abbia almeno un campo:

```
awk 'NF > 0' data
```

Questa è una maniera facile per eliminare le righe vuote dal file (o piuttosto, per creare un nuovo file, simile al vecchio, ma nel quale le linee vuote sono state tolte).

- Stampare sette numeri casuali compresi tra 0 e 100, inclusi:

```
awk 'BEGIN { for (i = 1; i <= 7; i++)
           print int(101 * rand()) }'
```

- Stampare il numero totale di byte usato da un *elenco-file*:

```
ls -l elenco-file | awk '{ x += $5 }
                        END { print "byte totali: " x }'
```

- Stampare il numero totale di kilobyte usati da *elenco-file*:

```
ls -l elenco-file | awk '{ x += $5 }
                        END { print "K-byte totali:", x / 1024 }'
```

- Stampare una lista in ordine alfabetico di tutti gli utenti del sistema [Unix]:

```
awk -F: '{ print $1 }' /etc/passwd | sort
```

- Contare le righe in un file:

```
awk 'END { print NR }' data
```

- Stampare le righe pari nel file-dati:

```
awk 'NR % 2 == 0' data
```

Se aveste usato invece l'espressione `'NR % 2 == 1'`, il programma avrebbe stampato le righe dispari.

## 1.4 Un esempio che usa due regole

Il programma `awk` legge il file in input una riga alla volta. Per ogni riga `awk` controlla la corrispondenza con ogni regola. Se viene trovata più di una corrispondenza, vengono eseguite altrettante azioni, nell'ordine in cui appaiono nel programma `awk`. Se non viene trovata nessuna corrispondenza, non viene eseguita alcuna azione.

Dopo aver elaborato tutte le regole che hanno corrispondenza con la riga (e può darsi che nessuna corrisponda), `awk` legge la riga successiva. Comunque si veda la [Sezione 7.4.8 \[L'istruzione `next`\]](#), [pagina 159](#), e anche si veda la [Sezione 7.4.9 \[L'istruzione `nextfile`\]](#), [pagina 160](#).) Si prosegue così finché il programma raggiunge la fine del file. Per esempio, il seguente programma `awk` contiene due regole:

```
/12/ { print $0 }
/21/ { print $0 }
```

La prima regola ha la stringa `'12'` da cercare e `'print $0'` come azione. La seconda regola ha la stringa `'21'` da cercare e ha ancora `'print $0'` come azione. L'azione di ciascuna regola è racchiusa in una coppia di parentesi graffe.

Questo programma stampa ogni riga che contiene la stringa '12' oppure la stringa '21'. Se una riga contiene entrambe le stringhe, è stampata due volte, una volta per ogni regola.

Questo è ciò che capita se eseguiamo questo programma sui nostri file-dati, `mail-list` e `inventory-shipped`:

```
$ awk '/12/ { print $0 }
>      /21/ { print $0 }' mail-list inventory-shipped
+ Anthony      555-3412      anthony.asserturo@hotmail.com      A
+ Camilla      555-2912      camilla.infusarum@skynet.be      R
+ Fabius       555-1234      fabius.undevicesimus@ucb.edu      F
+ Jean-Paul    555-2127      jeanpaul.campanorum@nyu.edu      R
+ Jean-Paul    555-2127      jeanpaul.campanorum@nyu.edu      R
+ Jan  21  36  64 620
+ Apr  21  70  74 514
```

Si noti che la riga che inizia con 'Jean-Paul' nel file `mail-list` è stata stampata due volte, una volta per ogni regola.

## 1.5 Un esempio più complesso

Dopo aver imparato a eseguire alcuni semplici compiti, vediamo cosa possono fare i tipici programmi `awk`. Questo esempio mostra come `awk` può essere usato per riassumere, selezionare e riordinare l'output di un altro comando. Sono usate funzionalità di cui non si è ancora parlato, quindi non ci si deve preoccupare se alcuni dettagli risulteranno oscuri:

```
ls -l | awk '$6 == "Nov" { somma += $5 }
            END { print somma }'
```

Questo comando stampa il numero totale di byte in tutti i file contenuti nella directory corrente, la cui data di modifica è novembre (di qualsiasi anno). La parte '`ls -l`' dell'esempio è un comando di sistema che fornisce un elenco dei file in una directory, con anche la dimensione di ogni file e la data di ultima modifica. Il suo output è del tipo:

```
-rw-r--r--  1 arnold  user   1933 Nov  7 13:05 Makefile
-rw-r--r--  1 arnold  user  10809 Nov  7 13:03 awk.h
-rw-r--r--  1 arnold  user    983 Apr 13 12:14 awk.tab.h
-rw-r--r--  1 arnold  user  31869 Jun 15 12:20 awkgram.y
-rw-r--r--  1 arnold  user  22414 Nov  7 13:03 awk1.c
-rw-r--r--  1 arnold  user  37455 Nov  7 13:03 awk2.c
-rw-r--r--  1 arnold  user  27511 Dec  9 13:07 awk3.c
-rw-r--r--  1 arnold  user   7989 Nov  7 13:03 awk4.c
```

Il primo campo contiene le autorizzazioni di lettura/scrittura [r/w], il secondo il numero dei collegamenti al file [cioè il numero di nomi con cui il file è conosciuto], e il terzo campo identifica il proprietario del file. Il quarto campo identifica il gruppo a cui appartiene il file. Il quinto campo contiene la dimensione del file, in byte. Il sesto, settimo e ottavo campo contengono il mese, il giorno e l'ora, rispettivamente, in cui il file è stato modificato. Infine, il nono campo contiene il nome-file.

L'espressione '`$6 == "Nov"`' nel nostro programma `awk` controlla se il sesto campo dell'output di '`ls -l`' corrisponda alla stringa 'Nov'. Ogni volta che una riga ha la stringa 'Nov' come suo sesto campo, `awk` esegue l'azione '`somma += $5`'. Questo aggiunge il quinto

campo (la dimensione del file) alla variabile `somma`. Come risultato, quando `awk` ha finito di leggere tutte le righe in input, `somma` contiene la somma totale delle dimensioni dei file che corrispondono al criterio di ricerca. (Ciò funziona contando sul fatto che le variabili `awk` sono automaticamente inizializzate a zero.)

Dopo che l'ultima riga dell'output di `ls` è stata elaborata, la regola `END` viene eseguita e viene stampato il valore di `somma`. In questo esempio, il valore di `somma` è 80600.

Queste tecniche più avanzate di `awk` sono trattate in sezioni successive (si veda la [Sezione 7.3 \[Azioni\], pagina 152](#)). Prima di poter passare a una programmazione più avanzata con `awk`, è necessario sapere come `awk` interpreta i file in input e visualizza quelli in output. Modificando campi e usando l'istruzione `print` è possibile produrre dei rapporti molto utili ed esteticamente gradevoli.

## 1.6 Istruzioni e righe in `awk`

Molto spesso, ogni riga di un programma `awk` è un'istruzione a sé stante o una regola isolata, come:

```
awk '/12/ { print $0 }
    /21/ { print $0 }' mail-list inventory-shipped
```

Comunque, `gawk` ignora i ritorni a capo dopo ognuno di questi simboli e istruzioni:

```
, { ? : || && do else
```

Un ritorno a capo in ogni altro punto del programma è considerato come la fine di un'istruzione.<sup>2</sup>

Volendo dividere una sola istruzione su due righe in un punto in cui andando a capo sarebbe considerata conclusa, è possibile *continuare* nella riga successiva terminando la prima riga con un carattere di barra inversa (`'\'`). La barra inversa dev'essere l'ultimo carattere sulla riga, per essere riconosciuto come un carattere di continuazione. Una barra inversa è consentita in ogni parte dell'istruzione, anche in mezzo a una stringa o a un'espressione regolare. Per esempio:

```
awk '/Questa espressione regolare è troppo lunga, quindi\
    la continuiamo sulla riga seguente/ { print $1 }'
```

Non abbiamo quasi mai usato la continuazione tramite barra inversa nei nostri programmi di esempio. `gawk` non pone limiti alla lunghezza di una riga, quindi la continuazione tramite barra inversa non è mai strettamente necessaria; serve soltanto a migliorare la leggibilità del programma. Per la stessa ragione, ma anche per amore di chiarezza, abbiamo tenuto concise molte istruzioni nei programmi presentati in questo libro. La continuazione tramite barra inversa è molto utile quando il proprio programma `awk` si trova in un file sorgente separato, invece di essere immesso nella riga di comando. Si noti anche che molte implementazioni di `awk` presentano delle differenze su dove è possibile usare la continuazione tramite barra inversa. Per esempio, potrebbero non consentire di spezzare una costante di tipo stringa usando la continuazione tramite barra inversa. Quindi, per ottenere la massima portabilità

---

<sup>2</sup> Il `'?'` e i `':'` elencati sopra sono usati nell'espressione condizionale in tre parti descritte in [Sezione 6.3.4 \[Espressioni condizionali\], pagina 137](#). Il cambio di riga dopo `'?'` e i `':'` è un'estensione minore in `gawk`; specificando `--posix` come opzione (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), quest'estensione non è valida.

dei propri programmi **awk**, è meglio non spezzare le righe nel mezzo di un'espressione regolare o di una stringa.

**ATTENZIONE:** *la continuazione tramite barra inversa non funziona come sopra descritto nella C shell.* Funziona per programmi **awk** contenuti in file e per programmi sulla riga di comando, *ammesso* che si stia usando una shell conforme a POSIX, come la Unix Bourne shell o Bash. Ma la C shell si comporta in maniera diversa! In quel caso, occorre usare due barre inverse consecutive, in fondo alla riga. Si noti anche che quando si usa la C shell *ogni* andata a capo nel vostro programma **awk** deve essere indicata con una barra inversa. Per esempio:

```
% awk 'BEGIN { \
?   print \
?       "ciao, mondo" \
? }'
└─ ciao, mondo
```

Qui, il '%' e il '?' sono i prompt primario e secondario della C shell, analogamente a quelli usati nella shell standard '\$' e '>'.

Si confronti l'esempio precedente, come viene scritto in una shell conforme a POSIX:

```
$ awk 'BEGIN {
>   print \
>       "ciao, mondo"
> }'
└─ ciao, mondo
```

**awk** è un linguaggio orientato alla riga. L'azione relativa a ogni regola deve iniziare sulla stessa riga del criterio di selezione. Per avere criterio di selezione e azione su righe separate, si *deve* usare la continuazione tramite barra inversa; non si può fare diversamente.

Un'altra cosa da tener presente è che la continuazione tramite barra inversa e i commenti non possono essere frammisti. Non appena **awk** incontra un '#' che inizia un commento, ignora *tutto* il resto della riga. Per esempio:

```
$ gawk 'BEGIN { print "Non allarmarti" # una amichevole \
>                               regola BEGIN
> }'
[error] gawk: riga com.:2:                regola BEGIN
[error] gawk: riga com.:2:                ^ syntax error
```

In questo caso, parrebbe che la barra inversa continui il commento sulla riga successiva. Invece, la combinazione barra inversa-ritorno a capo non viene per nulla notata, in quanto "nascosta" all'interno del commento. Quindi, il **BEGIN** è marcato come errore di sintassi.

Quando le istruzioni **awk** all'interno di una regola sono brevi, si potrebbe metterle più d'una su una riga sola. Ciò è possibile separando le istruzioni con un punto e virgola (;). Questo vale anche per le regole stesse. Quindi, il programma visto all'inizio di questa sezione poteva essere scritto anche così:

```
/12/ { print $0 } ; /21/ { print $0 }
```

**NOTA BENE:** La possibilità che più regole coesistano sulla stessa riga, se sono separate da un punto e virgola, non esisteva nel linguaggio **awk** originale; è stata

aggiunta per congruenza con quanto è consentito per le istruzioni all'interno di un'azione.

## 1.7 Altre funzionalità di awk

Il linguaggio **awk** mette a disposizione un numero di variabili *built-in*, o *predefinite*, che il programma dell'utente può usare per ottenere informazioni da **awk**. Ci sono pure altre variabili che il programma può impostare, per definire come **awk** deve gestire i dati.

Inoltre, **awk** mette a disposizione parecchie funzioni predefinite [*built-in*] per effettuare calcoli di tipo comune e operazioni che agiscono sulle stringhe di caratteri. **gawk** mette a disposizione funzioni predefinite per gestire le marcature temporali, per effettuare manipolazioni a livello di bit, per tradurre stringhe al momento dell'esecuzione del programma (internazionalizzazione), per determinare qual è il tipo di una variabile, e per ordinare dei vettori.

Nel seguito della presentazione del linguaggio **awk**, saranno introdotte molte delle variabili e parecchie funzioni. Esse sono descritte sistematicamente in [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162, e in [Capitolo 9 \[Funzioni\]](#), pagina 195.

## 1.8 Quando usare gawk

Ora che abbiamo visto qualcosa di quel che **awk** è in grado di fare, ci si potrà chiedere come **awk** potrebbe tornare utile. Usando programmi di utilità, criteri di ricerca sofisticati, separatori di campo, istruzioni aritmetiche, e altri criteri di selezione, è possibile produrre degli output molto più complessi. Il linguaggio **awk** è molto utile per fornire dei tabulati partendo da grandi quantità di dati grezzi, per esempio riassumendo informazioni dall'output di altri programmi di utilità come **ls**. (Si veda la [Sezione 1.5 \[Un esempio più complesso\]](#), pagina 27.)

I programmi scritti con **awk** sono normalmente molto più corti dei loro equivalenti in altri linguaggi. Ciò rende i programmi **awk** facili da comporre e da utilizzare. Spesso i programmi **awk** possono essere scritti al volo a terminale, usati una volta sola e buttati via. Poiché i programmi **awk** sono interpretati, si può evitare la (normalmente laboriosa) parte di compilazione nel ciclo tipico dello sviluppo software, ossia edita-compila-prova-correggi.

In **awk** sono stati scritti programmi complessi, compreso un assembler completo, piattaforma per microprocessori a 8-bit (si veda il [\[Glossario\]](#), pagina 515, per maggiori informazioni), e un assembler di microcodice per un computer dedicato esclusivamente al linguaggio Prolog. Le possibilità dell'originale **awk** erano messe a dura prova da programmi di questa complessità, ma le versioni moderne sono più robuste.

Se capita di scrivere programmi **awk** più lunghi di, diciamo, qualche centinaio di righe, si potrebbe considerare la possibilità di usare un linguaggio di programmazione differente da **awk**. La shell consente di ricercare stringhe ed espressioni regolari; inoltre consente di usare in maniera efficace i comandi di utilità del sistema. Python offre un piacevole equilibrio tra la facilità di una programmazione ad alto livello, e la possibilità di interagire a livello di sistema operativo.<sup>3</sup>

---

<sup>3</sup> Altri linguaggi di *script* popolari comprendono Ruby e Perl.

## 1.9 Sommario

- I programmi in **awk** consistono di coppie di *criterio di ricerca–azione*.
- Un’azione senza una *condizione di ricerca* viene sempre eseguita. L’azione di default per una condizione mancante è `{ print $0 }`.
- Usare `awk 'programma' file` oppure `awk -f file-programma file` per eseguire **awk**.
- Si può usare la notazione speciale `#!` nella prima riga per creare programmi **awk** che siano eseguibili direttamente.
- I commenti nei programmi **awk** iniziano con `#` e continuano fino alla fine della stessa riga.
- Prestare attenzione ai problemi con gli apici nei programmi **awk** che facciano parte di uno *script* della shell (o di un file `.BAT` di MS-Windows).
- Si può usare la continuazione tramite barra inversa per continuare righe di codice sorgente. Le righe sono continuate automaticamente dopo i simboli virgola, parentesi aperta, punto interrogativo, punto e virgola, `||`, `&&`, `do` ed `else`.



## 2 Eseguire awk e gawk

Questo capitolo tratta di come eseguire **awk**, delle opzioni da riga di comando, sia quelle dello standard POSIX che quelle specifiche di **gawk**, e di cosa fanno **awk** e **gawk** con gli argomenti che non sono opzioni. Prosegue poi spiegando come **gawk** cerca i file sorgenti, leggendo lo standard input assieme ad altri file, le variabili d'ambiente di **gawk**, lo stato di ritorno di **gawk**, l'uso dei file inclusi, e opzioni e/o funzionalità obsolete e non documentate.

Molte delle opzioni e funzionalità qui descritte sono trattate con maggior dettaglio nei capitoli successivi del libro; gli argomenti presenti in questo capitolo che al momento non interessano si possono tranquillamente saltare.

### 2.1 Come eseguire awk

Ci sono due modi di eseguire **awk**: con un programma esplicito o con uno o più file di programma. Qui è mostrata la sintassi di entrambi; le voci racchiuse tra [...] sono opzionali:

```
awk [opzioni] -f file_di_programma [--] file ...
awk [opzioni] [--] 'programma' file ...
```

In aggiunta alle tradizionali opzioni di una sola lettera in stile POSIX, **gawk** consente anche le opzioni estese GNU.

È possibile invocare **awk** con un programma vuoto:

```
awk '' file_dati_1 file_dati_2
```

Fare così ha comunque poco senso; **awk** termina silenziosamente quando viene fornito un programma vuoto. Se è stato specificato **--lint** sulla riga di comando, **gawk** emette un avviso che avverte che il programma è vuoto.



### 2.2 Opzioni sulla riga di comando

Le opzioni sono precedute da un trattino e consistono in un unico carattere. Le opzioni estese in stile GNU sono precedute da un doppio trattino e consistono in una parola chiave. La parola chiave può essere abbreviata, a condizione che l'abbreviazione identifichi univocamente l'opzione. Se l'opzione prevede un argomento, la parola chiave è immediatamente seguita da un segno di uguale (=) e dal valore dell'argomento, oppure la parola chiave e il valore dell'argomento sono separati da spazi. Se un'opzione con un valore viene immessa più di una volta, l'ultimo valore è quello che conta.

Ogni opzione estesa di **gawk** ha una corrispondente opzione breve in stile POSIX. Le opzioni estese e brevi sono intercambiabili in tutti i contesti. L'elenco seguente descrive le opzioni richieste dallo standard POSIX:

**-F fs**

**--field-separator fs**

Imposta la variabile **FS** a *fs* (si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\]](#), pagina 71).

**-f file-sorgente**

**--file file-sorgente**

Legge il sorgente del programma **awk** da *file-sorgente* anziché prenderlo dal primo argomento che non è un'opzione. Quest'opzione può essere data più volte;

il programma `awk` è formato dalla concatenazione del contenuto di ogni *file-sorgente* specificato.

`-v var=val`

`--assign var=val`

Imposta la variabile `var` al valore `val` *prima* che inizi l'esecuzione del programma. Tali valori di variabile sono disponibili all'interno della regola `BEGIN` (si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\]](#), pagina 40).

L'opzione `-v` può impostare una sola variabile per volta, ma può essere usata più di una volta, impostando ogni volta una variabile differente, in questo modo: `'awk -v pippo=1 -v pluto=2 ...'`.

**ATTENZIONE:** Usare `-v` per impostare valori di variabili predefinite può condurre a risultati sorprendenti. `awk` reimposterà i valori di quelle variabili secondo le sue necessità, anche ignorando eventuali valori iniziali che possono essere stati assegnati.

`-W gawk-opt`

Fornisce un'opzione specifica dell'implementazione. Questa è la convenzione POSIX per fornire opzioni specifiche dell'implementazione. Queste opzioni hanno anche una corrispondente opzione estesa scritta in stile GNU. Si noti che le opzioni estese possono essere abbreviate, sempre che le abbreviazioni siano univoche. L'elenco completo delle opzioni specifiche di `gawk` è riportato di seguito.

`--`

Segnale della fine delle opzioni da riga di comando. I seguenti argomenti non sono trattati come opzioni anche se iniziano con `'-'`. Questa interpretazione di `--` segue le convenzioni POSIX per l'analisi degli argomenti.

È utile se si hanno nomi-file che iniziano con `'-'`, o negli *script* di shell, se si hanno nomi-file che devono essere specificati dall'utente che potrebbero iniziare con `'-'`. È utile anche per passare opzioni al programma `awk`; si veda [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), pagina 263.

L'elenco che segue descrive le opzioni specifiche di `gawk`:

`-b`

`--characters-as-bytes`

Fa sì che `gawk` tratti tutti i dati in input come caratteri di un solo byte. In aggiunta, tutto l'output scritto con `print` o `printf` viene trattato come composto da caratteri contenuti in un solo byte.

Normalmente, `gawk` segue lo standard POSIX e cerca di elaborare i suoi dati di input in accordo con la localizzazione corrente (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141). Questo spesso può comportare la conversione di caratteri multibyte in caratteri estesi (internamente), e può creare problemi o confusione se i dati di input non contengono caratteri multibyte validi. Quest'opzione è una maniera facile di dire a `gawk`: "Giù le mani dai miei dati!".

**-c**

**--traditional**

Specifica la *modalità di compatibilità*, nella quale le estensioni GNU al linguaggio **awk** sono disabilitate; in questo modo **gawk** si comporta proprio come la versione di BWK **awk**.

Si veda la [Sezione A.5 \[Estensioni di gawk non in POSIX awk\]](#), pagina 464, che riassume le estensioni. Si veda anche [Sezione C.1 \[Compatibilità all'indietro e debug\]](#), pagina 499.

**-C**

**--copyright**

Stampa la versione ridotta della General Public License ed esce.

**-d[file]**

**--dump-variables[=file]**

Stampa una lista ordinata di variabili globali, i loro tipi, e i valori finali in *file*. Se non viene fornito alcun *file*, stampa questa lista in un file chiamato **awkvars.out** nella directory corrente. Non sono consentiti spazi tra **-d** e *file*, se *file* viene specificato.

Avere una lista di tutte le variabili globali è un buon modo per cercare refusi nei propri programmi. Si può usare quest'opzione anche se si ha un grosso programma con tantissime funzioni, e si vuol essere sicuri che le funzioni non usino inavvertitamente variabili globali che sarebbero dovute essere locali (questo è un errore particolarmente facile da fare con nomi di variabile semplici come *i*, *j*, etc.).

**-D[file]**

**--debug[=file]**

Abilita l'esecuzione del debug di programmi **awk** (si veda la [Sezione 14.1 \[Introduzione al debugger di gawk\]](#), pagina 361). Per default, il debugger legge i comandi interattivamente dalla tastiera (standard input). L'argomento opzionale *file* consente di specificare un file con una lista di comandi per il debugger da eseguire in maniera non interattiva. Non sono consentiti spazi tra **-D** e *file*, se *file* viene indicato.

**-e testo-del-programma**

**--source testo-del-programma**

Fornisce del codice sorgente nel *testo-del-programma*. Quest'opzione consente di combinare il codice sorgente contenuto in file col codice sorgente immesso sulla riga di comando. Questo è particolarmente utile quando si hanno funzioni di libreria che si vogliono usare dai programmi da riga di comando (si veda la [Sezione 2.5.1 \[Ricerca di programmi awk in una lista di directory.\]](#), pagina 42).

**-E file**

**--exec file**

Simile a **-f**, legge il testo del programma **awk** da *file*. Ci sono due differenze rispetto a **-f**:

- Quest'opzione fa terminare l'elaborazione delle opzioni; qualsiasi altra cosa sulla riga di comando viene inoltrata direttamente al programma **awk**.

- Le variabili da riga di comando della forma `'var=value'` non sono ammesse.

Quest'opzione è particolarmente necessaria per le applicazioni World Wide Web CGI che passano argomenti attraverso le URL; l'uso di quest'opzione impedisce a un utente malintenzionato (o ad altri) di passare opzioni, assegnamenti o codice sorgente `awk` (con `-e`) all'applicazione CGI.<sup>1</sup> Quest'opzione dovrebbe essere usata con *script* `'#!'` (si veda la [Sezione 1.1.4 \[Programmi `awk` da eseguire come \*script\*\], pagina 19](#)), in questo modo:

```
#!/usr/local/bin/gawk -E
```

```
il programma awk è qui ...
```

`-g`

`--gen-pot`

Analizza il programma sorgente e genera un file GNU *gettext portable object template* sullo standard output per tutte le costanti di tipo stringa che sono state marcate come da tradurre. Si veda la [Capitolo 13 \[Internazionalizzazione con `gawk`\], pagina 349](#), per informazioni su quest'opzione.

`-h`

`--help`

Stampa un messaggio sull'“uso” riassumendo le opzioni brevi ed estese accettate da `gawk` ed esce.

`-i file-sorgente`

`--include file-sorgente`

Legge una libreria di sorgenti `awk` da *file-sorgente*. Quest'opzione è del tutto equivalente a usare la direttiva `@include` all'interno del proprio programma. È molto simile all'opzione `-f`, ma ci sono due differenze importanti. Primo, quando viene usata l'opzione `-i`, il sorgente del programma non viene caricato se è stato caricato in precedenza, mentre con `-f`, `gawk` carica sempre il file. Secondo, poiché quest'opzione è pensata per essere usata con librerie di codice, `gawk` non riconosce tali file come costituenti l'input del programma principale. Così, dopo l'elaborazione di un argomento `-i`, `gawk` si aspetta di trovare il codice sorgente principale attraverso l'opzione `-f` o sulla riga di comando.

`-l ext`

`--load ext`

Carica un'estensione dinamica denominata *ext*. Le estensioni sono memorizzate come librerie condivise di sistema. Quest'opzione ricerca la libreria usando la variabile d'ambiente `AWKLIBPATH`. Il suffisso corretto per la piattaforma in uso verrà fornito per default, perciò non è necessario specificarlo nel nome dell'estensione. La routine di inizializzazione dell'estensione dovrebbe essere denominata `dl_load()`. Un'alternativa è quella di usare la direttiva `@load` all'interno del programma per caricare una libreria condivisa. Questa funzionalità avanzata è descritta in dettaglio in [Capitolo 16 \[Scrivere estensioni per `gawk`\], pagina 395](#).

<sup>1</sup> per maggiori dettagli, si veda la Sezione 4.4 di [RFC 3875](#). Si veda anche [note esplicative spedite alla mailing list `gawk bug`](#).

`-L[valore]`

`--lint[=valore]`

Emette messaggi d'avvertimento relativi a costrutti dubbi o non portabili ad altre implementazioni di **awk**. Non sono consentiti spazi tra `-L` e *valore*, se viene indicato il *valore*. Alcuni avvertimenti vengono emessi quando **gawk** legge preliminarmente il programma. Altri vengono emessi quando il programma viene eseguito. Con l'argomento opzionale *'fatal'*, gli avvertimenti *lint* sono considerati come errori gravi. Potrebbe essere una misura drastica, però il suo uso incoraggerà certamente lo sviluppo di programmi **awk** più corretti. Con l'argomento opzionale *'invalid'*, vengono emessi solo gli avvertimenti relativi a quello che è effettivamente non valido (funzionalità non ancora completamente implementata).

Alcuni avvertimenti vengono stampati solo una volta, anche se i costrutti dubbi per i quali vengono emessi avvisi ricorrono diverse volte nel programma **awk**. Perciò, nell'eliminazione dei problemi rilevati da `--lint`, bisogna porre attenzione a cercare tutte le occorrenze di ogni costrutto inappropriato. Siccome i programmi **awk** generalmente sono brevi, questa non è un'operazione gravosa.

`-M`

`--bignum` Chiede il calcolo con precisione arbitraria sui numeri. Quest'opzione non ha alcun effetto se **gawk** non è compilato per l'uso delle librerie GNU MPFR e MP (si veda la [Capitolo 15 \[Calcolo con precisione arbitraria con gawk\]](#), pagina 379).

`-n`

`--non-decimal-data`

Abilita l'interpretazione automatica di valori ottali ed esadecimali nei dati di input (si veda la [Sezione 12.1 \[Consentire dati di input non decimali\]](#), pagina 331).

**ATTENZIONE:** Quest'opzione può generare gravi malfunzionamenti nei vecchi programmi. Usare con cautela. Si noti anche che quest'opzione potrebbe non essere più disponibile in una futura versione di **gawk**.

`-N`

`--use-lc-numeric`

Forza l'uso del carattere di separazione decimale della localizzazione quando analizza i dati in input (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141).

`-o[file]`

`--pretty-print[=file]`

Consente la stampa di una versione formattata elegantemente dei programmi **awk**. Implica l'opzione `--no-optimize`. Per default il programma di output viene creato in un file chiamato **awkprof.out** (si veda la [Sezione 12.5 \[Profilare i propri programmi awk\]](#), pagina 343). L'argomento opzionale *file* consente di specificare un nome-file differente per l'output. Non sono consentiti spazi tra `-o` e *file*, se *file* viene indicato.

**NOTA:** Nel passato, quest'opzione eseguiva anche il programma. Ora non è più così.

-O

--optimize

Abilita le ottimizzazioni di default nella rappresentazione interna del programma. Attualmente, questo comprende delle semplificazioni nell'uso di costanti e l'eliminazione delle code di chiamata nelle funzioni ricorsive [sostituzione della chiamata di funzione con dei salti diretti alla funzione].

Queste ottimizzazioni sono abilitate per default. Quest'opzione rimane disponibile per compatibilità all'indietro. Tuttavia può essere usata per annullare l'effetto di una precedente opzione `-s` (si veda più sotto in questa lista).

-p[*file*]

--profile[=*file*]

Abilita la creazione del profilo di esecuzione di programmi `awk` (si veda la [Sezione 12.5 \[Profilare i propri programmi awk\]](#), pagina 343). Implicitamente viene forzata l'opzione `--no-optimize`. Per default, i profili vengono creati in un file chiamato `awkprof.out`. L'argomento opzionale *file* consente di specificare un altro nome-file per il file del profilo. Non sono consentiti spazi tra `-p` e *file*, se viene indicato un *file*.

Il profilo contiene il numero di esecuzioni di ogni istruzione sul margine sinistro e il conteggio delle chiamate di funzione per ogni funzione.

-P

--posix

Opera in modalità POSIX rigorosa. Disabilita tutte le estensioni di `gawk` (proprio come `--traditional`) e disabilita tutte le estensioni non consentite da POSIX.

Si veda la [Sezione A.7 \[Sommario Estensioni Comuni\]](#), pagina 473, per un sommario delle estensioni di `gawk` che sono disabilitate da quest'opzione. Inoltre, vengono applicate le seguenti restrizioni:

- I ritorni a capo non sono consentiti dopo '?' o ':' (si veda la [Sezione 6.3.4 \[Espressioni condizionali\]](#), pagina 137).
- Specificando `'-Ft'` sulla riga di comando non si imposta il valore della variabile `FS` a un singolo carattere TAB (si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\]](#), pagina 71).
- Il carattere di separatore decimale della localizzazione è usato per analizzare i dati di input (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141).

Se si forniscono entrambe le opzioni `--traditional` e `--posix` sulla riga di comando, `--posix` ha la precedenza. Se vengono fornite entrambe le opzioni `gawk` emette un avviso.

-r

--re-interval

Consente le espressioni di intervallo (si veda la [Sezione 3.3 \[Operatori di espressioni regolari\]](#), pagina 52) nelle espressioni regolari. Questo è ora il comportamento di default di `gawk`. Tuttavia, quest'opzione rimane (sia per retrocompatibilità che per l'uso in combinazione con `--traditional`).

-s

--no-optimize

Disabilita le opzioni di ottimizzazione di default di **gawk** effettuate sulla rappresentazione interna del programma.

-S

--sandbox

Disabilita la funzione **system()**, la ridirezione dell'input con **getline**, la ridirezione dell'output con **print** e **printf**, e le estensioni dinamiche. È particolarmente utile quando si vogliono eseguire *script awk* da sorgenti dubbie e si vuol essere ricuri che gli *script* non abbiano accesso al sistema (oltre al file-dati di input specificato).

-t

--lint-old

Avvisa su costrutti che non sono disponibili nella versione originale di **awk** dalla versione 7 di Unix (si veda la [Sezione A.1 \[Differenze importanti tra V7 e System V Release 3.1\]](#), pagina 461).

-V

--version

Stampa informazioni sulla versione di questa specifica copia di **gawk**. Consente di determinare se la copia di **gawk** in uso è aggiornata rispetto a quello che è attualmente in distribuzione da parte della Free Software Foundation. È utile anche per la segnalazione di bug (si veda la [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493).

Ogni altra opzione, se è stato specificato il testo di un programma è contrassegnata come non valida con un messaggio di avvertimento, altrimenti è ignorata.

In modalità di compatibilità, come caso particolare, se il valore di *fs* fornito all'opzione **-F** è **'t'**, **FS** è impostata al carattere TAB (**"\t"**). Questo è vero solo per **--traditional** e non per **--posix** (si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\]](#), pagina 71).

L'opzione **-f** può essere usata più di una volta nella riga di comando. In questo caso, **awk** legge il sorgente del suo programma da tutti i file indicati, come se fossero concatenati assieme a formare un unico grande file. Questo è utile per creare librerie di funzioni di **awk**. Queste funzioni possono venir scritte una volta e in seguito recuperate da una posizione standard, invece di doverle includere in ogni singolo programma. L'opzione **-i** è simile in questo senso. (Come indicato in [Sezione 9.2.1 \[Come scrivere definizioni e cosa significano\]](#), pagina 224, i nomi di funzione devono essere univoci).

Con **awk** standard, le funzioni di libreria si possono ancora usare, anche se il programma è immesso dalla tastiera, specificando **'-f /dev/tty'**. Dopo aver scritto il programma, premere **Ctrl-d** (il carattere di fine file) per terminarlo. (Si potrebbe anche usare **'-f -'** per leggere il sorgente del programma dallo standard input, ma poi non si potrà usare lo standard input come sorgente di dati).

Siccome è scomodo usare il meccanismo di **awk** standard per combinare file sorgenti e programmi **awk** da riga di comando, **gawk** fornisce l'opzione **-e**. Questo non richiede di evitare l'uso dello standard input per immettere codice sorgente; consente di combinare

facilmente codice sorgente da riga di comando e da libreria (si veda la [Sezione 2.5.1 \[Ricerca di programmi `awk` in una lista di directory.\]](#), pagina 42). Come per `-f`, le opzioni `-e` e `-i` si possono usare più volte nella riga di comando.

Se non sono specificate opzioni `-f` o `-e`, `gawk` usa il primo argomento che non è un'opzione come testo del codice sorgente del programma.

Se la variabile d'ambiente `POSIXLY_CORRECT` esiste, `gawk` si comporta in modalità POSIX rigorosa, esattamente come se fosse stata fornita l'opzione `--posix`. Molti programmi GNU cercano questa variabile d'ambiente per eliminare estensioni che confliggono con POSIX, ma `gawk` si comporta in modo diverso: sopprime tutte le estensioni, anche quelle che non confliggono con POSIX, e funziona rigorosamente in modalità POSIX. Se viene fornita l'opzione `--lint` sulla riga di comando e `gawk` passa alla modalità POSIX a causa di `POSIXLY_CORRECT`, viene emesso un messaggio di avvertimento indicando che è attiva la modalità POSIX. Normalmente questa variabile si imposta nel file di avvio della shell a livello utente. Per una shell compatibile con Bourne (come Bash), queste righe andranno aggiunte nel file `.profile` della directory "home" dell'utente:

```
POSIXLY_CORRECT=true
export POSIXLY_CORRECT
```

Per una shell compatibile con C,<sup>2</sup> questa riga andrà aggiunta nel file `.login` nella directory "home" dell'utente:

```
setenv POSIXLY_CORRECT true
```

Avere `POSIXLY_CORRECT` impostata non è raccomandato per l'uso quotidiano, ma è utile per provare la portabilità dei programmi su altri ambienti.

## 2.3 Altri argomenti della riga di comando

Qualsiasi altro argomento sulla riga di comando è trattato normalmente come file in input da elaborare nell'ordine con cui è specificato. Comunque, un argomento che ha la forma `var=valore`, assegna il valore `valore` alla variabile `var`—non specifica affatto un file. (Si veda [Sezione 6.1.3.2 \[Assegnare una variabile dalla riga di comando\]](#), pagina 120.) Nel seguente esempio, `count=1` è un assegnamento di variabile, non un nome-file:

```
awk -f programma.awk file1 count=1 file2
```

Tutti gli argomenti da riga di comando sono resi disponibili al programma `awk` nel vettore `ARGV` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162). Opzioni da riga di comando e il testo del programma (se presente) sono esclusi da `ARGV`. Tutti gli altri argomenti, compresi gli assegnamenti di variabile, sono inclusi. Come ogni elemento di `ARGV` viene elaborato, `gawk` imposta `ARGIND` all'indice in `ARGV` dell'elemento corrente.

La modifica di `ARGC` e `ARGV` nel proprio programma `awk` consente di controllare come `awk` elabora i file in input; questo è descritto più dettagliatamente in [Sezione 7.5.3 \[Usare `ARGC` e `ARGV`\]](#), pagina 172.

La distinzione tra argomenti che sono nome-file e argomenti di assegnamento di variabili vien fatta quando `awk` deve aprire il successivo file di input. A quel punto dell'esecuzione, controlla la variabile nome-file per vedere se è piuttosto un assegnamento di variabile; se così è, `awk` imposta la variabile invece di leggere un file.

---

<sup>2</sup> Non raccomandato.

Dunque, le variabili ricevono effettivamente i valori loro assegnati dopo che tutti i file precedentemente specificati sono stati letti. In particolare, i valori delle variabili assegnati in questo modo *non* sono disponibili all'interno di una regola BEGIN (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148), poiché tali regole vengono eseguite prima che **awk** cominci a esaminare la lista degli argomenti.

I valori delle variabili dati sulla riga di comando sono elaborati per rimuovere sequenze di protezione (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50).



In alcune implementazioni di **awk** molto vecchie, quando un assegnamento di variabile capitava prima di un qualsiasi nome-file, l'assegnamento avveniva *prima* che fosse stata eseguita la regola BEGIN. Il comportamento di **awk** era in questo modo ambiguo; alcuni assegnamenti da riga di comando erano disponibili all'interno della regola BEGIN, mentre altri no. Sfortunatamente, alcune applicazioni finivano per essere dipendenti da questa “funzionalità”. Quando **awk** fu modificato per essere più coerente, fu aggiunta l'opzione **-v** a beneficio delle applicazioni che dipendevano dal vecchio comportamento.

La funzionalità dell'assegnamento di variabile è molto utile per assegnare valori a variabili come **RS**, **OFS**, e **ORS**, che controllano i formati di input e di output, prima di effettuare la scansione dei file-dati. È utile anche per effettuare passaggi multipli su un o stesso file-dati. Per esempio:

```
awk 'pass == 1 { pass 1 stuff }
    pass == 2 { pass 2 stuff }' pass=1 mydata pass=2 mydata
```

Una volta disponibile la funzionalità per assegnare una variabile, l'opzione **-F** per impostare il valore di **FS** non è più strettamente necessaria. Rimane per compatibilità all'indietro.

## 2.4 Come specificare lo standard input insieme ad altri file

Capita spesso di voler leggere lo standard input assieme ad altri file. Per esempio, leggere un file, leggere lo standard input derivante da una *pipe*, e poi leggere un altro file.

Il modo di indicare lo standard input, con tutte le versioni di **awk**, è quello di usare un segno meno o trattino da solo, **'-'**. Per esempio:

```
qualche_comando | awk -f ilmioprogramma.awk file1 - file2
```

In questo caso, **awk** legge prima **file1**, poi legge l'output di *qualche\_comando*, e infine legge **file2**.

Si può anche usare **"-"** per indicare lo standard input quando si leggono i file con **getline** (si veda la [Sezione 4.9.3 \[Usare getline da un file\]](#), pagina 85).

In aggiunta, **gawk** consente di specificare il nome-file speciale **/dev/stdin**, sia sulla riga di comando che quando si usa **getline**. Anche qualche altra versione di **awk** include questa funzionalità, ma non è standard. (Alcuni sistemi operativi prevedono un file **/dev/stdin** nel filesystem; comunque, **gawk** elabora sempre questo nome-file per conto suo [ossia non importa se il sistema operativo rende disponibile il file o no].)

## 2.5 Le variabili d'ambiente usate da gawk

Diverse variabili d'ambiente influiscono sul comportamento di **gawk**.

### 2.5.1 Ricerca di programmi awk in una lista di directory.

Nella maggior parte delle implementazioni di **awk** si deve indicare il percorso completo di ogni file di programma, a meno che il file non sia nella directory corrente. Con **gawk**, invece, se la variabile nome-file impostata con le opzioni **-f** o **-i** non contiene un separatore di directory '/', **gawk** cerca un file con quel nome in un elenco di directory (chiamato *percorso di ricerca*), scorrendole una per una.

Il percorso di ricerca è una stringa di nomi di directory separati da due punti<sup>3</sup>. **gawk** prende il percorso di ricerca dalla variabile d'ambiente **AWKPATH**. Se questa variabile non esiste, o se ha un come valore la stringa nulla, **gawk** usa un percorso di default (descritto tra poco).

La funzionalità del percorso di ricerca è particolarmente utile per costruire librerie di funzioni di **awk**. I file di libreria possono essere messi in una directory standard inclusa nel percorso di ricerca e richiamati sulla riga di comando con un nome-file breve. Altrimenti, si dovrebbe scrivere l'intero nome-file per ciascun file.

Usando l'opzione **-i**, o l'opzione **-f**, i programmi di **awk** scritti sulla riga di comando possono usare le funzionalità contenute nei file di libreria di **awk** (si veda il [Capitolo 10 \[Una libreria di funzioni awk\]](#), pagina 245). La ricerca del percorso non viene eseguita se **gawk** è in modalità di compatibilità, sia con l'opzione **--traditional** che con l'opzione **--posix**. Si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33.

Se il file del codice sorgente non viene trovato con una prima ricerca, il percorso viene cercato di nuovo dopo aver aggiunto il suffisso **.awk** al nome-file.

Il meccanismo di ricerca del percorso di **gawk** è simile a quello della shell. (Si veda *The Bourne-Again SHell manual*.) Un elemento nullo nel percorso indica la directory corrente. (Un elemento nullo è indicato iniziando o terminando il percorso con un segno di ':' oppure mettendo due ':' consecutivi ['::'].)

**NOTA:** Per includere la directory corrente nel percorso di ricerca, si può aggiungere **.** come un elemento del percorso di ricerca, oppure inserire un elemento nullo.

Diverse passate versioni di **gawk** avrebbero effettuato anche una ricerca esplicita nella directory corrente, prima o dopo aver esaminato il percorso di ricerca. A partire dalla versione 4.1.2, questo non vale più; se si desidera una ricerca nella directory corrente, è necessario aggiungere **.** esplicitamente, oppure aggiungendo un elemento nullo al percorso di ricerca.

Il valore di default di **AWKPATH** è **./usr/local/share/awk**.<sup>4</sup> Poiché **.** è incluso all'inizio, **gawk** cerca dapprima nella directory corrente, e poi in **/usr/local/share/awk**. In pratica, questo vuol dire che solo raramente ci sarà bisogno di cambiare il valore di **AWKPATH**.

Si veda la [Sezione B.2.2 \[File di inizializzazione della shell\]](#), pagina 484, per informazioni su funzioni che possono essere di aiuto per gestire la variabile **AWKPATH**.

<sup>3</sup> Punti e virgola in MS-Windows.

<sup>4</sup> La versione di **gawk** che state usando potrebbe usare una directory diversa; ciò dipende da come **gawk** è stato compilato e installato. La directory effettiva è il valore di **\$(datadir)** generato quando è stato configurato **gawk**. Non è comunque il caso di preoccuparsi per questo.

**gawk** memorizza il valore del percorso di ricerca in uso in `ENVIRON["AWKPATH"]`. Questo consente di aver accesso al valore del percorso di ricerca in uso all'interno di un programma **awk**.

Sebbene la variabile `ENVIRON["AWKPATH"]` possa essere cambiata anche all'interno di un programma **awk**, questo non modifica il comportamento del programma in esecuzione. Questo comportamento ha una sua logica: la variabile d'ambiente `AWKPATH` è usata per trovare i file sorgenti del programma; una volta che il programma è in esecuzione, tutti i file sono stati trovati, e **gawk** non ha più bisogno di usare `AWKPATH`.

### 2.5.2 Ricerca di librerie condivise awk su varie directory.

La variabile d'ambiente `AWKLIBPATH` è simile alla variabile `AWKPATH`, ma è usata per ricercare estensioni caricabili (memorizzate come librerie condivise di sistema) specificate con l'opzione `-l`, anziché file sorgenti. Se l'estensione non viene trovata, il percorso viene cercato nuovamente dopo aver aggiunto il suffisso per la libreria condivisa appropriato per la piattaforma. Per esempio, sui sistemi GNU/Linux viene usato il suffisso `‘.so’`. Il percorso di ricerca specificato è usato anche attraverso la direttiva `@load` (si veda la [Sezione 2.8 \[Caricare librerie condivise nel proprio programma\]](#), pagina 47).

Se la variabile d'ambiente `AWKLIBPATH` non esiste, o se ha come valore la stringa nulla, **gawk** usa un percorso di ricerca di default; questo normalmente vale `‘/usr/local/lib/gawk’`, anche se il suo valore può essere diverso, a seconda di come è stato installato **gawk**.

Si veda la [Sezione B.2.2 \[File di inizializzazione della shell\]](#), pagina 484, per informazioni su funzioni che possono essere di aiuto per gestire la variabile `AWKPATH`.

**gawk** memorizza il valore del percorso di ricerca in uso in `ENVIRON["AWKLIBPATH"]`. Questo consente di aver accesso al valore del percorso di ricerca in uso all'interno di un programma **awk**.

### 2.5.3 Le variabili d'ambiente.

Molte altre variabili d'ambiente influenzano il comportamento di **gawk**, ma esse sono più specializzate. Quelle dell'elenco seguente sono quelle più utili agli utenti normali:

#### `GAWK_MSEC_SLEEP`

Specifica l'intervallo tra due tentativi di riconnessione, in millisecondi. Sui sistemi che non prevedono la chiamata di sistema `usleep()`, il valore è arrotondato a un numero intero di secondi.

#### `GAWK_READ_TIMEOUT`

Specifica per quanto tempo, in millisecondi, **gawk** aspetta l'input prima di emettere un messaggio di errore.

#### `GAWK_SOCKET_RETRIES`

Controlla il numero di volte che **gawk** cerca di ristabilire una connessione bidirezionale TCP/IP (*socket*) prima di rinunciare a farlo. Si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341. Si noti che quando è attiva l'opzione di continuazione dopo errori di I/O (si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112), **gawk** tenta di aprire un *socket* TCP/IP soltanto una volta.

**POSIXLY\_CORRECT**

Provoca il passaggio di **gawk** alla modalità di compatibilità POSIX, disabilitando tutte le estensioni tradizionali e GNU. Si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33.

Le variabili d'ambiente nell'elenco che segue sono utili soprattutto agli sviluppatori di **gawk** per il collaudo e la messa a punto del programma. Sono soggette a cambiamenti. Le variabili sono:

**AWKBUFSIZE**

Questa variabile riguarda solo **gawk** installato su sistemi conformi a POSIX. Col valore di **'exact'**, **gawk** usa la dimensione di ogni file di input come dimensione del buffer di memoria da allocare per I/O. Altrimenti, il valore dovrebbe essere un numero, e **gawk** usa questo numero come dimensione del buffer da allocare. (Quando questa variabile non è impostata, **gawk** usa la più piccola tra le dimensioni del file e la dimensione del blocco di "default", che normalmente è la dimensione del blocco I/O del filesystem).

**AWK\_HASH** Se questa variabile è impostata con un valore di **'gst'**, **gawk** usa la funzione hash di GNU Smalltalk per gestire i vettori. Questa funzione può essere leggermente più veloce della funzione standard.

**AWKREADFUNC**

Se questa variabile esiste, **gawk** legge i file sorgenti una riga per volta, anziché a blocchi. Questa variabile è presente per problemi di debug su filesystem di sistemi operativi non POSIX, dove l'I/O è elaborato a record, non a blocchi.

**GAWK\_MSG\_SRC**

Se questa variabile esiste, **gawk** include il nome-file e il numero di riga all'interno del codice sorgente **gawk** dal quale sono stati generati i messaggi di avvertimento o i messaggi di errore grave. Il suo intento è quello di aiutare a isolare l'origine di un messaggio, poiché ci possono essere più righe di codice che producono lo stesso messaggio di avvertimento o di errore.

**GAWK\_LOCALE\_DIR**

Specifica la posizione dei file oggetto compilati contenenti la traduzione dei messaggi emessi da **gawk** stesso. Questa variabile è passata alla funzione `bindtextdomain()` nella fase di partenza di **gawk**.

**GAWK\_NO\_DFA**

Se questa variabile esiste, **gawk** non usa il riconoscitore di espressioni regolari ASFD [automa a stati finiti deterministico] per i tipi di test di corrispondenza. Questo può causare un rallentamento di **gawk**. Il suo intento è quello di aiutare a isolare le differenze tra i due riconoscitori di espressioni regolari che **gawk** usa internamente (non dovrebbero esserci differenze, ma a volte la teoria non coincide con la pratica).

**GAWK\_STACKSIZE**

Specifica di quanto **gawk** dovrebbe accrescere il suo stack di valutazione interno, all'occorrenza.

**INT\_CHAIN\_MAX**

Specifica il numero massimo previsto di elementi che `gawk` mantiene su una catena hash per gestire i vettori indicizzati da numeri interi.

**STR\_CHAIN\_MAX**

Specifica il numero massimo previsto di elementi che `gawk` mantiene su una catena hash per gestire i vettori indicizzati da stringhe.

**TIDYMEM**

Se questa variabile esiste, `gawk` usa le chiamate di libreria `mtrace()` della *GNU C library* per aiutare a scoprire possibili sprechi di memoria.

## 2.6 Il codice di ritorno all'uscita da `gawk`

Se l'istruzione `exit` viene usata con un valore (si veda la [Sezione 7.4.10 \[L'istruzione `exit`\]](#), [pagina 161](#)), `gawk` termina l'esecuzione con il valore numerico specificato.

Altrimenti, se non ci sono stati problemi durante l'esecuzione, `gawk` esce col valore della costante C `EXIT_SUCCESS`, che normalmente è zero.

Se si verifica un errore, `gawk` esce col valore della costante C `EXIT_FAILURE`, che normalmente è uguale a uno.

Se `gawk` esce a causa di un errore grave, il codice di ritorno è due. Sui sistemi non POSIX questo valore può essere mappato a `EXIT_FAILURE`.

## 2.7 Come includere altri file nel proprio programma

Questa sezione descrive una funzionalità disponibile solo in `gawk`.

La direttiva `@include` può essere usata per leggere file sorgenti di `awk` esterni. Questo dà la possibilità di suddividere file sorgenti di `awk` di grandi dimensioni in porzioni più piccole e più maneggevoli, e anche di riutilizzare codice `awk` di uso comune da diversi *script awk*. In altre parole, si possono raggruppare funzioni di `awk` usate per eseguire determinati compiti all'interno di file esterni. Questi file possono essere usati proprio come librerie di funzioni, usando la direttiva `@include` assieme alla variabile d'ambiente `AWKPATH`. Si noti che i file sorgenti possono venire inclusi anche usando l'opzione `-i`.

Vediamolo con un esempio. Iniziamo con due *script awk* (banali), che chiameremo `test1` e `test2`. Questo è lo *script test1*:

```
BEGIN {
    print "Questo è lo script test1."
}
```

e questo è `test2`:

```
@include "test1"
BEGIN {
    print "Questo è lo script test2."
}
```

L'esecuzione di `gawk` con `test2` produce il seguente risultato:

```
$ gawk -f test2
+ Questo è lo script test1.
+ Questo è lo script test2.
```

**gawk** esegue lo *script* `test2`, il quale include `test1`, usando la direttiva `@include`. Così, per includere file sorgenti di **awk** esterni, basta usare `@include` seguito dal nome del file da includere, racchiuso tra doppi apici.

**NOTA:** Si tenga presente che questo è un costrutto del linguaggio e che nome-file non può essere una variabile di tipo stringa, ma solo una costante di tipo letterale racchiusa tra doppi apici.

I file da includere possono essere nidificati; p.es., dato un terzo *script*, che chiameremo `test3`:

```
@include "test2"
BEGIN {
    print "Questo è lo script test3."
}
```

L'esecuzione di **gawk** con lo *script* `test3` produce i seguenti risultati:

```
$ gawk -f test3
+ Questo è lo script test1.
+ Questo è lo script test2.
+ Questo è lo script test3.
```

Il nome-file, naturalmente, può essere un nome di percorso. Per esempio:

```
@include "../funzioni_di_i_o"
```

e:

```
@include "/usr/awklib/network"
```

sono entrambi percorsi validi. La variabile d'ambiente `AWKPATH` può rivestire grande importanza quando si usa `@include`. Le stesse regole per l'uso della variabile d'ambiente `AWKPATH` nelle ricerche da riga di comando (si veda la [Sezione 2.5.1 \[Ricerca di programmi awk in una lista di directory.\]](#), pagina 42) si applicano anche a `@include`.

Questo è di grande aiuto nella costruzione di librerie di funzioni di **gawk**. Se si ha uno *script* di grandi dimensioni contenente utili funzioni **awk** di uso comune, lo si può suddividere in file di libreria e mettere questi file in una directory dedicata. In seguito si possono includere queste "librerie" usando il percorso completo dei file, o impostando opportunamente la variabile d'ambiente `AWKPATH` e quindi usando `@include` con la sola parte del percorso completo che designa il file. Naturalmente, si possono tenere i file di libreria in più di una directory; più è complesso l'ambiente di lavoro, più directory possono essere necessarie per organizzare i file da includere.

Vista la possibilità di specificare opzioni `-f` multiple, il meccanismo `@include` non è strettamente necessario. Comunque, la direttiva `@include` può essere d'aiuto nel costruire programmi **gawk** autosufficienti, riducendo così la necessità di scrivere righe di comando complesse e tediose. In particolare, `@include` è molto utile per scrivere *script* CGI eseguibili da pagine web.

Come è stato detto in [Sezione 2.5.1 \[Ricerca di programmi awk in una lista di directory.\]](#), pagina 42, i file sorgenti vengono sempre cercati nella directory corrente, prima di eseguire la ricerca in `AWKPATH`; questo si applica anche ai file indicati con `@include`.

## 2.8 Caricare librerie condivise nel proprio programma

Questa sezione descrive una funzionalità disponibile solo in **gawk**.

La direttiva `@load` può essere usata per leggere estensioni di **awk** esterne (memorizzate come librerie condivise di sistema). Questo consente di collegare del codice compilato che può offrire prestazioni migliori o dare l'accesso a funzionalità estese non incluse nel linguaggio **awk**. La variabile `AWKLIBPATH` viene usata per ricercare l'estensione. Usare `@load` è del tutto equivalente a usare l'opzione da riga di comando `-l`.

Se l'estensione non viene trovata in `AWKLIBPATH`, viene effettuata un'altra ricerca dopo aver aggiunto al nome-file il suffisso della libreria condivisa comunemente in uso per la piattaforma corrente. Per esempio, sui sistemi GNU/Linux viene usato il suffisso `'.so'`:

```
$ gawk '@load "ordchr"; BEGIN {print chr(65)}'
+ A
```

Questo è equivalente all'esempio seguente:

```
$ gawk -lordchr 'BEGIN {print chr(65)}'
+ A
```

Per l'uso da riga di comando è più conveniente l'opzione `-l`, ma `@load` è utile da inserire all'interno di un file sorgente di **awk** che richieda l'accesso a un'estensione.

Capitolo 16 [Scrivere estensioni per gawk], pagina 395, descrive come scrivere estensioni (in C o C++) che possono essere caricate sia con `@load` che con l'opzione `-l`. È anche descritta l'estensione `ordchr`.

## 2.9 Opzioni e/o funzionalità obsolete

Questa sezione descrive funzionalità o opzioni da riga di comando provenienti da precedenti versioni di **gawk** che non sono più disponibili nella versione corrente, o che sono ancora utilizzabili ma sono deprecate (ciò significa che *non* saranno presenti nella prossima versione).

I file speciali relativi ai processi `/dev/pid`, `/dev/ppid`, `/dev/pgrpid` e `/dev/user` erano deprecati, ma ancora disponibili, in **gawk** 3.1. A partire dalla versione 4.0, non sono più interpretati da **gawk** in modo speciale (al loro posto usare invece `PROCINFO`; si veda Sezione 7.5.2 [Variabili predefinite con cui awk fornisce informazioni], pagina 165).

## 2.10 Opzioni e funzionalità non documentate

*Usa il codice sorgente, Luke!*  
—Obi-Wan

Questa sezione è stata lasciata intenzionalmente vuota.

## 2.11 Sommario

- Per eseguire **awk** usare, o `'awk 'programma' file'` o `'awk -f file-del-programma file'`.
- Le tre opzioni standard per tutte le versioni di **awk** sono `-f`, `-F` e `-v`. **gawk** fornisce queste e molte altre, come pure le opzioni estese corrispondenti scritte in stile GNU.
- Gli argomenti da riga di comando che non sono opzioni sono trattati normalmente come nomi-file, a meno che non abbiano la forma `'var=valore'`; nel qual caso vengono

riconosciuti come assegnamenti di variabile da eseguire in quel punto nell'elaborazione dell'input.

- Tutti gli argomenti da riga di comando che non sono opzioni, escluso il testo del programma, vengono messe nel vettore `ARGV`. Modifiche a `ARGC` e `ARGV` influiscono su come `awk` elabora l'input.
- Si può usare un segno meno a sé stante ('-') per designare lo standard input sulla riga di comando. `gawk` consente anche di usare il nome-file speciale `/dev/stdin`.
- `gawk` tiene conto di diverse variabili d'ambiente; `AWKPATH`, `AWKLIBPATH` e `POSIXLY_CORRECT` sono le più importanti.
- Lo stato d'uscita di `gawk` invia informazioni al programma che lo ha invocato. Usare l'istruzione `exit` dall'interno di un programma `awk` per impostare il codice di ritorno.
- `gawk` consente di includere nel proprio programma file sorgenti di `awk` con la direttiva `@include` o con le opzioni da riga di comando `-i` e `-f`.
- `gawk` consente di caricare funzioni aggiuntive scritte in C o C++ con la direttiva `@load` e/o con l'opzione `-l` (questa funzionalità avanzata è descritta più avanti, in [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395).

## 3 Espressioni regolari

Una *espressione regolare*, o *regex*, è un modo per descrivere un insieme di stringhe. Poiché le espressioni regolari sono una parte fondamentale della programmazione in **awk**, il loro formato e il loro uso meritano un capitolo a sé stante.

Un'espressione regolare racchiusa tra barre ('/') è un modello di ricerca **awk** che individua tutti i record in input il cui testo corrisponde al modello stesso. L'espressione regolare più semplice è una sequenza di lettere o di numeri, o di entrambi. Una tale *regex* individua ogni stringa che contenga quella particolare sequenza. Quindi, la *regex* 'pippo' individua ogni stringa che contenga 'pippo'. In altre parole, al modello di ricerca /pippo/ corrisponde ogni record in input che contiene i cinque caratteri consecutivi 'pippo' *in qualsiasi parte* del record. Altri tipi di *regex* permettono di specificare classi di stringhe molto più complesse.

All'inizio, gli esempi in questo capitolo sono semplici. Man mano che entriamo nei dettagli su come funzionano le espressioni regolari utilizzeremo formulazioni più complesse.

### 3.1 Uso di espressioni regolari

Un'espressione regolare può essere usata come modello di ricerca racchiudendola tra barre. L'espressione regolare è quindi confrontata con tutto il testo di ogni record (normalmente, basta che corrisponda a una parte qualsiasi del testo per risultare soddisfatta). Per esempio, il seguente programma stampa il secondo campo di ogni record in cui compaia la stringa 'li', in qualsiasi parte del record:

```
$ awk '/li/ { print $2 }' mail-list
+ 555-5553
+ 555-0542
+ 555-6699
+ 555-3430
```

Espressioni regolari possono anche essere usate in espressioni di confronto. Queste espressioni consentono di specificare le stringhe da riconoscere; non devono necessariamente comprendere l'intero record corrente. I due operatori '~' e '!~' confrontano espressioni regolari. Le espressioni che usano questi operatori possono essere usate come modelli di ricerca, o nelle istruzioni **if**, **while**, **for**, e **do**. (Si veda la [Sezione 7.4 \[Istruzioni di controllo nelle azioni\]](#), [pagina 153](#).) Per esempio:

```
exp ~ /regex/
```

è verificata se l'espressione *exp* (intesa come stringa) corrisponde a *regex*. L'esempio che segue individua, o sceglie, tutti i record in input in cui la lettera maiuscola 'J' è presente da qualche parte nel primo campo:

```
$ awk '$1 ~ /J/' inventory-shipped
+ Jan 13 25 15 115
+ Jun 31 42 75 492
+ Jul 24 34 67 436
+ Jan 21 36 64 620
```

Lo stesso risultato si può ottenere anche così:

```
awk '{ if ($1 ~ /J/) print }' inventory-shipped
```

Il prossimo esempio chiede che l'espressione *exp* (intesa come stringa) *NON* corrisponda a *regex*:

```
exp !~ /regex/
```

L'esempio che segue individua o sceglie tutti i record in input il cui primo campo *NON* contiene la lettera maiuscola 'J':

```
$ awk '$1 !~ /J/' inventory-shipped
+ Feb 15 32 24 226
+ Mar 15 24 34 228
+ Apr 31 52 63 420
+ May 16 34 29 208
...
```

Quando una *regex* è racchiusa tra barre, come */pippo/*, la chiamiamo una *costante regex*, proprio come 5.27 è una costante numerica e "pippo" è una costante [di tipo] stringa.

## 3.2 Sequenze di protezione

Alcuni caratteri non possono essere inclusi letteralmente in costanti stringa ("pippo") o in costanti *regex* (/pippo/). Vanno invece rappresentati usando *sequenze di protezione*, ossia sequenze di caratteri preceduti da una barra inversa ('\''). Una sequenza di protezione può essere usata per includere un carattere di "doppio apice" in una costante stringa. Poiché un semplice doppio apice termina la stringa, va usato '\\" per richiedere che un doppio apice sia presente all'interno di una stringa. Per esempio:

```
$ awk 'BEGIN { print "Egli le disse \"ciao!\".\" }'
+ Egli le disse "ciao!".
```

Lo stesso carattere di barra inversa è un altro carattere che non può essere incluso normalmente; occorre scrivere '\\ per inserire una barra inversa nella stringa o *regex*. Quindi, la stringa costituita dai due caratteri '\" e \' deve essere scritta come "\\\"\\\".

Altre sequenze di protezione rappresentano caratteri non stampabili come TAB o il ritorno a capo. Anche se è possibile immettere la maggior parte dei caratteri non stampabili direttamente in una costante stringa o *regex*, essi possono non essere di facile comprensione.

La seguente lista elenca tutte le sequenze di protezione usate in *awk* e cosa rappresentano. Se non è detto altrimenti, tutte queste sequenze di protezione valgono sia per costanti stringa che per costanti *regex*:

\\	Barra inversa letterale, '\\.
\a	Il carattere "campanello", <i>Ctrl-g</i> , codice ASCII 7 (BEL). (Spesso genera qualche tipo di segnale sonoro udibile.)
\b	Barra inversa, <i>Ctrl-h</i> , codice ASCII 8 (BS).
\f	Nuova pagina, <i>Ctrl-l</i> , codice ASCII 12 (FF).
\n	A capo, <i>Ctrl-j</i> , codice ASCII 10 (LF).
\r	Ritorno del carrello, <i>Ctrl-m</i> , codice ASCII 13 (CR).
\t	Tabulazione orizzontale, <i>Ctrl-i</i> , codice ASCII 9 (HT).

- `\v` Tabulazione verticale, *Ctrl-k*, codice ASCII 11 (VT).
- `\nnn` Il valore ottale *nnn*, dove *nnn* può essere da 1 a 3 cifre ottali, tra ‘0’ e ‘7’. Per esempio, il codice per il carattere ASCII ESC (escape) è ‘\033’.
- `\xhh...` Il valore esadecimale *hh*, dove *hh* indica una sequenza di cifre esadecimali (‘0’–‘9’, e ‘A’–‘F’ o ‘a’–‘f’). Dopo ‘\x’ è consentito un massimo di due cifre. Ogni ulteriore cifra esadecimale è considerata come una semplice lettera o numero. (e.c.) (La sequenza di protezione ‘\x’ non è permessa in POSIX awk.)

**ATTENZIONE:** In ISO C, la sequenza di protezione continua fino a raggiungere il primo carattere che non sia una cifra esadecimale. In passato, **gawk** avrebbe continuato ad aggiungere cifre esadecimali al valore finché non trovava una cifra non esadecimale oppure fino a raggiungere la fine della stringa. Comunque usare più di due cifre esadecimali produceva risultati indefiniti. Dalla versione 4.2, vengono elaborate solo due cifre.

- `\/` Una barra (necessario solo per costanti *regex*). Questa sequenza si usa per inserire una costante *regex* che contiene una barra (come `/.*/\home\/[[:alnum:]]+.*;/`; la notazione ‘[[:alnum:]]’ verrà spiegata più avanti, nella [Sezione 3.4 \[Usare espressioni tra parentesi quadre\]](#), [pagina 55](#)). Poiché una *regex* è racchiusa tra barre, si deve proteggere ogni barra che sia parte dell’espressione, per dire ad **awk** di andare avanti a scandire il resto della *regex*.
- `\"` Un doppio apice (necessario solo per costanti stringa). Questa sequenza si usa per inserire in una costante stringa il carattere doppio apice (come `"Egli le disse \"ciao!\"."`). Poiché la stringa è racchiusa tra doppi apici, si deve proteggere ogni doppio apice che sia parte della stringa per dire ad **awk** di andare avanti a elaborare il resto della stringa.

In **gawk**, parecchie altre sequenze di due caratteri inizianti con con una barra inversa hanno un significato speciale nelle *regex*. [Sezione 3.7 \[Operatori \*regex\* propri di \*\*gawk\*\*\]](#), [pagina 59](#).

In una *regex*, una barra inversa che preceda un carattere non presente nella lista precedente, e non elencato in [Sezione 3.7 \[Operatori \*regex\* propri di \*\*gawk\*\*\]](#), [pagina 59](#), significa che il carattere seguente dovrebbe essere preso letteralmente, anche se normalmente sarebbe un operatore di *regex*. Per esempio, `/a\+b/` individua i tre caratteri ‘a+b’.

Per una completa portabilità, non usare una barra inversa prima di qualsiasi carattere non incluso nella lista precedente, o che non sia un operatore.

**Barra inversa prima di un carattere normale**

Se si mette una barra inversa in una costante stringa prima di qualcosa che non sia uno dei caratteri elencati sopra, POSIX `awk` di proposito lascia indefinito il comportamento. Ci sono due possibilità:

Togliere la barra inversa

Questo è quel che sia BWK `awk` che `gawk` fanno. Per esempio, `"a\qc"` equivale a `"aqc"`. (Poiché questo è un errore che può capitare o non capitare con la stessa probabilità, `gawk` lo segnala). Volendo usare come separatore di campo `'FS = "[\t]+\|[\t]+"'` ossia delle barre verticali precedute e seguite da almeno uno spazio, occorre mettere due barre inverse nella stringa: `'FS = "[\t]+\|[\t]+"'`.)

Tenere la barra inversa così com'è.

Alcune altre implementazioni di `awk` fanno questo. In quelle implementazioni, immettere `"a\qc"` equivale a immettere `"a\\qc"`.

Ricapitolando:

- Le sequenze di protezione nella lista di cui sopra sono sempre elaborate per prime, sia per le costanti stringa che per le costanti *regex*. Questo viene fatto quasi subito, non appena `awk` legge il programma.
- `gawk` elabora sia costanti *regex* che *regex* dinamiche (si veda la [Sezione 3.6 \[Usare \*regex\* dinamiche\]](#), pagina 57), per gli operatori speciali elencati in [Sezione 3.7 \[Operatori \*regex\* propri di `gawk`\]](#), pagina 59.
- Una barra inversa prima di ogni altro carattere richiede di trattare quel carattere letteralmente.

**Sequenze di protezione per metacaratteri**

Supponiamo che si usi una protezione ottale o esadecimale per rappresentare un metacarattere di *regex* (si veda [Sezione 3.3 \[Operatori di espressioni regolari\]](#), pagina 52). `awk` considera il carattere come un carattere letterale o come un operatore di *regex*?

Storicamente, tali caratteri erano considerati letteralmente. Invece, lo standard POSIX richiede che siano considerati come metacaratteri veri e propri, e questo è ciò che `gawk` fa. In modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), `gawk` tratta i caratteri scritti come sequenze ottali ed esadecimali letteralmente, quando sono usati in costanti *regex*. Quindi, `/a\52b/` è equivalente a `/a\*b/`.



### 3.3 Operatori di espressioni regolari

È possibile inserire in espressioni regolari dei caratteri speciali, detti *operatori di espressioni regolari* o *metacaratteri*, per aumentarne il potere e la versatilità.

Le sequenze di protezione descritte prima in [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50, sono valide all'interno di una *regex*. Sono precedute da una `'\'` e sono riconosciute e convertite nei caratteri reali corrispondenti nella primissima fase dell'elaborazione delle *regex*.

Ecco una lista dei metacaratteri. Tutti i caratteri che non sono sequenze di protezione e che non sono elencati qui rappresentano se stessi:

- `\` Si usa per togliere il significato speciale a un carattere quando si effettuano confronti. Per esempio, `\$` individua il carattere `$`.
- `^` Si usa per indicare l'inizio di una stringa. Per esempio, `^@chapter` individua `@chapter` all'inizio di una stringa e si può usare per identificare inizi di capitoli in file sorgenti Texinfo. Il simbolo `^` è conosciuto come *àncora*, perché *àncora* la ricerca solo all'inizio della stringa.
- È importante notare che `^` non individua un inizio di riga (il punto subito dopo un ritorno a capo `\n`) che si trovi all'interno di una stringa. La condizione non è verificata nell'esempio seguente:
- ```
if ("riga1\nRIGA 2" ~ /^R/) ...
```
- `$` Simile a `^`, ma serve a indicare la fine di una stringa. Per esempio, `p$` individua un record che termina con la lettera `p`. Il `$` è un'ancora e non individua una fine di riga (il punto immediatamente prima di un carattere di ritorno a capo `\n`) contenuta in una stringa. La condizione nell'esempio seguente non è verificata:
- ```
if ("riga1\nRIGA 2" ~ /1$/) ...
```
- `.` (punto) Individua un qualsiasi carattere, *incluso* il carattere di ritorno a capo. Per esempio, `.P` individua ogni carattere in una stringa che sia seguito da una `P`. Usando la concatenazione, si può formare un'espressione regolare come `U.A`, che individua qualsiasi sequenza di tre caratteri che inizia con `U` e finisce con `A`.
- In modalità POSIX stretta (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), `.` non individua il carattere NUL, ossia il carattere con tutti i bit uguali a zero. In altri contesti, NUL è solo un carattere qualsiasi. Altre versioni di `awk` possono non essere in grado di individuare il carattere NUL.
- `[...]` Questa è chiamata una *espressione tra parentesi quadre*.<sup>1</sup> Individua *uno* qualsiasi dei caratteri racchiusi tra parentesi quadre. Per esempio, `[MVX]` individua uno qualsiasi dei caratteri `M`, `V`, o `X` in una stringa. Una spiegazione esauriente di quel che si può mettere all'interno di un'espressione tra parentesi quadre è data in [Sezione 3.4 \[Usare espressioni tra parentesi quadre\]](#), [pagina 55](#).
- `[^...]` Questa è una *espressione tra parentesi quadre complementata*. Il primo carattere dopo la `[` deve essere un `^`. Individua qualsiasi carattere *tranne* quelli tra parentesi quadre. Per esempio, `[^awk]` individua qualsiasi carattere che non sia una `a`, `w`, o `k`.
- `|` Questo è un *operatore alternativa* ed è usato per specificare delle alternative. La `|` ha la precedenza più bassa tra tutti gli operatori di espressioni regolari. Per esempio, `^P|[aeiouy]` individua tutte le stringhe corrispondenti a `^P` oppure a `[aeiouy]`. Ciò significa che individua qualsiasi stringa che inizi con `P` o contenga (in qualsiasi posizione al suo interno) una vocale inglese minuscola. L'alternativa si applica alle *regexp* più ampie individuabili in ogni lato.

<sup>1</sup> In altri testi, un'espressione tra parentesi quadre potrebbe essere definita come *insieme di caratteri*, *classe di caratteri* o *lista di caratteri*.

(...)

Le parentesi sono usate per raggruppare, sia nelle espressioni regolari sia in quelle aritmetiche. Si possono usare per concatenare espressioni regolari che contengono l'operatore alternativa, '|'. Per esempio, '@(samp|code)\{[~}]+\}' individua sia '@code{pippo}' sia '@samp{pluto}'. (Queste sono sequenze in linguaggio Texinfo per controllare la formattazione. Il significato di '+' è spiegato più avanti in questa lista.)

\*

Questo simbolo richiede che la precedente espressione regolare sia ripetuta tante volte quanto serve per trovare una corrispondenza. Per esempio, 'ph\*' applica il simbolo '\*' al carattere 'h' che lo precede immediatamente e ricerca corrispondenze costituite da una 'p' seguita da un numero qualsiasi di 'h'. Viene individuata anche solo la 'p', se non ci sono 'h'.

Ci sono due sfumature da capire sul funzionamento di '\*'. Primo, '\*' tiene conto solo del singolo componente dell'espressione regolare che lo precede (p.es., in 'ph\*' vale solo per 'h'). Per fare sì che '\*' si applichi a una sottoespressione più estesa, occorre metterla tra parentesi: '(ph)\*' individua 'ph', 'phph', 'phphph' e così via.

Secondo, '\*' trova quante più ripetizioni siano possibili. Se il testo da ricercare è 'phhhhhhhhhhhhhhoey', 'ph\*' individua tutte le 'h'.

+

Questo simbolo è simile a '\*', tranne per il fatto che l'espressione precedente deve essere trovata almeno una volta. Questo significa che 'wh+y' individuerrebbe 'why' e 'whhy', ma non 'wy', mentre 'wh\*y' li troverebbe tutti e tre.

?

Questo simbolo è simile a '\*', tranne per il fatto che l'espressione che precede può essere trovata una volta sola oppure non trovata affatto. Per esempio, 'fe?d' individua 'fed' e 'fd', ma nient'altro.

{n}

{n,}

{n,m}

Uno o due numeri tra parentesi graffe rappresentano una *espressione di intervallo*. Se c'è un numero tra graffe, la *regex* precedente è ripetuta *n* volte. Se ci sono due numeri separati da una virgola, la *regex* precedente è ripetuta da *n* a *m* volte. Se c'è un numero seguito da una virgola, allora la *regex* precedente è ripetuta almeno *n* volte:

wh{3}y     Riconosce 'whhhy', ma non 'why' o 'whhhhy'.

wh{3,5}y   Riconosce soltanto 'whhhy', 'whhhhy', o 'whhhhhhy'.

wh{2,}y     Riconosce 'whhy', 'whhhy' e così via.

Le espressioni di intervallo non erano tradizionalmente disponibili in **awk**. Sono state aggiunte come parte dello standard POSIX per rendere **awk** ed **egrep** coerenti tra di loro.

In passato, poiché vecchi programmi possono usare '{' e '}' in costanti *regex*, **gawk** non riconosceva espressioni di intervallo nelle *regex*.

Comunque, a partire dalla versione 4.0, **gawk** riconosce espressioni di intervallo per default. Ciò accade perché la compatibilità con POSIX è ritenuta più importante da molti utenti **gawk** rispetto alla compatibilità con dei vecchi programmi.

Per programmi che usano ‘{’ e ‘}’ in costanti *regexp*, è buona pratica proteggerli sempre con una barra inversa. Allora le costanti *regexp* sono valide e si comportano come desiderato, usando qualsiasi versione di **awk**.<sup>2</sup>

Infine, quando ‘{’ e ‘}’ appaiono in costanti *regexp* in un modo non interpretabile come espressione di intervallo (come in `/q{a}/`), allora sono prese letteralmente.

Nelle espressioni regolari, gli operatori ‘\*’, ‘+’, e ‘?’, come pure le parentesi graffe ‘{’ e ‘}’, hanno la precedenza più alta, seguiti dalla concatenazione, e infine da ‘|’. Come nell’algebra, le parentesi possono cambiare il raggruppamento degli operatori. In POSIX **awk** e in **gawk**, gli operatori ‘\*’, ‘+’, e ‘?’ rappresentano se stessi quando non c’è nulla nella *regexp* che li precede. Per esempio, `/+ /` individua un semplice segno più. Comunque, molte altre versioni di **awk** trattano una simile notazione come un errore di sintassi.

Se **gawk** è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), le espressioni di intervallo non si possono usare nelle espressioni regolari.

### 3.4 Usare espressioni tra parentesi quadre

Come detto sopra, un’espressione tra parentesi quadre individua qualsiasi carattere incluso tra le parentesi quadre aperta e chiusa.

All’interno di un’espressione tra parentesi quadre, una *espressione di intervallo* è formata da due caratteri separati da un trattino. Individua ogni singolo carattere compreso tra i due caratteri, ordinati secondo l’insieme di caratteri in uso nel sistema. Per esempio, ‘`[0-9]`’ è equivalente a ‘`[0123456789]`’. (Si veda [Sezione A.8 \[Intervalli \*regexp\* e localizzazione: una lunga e triste storia\]](#), [pagina 474](#), per una spiegazione di come lo standard POSIX e **gawk** sono cambiati nel corso degli anni. La cosa ha un interesse principalmente storico.)

Con la crescente popolarità dello [standard di caratteri Unicode](#), c’è un’ulteriore dettaglio da tenere in conto. Le sequenze di protezione ottali ed esadecimali utilizzabili per inserire valori all’interno di espressioni tra parentesi quadre sono considerate contenere solo caratteri che occupano un unico byte (caratteri il cui valore stia nell’intervallo 0–256). Per individuare un intervallo di caratteri in cui i punti di inizio e fine dell’intervallo abbiano valori maggiori di 256, occorre immettere direttamente le codifiche multi-byte dei caratteri in questione.

Per includere uno dei caratteri ‘\’, ‘]’, ‘-’, o ‘^’ in un’espressione tra parentesi quadre, occorre inserire un ‘\’ prima del carattere stesso. Per esempio:

```
[d\]]
```

individua sia ‘d’ che ‘]’. Inoltre, se si mette una ‘]’ subito dopo la ‘[’ aperta, la parentesi quadra chiusa è considerata come uno dei caratteri da individuare.

L’utilizzo di ‘\’ nelle espressioni tra parentesi quadre è compatibile con altre implementazioni di **awk** ed è anche richiesto da POSIX. Le espressioni regolari in **awk** sono un insieme più esteso delle specificazioni POSIX per le espressioni regolari estese (ERE). Le ERE POSIX sono basate sulle espressioni regolari accettate dal tradizionale programma di utilità **egrep**.

<sup>2</sup> È meglio usare due barre inverse se si sta usando una costante stringa con un operatore *regexp* o una funzione.

Le *classi di caratteri* sono una funzionalità introdotta nello standard POSIX. Una classe di caratteri è una particolare notazione per descrivere liste di caratteri che hanno un attributo specifico, ma i caratteri veri e propri possono variare da paese a paese e/o da insieme di caratteri a insieme di caratteri. Per esempio, la nozione di cosa sia un carattere alfabetico è diversa tra gli Stati Uniti e la Francia.

Una classe di caratteri è valida solo in una *regex contenuta* tra le parentesi quadre di un'espressione tra parentesi quadre. Le classi di caratteri consistono di '[:', una parola chiave che segnala la classe, e ':]'. La [Tabella 3.1](#) elenca le classi di caratteri definite dallo standard POSIX.

Classe	Significato
<code>[:alnum:]</code>	Caratteri alfanumerici.
<code>[:alpha:]</code>	Caratteri alfabetici.
<code>[:blank:]</code>	Caratteri spazio e TAB.
<code>[:cntrl:]</code>	Caratteri di controllo.
<code>[:digit:]</code>	Caratteri numerici.
<code>[:graph:]</code>	Caratteri che sono stampabili e visibili. (Uno <i>spazio</i> è stampabile ma non visibile, mentre una <i>'a'</i> è l'uno e l'altro.)
<code>[:lower:]</code>	Caratteri alfabetici minuscoli.
<code>[:print:]</code>	Caratteri stampabili (caratteri che non sono caratteri di controllo).
<code>[:punct:]</code>	Caratteri di punteggiatura (caratteri che non sono lettere, cifre, caratteri di controllo, o caratteri di spazio).
<code>[:space:]</code>	Caratteri di spazio (come <i>spazio</i> , TAB, e <i>formfeed</i> , per citarne alcuni).
<code>[:upper:]</code>	Caratteri alfabetici maiuscoli.
<code>[:xdigit:]</code>	Caratteri che sono cifre esadecimali.

Tabella 3.1: classi di caratteri POSIX

Per esempio, prima dello standard POSIX, si doveva scrivere `/[A-Za-z0-9]/` per individuare i caratteri alfanumerici. Se l'insieme di caratteri in uso comprendeva altri caratteri alfabetici, l'espressione non li avrebbe individuati. Con le classi di caratteri POSIX si può scrivere `/[[:alnum:]]/` per designare i caratteri alfabetici e numerici dell'insieme di caratteri in uso.

Alcuni programmi di utilità che cercano espressioni regolari prevedono una classe di caratteri, non standard, `[:ascii:]`; `awk` non la prevede. Tuttavia, è possibile ottenere lo stesso risultato utilizzando `[\x00-\x7F]`. Quest'espressione individua tutti i valori numerici tra zero e 127, che è l'intervallo definito dell'insieme di caratteri ASCII. Usando una lista di caratteri che esclude (`[\x00-\x7F]`) si individuano tutti i caratteri mono-byte che non sono nell'intervallo ASCII.

In espressioni tra parentesi quadre possono apparire due ulteriori sequenze speciali. Riguardano insiemi di caratteri non-ASCII, che possono avere simboli singoli (chiamati *elementi di collazione*) che sono rappresentati con più di un carattere. Possono designare anche parecchi caratteri che sono equivalenti tra loro ai fini della *collazione*, o dell'ordinamento. (Per esempio, in francese, la semplice "e" e la sua versione con accento grave "è" sono equivalenti). Queste sequenze sono:

elementi di collazione

Elementi di collazione multi-byte racchiusi fra ‘[.]’ e ‘[.]’.

Per esempio, se ‘ch’ è un elemento di collazione, ‘[.ch.]’ è una *regex* che individua questo elemento di collazione, mentre ‘[ch]’ è una *regex* che individua le lettere ‘c’ o ‘h’.

classi di equivalenza

Sono nomi, specifici a una particolare localizzazione, per una lista di caratteri equivalenti tra loro. Il nome è racchiuso fra ‘[=’ e ‘=]’.

Per esempio, il nome ‘e’ potrebbe essere usato per designare “e”, “ê”, “è”, e “é”. In questo caso, ‘[=e=]’ è una *regex* che corrisponde a ‘e’, ‘ê’, ‘è’ e ‘é’.

Queste funzionalità sono molto utili in localizzazioni non inglesi.

**ATTENZIONE:** Le funzioni di libreria che **gawk** usa per individuare le espressioni regolari per ora riconoscono solo le classi di caratteri POSIX; non riconoscono simboli di collazione o classi di equivalenza.

In un’espressione tra parentesi quadre, una parentesi aperta (‘[’) che non costituisca l’inizio della specificazione di una classe di caratteri, di simboli di collazione o di una classe di equivalenza è interpretata letteralmente. Questo vale anche per ‘.’ e ‘\*’.

### 3.5 Quanto è lungo il testo individuato?

Si consideri il caso seguente:

```
echo aaaabcd | awk '{ sub(/a+/, "<A>"); print }'
```

Questo esempio usa la funzione `sub()` per modificare il record in input. (`sub()` sostituisce la prima ricorrenza in ogni testo individuato dal primo argomento con la stringa fornita come secondo argomento; si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198.](#)) Qui, la *regex* `/a+/,` richiede “uno o più caratteri ‘a’,” e il testo da sostituire è ‘<A>’.

L’input contiene quattro caratteri ‘a’. Le espressioni regolari **awk** (e POSIX) individuano sempre la sequenza *più lunga*, partendo da sinistra, di caratteri in input che corrispondono. Quindi, tutti e quattro i caratteri ‘a’ sono rimpiazzati con ‘<A>’ in questo esempio:

```
$ echo aaaabcd | awk '{ sub(/a+/, "<A>"); print }'
-> <A>bcd
```

Per semplici test corrisponde/non corrisponde, la cosa ha poca importanza. Ma se si sta controllando un testo o si fanno sostituzioni usando le funzioni `match()`, `sub()`, `gsub()` e `gensub()`, è invece molto importante. Tenere in conto questo principio è importante anche quando si suddividono record e campi usando delle *regex* (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\], pagina 63,](#) e anche si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\], pagina 71.](#))

### 3.6 Usare *regex* dinamiche

L’espressione a destra di un operatore ‘~’ o ‘!~’ non deve necessariamente essere una costante *regex* (cioè, una stringa di caratteri tra barre). Può essere una qualsiasi espressione. L’espressione è valutata e convertita in una stringa se necessario; il contenuto della stringa

è poi usato come una *regexp*. Una *regexp* calcolata in questo modo è detta una *regexp dinamica* o una *regexp calcolata*:

```
BEGIN { regexp_numerica = "[[:digit:]]+" }
$0 ~ regexp_numerica    { print }
```

Questo *script* imposta `regexp_numerica` come una *regexp* che descrive una o più cifre, e poi controlla se un record in input corrisponde a questa *regexp*.

**NOTA:** Usando gli operatori ‘~’ e ‘!~’, si tenga presente che c’è una differenza tra una costante *regexp* racchiusa tra barre e una costante stringa racchiusa tra doppi apici. Se si intende utilizzare una costante stringa, occorre comprendere che la stringa è, in sostanza, scandita *due volte*: la prima volta quando **awk** legge il programma, e la seconda volta quando va a confrontare la stringa a sinistra dell’operatore con il modello che sta alla sua destra. Questo vale per ogni espressione (come la `regexp_numerica`, vista nel precedente esempio), non solo per le costanti stringa.

Che differenza fa la doppia scansione di una stringa? La risposta ha a che vedere con le sequenze di protezione e particolarmente con le barre inverse. Per inserire una barra inversa in un’espressione regolare all’interno di una stringa, occorre inserire *due* barre inverse.

Per esempio, `/\*/` è una costante *regexp* per designare un ‘\*’ letterale. È richiesta una sola barra inversa. Per fare lo stesso con una stringa, occorre immettere `"\\*"`. La prima barra inversa protegge la seconda in modo che la stringa in realtà contenga i due caratteri ‘\’ e ‘\*’.

Dato che si possono usare sia costanti *regexp* che costanti stringa per descrivere espressioni regolari, qual è da preferire? La risposta è “costanti *regexp*”, per molti motivi:

- Le costanti stringa sono più complicate da scrivere e più difficili da leggere. Usare costanti *regexp* rende i programmi meno inclini all’errore. Non comprendere la differenza tra i due tipi di costanti è una fonte frequente di errori.
- È più efficiente usare costanti *regexp*. **awk** può accorgersi che è stata fornita una *regexp* e memorizzarla internamente in una forma che rende la ricerca di corrispondenze più efficiente. Se si usa una costante stringa, **awk** deve prima convertire la stringa nel suo formato interno e quindi eseguire la ricerca di corrispondenze.
- Usare costanti *regexp* è la forma migliore; lascia comprendere chiaramente che si vuole una corrispondenza con una *regexp*.

**Usare \n in espressioni tra parentesi quadre in *regexp* dinamiche**

Alcune delle prime versioni di **awk** non consentono di usare il carattere di ritorno a capo all'interno di un'espressione tra parentesi quadre in *regexp* dinamiche:

```
$ awk '$0 ~ "[ \t\n]"'
```

```
error awk: newline in character class [
error ]...
error source line number 1
error context is
error $0 ~ "[ >>> \t\n]" <<<
```

Ma un ritorno a capo in una costante *regexp* non dà alcun problema:

```
$ awk '$0 ~ /[ \t\n]/'
```

ecco una riga di esempio

⊢ ecco una riga di esempio

Ctrl-d

**gawk** non ha questo problema, e non dovrebbe accadere spesso in pratica, ma val la pena di notarlo a futura memoria.

**3.7 Operatori *regexp* propri di **gawk****

Il software GNU che ha a che fare con espressioni regolari comprende alcuni operatori *regexp* aggiuntivi. Questi operatori sono descritti in questa sezione e sono specificamente per **gawk**; non sono disponibili in altre implementazioni di **awk**. La maggior parte degli operatori aggiuntivi riguarda l'identificazione di parole. Ai nostri fini, una *parola* è una sequenza di uno o più lettere, cifre, o trattini bassi ('\_'):

- \s** Corrisponde a ogni carattere bianco. Lo si può pensare come un'abbreviazione di '[[:space:]]'.
- \S** Corrisponde a ogni carattere che non è uno spazio bianco. Lo si può pensare come un'abbreviazione di '[^[:space:]]'.
- \w** Corrisponde a ogni carattere che componga una parola; ovvero, corrisponde a ogni lettera, cifra, o trattino basso. Lo si può pensare come un'abbreviazione di '[[:alnum:]]\_]'.
- \W** Corrisponde a ogni carattere che non è parte di una parola. Lo si può pensare come un'abbreviazione di '[^[:alnum:]]\_]'.
- \<** Individua la stringa nulla all'inizio di una parola. Per esempio, /\<via/ individua 'via' ma non 'funivia'.
- \>** Individua la stringa nulla alla fine di una parola. Per esempio, /via\>/ individua 'via' ma non 'viadotto'.
- \y** Individua la stringa nulla o alla fine o all'inizio di una parola. (cioè, il limite di una parola - *boundary* in inglese). Per esempio, '\yradar?\y' individua sia 'rada' che 'radar', come parole separate.
- \B** Individua la stringa nulla che ricorre all'interno di una parola. Per esempio, /\Bora\B/ individua 'Colorado', ma non individua 'che ora è'. '\B' è essenzialmente l'opposto di '\y'.

Ci sono due altri operatori che operano sui buffer. In Emacs un *buffer* è, naturalmente, un buffer di Emacs. In altri programmi GNU, fra cui **gawk**, le routine di libreria delle *regex* considerano come buffer l'intera stringa su cui effettuare il confronto. Gli operatori sono:

- \^ Individua la stringa nulla che occorre all'inizio di un buffer (di una stringa)
- \\$ Individua la stringa nulla che occorre alla fine di un buffer (di una stringa)

Poiché '^' e '\$' si riferiscono sempre all'inizio e alla fine di stringhe, questi operatori non aggiungono nuove funzionalità ad **awk**. Sono inclusi per compatibilità con altro software GNU.

In altro software GNU, l'operatore di limite-di-parola è '\b'. Questo, comunque, è in conflitto con la definizione, nel linguaggio **awk**, di '\b' come backspace, quindi **gawk** usa una lettera differente. Un metodo alternativo sarebbe stato di richiedere due barre inverse negli operatori GNU, ma questo è stato ritenuto troppo arzigogolato. Il metodo corrente di usare '\y' al posto del '\b' di GNU sembra essere il male minore.

Le varie opzioni sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)) controllano come **gawk** interpreta i caratteri nelle *regex*:

Nessuna opzione

Per default, **gawk** fornisce tutte le funzionalità delle *regex* POSIX e gli operatori *regex* GNU precedentemente descritti.

**--posix** Sono ammesse solo le *regex* POSIX; gli operatori GNU non sono speciali (p.es., '\w' individua una semplice lettera 'w'). Le espressioni di intervallo sono ammesse.

**--traditional**

Le *regex* Unix tradizionali di **awk** sono ammesse. Gli operatori GNU non sono speciali, e le espressioni di intervallo non sono ammesse. Le classi di caratteri POSIX ('[:alnum:]', etc.) sono ammesse, poiché BWK **awk** le prevede. I caratteri descritti usando sequenze di protezione ottali ed esadecimali sono trattati letteralmente, anche se rappresentano metacaratteri di *regex*.

**--re-interval**

Sono consentite espressioni di intervallo in *regex*, se **--traditional** è stata specificata. Altrimenti, le espressioni di intervallo sono disponibili per default.

### 3.8 Fare confronti ignorando maiuscolo/minuscolo

Il tipo di carattere (maiuscolo/minuscolo) è normalmente rilevante nelle espressioni regolari, sia nella ricerca di caratteri normali (cioè, non metacaratteri), sia all'interno di espressioni fra parentesi. Quindi, una 'w' in un'espressione regolare individua solo una 'w' e non la corrispondente maiuscola 'W'.

Il modo più semplice per richiedere una ricerca non sensibile al maiuscolo/minuscolo è di usare un'espressione tra parentesi quadre, per esempio '[Ww]'. Comunque, questo può essere pesante se si usa spesso, e può rendere le espressioni regolari di difficile lettura. Ci sono due alternative che potrebbero essere preferibili.

Un modo per fare un confronto non sensibile a maiuscolo/minuscolo in un particolare punto del programma è di convertire i dati in un solo tipo (o minuscole o maiuscole),

usando le funzioni di stringa predefinite `tolower()` o `toupper()` (che non abbiamo ancora introdotto; si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Per esempio:

```
tolower($1) ~ /foo/ { ... }
```

converte il primo campo in minuscole, prima di fare un confronto. Questo funziona in ogni `awk` conforme allo standard POSIX.

Un altro metodo, proprio di `gawk`, è di impostare la variabile `IGNORECASE` a un valore diverso da zero (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162). Quando `IGNORECASE` è diverso da zero, *tutte* le operazioni con regexp e stringhe ignorano la distinzione maiuscolo/minuscolo.

Il cambio del valore di `IGNORECASE` controlla dinamicamente la sensibilità a maiuscolo/minuscolo del programma quando è in esecuzione. Il tipo di carattere (maiuscolo/minuscolo) è rilevante per default, poiché `IGNORECASE` (come la maggior parte delle variabili) è inizializzata a zero:

```
x = "aB"
if (x ~ /ab/) ...    # questo test non risulterà verificato

IGNORECASE = 1
if (x ~ /ab/) ...    # adesso sarà verificato
```

In generale, non è possibile usare `IGNORECASE` per rendere certe regole non sensibili a maiuscolo/minuscolo e altre regole invece sì, perché non c'è una maniera diretta per impostare `IGNORECASE` solo per l'espressione di una particolare regola.<sup>3</sup> Per fare questo, si usino espressioni tra parentesi quadre oppure `tolower()`. Comunque, una cosa che si può fare con `IGNORECASE` soltanto è di utilizzare o di ignorare la sensibilità a maiuscolo/minuscolo per tutte le regole contemporaneamente.

`IGNORECASE` è impostabile dalla riga di comando o in una regola `BEGIN` (si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\]](#), pagina 40; e si veda la [Sezione 7.1.4.1 \[Azioni di inizializzazione e pulizia\]](#), pagina 148). Impostare `IGNORECASE` dalla riga di comando è un modo per rendere un programma insensibile a maiuscolo/minuscolo senza doverlo modificare.

In localizzazioni multibyte, le equivalenze tra caratteri maiuscoli e minuscoli sono controllate usando i valori in formato esteso dell'insieme di caratteri della localizzazione. Per il resto, i caratteri sono controllati usando l'insieme di caratteri ISO-8859-1 (ISO Latin-1). Questo insieme di caratteri è un'estensione del tradizionale insieme con 128 caratteri ASCII, che include anche molti caratteri adatti per le lingue europee.<sup>4</sup>

Il valore di `IGNORECASE` non ha effetto se `gawk` è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). Il tipo di carattere (maiuscolo o minuscolo) è sempre rilevante in modalità compatibile.

<sup>3</sup> Programmatori esperti in C e C++ noteranno che questo è possibile, usando qualcosa come '`IGNORECASE = 1 && /fooBar/ { ... }`' e '`IGNORECASE = 0 || /foobar/ { ... }`'. Comunque, questo è un po' tortuoso e non è raccomandato.

<sup>4</sup> Se questo sembra oscuro, non c'è ragione di preoccuparsi; significa solo che `gawk` fa la cosa giusta.

### 3.9 Sommario

- Le espressioni regolari descrivono insiemi di stringhe da confrontare. In **awk**, le costanti *regexp* sono scritte racchiuse fra barre: `/.../`.
- Le costanti *regexp* possono essere usate da sole in modelli di ricerca e in espressioni condizionali, o come parte di espressioni di ricerca usando gli operatori `'~'` e `'!~'`.
- Le sequenze di protezione consentono di rappresentare caratteri non stampabili e consentono anche di rappresentare metacaratteri *regexp* come caratteri letterali per i quali cercare corrispondenze.
- Gli operatori *regexp* consentono raggruppamento, alternativa e ripetizione.
- Le espressioni tra parentesi quadre sono delle notazioni abbreviate per specificare insiemi di caratteri che possono avere corrispondenze in un punto particolare di una *regexp*. All'interno di espressioni tra parentesi quadre, le classi di caratteri POSIX consentono di specificare certi gruppi di caratteri in maniera indipendente dalla localizzazione.
- Le espressioni regolari individuano il testo più lungo possibile, a partire da sinistra nella stringa in esame. Questo ha importanza nei casi in cui serve conoscere la lunghezza della corrispondenza, come nella sostituzione di testo e quando il separatore di record sia una *regexp*.
- Espressioni di ricerca possono usare *regexp* dinamiche, ossia, i valori delle stringhe sono considerato come espressioni regolari.
- La variabile **gawk IGNORECASE** consente di controllare la differenza maiuscolo/minuscolo nel confronto mediante *regexp*. In altre versioni di **awk**, vanno usate invece le funzioni `tolower()` o `toupper()`.

## 4 Leggere file in input

Nel tipico programma `awk`, `awk` legge tutto l'input sia dallo standard input (per default è la tastiera, ma spesso è una *pipe* da un altro comando) o da file i cui nomi vengono specificati sulla riga di comando di `awk`. Se si specificano file in input, `awk` li legge nell'ordine, elaborando tutti i dati di uno prima di passare al successivo. Il nome del file in input corrente si trova nella variabile predefinita `FILENAME` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162).

L'input è letto in unità chiamate *record*, e viene elaborato, secondo le regole del programma, un record alla volta. Per default, ogni record è una riga. Ogni record è suddiviso automaticamente in "pezzi" chiamati *campi*. Questo rende più pratico far lavorare i programmi sulle parti di un record.

In rare occasioni, si potrebbe aver bisogno di usare il comando `getline`. Il comando `getline` è utile sia perché può procurare un input esplicito da un numero indeterminato di file, sia perché non vanno specificati sulla riga di comando di `awk` i nomi dei file usati con `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83).

### 4.1 Controllare come i dati sono suddivisi in record

`awk` suddivide l'input per il programma in record e campi. Tiene traccia del numero di record già letti dal file in input corrente. Questo valore è memorizzato in una variabile predefinita chiamata `FNR` che è reimpostata a zero ogni volta che si inizia un nuovo file. Un'altra variabile predefinita, `NR`, registra il numero totale di record in input già letti da tutti i file-dati. Il suo valore iniziale è zero ma non viene mai reimpostata a zero automaticamente.

#### 4.1.1 Come `awk` standard divide i record.

I record sono separati da un carattere chiamato *separatore di record*. Per default, il separatore di record è il carattere di ritorno a capo. Questo è il motivo per cui i record sono, per default, righe singole. Per usare un diverso carattere come separatore di record basta assegnare quel carattere alla variabile predefinita `RS`.

Come per ogni altra variabile, il valore di `RS` può essere cambiato nel programma `awk` con l'operatore di assegnamento, '=' (si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126). Il nuovo separatore di record dovrebbe essere racchiuso tra doppi apici, per indicare una costante di stringa. Spesso il momento giusto per far questo è all'inizio dell'esecuzione, prima che sia elaborato qualsiasi input, in modo che il primo record sia letto col separatore appropriato. Per far ciò, si usa il criterio speciale `BEGIN` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148). Per esempio:

```
awk 'BEGIN { RS = "u" }
     { print $0 }' mail-list
```

cambia il valore di `RS` in 'u', prima di leggere qualsiasi input. Il nuovo valore è una stringa il cui primo carattere è la lettera "u"; come risultato, i record sono separati dalla lettera "u". Poi viene letto il file in input, e la seconda riga nel programma `awk` (l'azione eseguita se non si specifica un criterio) stampa ogni record. Poiché ogni istruzione `print` aggiunge un ritorno a capo alla fine del suo output, questo programma `awk` copia l'input con ogni 'u' trasformato in un ritorno a capo. Qui vediamo il risultato dell'esecuzione del programma sul file `mail-list`:

```

$ awk 'BEGIN { RS = "u" }
>      { print $0 }' mail-list
+ Amelia      555-5553      amelia.zodiac
+ sq
+ e@gmail.com  F
+ Anthony     555-3412      anthony.assert
+ ro@hotmail.com A
+ Becky       555-7685      becky.algebrar
+ m@gmail.com  A
+ Bill        555-1675      bill.drowning@hotmail.com      A
+ Broderick   555-0542      broderick.aliq
+ otiens@yahoo.com R
+ Camilla     555-2912      camilla.inf
+ sar
+ m@skynet.be  R
+ Fabi
+ s           555-1234      fabi
+ s.
+ ndevesim
+ s@
+ cb.ed
+ F
+ J
+ lie         555-6699      j
+ lie.perscr
+ tabor@skeeve.com F
+ Martin      555-6480      martin.codicib
+ s@hotmail.com A
+ Sam
+ el          555-3430      sam
+ el.lanceolis@sh
+ .ed
+ A
+ Jean-Pa
+ l           555-2127      jeanpa
+ l.campanor
+ m@ny
+ .ed
+ R
+

```

Si noti che la voce relativa al nome ‘Bill’ non è divisa. Nel file-dati originale (si veda la [Sezione 1.2 \[File-dati per gli esempi\], pagina 23](#)), la riga appare in questo modo:

```

Bill      555-1675      bill.drowning@hotmail.com      A

```

Essa non contiene nessuna ‘u’, per cui non c’è alcun motivo di dividere il record, diversamente dalle altre, che hanno una o più ricorrenze della ‘u’. Infatti, questo record è trattato

come parte del record precedente; il ritorno a capo che li separa nell'output è l'originale ritorno a capo nel file-dati, non quella aggiunta da **awk** quando ha stampato il record!

Un altro modo per cambiare il separatore di record è sulla riga di comando, usando la funzionalità dell'assegnamento di variabile (si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\]](#), pagina 40):

```
awk '{ print $0 }' RS="u" mail-list
```

Questo imposta **RS** a 'u' prima di elaborare **mail-list**.

Usando un carattere alfabetico come 'u' come separatore di record è molto probabile che si ottengano risultati strani. Usando un carattere insolito come '/' è più probabile che si ottenga un comportamento corretto nella maggioranza dei casi, ma non c'è nessuna garanzia. La morale è: conosci i tuoi dati!

Quando si usano caratteri normali come separatore di record, c'è un caso insolito che capita quando **gawk** è reso completamente conforme a POSIX (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). In quel caso, la seguente (estrema) *pipeline* stampa un sorprendente '1':

```
$ echo | gawk --posix 'BEGIN { RS = "a" } ; { print NF }'
+ 1
```

C'è un solo campo, consistente in un ritorno a capo. Il valore della variabile predefinita **NF** è il numero di campi nel record corrente. (Normalmente **gawk** tratta il ritorno a capo come uno spazio vuoto, stampando '0' come risultato. Anche molte altre versioni di **awk** agiscono in questo modo.)

Il raggiungimento della fine di un file in input fa terminare il record di input corrente, anche se l'ultimo carattere nel file non è il carattere in **RS**.



La stringa nulla "" (una stringa che non contiene alcun carattere) ha un significato particolare come valore di **RS**. Significa che i record sono separati soltanto da una o più righe vuote. Si veda la [Sezione 4.8 \[Record su righe multiple\]](#), pagina 80, per maggiori dettagli.

Se si cambia il valore di **RS** nel mezzo di un'esecuzione di **awk**, il nuovo valore è usato per delimitare i record successivi, ma non riguarda il record in corso di elaborazione e neppure quelli già elaborati.

Dopo che è stata determinata la fine di un record, **gawk** imposta la variabile **RT** al testo nell'input che corrisponde a **RS**.

### 4.1.2 Divisione dei record con **gawk**

Quando si usa **gawk**, il valore di **RS** non è limitato a una stringa costituita da un solo carattere, ma può essere qualsiasi espressione regolare (si veda il [Capitolo 3 \[Espressioni regolari\]](#), pagina 49). (e.c.) In generale, ogni record termina alla stringa più vicina che corrisponde all'espressione regolare; il record successivo inizia alla fine della stringa che corrisponde. Questa regola generale è in realtà applicata anche nel caso normale, in cui **RS** contiene solo un ritorno a capo: un record termina all'inizio della prossima stringa che corrisponde (il prossimo ritorno a capo nell'input), e il record seguente inizia subito dopo la fine di questa stringa (al primo carattere della riga seguente). Il ritorno a capo, poiché corrisponde a **RS**, non appartiene a nessuno dei due record.

Quando **RS** è un singolo carattere, **RT** contiene lo stesso singolo carattere. Peraltro, quando **RS** è un'espressione regolare, **RT** contiene l'effettivo testo in input corrispondente all'espressione regolare.

Se il file in input termina senza che vi sia un testo che corrisponda a **RS**, **gawk** imposta **RT** alla stringa nulla.

Il seguente esempio illustra entrambe queste caratteristiche. In quest'esempio **RS** è impostato a un'espressione regolare che cerca sia un ritorno a capo che una serie di una o più lettere maiuscole con uno spazio vuoto opzionale iniziale e/o finale:

```
$ echo record 1 AAAA record 2 BBBB record 3 |
> gawk 'BEGIN { RS = "\n| ( *[:upper:]]+ *)" }
>          { print "Record =", $0,"e RT = [" RT "]" }'
+ Record = record 1 e RT = [ AAAA ]
+ Record = record 2 e RT = [ BBBB ]
+ Record = record 3 e RT = [
+ ]
```

Le parentesi quadre racchiudono il contenuto di **RT**, rendendo visibile lo spazio vuoto iniziale e quello finale. L'ultimo valore di **RT** è un ritorno a capo. Si veda la [Sezione 11.3.8 \[Un semplice editor di flusso\]](#), pagina 316, per un esempio più utile su **RS** come espressione regolare e su **RT**.

Se si imposta **RS** a un'espressione regolare che consente del testo finale opzionale, come '**RS** = "abc(XYZ)?"' è possibile, per via di limitazioni dell'implementazione, che **gawk** possa trovare la parte iniziale dell'espressione regolare, ma non la parte finale, in modo particolare se il testo di input che potrebbe avere una corrispondenza con la parte finale è piuttosto lungo. **gawk** cerca di evitare questo problema, ma al momento non ci sono garanzie che questo funzioni sempre.

**NOTA:** Si ricordi che in **awk**, i metacaratteri di ancoraggio '^' e '\$' trovano l'inizio e la fine di una *stringa*, e non l'inizio e la fine di una *riga*. Come risultato, qualcosa come '**RS** = "^[:upper:]]"' può solo corrispondere all'inizio di un file. Questo perché **gawk** vede il file in input come un'unica lunga stringa in cui possono essere presenti dei caratteri di ritorno a capo. È meglio perciò evitare metacaratteri di ancoraggio nel valore di **RS**.

L'uso di **RS** come espressione regolare e la variabile **RT** sono estensioni **gawk**; non sono disponibili in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). In modalità compatibile, solo il primo carattere del valore di **RS** determina la fine del record.

**RS = "\0" non è portabile**

Ci sono casi in cui capita di dover trattare un intero file-dati come un record unico. L'unico modo di far questo è quello di dare a **RS** un valore che non ricorre nel file in input. Ciò è difficile da fare in modo generale, così che un programma possa funzionare con file in input arbitrari.

Si potrebbe pensare che per i file di testo il carattere NUL, che consiste di un carattere con tutti i bit uguali a zero, sia un buon valore da usare per **RS** in questo caso:

```
BEGIN { RS = "\0" } # l'intero file diventa un record?
```

**gawk** di fatto lo accetta, e usa il carattere NUL come separatore di record. Questo funziona per certi file speciali, come `/proc/environ` su sistemi GNU/Linux, dove il carattere NUL è di fatto un separatore di record.. Comunque, quest'uso *non* è portabile sulla maggior parte delle implementazioni di **awk**.

Quasi tutte le altre implementazioni di **awk**<sup>1</sup> memorizzano internamente le stringhe come stringhe in stile C. Le stringhe in stile C usano il carattere NUL come terminatore di stringa. In effetti, questo significa che `RS = "\0"` è lo stesso di `RS = ""`.

Capita che recenti versioni di **mawk** possano usare il carattere NUL come separatore di record. Comunque questo è un caso particolare: **mawk** non consente di includere caratteri NUL nelle stringhe. (Ciò potrebbe cambiare in una versione futura di **mawk**.)

Si veda la [Sezione 10.2.8 \[Leggere un intero file in una sola volta\]](#), pagina 255, per un modo interessante di leggere file interi. Se si usa **gawk**, si veda [Sezione 16.7.10 \[Leggere un intero file in una stringa\]](#), pagina 454, per un'altra opzione.



## 4.2 Un'introduzione ai campi

Quando **awk** legge un record in input, il record è automaticamente *analizzato* o separato da **awk** in "pezzi" chiamati *campi*. Per default, i campi sono separati da *spazi vuoti*, come le parole in una riga stampata. Uno spazio vuoto in **awk** è qualsiasi stringa composta da uno o più spazi, segni di tabulazione o ritorni a capo; altri caratteri, come interruzione di pagina, tabulazione verticale, etc., che sono considerati spazi vuoti in altri linguaggi, *non* sono considerati tali da **awk**.

Lo scopo dei campi è quello di rendere più conveniente per l'utente far riferimento a questi frammenti dei record. Non è necessario usarli—si può operare sull'intero record, se si vuole—ma i campi sono ciò che rende così potenti dei semplici programmi **awk**.

Si usa il simbolo del dollaro ('\$') per far riferimento a un campo in un programma **awk**, seguito dal numero del campo desiderato. Quindi, **\$1** si riferisce al primo campo, **\$2** al secondo, e così via. (Diversamente che nelle shell Unix, i numeri di campo non sono limitati a una sola cifra; **\$127** è il centoventisettesimo campo nel record.) Per esempio, supponiamo che la seguente sia una riga in input:

**Questo pare essere un esempio proprio carino.**

Qui il primo campo, o **\$1**, è 'Questo', il secondo campo, o **\$2**, è 'pare', e via dicendo. Si noti che l'ultimo campo, **\$7**, è 'carino.'. Poiché non ci sono spazi tra la 'o' e il '.', il punto è considerato parte del settimo campo.

<sup>1</sup> Almeno quelle che ci sono note.

NF è una variabile predefinita il cui valore è il numero di campi nel record corrente. `awk` aggiorna automaticamente il valore di NF ogni volta che legge un record. Indipendentemente da quanti campi ci possano essere, l'ultimo campo in un record può essere rappresentato da `$NF`. Così, `$NF` è lo stesso di `$7`, che è 'carino.'. Se si cerca di far riferimento a un campo oltre l'ultimo (come `$8` quando il record ha solo sette campi), si ottiene la stringa nulla. (Se usato in un'operazione numerica si ottiene zero.)

L'uso di `$0`, che sarebbe come un riferimento al campo "numero zero", è un caso particolare: rappresenta l'intero record in input. Si usa quando non si è interessati a un campo specifico. Vediamo qualche altro esempio:

```
$ awk '$1 ~ /li/ { print $0 }' mail-list
+ Amelia      555-5553      amelia.zodiacusque@gmail.com    F
+ Julie       555-6699      julie.perscrutabor@skeeeve.com  F
```

Questo esempio stampa ogni record del file `mail-list` il cui primo campo contiene la stringa 'li'.

Per converso, il seguente esempio cerca 'li' nell'intero record e stampa il primo e l'ultimo campo di ogni record in input per cui è stata trovata una corrispondenza:

```
$ awk '/li/ { print $1, $NF }' mail-list
+ Amelia F
+ Broderick R
+ Julie F
+ Samuel A
```

### 4.3 Numeri di campo variabili

Un numero di campo non è necessario che sia una costante. Nel linguaggio `awk` si può usare qualsiasi espressione dopo '\$' per far riferimento a un campo. Il valore dell'espressione specifica il numero di campo. Se il valore è una stringa, piuttosto che un numero, viene convertito in un numero. Consideriamo questo esempio:

```
awk '{ print $NR }'
```

Ricordiamo che NR è il numero dei record letti fino a questo punto: uno nel primo record, due nel secondo, etc. Così quest'esempio stampa il primo campo del primo record, il secondo campo del secondo record, e così via. Per il ventesimo record, è stampato il campo numero 20; molto probabilmente il record ha meno di 20 campi, perciò stampa una riga vuota. Questo è un altro esempio sull'uso di espressioni come numeri di campo:

```
awk '{ print $(2*2) }' mail-list
```

`awk` calcola l'espressione '(2\*2)' e usa il suo valore come numero del campo da stampare. Qui '\*' rappresenta la moltiplicazione, quindi l'espressione '2\*2' ha il valore quattro. Le parentesi vengono usate affinché la moltiplicazione sia eseguita prima dell'operazione '\$'; sono necessarie ogni volta che c'è un operatore binario<sup>2</sup> nell'espressione del numero di campo. Questo esempio, dunque, stampa il tipo di relazione (il quarto campo) per ogni riga del file `mail-list`. (Tutti gli operatori di `awk` sono elencati, in ordine decrescente di precedenza, in Sezione 6.5 [Precedenza degli operatori (Come si nidificano gli operatori)], pagina 140.)

<sup>2</sup> A un operatore binario, come '\*' per la moltiplicazione, servono due operandi. La distinzione è necessaria poiché `awk` ha anche operatori unari (un operando) e ternari (tre operandi).

Se il numero di campo calcolato è zero, si ottiene l'intero record. Quindi, '\$(2-2)' ha lo stesso valore di \$0. Numeri di campo negativi non sono consentiti; tentare di far riferimento a uno di essi normalmente fa terminare il programma. (Lo standard POSIX non chiarisce cosa succede quando si fa riferimento a un numero di campo negativo. **gawk** avvisa di questo e fa terminare il programma. Altre implementazioni di **awk** possono comportarsi in modo diverso.)

Come accennato in [Sezione 4.2 \[Un'introduzione ai campi\], pagina 67](#), **awk** memorizza il numero di campi del record corrente nella variabile predefinita **NF** (si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)). Quindi, l'espressione **\$NF** non è una funzionalità speciale—è la diretta conseguenza della valutazione di **NF** e dell'uso di questo valore come numero di campo.

## 4.4 Cambiare il contenuto di un campo

Il contenuto di un campo, così come è visto da **awk**, può essere cambiato all'interno di un programma **awk**; questo cambia quello che **awk** percepisce come record in input corrente. (Il reale file in input non viene toccato; **awk** non modifica *mai* il file in input). Si consideri il seguente esempio e il suo output:

```
$ awk '{ numero_pacchi = $3 ; $3 = $3 - 10
>      print numero_pacchi, $3 }' inventory-shipped
+ 25 15
+ 32 22
+ 24 14
...
```

Il programma per prima cosa salva il valore originale del campo tre nella variabile **numero\_pacchi**. Il segno '-' rappresenta la sottrazione, così questo programma riassegna il campo tre, **\$3**, come il valore originale del campo meno dieci: '**\$3 - 10**'. (Si veda la [Sezione 6.2.1 \[Operatori aritmetici\], pagina 123](#).) Poi stampa il valore originale e quello nuovo del campo tre. (Qualcuno nel magazzino ha fatto un errore ricorrente nell'inventariare le scatole rosse.)

Perché questo funzioni, il testo in **\$3** deve poter essere riconosciuto come un numero; la stringa di caratteri dev'essere convertita in un numero affinché il computer possa eseguire operazioni aritmetiche su di essa. Il numero che risulta dalla sottrazione viene nuovamente convertito in una stringa di caratteri che quindi diventa il campo tre. Si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\], pagina 121](#).

Quando il valore di un campo è cambiato (come percepito da **awk**), il testo del record in input viene ricalcolato per contenere il nuovo campo al posto di quello vecchio. In altre parole, **\$0** cambia per riflettere il campo modificato. Questo programma stampa una copia del file in input, con 10 sottratto dal secondo campo di ogni riga:

```
$ awk '{ $2 = $2 - 10; print $0 }' inventory-shipped
+ Jan 3 25 15 115
+ Feb 5 32 24 226
+ Mar 5 24 34 228
...
```

È possibile inoltre assegnare contenuti a campi che sono fuori intervallo. Per esempio:

```
$ awk '{ $6 = ($5 + $4 + $3 + $2)
```

```
>      print $6 }' inventory-shipped
+ 168
+ 297
+ 301
...
```

Abbiamo appena creato `$6`, il cui valore è la somma dei campi `$2`, `$3`, `$4` e `$5`. Il segno `+` rappresenta l'addizione. Per il file `inventory-shipped`, `$6` rappresenta il numero totale di pacchi spediti in un determinato mese.

La creazione di un nuovo campo cambia la copia interna di `awk` nel record in input corrente, che è il valore di `$0`. Così, se si scrive `'print $0'` dopo aver aggiunto un campo, il record stampato include il nuovo campo, col numero di separatori di campo appropriati tra esso e i campi originariamente presenti.

Questa ridefinizione influenza ed è influenzata da `NF` (il numero dei campi; si veda la [Sezione 4.2 \[Un'introduzione ai campi\]](#), pagina 67). Per esempio, il valore di `NF` è impostato al numero del campo più elevato che è stato creato. Il formato preciso di `$0` è influenzato anche da una funzionalità che non è ancora stata trattata: il *separatori di campo di output*, `OFS`, usato per separare i campi (si veda la [Sezione 5.3 \[I separatori di output e come modificarli\]](#), pagina 97).

Si noti, comunque, che il mero *riferimento* a un campo fuori intervallo *non* cambia il valore di `$0` o di `NF`. Far riferimento a un campo fuori intervallo produce solo una stringa nulla. Per esempio:

```
if ($(NF+1) != "")
    print "non è possibile"
else
    print "è tutto normale"
```

dovrebbe stampare `'è tutto normale'`, perché `NF+1` è certamente fuori intervallo. (Si veda la [Sezione 7.4.1 \[L'istruzione if-else\]](#), pagina 153, per maggiori informazioni sulle istruzioni `if-else` di `awk`. Si veda la [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), pagina 130, per maggiori informazioni sull'operatore `'!='`.)

È importante notare che facendo un assegnamento a un campo esistente cambia il valore di `$0` ma non cambia il valore di `NF`, anche qualora si assegni a un campo la stringa nulla. Per esempio:

```
$ echo a b c d | awk '{ OFS = ":"; $2 = ""
>                        print $0; print NF }'
+ a::c:d
+ 4
```

Il campo è ancora lì; ha solo un valore vuoto, delimitato dai due "due punti" tra `'a'` e `'c'`. Questo esempio mostra cosa succede se si crea un nuovo campo:

```
$ echo a b c d | awk '{ OFS = ":"; $2 = ""; $6 = "nuovo"
>                        print $0; print NF }'
+ a::c:d::nuovo
+ 6
```

Il campo intermedio, `$5`, è creato con un valore vuoto (indicato dalla seconda coppia di due punti adiacenti), e `NF` è aggiornato col valore sei.



Decrementando `NF` si eliminano i campi dopo il nuovo valore di `NF` e si ricalcola `$0`. Vediamo un esempio:

```
$ echo a b c d e f | awk '{ print "NF =", NF;
>                               NF = 3; print $0 }'
+ NF = 6
+ a b c
```

**ATTENZIONE:** Alcune versioni di `awk` non ricostruiscono `$0` quando `NF` viene diminuito.

Infine, ci sono casi in cui conviene forzare `awk` a ricostruire l'intero record, usando i valori correnti dei campi e `OFS`. Per far ciò, si usa l'apparentemente innocuo assegnamento:

```
$1 = $1 # forza la ricostruzione del record
print $0 # o qualsiasi altra cosa con $0
```

Questo forza `awk` a ricostruire il record. Aggiungere un commento rende tutto più chiaro, come abbiamo appena visto.

C'è un rovescio della medaglia nella relazione tra `$0` e i campi. Qualsiasi assegnamento a `$0` fa sì che il record sia rianalizzato (sintatticamente) e ridiviso in campi usando il valore *corrente* di `FS`. Questo si applica anche a qualsiasi funzione predefinita che aggiorna `$0`, come `sub()` e `gsub()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)).

#### Comprendere `$0`

È importante ricordare che `$0` è *l'intero* record, esattamente com'è stato letto dall'input, compresi tutti gli spazi vuoti iniziali e finali, e l'esatto spazio vuoto (o altri caratteri) che separa i campi.

È un errore comune tentare di cambiare il separatore di campo in un record semplicemente impostando `FS` e `OFS`, e poi aspettarsi che un semplice `'print'` or `'print $0'` stampi il record modificato.

Questo non funziona, poiché non è stato fatto niente per cambiare quello stesso record. Invece, si deve forzare la ricostruzione del record, tipicamente con un'istruzione come `'$1 = $1'`, come descritto in precedenza.

## 4.5 Specificare come vengono separati i campi

Il *separatore di campo*, che è un carattere singolo o un'espressione regolare, controlla il modo in cui `awk` suddivide un record in input in campi. `awk` fa una scansione del record in input per trovare i caratteri che individuano il separatore; i campi sono il testo compreso tra i separatori trovati.

Nell'esempio che segue, usiamo il simbolo del punto elenco (`•`) per rappresentare gli spazi nell'output. Se il separatore di campo è `'oo'`, la seguente riga:

```
moo goo gai pan
```

è suddivisa in tre campi: `'m'`, `'•g'`, e `'•gai•pan'`. Notare gli spazi iniziali nei valori del secondo e del terzo campo.

Il separatore di campo è rappresentato dalla variabile predefinita `FS`. I programmatori di shell notino: `awk` non usa il nome `IFS` che è usato dalle shell conformi a POSIX (come la Unix Bourne shell, `sh`, o `Bash`).

Il valore di `FS` si può cambiare nel programma `awk` con l'operatore di assegnamento, `'='` (si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126). Spesso il momento giusto per far ciò è all'inizio dell'esecuzione prima che sia stato elaborato qualsiasi input, così che il primo record sia letto col separatore appropriato. Per far questo, si usa il modello di ricerca speciale `BEGIN` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali `BEGIN` ed `END`\]](#), pagina 148). Per esempio, qui impostiamo il valore di `FS` alla stringa `",,":`

```
awk 'BEGIN { FS = ",," } ; { print $2 }'
```

Data la riga in input:

```
John Q. Smith, 29 Oak St., Walamazoo, MI 42139
```

questo programma `awk` estrae e stampa la stringa `'●29●Oak●St.'`.

A volte i dati in input contengono caratteri separatori che non separano i campi nel modo in cui ci si sarebbe atteso. Per esempio, il nome della persona dell'esempio che abbiamo appena usato potrebbe avere un titolo o un suffisso annesso, come:

```
John Q. Smith, LXIX, 29 Oak St., Walamazoo, MI 42139
```

Lo stesso programma estrarrebbe `'●LXIX'` invece di `'●29●Oak●St.'`. Se ci si aspetta che il programma stampi l'indirizzo, si rimarrà sorpresi. La morale è quella di scegliere la struttura dei dati e i caratteri di separazione attentamente per evitare questi problemi. (Se i dati non sono in una forma facile da elaborare, può darsi che si possano manipolare con un programma `awk` separato.)

### 4.5.1 Lo spazio vuoto normalmente separa i campi

I campi sono separati normalmente da spazi vuoti (spazi, tabulazioni e ritorni a capo), non solo da spazi singoli. Due spazi in una riga non delimitano un campo vuoto. Il valore di default del separatore di campo `FS` è una stringa contenente un singolo spazio, `" "`. Se `awk` interpretasse questo valore nel modo usuale, ogni carattere di spazio separerebbe campi, quindi due spazi in una riga creerebbero un campo vuoto tra di essi. Il motivo per cui questo non succede è perché un singolo spazio come valore di `FS` è un caso particolare: è preso per specificare il modo di default di delimitare i campi.

Se `FS` è qualsiasi altro carattere singolo, come `",,":`, ogni ricorrenza di quel carattere separa due campi. Due ricorrenze consecutive delimitano un campo vuoto. Se il carattere si trova all'inizio o alla fine della riga, anche quello delimita un campo vuoto. Il carattere di spazio è il solo carattere singolo che non segue queste regole.

### 4.5.2 Usare *regexp* come separatori di campo

La precedente sottosezione ha illustrato l'uso di caratteri singoli o di stringhe semplici come valore di `FS`. Più in generale, il valore di `FS` può essere una stringa contenente qualsiasi espressione regolare. Se questo è il caso, ogni corrispondenza nel record con l'espressione regolare separa campi. Per esempio, l'assegnamento:

```
FS = ",, \t"
```

trasforma ogni parte di una riga in input che consiste di una virgola seguita da uno spazio e una tabulazione in un separatore di campo.

Per un esempio meno banale di espressione regolare, si provi a usare spazi singoli per separare campi nel modo in cui sono usate le virgole. FS può essere impostato a "[ ]" (parentesi quadra sinistra, spazio, parentesi quadra destra). Quest'espressione regolare corrisponde a uno spazio singolo e niente più. (si veda il [Capitolo 3 \[Espressioni regolari\]](#), [pagina 49](#)). C'è una differenza importante tra i due casi di 'FS = " "' (uno spazio singolo) e 'FS = "[\t\n]+"' (un'espressione regolare che individua uno o più spazi, tabulazioni o ritorni a capo). Per entrambi i valori di FS, i campi sono separati da *serie* (ricorrenze adiacenti multiple) di spazi, tabulazioni e/o ritorni a capo. Comunque, quando il valore di FS è " ", **awk** prima toglie lo spazio vuoto iniziale e finale dal record e poi stabilisce dove sono i campi. Per esempio, la seguente *pipeline* stampa 'b':

```
$ echo ' a b c d ' | awk '{ print $2 }'
+ b
```

Invece la *pipeline* che segue stampa 'a' (notare lo spazio extra intorno a ogni lettera):

```
$ echo ' a b c d ' | awk 'BEGIN { FS = "[\t\n]+" }
>                               { print $2 }'
+ a
```

In questo caso, il primo campo è nullo, o vuoto. Il taglio degli spazi vuoti iniziale e finale ha luogo anche ogniqualvolta \$0 è ricalcolato. Per esempio, si consideri questa *pipeline*:

```
$ echo ' a b c d ' | awk '{ print; $2 = $2; print }'
+ a b c d
+ a b c d
```

La prima istruzione **print** stampa il record così come è stato letto, con lo spazio vuoto intatto. L'assegnamento a \$2 ricostruisce \$0 concatenando insieme \$1 fino a \$NF, separati dal valore di OFS (che è uno spazio per default). Poiché lo spazio vuoto iniziale è stato ignorato quando si è trovato \$1, esso non fa parte del nuovo \$0. Alla fine, l'ultima istruzione **print** stampa il nuovo \$0.

C'è un'ulteriore sottigliezza da considerare quando si usano le espressioni regolari per separare i campi. Non è ben specificato nello standard POSIX, né altrove, cosa significhi '^' nella divisione dei campi. Il '^' cerca corrispondenze solo all'inizio dell'intero record? Oppure ogni separatore di campo è una nuova stringa? Di fatto versioni differenti di **awk** rispondono a questo quesito in modo diverso, e non si dovrebbe far affidamento su alcun comportamento specifico nei propri programmi.



Di sicuro, BWK **awk** individua con '^' solo l'inizio del record. Anche **gawk** funziona in questo modo. Per esempio:

```
$ echo 'xxAA xxBxx C' |
> gawk -F ' (^x+)| ( +)' '{ for (i = 1; i <= NF; i++)
>                               printf "-->%s<--\n", $i }'
+ --><--
+ -->AA<--
+ -->xxBxx<--
+ -->C<--
```

### 4.5.3 Fare di ogni carattere un campo separato

Ci sono casi in cui si abbia la necessità di analizzare ciascun carattere di un record separatamente. Questo si può fare in **gawk** semplicemente assegnando la stringa nulla ("" ) a

FS. (e.c.) In questo caso, ogni singolo carattere nel record diventa un campo separato. Per esempio:

```
$ echo a b | gawk 'BEGIN { FS = "" }
>
>               {
>                   for (i = 1; i <= NF; i = i + 1)
>                       print "Il campo", i, "è", $i
>               },'
+ Il campo 1 è a
+ Il campo 2 è
+ Il campo 3 è b
```

Tradizionalmente, il comportamento di FS quando è impostato a "" non è stato definito. In questo caso, la maggior parte delle versioni UNIX di **awk** trattano l'intero record come se avesse un unico campo. In modalità di compatibilità (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), se FS è la stringa nulla, anche **gawk** si comporta in questo modo.

#### 4.5.4 Impostare FS dalla riga di comando

FS può essere impostata sulla riga di comando. Per far questo si usa l'opzione **-F**. Per esempio:

```
awk -F, 'programma' i-file-di-input
```

imposta FS al carattere ','. Si noti che l'opzione richiede un carattere maiuscolo **'F'** anziché minuscolo **'f'**. Quest'ultima opzione (**-f**) serve a specificare il file contenente un programma **awk**.

Il valore usato per l'argomento di **-F** è elaborato esattamente nello stesso modo degli assegnamenti alla variabile predefinita FS. Qualsiasi carattere speciale nel separatore di campo dev'essere protetto in modo appropriato. Per esempio, per usare un **'\'** come separatore di campo sulla riga di comando, si dovrebbe battere:

```
# equivale a FS = "\"
awk -F\\ '...' file ...
```

Poiché **'\'** è usato nella shell per proteggere caratteri, a **awk** arriva **'-F\\'**. Quindi **awk** elabora **'\\'** per caratteri di protezione (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), [pagina 50](#)), producendo alla fine un unico **'\'** da usare come separatore di campo.

Come caso particolare, in modalità di compatibilità (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), se l'argomento di **-F** è **'t'**, FS è impostato al carattere di tabulazione. Se si immette **'-F\t'** nella shell, senza che sia tra apici, **'\'** viene cancellata, così **awk** conclude che si vuole realmente che i campi siano separati da tabulazioni e non da delle **'t'**. Si usi **'-v FS="t"'** o **'-F"[t]"'** sulla riga di comando se si vuole separare i campi con delle **'t'**. Quando non si è in modalità di compatibilità si deve usare **'-F '\t''** per specificare che le tabulazioni separano i campi.

Come esempio, usiamo un file di programma **awk** chiamato **edu.awk** che contiene il criterio di ricerca **/edu/** e l'azione **'print \$1'**:

```
/edu/ { print $1 }
```

Impostiamo inoltre FS al carattere **'-'** ed eseguiamo il programma sul file **mail-list**. Il seguente comando stampa un elenco dei nomi delle persone che lavorano all'università o che la frequentano, e le prime tre cifre dei loro numeri di telefono:

```
$ awk -F- -f edu.awk mail-list
-| Fabius          555
-| Samuel          555
-| Jean
```

Si noti la terza riga di output. La terza riga nel file originale è simile a questa:

```
Jean-Paul 555-2127 jeanpaul.campanorum@nyu.edu R
```

Il ‘-’ che fa parte del nome della persona è stato usato come separatore di campo, al posto del ‘-’ presente nel numero di telefono, che ci si aspettava venisse usato. Questo lascia intuire il motivo per cui si deve stare attenti nella scelta dei separatori di campo e di record.

Forse l’uso più comune di un solo carattere come separatore di campo avviene quando si elabora il file delle password di un sistema Unix. Su molti sistemi Unix, ogni utente è descritto da un elemento nel file delle password del sistema, che contiene una riga singola per ogni utente. In queste righe le informazioni sono separate da dei caratteri “:”. Il primo campo è il nome di login dell’utente e il secondo è la password dell’utente criptata o oscurata (una password oscurata è indicata dalla presenza di una sola ‘x’ nel secondo campo). Una riga nel file delle password potrebbe essere simile a questa:

```
arnold:x:2076:10:Arnold Robbins:/home/arnold:/bin/bash
```

Il seguente programma esamina il file delle password di sistema e stampa le voci relative agli utenti il cui nome completo non è presente nel file:

```
awk -F: '$5 == ""' /etc/passwd
```

#### 4.5.5 Fare di una riga intera un campo solo

Occasionalmente, è utile trattare l’intera riga in input come un solo campo. Questo si può fare facilmente e in modo portabile semplicemente impostando FS a “\n” (un ritorno a capo).<sup>3</sup>

```
awk -F'\n' 'programma' file ...
```

In questo caso, \$1 coincide con \$0.

---

<sup>3</sup> Grazie ad Andrew Schorr per questo suggerimento.

**Cambiare FS non incide sui campi**

Secondo lo standard POSIX, si suppone che **awk** si comporti come se ogni record sia stato diviso in campi nel momento in cui è stato letto. In particolare, ciò vuol dire che se si cambia il valore di **FS** dopo che un record è stato letto, il valore dei campi (cioè la loro suddivisione) sarà ancora quello ottenuto usando il precedente valore di **FS**, non quello nuovo.

Comunque, molte delle più vecchie implementazioni di **awk** non funzionano in questo modo. Invece, rimandano la divisione dei campi fino a quando si fa effettivamente riferimento a un campo. I campi sono divisi usando il valore *corrente* di **FS**! Questo comportamento può essere di difficile identificazione. Il seguente esempio illustra la differenza tra i due metodi. Lo script

```
sed 1q /etc/passwd | awk '{ FS = ":" ; print $1 }'
```

normalmente stampa:

```
└ root
```

su un'implementazione non standard di **awk**, mentre **gawk** stampa l'intera prima riga del file, qualcosa come:

```
root:x:0:0:Root:/:
```

(Il comando **sed**<sup>4</sup> appena visto stampa solo la prima riga di **/etc/passwd**.)

**4.5.6 Sommario sulla separazione dei campi**

È importante ricordare che quando si assegna una costante stringa come valore di **FS**, questa subisce una normale elaborazione di stringa da parte di **awk**. Per esempio, con Unix **awk** e **gawk**, l'assegnamento '**FS = "\."**' assegna la stringa di caratteri **"."** a **FS** (la barra inversa è tolta). Questo crea un'espressione regolare che significa "i campi sono separati da ricorrenze di due caratteri qualsiasi". Se invece si vuole che i campi siano separati da un punto seguito da un qualsiasi carattere singolo, si deve usare '**FS = "\\."**'.

Il seguente elenco riassume come i campi vengono divisi, in base al valore di **FS** ('==' significa "è uguale a"):

**FS == " "** I campi sono separati da serie di spazi vuoti. Gli spazi vuoti iniziale e finale sono ignorati. Questo è il comportamento di default.

**FS == qualsiasi altro carattere singolo**

I campi sono separati da ogni ricorrenza del carattere. Ricorrenze successive multiple delimitano campi vuoti, e lo stesso fanno le ricorrenze iniziali e finali del carattere. Il carattere può essere anche un metacarattere di espressione regolare, che non è necessario proteggere.

**FS == espressione regolare**

I campi sono separati da ricorrenze di caratteri che corrispondono alla *espressione regolare*. Corrispondenze iniziali e finali della *regex* delimitano campi vuoti.

**FS == ""** Ogni singolo carattere nel record diventa un campo separato. (Questa è un'estensione comune; non è specificata dallo standard POSIX.)

<sup>4</sup> Il programma di utilità **sed** è un "editore di flusso". Anche il suo comportamento è definito dallo standard POSIX.

## FS e IGNORECASE

La variabile `IGNORECASE` (si veda la [Sezione 7.5.1 \[Variabili predefinite modificabili per controllare `awk`\], pagina 162](#)) influisce sulla divisione del campo *solo* quando il valore di `FS` è un'espressione regolare. Non ha nessun effetto quando `FS` è un singolo carattere, anche se quel carattere è una lettera. Quindi, nel seguente codice:

```
FS = "c"
IGNORECASE = 1
$0 = "aCa"
print $1
```

L'output è `'aCa'`. Se si vuol veramente dividere i campi su un carattere alfabetico ignorandone il maiuscolo/minuscolo, si deve usare un'espressione regolare che lo farà in automatico (p.es., `'FS = "[c]"`). In questo caso, `IGNORECASE` avrà effetto.

## 4.6 Leggere campi di larghezza costante

Questa sezione tratta una funzionalità avanzata di `gawk`. Se si è un utente alle prime armi di `awk`, la si può saltare in prima lettura.

`gawk` fornisce una funzionalità per il trattamento di campi a larghezza fissa senza un separatore di campo distintivo. Per esempio, dati di questo tipo si trovano nell'input per vecchi programmi Fortran dove dei numeri sono elencati uno dopo l'altro, o nell'output di programmi che non prevedono che il loro output sia dato in input ad altri programmi.

Un esempio di quest'ultimo caso è una tabella dove tutte le colonne sono allineate usando un numero variabile di spazi e dove *i campi vuoti sono solo spazi*. Chiaramente, la normale divisione in campi di `awk` basata su `FS` non funziona bene in questa situazione. Sebbene un programma `awk` portabile possa usare una serie di chiamate `substr()` su `$0` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)), questo è scomodo e inefficiente se il numero dei campi è elevato.

La suddivisione di un record in input in campi a larghezza fissa viene specificata assegnando una stringa contenente numeri separati da spazi alla variabile predefinita `FIELDWIDTHS`. Ogni numero specifica la larghezza del campo, *comprese* le colonne tra i campi. Se si vogliono ignorare le colonne tra i campi si può specificare la loro larghezza come un campo separato che verrà poi ignorato. È un errore fatale definire una larghezza di campo che abbia un valore negativo. I dati seguenti costituiscono l'output del programma di utilità Unix `w`. È utile per spiegare l'uso di `FIELDWIDTHS`:

```
10:06pm up 21 days, 14:04, 23 users
User      tty      login idle   JCPU   PCPU   what
hzuo      ttyV0    8:58pm      9      5   vi p24.tex
hzang     ttyV3    6:37pm    50           -csh
eklye     ttyV5    9:53pm      7      1   em thes.tex
dportein  ttyV6    8:17pm  1:47           -csh
gierd     ttyD3   10:00pm      1           elm
dave      ttyD4    9:47pm      4      4    w
brent     ttyD4   26Jun91  4:46  26:46  4:41  bash
dave      ttyq4   26Jun91 15days  46     46  wnewmail
```

Il seguente programma prende l'input sopra mostrato, converte il tempo di inattività in numero di secondi, e stampa i primi due campi e il tempo di inattività calcolato:

```

BEGIN { FIELDWIDTHS = "9 6 10 6 7 7 35" }
NR > 2 {
    inat = $4
    sub(/^ +/, "", inat)    # toglie spazi prima del valore
    if (inat == "")
        inat = 0
    if (inat ~ /:/) {
        split(inat, t, ":")
        inat = t[1] * 60 + t[2]
    }
    if (inat ~ /days/)
        inat *= 24 * 60 * 60

    print $1, $2, inat
}

```

**NOTA:** Questo programma usa diverse funzionalità di **awk** non ancora trattate.

L'esecuzione del programma sui dati produce il seguente risultato:

```

hzuo      ttyV0  0
hzang     ttyV3  50
eklye     ttyV5  0
dportein  ttyV6  107
gierd     ttyD3  1
dave      ttyD4  0
brent     ttyP0  286
dave      ttyq4  1296000

```

Un altro esempio (forse più pratico) di dati di input con larghezza costante è l'input da un mazzo di schede elettorali. In alcune parti degli Stati Uniti, i votanti marcano le loro scelte perforando delle schede elettroniche.

Queste schede vengono poi elaborate per contare i voti espressi per ogni singolo candidato o su ogni determinato quesito. Siccome un votante può scegliere di non votare su alcune questioni, qualsiasi colonna della scheda può essere vuota. Un programma **awk** per elaborare tali dati potrebbe usare la funzionalità **FIELDWIDTHS** per semplificare la lettura dei dati. (Naturalmente, riuscire a eseguire **gawk** su un sistema con lettori di schede è un'altra storia!)

L'assegnazione di un valore a **FS** fa sì che **gawk** usi **FS** per separare nuovamente i campi. Si può usare **'FS = FS'** per ottenere questo effetto, senza dover conoscere il valore corrente di **FS**. Per vedere quale tipo di separazione sia in atto, si può usare **PROCINFO["FS"]** (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\]](#), pagina 165). Il suo valore è **"FS"** se si usa la normale separazione in campi, o **"FIELDWIDTHS"** se si usa la separazione in campi a larghezza fissa:

```

if (PROCINFO["FS"] == "FS")
    separazione in campi normale...
else if (PROCINFO["FS"] == "FIELDWIDTHS")
    separazione in campi a larghezza fissa...
else
    separazione dei campi in base al contenuto... (si veda

```

*la sezione successiva)*

Quest'informazione è utile quando si scrive una funzione che necessita di cambiare temporaneamente FS o FIELDWIDTHS, leggere alcuni record, e poi ripristinare le impostazioni originali (si veda la [Sezione 10.5 \[Leggere la lista degli utenti\]](#), [pagina 268](#), per un esempio di tale funzione).

## 4.7 Definire i campi in base al contenuto

Questa sezione tratta una funzionalità avanzata di **gawk**. Se si è un utente alle prime armi di **awk**, la si può saltare in prima lettura.

Normalmente, quando si usa FS, **gawk** definisce i campi come le parti del record che si trovano tra due separatori di campo. In altre parole, FS definisce cosa un campo *non è*, invece di cosa *è*. Tuttavia, ci sono casi in cui effettivamente si ha bisogno di definire i campi in base a cosa essi sono, e non in base a cosa non sono.

Il caso più emblematico è quello dei dati cosiddetti *comma-separated value* (CSV). Molti fogli elettronici, per esempio, possono esportare i dati in file di testo, dove ogni record termina con un ritorno a capo e i campi sono separati tra loro da virgole. Se le virgole facessero solo da separatore fra i dati non ci sarebbero problemi. Il problema sorge se uno dei campi contiene una virgola *al suo interno*. In queste situazioni, la maggioranza dei programmi include il campo fra doppi apici.<sup>5</sup> Così, potremmo avere dei dati di questo tipo:

```
Robbins,Arnold,"1234 A Pretty Street, NE",MyTown,MyState,12345-6789,USA
```

La variabile FPAT offre una soluzione per casi come questo. Il valore di FPAT dovrebbe essere una stringa formata da un'espressione regolare. L'espressione regolare descrive il contenuto di ciascun campo.

Nel caso dei dati CSV visti prima, ogni campo è “qualsiasi cosa che non sia una virgola,” oppure “doppi apici, seguiti da qualsiasi cosa che non siano doppi apici, e doppi apici di chiusura”. Se fosse scritta come una costante *regexp* (si veda il [Capitolo 3 \[Espressioni regolari\]](#), [pagina 49](#)), sarebbe `/([^\,]+)|("[^"]+" )/`. Dovendola scrivere come stringa si devono proteggere i doppi apici, e quindi si deve scrivere:

```
FPAT = "([^\,]+)|(\"[^\"]\"+\\\" )"
```

Come esempio pratico, si può vedere questo semplice programma che analizza e divide i dati:

```
BEGIN {
    FPAT = "([^\,]+)|(\"[^\"]\"+\\\" )"
}

{
    print "NF = ", NF
    for (i = 1; i <= NF; i++) {
        printf("%d = <%s>\n", i, $i)
    }
}
```

---

<sup>5</sup> Il formato CSV non ha avuto, per molti anni, una definizione standard formale. [RFC 4180](#) standardizza le pratiche più comuni.

Eseguendolo, avendo in input la riga vista sopra, si ottiene:

```
$ gawk -f simple-csv.awk addresses.csv
NF = 7
$1 = <Robbins>
$2 = <Arnold>
$3 = <"1234 A Pretty Street, NE">
$4 = <MyTown>
$5 = <MyState>
$6 = <12345-6789>
$7 = <USA>
```

Si noti la virgola contenuta nel valore del campo \$3.

Un semplice miglioramento se si elaborano dati CSV di questo tipo potrebbe essere quello di rimuovere i doppi apici, se presenti, con del codice di questo tipo:

```
if (substr($i, 1, 1) == "\"") {
    len = length($i)
    $i = substr($i, 2, len - 2) # Ottiene il testo tra doppi apici
}
```

Come per FS, la variabile IGNORECASE (si veda la [Sezione 7.5.1 \[Variabili predefinite modificabili per controllare awk\], pagina 162](#)) ha effetto sulla separazione dei campi con FPAT.

Se si assegna un valore a FPAT la divisione in campi non viene effettuata utilizzando FS o FIELDWIDTHS. Analogamente a FIELDWIDTHS, il valore di PROCINFO["FS"] sarà "FPAT" se è in uso la suddivisione in campi in base al contenuto.

**NOTA:** Alcuni programmi esportano dei dati CSV che contengono dei ritorni a capo al loro interno in campi rinchiusi tra doppi apici. *gawk* non è in grado di trattare questi dati. Malgrado esista una specifica ufficiale per i dati CSV, non c'è molto da fare; il meccanismo di FPAT fornisce una soluzione elegante per la maggioranza dei casi, e per gli sviluppatori di *gawk* ciò può bastare.

Come visto, l'espressione regolare usata per FPAT richiede che ogni campo contenga almeno un carattere. Una semplice modifica (cambiare il primo '+' con '\*') permette che siano presenti dei campi vuoti:

```
FPAT = "([^\,]*)|(\"[^\"]\"+\\")"
```

Infine, la funzione `patsplit()` rende la stessa funzionalità disponibile per suddividere normali stringhe (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)).

Per ricapitolare, *gawk* fornisce tre metodi indipendenti per suddividere in campi i record in input. Il meccanismo usato è determinato da quella tra le tre variabili—FS, FIELDWIDTHS, o FPAT—a cui sia stato assegnato un valore più recentemente.

## 4.8 Record su righe multiple

In alcune banche-dati, una sola riga non può contenere in modo adeguato tutte le informazioni di una voce. In questi casi si possono usare record multiriga. Il primo passo è quello di scegliere il formato dei dati.

Una tecnica è quella di usare un carattere o una stringa non usuali per separare i record. Per esempio, si può usare il carattere di interruzione di pagina (scritto ‘\f’ sia in `awk` che in C) per separarli, rendendo ogni record una pagina del file. Per far ciò, basta impostare la variabile `RS` a “\f” (una stringa contenente il carattere di interruzione di pagina). Si potrebbe ugualmente usare qualsiasi altro carattere, sempre che non faccia parte dei dati di un record.

Un’altra tecnica è quella di usare righe vuote per separare i record. Per una particolare convenzione, una stringa nulla come valore di `RS` indica che i record sono separati da una o più righe vuote. Quando `RS` è impostato alla stringa nulla, ogni record termina sempre alla prima riga vuota che è stata trovata. Il record successivo non inizia prima della successiva riga non vuota. Indipendentemente dal numero di righe vuote presenti in successione, esse costituiscono sempre un unico separatore di record. (Le righe vuote devono essere completamente vuote; righe che contengono spazi bianchi *non* sono righe vuote.)

Si può ottenere lo stesso effetto di ‘`RS = ""`’ assegnando la stringa “\n\n+” a `RS`. Quest’espressione regolare individua il ritorno a capo alla fine del record e una o più righe vuote dopo il record. In aggiunta, un’espressione regolare individua sempre la sequenza più lunga possibile quando una tale stringa sia presente. (si veda la [Sezione 3.5 \[Quanto è lungo il testo individuato?\]](#), pagina 57). Quindi, il record successivo non inizia prima della successiva riga non vuota; indipendentemente dal numero di righe vuote presenti in una voce di banca-dati, esse sono considerate come un unico separatore di record.

Comunque, c’è una sostanziale differenza tra ‘`RS = ""`’ e ‘`RS = "\n\n+"`’. Nel primo caso, i ritorni a capo iniziali nel file-dati di input vengono ignorati, e se un file termina senza righe vuote aggiuntive dopo l’ultimo record, il ritorno a capo viene rimosso dal record. Nel secondo caso, questa particolare elaborazione non viene fatta.



Ora che l’input è separato in record, il secondo passo è quello di separare i campi all’interno dei record. Un modo per farlo è quello di dividere in campi ognuna delle righe in input nel modo solito. Questo viene fatto per default tramite una speciale funzionalità. Quando `RS` è impostato alla stringa nulla e `FS` è impostato a un solo carattere, il carattere di ritorno a capo agisce *sempre* come separatore di campo. Questo in aggiunta a tutte le separazioni di campo che risultano da `FS`.<sup>6</sup>

La motivazione originale per questa particolare eccezione probabilmente era quella di prevedere un comportamento che fosse utile nel caso di default (cioè, `FS` uguale a “ ”). Questa funzionalità può costituire un problema se non si vuole che il carattere di ritorno a capo faccia da separatore tra i campi, perché non c’è alcun modo per impedirlo. Tuttavia, si può aggirare il problema usando la funzione `split()` per spezzare i record manualmente. (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Se si ha un separatore di campo costituito da un solo carattere, si può aggirare la funzionalità speciale in modo diverso, trasformando `FS` in un’espressione regolare contenente quel carattere singolo. Per esempio, se il separatore di campo è un carattere di percentuale, al posto di ‘`FS = "%"`’, si può usare ‘`FS = "[%]"`’.

Un altro modo per separare i campi è quello di mettere ciascun campo su una riga separata: per far questo basta impostare la variabile `FS` alla stringa “\n”. (Questo separatore di un solo carattere individua un singolo ritorno a capo.) Un esempio pratico di un file-dati

<sup>6</sup> Quando `FS` è la stringa nulla (“”), o un’espressione regolare, questa particolare funzionalità di `RS` non viene applicata; si applica al separatore di campo quando è costituito da un solo spazio: ‘`FS = " "`’.

organizzato in questo modo potrebbe essere un elenco di indirizzi, in cui delle righe vuote fungono da separatore fra record. Si consideri un elenco di indirizzi in un file chiamato *indirizzi*, simile a questo:

```
Jane Doe
123 Main Street
Anywhere, SE 12345-6789

John Smith
456 Tree-lined Avenue
Smallville, MW 98765-4321
...
```

Un semplice programma per elaborare questo file è il seguente:

```
# addr.s.awk --- semplice programma per una lista di indirizzi postali

# I record sono separati da righe bianche
# Ogni riga è un campo.
BEGIN { RS = "" ; FS = "\n" }

{
    print "Il nome è:", $1
    print "L'indirizzo è:", $2
    print "Città e Stato sono:", $3
    print ""
}
```

L'esecuzione del programma produce questo output:

```
$ awk -f addr.s.awk addresses
+ Il nome è: Jane Doe
+ L'indirizzo è: 123 Main Street
+ Città e Stato sono: Anywhere, SE 12345-6789
+
+ Il nome è: John Smith
+ L'indirizzo è: 456 Tree-lined Avenue
+ Città e Stato sono: Smallville, MW 98765-4321
+
...
```

Si veda la [Sezione 11.3.4 \[Stampare etichette per lettere\]](#), pagina 308, per un programma più realistico per gestire elenchi di indirizzi. Il seguente elenco riassume come sono divisi i record, a seconda del valore assunto da *RS*:

**RS == "\n"**

I record sono separati dal carattere di ritorno a capo ('\n'). In effetti, ogni riga nel file-dati è un record separato, comprese le righe vuote. Questo è il comportamento di default.

**RS == qualsiasi carattere singolo**

I record sono separati da ogni ricorrenza del carattere specificato. Più ricorrenze adiacenti delimitano record vuoti.

**RS == ""** I record sono separati da una o più righe vuote. Quando FS è un carattere singolo, il carattere di ritorno a capo serve sempre come separatore di campo, in aggiunta a qualunque valore possa avere FS. I ritorni a capo all'inizio e alla fine del file sono ignorati.

**RS == *regex***

I record sono separati da ricorrenze di caratteri corrispondenti a *regex*. Le corrispondenze iniziali e finali di *regex* designano record vuoti. (Questa è un'estensione di **gawk**; non è specificata dallo standard POSIX.)

Se non è eseguito in modalità di compatibilità (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), **gawk** imposta RT al testo di input corrispondente al valore specificato da RS. Ma se al termine del file in input non è stato trovato un testo che corrisponde a RS, **gawk** imposta RT alla stringa nulla.

## 4.9 Richiedere input usando `getline`

Finora abbiamo ottenuto i dati di input dal flusso di input principale di **awk**: lo standard input (normalmente la tastiera, a volte l'output di un altro programma) o i file indicati sulla riga di comando. Il linguaggio **awk** ha uno speciale comando predefinito chiamato `getline` che può essere usato per leggere l'input sotto il diretto controllo dell'utente.

Il comando `getline` è usato in molti modi diversi e *non* dovrebbe essere usato dai principianti. L'esempio che segue alla spiegazione del comando `getline` comprende del materiale che ancora non è stato trattato. Quindi, è meglio tornare indietro e studiare il comando `getline` *dopo* aver rivisto il resto delle Parti I e II e avere acquisito una buona conoscenza di come funziona **awk**.

Il comando `getline` restituisce 1 se trova un record e 0 se trova la fine del file. Se si verifica qualche errore cercando di leggere un record, come un file che non può essere aperto, `getline` restituisce -1. In questo caso, **gawk** imposta la variabile `ERRNO` a una stringa che descrive l'errore in questione.

Se il messaggio di errore `ERRNO` indica che l'operazione di I/O può essere ritentata e la variabile `PROCINFO["input", "RETRY"]` è impostata a 1, `getline` restituisce un codice di ritorno -2 invece che -1, e si può provare a chiamare ulteriormente `getline`. Si veda la [Sezione 4.11 \[Elaborare ulteriore input dopo certi errori di I/O\]](#), [pagina 91](#), per ulteriori informazioni riguardo a questa funzionalità.

Negli esempi seguenti, *comando* sta per un valore di stringa che rappresenta un comando della shell.

**NOTA:** Quando è stata specificata l'opzione `--sandbox` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), la lettura di input da file, *pipe* e coprocessi non è possibile.

### 4.9.1 Usare `getline` senza argomenti

Il comando `getline` può essere usato senza argomenti per leggere l'input dal file in input corrente. Tutto quel che fa in questo caso è leggere il record in input successivo e dividerlo in campi. Questo è utile se è finita l'elaborazione del record corrente, e si vogliono fare delle elaborazioni particolari sul record successivo *proprio adesso*. Per esempio:

```
# rimuovere il testo tra /* e */, compresi
```

```

{
  if ((i = index($0, "/*")) != 0) {
    prima = substr($0, 1, i - 1) # la parte iniziale della stringa
    dopo = substr($0, i + 2)     # ... */ ...
    j = index(dopo, "*/")       # */ è nella parte finale?
    if (j > 0) {
      dopo = substr(dopo, j + 2) # rimozione del commento
    } else {
      while (j == 0) {
        # passa ai record seguenti
        if (getline <= 0) {
          print("Fine file inattesa o errore:", ERRNO) > "/dev/stderr"
          exit
        }
        # incrementare la riga usando la concatenazione di stringhe
        dopo = dopo $0
        j = index(dopo, "*/") # è */ nella parte finale?
        if (j != 0) {
          dopo = substr(dopo, j + 2)
          break
        }
      }
      # incrementare la riga di output usando la concatenazione
      # di stringhe
      $0 = prima dopo
    }
    print $0
  }
}

```

Questo programma `awk` cancella i commenti in stile C (`/* ... */`) dall'input. Usa diverse funzionalità che non sono ancora state trattate, incluse la concatenazione di stringhe (si veda la [Sezione 6.2.2 \[Concatenazione di stringhe\]](#), pagina 124) e le funzioni predefinite `index()` e `substr()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Sostituendo `'print $0'` con altre istruzioni, si possono effettuare elaborazioni più complesse sull'input decommentato, come ricercare corrispondenze di un'espressione regolare. (Questo programma ha un piccolo problema: non funziona se c'è più di un commento che inizia e finisce sulla stessa riga.)

Questa forma del comando `getline` imposta `NF`, `NR`, `FNR`, `RT` e il valore di `$0`.

**NOTA:** Il nuovo valore di `$0` è usato per verificare le espressioni di ricerca di ogni regola successiva. Il valore originale di `$0` che ha attivato la regola che ha eseguito la `getline` viene perso. A differenza di `getline`, l'istruzione `next` legge un nuovo record ma inizia a elaborarlo normalmente, a partire dalla prima regola presente nel programma. Si veda la [Sezione 7.4.8 \[L'istruzione next\]](#), pagina 159.

### 4.9.2 Usare getline in una variabile

Si può usare `'getline var'` per leggere il record successivo in input ad `awk` nella variabile `var`. Non vien fatta nessun'altra elaborazione. Per esempio, supponiamo che la riga successiva sia un commento o una stringa particolare, e la si voglia leggere senza innescare nessuna regola. Questa forma di `getline` permette di leggere quella riga e memorizzarla in una variabile in modo che il ciclo principale di `awk` che "legge una riga e controlla ogni regola" non la veda affatto. L'esempio seguente inverte tra loro a due a due le righe in input:

```
{
    if ((getline tmp) > 0) {
        print tmp
        print $0
    } else
        print $0
}
```

Prende la seguente lista:

```
wan
tew
free
phore
```

e produce questo risultato:

```
tew
wan
phore
free
```

Il comando `getline` usato in questo modo imposta solo le variabili `NR`, `FNR` e `RT` (e, naturalmente, `var`). Il record non viene suddiviso in campi, e quindi i valori dei campi (compreso `$0`) e il valore di `NF` non cambiano.

### 4.9.3 Usare getline da un file

Si usa `'getline < file'` per leggere il record successivo da `file`. Qui, `file` è un'espressione di tipo stringa che specifica il nome-file. `'< file'` è una cosiddetta *ridirezione* perché richiede che l'input provenga da un posto differente. Per esempio, il seguente programma legge il suo record in input dal file `secondary.input` quando trova un primo campo con un valore uguale a 10 nel file in input corrente:

```
{
    if ($1 == 10) {
        getline < "secondary.input"
        print
    } else
        print
}
```

Poiché non viene usato il flusso principale di input, i valori di `NR` e `FNR` restano immutati. Comunque, il record in input viene diviso in modo normale, per cui vengono cambiati i valori di `$0` e degli altri campi, producendo un nuovo valore di `NF`. Viene impostato anche `RT`.

Per lo standard POSIX, `'getline < espressione'` è ambiguo se *espressione* contiene operatori che non sono all'interno di parentesi, ad esclusione di '\$'; per esempio, `'getline < dir "/" file'` è ambiguo perché l'operatore di concatenazione (non ancora trattato; si veda la [Sezione 6.2.2 \[Concatenazione di stringhe\], pagina 124](#)) non è posto tra parentesi. Si dovrebbe scrivere invece `'getline < (dir "/" file)'`, se il programma dev'essere portabile su tutte le implementazioni di `awk`.

#### 4.9.4 Usare getline in una variabile da un file

Si usa `'getline var < file'` per leggere l'input dal file *file*, e metterlo nella variabile *var*. Come prima, *file* è un'espressione di tipo stringa che specifica il file dal quale leggere.

In questa versione di `getline`, nessuna delle variabili predefinite è cambiata e il record non è diviso in campi. La sola variabile cambiata è *var*.<sup>7</sup> Per esempio, il seguente programma copia tutti i file in input nell'output, ad eccezione dei record che dicono `'@include nomefile'`. Tale record è sostituito dal contenuto del file *nomefile*:

```
{
    if (NF == 2 && $1 == "@include") {
        while ((getline line < $2) > 0)
            print line
        close($2)
    } else
        print
}
```

Si noti come il nome del file in input aggiuntivo non compaia all'interno del programma; è preso direttamente dai dati, e precisamente dal secondo campo della riga di `@include`.

La funzione `close()` viene chiamata per assicurarsi che se nell'input appaiono due righe `@include` identiche, l'intero file specificato sia incluso ogni volta. Si veda la [Sezione 5.9 \[Chiudere ridirezioni in input e in output\], pagina 109](#).

Una carenza di questo programma è che non gestisce istruzioni `@include` nidificate (cioè, istruzioni `@include` contenute nei file inclusi) nel modo in cui ci si aspetta che funzioni un vero preelaboratore di macro. Si veda la [Sezione 11.3.9 \[Una maniera facile per usare funzioni di libreria\], pagina 317](#), per un programma che gestisce le istruzioni `@include` nidificate.

#### 4.9.5 Usare getline da una pipe

*L'onniscienza ha molti aspetti positivi. Se non si può ottenerla, l'attenzione ai dettagli può aiutare.*

—Brian Kernighan

L'output di un comando può anche essere convogliato in `getline`, usando `'comando | getline'`. In questo caso, la stringa *comando* viene eseguita come comando di shell e il suo output è passato ad `awk` per essere usato come input. Questa forma di `getline` legge un record alla volta dalla *pipe*. Per esempio, il seguente programma copia il suo input nel suo output, ad eccezione delle righe che iniziano con `'@execute'`, che sono sostituite dall'output prodotto dall'esecuzione del resto della riga costituito da un comando di shell.

```
{
```

<sup>7</sup> Questo non è completamente vero. RT può essere cambiato se RS è un'espressione regolare.

```

    if ($1 == "@execute") {
        tmp = substr($0, 10)          # Rimuove "@execute"
        while ((tmp | getline) > 0)
            print
        close(tmp)
    } else
        print
}

```

La funzione `close()` viene chiamata per assicurarsi che, se appaiono nell'input due righe '@execute' identiche, il comando sia eseguito per ciascuna di esse. Dato l'input:

```

pippo
pluto
paperino
@execute who
gastone

```

il programma potrebbe produrre:

```

pippo
pluto
paperino
arnold      ttyv0    Jul 13 14:22
miriam      ttyp0     Jul 13 14:23      (murphy:0)
bill        ttyp1     Jul 13 14:23      (murphy:0)
gastone

```

Si osservi che questo programma ha eseguito `who` e stampato il risultato. (Eseguendo questo programma, è chiaro che ciascun utente otterrà risultati diversi, a seconda di chi è collegato al sistema.)

Questa variante di `getline` divide il record in campi, imposta il valore di `NF`, e ricalcola il valore di `$0`. I valori di `NR` e `FNR` non vengono cambiati. Viene impostato `RT`.

Per lo standard POSIX, '*espressione* | `getline`' è ambiguo se *espressione* contiene operatori che non sono all'interno di parentesi, ad esclusione di '\$'. Per esempio, "`echo "date" | getline`" è ambiguo perché l'operatore di concatenazione non è tra parentesi. Si dovrebbe scrivere invece '`("echo " "date") | getline`', se il programma dev'essere portabile su tutte le implementazioni di `awk`.

**NOTA:** Sfortunatamente, `gawk` non ha un comportamento uniforme nel trattare un costrutto come "`echo "date" | getline`". La maggior parte delle versioni, compresa la versione corrente, lo tratta come '`("echo " "date") | getline`'. (Questo è anche il comportamento di BWK `awk`.) Alcune versioni invece lo trattano come "`echo " ("date" | getline)`". (Questo è il comportamento di `mawk`.) In breve, per evitare problemi, è *sempre* meglio usare parentesi esplicite.

#### 4.9.6 Usare `getline` in una variabile da una *pipe*

Quando si usa '`comando | getline var`', l'output di *comando* è inviato tramite una *pipe* a `getline` ad una variabile *var*. Per esempio, il seguente programma legge la data e l'ora corrente nella variabile `current_time`, usando il programma di utilità `date`, e poi lo stampa:

```

BEGIN {

```

```

    "date" | getline current_time
    close("date")
    print "Report printed on " current_time
}

```

In questa versione di `getline`, nessuna delle variabili predefinite è cambiata e il record non è diviso in campi. In ogni caso, `RT` viene impostato.

#### 4.9.7 Usare `getline` da un coprocesso

Leggere dell'input in `getline` da una *pipe* è un'operazione unidirezionale. Il comando avviato con '`comando | getline`' invia dati *al* programma `awk`.

Occasionalmente, si potrebbe avere la necessità di inviare dei dati a un altro programma che li elabori, per poi leggere il risultato che esso genera. `gawk` permette di avviare un *coprocesso*, col quale sono possibili comunicazioni bidirezionali. Questo vien fatto con l'operatore '`|&`'. Tipicamente, dapprima si inviano dati al coprocesso e poi si leggono i risultati da esso prodotto, come mostrato di seguito:

```

    print "some query" |& "db_server"
    "db_server" |& getline

```

esso invia una richiesta a `db_server` e poi legge i risultati.

I valori di `NR` e `FNR` non vengono cambiati, perché non è cambiato il flusso principale. In ogni caso, il record è diviso in campi nel solito modo, cambiando così i valori di `$0`, degli altri campi, e di `NF` e `RT`.

I coprocessi sono una funzionalità avanzata. Vengono trattati qui solo perché questa è la sezione su `getline`. Si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\], pagina 339](#), dove i coprocessi vengono trattati più dettagliatamente.

#### 4.9.8 Usare `getline` in una variabile da un coprocesso

Quando si usa '`comando |& getline var`', l'output dal coprocesso *comando* viene inviato tramite una *pipe* bidirezionale a `getline` e nella variabile *var*.

In questa versione di `getline`, nessuna delle variabili predefinite viene cambiata e il record non viene diviso in campi. La sola variabile che cambia è *var*. In ogni caso, `RT` viene impostato.

#### 4.9.9 Cose importanti da sapere riguardo a `getline`

Qui sono elencate diverse considerazioni su `getline` da tener presenti:

- Quando `getline` cambia il valore di `$0` e `NF`, `awk` *non* salta automaticamente all'inizio del programma per iniziare a provare il nuovo record su ogni criterio di ricerca. Comunque, il nuovo record viene provato su ogni regola successiva.
- Alcune tra le prime implementazioni di `awk` limitano a una sola il numero di *pipeline* che un programma `awk` può tenere aperte. In `gawk`, non c'è questo limite. Si possono aprire tante *pipeline* (e coprocessi) quante ne permette il sistema operativo in uso.
- Un interessante effetto collaterale si ha se si usa `getline`, senza una ridirezione, all'interno di una regola `BEGIN`. Poiché una `getline` non ridiretta legge dai file-dati specificati nella riga di comando, il primo comando `getline` fa sì che `awk` imposti il valore di `FILENAME`. Normalmente, `FILENAME` non ha ancora un valore all'interno



delle regole **BEGIN**, perché non si è ancora iniziato a elaborare il file-dati della riga di comando. (Si veda [Sezione 7.1.4 \[I criteri di ricerca speciali \*\*BEGIN\*\* ed \*\*END\*\*\]](#), pagina 148; e si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\]](#), pagina 165.)

- Usare **FILENAME** con **getline** (**getline < FILENAME**) può essere fonte di confusione. **awk** apre un flusso separato di input, diverso dal file in input corrente. Comunque, poiché non si usa una variabile, **\$0** e **NF** vengono aggiornati. Se si sta facendo questo, è probabilmente per sbaglio, e si dovrebbe rivedere quello che si sta cercando di fare.
- [Sezione 4.9.10 \[Sommario delle varianti di \*\*getline\*\*\]](#), pagina 89, contiene una tabella che sintetizza le varianti di **getline** e le variabili da esse modificate. È degno di nota che le varianti che non usano la ridirezione possono far sì che **FILENAME** venga aggiornato se chiedono ad **awk** di iniziare a leggere un nuovo file in input.
- Se la variabile assegnata è un'espressione con effetti collaterali, versioni differenti di **awk** si comportano in modo diverso quando trovano la fine-del-file [EOF]. Alcune versioni non valutano l'espressione; molte versioni (compreso **gawk**) lo fanno. Si veda un esempio, gentilmente fornito da Duncan Moore:

```
BEGIN {
    system("echo 1 > f")
    while ((getline a[++c] < "f") > 0) { }
    print c
}
```

Qui l'effetto secondario è **++c**. Se viene trovata la fine del file *prima* di assegnare l'elemento **a**, **c** è incrementato o no?

**gawk** tratta **getline** come una chiamata di funzione, e valuta l'espressione **a[++c]** prima di cercare di leggere da **f**. Comunque, alcune versioni di **awk** valutano l'espressione solo se **c** è un valore di stringa da assegnare.

#### 4.9.10 Sommario delle varianti di **getline**

La [Tabella 4.1](#) riassume le otto varianti di **getline**, elencando le variabili predefinite che sono impostate da ciascuna di esse, e se la variante è standard o è un'estensione di **gawk**. Nota: per ogni variante, **gawk** imposta la variabile predefinita **RT**.

Variante	Effetto	awk / gawk
<b>getline</b>	Imposta <b>\$0</b> , <b>NF</b> , <b>FNR</b> , <b>NR</b> , e <b>RT</b>	awk
<b>getline var</b>	Imposta <b>var</b> , <b>FNR</b> , <b>NR</b> , e <b>RT</b>	awk
<b>getline &lt; file</b>	Imposta <b>\$0</b> , <b>NF</b> , e <b>RT</b>	awk
<b>getline var &lt; file</b>	Imposta <b>var</b> e <b>RT</b>	awk
<b>comando   getline</b>	Imposta <b>\$0</b> , <b>NF</b> , e <b>RT</b>	awk
<b>comando   getline var</b>	Imposta <b>var</b> e <b>RT</b>	awk
<b>comando  &amp; getline</b>	Imposta <b>\$0</b> , <b>NF</b> , e <b>RT</b>	gawk
<b>comando  &amp; getline var</b>	Imposta <b>var</b> e <b>RT</b>	gawk

Tabella 4.1: Varianti di **getline** e variabili impostate da ognuna

## 4.10 Leggere input entro un tempo limite

Questa sezione descrive una funzionalità disponibile solo in `gawk`.

Si può specificare un tempo limite in millisecondi per leggere l'input dalla tastiera, da una *pipe* o da una comunicazione bidirezionale, compresi i *socket* TCP/IP. Questo può essere fatto per input, per comando o per connessione, impostando un elemento speciale nel vettore PROCINFO (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui awk fornisce informazioni\]](#), pagina 165):

```
PROCINFO["nome_input", "READ_TIMEOUT"] = tempo limite in millisecondi
```

Se è impostato, `gawk` smette di attendere una risposta e restituisce insuccesso se non sono disponibili dati da leggere entro il limite di tempo specificato. Per esempio, un cliente TCP può decidere di abbandonare se non riceve alcuna risposta dal server dopo un certo periodo di tempo:

```
Service = "/inet/tcp/0/localhost/daytime"
PROCINFO[Service, "READ_TIMEOUT"] = 100
if ((Service |& getline) > 0)
    print $0
else if (ERRNO != "")
    print ERRNO
```

Qui vediamo come ottenere dati interattivamente dall'utente<sup>8</sup> aspettando per non più di cinque secondi:

```
PROCINFO["/dev/stdin", "READ_TIMEOUT"] = 5000
while ((getline < "/dev/stdin") > 0)
    print $0
```

`gawk` termina l'operazione di lettura se l'input non arriva entro il periodo di tempo limite, restituisce insuccesso e imposta `ERRNO` a una stringa di valore adeguato. Un valore del tempo limite negativo o pari a zero equivale a non specificare affatto un tempo limite.

Si può impostare un tempo limite anche per leggere dalla tastiera nel ciclo implicito che legge i record in input e li confronta coi criteri di ricerca, come:

```
$ gawk 'BEGIN { PROCINFO["-", "READ_TIMEOUT"] = 5000 }
> { print "You entered: " $0 }'
gawk
└─ You entered: gawk
```

In questo caso, la mancata risposta entro cinque secondi dà luogo al seguente messaggio di errore:

```
error gawk: linea com.:2: (FILENAME=- FNR=1) fatale: errore leggendo
error file in input '-': Connessione scaduta
```

Il tempo limite può essere impostato o cambiato in qualsiasi momento, e avrà effetto al tentativo successivo di leggere dal dispositivo di input. Nel seguente esempio, partiamo con un valore di tempo limite di un secondo e lo riduciamo progressivamente di un decimo di secondo finché l'attesa per l'input diventa illimitata.

```
PROCINFO[Service, "READ_TIMEOUT"] = 1000
while ((Service |& getline) > 0) {
```

<sup>8</sup> Questo presuppone che lo standard input provenga dalla tastiera.

```

    print $0
    PROCINFO[Service, "READ_TIMEOUT"] -= 100
}

```

**NOTA:** Non si deve dare per scontato che l'operazione di lettura si blocchi esattamente dopo che è stato stampato il decimo record. È possibile che **gawk** legga e tenga in memoria i dati di più di un record la prima volta. Per questo, cambiare il valore del tempo limite come nell'esempio appena visto non è molto utile.

Se l'elemento di **PROCINFO** non è presente e la variabile d'ambiente **GAWK\_READ\_TIMEOUT** esiste, **gawk** usa il suo valore per inizializzare il valore di tempo limite. L'uso esclusivo della variabile d'ambiente per specificare il tempo limite ha lo svantaggio di non essere adattabile per ogni comando o per ogni connessione.

**gawk** considera errore un superamento di tempo limite anche se il tentativo di leggere dal dispositivo sottostante potrebbe riuscire in un tentativo successivo. Questa è una limitazione, e inoltre significa che non è possibile usarlo per ottenere input multipli, provenienti da due o più sorgenti. Si veda la [Sezione 4.11 \[Elaborare ulteriore input dopo certi errori di I/O\], pagina 91](#), per una modalità che consente di tentare ulteriori operazioni di I/O.

Assegnare un valore di tempo limite previene un blocco a tempo indeterminato legato a operazioni di lettura. Si tenga però presente che ci sono altre situazioni in cui **gawk** può restare bloccato in attesa che un dispositivo di input sia pronto. Un cliente di rete a volte può impiegare molto tempo per stabilire una connessione prima di poter iniziare a leggere qualsiasi dato, oppure il tentativo di aprire un file speciale FIFO in lettura può bloccarsi indefinitamente in attesa che qualche altro processo lo apra in scrittura.

## 4.11 Elaborare ulteriore input dopo certi errori di I/O

Questa sezione descrive una funzionalità disponibile solo in **gawk**.

Qualora **gawk** incontri un errore durante la lettura dell'input, per default **getline** ha come codice di ritorno `-1`, e i successivi tentativi di leggere dallo stesso file restituiscono una indicazione di fine-file. È tuttavia possibile chiedere a **gawk** di consentire un ulteriore tentativo di lettura in presenza di certi errori, impostando uno speciale elemento del vettore **PROCINFO** (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\], pagina 165](#)):

```
PROCINFO["nome_input_file", "RETRY"] = 1
```

Quando un tale elemento esiste, **gawk** controlla il valore della variabile di sistema (nel linguaggio C) **errno** quando si verifica un errore di I/O. Se **errno** indica che un ulteriore tentativo di lettura può terminare con successo, **getline** ha come codice di ritorno `-2` e ulteriori chiamate a **getline** possono terminare correttamente. Questo vale per i seguenti valori di **errno**: **EAGAIN**, **EWOULDBLOCK**, **EINTR**, e **ETIMEDOUT**.

Questa funzionalità è utile quando si assegna un valore all'elemento **PROCINFO["nome\_input\_file", "READ\_TIMEOUT"]** o in situazioni in cui un descrittore di file sia stato configurato per comportarsi in modo non bloccante.

## 4.12 Directory sulla riga di comando

Per lo standard POSIX, i file che compaiono sulla riga di comando di **awk** devono essere file di testo; è un errore fatale se non lo sono. La maggior parte delle versioni di **awk** genera un errore fatale quando trova una directory sulla riga di comando.

Per default, **gawk** emette un avvertimento se c'è una directory sulla riga di comando, e in ogni caso la ignora. Questo rende più facile usare metacaratteri di shell col proprio programma **awk**:

```
$ gawk -f whizprog.awk * Le directory potrebbero far fallire il programma
```

Se viene data una delle opzioni **--posix** o **--traditional**, **gawk** considera invece una directory sulla riga di comando come un errore fatale.

Si veda la [Sezione 16.7.6 \[Leggere directory\], pagina 451](#), per un modo di trattare le directory come dati usabili da un programma **awk**.

## 4.13 Sommario di Input

- L'input è diviso in record in base al valore di **RS**. Le possibilità sono le seguenti:

Valore di RS	Record separati da ...	awk / gawk
Un carattere singolo	Quel carattere	awk
La stringa nulla ("")	Serie di due o più ritorni a capo	awk
Un'espressione regolare	Testo corrispondente alla <i>regex</i>	gawk

- **FNR** indica quanti record sono stati letti dal file in input corrente; **NR** indica quanti record sono stati letti in totale.
- **gawk** imposta **RT** al testo individuato da **RS**.
- Dopo la divisione dell'input in record, **awk** divide i record in singoli campi, chiamati **\$1**, **\$2** e così via. **\$0** è l'intero record, e **NF** indica quanti campi contiene. Il metodo di default per dividere i campi utilizza i caratteri di spazio vuoto.
- Si può far riferimento ai campi usando una variabile, come in **\$NF**. Ai campi possono anche essere assegnati dei valori, e questo implica che il valore di **\$0** sia ricalcolato se ad esso si fa riferimento in seguito. Fare un assegnamento a un campo con un numero maggiore di **NF** crea il campo e ricostruisce il record, usando **OFS** per separare i campi. Incrementare **NF** fa la stessa cosa. Decrementare **NF** scarta dei campi e ricostruisce il record.
- Separare i campi è più complicato che separare i record.

Valore del separatore di campo	Campi separati ...	awk / gawk
<b>FS</b> == " "	Da serie di spazi vuoti	awk
<b>FS</b> == <i>un solo carattere</i>	Da quel carattere	awk
<b>FS</b> == <i>espr. reg.</i>	Dal testo che corrisponde alla <i>regex</i>	awk
<b>FS</b> == ""	Così ogni singolo carattere è un campo separato	gawk

`FIELDWIDTHS == lista di colonne` Basata sulla posizione del `gawk` carattere

`FPAT == regexp` Dal testo attorno al testo corrispondente alla `regexp` `gawk`

- Usando `'FS = "\n"'` l'intero record sarà un unico campo (nell'ipotesi che i record siano separati da caratteri di ritorno a capo).
- `FS` può essere impostato dalla riga di comando con l'opzione `-F`. Si può fare la stessa cosa usando un assegnamento di variabile da riga di comando.
- `PROCINFO["FS"]` permette di sapere come i campi sono separati.
- `getline` nelle sue diverse forme serve per leggere record aggiuntivi provenienti dal flusso di input di default, da un file, o da una *pipe* o da un coprocesso.
- `PROCINFO[file, "READ_TIMEOUT"]` si può usare per impostare un tempo limite alle operazioni di lettura da *file*.
- Le directory sulla riga di comando generano un errore fatale per `awk` standard; `gawk` le ignora se non è in modalità POSIX.

## 4.14 Esercizi

1. Usando la variabile `FIELDWIDTHS` (si veda la [Sezione 4.6 \[Leggere campi di larghezza costante\], pagina 77](#)), scrivere un programma per leggere i dati delle elezioni, dove ogni record rappresenta i voti di un votante. Trovare un modo per definire quali colonne sono associate a ogni quesito elettorale, e stampare i voti totali, comprese le astensioni, per ciascun quesito.
2. La [Sezione 4.9.1 \[Usare `getline` senza argomenti\], pagina 83](#), ha illustrato un programma per rimuovere i commenti in stile C (`/* ... */`) dall'input. Quel programma non funziona se un commento termina in una riga e il successivo commento inizia nella stessa riga. Il problema si può risolvere con una semplice modifica. Quale?



## 5 Stampare in output

Una delle azioni che un programma fa più comunemente, è quella di produrre *stampe*, ossia scrivere in output l'input letto, tutto o in parte. Si può usare l'istruzione `print` per una stampa semplice, e l'istruzione `printf` per una formattazione dell'output più sofisticata. L'istruzione `print` non ha un limite al numero di elementi quando calcola *quali* valori stampare. Peraltro, con due eccezioni, non è possibile specificare *come* stamparli: quante colonne, se usare una notazione esponenziale o no, etc. (Per le eccezioni, si veda la [Sezione 5.3 \[I separatori di output e come modificarli\]](#), pagina 97, e la [Sezione 5.4 \[Controllare l'output di numeri con `print`\]](#), pagina 98.) Per stampare fornendo delle specifiche, è necessario usare l'istruzione `printf` (si veda la [Sezione 5.5 \[Usare l'istruzione `printf` per stampe sofisticate\]](#), pagina 98).

Oltre alla stampa semplice e formattata, questo capitolo esamina anche le ridirezioni di I/O verso file e *pipe*, introduce i nomi-file speciali che `gawk` elabora internamente, e parla della funzione predefinita `close()`.

### 5.1 L'istruzione `print`

L'istruzione `print` si usa per produrre dell'output formattato in maniera semplice, standardizzata. Si specificano solo le stringhe o i numeri da stampare, in una lista separata da virgole. Questi elementi sono stampati, separati tra loro da spazi singoli, e alla fine viene stampato un ritorno a capo. L'istruzione è simile a questa:

```
print elemento1, elemento2, ...
```

L'intera lista di elementi può facoltativamente essere racchiusa fra parentesi. Le parentesi sono obbligatorie se qualche espressione presente in uno degli elementi usa l'operatore relazionale `>`, che potrebbe essere confuso con una ridirezione dell'output (si veda la [Sezione 5.6 \[Ridirigere l'output di `print` e `printf`\]](#), pagina 104).

Gli elementi da stampare possono essere stringhe costanti o numeri, campi del record corrente (come `$1`), variabili, o qualsiasi espressione `awk`. I valori numerici sono convertiti in stringhe prima di essere stampati.

Una semplice istruzione `'print'` senza specificare elementi equivale a `'print $0'`: stampa l'intero record corrente. Per stampare una riga vuota, si usa `'print ""'`. Per stampare un testo che non cambia, si usi come elemento una costante stringa, per esempio `"Non v'allarmate"`. Dimenticandosi di mettere i doppi apici, il testo è preso per un'espressione `awk`, e probabilmente verrà emesso un messaggio di errore. Occorre tener presente che tra ogni coppia di elementi viene stampato uno spazio.

Si noti che l'istruzione `print` è un'istruzione, e non un'espressione: non è possibile usarla nella parte modello [di ricerca] di un'istruzione *criterio di ricerca-azione*, per esempio.

### 5.2 Esempi di istruzioni `print`

Ogni istruzione `print` produce almeno una riga in output. Comunque, non è limitata a una sola riga. Se il valore di un elemento è una stringa che contiene un ritorno a capo, il ritorno a capo è stampato insieme al resto della stringa. Una singola istruzione `print` può in questo modo generare un numero qualsiasi di righe.

Quel che segue è un esempio di stampa di una stringa che contiene al suo interno dei ritorni a capo:

```
$ awk 'BEGIN { print "riga uno\nriga due\nriga tre" }'
+ riga uno
+ riga due
+ riga tre
```

Il prossimo esempio, eseguito sul file `inventory-shipped`, stampa i primi due campi di ogni record in input, separandoli con uno spazio:

```
$ awk '{ print $1, $2 }' inventory-shipped
+ Jan 13
+ Feb 15
+ Mar 15
...
```

Un errore frequente usando l'istruzione `print` è quello di tralasciare la virgola tra due elementi. Questo ha spesso come risultato la stampa di elementi attaccati tra loro, senza lo spazio di separazione. Il motivo per cui ciò accade è che la scrittura di due espressioni di stringa in `awk` ne indica la concatenazione. Qui si vede l'effetto dello stesso programma, senza le virgole:

```
$ awk '{ print $1 $2 }' inventory-shipped
+ Jan13
+ Feb15
+ Mar15
...
```

Per chi non conosce il file `inventory-shipped` nessuno dei due output di esempio risulta molto comprensibile. Una riga iniziale di intestazione li renderebbe più chiari. Aggiungiamo qualche intestazione alla nostra tabella dei mesi (\$1) e dei contenitori verdi spediti (\$2). Lo facciamo usando una regola `BEGIN` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\], pagina 148](#)) in modo che le intestazioni siano stampate una volta sola:

```
awk 'BEGIN { print "Mese  Contenitori"
              print "-----" }
      { print $1, $2 }' inventory-shipped
```

Una volta eseguito, il programma stampa questo:

```
Mese  Contenitori
-----
Jan 13
Feb 15
Mar 15
...
```

Il solo problema, in effetti, è che le intestazioni e i dati della tabella non sono allineati! Possiamo provvedere stampando alcuni spazi tra i due campi:

```
awk 'BEGIN { print "Mese  Contenitori"
              print "-----" }
      { print $1, "      ", $2 }' inventory-shipped
```

Allineare le colonne in questo modo può diventare piuttosto complicato, quando ci sono parecchie colonne da tenere allineate. Contare gli spazi per due o tre colonne è semplice, ma

oltre questo limite comincia a volerci molto tempo. Ecco perché è disponibile l'istruzione `printf` (si veda la [Sezione 5.5 \[Usare l'istruzione `printf` per stampe sofisticate\]](#), pagina 98); una delle possibilità che offre è quella di allineare colonne di dati.

**NOTA:** Si può continuare su più righe sia l'istruzione `print` che l'istruzione `printf` semplicemente mettendo un ritorno a capo dopo una virgola qualsiasi (si veda la [Sezione 1.6 \[Istruzioni e righe in `awk`\]](#), pagina 28).

### 5.3 I separatori di output e come modificarli

Come detto sopra, un'istruzione `print` contiene una lista di elementi separati da virgole. Nell'output, gli elementi sono solitamente separati da spazi singoli. Non è detto tuttavia che debba sempre essere così; uno spazio singolo è semplicemente il valore di default. Qualsiasi stringa di caratteri può essere usata come *separatore di campo in output* impostando la variabile predefinita `OFS`. Il valore iniziale di questa variabile è la stringa " " (cioè, uno spazio singolo).

L'output di un'istruzione `print` completa è detto un *record di output*. Ogni istruzione `print` stampa un record di output, e alla fine ci aggiunge una stringa detta *separatore record in output* (o `ORS`). Il valore iniziale di `ORS` è la stringa "\n" (cioè, un carattere di ritorno a capo). Quindi, ogni istruzione `print` normalmente genera [almeno] una riga a sé stante.

Per cambiare il tipo di separazione in output di campi e record, si impostano valori differenti alle variabili `OFS` e `ORS`. Il posto più indicato per farlo è nella regola `BEGIN` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali `BEGIN` ed `END`\]](#), pagina 148), in modo che l'assegnazione abbia effetto prima dell'elaborazione di ogni record in input. Questi valori si possono anche impostare dalla riga di comando, prima della lista dei file in input, oppure usando l'opzione della riga di comando `-v` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). L'esempio seguente stampa il primo e il secondo campo di ogni record in input, separati da un punto e virgola, con una riga vuota aggiunta dopo ogni ritorno a capo:

```
$ awk 'BEGIN { OFS = ";"; ORS = "\n\n" }
>      { print $1, $2 }' mail-list
+ Amelia;555-5553
+
+ Anthony;555-3412
+
+ Becky;555-7685
+
+ Bill;555-1675
+
+ Broderick;555-0542
+
+ Camilla;555-2912
+
+ Fabius;555-1234
+
+ Julie;555-6699
```

```

└─
└─ Martin;555-6480
└─
└─ Samuel;555-3430
└─
└─ Jean-Paul;555-2127
└─

```

Se il valore di `ORS` non contiene un ritorno a capo, l'output del programma viene scritto tutto su un'unica riga.

## 5.4 Controllare l'output di numeri con `print`

Quando si stampano valori numerici con l'istruzione `print`, `awk` converte internamente ogni numero in una stringa di caratteri e stampa quella stringa. `awk` usa la funzione `sprintf()` per effettuare questa conversione (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 98). Per ora, basta dire che la funzione `sprintf()` accetta una *specifica di formato* che indica come formattare i numeri (o le stringhe), e che ci sono svariati modi per formattare i numeri. Le differenti specifiche di formato sono trattate più esaurientemente nella [Sezione 5.5.2 \[Lettere di controllo del formato\]](#), pagina 99.

La variabile predefinita `OFMT` contiene la specifica di formato che `print` usa con `sprintf()` per convertire un numero in una stringa per poterla stampare. Il valore di default di `OFMT` è `"%.6g"`. Il modo in cui `print` stampa i numeri si può cambiare fornendo una specifica di formato differente per il valore di `OFMT`, come mostrato nell'esempio seguente:

```

$ awk 'BEGIN {
>   OFMT = "%.0f" # Stampa numeri come interi (arrotonda)
>   print 17.23, 17.54 }'
└─ 17 18

```



Per lo standard POSIX, il comportamento di `awk` è indefinito se `OFMT` contiene qualcosa di diverso da una specifica di conversione di un numero a virgola mobile.

## 5.5 Usare l'istruzione `printf` per stampe sofisticate

Per un controllo più ampio sul formato di output di quello fornito da `print`, si può usare `printf`. Con `printf` si può specificare lo spazio da utilizzare per ogni elemento, e anche le varie scelte di formattazione disponibile per i numeri (come la base da usare in output, se stampare con notazione esponenziale, se inserire un segno, e quante cifre stampare dopo il separatore decimale).

### 5.5.1 Sintassi dell'istruzione `printf`

Una semplice istruzione `printf` è qualcosa di simile a questo:

```
printf formato, elemento1, elemento2, ...
```

Come nel caso di `print`, l'intera lista degli argomenti può facoltativamente essere racchiusa fra parentesi. Anche qui, le parentesi sono obbligatorie se l'espressione di qualche elemento usa l'operatore relazionale `>`, che potrebbe essere confuso con una ridirezione dell'output (si veda la [Sezione 5.6 \[Ridirigere l'output di `print` e `printf`\]](#), pagina 104).

La differenza tra `printf` e `print` è l'argomento *formato*. Questo è un'espressione il cui valore è visto come una stringa; specifica come scrivere in output ognuno degli altri argomenti. È chiamata *stringa di formato*.

La stringa di formato è molto simile a quella usata dalla funzione di libreria ISO C `printf()`. Buona parte del *formato* è testo da stampare così come è scritto. All'interno di questo testo ci sono degli *specificatori di formato*, uno per ogni elemento da stampare. Ogni specificatore di formato richiede di stampare l'elemento successivo nella lista degli argomenti in quella posizione del formato.

L'istruzione `printf` non aggiunge in automatico un ritorno a capo al suo output. Scrive solo quanto specificato dalla stringa di formato. Quindi, se serve un ritorno a capo, questo va incluso nella stringa di formato. Le variabili di separazione dell'output `OFS` e `ORS` non hanno effetto sulle istruzioni `printf`. Per esempio:

```
$ awk 'BEGIN {
>   ORS = "\nAHI!\n"; OFS = "+"
>   msg = "Non v\47allarmate!"
>   printf "%s\n", msg
> }'
+ Non v'allarmate!
```

Qui, né il '+' né l'esclamazione 'AHI!' compaiono nel messaggio in output.

### 5.5.2 Lettere di controllo del formato

Uno specificatore di formato inizia col carattere '%' e termina con una *lettera di controllo del formato*; e dice all'istruzione `printf` come stampare un elemento. La lettera di controllo del formato specifica che *tipo* di valore stampare. Il resto dello specificatore di formato è costituito da *modificatori* facoltativi che controllano *come* stampare il valore, per esempio stabilendo la larghezza del campo. Ecco una lista delle lettere di controllo del formato:

**%c**      Stampa un numero come un carattere; quindi, '`printf "%c", 65`' stampa la lettera 'A'. L'output per un valore costituito da una stringa è il primo carattere della stringa stessa.

**NOTA:** Lo standard POSIX richiede che il primo carattere di una stringa sia stampato. In localizzazioni con caratteri multibyte, `gawk` tenta di convertire i primi byte della stringa in un carattere multibyte valido e poi di stampare la codifica multibyte di quel carattere. Analogamente, nella stampa di un valore numerico, `gawk` ammette che il valore appartenga all'intervallo numerico di valori che possono essere contenuti in un carattere multibyte. Se la conversione alla codifica multibyte non riesce, `gawk` usa gli ultimi otto bit della cifra (quelli meno significativi) come carattere da stampare.

Altre versioni di `awk` generalmente si limitano a stampare il primo byte di una stringa o i valori numerici che possono essere rappresentati in un singolo byte (0-255).

**%d, %i**      Stampa un numero intero in base decimale. Le due lettere di controllo sono equivalenti. (La specificazione '%i' è ammessa per compatibilità con ISO C.)

- %e, %E** Stampa un numero nella notazione scientifica (con uso di esponente). Per esempio:
- ```
printf "%4.3e\n", 1950
```
- stampa '1.950e+03', con un totale di quattro cifre significative, tre delle quali seguono il punto che separa la parte intera da quella decimale [in Italia si usa la virgola al posto del punto] (L'espressione '4.3' rappresenta due modificatori, introdotti nella prossima sottosezione). '%E' usa 'E' invece di 'e' nell'output.
- %f** Stampa un numero in notazione a virgola mobile. Per esempio:
- ```
printf "%4.3f", 1950
```
- stampa '1950.000', con un totale di quattro cifre significative, tre delle quali vengono dopo il punto decimale. (L'espressione '4.3' rappresenta due modificatori, introdotti nella prossima sottosezione).
- In sistemi che implementano il formato a virgola mobile, come specificato dallo standard IEEE 754, il valore infinito negativo è rappresentato come '-inf' o '-infinity', e l'infinito positivo come 'inf' o 'infinity'. Il valore speciale "not a number" [non è un numero] viene scritto come '-nan' o 'nan' (si veda la [Sezione 15.2 \[Altre cose da sapere\], pagina 380](#)).
- %F** Come '%f', ma i valori di infinito e di "not a number" sono scritti in lettere maiuscole.
- Il formato '%F' è un'estensione POSIX allo standard ISO C; non tutti i sistemi lo prevedono. In tali casi, **gawk** usa il formato '%f'.
- %g, %G** Stampa un numero usando o la notazione scientifica o quella a virgola mobile, scegliendo la forma più concisa; se il risultato è stampato usando la notazione scientifica, '%G' usa 'E' invece di 'e'.
- %o** Stampa un numero intero in ottale, senza segno (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\], pagina 115](#)).
- %s** Stampa una stringa.
- %u** Stampa un numero intero decimale, senza segno. (Questo formato è poco usato, perché tutti i numeri in **awk** sono a virgola mobile; è disponibile principalmente per compatibilità col linguaggio C.)
- %x, %X** Stampa un intero esadecimale senza segno; '%X' usa le lettere da 'A' a 'F' invece che da 'a' a 'f' (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\], pagina 115](#)).
- %%** Stampa un solo carattere '%'. Questa notazione non serve per stampare alcun argomento e ignora eventuali modificatori.

**NOTA:** Quando si usano lettere di controllo del formato per numeri interi per stampare valori esterni all'intervallo massimo disponibile nel linguaggio C per i numeri interi, **gawk** usa lo specificatore di formato '%g'. Se si specifica l'opzione **--lint** sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), **gawk** emette un messaggio di avvertimento. Altre versioni di **awk** possono stampare valori non validi, o comportarsi in modo completamente differente.



### 5.5.3 Modificatori per specifiche di formato printf

Una specifica di formato può anche includere dei *modificatori* che possono controllare che parte stampare del valore dell'elemento, e anche quanto spazio utilizzare per stamparlo. I modificatori sono posizionati tra il '%' e la lettera che controlla il formato. Negli esempi seguenti verrà usato il simbolo del punto elenco "•" per rappresentare spazi nell'output. Questi sono i modificatori previsti, nell'ordine in cui possono apparire:

**N\$** Una costante intera seguita da un '\$' è uno *specificatore posizionale*. Normalmente, le specifiche di formato sono applicate agli argomenti nell'ordine in cui appaiono nella stringa di formato. Con uno specificatore posizionale, la specifica di formato è applicata a un argomento indicato per numero, invece che a quello che sarebbe il prossimo argomento nella lista. Gli specificatori posizionali iniziano a contare partendo da uno. Quindi:

```
printf "%s %s\n", "Non", "v'allarmate"
printf "%2$s %1$s\n", "v'allarmate", "Non"
```

stampa per due volte il famoso consiglio amichevole.

A prima vista, questa funzionalità non sembra di grande utilità. Si tratta in effetti di un'estensione *gawk*, pensata per essere usata nella traduzione di messaggi emessi in fase di esecuzione. Si veda la [Sezione 13.4.2 \[Riordinare argomenti di printf\], pagina 354](#), che descrive come e perché usare specificatori posizionali. Per ora li possiamo ignorare.

- - (Segno meno)

Il segno meno, usato prima del modificatore di larghezza (si veda più avanti in questa lista), richiede di allineare a sinistra l'argomento mantenendo la larghezza specificata. Normalmente, l'argomento è stampato allineato a destra, con la larghezza specificata. Quindi:

```
printf "%-6s", "pippo"
```

stampa 'pippo•'.

**spazio** Applicabile a conversioni numeriche, richiede di inserire uno spazio prima dei valori positivi e un segno meno prima di quelli negativi.

**+** Il segno più, usato prima del modificatore di larghezza (si veda più avanti in questa lista), richiede di mettere sempre un segno nelle conversioni numeriche, anche se il dato da formattare ha valore positivo. Il '+' prevale sul modificatore *spazio*.

**#** Richiede di usare una "forma alternativa" per alcune lettere di controllo. Per '%o', preporre uno zero. Per '%x' e '%X', preporre '0x' o '0X' se il numero è diverso da zero. Per '%e', '%E', '%f', e '%F', il risultato deve contenere sempre un separatore decimale. Per '%g' e '%G', gli zeri finali non significativi non sono tolti dal numero stampato.

**0** Uno '0' (zero) iniziale serve a richiedere che l'output sia riempito con zeri (invece che con spazi), prima delle cifre significative. Questo si applica solo ai formati di output di tipo numerico. Questo *flag* ha un effetto solo se la larghezza del campo è maggiore di quella del valore da stampare.

, Un carattere di apice singolo o un apostrofo è un'estensione POSIX allo standard ISO C. Indica che la parte intera di un valore a virgola mobile, o la parte intera di un valore decimale intero, ha un carattere di separazione delle migliaia. Ciò è applicabile solo alle localizzazioni che prevedono un tale carattere. Per esempio:

```
$ cat migliaiaia.awk          Visualizza il programma sorgente
+ BEGIN { printf "%'d\n", 1234567 }
$ LC_ALL=C gawk -f migliaiaia.awk
+ 1234567                      Risultato nella localizzazione "C"
$ LC_ALL=en_US.UTF-8 gawk -f migliaiaia.awk
+ 1,234,567                    Risultato nella localizzazione UTF inglese americana
```

Per maggiori informazioni relative a localizzazioni e internazionalizzazioni, si veda [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141.

**NOTA:** Il flag `'` è una funzionalità interessante, ma utilizza un carattere che è fonte di complicazioni, perché risulta difficile da usare nei programmi scritti direttamente sulla riga di comando. Per informazioni sui metodi appropriati per gestire la cosa, si veda [Sezione 1.1.6 \[Uso di apici nella shell.\]](#), pagina 21.

**larghezza** Questo è un numero che specifica la larghezza minima che deve occupare un campo. L'inserimento di un numero tra il segno `'%` e il carattere di controllo del formato fa sì che il campo si espanda a quella larghezza. Il modo di default per fare questo è di aggiungere degli spazi a sinistra. Per esempio:

```
printf "%6s", "pippo"
```

stampa `'•pippo'`.

Il valore di *larghezza* indica la larghezza minima, non la massima. Se il valore dell'elemento richiede più caratteri della *larghezza* specificata, questa può essere aumentata secondo necessità. Quindi, per esempio:

```
printf "%6s", "pippo-pluto"
```

stampa `'pippo-pluto'`.

Anteponendo un segno meno alla *larghezza* si richiede che l'output sia esteso con spazi a destra, invece che a sinistra.

#### **.precisione**

Un punto, seguito da una costante intera specifica la precisione da usare nella stampa. Il tipo di precisione varia a seconda della lettera di controllo:

`%d, %i, %o, %u, %x, %X`

Minimo numero di cifre da stampare.

`%e, %E, %f, %F`

Numero di cifre alla destra del separatore decimale.

`%g, %G`

Massimo numero di cifre significative.

`%s`

Massimo numero di caratteri della stringa che possono essere stampati.

Quindi, l'istruzione:

```
printf "%.4s", "foobar"

```

stampa 'foob'.

Le funzionalità di *larghezza* e *precisione* dinamiche (cioè, "%\*.\*s") disponibili nell'istruzione `printf` della libreria C sono utilizzabili. Invece che fornire esplicitamente una *larghezza* e/o una *precisione* nella stringa di formato, queste sono fornite come parte della lista degli argomenti. Per esempio:

```
w = 5
p = 3
s = "abcdefg"
printf "%*.*s\n", w, p, s

```

equivale esattamente a:

```
s = "abcdefg"
printf "%5.3s\n", s

```

Entrambi i programmi stampano '••abc'. Versioni più datate di `awk` non consentivano questa possibilità. Dovendo usare una di queste versioni, è possibile simulare questa funzionalità usando la concatenazione per costruire una stringa di formato, come per esempio:

```
w = 5
p = 3
s = "abcdefg"
printf "%" w "." p "s\n", s

```

Questo codice non è di facile lettura, ma funziona.

Chi programma in C probabilmente è abituato a specificare modificatori aggiuntivi ('h', 'j', 'l', 'L', 't' e 'z') nelle stringhe di formato di `printf`. Questi modificatori non sono validi in `awk`. La maggior parte delle implementazioni di `awk` li ignora senza emettere messaggi. Se si specifica l'opzione `--lint` sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), `gawk` emette un messaggio di avvertimento quando li si usa. Se si specifica l'opzione `--posix`, il loro uso genera un errore fatale.

#### 5.5.4 Esempi d'uso di printf

Il seguente semplice esempio mostra come usare `printf` per preparare una tabella allineata:

```
awk '{ printf "%-10s %s\n", $1, $2 }' mail-list

```

Questo comando stampa i nomi delle persone (\$1) nel file `mail-list` come una stringa di 10 caratteri allineati a sinistra. Stampa anche i numeri telefonici (\$2) a fianco, sulla stessa riga. Il risultato è una tabella allineata, contenente due colonne, di nomi e numeri telefonici, come si può vedere qui:

```
$ awk '{ printf "%-10s %s\n", $1, $2 }' mail-list
+ Amelia      555-5553
+ Anthony     555-3412
+ Becky       555-7685
+ Bill        555-1675
+ Broderick   555-0542
+ Camilla     555-2912
+ Fabius      555-1234

```

```

+ Julie      555-6699
+ Martin     555-6480
+ Samuel     555-3430
+ Jean-Paul  555-2127

```

In questo caso, i numeri telefonici debbono essere stampati come stringhe, poiché includono un trattino. Una stampa dei numeri telefonici come numeri semplici avrebbe visualizzato solo le prime tre cifre: '555', e questo non sarebbe stato di grande utilità.

Non era necessario specificare una larghezza per i numeri telefonici poiché sono nell'ultima colonna di ogni riga. Non c'è bisogno di avere un allineamento di spazi dopo di loro.

La tabella avrebbe potuto essere resa più leggibile aggiungendo intestazioni in cima alle colonne. Questo si può fare usando una regola **BEGIN** (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\], pagina 148](#)) in modo che le intestazioni siano stampate una sola volta, all'inizio del programma **awk**:

```

awk 'BEGIN { print "Nome      Numero"
              print "----      -" }
     { printf "%-10s %s\n", $1, $2 }' mail-list

```

L'esempio precedente usa sia l'istruzione **print** che l'istruzione **printf** nello stesso programma. Si possono ottenere gli stessi risultati usando solo istruzioni **printf**:

```

awk 'BEGIN { printf "%-10s %s\n", "Nome", "Numero"
              printf "%-10s %s\n", "----", "-" }
     { printf "%-10s %s\n", $1, $2 }' mail-list

```

Stampare ogni intestazione di colonna con la stessa specifica di formato usata per gli elementi delle colonne ci dà la certezza che le intestazioni sono allineate esattamente come le colonne.

Il fatto che usiamo per tre volte la stessa specifica di formato si può evidenziare memorizzandola in una variabile, così:

```

awk 'BEGIN { format = "%-10s %s\n"
              printf format, "Nome", "Numero"
              printf format, "----", "-" }
     { printf format, $1, $2 }' mail-list

```

## 5.6 Ridirigere l'output di **print** e **printf**

Finora, l'output di **print** e **printf** è stato diretto verso lo standard output, che di solito è lo schermo. Sia **print** che **printf** possono anche inviare il loro output in altre direzioni. È quel che si chiama *ridirezione*.

**NOTA:** Quando si specifica **--sandbox** (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), la ridirezione dell'output verso file, *pipe* e coprocessi non è consentita.

Una ridirezione è posta dopo l'istruzione **print** o **printf**. Le ridirezioni in **awk** sono scritte come le ridirezioni nei comandi della shell, l'unica differenza è che si trovano all'interno di un programma **awk**.

Ci sono quattro forme di ridirezione dell'output: output scritto su un file, output aggiunto in fondo a un file, output che fa da input a un altro comando (usando una *pipe*) e output

diretto a un coprocesso. Vengono descritti per l'istruzione `print`, ma funzionano allo stesso modo per `printf`:

`print elementi > output-file`

Questa ridirezione stampa gli elementi nel file di output chiamato *output-file*. Il nome-file *output-file* può essere una qualsiasi espressione. Il suo valore è trasformato in una stringa e quindi usato come nome-file (si veda il [Capitolo 6 \[Espressioni\]](#), [pagina 115](#)). Quando si usa questo tipo di ridirezione, il file *output-file* viene cancellato prima che su di esso sia stato scritto il primo record in uscita. Le successive scritture verso lo stesso file *output-file* non cancellano *output-file*, ma continuano ad aggiungervi record. (Questo comportamento è differente da quello delle ridirezioni usate negli script della shell.) Se *output-file* non esiste, viene creato. Per esempio, ecco come un programma `awk` può scrivere una lista di nomi di persone su un file di nome *lista-nomi*, e una lista di numeri telefonici su un altro file di nome *lista-telefoni*:

```
$ awk '{ print $2 > "lista-telefoni"
>      print $1 > "lista-nomi" }' mail-list
$ cat lista-telefoni
+ 555-5553
+ 555-3412
...
$ cat lista-nomi
+ Amelia
+ Anthony
...
```

Ogni file in output contiene un nome o un numero su ogni riga.

`print elementi >> output-file`

Questa ridirezione stampa gli elementi in un file di output preesistente, di nome *output-file*. La differenza tra questa ridirezione e quella con un solo `>` è che il precedente contenuto (se esiste) di *output-file* non viene cancellato. Invece, l'output di `awk` è aggiunto in fondo al file. Se *output-file* non esiste, viene creato.

`print elementi | comando`

È possibile inviare output a un altro programma usando una *pipe* invece di inviarlo a un file. Questa ridirezione apre una *pipe* verso *comando*, e invia i valori di *elementi*, tramite questa *pipe*, a un altro processo creato per eseguire *comando*.

L'argomento *comando*, verso cui è rivolta la ridirezione, è in realtà un'espressione `awk`. Il suo valore è convertito in una stringa il cui contenuto costituisce un comando della shell che deve essere eseguito. Per esempio, il seguente programma produce due file, una lista non ordinata di nomi di persone e una lista ordinata in ordine alfabetico inverso:

```
awk '{ print $1 > "nomi.non.ordinati"
      comando = "sort -r > nomi.ordinati"
      print $1 | comando }' mail-list
```

La lista non ordinata è scritta usando una ridirezione normale, mentre la lista ordinata è scritta inviando una *pipe* in input al programma di utilità `sort`.

Il prossimo esempio usa la ridirezione per inviare un messaggio alla mailing list `bug-sistema`. Questo può tornare utile se si hanno problemi con uno script `awk` eseguito periodicamente per la manutenzione del sistema:

```
report = "mail bug-sistema"
print("Script awk in errore:", $0) | report
print("al record numero", FNR, "di", NOME_FILE) | report
close(report)
```

La funzione `close()` è stata chiamata perché è una buona idea chiudere la *pipe* non appena tutto l'output da inviare alla *pipe* è stato inviato. Si veda la [Sezione 5.9 \[Chiudere ridirezioni in input e in output\]](#), pagina 109, per maggiori informazioni.

Questo esempio illustra anche l'uso di una variabile per rappresentare un *file* o un *comando*; non è necessario usare sempre una costante stringa. Usare una variabile è di solito una buona idea, perché (se si vuole riusare lo stesso file o comando) `awk` richiede che il valore della stringa sia sempre scritto esattamente nello stesso modo.

```
print elementi |& comando
```

Questa ridirezione stampa gli elementi nell'input di *comando*. La differenza tra questa ridirezione e quella con la sola `|` è che l'output da *comando* può essere letto tramite `getline`. Quindi, *comando* è un *coprocesso*, che lavora in parallelo al programma `awk`, ma è al suo servizio.

Questa funzionalità è un'estensione `gawk`, e non è disponibile in POSIX `awk`. Si veda la [Sezione 4.9.7 \[Usare `getline` da un coprocesso\]](#), pagina 88, per una breve spiegazione. [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339, per un'esposizione più esauriente.

Ridirigere l'output usando `>`, `>>`, `|` o `|&` richiede al sistema di aprire un file, una *pipe* o un coprocesso solo se il particolare *file* o *comando* che si è specificato non è già stato utilizzato in scrittura dal programma o se è stato chiuso dopo l'ultima scrittura.

È un errore comune usare la ridirezione `>` per la prima istruzione `print` verso un file, e in seguito usare `>>` per le successive scritture in output:

```
# inizializza il file
print "Non v'allarmate" > "guida.txt"
...
# aggiungi in fondo al file
print "Evitate generatori di improbabilità" >> "guide.txt"
```

Questo è il modo in cui le ridirezioni devono essere usate lavorando con la shell. Ma in `awk` ciò non è necessario. In casi di questo genere, un programma dovrebbe usare `>` per tutte le istruzioni `print`, perché il file di output è aperto una sola volta. (Usando sia `>` che `>>` nello stesso programma, l'output è prodotto nell'ordine atteso. Tuttavia il mischiare gli operatori per lo stesso file è sintomo di uno stile di programmazione inelegante, e può causare confusione in chi legge il programma.)

Come visto in precedenza (si veda la [Sezione 4.9.9 \[Cose importanti da sapere riguardo a `getline`\]](#), pagina 88), molte tra le più vecchie implementazioni di `awk` limitano il numero di *pipeline* che un programma `awk` può mantenere aperte a una soltanto! In `gawk`, non c'è

un tale limite. **gawk** consente a un programma di aprire tante *pipeline* quante ne consente il sistema operativo su cui viene eseguito.

#### Inviare pipe alla sh

Una maniera particolarmente efficace di usare la ridirezione è quella di preparare righe di comando da passare come *pipe* alla shell, **sh**. Per esempio, si supponga di avere una lista di file provenienti da un sistema in cui tutti i nomi-file sono memorizzati in maiuscolo, e di volerli rinominare in modo da avere nomi tutti in minuscolo. Il seguente programma è sia semplice che efficiente:

```
{ printf("mv %s %s\n", $0, tolower($0)) | "sh" }

END { close("sh") }
```

La funzione `tolower()` restituisce la stringa che gli viene passata come argomento con tutti i caratteri maiuscoli convertiti in minuscolo (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)). Il programma costruisce una lista di righe di comando, usando il programma di utilità `mv` per rinominare i file. Poi invia la lista alla shell per l'elaborazione.

Si veda la [Sezione 10.2.9 \[Stringhe con apici da passare alla shell\], pagina 257](#), per una funzione che può essere utile nel generare righe di comando da passare alla shell.

## 5.7 File speciali per flussi standard di dati pre-aperti

I programmi in esecuzione hanno convenzionalmente tre flussi di input e output a disposizione, già aperti per la lettura e la scrittura. Questi sono noti come lo *standard input*, lo *standard output* e lo *standard error output*. A questi flussi aperti (e a tutti gli altri file aperti o *pipe*) si fa spesso riferimento usando il termine tecnico *descrittori di file* [FD].

Questi flussi sono, per default, associati alla tastiera e allo schermo, ma spesso sono ridiretti, nella shell, utilizzando gli operatori '<', '<<', '>', '>>', '>&' e '|'. Lo standard error è tipicamente usato per scrivere messaggi di errore; la ragione per cui ci sono due flussi distinti, standard output e standard error, è per poterli ridirigere indipendentemente l'uno dall'altro.

Nelle tradizionali implementazioni di **awk**, il solo modo per scrivere un messaggio di errore allo standard error in un programma **awk** è il seguente:

```
print "Ho trovato un errore grave!" | "cat 1>&2"
```

Con quest'istruzione si apre una *pipeline* verso un comando della shell che è in grado di accedere al flusso di standard error che eredita dal processo **awk**. Questo è molto poco elegante, e richiede anche di innescare un processo separato. Per questo chi scrive programmi **awk** spesso non usa questo metodo. Invece, invia i messaggi di errore allo schermo, in questo modo:

```
print "Ho trovato un errore grave!" > "/dev/tty"
```

(`/dev/tty` è un file speciale fornito dal sistema operativo ed è connesso alla tastiera e allo schermo. Rappresenta il “terminale”,<sup>1</sup> che nei sistemi odierni è una tastiera e uno schermo, e non una *console* seriale). Questo ha generalmente lo stesso effetto, ma non è sempre detto:

<sup>1</sup> “tty” in `/dev/tty` è un'abbreviazione di “TeleTYpe” [telescrivente], un terminale seriale.

sebbene il flusso di standard error sia solitamente diretto allo schermo, potrebbe essere stato ridiretto; se questo è il caso, scrivere verso lo schermo non serve. In effetti, se **awk** è chiamato da un lavoro che non è eseguito interattivamente, può non avere a disposizione alcun terminale su cui scrivere. In quel caso, l'apertura di `/dev/tty` genera un errore.

**gawk**, **BWK awk**, e **mawk** mettono a disposizione speciali nomi-file per accedere ai tre flussi standard. Se il nome-file coincide con uno di questi nomi speciali, quando **gawk** (o uno degli altri) ridirige l'input o l'output, usa direttamente il descrittore di file identificato dal nome-file. Questi nomi-file sono gestiti così in tutti i sistemi operativi nei quali **gawk** è disponibile, e non solo in quelli che aderiscono allo standard POSIX:

`/dev/stdin`

Lo standard input (descrittore di file 0).

`/dev/stdout`

Lo standard output (descrittore di file 1).

`/dev/stderr`

Lo standard error output (descrittore di file 2).

Usando questa funzionalità la maniera corretta di scrivere un messaggio di errore diviene quindi:

```
print "Ho trovato un errore grave!" > "/dev/stderr"
```

Si noti l'uso di doppi apici per racchiudere il nome-file. Come per ogni altra ridirezione, il valore dev'essere una stringa. È un errore comune omettere i doppi apici, il che conduce a risultati inattesi.

**gawk** non tratta questi nomi-file come speciali quando opera in modalità di compatibilità POSIX. Comunque, poiché **BWK awk** li prevede, **gawk** li ammette anche quando viene invocato con l'opzione `--traditional` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)).

## 5.8 Nomi-file speciali in gawk

Oltre all'accesso a standard input, standard output e standard error, **gawk** consente di accedere a ogni descrittore di file aperto. In più, ci sono dei nomi-file speciali riservati per accedere a reti TCP/IP.

### 5.8.1 Accedere ad altri file aperti con gawk

Oltre ai valori speciali di nomi-file `/dev/stdin`, `/dev/stdout` e `/dev/stderr` già menzionati, **gawk** prevede una sintassi per accedere a ogni altro file aperto ereditato:

`/dev/fd/N`

Il file associato al descrittore di file *N*. Il file indicato deve essere aperto dal programma che inizia l'esecuzione di **awk** (tipicamente la shell). Se non sono state poste in essere iniziative speciali nella shell da cui **gawk** è stato invocato, solo i descrittori 0, 1, e 2 sono disponibili.

I nomi-file `/dev/stdin`, `/dev/stdout` e `/dev/stderr` sono essenzialmente alias per `/dev/fd/0`, `/dev/fd/1` e `/dev/fd/2`, rispettivamente. Comunque, i primi nomi sono più autoesplicativi.

Si noti che l'uso di `close()` su un nome-file della forma `"/dev/fd/N"`, per numeri di descrittore di file oltre il due, effettivamente chiude il descrittore di file specificato.

### 5.8.2 File speciali per comunicazioni con la rete

I programmi `gawk` possono aprire una connessione bidirezionale TCP/IP, fungendo o da *client* o da *server*. Questo avviene usando uno speciale nome-file della forma:

`/tipo-rete/protocollo/porta-locale/host-remoto/porta-remota`

il *tipo-rete* può essere `'inet'`, `'inet4'` o `'inet6'`. Il *protocollo* può essere `'tcp'` o `'udp'`, e gli altri campi rappresentano gli altri dati essenziali necessari per realizzare una connessione di rete. Questi nomi-file sono usati con l'operatore `'|&'` per comunicare con un coprocesso (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339). Questa è una funzionalità avanzata, qui riferita solo per completezza. Una spiegazione esauriente sarà fornita nella [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341.

### 5.8.3 Avvertimenti speciali sui nomi-file

Sono qui elencate alcune cose da tener presente usando i nomi-file speciali forniti da `gawk`:

- Il riconoscimento dei nomi-file per i tre file standard pre-aperti è disabilitato solo in modalità POSIX.
- Il riconoscimento degli altri nomi-file speciali è disabilitato se `gawk` è in modalità compatibile (o `--traditional` o `--posix`; si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).
- `gawk` interpreta *sempre* questi nomi-file speciali. Per esempio, se si usa `'/dev/fd/4'` per l'output, si scrive realmente sul descrittore di file 4, e non su un nuovo descrittore di file generato duplicando (con `dup()`) il descrittore file 4. Solitamente questo non ha importanza; comunque, è importante *non* chiudere alcun file correlato ai descrittori di file 0, 1 e 2. Se lo si fa, il comportamento risultante è imprevedibile.

## 5.9 Chiudere ridirezioni in input e in output

Se lo stesso nome-file o lo stesso comando di shell è usato con `getline` più di una volta durante l'esecuzione di un programma `awk` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83), il file viene aperto (o il comando viene eseguito) solo la prima volta. A quel punto, il primo record in input è letto da quel file o comando. La prossima volta che lo stesso file o comando è usato con `getline`, un altro record è letto da esso, e così via.

Analogamente, quando un file o una *pipe* sono aperti in output, `awk` ricorda il nome-file o comando a essi associato, e le successive scritture verso lo stesso file o comando sono accodate alle precedenti scritture. Il file o la *pipe* rimangono aperti fino al termine dell'esecuzione di `awk`.

Questo implica che sono necessari dei passi speciali per rileggere nuovamente lo stesso file dall'inizio, o per eseguire di nuovo un comando di shell (invece che leggere ulteriore output dal precedente comando). La funzione `close()` rende possibile fare queste cose:

```
close(NOME_FILE)
```

o:

```
close(comando)
```

l'argomento *NOME\_FILE* o *comando* può essere qualsiasi espressione, il cui valore dev'essere *esattamente* uguale alla stringa usata per aprire il file o eseguire il comando (spazi e altri caratteri "irrilevanti" inclusi). Per esempio, se si apre una *pipe* così:

```
"sort -r nomi" | getline pippo
```

essa va chiusa in questo modo:

```
close("sort -r nomi")
```

Una volta eseguita questa chiamata di funzione, la successiva `getline` da quel file o comando, o la successiva `print` o `printf` verso quel file o comando, riaprono il file o eseguono nuovamente il comando. Poiché l'espressione da usare per chiudere un file o una *pipeline* deve essere uguale all'espressione usata per aprire il file o eseguire il comando, è buona norma usare una variabile che contenga il nome-file o il comando. Il precedente esempio cambia come segue:

```
sortcom = "sort -r nomi"
sortcom | getline pippo
...
close(sortcom)
```

Questo aiuta a evitare nei programmi **awk** errori di battitura difficili da scoprire. Queste sono alcune delle ragioni per cui è bene chiudere un file di output:

- Scrivere un file e rileggerlo in seguito all'interno dello stesso programma **awk**. Occorre chiudere il file dopo aver finito di scriverlo, e poi iniziare a rileggerlo con `getline`.
- Per scrivere numerosi file, uno dopo l'altro, nello stesso programma **awk**. Se i file non vengono chiusi, prima o poi **awk** può superare il limite di sistema per il numero di file aperti in un processo. È meglio chiudere ogni file quando il programma ha finito di scriverlo.
- Per terminare un comando. Quando l'output è ridiretto usando una *pipe*, il comando che legge la *pipe* normalmente continua a tentare di leggere input finché la *pipe* rimane aperta. Spesso questo vuol dire che il comando non può eseguire il compito a lui assegnato finché la *pipe* non viene chiusa. Per esempio, se l'output è ridiretto al programma `mail`, il messaggio non è effettivamente inviato finché la *pipe* non viene chiusa.
- Eseguire lo stesso programma una seconda volta, con gli stessi argomenti. Questo non equivale a fornire ulteriore input alla prima esecuzione!

Per esempio, si supponga che una programma invii tramite una *pipe* dell'output al programma `mail`. Se invia parecchie linee ridirigendole a questa *pipe* senza chiuderla, queste costituiscono un solo messaggio di parecchie righe. Invece, se il programma chiude la *pipe* dopo ogni riga dell'output, allora ogni riga costituisce un messaggio separato.

Se si usano file in numero superiore a quelli che il sistema permette di mantenere aperti, **gawk** tenta di riutilizzare i file aperti disponibili fra i file-dati. La possibilità che **gawk** lo faccia dipende dalle funzionalità del sistema operativo, e quindi non è detto che questo riesca sempre. È quindi sia una buona pratica che un buon suggerimento per la portabilità quello di usare sempre `close()` sui file, una volta che si è finito di operare su di essi. In effetti, se si usano molte *pipe*, è fondamentale che i comandi vengano chiusi, una volta finita la loro elaborazione. Per esempio, si consideri qualcosa del tipo:

```
{
```

```

...
comando = ("grep " $1 " /qualche/file | un_mio_programma -q " $3)
while ((comando | getline) > 0) {
    elabora output di comando
}
# qui serve close(comando)
}

```

Questo esempio crea una nuova *pipeline* a seconda dei dati contenuti in *ogni* record. Senza la chiamata a `close()` indicata come commento, `awk` genera processi-figlio per eseguire i comandi, fino a che non finisce per esaurire i descrittori di file necessari per creare ulteriori *pipeline*.

Sebbene ogni comando sia stato completato (come si deduce dal codice di fine-file restituito dalla `getline`), il processo-figlio non è terminato;<sup>2</sup> inoltre, e questo è ciò che più ci interessa, il descrittore di file per la *pipe* non è chiuso e liberato finché non si chiama `close()` o finché il programma `awk` non termina.

`close()` non fa nulla (e non emette messaggi) se le viene fornito come argomento una stringa che non rappresenta un file, una *pipe* o un coprocesso che sia stato aperto mediante una ridirezione. In quel caso, `close()` restituisce un valore di ritorno negativo, che indica un errore. Inoltre, `gawk` imposta `ERRNO` a una stringa che indica il tipo di errore.

Si noti anche che ‘`close(NOME_FILE)`’ non ha effetti “magici” sul ciclo implicito che legge ogni record dei file indicati nella riga di comando. Si tratta, con ogni probabilità, della chiusura di un file che non era mai stato aperto con una ridirezione, e per questo `awk` silenziosamente non fa nulla, tranne impostare un codice di ritorno negativo.

Quando si usa l’operatore ‘`|&`’ per comunicare con un coprocesso, è talora utile essere in grado di chiudere un lato della *pipe* bidirezionale, senza chiudere l’altro lato. Questo è possibile fornendo un secondo argomento a `close()`. Come in ogni altra invocazione di `close()`, il primo argomento è il nome del comando o file speciale usato per iniziare il coprocesso. Il secondo argomento dovrebbe essere una stringa, con uno dei due valori “to” [a] oppure “from” [da]. Maiuscolo/minuscolo non fa differenza. Poiché questa è una funzionalità avanzata, la trattazione è rinviata alla [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339, che ne parla più dettagliatamente e fornisce un esempio.

---

<sup>2</sup> La terminologia tecnica è piuttosto macabra. Il processo-figlio terminato è chiamato “zombie,” e la pulizia alla fine dello stesso è chiamata “reaping” [mietitura].

**Usare il valore di ritorno di `close()`**

In molte versioni di Unix `awk`, la funzione `close()` è in realtà un'istruzione. È un errore di sintassi tentare di usare il valore di ritorno da `close()`:

```
comando = "..."  
comando | getline info  
retval = close(comando) # errore di sintassi in parecchi Unix awk
```

`gawk` gestisce `close()` come una funzione. Il valore di ritorno è `-1` se l'argomento designa un file che non era mai stato aperto con una ridirezione, o se c'è un problema di sistema nella chiusura del file o del processo. In tal caso, `gawk` imposta la variabile predefinita `ERRNO` a una stringa che descrive il problema.

In `gawk`, a partire dalla versione 4.2, quando si chiude una *pipe* o un coprocesso (in input o in output), il valore di ritorno è quello restituito dal comando, come descritto in [Tabella 5.1](#).<sup>3</sup>. Altrimenti, è il valore di ritorno dalla chiamata alla funzione di sistema `close()` o alla funzione C `fclose()` se si sta chiudendo un file in input o in output, rispettivamente. Questo valore è zero se la chiusura riesce, o `-1` se non riesce.

**Situazione**

Uscita normale dal comando  
Uscita dal comando per *signal*  
Uscita dal comando per *signal* con *dump*  
Errore di qualsiasi tipo

**Valore restituito da `close()`**

Il codice di ritorno del comando  
256 + numero del segnale assassino  
512 + numero del segnale assassino  
`-1`

Tabella 5.1: Valori di ritorno dalla `close()` di una *pipe*

Lo standard POSIX è molto generico; dice che `close()` restituisce zero se è terminata correttamente, e un valore diverso da zero nell'altro caso. In generale, implementazioni differenti variano in quel che restituiscono chiudendo una *pipe*; quindi, il valore di ritorno non può essere usato in modo portabile. In modalità POSIX (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), `gawk` restituisce solo zero quando chiude una *pipe*.

## 5.10 Abilitare continuazione dopo errori in output

Questa sezione descrive una funzionalità specifica di `gawk`.

In `awk` standard, l'output con `print` o `printf` a un file che non esiste, o qualche altro errore di I/O (come p.es. esaurire lo spazio disponibile su disco) è un errore fatale (termina l'esecuzione del programma).

```
$ gawk 'BEGIN { print "ciao" > "/file/non/esistente" }'  
error gawk: riga com.:1: fatale: non riesco a ri-dirigere a  
error '/file/non/esistente' (/file o directory non esistente)
```

`gawk` rende possibile accorgersi che c'è stato un errore, permettendo di tentare di rimediare, o almeno di stampare un messaggio di errore prima di terminare il programma. È possibile fare questo in due modi differenti:

- Per tutti i file in output, assegnando un valore qualsiasi a `PROCINFO["NONFATAL"]`.

<sup>3</sup> Prima della versione 4.2, il valore di ritorno dopo la chiusura di una *pipe* o di un coprocesso era una cifra di due byte (16-bit), contenente il valore restituito dalla chiamata di sistema `wait()`

- Specificamente per un solo file, assegnando un valore qualsiasi a `PROCINFO[nome_file, "NONFATAL"]`. *nome\_file* è il nome del file per il quale si desidera che l'errore di output non faccia terminare il programma.

Una volta abilitata la continuazione dopo un errore di output, si dovrà controllare la variabile `ERRNO` dopo ogni istruzione `print` o `printf` diretta a quel file, per controllare che tutto sia andato a buon fine. È anche una buona idea inizializzare `ERRNO` a zero prima di tentare l'operazione di scrittura. Per esempio:

```
$ gawk '
> BEGIN {
>     PROCINFO["NONFATAL"] = 1
>     ERRNO = 0
>     print "ciao" > "/file/non/esistente"
>     if (ERRNO) {
>         print("Output non riuscito:", ERRNO) > "/dev/stderr"
>         exit 1
>     }
> }'
```

```
error Output non riuscito: No such file or directory
```

`gawk` non genera un errore fatale; permette invece al programma `awk` di rendersi conto del problema e di gestirlo.

Questo meccanismo si applica anche allo standard output e allo standard error. Per lo standard output, si può usare `PROCINFO["-", "NONFATAL"]` o `PROCINFO["/dev/stdout", "NONFATAL"]`. Per lo standard error, occorre usare `PROCINFO["/dev/stderr", "NONFATAL"]`.

Se si tenta di aprire un *socket* TCP/IP (si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\], pagina 341](#)), `gawk` tenta di farlo per un certo numero di volte. La variabile d'ambiente `GAWK_SOCK_RETRIES` (si veda la [Sezione 2.5.3 \[Le variabili d'ambiente.\], pagina 43](#)) consente di alterare il numero di tentativi che `gawk` farebbe per default. Tuttavia, una volta abilitata la continuazione dopo un errore di I/O per un certo *socket*, `gawk` si limita a un solo tentativo, lasciando al codice del programma `awk` il compito di gestire l'eventuale problema.

## 5.11 Sommario.

- L'istruzione `print` stampa una lista di espressioni separate da virgole. Le espressioni sono separate tra loro dal valore di `OFS` e completate dal valore di `ORS`. `OFMT` fornisce il formato di conversione dei valori numerici per l'istruzione `print`.
- L'istruzione `printf` fornisce un controllo più preciso sull'output, con lettere di controllo del formato per diversi tipi di dati e vari modificatori che cambiano il comportamento delle lettere di controllo del formato.
- L'output sia di `print` che di `printf` può essere ridiretto a file, *pipe*, e coprocessi.
- `gawk` fornisce nomi-file speciali per accedere allo standard input, standard output e standard error, e per comunicazioni di rete.
- Usare `close()` per chiudere file aperti, *pipe* e ridirezioni a coprocessi. Per i coprocessi, è possibile chiudere anche soltanto una delle due direzioni di comunicazione.

- Normalmente se si verificano errori eseguendo istruzioni `print` o `printf`, questi causano la fine del programma. `gawk` consente di proseguire l'elaborazione anche in questo caso, o per un errore su qualsiasi file in output, o per un errore relativo a qualche file in particolare. Resta a carico del programma controllare se si sono verificati errori dopo l'esecuzione di ogni istruzione di output interessata.

## 5.12 Esercizi

1. Riscrivere il programma:

```
awk 'BEGIN { print "Mese  Contenitori"
           print "-----" }
     { print $1, $2 }' inventory-shipped
```

come nella [Sezione 5.3 \[I separatori di output e come modificarli\]](#), pagina 97, usando un nuovo valore per `OFS`.

2. Usare l'istruzione `printf` per allineare le intestazioni e i dati della tabella per il file di esempio `inventory-shipped` utilizzato nella [Sezione 5.1 \[L'istruzione `print`\]](#), pagina 95.
3. Spiegare cosa succede se si dimenticano i doppi apici ridirigendo l'output, come nel caso che segue:

```
BEGIN { print "Ho trovato un errore grave!" > /dev/stderr }
```

## 6 Espressioni

Le espressioni sono i mattoni fondamentali dei modelli di ricerca e delle azioni di **awk**. Un'espressione genera un valore che si può stampare, verificare o passare a una funzione. Oltre a ciò, un'espressione può assegnare un nuovo valore a una variabile o a un campo usando un operatore di assegnamento.

Un'espressione può servire come modello di ricerca o istruzione di azione a sé stante. La maggior parte degli altri tipi di istruzione contengono una o più espressioni che specificano i dati sui quali operare. Come in altri linguaggi, le espressioni in **awk** possono includere variabili, riferimenti a vettori e a costanti, e chiamate di funzione, come pure combinazioni tra questi usando diversi operatori.

### 6.1 Costanti, variabili e conversioni

Le espressioni sono costruite a partire da valori e dalle operazioni eseguite su di essi. Questa sezione descrive gli oggetti elementari che forniscono i valori usati nelle espressioni.

#### 6.1.1 Espressioni costanti

Il tipo di espressione più semplice è una *costante*, che ha sempre lo stesso valore. Ci sono tre tipi di costanti: numeriche, di stringa e di espressione regolare.

Ognuna di esse si usa nel contesto appropriato quando occorre assegnare ai dati un valore che non dovrà essere cambiato. Le costanti numeriche possono avere forme diverse, ma sono memorizzate internamente sempre allo stesso modo.

##### 6.1.1.1 Costanti numeriche e stringhe

Una *costante numerica* è un numero. Questo numero può essere un numero intero, una frazione decimale o un numero in notazione scientifica (esponenziale).<sup>1</sup> Ecco alcuni esempi di costanti numeriche che hanno tutte lo stesso valore:

```
105
1.05e+2
1050e-1
```

Una *costante stringa* è formata da una sequenza di caratteri racchiusi tra doppi apici. Per esempio:

```
"pappagallo"
```

rappresenta la stringa il cui contenuto è la parola 'pappagallo'. Le stringhe in **gawk** possono essere di qualsiasi lunghezza, e possono contenere tutti i possibili caratteri ASCII a otto bit, compreso il carattere ASCII NUL (carattere con codice zero). Altre implementazioni di **awk** possono avere difficoltà con alcuni particolari codici di carattere.

##### 6.1.1.2 Numeri ottali ed esadecimali

In **awk**, tutti i numeri sono espressi nel sistema decimale (cioè a base 10). Molti altri linguaggi di programmazione consentono di specificare i numeri in altre basi, spesso in ottale (base

---

<sup>1</sup> La rappresentazione interna di tutti i numeri, compresi gli interi, usa numeri in virgola mobile a doppia precisione. Sui sistemi più moderni, questi sono nel formato standard IEEE 754. Si veda la [Capitolo 15 \[Calcolo con precisione arbitraria con gawk\]](#), pagina 379, per maggiori informazioni.

8) e in esadecimale (base 16). Nel sistema ottale i numeri hanno la sequenza 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, e così via. Come ‘11’ decimale è una volta 10 più 1, così ‘11’ ottale è una volta 8 più 1. Questo equivale a 9 nel sistema decimale. Nell’esadecimale ci sono 16 cifre. Poiché l’usuale sistema numerico decimale ha solo dieci cifre (‘0’–‘9’), le lettere da ‘a’ a ‘f’ rappresentano le cifre ulteriori. (normalmente è irrilevante se le lettere sono maiuscole o minuscole; gli esadecimali ‘a’ e ‘A’ hanno lo stesso valore). Così, ‘11’ esadecimale è 1 volta 16 più 1, il che equivale a 17 decimale.

Guardando solamente un ‘11’ puro e semplice, non si capisce in quale base sia. Così, in C, C++, e in altri linguaggi derivati da C, c’è una speciale notazione per esprimere la base. I numeri ottali iniziano con uno ‘0’, e i numeri esadecimali iniziano con uno ‘0x’ o ‘0X’:

11	Valore decimale 11
011	11 ottale, valore decimale 9
0x11	11 esadecimale, valore decimale 17

Quest’esempio mostra le differenze:

```
$ gawk 'BEGIN { printf "%d, %d, %d\n", 011, 11, 0x11 }'
+ 9, 11, 17
```

Poter usare costanti ottali ed esadecimali nei propri programmi è molto utile quando si lavora con dati che non possono essere rappresentati convenientemente come caratteri o come numeri regolari, come i dati binari di vario tipo.

**gawk** permette l’uso di costanti ottali ed esadecimali nel testo di un programma. Comunque, se numeri non-decimali sono presenti tra i dati in input, essi non sono trattati in maniera speciale; trattarli in modo speciale per default significherebbe che vecchi programmi **awk** produrrebbero risultati errati. (Se si vuole che i numeri siano trattati in maniera speciale, si deve specificare l’opzione da riga di comando `--non-decimal-data`; si veda la [Sezione 12.1 \[Consentire dati di input non decimali\]](#), pagina 331.) Se si devono gestire dati ottali o esadecimali, si può usare la funzione `strtonum()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198) per convertire i dati in numeri. La maggior parte delle volte, le costanti ottali o esadecimali si usano quando si lavora con le funzioni predefinite di manipolazione di bit; si veda [Sezione 9.1.6 \[Funzioni per operazioni di manipolazione bit\]](#), pagina 219, per maggiori informazioni.

Diversamente da alcune tra le prime implementazioni di C, ‘8’ e ‘9’ non sono cifre valide nelle costanti ottali. Per esempio, **gawk** tratta ‘018’ come un 18 decimale:

```
$ gawk 'BEGIN { print "021 è", 021 ; print 018 }'
+ 021 è 17
+ 18
```

Le costanti ottali ed esadecimali nel codice sorgente sono un’estensione di **gawk**. Se **gawk** è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), non sono disponibili.

### La base di una costante non influisce sul suo valore

Una volta che una costante numerica è stata convertita internamente in un numero [decimale], **gawk** non considera più la forma originale della costante; viene sempre usato il valore interno. Questo ha delle precise conseguenze per la conversione dei numeri in stringhe:

```
$ gawk 'BEGIN { printf "0x11 vale <%s>\n", 0x11 }'
+ 0x11 vale <17>
```

#### 6.1.1.3 Costanti fornite tramite espressioni regolari

Una *costante regexp* è la descrizione di un'espressione regolare delimitata da barre, come `/^inizio e fine$/. La maggior parte delle regexp usate nei programmi awk sono costanti, ma gli operatori di confronto '~' e '!~' possono confrontare anche regexp calcolate o dinamiche (che tipicamente sono solo stringhe ordinarie o variabili che contengono un'espressione regolare, ma potrebbero anche essere espressioni più complesse).`

#### 6.1.2 Usare espressioni regolari come costanti

Le costanti *regexp* sono costituite da testo che descrive un'espressione regolare, racchiusa fra barre (come p.e. `/la +risposta/`). Questa sezione tratta del comportamento di tali costanti in POSIX **awk** e in **gawk**, e prosegue descrivendo le *costanti regexp fortemente tipizzate*, che sono un'estensione **gawk**.

##### 6.1.2.1 Costanti *regexp* normali in **awk**.

Quando è usata a destra degli operatori `'~'` o `'!~'`, una costante *regexp* rappresenta semplicemente l'espressione regolare che dev'essere confrontata. Comunque, le costanti *regexp* (come `/pippo/`) possono essere usate come semplici espressioni. Quando una costante *regexp* compare da sola, ha lo stesso significato di quando compare in un criterio di ricerca (cioè `('$0 ~ /pippo/)`). Si veda la [Sezione 7.1.2 \[Espressioni come criteri di ricerca\]](#), pagina 146. Ciò vuol dire che i due frammenti di codice seguenti:

```
if ($0 ~ /barfly/ || $0 ~ /camelot/)
    print "trovato"
```

e:

```
if (/barfly/ || /camelot/)
    print "trovato"
```

sono esattamente equivalenti. Una conseguenza piuttosto bizzarra di questa regola è che la seguente espressione booleana è valida, ma non fa quel che probabilmente l'autore si aspettava:

```
# Notare che /pippo/ è alla sinistra della ~
if (/pippo/ ~ $1) print "trovato pippo"
```

Questo codice “ovviamente” intende verificare se `$1` contiene l'espressione regolare `/pippo/`. Ma in effetti l'espressione `'/pippo/ ~ $1'` significa realmente `('$0 ~ /pippo/) ~ $1'`. In altre parole, prima confronta il record in input con l'espressione regolare `/pippo/`. Il risultato è zero o uno, a seconda che il confronto dia esito positivo o negativo. Questo risultato è poi confrontato col primo campo nel record. Siccome è improbabile che qualcuno voglia mai fare realmente questo genere di test, **gawk** emette un avvertimento quando vede questo costrutto in un programma. Un'altra conseguenza di questa regola è che l'istruzione di assegnamento:

```
confronta = /pippo/
```



assegna zero o uno alla variabile `confronta`, a seconda del contenuto del record in input corrente.

Le espressioni regolari costanti possono essere usate anche come primo argomento delle funzioni `gensub()`, `sub()`, e `gsub()`, come secondo argomento della funzione `match()`, e come terzo argomento delle funzioni `split()` e `patsplit()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Le moderne implementazioni di `awk`, incluso `gawk`, permettono di usare come terzo argomento di `split()` una costante *regex*, ma alcune implementazioni più vecchie non lo consentono. Poiché alcune funzioni predefinite accettano costanti *regex* come argomenti, può generare confusione l'uso di costanti *regex* come argomenti di funzioni definite dall'utente (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224). Per esempio:

```
function mysub(modello, sostituzione, stringa, globale)
{
    if (globale)
        gsub(modello, sostituzione, stringa)
    else
        sub(modello, sostituzione, stringa)
    return stringa
}

{
    ...
    text = "salve! salve a te!"
    mysub(/salve/, "ciao", text, 1)
    ...
}
```

In quest'esempio, il programmatore vuol passare una costante *regex* alla funzione definita dall'utente `mysub()`, che a sua volta la passa o a `sub()` o a `gsub()`. Comunque, quel che realmente succede è che al parametro `modello` è assegnato un valore di uno o zero, a seconda che `$0` corrisponda a `/salve/` o no. `gawk` emette un avvertimento quando vede una costante *regex* usata come parametro di una funzione definita dall'utente, poiché passare un valore vero/falso in questo modo probabilmente non è quello che si intendeva fare.

### 6.1.2.2 Costanti *regex* fortemente tipizzate

Questa sezione descrive una funzionalità specifica di `gawk`.

Come visto nella precedente, le costanti *regex* (`/.../`) hanno uno strano posto nel linguaggio `awk`. In molti contesti, si comportano come un'espressione: `'$0 ~ /.../'`. In altri contesti, denotano solo una *regex* da individuare. In nessun caso esse sono davvero “cittadine di serie A” del linguaggio. Ovvero, non è possibile definire una variabile scalare il cui tipo sia “*regex*” nello stesso senso in cui si può definire una variabile che sia un numero o una stringa:

<code>num = 42</code>	<i>Variabile numerica</i>
<code>str = "hi"</code>	<i>Variabile di tipo stringa</i>
<code>re = /pippo/</code>	<i>Errore! re è il risultato di \$0 ~ /pippo/</i>

Per alcuni casi di uso più avanzati, sarebbe bello poter avere costanti *regex* che sono *fortemente tipizzate*; in altre parole, che sostituiscono una *regex* utile per effettuare dei confronti, e non una semplice espressione regolare.

**gawk** prevede questa funzionalità. Un'espressione regolare fortemente tipizzata è molto simile a un'espressione regolare normale, tranne per il fatto di essere preceduta dal simbolo '@':

```
re = @/foo/      variabile regex fortemente tipizzata
```

Le variabili *regex* fortemente tipizzate *non possono* essere usate in ogni istruzione in cui compare un'espressione regolare normale, perché ciò renderebbe il linguaggio ancora più fuorviante. Queste espressioni possono essere usate solo in alcuni contesti:

- Sul lato destro degli operatori '~' e '!~': `qualche_variabile ~ @/foo/` (si veda la [Sezione 3.1 \[Uso di espressioni regolari\]](#), pagina 49).
- Nella parte **case** di un'istruzione **switch** (si veda la [Sezione 7.4.5 \[L'istruzione switch\]](#), pagina 156).
- Come argomento in una delle funzioni predefinite che possono utilizzare costanti *regex*: `gensub()`, `gsub()`, `match()`, `patsplit()`, `split()` e `sub()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- Come parametro in una chiamata a una funzione definita dall'utente (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224).
- Sul lato destro di un assegnamento di variabile: `qualche_variabile = @/foo/`. In tal caso, `qualche_variabile` è di tipo *regex*. Inoltre, `qualche_variabile` può essere usata con gli operatori '~' e '!~', passata a una delle funzioni predefinite sopra elencate o passata come parametro a una funzione definita dall'utente.

Si può usare la funzione predefinita `typeof()` (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), pagina 223) per determinare se un parametro passato a una funzione è una variabile di tipo *regex*.

La vera efficacia di questa funzionalità consiste nella capacità di creare variabili il cui tipo è *regex*. Tali variabili possono essere passate a funzioni definite dall'utente, senza gli inconvenienti che si hanno usando espressioni regolari calcolate, a partire da stringhe o da costanti di tipo stringa. Queste variabili possono anche essere passate utilizzando chiamate indirette a funzioni (si veda la [Sezione 9.3 \[Chiamate indirette di funzione\]](#), pagina 234) e alle funzioni predefinite che accettano costanti di tipo *regesp*.

Quando sono usate per effettuare conversioni numeriche, le variabili *regex* fortemente tipizzate vengono convertite alla cifra zero. Quando sono usate per effettuare conversioni a stringhe, vengono convertite al valore di stringa del testo della *regex* originale.

### 6.1.3 Variabili

Le *variabili* consentono di memorizzare valori in un certo punto di un programma, per usarli più tardi in un'altra parte del programma. Le variabili possono essere gestite interamente all'interno del testo del programma, ma ad esse possono essere assegnati valori sulla riga di comando, in fase di invocazione di **awk**.

#### 6.1.3.1 Usare variabili in un programma

Le variabili permettono di dare nomi ai valori e di far riferimento ad essi in un secondo momento. Alcune variabili sono già state usate in molti degli esempi. Il nome di una variabile

dev'essere una sequenza di lettere, cifre o trattini bassi, e non deve iniziare con una cifra. Qui, una *lettera* è una qualsiasi delle 52 lettere maiuscole e minuscole dell'alfabeto inglese. Altri caratteri che possono essere definiti come lettere in localizzazioni non inglesi non sono validi nei nomi di variabile. Il maiuscolo o minuscolo sono significativi nei nomi di variabile; `a` e `A` sono variabili diverse.

Un nome di variabile è un'espressione valida in se stessa; rappresenta il valore corrente della variabile. I valori delle variabili possono essere modificati tramite *operatori di assegnamento*, *operatori di incremento* e *operatori di decremento* (Si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126). Inoltre, le funzioni `sub()` e `gsub()` possono cambiare il valore di una variabile e le funzioni `match()`, `split()`, e `patsplit()` possono cambiare il contenuto dei loro parametri che sono costituiti da vettori (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).

Alcune variabili hanno un significato speciale predefinito, come `FS` (il separatore di campo) e `NF` (il numero di campi nel record di input corrente). Si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162, per un elenco delle variabili predefinite. Queste variabili predefinite possono essere usate e possono ricevere assegnamenti come tutte le altre variabili, ma i loro valori sono anche usati o cambiati automaticamente da `awk`. Tutti i nomi delle variabili predefinite sono in caratteri maiuscoli.

Alle variabili in `awk` possono essere assegnati valori numerici o valori di stringa. Il tipo di valore che una variabile contiene può cambiare durante la vita di un programma. Per default, le variabili sono inizializzate alla stringa nulla, che vale zero se viene convertita in un numero. Non c'è alcuna necessità di inizializzare esplicitamente una variabile in `awk`, come invece occorre fare in C e nella maggior parte dei linguaggi tradizionali.

### 6.1.3.2 Assegnare una variabile dalla riga di comando

Si può impostare qualsiasi variabile `awk` includendo un *assegnamento di variabile* tra gli argomenti sulla riga di comando quando viene invocato `awk` (si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\]](#), pagina 40). Tale assegnamento ha la seguente forma:

```
variabile=testo
```

Con questo assegnamento, una variabile viene impostata o all'inizio dell'esecuzione di `awk` o tra la lettura di un file in input e il successivo file in input. Quando l'assegnamento è preceduto dall'opzione `-v`, come nel seguente esempio:

```
-v variabile=testo
```

la variabile è impostata proprio all'inizio, ancor prima che sia eseguita la regola `BEGIN`. L'opzione `-v` e il suo assegnamento deve precedere tutti gli argomenti nome-file, e anche il testo del programma. (Si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33, per maggiori informazioni sull'opzione `-v`.) In alternativa, l'assegnamento di variabile è effettuata in un momento determinato dalla posizione dell'opzione tra gli argomenti "file in input", cioè dopo l'elaborazione del precedente argomento "file in input". Per esempio:

```
awk '{ print $n }' n=4 inventory-shipped n=2 mail-list
```

stampa il valore del campo numero `n` per tutti i record in input. Prima che venga letto il primo file, la riga di comando imposta la variabile `n` al valore quattro. Questo fa sì che venga stampato il quarto campo delle righe del file `inventory-shipped`. Dopo la fine del primo file, ma prima che inizi il secondo file, `n` viene impostato a due, e quindi poi viene stampato il secondo campo delle righe di `mail-list`:

```
$ awk '{ print $n }' n=4 inventory-shipped n=2 mail-list
+ 15
+ 24
...
+ 555-5553
+ 555-3412
...
```

Gli argomenti da riga di comando sono resi disponibili dal programma `awk` nel vettore `ARGV` per poter essere esaminati esplicitamente (si veda la [Sezione 7.5.3 \[Usare ARGV e ARGV\]](#), [pagina 172](#)). `awk` elabora i valori degli assegnamenti da riga di comando per sequenze di protezione (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), [pagina 50](#)).



### 6.1.4 Conversione di stringhe e numeri

Le conversioni di numeri in stringhe e di stringhe in numeri sono generalmente semplici. Ci possono essere delle sottigliezze che bisogna tenere presenti; questa sezione tratta di quest'importante sfaccettatura di `awk`.

#### 6.1.4.1 Come awk converte tra stringhe e numeri

Le stringhe sono convertite in numeri e i numeri sono convertiti in stringhe, se il contesto del programma `awk` lo richiede. Per esempio, se il valore di `pip` o `pluto` nell'espressione '`pip + pluto`' è una stringa, viene convertita in un numero prima di eseguire l'addizione. Se in una concatenazione di stringhe ci sono valori numerici, questi sono convertiti in stringhe. Si consideri il seguente esempio:

```
due = 2; tre = 3
print (due tre) + 4
```

Stampa il valore (numerico) di 27. I valori numerici delle variabili `due` e `tre` sono convertiti in stringhe e concatenati insieme. La stringa risultante è riconvertita nel numero 23, al quale poi viene aggiunto 4.

Se, per qualche ragione, si vuole forzare la conversione di un numero in una stringa, basta concatenare a quel numero la stringa nulla, `""`. Per forzare la conversione di una stringa in un numero, basta aggiungere zero a quella stringa. Una stringa viene convertita in un numero interpretando qualsiasi prefisso numerico della stringa come numero: `"2.5"` si converte in 2.5, `"1e3"` si converte in 1000, e `"25fix"` ha un valore numerico di 25. Le stringhe che non possono essere interpretate come numeri validi vengono convertite al valore zero.

Il modo esatto in cui i numeri sono convertiti in stringhe è controllato dalla variabile predefinita di `awk` `CONVFMT` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), [pagina 162](#)). I numeri vengono convertiti usando la funzione `sprintf()` con `CONVFMT` come specificatore di formato (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)).

Il valore di default di `CONVFMT` è `"%.6g"`, che crea un valore con un massimo di sei cifre significative. Per alcune applicazioni potrebbe essere opportuno cambiare questo valore per ottenere una maggiore precisione. Sulla maggior parte delle macchine moderne normalmente bastano 17 cifre per esprimere esattamente il valore di un numero in virgola mobile.<sup>2</sup>

<sup>2</sup> Per casi eccezionali possono essere richieste fino a 752 cifre (!), non sembra che sia il caso di preoccuparsene qui.

Si possono avere strani risultati se si imposta `CONVFMT` a una stringa che non indica a `sprintf()` come formattare i numeri in virgola mobile in un modo utile. Per esempio, se ci si dimentica la `'%`' nel formato, `awk` converte tutti i numeri alla stessa stringa costante.

Come caso particolare, per un numero intero, il risultato della conversione a una stringa è *sempre* un numero intero, indipendentemente da quale sia il valore di `CONVFMT`. Dato il seguente frammento di codice:

```
CONVFMT = "%2.2f"
a = 12
b = a ""
```

**b** ha valore "12", non "12.00".



#### **awk prima di POSIX usava OFMT per la conversione di stringhe**

Prima dello standard POSIX, `awk` usava il valore di `OFMT` per convertire i numeri in stringhe. `OFMT` specifica il formato di output da usare per la stampa dei numeri con `print`. `CONVFMT` fu introdotto per separare la semantica della conversione dalla semantica della stampa. Sia `CONVFMT` che `OFMT` hanno lo stesso valore di default: `"%.6g"`. Nella stragrande maggioranza dei casi, i vecchi programmi di `awk` non cambiano questo comportamento. Si veda la [Sezione 5.1 \[L'istruzione `print`\], pagina 95](#), per maggiori informazioni sull'istruzione `print`.

### **6.1.4.2 Le localizzazioni possono influire sulle conversioni**

Il luogo dove si è può avere importanza quando si tratta di convertire numeri e stringhe. La lingua e i caratteri—la *localizzazione*—possono influire sui formati numerici. In particolare, per i programmi `awk`, influiscono sui caratteri separatore decimale e separatore delle migliaia. La localizzazione `"C"`, e la maggior parte delle localizzazioni inglesi, usano il punto (`'.'`) come separatore decimale e non prevedono un separatore delle migliaia. Tuttavia, molte (se non la maggior parte) delle localizzazioni europee e non inglesi usano la virgola (`'.'`) come separatore dei decimali. Le localizzazioni europee spesso usano o lo spazio o il punto come separatore delle migliaia, all'occorrenza.

Lo standard POSIX prevede che `awk` usi sempre il punto come separatore dei decimali nel codice sorgente del programma `awk`, e per gli assegnamenti di variabile da riga di comando (si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\], pagina 40](#)). Tuttavia, nell'interpretazione dei dati in input, per l'output di `print` e `printf`, e per la conversione da numeri a stringhe, viene usato il separatore decimale locale. In ogni caso, i numeri nel codice sorgente e nei dati di input non possono avere un separatore delle migliaia. Di seguito sono riportati alcuni esempi che illustrano la differenza di comportamento, su un sistema GNU/Linux:

```
$ export POSIXLY_CORRECT=1                                Forzare aderenza a stan-
dard POSIX
$ gawk 'BEGIN { printf "%g\n", 3.1415927 }'
+ 3.14159
$ LC_ALL=en_DK.utf-8 gawk 'BEGIN { printf "%g\n", 3.1415927 }'
+ 3,14159
$ echo 4,321 | gawk '{ print $1 + 1 }'
+ 5
```



```
$ echo 4,321 | LC_ALL=en_DK.utf-8 gawk '{ print $1 + 1 }'
+ 5,321
```

La localizzazione `en_DK.utf-8` è per l'inglese in Danimarca, dove le virgole fungono da separatore decimale. Nella localizzazione "C" normale, `gawk` tratta `'4,321'` come 4, mentre nella localizzazione danese è trattato come numero completo comprendente la parte frazionaria, 4.321.

Alcune delle prime versioni di `gawk` si conformavano completamente con quest'aspetto dello standard. Tuttavia, molti utenti di localizzazioni non inglesi si lamentavano di questo comportamento, perché i loro dati usavano il punto come separatore decimale, per cui fu ripristinato il comportamento di default che usava il punto come carattere di separazione decimale. Si può usare l'opzione `--use-lc-numeric` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)) per forzare `gawk` a usare il carattere separatore decimale della localizzazione. (`gawk` usa il separatore decimale della localizzazione anche quando è in modalità POSIX, o con l'opzione `--posix` o con la variabile d'ambiente `POSIXLY_CORRECT`, come appena visto.)

[Tabella 6.1](#) descrive i casi in cui si usa il separatore decimale locale e quando si usa il punto. Alcune di queste funzionalità non sono state ancora descritte.

Funzione	Default	<code>--posix</code> o <code>--use-lc-numeric</code>
<code>%'g</code>	Usa la localizzazione	Usa la localizzazione
<code>%g</code>	Usa il punto	Usa la localizzazione
Input	Usa il punto	Usa la localizzazione
<code>strtonum()</code>	Usa il punto	Usa la localizzazione

Tabella 6.1: Separatore decimale locale o punto

Infine, gli standard ufficiali correnti e la rappresentazione dei numeri in virgola mobile dello standard IEEE possono avere un effetto insolito ma importante sul modo in cui `gawk` converte alcuni valori di stringa speciali in numeri. I dettagli sono illustrati in [Sezione 15.6 \[Confronto tra standard e uso corrente\]](#), [pagina 391](#).

## 6.2 Operatori: fare qualcosa coi valori

Questa sezione introduce gli *operatori* che fanno uso dei valori forniti da costanti e variabili.

### 6.2.1 Operatori aritmetici

Il linguaggio `awk` usa i comuni operatori aritmetici nella valutazione delle espressioni. Tutti questi operatori aritmetici seguono le normali regole di precedenza e funzionano come ci si aspetta.

Il seguente esempio usa un file chiamato `grades`, che contiene una lista di nomi di studenti e anche i voti di tre verifiche per ogni studente (è una piccola classe):

```
Pat    100 97 58
Sandy  84 72 93
Chris  72 92 89
```

Questo programma prende il file `grades` e stampa la media dei voti:

```
$ awk '{ sum = $2 + $3 + $4 ; avg = sum / 3
```

```
>      print $1, avg }' grades
+ Pat 85
+ Sandy 83
+ Chris 84.3333
```

La lista seguente elenca gli operatori aritmetici in **awk**, in ordine di precedenza, da quella più alta a quella più bassa:

$x \wedge y$	
$x ** y$	Elevamento a potenza; $x$ elevato alla potenza $y$ . ‘ $2 \wedge 3$ ’ ha il valore otto; la sequenza di caratteri ‘ $**$ ’ è equivalente a ‘ $\wedge$ ’. (e.c.)
$- x$	Negazione.
$+ x$	Più unario; l’espressione è convertita in un numero.
$x * y$	Moltiplicazione.
$x / y$	Divisione; poiché tutti i numeri in <b>awk</b> sono numeri in virgola mobile, il risultato <i>non</i> è arrotondato all’intero—‘ $3 / 4$ ’ ha il valore di 0.75. (Un errore comune, specialmente tra i programmatori in C, è quello di dimenticare che <i>tutti</i> i numeri in <b>awk</b> sono in virgola mobile, e che la divisione di costanti rappresentate da numeri interi produce un numero reale, non un numero intero.)
$x \% y$	Resto della divisione; subito dopo questa lista, l’argomento viene ulteriormente dettagliato.
$x + y$	Addizione.
$x - y$	Sottrazione.

Il più e il meno unari hanno la stessa precedenza, gli operatori di moltiplicazione hanno tutti la stessa precedenza, e l’addizione e la sottrazione hanno la stessa precedenza.

Quando si calcola il resto di ‘ $x \% y$ ’, il quoziente è troncato all’intero e moltiplicato per  $y$ . Questo risultato è sottratto da  $x$ ; quest’operazione è nota anche come “modulo”. La seguente relazione è sempre verificata:

$$b * \text{int}(a / b) + (a \% b) == a$$

Un possibile effetto indesiderato di questa definizione di resto è che ‘ $x \% y$ ’ sia negativo se  $x$  è negativo. Così:

$$-17 \% 8 = -1$$

In altre implementazioni di **awk** il segno del resto può essere dipendente dalla macchina.

**NOTA:** Lo standard POSIX specifica solo l’uso di ‘ $\wedge$ ’ per l’elevamento a potenza.

Per garantire la massima portabilità è meglio non usare l’operatore ‘ $**$ ’.

## 6.2.2 Concatenazione di stringhe

*Allora ci era sembrata una buona idea.*

—Brian Kernighan

C’è una sola operazione di stringa: la concatenazione. Non ha un operatore specifico per rappresentarla. Piuttosto, la concatenazione è effettuata scrivendo le espressioni l’una vicino all’altra, senza alcun operatore. Per esempio:

```
$ awk '{ print "Campo numero uno: " $1 }' mail-list
```

```

-| Campo numero uno: Amelia
-| Campo numero uno: Anthony
...

```

Senza lo spazio nella costante stringa dopo `':'`, la riga rimane unita. Per esempio:

```

$ awk '{ print "Campo numero uno:" $1 }' mail-list
-| Campo numero uno:Amelia
-| Campo numero uno:Anthony
...

```

Poiché la concatenazione di stringhe non ha un operatore esplicito, è spesso necessario assicurarsi che venga effettuata al momento giusto usando le parentesi per racchiudere gli elementi da concatenare. Per esempio, ci si potrebbe aspettare che il seguente frammento di codice concateni `nome` e `file`:

```

nome = "nome"
file = "file"
print "qualcosa di significativo" > nome file

```

Questo produce un errore di sintassi in alcune versioni di `awk` per Unix.<sup>3</sup> È necessario usare la seguente sintassi:

```

print "qualcosa di significativo" > (nome file)

```

Si dovrebbero usare le parentesi attorno alle concatenazioni in tutti i contesti non comuni, come, per esempio, sul lato destro di `=`. Bisogna stare attenti al tipo di espressioni usate nella concatenazione di stringhe. In particolare, l'ordine di valutazione di espressioni usate per la concatenazione non è definita nel linguaggio `awk`. Si consideri quest'esempio:

```

BEGIN {
    a = "Non"
    print (a " " (a = "v'allarmate"))
}

```

Non è definito se il secondo assegnamento ad `a` debba avvenire prima o dopo il recupero del valore di `a` per produrre il valore concatenato. Il risultato potrebbe essere sia `'Non v'allarmate'`, sia `'v'allarmate v'allarmate'`.

La precedenza della concatenazione, quando è in combinazione con altri operatori, è spesso controintuitiva. Si consideri questo esempio:

```

$ awk 'BEGIN { print -12 " " -24 }'
-| -12-24

```

Quest'esempio, "ovviamente" concatena `-12`, uno spazio, e `-24`. Ma dov'è finito lo spazio? La risposta sta nella combinazione di precedenze di operatori e nelle regole di conversione automatica di `awk`. Per ottenere il risultato desiderato, si deve scrivere il programma in questo modo:

```

$ awk 'BEGIN { print -12 " " (-24) }'
-| -12 -24

```

Questo forza il trattamento, da parte di `awk`, del `'-'` del `'-24'` come operatore unario. Altrimenti è analizzato in questo modo:

```

-12 (" " - 24)

```

---

<sup>3</sup> Può capitare che `BWK awk`, `gawk` e `mawk` lo interpretino nel modo giusto, ma non ci si dovrebbe fare affidamento.

```

⇒ -12 (0 - 24)
⇒ -12 (-24)
⇒ -12-24

```

Come si è detto precedentemente, quando si usa la concatenazione insieme ad altri operatori, è necessario *usare le parentesi*. Altrimenti, non si può essere mai completamente certi di quel che si ottiene.

### 6.2.3 Espressioni di assegnamento

Un *assegnamento* è un'espressione che memorizza un valore (generalmente diverso da quello che la variabile aveva in precedenza) in una variabile. Per esempio, si assegna il valore uno alla variabile **z**:

```
z = 1
```

Dopo l'esecuzione di quest'espressione, la variabile **z** ha il valore uno. Qualsiasi precedente valore di **z** prima dell'assegnamento viene dimenticato.

Gli assegnamenti possono anche memorizzare valori di stringa. Il seguente esempio memorizza il valore "questo cibo è buono" nella variabile **messaggio**:

```

cosa = "cibo"
predicato = "buono"
messaggio = "questo " cosa " è " predicato

```

Quest'esempio illustra anche la concatenazione di stringhe. Il segno '=' è un *operatore di assegnamento*. È il più semplice fra gli operatori di assegnamento perché il valore dell'operando di destra è memorizzato invariato. La maggior parte degli operatori (addizione, concatenazione e così via) non fanno altro che calcolare un valore. Se il valore non viene poi utilizzato non c'è alcun motivo per usare l'operatore. Un operatore di assegnamento è differente; produce un valore; anche se poi non lo si usa, l'assegnamento svolge ancora una funzione alterando la variabile. Chiamiamo questo un *effetto collaterale*.

L'operando di sinistra non dev'essere necessariamente una variabile (si veda la [Sezione 6.1.3 \[Variabili\], pagina 119](#)); può essere anche un campo (si veda la [Sezione 4.4 \[Cambiare il contenuto di un campo\], pagina 69](#)) o un elemento di un vettore (si veda il [Capitolo 8 \[Vettori in awk\], pagina 177](#)). Questi operandi sono chiamati *lvalue*, il che significa che possono apparire sul lato sinistro di un operatore di assegnamento. L'operando sul lato destro può essere qualsiasi espressione; produce un nuovo valore che l'assegnamento memorizza nella variabile, nel campo o nell'elemento di vettore specificati. Tali valori sono chiamati *rvalue*.

È importante notare che le variabili *non* hanno dei tipi permanenti. Il tipo di una variabile è semplicemente quello di qualsiasi valore le sia stato assegnato per ultimo. Nel seguente frammento di programma, la variabile **pippo** ha dapprima un valore numerico, e in seguito un valore di stringa:

```

pippo = 1
print pippo
pippo = "pluto"
print pippo

```

Quando il secondo assegnamento dà a **pippo** un valore di stringa, il fatto che avesse precedentemente un valore numerico viene dimenticato.

Ai valori di stringa che non iniziano con una cifra viene assegnato il valore numerico zero. Dopo l'esecuzione del seguente codice, il valore di `pippo` è cinque:

```
pippo = "una stringa"
pippo = pippo + 5
```

**NOTA:** Usare una variabile sia come numero che come stringa può originare confusione e denota uno stile di programmazione scadente. I due esempi precedenti illustrano come funziona `awk`, *non* come si dovrebbero scrivere i programmi!

Un assegnamento è un'espressione, per cui ha un valore: lo stesso valore che le è stato assegnato. Così, `'z = 1'` è un'espressione col valore uno. Una conseguenza di ciò è che si possono scrivere più assegnamenti insieme, come:

```
x = y = z = 5
```

Quest'esempio memorizza il valore cinque in tutte e tre le variabili, (`x`, `y` e `z`). Questo perché il valore di `'z = 5'`, che è cinque, è memorizzato in `y` e poi il valore di `'y = z = 5'`, che è cinque, è memorizzato in `x`.

Gli assegnamenti possono essere usati ovunque sia prevista un'espressione. Per esempio, è valido scrivere `'x != (y = 1)'` per impostare `y` a uno, e poi verificare se `x` è uguale a uno. Però questo stile rende i programmi difficili da leggere; una tale nidificazione di assegnamenti dovrebbe essere evitata, eccetto forse in un programma `usa-e-getta`.

Accanto a `'='`, ci sono diversi altri operatori di assegnamento che eseguono calcoli col vecchio valore di una variabile. Per esempio, l'operatore `'+='` calcola un nuovo valore aggiungendo il valore sul lato destro al vecchio valore di una variabile. Così, il seguente assegnamento aggiunge cinque al valore di `pippo`:

```
pippo += 5
```

Questo è equivalente a:

```
pippo = pippo + 5
```

Si usi la notazione che rende più chiaro il significato del programma.

Ci sono situazioni in cui usare `'+='` (o qualunque operatore di assegnamento) *non* è la stessa cosa che ripetere semplicemente l'operando di sinistra nell'espressione di destra. Per esempio:

```
# Grazie a Pat Rankin per quest'esempio
BEGIN {
    pippo[rand()] += 5
    for (x in pippo)
        print x, pippo[x]

    pluto[rand()] = pluto[rand()] + 5
    for (x in pluto)
        print x, pluto[x]
}
```

È praticamente certo che gli indici di `pluto` siano differenti, perché `rand()` restituisce valori differenti ogni volta che viene chiamata. (I vettori e la funzione `rand()` non sono ancora stati trattati. si veda il [Capitolo 8 \[Vettori in awk\]](#), [pagina 177](#), e si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), [pagina 196](#), per maggiori informazioni.) Quest'esempio illustra un

fatto importante riguardo agli operatori di assegnamento: l'espressione di sinistra viene valutata *una volta sola*.

Dipende dall'implementazione stabilire quale espressione valutare per prima, se quella di sinistra o quella di destra. Si consideri quest'esempio:

```
i = 1
a[i += 2] = i + 1
```

Il valore di `a[3]` potrebbe essere sia due sia quattro.

La **Tabella 6.2** elenca gli operatori di assegnamento aritmetici. In ogni caso, l'operando di destra è un'espressione il cui valore è convertito in un numero.

Operatore	Effetto
<code>lvalue += incremento</code>	Aggiunge <i>incremento</i> al valore di <i>lvalue</i> .
<code>lvalue -= decremento</code>	Sottrae <i>decremento</i> dal valore di <i>lvalue</i> .
<code>lvalue *= coefficiente</code>	Moltiplica il valore di <i>lvalue</i> per <i>coefficiente</i> .
<code>lvalue /= divisore</code>	Divide il valore di <i>lvalue</i> per <i>divisore</i> .
<code>lvalue %= modulo</code>	Imposta <i>lvalue</i> al resto della sua divisione per <i>modulo</i> .
<code>lvalue ^= esponente</code>	Eleva <i>lvalue</i> alla potenza <i>esponente</i> .
<code>lvalue **= esponente</code>	Eleva <i>lvalue</i> alla potenza <i>esponente</i> . (e.c.)

Tabella 6.2: Operatori di assegnamento aritmetici

**NOTA:** Soltanto l'operatore `'^='` è definito da POSIX. Per avere la massima portabilità, non usare l'operatore `'**='`.

#### Ambiguità sintattiche tra `'/'=` e le espressioni regolari

C'è un'ambiguità sintattica tra l'operatore di assegnamento `/=` e le costanti *regexp* il cui primo carattere sia `'='`. Questo è più evidente in alcune versioni commerciali di **awk**. Per esempio:

```
$ awk /=/ /dev/null
error awk: syntax error at source line 1
error context is
error >>> /= <<<
error awk: bailing out at source line 1
```

Un espediente è:

```
awk '/[=]/' /dev/null
```

**gawk** non ha questo problema, e neppure lo hanno **BWK awk** e **mawk**.

### 6.2.4 Operatori di incremento e di decremento

Gli operatori di *incremento* e *decremento* incrementano o riducono il valore di una variabile di uno. Un operatore di assegnamento può fare la stessa cosa, per cui gli operatori di incremento non aggiungono funzionalità all'ingaggio **awk**; in ogni caso, sono delle convenienti abbreviazioni per operazioni molto comuni.

L'operatore per aggiungere uno è `'++'`. Può essere usato per incrementare una variabile prima o dopo aver stabilito il suo valore. Per *preincrementare* una variabile `v`, si scrive `'++v'`.

Questo aggiunge uno al valore di `v`; questo nuovo valore è anche il valore dell'espressione. (L'espressione di assegnamento '`v += 1`' è totalmente equivalente.) Scrivendo '`++`' dopo la variabile si specifica un *postincremento*. Questo incrementa il valore della variabile nello stesso modo; la differenza è che il valore dell'espressione d'incremento è il *vecchio* valore della variabile. Così, se `pippo` ha il valore quattro, l'espressione '`pippo++`' ha il valore quattro, ma cambia il valore di `pippo` in cinque. In altre parole, l'operatore restituisce il vecchio valore della variabile, ma con l'effetto collaterale di incrementarlo.

Il postincremento '`pippo++`' è quasi come scrivere '`(pippo += 1) - 1`'. Non è perfettamente equivalente perché tutti i numeri in `awk` sono in virgola mobile. In virgola mobile, '`pippo + 1 - 1`' non è necessariamente uguale a `pippo`, ma la differenza è molto piccola finché si ha a che fare con numeri relativamente piccoli (inferiori a  $10^{12}$ ).

I campi di un record e gli elementi di un vettore vengono incrementati proprio come le variabili. (Si deve usare '`$(i++)`' quando si deve fare un riferimento a un campo e incrementare una variabile allo stesso tempo. Le parentesi sono necessarie a causa della precedenza dell'operatore di riferimento '\$'.)

L'operatore di decremento '`--`' funziona proprio come '`++`', solo che sottrae uno anziché aggiungerlo. Come '`++`', si può usare prima di *lvalue* per predecrementare o dopo per postdecrementare. Quel che segue è un sommario delle espressioni di incremento e di decremento:

- `++lvalue` Incrementa *lvalue*, restituendo il nuovo valore come valore dell'espressione.
- `lvalue++` Incrementa *lvalue*, restituendo il *vecchio* valore di *lvalue* come valore dell'espressione.
- `--lvalue` Decrementa *lvalue*, restituendo il nuovo valore come valore dell'espressione. (Quest'espressione è come '`++lvalue`', ma invece di aggiungere, sottrae.)
- `lvalue--` Decrementa *lvalue*, restituendo il *vecchio* valore di *lvalue* come valore dell'espressione. (Quest'espressione è come '`lvalue++`', ma invece di aggiungere, sottrae.)

### Ordine di valutazione degli operatori

*Dottore, quando faccio così mi fa male!  
E allora non farlo!*  
—Groucho Marx

Che cosa succede con qualcosa come questo?

```
b = 6
print b += b++
```

O con qualcosa di più strano ancora?

```
b = 6
b += ++b + b++
print b
```

In altre parole, quando hanno effetto i vari effetti collaterali previsti dagli operatori col suffisso (`'b++'`)? Quando gli effetti collaterali si verificano è *definito dall'implementazione*. Per dirla diversamente, questo è compito di ogni specifica versione di `awk`. Il risultato del primo esempio può essere 12 o 13, e del secondo può essere 22 o 23.

In breve, è sconsigliato fare cose come questa e, in ogni caso, ogni cosa che possa incidere sulla portabilità. Si dovrebbero evitare cose come queste nei programmi.

## 6.3 Valori e condizioni di verità

In certi contesti, i valori delle espressioni servono anche come “valori di verità”; cioè, determinano quale sarà la direzione che il programma prenderà durante la sua esecuzione. Questa sezione descrive come `awk` definisce “vero” e “falso” e come questi valori sono confrontati.

### 6.3.1 Vero e falso in `awk`

Molti linguaggi di programmazione hanno una particolare rappresentazione per i concetti di “vero” e “falso.” Questi linguaggi usano normalmente le costanti speciali `true` e `false`, o forse i loro equivalenti maiuscoli. Però `awk` è differente. Prende in prestito un concetto molto semplice di vero e falso dal linguaggio C. In `awk`, ogni valore numerico diverso da zero *oppure* ogni valore di stringa non vuota è vero. Ogni altro valore (zero o la stringa nulla, `""`) è falso. Il seguente programma stampa ‘Uno strano valore di verità’ tre volte:

```
BEGIN {
    if (3.1415927)
        print "Uno strano valore di verità"
    if ("Ottanta e sette anni or sono")
        print "Uno strano valore di verità"
    if (j = 57)
        print "Uno strano valore di verità"
}
```

C'è una conseguenza sorprendente della regola “non zero o non nullo”: la costante di stringa `"0"` sta effettivamente per vero, perché è non nulla.



### 6.3.2 Tipi di variabile ed espressioni di confronto

*La Guida galattica è infallibile. È la realtà, spesso, a essere inesatta.*  
—Douglas Adams, Guida galattica per autostoppisti

Diversamente che in altri linguaggi di programmazione, le variabili di **awk** non hanno un tipo fisso. Possono essere sia un numero che una stringa, a seconda del valore loro assegnato. Vediamo ora come viene assegnato il tipo a una variabile, e come **awk** le confronta.

### 6.3.2.1 Tipo stringa rispetto a tipo numero

Per gli elementi scalari in **awk** (variabili, elementi di vettore e campi), il tipo degli stessi viene attribuito *dinamicamente*. Ciò significa che il tipo di un elemento può cambiare nel corso dell'esecuzione di un programma, da *untyped* (non ancora tipizzata), valore assunto prima che la variabile sia utilizzata,<sup>4</sup> a stringa oppure a numero, e in seguito da stringa a numero o da numero a stringa, nel prosieguo del programma. (**gawk** prevede anche degli scalari di tipo *regexp*, ma per ora possiamo ignorarli; si veda la [Sezione 6.1.2.2 \[Costanti regexp fortemente tipizzate\]](#), pagina 118.)

Non si può fare molto riguardo alle variabili di tipo *untyped*, oltre a constatare che ancora non è stato loro attribuito un tipo. Il seguente programma confronta la variabile **a** con i valori "" e 0; il confronto dà esito positivo se alla variabile **a** non è mai stato assegnato un valore. L'esempio usa la funzione predefinita **typeof()** (non ancora trattata; si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), pagina 223) per visualizzare il tipo della variabile **a**:

```
$ gawk 'BEGIN { print (a == "" && a == 0 ?
> "a non ha un tipo" : "a ha un tipo!") ; print typeof(a) }'
+ a non ha un tipo
+ unassigned
```

Una variabile scalare diviene di tipo numerico quando le viene assegnato un valore numerico, per mezzo di una costante numerica, o tramite un'altra variabile scalare di tipo numerico:

```
$ gawk 'BEGIN { a = 42 ; print typeof(a)
> b = a ; print typeof(b) }'
number
number
```

Analogamente, una variabile scalare diviene di tipo stringa quando le viene assegnato come valore una stringa, per mezzo di una costante stringa, o tramite un'altra variabile scalare di tipo stringa:

```
$ gawk 'BEGIN { a = "quarantadue" ; print typeof(a)
> b = a ; print typeof(b) }'
string
string
```

Fin qui, tutto semplice e chiaro. Cosa succede, però, quando **awk** deve trattare dati forniti dall'utente? Il primo caso da considerare è quello dei campi di dati in input. Quale dovrebbe essere l'output del seguente programma?

```
echo ciao | awk '{ printf("%s %s < 42\n", $1,
                        ($1 < 42 ? "è" : "non è")) }'
```

Poiché 'ciao' è un dato di tipo alfabetico, **awk** può solo effettuare un confronto di tipo stringa. Internamente, il numero 42 viene convertito in "42" e vengono confrontate le due stringhe di valore "ciao" e "42". Questo è il risultato:

<sup>4</sup> **gawk** chiama queste variabili *unassigned* (non ancora assegnate), come si vede dall'esempio che segue.

```
$ echo ciao | awk '{ printf("%s %s < 42\n", $1,
>                                ($1 < 42 ? "è" : "non è")) }'
+ ciao non è < 42
```

Tuttavia, cosa succede quando un dato utente *assomiglia* a un numero? Da un lato, in realtà, il dato in input è formato da alcuni caratteri, non da valori numerici in formato binario. Ma, d'altro lato, il dato sembra numerico, e `awk` dovrebbe davvero trattarlo come tale. E in effetti questo è ciò che avviene:

```
$ echo 37 | awk '{ printf("%s %s < 42\n", $1,
>                                ($1 < 42 ? "è" : "non è")) }'
+ 37 is < 42
```

Queste sono le regole seguite per determinare quando `awk` tratta dei dati in input come numeri, e quando li considera stringhe.

Lo standard POSIX usa il concetto di *stringa numerica*, per dei dati in input che appaiono essere numerici. Il '37' nell'esempio precedente è una stringa numerica. Quindi, qual è il tipo di una stringa numerica? Risposta: numerico.

Il tipo di una variabile è importante perché il tipo di due variabili determina il modo con cui le stesse vengono confrontate. La determinazione del tipo di variabile segue queste regole:

- Una costante numerica o il risultato di un'operazione numerica ha l'attributo *numeric*.
- Una costante di stringa o il risultato di un'operazione di stringa ha l'attributo *string*.
- Campi, input tramite `getline`, `FILENAME`, elementi di `ARGV`, elementi di `ENVIRON`, e gli elementi di un vettore creato da `match()`, `split()` e `patsplit()` che sono stringhe numeriche hanno l'attributo *strnum*.<sup>5</sup> Altrimenti, hanno l'attributo *string*. Anche le variabili non inizializzate hanno l'attributo *strnum*.
- Gli attributi si trasmettono attraverso gli assegnamenti ma non vengono cambiati da nessun uso.

L'ultima regola è particolarmente importante. Nel seguente programma, `a` è di tipo numerico, anche se viene usata in un secondo momento in un'operazione di stringa:

```
BEGIN {
    a = 12.345
    b = a " è un numero carino"
    print b
}
```

Quando si confrontano due operandi, può essere usata sia il confronto come stringa che il confronto numerico, a seconda degli attributi degli operandi, secondo questa matrice simmetrica:

	STRING	NUMERIC	STRNUM
STRING	string	string	string
NUMERIC	string	numeric	numeric
STRNUM	string	numeric	numeric

<sup>5</sup> Quindi, una stringa numerica POSIX e una variabile tipo *strnum* di `gawk` sono equivalenti.

L'idea di base è che l'input dell'utente che appare come numerico—e *solo* l'input dell'utente—dovrebbe essere trattato come numerico, anche se in realtà è un insieme di caratteri e quindi anche una stringa. Così, ad esempio, la costante di stringa "+3.14", quando appare nel codice sorgente di un programma, è una stringa—anche se sembra numerica—e non viene *mai* trattato come numero ai fini di un confronto.

In breve, quando un operando è una stringa “pura”, come una costante di stringa, viene effettuato un confronto di stringa. In caso contrario viene effettuato un confronto numerico. (La differenza principale tra un numero e uno *strnum* è che per gli *strnum* **gawk** conserva anche il valore originale della stringa che la variabile scalare aveva al momento in cui è stata letta.

Questo punto merita di essere ulteriormente ribadito: l'input che appare essere un numero è numerico. Tutto il resto dell'input è considerato essere una stringa.

Così, la stringa in input di sei caratteri '+3.14' riceve l'attributo *strnum*. Al contrario, la stringa di sei caratteri "+3.14" che compaia nel testo di un programma rimane una costante di stringa. I seguenti esempi stampano '1' quando il confronto fra due diverse costanti è vero, altrimenti stampano '0':

```
$ echo ' +3.14' | awk '{ print($0 == " +3.14") }'      Vero
+ 1
$ echo ' +3.14' | awk '{ print($0 == "+3.14") }'      Falso
+ 0
$ echo ' +3.14' | awk '{ print($0 == "3.14") }'      Falso
+ 0
$ echo ' +3.14' | awk '{ print($0 == 3.14) }'        Vero
+ 1
$ echo ' +3.14' | awk '{ print($1 == " +3.14") }'    Falso
+ 0
$ echo ' +3.14' | awk '{ print($1 == "+3.14") }'    Vero
+ 1
$ echo ' +3.14' | awk '{ print($1 == "3.14") }'    Falso
+ 0
$ echo ' +3.14' | awk '{ print($1 == 3.14) }'        Vero
+ 1
```

Per controllare il tipo di un campo in input (o di altro input immesso dall'utente, si può usare `typeof()`:

```
$ echo salve 37 | gawk '{ print typeof($1), typeof($2) }'
+ string strnum
```

### 6.3.2.2 Operatori di confronto

Le *espressioni di confronto* confrontano stringhe o numeri per metterli in relazione tra di loro, come ad esempio nella relazione di uguaglianza. Sono scritte usando *operatori di relazione*, che sono un superinsieme di quelli in C. Sono descritti nella [Tabella 6.3](#).

Espressione	Risultato
<code>x &lt; y</code>	Vero se <code>x</code> è minore di <code>y</code>
<code>x &lt;= y</code>	Vero se <code>x</code> è minore o uguale a <code>y</code>
<code>x &gt; y</code>	Vero se <code>x</code> è maggiore di <code>y</code>
<code>x &gt;= y</code>	Vero se <code>x</code> è maggiore o uguale a <code>y</code>
<code>x == y</code>	Vero se <code>x</code> è uguale a <code>y</code>
<code>x != y</code>	Vero se <code>x</code> è diverso da <code>y</code>
<code>x ~ y</code>	Vero se la stringa <code>x</code> corrisponde alla <i>regex</i> rappresentata da <code>y</code>
<code>x !~ y</code>	Vero se la stringa <code>x</code> non corrisponde alla <i>regex</i> rappresentata da <code>y</code>
<code>indice in vettore</code>	Vero se il vettore <i>vettore</i> ha un elemento con indice <i>indice</i>

Tabella 6.3: Operatori di relazione

Le espressioni di confronto valgono uno se sono vere e zero se false. Quando si confrontano operandi di tipi diversi, gli operandi numerici sono convertiti in stringhe usando il valore di `CONVFMT` (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121).

Il confronto tra stringhe avviene confrontando il primo carattere di ciascuna stringa, poi il secondo carattere, e così via. Quindi, "10" è minore di "9". Se vi sono due stringhe di cui una è il prefisso dell'altra, la stringa più corta è minore di quella più lunga. Così, "abc" è minore di "abcd".

È molto facile sbagliarsi scrivendo l'operatore '==' e omettendo uno dei due caratteri '='. Il risultato è sempre un codice **awk** valido, ma il programma non fa quel che si voleva:

```
if (a = b)    # oops! dovrebbe essere == b
    ...
else
    ...
```

A meno che `b` non sia zero o la stringa nulla, la parte `if` del test ha sempre successo. Poiché gli operatori sono molto simili, questo tipo di errore è molto difficile da individuare rileggendo il codice sorgente.

Il seguente elenco di espressioni illustra il tipo di confronti che **awk** effettua e il risultato di ciascun confronto:

```
1.5 <= 2.0
    Confronto numerico (vero)

"abc" >= "xyz"
    Confronto tra stringhe (falso)

1.5 != " +2"
    Confronto tra stringhe (vero)

"1e2" < "3"
    Confronto tra stringhe (vero)

a = 2; b = "2"
a == b    Confronto tra stringhe (vero)

a = 2; b = " +2"
a == b    Confronto tra stringhe (falso)
```

In quest'esempio:

```
$ echo 1e2 3 | awk '{ print ($1 < $2) ? "vero" : "falso" }'
```

└ falso

il risultato è 'falso' perché sia \$1 che \$2 sono immessi dall'utente. Sono stringhe numeriche—quindi hanno entrambe l'attributo *strnum*, che richiede un confronto di tipo numerico. Lo scopo delle regole di confronto e dell'uso di stringhe numeriche è quello di cercare di produrre il comportamento "meno inaspettato possibile", pur "facendo la cosa giusta".

I confronti di stringhe e i confronti di espressioni regolari sono molto diversi. Per esempio:

```
x == "att"
```

ha il valore uno, ossia è vero, se la variabile *x* è precisamente 'att'. Al contrario:

```
x ~ /att/
```

ha il valore uno se *x* contiene 'att', come "Oh, che matto che sono!".

L'operando di destra degli operatori '~' e '!~' può essere sia una costante *regex* (*/.../*) che un'espressione ordinaria. In quest'ultimo caso, il valore dell'espressione come stringa è usato come una *regex* dinamica (si veda la [Sezione 3.1 \[Uso di espressioni regolari\]](#), [pagina 49](#); e si veda la [Sezione 3.6 \[Usare regex dinamiche\]](#), [pagina 57](#)).

Un'espressione regolare costante tra due barre è di per sé anche un'espressione. */regex/* è un'abbreviazione per la seguente espressione di confronto:

```
$0 ~ /regex/
```

Una particolare posizione dove */pippo/* non è un'abbreviazione di '\$0 ~ /pippo/' è quella in cui è l'operando di destra di '~' o '!~'. Si veda la [Sezione 6.1.2 \[Usare espressioni regolari come costanti\]](#), [pagina 117](#), dove questo punto è trattato in maggiore dettaglio.

### 6.3.2.3 Confronto tra stringhe usando l'ordine di collazione locale

Lo standard POSIX diceva che il confronto di stringhe viene effettuato secondo l'*ordine di collazione* locale. Questo è l'ordine secondo il quale sono disposti i caratteri, come definito dalla localizzazione (per una trattazione più dettagliata, si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), [pagina 141](#)). Quest'ordine normalmente è molto diverso dal risultato ottenuto quando si esegue un confronto rigorosamente "carattere per carattere".<sup>6</sup>

Poiché questo comportamento differisce sensibilmente dalla pratica corrente, *gawk* lo implementava solo quando eseguito in modalità POSIX (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)). Quest'esempio illustra la differenza, in una localizzazione *en\_US.UTF-8*:

```
$ gawk 'BEGIN { printf("ABC < abc = %s\n",
>                      ("ABC" < "abc" ? "TRUE" : "FALSE")) }'
```

└ ABC < abc = TRUE

```
$ gawk --posix 'BEGIN { printf("ABC < abc = %s\n",
>                      ("ABC" < "abc" ? "TRUE" : "FALSE")) }'
```

└ ABC < abc = FALSE

<sup>6</sup> Tecnicamente, il confronto di stringhe dovrebbe funzionare come se le stringhe fossero confrontate usando la funzione *strcoll()* di C.

Fortunatamente, dal mese di agosto 2016, un confronto basato sull'ordine di collazione locale non è più richiesto per gli operatori `==` e `!=`.<sup>7</sup> Tuttavia, un confronto basato sull'ordine di collazione locale è ancora richiesto per gli operatori `<`, `<=`, `>` e `>=`. POSIX, quindi, raccomanda quanto segue:

Poiché l'operatore `==` controlla che se le stringhe sono identiche, e non se sono nell'ordine di collazione locale, le applicazioni che devono controllare se le stringhe sono nell'ordine di collazione locale possono usare:

```
a <= b && a >= b
```

A partire dalla versione 4.2, **gawk** continua a usare l'ordine di collazione locale per `<`, `<=`, `>` e `>=` solo se eseguito nella modalità POSIX.

### 6.3.3 Espressioni booleane

Un'espressione *booleana* è una combinazione di espressioni di confronto o espressioni di ricerca, che usa gli operatori booleani "or" (`||`), "and" (`&&`), e "not" (`!`), facendo uso di parentesi per controllare le nidificazioni. Il valore di verità dell'espressione booleana è calcolato calcolando i valori di verità delle espressioni componenti. Le espressioni booleane sono conosciute anche come *espressioni logiche*. I due termini sono equivalenti.

Le espressioni booleane possono essere usate in tutti i casi in cui è possibile usare espressioni di confronto e di ricerca di corrispondenze. Si possono usare nelle istruzioni `if`, `while`, `do` e `for` (si veda la [Sezione 7.4 \[Istruzioni di controllo nelle azioni\]](#), pagina 153). Hanno valori numerici (uno se vero, zero se falso) che entrano in gioco se il risultato di un'espressione booleana è memorizzato in una variabile o se è usato nei calcoli.

Inoltre, ogni espressione booleana è anche un modello di ricerca valido, così se ne può usare uno come modello di ricerca per controllare l'esecuzione di regole. Gli operatori booleani sono:

***booleano1 && booleano2***

Vero se *booleano1* e *booleano2* sono entrambi veri. Per esempio, la seguente istruzione stampa il record in input corrente se contiene sia `'edu'` che `'li'`:

```
if ($0 ~ /edu/ && $0 ~ /li/) print
```

La sottoespressione *booleano2* viene valutata solo se *booleano1* è vero. Questo può comportare una differenza laddove *booleano2* contenga espressioni che hanno effetti collaterali. Nel caso di `'$0 ~ /pippo/ && ($2 == pluto++)'`, la variabile `pluto` non viene incrementata se non c'è nessuna sottostringa `'pippo'` nel record.

***booleano1 || booleano2***

Vero se almeno uno tra *booleano1* e *booleano2* è vero. Per esempio, la seguente istruzione stampa tutti i record dell'input che contengono `'edu'` oppure `'li'`:

```
if ($0 ~ /edu/ || $0 ~ /li/) print
```

La sottoespressione *booleano2* viene valutata solo se *booleano1* è falso. Questo può comportare una differenza quando *booleano2* contiene espressioni che hanno effetti collaterali. (Perciò, questo confronto non individua mai i record che contengono sia `'edu'` che `'li'`: non appena `'edu'` viene trovato, l'intero confronto è concluso positivamente.)

<sup>7</sup> Si consulti il sito web [dell'Austin Group](#).

`! booleano`

Vero se *booleano* è falso. Per esempio, il seguente programma stampa `'nessuna home!'` nel caso insolito che la variabile d'ambiente `HOME` non sia stata definita:

```
BEGIN { if (! ("HOME" in ENVIRON))
        print "nessuna home!" }
```

(L'operatore `in` è descritto in [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\]](#), pagina 179.)

Gli operatori `&&` e `||` sono chiamati operatori di *cortocircuito* per il modo in cui funzionano. La valutazione dell'intera espressione è "cortocircuitata" se il risultato può già essere determinato prima di aver completato interamente la valutazione.

Le istruzioni che finiscono con `&&` o `||` si possono continuare semplicemente mettendo un ritorno a capo dopo di esse. Però non si può mettere un ritorno a capo *prima* di questi operatori senza usare la continuazione tramite la barra inversa (si veda la [Sezione 1.6 \[Istruzioni e righe in awk\]](#), pagina 28).

Il valore reale di un'espressione che usa l'operatore `!` è uno o zero, a seconda del valore di verità dell'espressione a cui è applicato. L'operatore `!` spesso è utile per cambiare il senso di una variabile indicatore [*flag*] da falso a vero e viceversa. Per esempio, il seguente programma è un modo per stampare righe poste tra due speciali righe delimitatrici:

```
$1 == "START" { pertinente = ! pertinente; next }
pertinente   { print }
$1 == "END"   { pertinente = ! pertinente; next }
```

La variabile `pertinente`, così come tutte le variabili di `awk`, è inizializzata a zero, che vale anche "falso". Quando viene trovata una riga il cui primo campo è `'START'`, il valore di `pertinente` viene commutato a vero, usando `!`. La regola nella riga seguente stampa righe finché `pertinente` resta vero. Quando viene trovata una riga il cui primo campo è `'END'`, `pertinente` viene nuovamente commutata a falso.<sup>8</sup>

Più comunemente, l'operatore `!` viene usato nelle condizioni delle istruzioni `if` e `while`, dove ha più senso formulare espressioni logiche in negativo:

```
if (! qualche condizione || qualche altra condizione) {
    ... fai un'operazione a piacere ...
}
```

**NOTA:** L'istruzione `next` viene trattata in [Sezione 7.4.8 \[L'istruzione next\]](#), pagina 159. `next` dice ad `awk` di tralasciare il resto delle regole, leggere il record successivo, e iniziare a elaborare le regole partendo nuovamente dalla prima. Il motivo è quello di evitare di stampare le righe delimitatrici `'START'` e `'END'`.

### 6.3.4 Espressioni condizionali

Un'espressione *condizionale* è un tipo particolare di espressione che ha tre operandi. Consente di usare il primo valore dell'espressione per scegliere una o l'altra delle due ulteriori espressioni. L'espressione condizionale in `awk` è la stessa di quella del linguaggio C, ovvero:

```
selettore ? espr-se-vero : espr-se-falso
```

<sup>8</sup> Questo programma ha un bug; stampa righe che iniziano con `'END'`. Come si può risolvere?

Ci sono tre sottoespressioni. La prima, *selettore*, viene sempre calcolata per prima. Se è "vera" (non zero o non nulla), viene poi calcolata *espr-se-vero* e il suo valore diventa il valore dell'intera espressione. Altrimenti, la seconda espressione che viene calcolata è *espr-se-falso* e il suo valore diventa il valore dell'intera espressione. Per esempio, la seguente espressione produce il valore assoluto di *x*:

```
x >= 0 ? x : -x
```

Ogni volta che viene calcolata un'espressione condizionale, solo una delle espressioni *espr-se-vero* e *espr-se-falso* viene usata; l'altra è ignorata. Questo è importante quando le espressioni hanno effetti collaterali. Per esempio, quest'espressione condizionale esamina l'elemento *i* del vettore *a* o del vettore *b*, e incrementa *i*:

```
x == y ? a[i++] : b[i++]
```

Questa istruzione garantisce che *i* sia incrementato una volta sola per ogni esecuzione dell'istruzione stessa, perché ogni volta viene eseguita solo una delle due espressioni di incremento, mentre l'altra viene ignorata. si veda il [Capitolo 8 \[Vettori in awk\]](#), [pagina 177](#), per maggiori informazioni sui vettori.

Come estensione minore di *gawk*, un'istruzione che usa `'?:'` si può continuare mettendo semplicemente un ritorno a capo dopo i due caratteri. Tuttavia, se si mette un ritorno a capo *prima* dei due caratteri, la continuazione non funziona, se non si aggiunge anche la barra inversa (si veda la [Sezione 1.6 \[Istruzioni e righe in awk\]](#), [pagina 28](#)). Se viene specificata l'opzione `--posix` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), quest'estensione è disabilitata.

## 6.4 Chiamate di funzione

Una *funzione* è un nome per richiedere un particolare calcolo. Il nome permette di richiamare la funzione da qualsiasi punto del programma. Per esempio, la funzione `sqrt()` calcola la radice quadrata di un numero.

Un certo numero di funzioni sono *predefinite*, ossia sono disponibili in ogni programma *awk*. La funzione `sqrt()` è una di queste. Si veda la [Sezione 9.1 \[Funzioni predefinite\]](#), [pagina 195](#), per un elenco di funzioni predefinite e per le loro rispettive descrizioni. In aggiunta, l'utente può definire delle funzioni da usare nel proprio programma. Si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), [pagina 224](#), per istruzioni su come farlo. Infine, *gawk* permette di scrivere funzioni in C o in C++ che possono essere chiamate dal proprio programma (si veda la [Capitolo 16 \[Scrivere estensioni per gawk\]](#), [pagina 395](#)).

Una funzione viene utilizzata invocandola tramite un'espressione di *chiamata di funzione*, che consiste nel nome della funzione seguito immediatamente da una lista di *argomenti* tra parentesi. Gli argomenti sono espressioni che forniscono i materiali grezzi su cui opera la funzione. Quando ci sono più argomenti, questi sono separati da virgole. Se non ci sono argomenti, basta scrivere `'()'` dopo il nome della funzione. Gli esempi che seguono mostrano chiamate di funzione con e senza argomenti:

<code>sqrt(x^2 + y^2)</code>	<i>un argomento</i>
<code>atan2(y, x)</code>	<i>due argomenti</i>
<code>rand()</code>	<i>nessun argomento</i>

**ATTENZIONE:** Non ci dev'essere nessuno spazio tra il nome della funzione e la parentesi aperta! Un nome di funzione definita dall'utente può essere scambiata

per il nome di una variabile: uno spazio renderebbe l'espressione simile alla concatenazione di una variabile con un'espressione racchiusa tra parentesi. Per le funzioni predefinite, lo spazio prima delle parentesi non crea problemi, ma è meglio non prendere l'abitudine di usare spazi per evitare errori con le funzioni definite dall'utente.

Ogni funzione richiede uno specifico numero di argomenti. Per esempio, la funzione `sqrt()` dev'essere chiamata con un solo argomento, il numero del quale si vuole estrarre la radice quadrata:

```
sqrt(argomento)
```

Alcune delle funzioni predefinite hanno uno o più argomenti opzionali. Se questi argomenti non vengono forniti, le funzioni usano un valore di default appropriato. Si veda la [Sezione 9.1 \[Funzioni predefinite\], pagina 195](#), per tutti i dettagli. Se sono omessi argomenti in chiamate a funzioni definite dall'utente, tali argomenti sono considerati essere variabili locali. Il valore di tali variabili locali è la stringa nulla se usate in un contesto che richiede una stringa di caratteri, e lo zero se è richiesto un valore numerico (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\], pagina 224](#)).

Come funzionalità avanzata, `gawk` prevede la possibilità di effettuare chiamate di funzione indirette, il che permette di scegliere la funzione da chiamare al momento dell'esecuzione, invece che nel momento in cui si scrive il codice sorgente del programma. Si rimanda la trattazione di questa funzionalità a un secondo momento; si veda [Sezione 9.3 \[Chiamate indirette di funzione\], pagina 234](#).

Come ogni altra espressione, la chiamata di funzione ha un valore, chiamato spesso *valore di ritorno*, che è calcolato dalla funzione in base agli argomenti dati. In quest'esempio, il valore di ritorno di `'sqrt(argomento)'` è la radice quadrata di *argomento*. Il seguente programma legge numeri, un numero per riga, e stampa la radice quadrata di ciascuno:

```
$ awk '{ print "La radice quadrata di", $1, "è", sqrt($1) }'
1
→ La radice quadrata di 1 è 1
3
→ La radice quadrata di 3 è 1.73205
5
→ La radice quadrata di 5 è 2.23607
Ctrl-d
```

Una funzione può avere anche effetti collaterali, come assegnare valori a certe variabili o effettuare operazioni di I/O. Questo programma mostra come la funzione `match()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)) cambia le variabili `RSTART` e `RLENGTH`:

```
{
    if (match($1, $2))
        print RSTART, RLENGTH
    else
        print "non uguali"
}
```

Qui vediamo un'esecuzione di esempio:

```
$ awk -f matchit.awk
```

```

aacdd c+
+ 3 2
pippo      pluto
+ non uguali
abcdefg e
+ 5 1

```

## 6.5 Precedenza degli operatori (Come si nidificano gli operatori)

La *precedenza degli operatori* determina come gli operatori vengono raggruppati quando diversi operatori appaiono uno vicino all'altro in un'espressione. Per esempio, '\*' ha una precedenza più alta di '+'; così, 'a + b \* c' moltiplica b e c, e poi aggiunge a al prodotto (ovvero esegue 'a + (b \* c)').

La normale precedenza degli operatori può essere cambiata usando delle parentesi. Si possono vedere le regole di precedenza come un modo per indicare dove si sarebbero dovute mettere delle parentesi. Di fatto, è cosa saggia usare sempre le parentesi in presenza di una combinazione di operatori insolita, perché altre persone che leggono il programma potrebbero non ricordare quale sia la precedenza in quel particolare caso. Anche dei programmatori esperti a volte dimenticano le regole esatte, il che porta a commettere errori. Usare esplicitamente le parentesi previene qualunque errore di questo tipo.

Quando vengono usati insieme operatori con uguale precedenza, quello più a sinistra viene raggruppato per primo, ad eccezione degli operatori di assegnamento, condizionali e di elevamento a potenza, che vengono raggruppati nell'ordine opposto. Quindi, 'a - b + c' raggruppa come '(a - b) + c' e 'a = b = c' raggruppa come 'a = (b = c)'.

Normalmente la precedenza degli operatori unari di prefisso non ha importanza, perché c'è un solo modo di interpretarli: prima il più interno. Quindi, '\$++i' significa '\$(++i)' e '++\$x' significa '++(\$x)'. Tuttavia, quando un altro operatore segue l'operando, la precedenza degli operatori unari può avere importanza. '\$x^2' significa '(\$x)^2', ma '-x^2' significa '-(x^2)', perché '-' ha precedenza più bassa rispetto a '^', mentre '\$' ha precedenza più alta. Inoltre, gli operatori non possono essere combinati in modo tale da violare le regole di precedenza; per esempio, '\$\$0+--' non è un'espressione valida perché il primo '\$' ha precedenza più alta di '++'; per evitare il problema l'espressione può essere scritta come '\$(\$0++)--'.

Questa lista illustra gli operatori di **awk**, in ordine di precedenza dalla più alta alla più bassa:

(...)	Raggruppamento.
\$	Riferimento a un campo.
++ --	Incremento, decremento.
^ **	Elevamento a potenza. Questi operatori sono raggruppati da destra verso sinistra.
+ - !	Più, meno, "not" logico, unari.
* / %	Moltiplicazione, divisione, resto di una divisione.

+ - Addizione, sottrazione.

Concatenazione di stringhe

Non c'è un simbolo speciale per la concatenazione. Gli operandi sono semplicemente scritti uno accanto all'altro. (si veda la [Sezione 6.2.2 \[Concatenazione di stringhe\], pagina 124](#)).

< <= == != > >= >> | |&

Operatori relazionali e ridirezione. Gli operatori relazionali e le ridirezioni hanno lo stesso livello di precedenza. I caratteri come '>' servono sia come operatori relazionali che come ridirezioni; la distinzione tra i due significati dipende dal contesto.

Si noti che gli operatori di ridirezione I/O nelle istruzioni `print` e `printf` appartengono al livello dell'istruzione, non alle espressioni. La ridirezione non produce un'espressione che potrebbe essere l'operando di un altro operatore. Di conseguenza, non ha senso usare un operatore di ridirezione vicino a un altro operatore con precedenza più bassa senza parentesi. Tali combinazioni generano errori di sintassi (p.es., '`print pippo > a ? b : c`'). Il modo corretto di scrivere quest'istruzione è '`print pippo > (a ? b : c)`'.

~ !~ Corrispondenza, non corrispondenza.

in Appartenenza a un vettore.

&& "and" logico.

|| "or" logico.

?: Operatore condizionale. Questo operatore raggruppa da destra verso sinistra.

= += -= \*= /= %= ^= \*\*=

Assegnamento. Questi operatori raggruppano da destra verso sinistra.

**NOTA:** Gli operatori '|&', '\*\*' e '\*\*=' non sono definiti da POSIX. Per la massima portabilità, è meglio non usarli.

## 6.6 Il luogo fa la differenza

I moderni sistemi prevedono la nozione di *localizzazione*: un modo per informare il sistema sulla serie di caratteri e sulla lingua locali. Lo standard ISO C definisce una localizzazione di default "C", che è l'ambiente tipico a cui molti programmatori in C sono abituati.

Un tempo, le impostazioni della localizzazione avevano influenza sulla ricerca di corrispondenze tramite *regex*, ma ora non è più così (si veda la [Sezione A.8 \[Intervalli \*regex\* e localizzazione: una lunga e triste storia\], pagina 474](#)).

La localizzazione può influire sulla separazione dei record. Per il caso normale di 'RS = "\n"', la localizzazione è generalmente irrilevante. Per altri separatori di record di un solo carattere, impostare la variabile d'ambiente 'LC\_ALL=C' garantisce una migliore efficienza nella lettura dei record. Altrimenti, *gawk* dovrebbe fare diverse chiamate di funzione, *per ogni carattere in input*, per determinare la fine del record.

La localizzazione può influire sulla formattazione delle date e delle ore (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\], pagina 214](#)). Per esempio, un

modo comune per abbreviare la data 4 settembre 2015, negli Stati Uniti è “9/4/15.” In molti paesi dell’Europa, invece, l’abbreviazione è “4.9.15”. Quindi, la specifica di formato `%x` in una localizzazione “US” potrebbe produrre ‘9/4/15’, mentre in una localizzazione “EUROPA”, potrebbe produrre ‘4.9.15’.

Secondo POSIX, anche il confronto tra stringhe è influenzato dalla localizzazione (come nelle espressioni regolari). I dettagli sono descritti in [Sezione 6.3.2.3 \[Confronto tra stringhe usando l’ordine di collazione locale\]](#), pagina 135.

Infine, la localizzazione influenza il valore del separatore decimale usato quando `gawk` analizza i dati in input. Questo è stato trattato nel dettaglio in [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121.

## 6.7 Sommario

- Le espressioni sono gli elementi base dei calcoli eseguiti nei programmi. Sono costruite a partire da costanti, variabili, chiamate di funzione e dalla combinazione di vari tipi di valori tramite operatori.
- `awk` fornisce tre tipi di costanti: numerica, di stringa e di *regex*. Le costanti numeriche in `gawk` si possono specificare nei sistemi ottale ed esadecimale (con base 8 e 16), e anche nel sistema decimale (base 10). In alcuni contesti, una costante *regex* isolata come `/pippo/` ha lo stesso significato di `$0 ~ /pippo/`.
- Le variabili contengono valori che possono essere usati diverse volte nei calcoli. Un certo numero di variabili predefinite forniscono informazioni al programma `awk`, mentre altre permettono il controllo del comportamento di `awk`.
- I numeri sono automaticamente convertiti in stringhe, e le stringhe in numeri, a seconda delle necessità di `awk`. I valori numerici sono convertiti come se fossero formattati con `sprintf()` usando il formato contenuto in `CONVFMT`. La localizzazione può influire sulle conversioni.
- In `awk` ci sono gli operatori aritmetici di uso comune (addizione, sottrazione, moltiplicazione, divisione, modulo), e il più e il meno unari. Ci sono anche operatori di confronto, operatori booleani, una verifica dell’esistenza di una chiave in un vettore, e operatori per la ricerca di corrispondenze con espressioni regolari. La concatenazione di stringhe è effettuata mettendo due espressioni una vicino all’altra; non c’è nessun operatore esplicito. L’operatore con tre operandi `?:` fornisce una verifica “if-else” all’interno delle espressioni.
- Gli operatori di assegnamento forniscono delle comode forme brevi per le comuni operazioni aritmetiche.
- In `awk`, un valore è considerato vero se è diverso da zero *oppure* non nullo. Altrimenti, il valore è falso.
- Il tipo di una variabile viene impostata a ogni assegnamento e può cambiare durante il suo ciclo di vita. Il tipo determina il comportamento della variabile nei confronti (di tipo stringa o numerici).
- Le chiamate di funzione restituiscono un valore che può essere usato come parte di un’espressione più lunga. Le espressioni usate per passare valori di parametro vengono valutate completamente prima di chiamare la funzione. `awk` fornisce funzioni predefi-

nite e prevede quelle definite dall'utente; questo è descritto in [Capitolo 9 \[Funzioni\]](#), [pagina 195](#).

- La precedenza degli operatori specifica l'ordine secondo il quale vengono effettuate le operazioni, a meno che quest'ordine non sia esplicitamente alterato tramite parentesi. Le regole di precedenza degli operatori di `awk` sono compatibili con quelle del linguaggio C.
- La localizzazione può influire sul formato dei dati in uscita da un programma `awk`, e occasionalmente sul formato dei dati letti in input.



## 7 Criteri di ricerca, azioni e variabili

Come già visto, ogni istruzione **awk** consiste di un criterio di ricerca [*pattern*] a cui è associata un'azione. Questo capitolo descrive come specificare criteri e azioni, cosa è possibile fare tramite le azioni e quali sono le variabili predefinite in **awk**.

Le regole *criterio di ricerca-azione* e le istruzioni che si possono dare all'interno delle azioni formano il nucleo centrale dei programmi scritti in **awk**. In un certo senso, tutto quanto si è visto finora costituisce le fondamenta sulle quali sono costruiti i programmi. È giunto il momento di iniziare a costruire qualcosa di utile.

### 7.1 Elementi di un criterio di ricerca

I criteri di ricerca in **awk** controllano l'esecuzione di azioni: un'azione viene eseguita quando il criterio di ricerca associato ad essa è soddisfatto dal record in input corrente. La tabella seguente è un sommario dei tipi di criteri di ricerca in **awk**:

*/espressione regolare/*

Un'espressione regolare. È verificata quando il testo di un record in input corrisponde all'espressione regolare. (Si veda il [Capitolo 3 \[Espressioni regolari\]](#), [pagina 49](#).)

*espressione*

Una singola espressione. È verificata quando il suo valore è diverso da zero (se di tipo numerico) o non nullo (se è una stringa). (Si veda la [Sezione 7.1.2 \[Espressioni come criteri di ricerca\]](#), [pagina 146](#).)

*inizio\_interv, fine\_interv*

Una coppia di criteri di ricerca separati da una virgola, che specificano un *intervallo* di record. L'intervallo comprende sia il record iniziale che corrisponde a *inizio\_interv* sia il record finale che corrisponde a *fine\_interv*. (Si veda la [Sezione 7.1.3 \[Specificare intervalli di record con i criteri di ricerca\]](#), [pagina 147](#).)

**BEGIN**

**END** Criteri di ricerca speciali che consentono azioni di inizializzazione o di pulizia in un programma **awk**. (Si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), [pagina 148](#).)

**BEGINFILE**

**ENDFILE** Criteri di ricerca speciali che consentono azioni di inizializzazione o di pulizia da eseguire all'inizio o alla fine di ogni file in input. (Si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), [pagina 150](#).)

*vuoto*

Il criterio di ricerca vuoto corrisponde a ciascun record in input. (Si veda la [Sezione 7.1.6 \[Il criterio di ricerca vuoto\]](#), [pagina 151](#).)

#### 7.1.1 Espressioni regolari come criteri di ricerca

Le espressioni regolari sono uno dei primi tipi di criteri di ricerca presentati in questo libro. Questo tipo di criterio di ricerca è semplicemente una costante *regexp* posta nella parte *criterio di ricerca* di una regola. Equivale a scrivere '\$0 ~ /*criterio di ricerca*/'. Il criterio di ricerca è verificato quando il record in input corrisponde alla *regexp*. Per esempio:

```
/pippo|pluto|paperino/ { personaggi_Disney++ }
```

```
END                { print personaggi_Disney, "Personaggi Disney visti" }
```

### 7.1.2 Espressioni come criteri di ricerca

Qualsiasi espressione `awk` può essere usata come un criterio di ricerca `awk`. Il criterio di ricerca è verificato se il valore dell'espressione è diverso da zero (se è un numero), o non nullo (se è una stringa). L'espressione è ricalcolata ogni volta che la regola viene applicata a un nuovo record in input. Se l'espressione usa campi come `$1`, il suo valore dipende direttamente dal contenuto del record in input appena letto; altrimenti, dipende solo da quel che è accaduto fino a quel momento durante l'esecuzione del programma `awk`.

Le espressioni di confronto, che usano gli operatori di confronto descritti in [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), [pagina 130](#), sono un tipo di criterio di ricerca usato frequentemente. Anche le *regex*, verificate o non verificate, sono tipi di espressioni molto frequenti. L'operando a sinistra degli operatori `'~'` e `'!~'` è una stringa. L'operando di destra è un'espressione regolare costante delimitata da barre (*/regex/*) o qualsiasi espressione il cui valore come stringa è usato come un'espressione regolare dinamica (si veda la [Sezione 3.6 \[Usare regex dinamiche\]](#), [pagina 57](#)). L'esempio seguente stampa il secondo campo di ogni record in input il cui primo campo sia esattamente `'li'`:

```
$ awk '$1 == "li" { print $2 }' mail-list
```

(Il programma non stampa alcun output, perché nessuna persona ha come nome esattamente `'li'`.) Si veda la differenza con il seguente confronto di espressione regolare, che individua invece qualsiasi record il cui primo campo *contenga* `'li'`:

```
$ awk '$1 ~ /li/ { print $2 }' mail-list
+ 555-5553
+ 555-6699
```

Una costante *regex* usata come criterio di ricerca è anche un caso speciale di criterio di ricerca costituito da un'espressione. All'espressione `/li/` viene assegnato il valore uno se `'li'` viene trovato nel record in input corrente. Quindi, come criterio di ricerca, `/li/` individua tutti i record che contengono la stringa `'li'`.

Anche le espressioni booleane sono frequentemente usate come criteri di ricerca. Se un criterio di ricerca individua o no un record in input dipende dalla verifica delle sottoespressioni da cui è composto. Per esempio, il seguente comando stampa tutti i record in `mail-list` che contengono sia `'edu'` che `'li'`:

```
$ awk '/edu/ && /li/' mail-list
+ Samuel      555-3430      samuel.lanceolis@shu.edu      A
```

Il seguente comando stampa tutti i record in `mail-list` che contengono `'edu'` oppure `'li'` (o entrambi, naturalmente):

```
$ awk '/edu/ || /li/' mail-list
+ Amelia      555-5553      amelia.zodiacusque@gmail.com  F
+ Broderick   555-0542      broderick.aliquotiens@yahoo.com R
+ Fabius      555-1234      fabius.undevicesimus@ucb.edu  F
+ Julie       555-6699      julie.perscrutabor@skeeve.com F
+ Samuel      555-3430      samuel.lanceolis@shu.edu      A
+ Jean-Paul   555-2127      jeanpaul.campanorum@nyu.edu    R
```

Il seguente comando stampa tutti i record in `mail-list` che *non* contengono la stringa `'li'`:

```
$ awk '! /li/' mail-list
+ Anthony      555-3412      anthony.asserturo@hotmail.com    A
+ Becky        555-7685      becky.algebrarum@gmail.com       A
+ Bill         555-1675      bill.drowning@hotmail.com        A
+ Camilla      555-2912      camilla.infusarum@skynet.be      R
+ Fabius       555-1234      fabius.undevicesimus@ucb.edu     F
+ Martin       555-6480      martin.codicibus@hotmail.com     A
+ Jean-Paul    555-2127      jeanpaul.campanorum@nyu.edu      R
```

Le sottoespressioni di un operatore booleano in un criterio di ricerca possono essere espressioni regolari costanti, confronti, o qualsiasi altra espressione di `awk`. Gli intervalli di ricerca non sono espressioni, e quindi non possono apparire all'interno di criteri di ricerca booleani. Analogamente, i criteri di ricerca speciali `BEGIN`, `END`, `BEGINFILE` ed `ENDFILE`, che non corrispondono ad alcun record in input, non sono espressioni e non possono essere usati all'interno di criteri di ricerca booleani.

L'ordine di precedenza dei differenti operatori che possono essere usati nei criteri di ricerca è descritto in [Sezione 6.5 \[Precedenza degli operatori \(Come si nidificano gli operatori\)\]](#), [pagina 140](#).

### 7.1.3 Specificare intervalli di record con i criteri di ricerca

Un *intervallo di ricerca* è composto da due criteri di ricerca separati da una virgola, nella forma '*inizio\_intervallo*, *fine\_intervallo*'. È usato per individuare una serie di record consecutivi nei file in input. Il primo criterio di ricerca, *inizio\_intervallo*, controlla dove inizia la serie di record, mentre *fine\_intervallo* controlla dove finisce la serie stessa. L'esempio seguente:

```
awk '$1 == "on", $1 == "off"' miofile
```

stampa ogni record in `miofile` incluso tra i due record che iniziano con '`on`'/'`off`', estremi compresi.

Un intervallo di ricerca inizia con la valutazione di *inizio\_intervallo* per ogni record in input. Quando un record soddisfa la condizione *inizio\_intervallo*, l'intervallo di ricerca è *attivato* e l'intervallo di ricerca include anche quel record. Finché l'intervallo di ricerca rimane attivo, automaticamente vengono trovate corrispondenze in ogni record in input letto. L'intervallo di ricerca verifica anche *fine\_intervallo* per ogni record in input; quando la verifica ha successo, il criterio di ricerca viene *disattivato* a partire dal record seguente. Quindi il criterio di ricerca torna a controllare *inizio\_intervallo* per ogni nuovo record.

Il record che segnala l'inizio dell'intervallo di ricerca e quello che segnala la fine di quell'intervallo soddisfano *entrambi* il criterio di ricerca. Se non si vuole agire su tali record si possono scrivere istruzioni `if` nella parte *azione* della regola per distinguerli dai record che il programma è interessato a trattare.

È possibile che un criterio di ricerca sia attivato e disattivato dallo stesso record. Se il record soddisfa entrambe le condizioni, l'azione è eseguita solo su quel record. Per esempio, si supponga che ci sia un testo tra due separatori identici (p.es., il simbolo '%'), ognuno dei quali sia su una riga a parte, che dovrebbero essere ignorati. Un primo tentativo potrebbe essere quello di combinare un intervallo di ricerca che descrive il testo delimitato con l'istruzione `next` (non ancora introdotta, si veda la [Sezione 7.4.8 \[L'istruzione next\]](#),

pagina 159). Con quest'istruzione `awk` non effettua alcuna azione sul record corrente e inizia di nuovo a elaborare il successivo record in input. Un tale programma è simile a questo:

```
/~%$/~/~%$/ { next }
{ print }
```

Questo programma non funziona, perché l'intervallo di ricerca è sia attivato che disattivato dalla prima riga incontrata, quella costituita da un `'%'`. Per ottenere l'effetto desiderato, si scriva il programma nella maniera che segue, utilizzando un *flag*:

```
/~%$/ { ignora = ! ignora; next }
ignora == 1 { next } # ignora righe quando 'ignora' è impostato a 1
```

In un intervallo di ricerca, la virgola (`,`) ha la precedenza più bassa tra tutti gli operatori (cioè, è l'ultima a essere valutata). Il programma seguente tenta di combinare un intervallo di ricerca con un altro controllo, più semplice:

```
echo Yes | awk '/1/,/2/ || /Yes/'
```

L'intenzione in questo programma è quello di esprimere le condizioni `'(/1/,/2/) || /Yes/'`. Tuttavia, `awk` lo interpreta come se fosse `'/1/, (/2/ || /Yes/)'`. Questo comportamento non può essere cambiato o evitato; gli intervalli di ricerca non si possono combinare con altri criteri di ricerca:

```
$ echo Yes | gawk '/1/,/2/) || /Yes/'
[error] gawk: riga com.:1: (/1/,/2/) || /Yes/
[error] gawk: riga com.:1: ^ syntax error
```



Come punto di secondaria importanza, nonostante sia stilisticamente poco elegante, lo standard POSIX consente di andare a capo dopo la virgola in un intervallo di ricerca.

### 7.1.4 I criteri di ricerca speciali BEGIN ed END

Tutti i criteri di ricerca fin qui descritti servono a individuare dei record in input. I criteri di ricerca speciali **BEGIN** ed **END** non hanno questo scopo. Servono invece per effettuare azioni di inizializzazione o di pulizia nei programmi `awk`. Le regole **BEGIN** ed **END** devono prevedere azioni; non c'è un'azione di default per queste regole, perché non c'è un record corrente quando sono invocate. Le regole **BEGIN** ed **END** sono spesso chiamate "blocchi **BEGIN** ed **END**" da programmatori che usano `awk` da molto tempo.

#### 7.1.4.1 Azioni di inizializzazione e pulizia

Una regola **BEGIN** è eseguita solo una volta, prima che sia letto il primo record in input. Analogamente, una regola **END** è eseguita solo una volta, dopo che tutto l'input è già stato letto. Per esempio:

```
$ awk '
> BEGIN { print "Analisi di \"li\"" }
> /li/ { ++n }
> END { print "\"li\" è presente in", n, "record." }' mail-list
+ Analisi di "li"
+ "li" è presente in 4 record.
```

Questo programma trova il numero di record nel file in input `mail-list` che contengono la stringa `'li'`. La regola **BEGIN** stampa un titolo per il rapporto. Non c'è bisogno di usare la regola **BEGIN** per inizializzare il contatore `n` a zero, poiché `awk` lo fa automaticamente (si

veda la [Sezione 6.1.3 \[Variabili\]](#), pagina 119). La seconda regola incrementa la variabile `n` ogni volta che si legge un record che soddisfa il criterio di ricerca `'li'`. La regola `END` stampa il valore di `n` alla fine del programma.

I criteri di ricerca speciali `BEGIN` ed `END` non possono essere usati negli intervalli, o con operatori booleani (in effetti, non possono essere combinati con nessun altro operatore). Un programma `awk` può avere molte regole `BEGIN` e/o `END`. Queste sono eseguite nell'ordine in cui compaiono nel programma: tutte le regole `BEGIN` a inizio programma e tutte le regole `END` a fine programma. Le regole `BEGIN` ed `END` possono apparire in qualsiasi posizione all'interno del programma. Questa funzionalità è stata aggiunta nella versione 1987 di `awk` ed è inclusa nello standard POSIX. La versione originale (1978) di `awk` richiedeva che la regola `BEGIN` fosse posta all'inizio del programma, e la regola `END` alla fine del programma, e solo una regola per tipo era ammessa. Ciò non è più obbligatorio, ma è una buona idea continuare a seguire questo modello per migliorare l'organizzazione e la leggibilità del programma.

Regole multiple `BEGIN` ed `END` sono utili per scrivere funzioni di libreria, poiché ogni file di libreria può avere la sua propria regola `BEGIN` e/o `END` per fare la propria inizializzazione e/o pulizia. L'ordine in cui le funzioni di libreria sono menzionate nella riga dei comandi determina l'ordine in cui le rispettive regole `BEGIN` ed `END` sono eseguite. Per questo motivi, occorre prestare attenzione nello scrivere tali regole nei file di libreria, in modo che non sia importante l'ordine in cui tali regole vengono eseguite. Si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33, per maggiori informazioni sull'uso di funzioni di libreria. si veda il [Capitolo 10 \[Una libreria di funzioni `awk`\]](#), pagina 245,, per parecchie utili funzioni di libreria.

Se un programma `awk` ha solo regole `BEGIN` e nessun'altra regola, il programma esce dopo aver eseguito le regole `BEGIN`.<sup>1</sup> Tuttavia, se una regola `END` esiste, l'intero input è letto, anche se non sono presenti altre regole nel programma. Ciò viene fatto necessariamente per permettere che la regola `END` faccia uso delle variabili `FNR` e `NR`.

#### 7.1.4.2 Input/Output dalle regole `BEGIN` ed `END`

Ci sono parecchi punti (talora insidiosi) da tener presente se si fa dell'I/O all'interno di una regola `BEGIN` o `END`. Il primo ha a che fare con il valore di `$0` in una regola `BEGIN`. Poiché le regole `BEGIN` sono eseguite prima della lettura di qualsiasi input, non c'è assolutamente alcun record in input, e quindi nessun campo, quando si eseguono delle regole `BEGIN`. I riferimenti a `$0` e ai campi restituiscono una stringa nulla o zero, a seconda del contesto. Un modo per assegnare un valore effettivo a `$0` è di eseguire un comando `getline` senza indicare una variabile (si veda la [Sezione 4.9 \[Richiedere input usando `getline`\]](#), pagina 83). Un altro modo è semplicemente quello di assegnare un valore a `$0`.

Il secondo punto è simile al primo, ma in direzione opposta. Tradizionalmente, più che altro per problemi di implementazione, `$0` e `NF` erano *indefiniti* all'interno di una regola `END`. Lo standard POSIX prescrive che `NF` sia disponibile all'interno di una regola `END`. Contiene il numero di campi dell'ultimo record in input. Probabilmente per una svista, lo standard non specifica che è reso disponibile anche `$0`, sebbene possa apparire logico che sia così. In effetti, `BWK awk`, `mawk` e `gawk` mantengono il valore di `$0` in modo che sia possibile usarlo all'interno delle regole `END`. Occorre peraltro tener presente che alcune altre implementazioni e parecchie tra le versioni più vecchie di Unix `awk` non si comportano così.

<sup>1</sup> La versione originale di `awk` continuava a leggere e ignorare i record in input fino alla fine del file.

Il terzo punto è una conseguenza dei primi due. Il significato di `'print'` all'interno di una regola `BEGIN` o `END` è quello di sempre: `'print $0'`. Se `$0` è la stringa nulla, stampa una riga vuota. Molti programmatori di lungo corso di `awk` usano un semplice `'print'` all'interno delle regole `BEGIN` ed `END`, intendendo `'print ""'`, contando sul fatto che `$0` sia una stringa nulla. Sebbene queste funzioni solitamente con le regole `BEGIN`, è una pessima idea nelle regole `END`, almeno in `gawk`. È anche stilisticamente inelegante, perché se serve una riga vuota in output, il programma dovrebbe stamparne una esplicitamente.

Per finire, le istruzioni `next` e `nextfile` non sono consentite all'interno di una regola `BEGIN`, perché il ciclo implicito leggi-un-record-e-confrontalo-con-le-regole non è ancora iniziato. Analogamente, tali istruzioni non sono valide all'interno di una regola `END`, perché tutto l'input è già stato letto. (Si veda la [Sezione 7.4.8 \[L'istruzione next\]](#), pagina 159, e si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160.)

### 7.1.5 I criteri di ricerca speciali BEGINFILE ed ENDFILE

Questa sezione descrive una funzionalità specifica di `gawk`.

Due tipi speciali di criterio di ricerca, `BEGINFILE` ed `ENDFILE`, forniscono degli “agganci” per intervenire durante il ciclo di elaborazione dei file specificati sulla riga di comando di `gawk`. Come con le regole `BEGIN` ed `END` (si veda la sezione precedente), tutte le regole `BEGINFILE` in un programma sono riunite, mantenendole nell'ordine in cui sono lette da `gawk` e lo stesso viene fatto per tutte le regole `ENDFILE`.

Il corpo delle regole `BEGINFILE` è eseguito subito prima che `gawk` legga il primo record da un file. La variabile `FILENAME` è impostata al nome del file corrente e `FNR` è impostata a zero.

La regola `BEGINFILE` dà la possibilità di eseguire due compiti che sarebbe difficile o impossibile effettuare altrimenti:

- Si può verificare che il file sia leggibile. Di solito, se un file presente nella riga dei comandi non può essere aperto in lettura, il programma `gawk` viene terminato. Comunque, questo si può evitare, per poi passare a elaborare il file successivo specificato sulla riga dei comandi.

Questo controllo è fattibile controllando se la variabile `ERRNO` è diversa dalla stringa nulla; se è questo il caso, `gawk` non è riuscito ad aprire il file. In questo caso il programma può eseguire un'istruzione `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160). In questo modo `gawk` salta completamente l'elaborazione di quel file. In caso contrario, `gawk` termina come al solito con un errore fatale.

- Se sono state scritte estensioni che modificano la gestione del record (tramite l'inserzione di un “analizzatore di input”; si veda la [Sezione 16.4.5.4 \[Analizzatori di input personalizzati\]](#), pagina 408), è possibile richiamarle a questo punto, prima che `gawk` inizi a elaborare il file. (Questa è una funzionalità *molto* avanzata, usata al momento solo dal [progetto gawkextlib](#).)

La regola `ENDFILE` è chiamata quando `gawk` ha finito di elaborare l'ultimo record di un file in input. Per l'ultimo file in input, è chiamata prima di ogni regola `END`. La regola `ENDFILE` è eseguita anche per file in input vuoti.

Normalmente, se si verifica un errore di lettura durante il normale ciclo di elaborazione dell'input, questo è considerato fatale (il programma termina). Tuttavia, se è presente una

regola `ENDFILE`, l'errore non è considerato fatale, ma viene impostato `ERRNO`. Ciò permette di intercettare ed elaborare errori di I/O a livello di programma `awk`.

L'istruzione `next` (si veda la [Sezione 7.4.8 \[L'istruzione `next`\], pagina 159](#)) non è permessa all'interno di una regola `BEGINFILE` o `ENDFILE`. L'istruzione `nextfile` è consentita solo all'interno di una regola `BEGINFILE`, non all'interno di una regola `ENDFILE`.

L'istruzione `getline` (si veda la [Sezione 4.9 \[Richiedere input usando `getline`\], pagina 83](#)) è limitata all'interno sia di `BEGINFILE` che di `ENDFILE`: solo le forme ridirette di `getline` sono permesse.

`BEGINFILE` ed `ENDFILE` sono estensioni `gawk`. In molte altre implementazioni di `awk` o se `gawk` è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), non sono regole speciali.

### 7.1.6 Il criterio di ricerca vuoto

Un criterio di ricerca vuoto (cioè omesso) corrisponde a *ogni* record in input. Per esempio, il programma:

```
awk '{ print $1 }' mail-list
```

stampa il primo campo di ogni record.

## 7.2 Usare variabili di shell in programmi

I programmi `awk` sono spesso usati come componenti di programmi più ampi, scritti in un linguaggio di shell. Per esempio, è molto comune usare una variabile di shell per specificare un criterio di ricerca che il programma `awk` deve poi individuare. Ci sono due modi per rendere disponibile il valore di una variabile di shell all'interno di un programma `awk`.

Un modo comune è quello di usare i doppi apici per sostituire il valore della variabile nel programma `awk` contenuto nello *script*:

Per esempio, si consideri il programma seguente:

```
printf "Immettere il criterio di ricerca: "
read criterio_di_ricerca
awk "/$criterio_di_ricerca/ '{ num_trov++ }
END { print num_trov, \"occorrenze trovate\" }' /nome/file/dati
```

Il programma `awk` consiste di due pezzi di testo tra apici che, concatenati insieme, formano il programma. La prima parte è tra doppi apici, per consentire la sostituzione della variabile di shell `criterio_di_ricerca` contenuta al loro interno. La seconda parte è racchiusa tra apici singoli.

La sostituzione di variabile attraverso gli apici funziona, ma può facilmente generare difficoltà. Richiede una buona comprensione delle regole per l'uso degli apici nella shell (si veda la [Sezione 1.1.6 \[Uso di apici nella shell\], pagina 21](#)), e spesso è difficile accoppiare i vari apici quando si legge il programma.

Un metodo migliore è quello di usare la funzionalità di assegnamento delle variabili di `awk` (si veda la [Sezione 6.1.3.2 \[Assegnare una variabile dalla riga di comando\], pagina 120](#)) per assegnare il valore di una variabile di shell a una variabile di `awk`. Poi si possono usare *regex* dinamiche come criterio di ricerca (si veda la [Sezione 3.6 \[Usare \*regex\* dinamiche\], pagina 57](#)). Quanto segue mostra come sarebbe l'esempio precedente usando questa tecnica:

```
printf "Immettere il criterio di ricerca: "
```

```
read criterio_di_ricerca
awk -v crit="$criterio_di_ricerca" '$0 ~ crit { num_trov++ }
    END { print num_trov, "occorrenze trovate" }' /nome/file/dati
```

Adesso il programma `awk` è solo una stringa tra apici semplici. L'assegnamento `'-v crit="$criterio_di_ricerca"'` richiede ancora doppi apici, per il caso in cui uno spazio vuoto sia presente nel valore di `$criterio_di_ricerca`. La variabile `awk crit` potrebbe avere come nome anche `criterio_di_ricerca`, ma ciò potrebbe essere causa di confusione. Usare una variabile permette una maggiore flessibilità, poiché la variabile può essere usata in ogni parte del programma—per stamparla, per indicizzare un vettore, o per qualsiasi altro scopo—senza che sia necessario l'artificio di doverla inserire usando gli apici.

## 7.3 Azioni

Un programma o script `awk` consiste in una serie di regole e definizioni di funzione frammiste tra loro. (Le funzioni sono descritte più avanti. Si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), [pagina 224](#).) Una regola contiene un criterio di ricerca e un'azione; l'uno o l'altra (ma non tutt'e due) possono essere omessi. Lo scopo di una *azione* è di specificare cosa deve fare `awk` quando si trova una corrispondenza con il criterio di ricerca. Quindi, schematicamente, un programma `awk` è normalmente simile a questo:

```
[criterio di ricerca] { azione }
criterio di ricerca [ { azione } ]
...
function nome(argomenti) { ... }
...
```

Un'azione consiste di una o più *istruzioni awk*, racchiuse fra parentesi graffe (`{...}`). Ogni istruzione specifica una cosa da fare. Le istruzioni sono separate tra loro da dei ritorni a capo o da dei punti e virgola. Le parentesi graffe attorno a un'azione vanno usate anche se l'azione contiene una sola istruzione o se non contiene alcuna istruzione. Comunque, se si omette completamente l'azione, si possono omettere anche le parentesi graffe. Un'azione omessa è equivalente a specificare `{ print $0 }`:

```
/pippo/ { }      se si trova pippo, non fare nulla — azione vuota
/pippo/          se si trova pippo, stampa il record — azione omessa
```

I seguenti tipi di istruzione sono disponibili in `awk`:

### Espressioni

Servono per chiamare funzioni o assegnare valori a variabili (si veda il [Capitolo 6 \[Espressioni\]](#), [pagina 115](#)). L'esecuzione di questo tipo di istruzione calcola semplicemente il valore dell'espressione. Ciò è utile quando l'espressione ha effetti collaterali (si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), [pagina 126](#)).

### Istruzioni di controllo

Specificano il flusso di controllo dei programmi `awk`. Il linguaggio `awk` utilizza dei costrutti simili a quelli del C, (`if`, `for`, `while` e `do`), e anche alcune altre di tipo speciale (si veda la [Sezione 7.4 \[Istruzioni di controllo nelle azioni\]](#), [pagina 153](#)).

#### Istruzioni composte

Sono una o più istruzioni racchiuse tra parentesi graffe. Un'istruzione composta è usata per riunire un gruppo di istruzioni all'interno di un'istruzione `if`, `while`, `do` o `for`.

#### Istruzioni di input

Usano il comando `getline` (si veda la [Sezione 4.9 \[Richiedere input usando `getline`\]](#), pagina 83). In `awk` sono anche disponibili le istruzioni `next` (si veda la [Sezione 7.4.8 \[L'istruzione `next`\]](#), pagina 159) e `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione `nextfile`\]](#), pagina 160).

#### Istruzioni di output

Come `print` e `printf`. Si veda il [Capitolo 5 \[Stampare in output\]](#), pagina 95.

#### Istruzioni di cancellazione

Per eliminare elementi di vettori. Si veda la [Sezione 8.4 \[L'istruzione `delete`\]](#), pagina 187.

## 7.4 Istruzioni di controllo nelle azioni

Le *istruzioni di controllo*, come `if`, `while` e così via, regolano il flusso di esecuzione nei programmi `awk`. Molte tra le istruzioni di controllo di `awk` sono modellate sulle corrispondenti istruzioni in C. Tutte le istruzioni di controllo iniziano con parole chiave speciali, come `if` e `while`, per distinguerle dalle semplici espressioni. Molte istruzioni di controllo contengono altre istruzioni. Per esempio, l'istruzione `if` contiene un'altra istruzione che può essere eseguita oppure no. Le istruzioni contenute sono chiamate *corpo*. Per includere più di un'istruzione nel corpo, queste vanno raggruppate in una sola *istruzione composta* tra parentesi graffe, e separate tra loro con dei ritorni a capo o dei punti e virgola.

### 7.4.1 L'istruzione `if-else`

L'istruzione `if-else` è quella che serve in `awk` per prendere decisioni. È simile a questa:

```
if (condizione) se-vera-fai [else se-falsa-fai]
```

La *condizione* è un'espressione che controlla quel che fa il resto dell'istruzione. Se la *condizione* è vera, viene eseguita la parte *se-vera-fai*; altrimenti viene eseguita la parte *se-falsa-fai*. La parte *else* dell'istruzione è facoltativa. La condizione è considerata falsa se il suo valore è zero o la stringa nulla; altrimenti, la condizione è vera. Si consideri quanto segue:

```
if (x % 2 == 0)
    print "x è pari"
else
    print "x è dispari"
```

In questo esempio, se l'espressione `'x % 2 == 0'` è vera (cioè, se il valore di `x` è esattamente divisibile per due), allora viene eseguita la prima istruzione `print`; altrimenti, viene eseguita la seconda istruzione `print`. Se la parola chiave `else` sta sulla stessa riga di *se-vera-fai* e se *se-vera-fai* non è un'istruzione composta (cioè, non è racchiusa tra parentesi graffe), allora un punto e virgola deve separare *se-vera-fai* dalla parola chiave `else`. Per chiarire questo, l'esempio precedente si può riscrivere come:

```
if (x % 2 == 0) print "x è pari"; else
    print "x è dispari"
```

Se il ‘;’ è omissso, **awk** non può interpretare l’istruzione e segnala un errore di sintassi. Non si dovrebbero scrivere programmi in questo modo, perché a chi li legge potrebbe sfuggire la parola chiave **else** se non è la prima parola della riga.

### 7.4.2 L’istruzione **while**

Nella programmazione, un *ciclo* è una parte di un programma che può essere eseguita due o più volte consecutivamente. L’istruzione **while** è la più semplice istruzione iterativa in **awk**. Esegue ripetutamente un’istruzione finché una data condizione è vera. Per esempio:

```
while (condizione)
    corpo-del-ciclo
```

*corpo-del-ciclo* è un’istruzione detta *corpo* del ciclo, e *condizione* è un’espressione che controlla per quante volte il ciclo deve continuare a essere ripetuto. La prima cosa che l’istruzione **while** fa è un controllo della *condizione*. Se la *condizione* è vera, viene eseguita l’istruzione *corpo-del-ciclo*. Dopo che le istruzioni in *corpo-del-ciclo* sono state eseguite, *condizione* è controllata nuovamente, e se è ancora vera, *corpo-del-ciclo* viene eseguito ancora. Questo processo è ripetuto finché *condizione* rimane vera. Se la *condizione* è falsa fin dall’inizio, il corpo del ciclo non viene eseguito per nulla, e **awk** continua con l’istruzione che viene dopo il ciclo. Questo esempio stampa i primi tre campi di ogni record in input, uno per riga:

```
awk '
{
    i = 1
    while (i <= 3) {
        print $i
        i++
    }
}' inventory-shipped
```

Il corpo di questo ciclo è un’istruzione composta racchiusa tra parentesi graffe, che contiene due istruzioni. Il ciclo funziona in questo modo: all’inizio, il valore di *i* è impostato a 1. Poi, l’istruzione **while** controlla se *i* è minore o uguale a tre. Ciò è vero quando *i* è uguale a 1, quindi il campo *i*-esimo viene stampato. Quindi l’istruzione ‘*i++*’ incrementa il valore di *i* e il ciclo viene ripetuto. Il ciclo termina quando *i* assume il valore quattro.

Un ritorno a capo non è richiesto tra la condizione e il corpo del ciclo; tuttavia, se lo si mette, il programma è di più facile comprensione, a meno che il corpo del ciclo non sia un’istruzione composta, oppure se è qualcosa di molto semplice. Neppure il ritorno a capo dopo la parentesi graffa aperta che inizia l’istruzione composta è necessario, ma il programma è di lettura più difficile se lo si omette.

### 7.4.3 L’istruzione **do-while**

Il ciclo **do** è una variazione dell’istruzione di ciclo **while**. Il ciclo **do** esegue il *corpo-del-ciclo* una volta e poi ripete il *corpo-del-ciclo* finché la *condizione* rimane vera. È simile a questo:

```
do
    corpo-del-ciclo
while (condizione)
```

Anche se la *condizione* è falsa fin dall'inizio, il *corpo-del-ciclo* viene eseguito almeno una volta (e solo una volta, a meno che l'esecuzione di *corpo-del-ciclo* non renda vera la *condizione*). Si confronti con il corrispondente `while`:

```
while (condizione)
    corpo-del-ciclo
```

Quest'istruzione non esegue il *corpo-del-ciclo* neppure una volta, se la *condizione* è falsa fin dall'inizio. Il seguente è un esempio di `do`:

```
{
    i = 1
    do {
        print $0
        i++
    } while (i <= 10)
}
```

Questo programma stampa ogni record in input per 10 volte. Non si tratta, peraltro, di un esempio molto realistico, perché in questo caso un semplice `while` sarebbe sufficiente. Questa osservazione riflette un'esperienza reale; solo occasionalmente è davvero necessario usare un `do`.

#### 7.4.4 L'istruzione `for`

L'istruzione `for` rende più agevole contare le iterazioni di un ciclo. La forma generale dell'istruzione `for` è simile a questa:

```
for (inizializzazione; condizione; incremento)
    corpo-del-ciclo
```

La parti *inizializzazione*, *condizione* e *incremento* sono espressioni `awk` a piacere, e *corpo-del-ciclo* indica qualsiasi istruzione `awk`.

L'istruzione `for` comincia con l'esecuzione di *inizializzazione*. Poi, finché la *condizione* è vera, esegue ripetutamente *corpo-del-ciclo*, e quindi *incremento*. Tipicamente, *inizializzazione* imposta una variabile a zero o a uno, *incremento* aggiunge uno alla stessa e *condizione* ne confronta il valore rispetto al numero desiderato di iterazioni. Per esempio:

```
awk '
{
    for (i = 1; i <= 3; i++)
        print $i
}' inventory-shipped
```

Questo programma stampa i primi tre campi di ogni record in input, mettendo un campo su ogni riga.

Non è possibile impostare più di una variabile nella parte di *inizializzazione* senza usare un'istruzione di assegnamento multiplo, come `'x = y = 0'`. Ciò ha senso solo se tutti i valori iniziali sono uguali. (Ma è possibile inizializzare ulteriori variabili scrivendo i relativi assegnamenti come istruzioni separate *prima* del ciclo `for`.)

Lo stesso vale per la parte *incremento*. Se serve incrementare ulteriori variabili, questo va fatto con istruzioni separate alla fine del ciclo. L'espressione composta del linguaggio C, che usa l'operatore virgola [,] del C, sarebbe utile in questo contesto, ma non è prevista in `awk`.

Molto spesso, *incremento* è un'espressione di incremento, come nell'esempio precedente. Ma questo non è obbligatorio; può trattarsi di un'espressione qualsiasi. Per esempio, la seguente istruzione stampa tutte le potenze di due comprese tra 1 e 100:

```
for (i = 1; i <= 100; i *= 2)
    print i
```

Se non è necessaria, ognuna delle tre espressioni fra parentesi che segue la parola chiave **for** può essere omessa. Quindi, '**for** (; x > 0;)' è equivalente a '**while** (x > 0)'. Se la *condizione* è omessa del tutto, è ritenuta sempre vera, producendo un *ciclo infinito* (cioè, un ciclo che non finisce mai).

Molto spesso, un ciclo **for** è un'abbreviazione per un ciclo **while**, come si può vedere qui:

```
inizializzazione
while (condizione) {
    corpo-del-ciclo
    incremento
}
```

La sola eccezione è quando l'istruzione **continue** (si veda la [Sezione 7.4.7 \[L'istruzione continue\]](#), [pagina 158](#)) è usata all'interno del ciclo. Se si modifica un'istruzione **for** sostituendola con un'istruzione **while** ciò può cambiare l'effetto dell'istruzione **continue** posta all'interno del ciclo.

Il linguaggio **awk** ha un'istruzione **for** oltre all'istruzione **while** perché un ciclo **for** è spesso più semplice da scrivere, e viene in mente più naturalmente. Contare il numero di iterazioni è molto frequente nei cicli. Può essere più facile pensare a questo conto come parte del ciclo, piuttosto che come qualcosa da fare all'interno del ciclo.

Esiste una versione alternativa al ciclo **for**, per esaminare tutti gli indici di un vettore:

```
for (i in vettore)
    fai qualcosa con vettore[i]
```

Si veda la [Sezione 8.1.5 \[Visitare tutti gli elementi di un vettore\]](#), [pagina 181](#), per maggiori informazioni su questa versione del ciclo **for**.

### 7.4.5 L'istruzione **switch**

Questa sezione descrive una funzionalità disponibile solo in **gawk**. Se **gawk** è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), la funzionalità non è disponibile.

L'istruzione **switch** consente di valutare un'espressione e di eseguire istruzioni se il valore trovato corrisponde a uno dei **case** [casi] previsti. Le istruzioni **case** sono esaminate per cercare una corrispondenza nell'ordine in cui i casi sono definiti nel programma. Se nessuno dei **case** corrisponde al valore dell'espressione, viene eseguita la sezione **default**, se è stata specificata.

Ogni **case** contiene una singola costante, che può essere un numero, una stringa, o una *regex*. Viene valutata l'espressione **switch**, e poi la costante di ogni **case** viene confrontata di volta in volta con il valore risultante. Il tipo di costante determina quale sarà il confronto: per i tipi numerici o stringa si seguono le regole abituali. Per una costante *regex* viene

effettuato un confronto tra l'espressione e il valore di tipo stringa dell'espressione originale. Il formato generale dell'istruzione `switch` è simile a questo:

```
switch (espressione) {
  case valore o espressione regolare:
    corpo-del-caso
  default:
    corpo-del-default
}
```

Il flusso di controllo dell'istruzione `switch` funziona come per il linguaggio C. Una volta stabilita una corrispondenza con un dato caso, le istruzioni che formano il corpo del caso sono eseguite, fino a che non venga trovata un'istruzione `break`, `continue`, `next`, `nextfile` o `exit`, o fino alla fine dell'istruzione `switch` medesima. Per esempio:

```
while ((c = getopt(ARGC, ARGV, "aksx")) != -1) {
  switch (c) {
    case "a":
      # stampa la dimensione di tutti i file
      all_files = TRUE;
      break
    case "k":
      BLOCK_SIZE = 1024      # in blocchi da 1 Kbyte
      break
    case "s":
      # fa solo le somme
      sum_only = TRUE
      break
    case "x":
      # non esce dal filesystem
      fts_flags = or(fts_flags, FTS_XDEV)
      break
    case "?":
    default:
      uso()
      break
  }
}
```

Si noti che se nessuna delle istruzioni specificate qui arresta l'esecuzione di un'istruzione `case` per la quale è stata trovata una corrispondenza; l'esecuzione continua fino al successivo `case` finché non viene interrotta. In questo esempio, il `case` per `"?"` esegue quello di `default`, che consiste nel chiamare una funzione di nome `uso()`. (La funzione `getopt()` qui chiamata è descritta in [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), pagina 263.)

#### 7.4.6 L'istruzione `break`

L'istruzione `break` esce dal ciclo più interno `for`, `while` o `do` dentro al quale si trova. L'esempio seguente trova, se esiste, il divisore più piccolo di un dato numero intero, oppure dichiara che si tratta di un numero primo:

```
# trova il divisore più piccolo di num
```

```

{
    num = $1
    for (divisore = 2; divisore * divisore <= num; divisore++) {
        if (num % divisore == 0)
            break
    }
    if (num % divisore == 0)
        printf "Il più piccolo divisore di %d è %d\n", num, divisore
    else
        printf "%d è un numero primo\n", num
}

```

Quando il resto della divisione è zero nella prima istruzione `if`, `awk` immediatamente esce, a causa del `break`, dal ciclo `for` in cui è contenuto. Ciò vuol dire che `awk` prosegue immediatamente fino all'istruzione che viene dopo il ciclo, e continua l'elaborazione. (L'istruzione `break` è molto differente dall'istruzione `exit`, la quale termina l'intero programma `awk`. Si veda la [Sezione 7.4.10 \[L'istruzione `exit`\]](#), pagina 161.)

Il seguente programma mostra come la *condizione* di un'istruzione `for` o `while` potrebbe essere sostituita da un'istruzione `break` all'interno di un `if`:

```

# trova il divisore più piccolo di num
{
    num = $1
    for (divisore = 2; ; divisore++) {
        if (num % divisore == 0) {
            printf "Il più piccolo divisore di %d è %d\n", num, divisore
            break
        }
        if (divisore * divisore > num) {
            printf "%d è un numero primo\n", num
            break
        }
    }
}

```

L'istruzione `break` è usata anche per terminare l'esecuzione di un'istruzione `switch`. Questo argomento è trattato in [Sezione 7.4.5 \[L'istruzione `switch`\]](#), pagina 156.

L'istruzione `break` non ha significato se usata fuori dal corpo di un ciclo o di un'istruzione `switch`. Tuttavia, anche se la cosa non è mai stata documentata, le prime implementazioni di `awk` consideravano l'istruzione `break` esterna a un ciclo come un'istruzione `next` (si veda la [Sezione 7.4.8 \[L'istruzione `next`\]](#), pagina 159). Versioni recenti di BWK `awk` non consentono più un tale uso, e lo stesso fa `gawk`.



### 7.4.7 L'istruzione `continue`

Analogamente a `break`, l'istruzione `continue` è usata solo all'interno di cicli `for`, `while` e `do`. L'istruzione ignora il resto del corpo del ciclo, facendo sì che la successiva iterazione del ciclo inizi immediatamente. Questo comportamento è differente da quello di `break`, che esce completamente dal ciclo.

L'istruzione `continue` in un ciclo `for` fa sì che `awk` ignori il resto del corpo del ciclo e prosegua l'esecuzione con l'espressione *incremento* dell'istruzione `for`. Il seguente programma è un esempio di ciò:

```
BEGIN {
    for (x = 0; x <= 20; x++) {
        if (x == 5)
            continue
        printf "%d ", x
    }
    print ""
}
```

Questo programma stampa tutti i numeri da 0 a 20—tranne il 5, in cui l'istruzione `printf` è saltata. Siccome l'incremento `'x++'` non viene saltato, `x` non rimane fermo al valore 5. Si confronti il ciclo `for` dell'esempio precedente con il seguente ciclo `while`:

```
BEGIN {
    x = 0
    while (x <= 20) {
        if (x == 5)
            continue
        printf "%d ", x
        x++
    }
    print ""
}
```

Questo programma inizia un ciclo infinito dopo che `x` ha assunto il valore 5, poiché l'istruzione di incremento (`'x++'`) non è mai raggiunta.

L'istruzione `continue` non ha un significato speciale se appare in un'istruzione `switch`, e non ha alcun significato se usata fuori dal corpo di un ciclo. Le prime versioni di `awk` trattavano le istruzioni `continue` che erano fuori da un ciclo allo stesso modo con cui trattavano l'istruzione `break` fuori da un ciclo: come se fosse un'istruzione `next` (si veda la [Sezione 7.4.8 \[L'istruzione next\], pagina 159](#)). Versioni recenti di BWK `awk` non consentono più un tale uso, e lo stesso vale per `gawk`.



### 7.4.8 L'istruzione next

L'istruzione `next` fa sì che `awk` termini immediatamente l'elaborazione del record corrente e proceda a elaborare il record successivo. Ciò vuol dire che nessuna delle eventuali regole successive viene eseguita per il record corrente e che il resto delle azioni presenti nella regola correntemente in esecuzione non viene eseguito.

Si confronti quanto sopra con quel che fa la funzione `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\], pagina 83](#)). Anch'essa fa sì che `awk` legga il record successivo immediatamente, ma non altera il flusso del controllo in alcun modo (cioè, il resto dell'azione in esecuzione prosegue con il nuovo record in input).

Al livello più alto, l'esecuzione di un programma `awk` è un ciclo che legge un record in input e quindi confronta il criterio di ricerca di ciascuna regola con il record stesso. Se si vede questo ciclo come un `for` il cui corpo contiene le regole, l'istruzione `next` è analoga a

un'istruzione **continue**. Salta, cioè, alla fine del corpo di questo ciclo implicito ed esegue l'incremento (ovvero legge un altro record).

Per esempio, si supponga che un programma **awk** agisca solo su record che hanno quattro campi, e che non dovrebbe terminare con un errore se trova dei record in input non validi. Per evitare di complicare il resto del programma, si può scrivere una regola “filtro” a inizio programma, come mostrato in questo esempio:

```
NF != 4 {
    printf("%s:%d: salto: NF != 4\n", FILENAME, FNR) > "/dev/stderr"
    next
}
```

Siccome è presente un **next**, le regole successive del programma non elaboreranno i record non validi. Il messaggio è ridiretto al flusso in output *standard error*, sul quale vanno scritti i messaggi di errore. Per maggiori dettagli, si veda [Sezione 5.8 \[Nomi-file speciali in gawk\]](#), pagina 108.

Se l'istruzione **next** provoca il raggiungimento della fine del file in input, vengono eseguite le eventuali regole **END** presenti. Si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148.

L'istruzione **next** non è consentita all'interno delle regole **BEGINFILE** ed **ENDFILE**. Si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), pagina 150.

Secondo lo standard POSIX, il comportamento di **awk** è indefinito se **next** è usato in una regola **BEGIN** o **END**. **gawk** considera questo come un errore di sintassi. Sebbene POSIX non proibisca di usarlo, molte altre implementazioni di **awk** non consentono che l'istruzione **next** sia usata all'interno del corpo di funzioni. (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224). Come tutte le altre istruzioni **next**, un'istruzione **next** all'interno del corpo di una funzione legge il record successivo e inizia a elaborarlo a partire dalla prima regola del programma.

### 7.4.9 L'istruzione **nextfile**

L'istruzione **nextfile** è simile all'istruzione **next**. Tuttavia, invece di terminare l'elaborazione del record corrente, l'istruzione **nextfile** richiede ad **awk** di terminare di elaborare il file-dati corrente.

Alla fine dell'esecuzione dell'istruzione **nextfile**, **FILENAME** è aggiornato per contenere il nome del successivo file-dati elencato sulla riga di comando, **FNR** è reimpostato a uno, e l'elaborazione riparte con la prima regola del programma. Se l'istruzione **nextfile** raggiunge la fine dei file in input, vengono eseguite le eventuali regole **END** presenti. Un'eccezione a questo si ha se **nextfile** è invocata durante l'esecuzione di qualche istruzione all'interno di una regola **END**; in questo caso, il programma viene terminato immediatamente. Si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148.

L'istruzione **nextfile** è utile quando ci sono parecchi file-dati da elaborare, ma non è necessario elaborare ogni record in ogni file. Senza **nextfile**, per passare al successivo file-dati, un programma dovrebbe continuare a leggere i record che non gli servono. L'istruzione **nextfile** è una maniera molto più efficiente per ottenere lo stesso risultato.

In **gawk**, l'esecuzione di **nextfile** produce ulteriori effetti: le eventuali regole **ENDFILE** sono eseguite se **gawk** non si trova correntemente all'interno di una regola **END** o **BEGINFILE**;

ARGIND è incrementato e le eventuali regole BEGINFILE sono eseguite. (ARGIND non è stato ancora trattato. Si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162.)

In `gawk`, `nextfile` è utile all'interno di una regola BEGINFILE per evitare di elaborare un file che altrimenti causerebbe un errore fatale in `gawk`. In questo caso, le regole ENDFILE non vengono eseguite. Si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), pagina 150.

Sebbene possa sembrare che `'close(FILENAME)'` ottenga lo stesso risultato di `nextfile`, non è così. `close()` può essere usato solo per chiudere file, *pipe* e coprocessi che siano stati aperti tramite ridirezioni. Non ha niente a che vedere con l'elaborazione principale che `awk` fa dei file elencati in ARGV.

**NOTA:** Per molti anni, `nextfile` è stata un'estensione comune. A settembre 2012 si è deciso di includerla nello standard POSIX. Si veda [il sito web dell'Austin Group](#).

Le versioni correnti di BWK `awk` e `mawk` entrambe prevedono `nextfile`. Tuttavia, non sono consentite istruzioni `nextfile` all'interno del corpo delle funzioni (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224). `gawk` lo permette; una `nextfile` all'interno del corpo di una funzione legge il primo record del file successivo e inizia l'elaborazione dello stesso a partire dalla prima regola del programma, esattamente come farebbe qualsiasi altra istruzione `nextfile`.

#### 7.4.10 L'istruzione `exit`

L'istruzione `exit` fa sì che `awk` termini immediatamente l'esecuzione della regola corrente e che termini di elaborare l'input; qualsiasi input ancora da elaborare è ignorato. L'istruzione `exit` è scritta come segue:

```
exit [codice di ritorno]
```

Quando un'istruzione `exit` è eseguita all'interno di una regola BEGIN, il programma termina completamente l'elaborazione. Nessun record in input viene letto. Tuttavia, se una regola END è presente, come parte dell'esecuzione dell'istruzione `exit`, la regola END viene eseguita (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148). Se `exit` è usata nel corpo di una regola END, il programma termina immediatamente.

Un'istruzione `exit` che non fa parte di una regola BEGIN o END termina l'esecuzione di qualsiasi ulteriore regola applicabile al record corrente, salta la lettura di qualsiasi record in input, ed esegue le eventuali regole END. `gawk` salta anche le eventuali regole ENDFILE, che non vengono eseguite.

In questo caso, se non si desidera che la regola END venga eseguita, si deve impostare una variabile a un valore diverso da zero, prima di invocare l'istruzione `exit` e controllarne il valore nella regola END. Si veda la [Sezione 10.2.2 \[Asserzioni\]](#), pagina 249, per un esempio di questo tipo.

Se si specifica un argomento all'istruzione `exit`, il suo valore è usato come codice di ritorno finale dell'elaborazione `awk`. Se non viene specificato alcun argomento, `exit` fa terminare `awk` con un codice di ritorno di "successo". Nel caso in cui un argomento sia specificato in una prima istruzione `exit` e poi `exit` sia chiamato una seconda volta all'interno di una regola END senza alcun argomento, `awk` usa il valore di ritorno specificato in precedenza.

Si veda la [Sezione 2.6 \[Il codice di ritorno all'uscita da `gawk`\]](#), pagina 45, per maggiori informazioni.



Per esempio, si supponga che si sia verificata una condizione di errore difficile o impossibile da gestire. Convenzionalmente, i programmi la segnalano terminando con un codice di ritorno diverso da zero. Un programma `awk` può farlo usando un'istruzione `exit` con un argomento diverso da zero, come mostrato nell'esempio seguente:

```
BEGIN {
    if (("date" | getline data_corrente) <= 0) {
        print "Non riesco a ottenere la data dal sistema" > "/dev/stderr"
        exit 1
    }
    print "la data corrente è", data_corrente
    close("date")
}
```

**NOTA:** Per una completa portabilità, i codici di ritorno dovrebbero essere compresi tra zero e 126, estremi compresi. Valori negativi e valori maggiori o uguali a 127, possono non generare risultati coerenti tra loro in sistemi operativi diversi.

## 7.5 Variabili predefinite

La maggior parte delle variabili `awk` sono disponibili per essere usate dall'utente secondo le proprie esigenze; queste variabili non cambiano mai di valore a meno che il programma non assegni loro dei valori, e non hanno alcuna influenza sul programma a meno che non si decida di utilizzarle nel programma. Tuttavia, alcune variabili in `awk` hanno dei significati speciali predefiniti. `awk` tiene conto di alcune di queste automaticamente, in modo da rendere possibile la richiesta ad `awk` di fare certe cose nella maniera desiderata. Altre variabili sono impostate automaticamente da `awk`, in modo da poter comunicare al programma in esecuzione informazioni sul modo di procedere interno di `awk`.

Questa sezione documenta tutte le variabili predefinite di `gawk`; molte di queste variabili sono anche documentate nei capitoli che descrivono le loro aree di influenza.

### 7.5.1 Variabili predefinite modificabili per controllare `awk`

La seguente è una lista alfabetica di variabili che è possibile modificare per controllare come `awk` gestisce alcuni compiti.

Le variabili che sono specifiche di `gawk` sono contrassegnate da un cancelletto (`#`). Queste variabili sono estensioni `gawk`. In altre implementazioni di `awk`, o se `gawk` è eseguito in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), non hanno un significato speciale. (Eventuali eccezioni sono menzionate nella descrizione di ogni variabile.)

**BINMODE #** Su sistemi non-POSIX, questa variabile specifica l'uso della modalità binaria per tutto l'I/O. I valori numerici di uno, due o tre specificano che i file in input, i file di output o tutti i file, rispettivamente, devono usare I/O binario. Un valore numerico inferiore a zero è trattato come zero e un valore numerico maggiore di tre è trattato come tre. Alternativamente, le stringhe `"r"` o `"w"` specificano che i file in input e i file in output, rispettivamente, devono usare I/O binario. Una stringa `"rw"` o `"wr"` indica che tutti i file devono usare I/O binario. Ogni altro valore di stringa è trattato come `"rw"`, ma `gawk` genera un messaggio di avvertimento. **BINMODE** è descritto in maggior dettaglio in [Sezione B.3.1.3 \[Usare](#)

`gawk` su sistemi operativi PC], pagina 487. `mawk` (si veda la Sezione B.5 [Altre implementazioni di `awk` liberamente disponibili], pagina 494) prevede questa variabile, ma consente solo valori numerici.

**CONVFMT** Una stringa che controlla la conversione di numeri in stringhe (si veda la Sezione 6.1.4 [Conversione di stringhe e numeri], pagina 121). In effetti è la stringa passata come primo argomento alla funzione `sprintf()` (si veda la Sezione 9.1.3 [Funzioni di manipolazione di stringhe], pagina 198). Il suo valore di default è `"%.6g"`. `CONVFMT` è stata introdotta dallo standard POSIX.

#### FIELDWIDTHS #

Una lista di posizioni di colonna, separate da spazi, per dire a `gawk` come dividere campi in input posti su colonne fisse. Assegnando un valore a `FIELDWIDTHS`, le variabili `FS` e `FPAT` *non* vengono usate per effettuare la divisione in campi. Si veda la Sezione 4.6 [Leggere campi di larghezza costante], pagina 77, per maggiori informazioni.

**FPAT #** Un'espressione regolare (di tipo stringa) per dire a `gawk` di creare i campi utilizzando come delimitatore il testo che corrisponde all'espressione regolare. Assegnando un valore a `FPAT` le variabili `FS` e `FIELDWIDTHS` *non* vengono usate per effettuare la divisione in campi. Si veda la Sezione 4.7 [Definire i campi in base al contenuto], pagina 79, per maggiori informazioni.

**FS** Il separatore dei campi in input (si veda la Sezione 4.5 [Specificare come vengono separati i campi], pagina 71). Il valore può essere una stringa di un solo carattere o un'espressione regolare composta da più caratteri che individua il separatore tra i campi dei record in input. Se il suo valore è la stringa nulla (`""`), ogni singolo carattere del record costituisce un campo. (Questo comportamento è un'estensione `gawk`. POSIX `awk` non specifica il comportamento quando `FS` è la stringa nulla. Nonostante questo, alcune altre versioni di `awk` trattano `""` in modo speciale.)

Il valore di default è `" "`, una stringa consistente in un singolo spazio. In via eccezionale, questo valore significa che qualsiasi sequenza di spazi, TAB, e/o ritorni a capo costituisce un solo separatore. Inoltre eventuali spazi, TAB e ritorni a capo all'inizio e alla fine del record in input vengono ignorati.

Si può impostare il valore di `FS` sulla riga dei comandi usando l'opzione `-F`:

```
awk -F, 'programma' file-in-input
```

Se `gawk` sta usando `FIELDWIDTHS` o `FPAT` per separare i campi, assegnare un valore a `FS` fa sì che `gawk` torni alla separazione dei campi normale, fatta utilizzando la variabile `FS`. Un modo semplice per fare questo è semplicemente quello di scrivere l'istruzione `'FS = FS'`, aggiungendo magari un commento esplicativo.

#### IGNORECASE #

Se la variabile `IGNORECASE` è diversa da zero o dalla stringa nulla, tutti i confronti tra stringhe e tutti i confronti tra espressioni regolari sono insensibili alle differenze maiuscolo/minuscolo. Questo vale per il confronto tra `regex` usando `'~'` e `'!~'`, per le funzioni `gensub()`, `gsub()`, `index()`, `match()`, `patsplit()`, `split()` e `sub()`, per la determinazione della fine record con `RS` e per la divisione in campi con `FS` e `FPAT`. Tuttavia, il valore di `IGNORECASE` *non* influenza

gli indici dei vettori e non influenza la separazione dei campi qualora si usi un separatore di campo costituito da un unico carattere. Si veda la [Sezione 3.8 \[Fare confronti ignorando maiuscolo/minuscolo\]](#), pagina 60.

**LINT #** Quando questa variabile è vera (non uguale a zero e non uguale alla stringa nulla), **gawk** si comporta come se fosse stata specificata sulla riga di comando l'opzione `--lint` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). Con un valore di `"fatal"`, gli avvertimenti di *lint* generano un errore fatale. Con un valore di `"invalid"`, sono inviati solo gli avvertimenti per cose che sono effettivamente non valide. (Questa parte non funziona ancora perfettamente.) Ogni altro valore vero stampa avvertimenti non fatali. Se **LINT** ha per valore *falso* nessun avvertimento *lint* viene stampato.

Questa variabile è un'estensione **gawk**. Non ha un valore speciale per altre implementazioni di **awk**. A differenza di altre variabili speciali, modificare il valore di **LINT** altera la produzione di avvertimenti *lint* anche se **gawk** è in modalità compatibile. Analogamente a come le opzioni `--lint` e `--traditional` controllano in maniera indipendente diversi aspetti del comportamento di **gawk**, il controllo degli avvertimenti di *lint* durante l'esecuzione del programma è indipendente dall'implementazione **awk** in esecuzione.

**OFMT** È questa la stringa che controlla la conversione di numeri in stringhe (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121) quando li si stampa con l'istruzione `print`. Funziona passandola come primo argomento alla funzione `sprintf()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Il suo valore di default è `"%.6g"`. Le prime versioni di **awk** usavano **OFMT** per specificare il formato da usare per convertire numeri in stringhe in espressioni generali; questo compito è ora svolto da **CONVFMT**.

**OFS** È il separatore dei campi in output (si veda la [Sezione 5.3 \[I separatori di output e come modificarli\]](#), pagina 97). È ciò che viene stampato in output per separare i campi stampati da un'istruzione `print`. Il suo valore di default è `" "`, una stringa costituita da un solo spazio.

**ORS** Il separatore dei record in output. Viene stampato alla fine di ogni istruzione `print`. Il suo valore di default è `"\n"`, il carattere di ritorno a capo. (Si veda la [Sezione 5.3 \[I separatori di output e come modificarli\]](#), pagina 97.)

**PREC #** La precisione disponibile nei numeri a virgola mobile a precisione arbitraria, per default 53 bit (si veda la [Sezione 15.4.4 \[Impostare la precisione\]](#), pagina 386).

**ROUNDMODE #** La modalità di arrotondamento da usare per operazioni aritmetiche a precisione arbitraria svolte sui numeri, per default `"N"` (`roundTiesToEven` nello standard IEEE 754; si veda la [Sezione 15.4.5 \[Impostare la modalità di arrotondamento\]](#), pagina 387).

**RS** Il separatore tra record in input. Il suo valore di default è una stringa contenente il solo carattere di ritorno a capo, il che significa che un record in input consiste di una sola riga di testo. Il suo valore può essere anche la stringa nulla, nel qual caso i record sono separati da una o più righe vuote. Se invece è una *regex*, i

record sono separati da corrispondenze alla *regex* nel testo in input. (Si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63.)

La possibilità che *RS* sia un'espressione regolare è un'estensione *gawk*. In molte altre implementazioni *awk*, oppure se *gawk* è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), è usato solo il primo carattere del valore di *RS*.

**SUBSEP** Il separatore di indici. Ha il valore di default di `"\034"` ed è usato per separare le parti di cui sono composti gli indici di un vettore multidimensionale. Quindi, l'espressione `'pippo["A", "B"]'` in realtà accede a `pippo["A\034B"]` (si veda la [Sezione 8.5 \[Vettori multidimensionali\]](#), pagina 188).

**TEXTDOMAIN #**

Usata per l'internazionalizzazione di programmi a livello di *awk*. Imposta il dominio di testo (*text domain*) di default per costanti stringa marcate in maniera speciale nel codice sorgente, e anche per le funzioni `dcgettext()`, `dcngettext()` e `bindtextdomain()` (si veda il [Capitolo 13 \[Internazionalizzazione con gawk\]](#), pagina 349). Il valore di default di *TEXTDOMAIN* è `"messages"`.

### 7.5.2 Variabili predefinite con cui *awk* fornisce informazioni

Quella che segue è una lista in ordine alfabetico di variabili che *awk* imposta automaticamente in determinate situazioni per fornire informazioni a un programma.

Le variabili specifiche di *gawk* sono contrassegnate da un cancelletto (`#`). Queste variabili sono estensioni *gawk*. In altre implementazioni di *awk* o se *gawk* è in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), non hanno un significato speciale:

**ARGC, ARGV**

Gli argomenti della riga di comando disponibili ai programmi *awk* sono memorizzati in un vettore di nome *ARGV*. *ARGC* è il numero di argomenti presenti sulla riga di comando. Si veda la [Sezione 2.3 \[Altri argomenti della riga di comando\]](#), pagina 40. A differenza di quasi tutti i vettori di *awk*, *ARGV* è indicizzato da 0 a *ARGC* - 1. Lo si può vedere nell'esempio seguente:

```
$ awk 'BEGIN {
>     for (i = 0; i < ARGC; i++)
>         print ARGV[i]
>     }' inventory-shipped mail-list
+ awk
+ inventory-shipped
+ mail-list
```

*ARGV[0]* contiene `'awk'`, *ARGV[1]* contiene `'inventory-shipped'` e *ARGV[2]* contiene `'mail-list'`. Il valore di *ARGC* è tre, ossia uno in più dell'indice dell'ultimo elemento di *ARGV*, perché gli elementi sono numerati a partire da zero.

I nomi *ARGC* e *ARGV*, come pure la convenzione di indicizzare il vettore da 0 a *ARGC* - 1, derivano dal modo in cui il linguaggio C accede agli argomenti presenti sulla riga di comando.

Il valore di *ARGV[0]* può variare da sistema a sistema. Va anche notato che il programma *non* è incluso in *ARGV*, e non sono incluse neppure le eventuali



opzioni di **awk** specificate sulla riga di comando. Si veda la [Sezione 7.5.3 \[Usare ARGC e ARGV\]](#), [pagina 172](#), per informazioni su come **awk** usa queste variabili.

**ARGIND #** L'indice in **ARGV** del file correntemente in elaborazione. Ogni volta che **gawk** apre un nuovo file-dati per elaborarlo, imposta **ARGIND** all'indice in **ARGV** del nome-file. Quando **gawk** sta elaborando i file in input, il confronto '**FILENAME == ARGV[ARGIND]**' è sempre verificato.

Questa variabile è utile nell'elaborazione dei file; consente di stabilire a che punto ci si trova nella lista di file-dati, e anche di distinguere tra successive occorrenze dello stesso nome-file sulla riga dei comandi.

Anche se è possibile modificare il valore di **ARGIND** all'interno del programma **awk**, **gawk** automaticamente lo imposta a un nuovo valore quando viene aperto il file successivo.

**ENVIRON** Un vettore associativo contenente i valori delle variabili d'ambiente. Gli indici del vettore sono i nomi delle variabili d'ambiente; gli elementi sono i valori della specifica variabile d'ambiente. Per esempio, **ENVIRON["HOME"]** potrebbe valere **/home/arnold**.

Per POSIX **awk**, le modifiche a questo vettore non cambiano le variabili d'ambiente passate a qualsivoglia programma che **awk** può richiamare tramite una ridirezione o usando la funzione **system()**.

Tuttavia, a partire dalla versione 4.2, se non si è in modalità compatibile POSIX, **gawk** aggiorna le proprie variabili d'ambiente, quando si modifica **ENVIRON**, e in questo modo sono modificate anche le variabili d'ambiente disponibili ai programmi richiamati. Un'attenzione speciale dovrebbe essere prestata alla modifica di **ENVIRON["PATH"]**, che è il percorso di ricerca usato per trovare i programmi eseguibili.

Queste modifiche possono anche influire sul programma **gawk**, poiché alcune funzioni predefinite possono tener conto di certe variabili d'ambiente. L'esempio più notevole di una tale situazione è **mktime()** (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), [pagina 214](#)) che, in molti sistemi, tiene conto del valore della variabile d'ambiente **TZ**.

Alcuni sistemi operativi possono non avere variabili d'ambiente. In tali sistemi, il vettore **ENVIRON** è vuoto (tranne che per le variabili **ENVIRON["AWKPATH"]** e **ENVIRON["AWKLIBPATH"]** eventualmente presenti; si veda la [Sezione 2.5.1 \[Ricerca di programmi awk in una lista di directory\]](#), [pagina 42](#), e si veda la [Sezione 2.5.2 \[Ricerca di librerie condivise awk su varie directory\]](#), [pagina 43](#)).

**ERRNO #** Se si verifica un errore di sistema durante una ridirezione per **getline**, durante una lettura per **getline** o durante un'operazione di **close()**, la variabile **ERRNO** contiene una stringa che descrive l'errore.

Inoltre, **gawk** annulla **ERRNO** prima di aprire ogni file in input presente sulla riga di comando. Questo consente di controllare se il file è accessibile all'interno di un criterio di ricerca **BEGINFILE** (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), [pagina 150](#)).

Per il resto, **ERRNO** si comporta analogamente alla variabile C **errno**. Tranne nel caso sopra accennato, **gawk** non annulla *mai* **ERRNO** (lo imposta a ze-

ro o a ""). Quindi, ci si deve aspettare che il suo valore sia significativo solo quando un'operazione di I/O restituisce un valore che indica il fallimento dell'operazione, come per esempio quando `getline` restituisce `-1`. Si è, naturalmente, liberi di annullarla prima di effettuare un'operazione di I/O.

Se il valore di `ERRNO` corrisponde a un errore di sistema della variabile `C errno`, `PROCINFO["errno"]` sarà impostato al valore di `errno`. Per errori non di sistema, `PROCINFO["errno"]` sarà impostata al valore zero.

**FILENAME** Il nome del file in input corrente. Quando non ci sono file-dati sulla riga dei comandi, `awk` legge dallo standard input e `FILENAME` è impostato a `"-"`. `FILENAME` cambia ogni volta che si legge un nuovo file (si veda il [Capitolo 4 \[Leggere file in input\]](#), pagina 63). All'interno di una regola `BEGIN`, il valore di `FILENAME` è "", perché non si sta elaborando alcun file in input.<sup>2</sup> Si noti, tuttavia, che l'uso di `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83) all'interno di una regola `BEGIN` può implicare l'assegnamento di un valore a `FILENAME`.



**FNR** Il numero del record corrente nel file corrente. `awk` incrementa `FNR` ogni volta che legge un nuovo record (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63). `awk` imposta nuovamente a zero `FNR` ogni volta che inizia a leggere un nuovo file in input.

**NF** Il numero di campi nel corrente record in input. `NF` è impostato ogni volta che si legge un nuovo record, quando un nuovo campo viene creato, o quando si modifica `$0` (si veda la [Sezione 4.2 \[Un'introduzione ai campi\]](#), pagina 67).

A differenza di molte altre variabili descritte in questa sottosezione, l'assegnamento di un valore a `NF` può potenzialmente influenzare il funzionamento interno di `awk`. In particolare, assegnamenti a `NF` si possono usare per aggiungere o togliere campi dal record corrente. Si veda la [Sezione 4.4 \[Cambiare il contenuto di un campo\]](#), pagina 69.

**FUNCTAB #** Un vettore i cui indici e i corrispondenti valori sono i nomi di tutte le funzioni predefinite, definite dall'utente ed estese, presenti nel programma.

**NOTA:** Il tentativo di usare l'istruzione `delete` per eliminare il vettore `FUNCTAB` genera un errore fatale. Genera un errore fatale anche ogni tentativo di impostare un elemento di `FUNCTAB`.

**NR** Il numero di record in input che `awk` ha elaborato dall'inizio dell'esecuzione del programma (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63). `awk` incrementa `NR` ogni volta che legge un nuovo record.

**PROCINFO #**

Gli elementi di questo vettore danno accesso a informazioni sul programma `awk` in esecuzione. I seguenti elementi (elencati in ordine alfabetico) sono sicuramente sempre disponibili:

`PROCINFO["egid"]`

Il valore restituito dalla chiamata di sistema `getegid()`.

<sup>2</sup> Alcune tra le prime implementazioni di Unix `awk` inizializzavano `FILENAME` a `"-"`, anche se vi erano file-dati da elaborare. Un tale comportamento era incorretto e non ci si dovrebbe poter contare nei programmi.

**PROCINFO["errno"]**  
 Il valore della variabile C `ERRNO` quando `ERRNO` è impostato al messaggio di errore a essa associato.

**PROCINFO["euid"]**  
 Il valore restituito dalla chiamata di sistema `geteuid()`.

**PROCINFO["FS"]**  
 Questo elemento vale "FS" se è in uso la separazione in campi con FS, "FIELDWIDTHS" se è in uso quella con `FIELDWIDTHS`, oppure "FPAT" se è in uso l'individuazione di campo con `FPAT`.

**PROCINFO["gid"]**  
 Il valore restituito dalla chiamata di sistema `getgid()`.

**PROCINFO["identifiers"]**  
 Un sottovettore, indicizzato dai nomi di tutti gli identificativi usati all'interno del programma **awk**. Un *identificativo* è semplicemente il nome di una variabile (scalare o vettoriale), una funzione predefinita, una funzione definita dall'utente, o una funzione contenuta in un'estensione. Per ogni identificativo, il valore dell'elemento è uno dei seguenti:

- "array"** L'identificativo è un vettore.
- "builtin"** L'identificativo è una funzione predefinita.
- "extension"** L'identificativo è una funzione in un'estensione caricata tramite `@load` o con l'opzione `-l`.
- "scalar"** L'identificativo è uno scalare.
- "untyped"** L'identificativo non ha ancora un tipo (potrebbe essere usato come scalare o come vettore; **gawk** non è ancora in grado di dirlo).
- "user"** L'identificativo è una funzione definita dall'utente.

I valori riportano ciò che **gawk** sa sugli identificativi dopo aver finito l'analisi iniziale del programma; questi valori *non* vengono più aggiornati durante l'esecuzione del programma.

**PROCINFO["pgrp"]**  
 Il *ID di gruppo del processo* del programma corrente.

**PROCINFO["pid"]**  
 Il *process ID* del programma corrente.

**PROCINFO["ppid"]**  
 Il *ID di processo del padre* del programma corrente.

`PROCINFO["strftime"]`

La stringa col formato di default usato per la funzione `strftime()`. Assegnando un nuovo valore a questo elemento si cambia quello di default. Si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214.

`PROCINFO["uid"]`

Il valore restituito dalla chiamata di sistema `getuid()`.

`PROCINFO["version"]`

La versione di `gawk`.

I seguenti elementi addizionali del vettore sono disponibili per fornire informazioni sulle librerie MPFR e GMP, se la versione in uso di `gawk` consente il calcolo con precisione arbitraria (si veda la [Capitolo 15 \[Calcolo con precisione arbitraria con gawk\]](#), pagina 379):

`PROCINFO["gmp_version"]`

La versione della libreria GNU MP.

`PROCINFO["mpfr_version"]`

La versione della libreria GNU MPFR.

`PROCINFO["prec_max"]`

La massima precisione consentita da MPFR.

`PROCINFO["prec_min"]`

La precisione minima richiesta da MPFR.

I seguenti elementi addizionali del vettore sono disponibili per fornire informazioni sulla versione dell'estensione API, se la versione di `gawk` prevede il caricamento dinamico di funzioni di estensione (si veda il [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395):

`PROCINFO["api_major"]`

La versione principale dell'estensione API.

`PROCINFO["api_minor"]`

La versione secondaria dell'estensione API.

Su alcuni sistemi, ci possono essere elementi nel vettore, da `"group1"` fino a `"groupN"`. *N* è il numero di gruppi supplementari che il processo [Unix] possiede. Si usi l'operatore `in` per verificare la presenza di questi elementi (si veda la [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\]](#), pagina 179).

I seguenti elementi consentono di modificare il comportamento di `gawk`:

`PROCINFO["NONFATAL"]`

Se questo elemento esiste, gli errori di I/O per tutte le ridirezioni consentono la prosecuzione del programma. Si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112.

`PROCINFO["nome_output", "NONFATAL"]`

Gli errori in output per il file `nome_output` consentono la prosecuzione del programma. Si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112.

`PROCINFO["comando", "pty"]`

Per una comunicazione bidirezionale con *comando*, si usi una pseudo-tty invece di impostare una *pipe* bidirezionale. Si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), [pagina 339](#), per ulteriori informazioni.

`PROCINFO["input_name", "READ_TIMEOUT"]`

Imposta un tempo limite per leggere dalla ridirezione di input *input\_name*. Si veda la [Sezione 4.10 \[Leggere input entro un tempo limite\]](#), [pagina 90](#), per ulteriori informazioni.

`PROCINFO["sorted_in"]`

Se questo elemento esiste in `PROCINFO`, il suo valore controlla l'ordine in cui gli indici dei vettori saranno elaborati nei cicli `for (indice in vettore)`. Questa è una funzionalità avanzata, la cui descrizione completa sarà vista più avanti; si veda [Sezione 8.1.5 \[Visitare tutti gli elementi di un vettore\]](#), [pagina 181](#).

**RLENGTH** La lunghezza della sottostringa individuata dalla funzione `match()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)). `RLENGTH` viene impostata quando si richiama la funzione `match()`. Il suo valore è la lunghezza della stringa individuata, oppure `-1` se non è stata trovata alcuna corrispondenza.

**RSTART** L'indice, in caratteri, da cui parte la sottostringa che è individuata dalla funzione `match()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)). `RSTART` viene impostata quando si richiama la funzione `match()`. Il suo valore è la posizione nella stringa da cui inizia la sottostringa individuata, oppure zero, se non è stata trovata alcuna corrispondenza.

**RT #** Il testo in input che corrisponde al testo individuato da `RS`, il separatore di record. Questa variabile viene impostata dopo aver letto ciascun record.

**SYMTAB #** Un vettore i cui indici sono i nomi di tutte le variabili globali e i vettori definiti nel programma. `SYMTAB` rende visibile al programmatore `awk` la tabella dei simboli di `gawk`. Viene preparata nella fase di analisi iniziale del programma `gawk` ed è completata prima di cominciare a eseguire il programma.

Il vettore può essere usato per accedere indirettamente, in lettura o in scrittura, al valore di una variabile:

```
pippo = 5
SYMTAB["pippo"] = 4
print pippo    # stampa 4
```

La funzione `isarray()` (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), [pagina 223](#)) si può usare per controllare se un elemento in `SYMTAB` è un vettore. Inoltre, non è possibile usare l'istruzione `delete` con il vettore `SYMTAB`.

È possibile aggiungere a `SYMTAB` un elemento che non sia un identificativo già esistente:

```
SYMTAB["xxx"] = 5
```

```
print SYMTAB["xxx"]
```

Il risultato è quello previsto: in questo caso **SYMTAB** si comporta come un normale vettore. La sola differenza è che non è poi possibile cancellare **SYMTAB["xxx"]**.

Il vettore **SYMTAB** è più interessante di quel che sembra. Andrew Schorr fa notare che effettivamente consente di ottenere dei puntatori ai dati in **awk**. Si consideri quest'esempio:

```
# Moltiplicazione indiretta di una qualsiasi variabile per un
# numero a piacere e restituzione del risultato

function multiply(variabile, numero)
{
    return SYMTAB[variabile] *= numero
}
```

Si potrebbe usare in questo modo:

```
BEGIN {
    risposta = 10.5
    multiply("risposta", 4)
    print "La risposta è", risposta
}
```

Eseguendo, il risultato è:

```
$ gawk -f risposta.awk
+ La risposta è 42
```

**NOTA:** Per evitare seri paradossi temporali,<sup>3</sup> né **FUNCTAB** né **SYMTAB** sono disponibili come elementi all'interno del vettore **SYMTAB**.

---

<sup>3</sup> Per non parlare dei grossi problemi di implementazione.

**Modificare NR e FNR**

**awk** incrementa le variabili **NR** e **FNR** ogni volta che legge un record, invece che impostarle al valore assoluto del numero di record letti. Ciò significa che un programma può modificare queste variabili e i valori così assegnati sono incrementati per ogni record. Si consideri l'esempio seguente:

```
$ echo '1
> 2
> 3
> 4' | awk 'NR == 2 { NR = 17 }
> { print NR }'
+ 1
+ 17
+ 18
+ 19
```

Prima che **FNR** venisse aggiunto al linguaggio **awk** (si veda la [Sezione A.1 \[Differenze importanti tra V7 e System V Release 3.1\]](#), pagina 461), molti programmi **awk** usavano questa modalità per contare il numero di record in un file impostando a zero **NR** al cambiare di **FILENAME**.

**7.5.3 Usare ARGV e ARGV**

La [Sezione 7.5.2 \[Variabili predefinite con cui awk fornisce informazioni\]](#), pagina 165, conteneva il programma seguente che visualizzava le informazioni contenute in **ARGC** e **ARGV**:

```
$ awk 'BEGIN {
>     for (i = 0; i < ARGC; i++)
>         print ARGV[i]
>     }' inventory-shipped mail-list
+ awk
+ inventory-shipped
+ mail-list
```

In questo esempio, **ARGV[0]** contiene **'awk'**, **ARGV[1]** contiene **'inventory-shipped'** e **ARGV[2]** contiene **'mail-list'**. Si noti che il nome del programma **awk** non è incluso in **ARGV**. Le altre opzioni della riga di comando, con i relativi argomenti, sono parimenti non presenti, compresi anche gli assegnamenti di variabile fatti tramite l'opzione **-v** (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). I normali assegnamenti di variabile sulla riga dei comandi *sono* trattati come argomenti e quindi inseriti nel vettore **ARGV**. Dato il seguente programma in un file di nome **vediargomenti.awk**:

```
BEGIN {
    printf "A=%d, B=%d\n", A, B
    for (i = 0; i < ARGC; i++)
        printf "\tARGV[%d] = %s\n", i, ARGV[i]
}
END { printf "A=%d, B=%d\n", A, B }
```

la sua esecuzione produce il seguente risultato:

```
$ awk -v A=1 -f vediargomenti.awk B=2 /dev/null
+ A=1, B=0
```

```

+      ARGV[0] = awk
+      ARGV[1] = B=2
+      ARGV[2] = /dev/null
+ A=1, B=2

```

Un programma può modificare **ARGC** e gli elementi di **ARGV**. Ogni volta che **awk** arriva alla fine di un file in input, usa il successivo elemento nel vettore **ARGV** come nome del successivo file in input. Cambiando il contenuto di quella stringa, un programma può modificare la lista dei file che sono letti. Si usi **"-"** per rappresentare lo standard input. Assegnando ulteriori elementi e incrementando **ARGC** verranno letti ulteriori file.

Se il valore di **ARGC** viene diminuito, vengono ignorati i file in input posti alla fine della lista. Memorizzando il valore originale di **ARGC** da qualche altra parte, un programma può gestire gli argomenti ignorati come se fossero qualcosa di diverso dai nome-file.

Per eliminare un file che sia in mezzo alla lista, si imposti in **ARGV** la stringa nulla (**"**) al posto del nome del file in questione. Come funzionalità speciale, **awk** ignora valori di nome-file che siano stati rimpiazzati con la stringa nulla. Un'altra possibilità è quella di usare l'istruzione **delete** per togliere elementi da **ARGV** (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187).

Tutte queste azioni sono tipicamente eseguite nella regola **BEGIN**, prima di iniziare l'elaborazione vera e propria dell'input. Si veda la [Sezione 11.2.4 \[Suddividere in pezzi un file grosso\]](#), pagina 292, e si veda la [Sezione 11.2.5 \[Inviare l'output su più di un file\]](#), pagina 294, per esempi su ognuno dei modi per togliere elementi dal vettore **ARGV**.

Per passare direttamente delle opzioni a un programma scritto in **awk**, si devono terminare le opzioni di **awk** con **--** e poi inserire le opzioni destinate al programma **awk**, come mostrato qui di seguito:

```
awk -f mio_programma.awk -- -v -q file1 file2 ...
```

Il seguente frammento di programma ispeziona **ARGV** per esaminare, e poi rimuovere, le opzioni sulla riga di comando viste sopra:

```

BEGIN {
    for (i = 1; i < ARGC; i++) {
        if (ARGV[i] == "-v")
            verbose = 1
        else if (ARGV[i] == "-q")
            debug = 1
        else if (ARGV[i] ~ /^-./) {
            e = sprintf("%s: opzione non riconosciuta -- %c",
                        ARGV[0], substr(ARGV[i], 2, 1))
            print e > "/dev/stderr"
        } else
            break
        delete ARGV[i]
    }
}

```

Terminare le opzioni di **awk** con **--** non è necessario in **gawk**. A meno che non si specifichi **--posix**, **gawk** inserisce, senza emettere messaggi, ogni opzione non riconosciuta nel vettore **ARGV** perché sia trattata dal programma **awk**. Dopo aver trovato un'opzione non riconosciuta,

**gawk** non cerca ulteriori opzioni, anche se ce ne fossero di riconoscibili. La riga dei comandi precedente sarebbe con **gawk**:

```
gawk -f mio_programma.awk -q -v file1 file2 ...
```

Poiché **-q** non è un'opzione valida per **gawk**, quest'opzione e l'opzione **-v** che segue sono passate al programma **awk**. (Si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), pagina 263, per una funzione di libreria **awk** che analizza le opzioni della riga di comando.)

Nel progettare un programma, si dovrebbero scegliere opzioni che non siano in conflitto con quelle di **gawk**, perché ogni opzione accettata da **gawk** verrà elaborata prima di passare il resto della riga dei comandi al programma **awk**. Usare **#!** con l'opzione **-E** può essere utile in questo caso (si veda la [Sezione 1.1.4 \[Programmi \*\*awk\*\* da eseguire come script\]](#), pagina 19, e si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

## 7.6 Sommario

- Coppie *criterio di ricerca-azione* sono gli elementi base di un programma **awk**. I criteri di ricerca possono essere espressioni normali, espressioni di intervallo, o costanti *regexp*; possono anche essere i criteri speciali **BEGIN**, **END**, **BEGINFILE** o **ENDFILE**; o essere omessi. L'azione viene eseguita se il record corrente soddisfa il criterio di ricerca. Criteri di ricerca vuoti (omessi) corrispondono a tutti i record in input.
- L'I/O effettuato nelle azioni delle regole **BEGIN** ed **END** ha alcuni vincoli. Questo vale a maggior ragione per le regole **BEGINFILE** ed **ENDFILE**. Queste ultime due forniscono degli "agganci" per interagire con l'elaborazione dei file fatta da **gawk**, consentendo di risolvere situazioni che altrimenti genererebbero degli errori fatali (ad esempio per un file che non si è autorizzati a leggere).
- Le variabili di shell possono essere usate nei programmi **awk** prestando la dovuta attenzione all'uso degli apici. È più facile passare una variabile di shell ad **awk** usando l'opzione **-v** e una variabile **awk**.
- Le azioni sono formate da istruzioni racchiuse tra parentesi graffe. Le istruzioni sono composte da espressioni, istruzioni di controllo, istruzioni composte, istruzioni di input/output e istruzioni di cancellazione.
- Le istruzioni di controllo in **awk** sono **if-else**, **while**, **for** e **do-while**. **gawk** aggiunge l'istruzione **switch**. Ci sono due tipi di istruzione **for**: uno per eseguire dei cicli, e l'altro per esaminare un vettore.
- Le istruzioni **break** e **continue** permettono di uscire velocemente da un ciclo, o di passare alla successiva iterazione dello stesso (o di uscire da un'istruzione **switch**).
- Le istruzioni **next** e **nextfile** permettono, rispettivamente, di passare al record successivo, ricominciando l'elaborazione dalla prima regola del programma, o di passare al successivo file in input, sempre ripartendo dalla prima regola del programma.
- L'istruzione **exit** termina il programma. Quando è eseguita dall'interno di un'azione (o nel corpo di una funzione), trasferisce il controllo alle eventuali istruzioni **END**. Se è eseguita nel corpo di un'istruzione **END**, il programma è terminato immediatamente. È possibile specificare un valore numerico da usare come codice di ritorno di **awk**.
- Alcune variabili predefinite permettono di controllare **awk**, principalmente per l'I/O. Altre variabili trasmettono informazioni da **awk** al programma.

- I vettori `ARGC` e `ARGV` rendono disponibili al programma gli argomenti della riga di comando. Una loro modifica all'interno di una regola `BEGIN` permette di controllare come `awk` elaborerà i file-dati in input.



## 8 Vettori in awk

Un *vettore* è una tabella di valori chiamati *elementi*. Gli elementi di un vettore sono individuati dai loro *indici*. Gli indici possono essere numeri o stringhe.

Questo capitolo descrive come funzionano i vettori in **awk**, come usare gli elementi di un vettore, come visitare tutti gli elementi di un vettore, e come rimuovere elementi da un vettore. Descrive anche come **awk** simula vettori multidimensionali, oltre ad alcuni aspetti meno ovvii sull'uso dei vettori. Il capitolo prosegue illustrando la funzionalità di ordinamento dei vettori di **gawk**, e termina con una breve descrizione della capacità di **gawk** di consentire veri vettori di vettori.

### 8.1 Informazioni di base sui vettori

Questa sezione espone le nozioni fondamentali: elaborare gli elementi di un vettore uno alla volta, e visitare sequenzialmente tutti gli elementi di un vettore.

#### 8.1.1 Introduzione ai vettori

*Visitare sequenzialmente un vettore associativo è come tentare di lapidare qualcuno usando una mitragliatrice Uzi carica.*

—Larry Wall

Il linguaggio **awk** mette a disposizione vettori monodimensionali per memorizzare gruppi di stringhe o di numeri correlati fra loro. Ogni vettore di **awk** deve avere un nome. I nomi dei vettori hanno la stessa sintassi dei nomi di variabile; qualsiasi nome valido di variabile potrebbe essere anche un valido nome di vettore. Un nome però non può essere usato in entrambi i modi (come vettore e come variabile) nello stesso programma **awk**.

I vettori in **awk** assomigliano superficialmente ai vettori in altri linguaggi di programmazione, ma ci sono differenze fondamentali. In **awk**, non è necessario specificare la dimensione di un vettore prima di iniziare a usarlo. In più, qualsiasi numero o stringa può essere usato come indice di un vettore, non solo numeri interi consecutivi.

Nella maggior parte degli altri linguaggi, i vettori devono essere *dichiarati* prima dell'uso, specificando quanti elementi o componenti contengono. In questi linguaggi, la dichiarazione causa l'allocazione, per questi elementi, di un blocco di memoria contiguo. Normalmente, un indice di un vettore dev'essere un intero non negativo. Per esempio, l'indice zero specifica il primo elemento nel vettore, che è effettivamente memorizzato all'inizio di un blocco di memoria. L'indice uno specifica il secondo elemento, che è memorizzato subito dopo il primo elemento, e così via. È impossibile aggiungere ulteriori elementi al vettore, perché esso può contenere solo il numero di elementi dichiarato. (Alcuni linguaggi consentono indici iniziali e finali arbitrari—p.es., '15 .. 27'—però la dimensione del vettore rimane fissa una volta che il vettore sia stato dichiarato.)

Un vettore contiguo di quattro elementi potrebbe essere come quello in [Figura 8.1](#), concettualmente, se i valori degli elementi sono 8, "pippo", "" e 30.

8	"pippo"	" "	30	Valore
0	1	2	3	Indice

Figura 8.1: Un vettore contiguo

Vengono memorizzati solo i valori; gli indici sono definiti implicitamente dall'ordine dei valori. Qui, 8 è il valore il cui indice è zero, perché 8 appare nella posizione con zero elementi prima di essa.

I vettori in **awk** non sono di questo tipo: sono invece *associativi*. Ciò significa che ogni vettore è un insieme di coppie, ognuna costituita da un indice e dal corrispondente valore dell'elemento del vettore:

Indice	Valore
3	30
1	"pippo"
0	8
2	" "

Le coppie sono elencate in ordine casuale perché il loro ordinamento è irrilevante.<sup>1</sup>

Un vantaggio dei vettori associativi è che si possono aggiungere nuove coppie in qualsiasi momento. Per esempio, supponiamo di aggiungere al vettore un decimo elemento il cui valore sia "numero dieci". Il risultato sarà:

Indice	Valore
10	"numero dieci"
3	30
1	"pippo"
0	8
2	" "

Ora il vettore è *sperso*, il che significa semplicemente che non sono usati alcuni indici. Ha gli elementi 0, 1, 2, 3 e 10, ma mancano gli elementi 4, 5, 6, 7, 8 e 9.

Un'altra caratteristica dei vettori associativi è che gli indici non devono essere necessariamente interi non negativi. Qualsiasi numero, o anche una stringa, può essere un indice. Per esempio, il seguente è un vettore che traduce delle parole dall'inglese all'italiano:

Indice	Valore
"dog"	"cane"
"cat"	"gatto"
"one"	"uno"

<sup>1</sup> L'ordine potrà variare nelle diverse implementazioni di **awk**, che tipicamente usa tabelle *hash* per memorizzare elementi e valori del vettore.

```
1      "uno"
```

Qui abbiamo deciso di tradurre il numero uno sia nella forma letterale che in quella numerica, per illustrare che un singolo vettore può avere come indici sia numeri che stringhe. (In effetti, gli indici dei vettori sono sempre stringhe. Ci sono alcune sottigliezze su come funzionano i numeri quando sono usati come indici dei vettori; questo verrà trattato in maggior dettaglio nella [Sezione 8.2 \[Usare numeri per indicizzare i vettori\]](#), pagina 185.) Qui sopra, il numero 1 non è tra doppi apici, perché `awk` lo converte automaticamente in una stringa.

Il valore di `IGNORECASE` non ha alcun effetto sull'indicizzazione dei vettori. Lo stesso valore di stringa usato per memorizzare un elemento di un vettore può essere usato per richiamarlo. Quando `awk` crea un vettore (p.es., con la funzione predefinita `split()`), gli indici di quel vettore sono numeri interi consecutivi a partire da uno. (Si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198.)

I vettori di `awk` sono efficienti: il tempo necessario per accedere a un elemento è indipendente dal numero di elementi nel vettore.

### 8.1.2 Come esaminare un elemento di un vettore

Il modo principale di usare un vettore è quello di esaminare uno dei suoi elementi. Un riferimento al vettore è un'espressione come questa:

```
vettore[espressione-indice]
```

Qui, *vettore* è il nome di un vettore. L'espressione *espressione-indice* è l'indice dell'elemento del vettore che si vuol esaminare.

Il valore del riferimento al vettore è il valore corrente di quell'elemento del vettore. Per esempio, `pippo[4.3]` è un'espressione che richiama l'elemento del vettore `pippo` il cui indice è '4.3'.

Un riferimento a un elemento di un vettore il cui indice non esiste ancora restituisce un valore uguale a "", la stringa nulla. Questo comprende elementi a cui non è stato assegnato un valore ed elementi che sono stati eliminati (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187).

**NOTA:** Un riferimento a un elemento inesistente crea *automaticamente* quell'elemento di vettore, con la stringa nulla come valore. (In certi casi, ciò è indesiderabile, perché potrebbe sprecare memoria all'interno di `awk`.)

I programmatori principianti di `awk` fanno spesso l'errore di verificare se un elemento esiste controllando se il valore è vuoto:

```
# Verifica se "pippo" esiste in a:          Non corretto!
if (a["pippo"] != "") ...
```

Questo è sbagliato per due motivi. Primo, *crea* `a["pippo"]` se ancora non esisteva! Secondo, assegnare a un elemento di un vettore la stringa vuota come valore è un'operazione valida (anche se un po' insolita).

Per determinare se un elemento con un dato indice esiste in un vettore, si usi la seguente espressione:

```
indice in vettore
```

Quest'espressione verifica se lo specifico indice *indice* esiste, senza l'effetto collaterale di creare quell'elemento nel caso che esso non sia presente. L'espressione ha il valore uno

(vero) se `vettore[indice]` esiste e zero (falso) se non esiste. Per esempio, quest'istruzione verifica se il vettore `frequenze` contiene l'indice '2':

```
if (2 in frequenze)
    print "L'indice 2 è presente."
```

Si noti che questo *non* verifica se il vettore `frequenze` contiene un elemento il cui *valore* è 2. Il solo modo far questo è quello di passare in rassegna tutti gli elementi. Inoltre, questo *non* crea `frequenze[2]`, mentre la seguente alternativa (non corretta) lo fa:

```
if (frequenze[2] != "")
    print "L'indice 2 è presente."
```

### 8.1.3 Assegnare un valore a elementi di un vettore

Agli elementi di un vettore possono essere assegnati valori proprio come alle variabili di `awk`:

```
vettore[espressione-indice] = valore
```

`vettore` è il nome di un vettore. L'espressione *espressione-indice* è l'indice dell'elemento del vettore a cui è assegnato il valore. L'espressione *valore* è il valore da assegnare a quell'elemento del vettore.

### 8.1.4 Esempio semplice di vettore

Il seguente programma prende una lista di righe, ognuna delle quali inizia con un numero di riga, e le stampa in ordine di numero di riga. I numeri di riga non sono ordinati al momento della lettura, ma sono invece in ordine sparso. Questo programma ordina le righe mediante la creazione di un vettore che usa i numeri di riga come indici. Il programma stampa poi le righe ordinate secondo il loro numero. È un programma molto semplice e non è in grado di gestire numeri ripetuti, salti di riga o righe che non iniziano con un numero:

```
{
    if ($1 > massimo)
        massimo = $1
    vett[$1] = $0
}

END {
    for (x = 1; x <= massimo; x++)
        print vett[x]
}
```

La prima regola tiene traccia del numero di riga più grande visto durante la lettura; memorizza anche ogni riga nel vettore `vett`, usando come indice il numero di riga. La seconda regola viene eseguita dopo che è stato letto tutto l'input, per stampare tutte le righe. Quando questo programma viene eseguito col seguente input:

```
5 Io sono l'uomo Cinque
2 Chi sei? Il nuovo numero due!
4 . . . E quattro a terra
1 Chi è il numero uno?
3 Sei il tre.
```

Il suo output è:

```
1 Chi è il numero uno?
2 Chi sei? Il nuovo numero due!
3 Sei il tre.
4 . . . E quattro a terra
5 Io sono l'uomo Cinque
```

Se un numero di riga appare più di una volta, l'ultima riga con quel dato numero prevale sulle altre. Le righe non presenti nel vettore si possono saltare con un semplice perfezionamento della regola `END` del programma, in questo modo:

```
END {
    for (x = 1; x <= massimo; x++)
        if (x in vett)
            print vett[x]
}
```

### 8.1.5 Visitare tutti gli elementi di un vettore

Nei programmi che usano vettori, è spesso necessario usare un ciclo che esegue un'azione su ciascun elemento di un vettore. In altri linguaggi, dove i vettori sono contigui e gli indici sono limitati ai numeri interi non negativi, questo è facile: tutti gli indici validi possono essere visitati partendo dall'indice più basso e arrivando a quello più alto. Questa tecnica non è applicabile in `awk`, perché qualsiasi numero o stringa può fare da indice in un vettore. Perciò `awk` ha un tipo speciale di istruzione `for` per visitare un vettore:

```
for (variabile in vettore)
    corpo
```

Questo ciclo esegue *corpo* una volta per ogni indice in *vettore* che il programma ha usato precedentemente, con la variabile *variabile* impostata a quell'indice.

Il seguente programma usa questa forma dell'istruzione `for`. La prima regola visita i record in input e tiene nota di quali parole appaiono (almeno una volta) nell'input, memorizzando un 1 nel vettore `usate` con la parola come indice. La seconda regola visita gli elementi di `usate` per trovare tutte le parole distinte che appaiono nell'input. Il programma stampa poi ogni parola che è più lunga di 10 caratteri e anche il numero di tali parole. Si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#), per maggiori informazioni sulla funzione predefinita `length()`.

```
# Registra un 1 per ogni parola usata almeno una volta
{
    for (i = 1; i <= NF; i++)
        usate[$i] = 1
}

# Trova il numero di parole distinte lunghe più di 10 caratteri
END {
    for (x in usate) {
        if (length(x) > 10) {
            ++numero_parole_lunghe
            print x
        }
    }
}
```

```

    }
  }
  print numero_parole_lunghe, "parole più lunghe di 10 caratteri"
}

```

Si veda la [Sezione 11.3.5 \[Generare statistiche sulla frequenza d'uso delle parole\]](#), pagina 309, per un esempio di questo tipo più dettagliato.

L'ordine nel quale gli elementi del vettore sono esaminati da quest'istruzione è determinato dalla disposizione interna degli elementi del vettore all'interno di **awk** e nell'**awk** standard non può essere controllato o cambiato. Questo può portare a dei problemi se vengono aggiunti nuovi elementi al *vettore* dall'istruzione eseguendo il corpo del ciclo; non è prevedibile se il ciclo **for** li potrà raggiungere. Similmente, modificare *variabile* all'interno del ciclo potrebbe produrre strani risultati. È meglio evitare di farlo.

Di sicuro, **gawk** imposta la lista di elementi su cui eseguire l'iterazione prima che inizi il ciclo, e non la cambia in corso d'opera. Ma non tutte le versioni di **awk** fanno così. Si consideri questo programma, chiamato **vediciclo.awk**:

```

BEGIN {
    a["questo"] = "questo"
    a["è"] = "è"
    a["un"] = "un"
    a["ciclo"] = "ciclo"
    for (i in a) {
        j++
        a[j] = i
        print i
    }
}

```

Ecco quel che accade quando viene eseguito con **gawk** (e **mawk**):

```

$ gawk -f vediciclo.awk
+ questo
+ ciclo
+ un
+ è

```

Se si usa invece **BWK awk**:

```

$ nawk -f vediciclo.awk
+ ciclo
+ questo
+ è
+ un
+ 1

```

### 8.1.6 Visita di vettori in ordine predefinito con **gawk**

Questa sottosezione descrive una funzionalità disponibile solo in **gawk**.

Per default, quando un ciclo **for** visita un vettore, l'ordine è indeterminato, il che vuol dire che l'implementazione di **awk** determina l'ordine in cui il vettore viene attraversato.

Quest'ordine normalmente è basato sull'implementazione interna dei vettori e varia da una versione di **awk** alla successiva.

Spesso, tuttavia, servirebbe fare qualcosa di semplice, come “visitare il vettore confrontando gli indici in ordine crescente,” o “visitare il vettore confrontando i valori in ordine decrescente.” **gawk** fornisce due meccanismi che permettono di farlo.

- Assegnare a `PROCINFO["sorted_in"]` un valore a scelta fra alcuni valori predefiniti. Si vedano più sotto i valori ammessi.
- Impostare `PROCINFO["sorted_in"]` al nome di una funzione definita dall'utente, da usare per il confronto tra gli elementi di un vettore. Questa funzionalità avanzata verrà descritta in seguito in [Sezione 12.2 \[Controllare la visita di un vettore e il suo ordinamento\]](#), pagina 332.

Sono disponibili i seguenti valori speciali per `PROCINFO["sorted_in"]`:

`"@unsorted"`

Lasciare gli elementi del vettore in ordine arbitrario (questo è il comportamento di default di **awk**).

`"@ind_str_asc"`

Ordinare in ordine crescente di indice, confrontando tra loro gli indici come stringhe; questo è l'ordinamento più normale. (Internamente, gli indici dei vettori sono sempre stringhe, per cui con `'a[2*5] = 1'` l'indice è la stringa `"10"` e non il numero 10.)

`"@ind_num_asc"`

Ordinare in ordine crescente di indice, ma nell'elaborazione gli indici vengono trattati come numeri. Gli indici con valore non numerico verranno posizionati come se fossero uguali a zero.

`"@val_type_asc"`

Ordinare secondo il valore degli elementi in ordine crescente (invece che in base agli indici). L'ordinamento è in base al tipo assegnato all'elemento (si veda la [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), pagina 130). Tutti i valori numerici precedono tutti i valori di tipo stringa, che a loro volta vengono prima dei sottovettori. (I sottovettori non sono ancora stati descritti; si veda la [Sezione 8.6 \[Vettori di vettori\]](#), pagina 190.)

`"@val_str_asc"`

Ordinare secondo il valore degli elementi in ordine crescente (invece che in base agli indici). I valori scalari sono confrontati come stringhe. I sottovettori, se presenti, vengono per ultimi.

`"@val_num_asc"`

Ordinare secondo il valore degli elementi in ordine crescente (invece che in base agli indici). I valori scalari sono confrontati come numeri. I sottovettori, se presenti, vengono per ultimi. Quando i valori numerici coincidono, vengono usati i valori di tipo stringa per stabilire un ordinamento: ciò garantisce risultati coerenti tra differenti versioni della funzione C `qsort()`,<sup>2</sup> che **gawk** usa internamente per effettuare l'ordinamento.

<sup>2</sup> Quando due elementi risultano uguali, la funzione C `qsort()` non garantisce che dopo l'ordinamento venga rispettato il loro ordine relativo originale. Usando il valore di stringa per stabilire un ordinamento

`"@ind_str_desc"`

Ordinare come fa `"@ind_str_asc"`, ma gli indici di tipo stringa sono ordinati dal più alto al più basso.

`"@ind_num_desc"`

Ordinare come fa `"@ind_num_asc"`, ma gli indici numerici sono ordinati dal più alto al più basso.

`"@val_type_desc"`

Ordinare come fa `"@val_type_asc"`, ma i valori degli elementi, a seconda del tipo, sono ordinati dal più alto al più basso. I sottovettori, se presenti, vengono per primi.

`"@val_str_desc"`

Ordinare come fa `"@val_str_asc"`, ma i valori degli elementi, trattati come stringhe, sono ordinati dal più alto al più basso. I sottovettori, se presenti, vengono per primi.

`"@val_num_desc"`

Ordinare come fa `"@val_num_asc"`, ma i valori degli elementi, trattati come numeri, sono ordinati dal più alto al più basso. I sottovettori, se presenti, vengono per primi.

L'ordine in cui il vettore è visitato viene determinato prima di iniziare l'esecuzione del ciclo `for`. Cambiare `PROCINFO["sorted_in"]` all'interno del corpo del ciclo non influisce sul ciclo stesso. Per esempio:

```
$ gawk '
> BEGIN {
>   a[4] = 4
>   a[3] = 3
>   for (i in a)
>     print i, a[i]
> }'
+ 4 4
+ 3 3
$ gawk '
> BEGIN {
>   PROCINFO["sorted_in"] = "@ind_str_asc"
>   a[4] = 4
>   a[3] = 3
>   for (i in a)
>     print i, a[i]
> }'
+ 3 3
+ 4 4
```

Quando si ordina un vettore in base al valore dei suoi elementi, se viene trovato un valore che è un sottovettore, questo è considerato più grande di qualsiasi stringa o valore

---

univoco quando i valori numerici sono uguali assicura che il comportamento di `gawk` sia coerente in differenti ambienti.

numerico, indipendentemente da quel che contiene lo stesso sottovettore, e tutti i sottovettori sono trattati come se fossero l'uno uguale all'altro. Il loro ordine reciproco è determinato dai loro indici, visti come stringhe.

Di seguito sono elencati alcuni punti da tener presenti sulla visita ordinata dei vettori:

- Il valore di `PROCINFO["sorted_in"]` è globale. Cioè, ha effetto su tutti i cicli `for` relativi a qualsiasi vettore. Se si deve cambiarlo all'interno del proprio codice, si dovrebbe vedere se era già stato definito in precedenza, e salvare il valore relativo, per ripristinarlo successivamente:

```
...
if ("sorted_in" in PROCINFO) {
    ordine_salvato = PROCINFO["sorted_in"]
    PROCINFO["sorted_in"] = "@val_str_desc" # o qualcos'altro
}
...
if (ordine_salvato)
    PROCINFO["sorted_in"] = ordine_salvato
```

- Come già accennato, l'ordine di visita di default del vettore è rappresentato da `"@unsorted"`. Si può ottenere il comportamento di default anche assegnando la stringa nulla a `PROCINFO["sorted_in"]` o semplicemente eliminando l'elemento `"sorted_in"` dal vettore `PROCINFO` con l'istruzione `delete`. (L'istruzione `delete` non è stata ancora descritta; si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187.)

Inoltre, `gawk` fornisce funzioni predefinite per l'ordinamento dei vettori; si veda [Sezione 12.2.2 \[Ordinare valori e indici di un vettore con gawk\]](#), pagina 336.

## 8.2 Usare numeri per indicizzare i vettori

Un aspetto importante da ricordare riguardo ai vettori è che *gli indici dei vettori sono sempre stringhe*. Quando un valore numerico è usato come indice, viene convertito in un valore di tipo stringa prima di essere usato per l'indicizzazione (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121). Ciò vuol dire che il valore della variabile predefinita `CONVFMT` può influire su come un programma ha accesso agli elementi di un vettore. Per esempio:

```
xyz = 12.153
dati[xyz] = 1
CONVFMT = "%.2f"
if (xyz in dati)
    printf "%s è in dati\n", xyz
else
    printf "%s non è in dati\n", xyz
```

Il risultato è `'12.15 non è in dati'`. La prima istruzione dà a `xyz` un valore numerico. L'assegnamento a `dati[xyz]` indicizza `dati` col valore di tipo stringa `"12.153"` (usando il valore di conversione di default `CONVFMT`, `"%.6g"`). Quindi, all'elemento del vettore `dati["12.153"]` è assegnato il valore uno. Il programma cambia poi il valore di `CONVFMT`. La verifica `'(xyz in dati)'` genera un nuovo valore di stringa da `xyz`—questa volta `"12.15"`—perché il valore di `CONVFMT` consente solo due cifre decimali. Questa verifica dà esito negativo, perché `"12.15"` è diverso da `"12.153"`.

Secondo le regole di conversione (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), [pagina 121](#)), i valori numerici interi vengono convertiti in stringhe sempre come interi, indipendentemente dal valore che potrebbe avere `CONVFMT`. E infatti il caso seguente produce il risultato atteso:

```
for (i = 1; i <= maxsub; i++)
    fa qualcosa con vettore[i]
```

La regola “i valori numerici interi si convertono sempre in stringhe intere” ha un’altra conseguenza per l’indicizzazione dei vettori. Le costanti ottali ed esadecimali (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), [pagina 115](#)) vengono convertite internamente in numeri, e la loro forma originale non viene più ricordata. Ciò significa, per esempio, che `vettore[17]`, `vettore[021]` e `vettore[0x11]` fanno riferimento tutti allo stesso elemento!

Come molte cose in `awk`, molto spesso le cose funzionano come ci si aspetta. È utile comunque avere una conoscenza precisa delle regole applicate, poiché a volte possono avere effetti difficili da individuare sui programmi.

### 8.3 Usare variabili non inizializzate come indici

Supponiamo che sia necessario scrivere un programma per stampare i dati di input in ordine inverso. Un tentativo ragionevole per far ciò (con qualche dato di prova) potrebbe essere qualcosa di questo tipo:

```
$ echo 'riga 1
> riga 2
> riga 3' | awk '{ l[righe] = $0; ++righe }
> END {
>     for (i = righe - 1; i >= 0; i--)
>         print l[i]
> }'
+ riga 3
+ riga 2
```

Sfortunatamente, la prima riga di dati in input non appare nell’output!

A prima vista, verrebbe da dire che questo programma avrebbe dovuto funzionare. La variabile `righe` non è inizializzata, e le variabili non inizializzate hanno il valore numerico zero. Così, `awk` dovrebbe aver stampato il valore `l[0]`.

Qui il problema è che gli indici per i vettori di `awk` sono *sempre* stringhe. Le variabili non inizializzate, quando sono usate come stringhe, hanno il valore `""`, e non zero. Quindi, `‘riga 1’` finisce per essere memorizzata in `l[""]`. La seguente variante del programma funziona correttamente:

```
{ l[righe++] = $0 }
END {
    for (i = righe - 1; i >= 0; i--)
        print l[i]
}
```

Qui, `‘++’` forza `righe` a essere di tipo numerico, rendendo quindi il “vecchio valore” uno zero numerico. Questo viene poi convertito in `"0"` come l’indice del vettore.



Anche se la cosa può sembrare strana, la stringa nulla ("" ) è un indice di vettore valido. Se viene fornita l'opzione `--lint` sulla riga di comando si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), `gawk` avvisa quando la stringa nulla viene usata come indice.

## 8.4 L'istruzione delete

Per rimuovere un singolo elemento da un vettore si usa l'istruzione `delete`:

```
delete vettore[espressione-indice]
```

Una volta che un elemento di un vettore è stato eliminato, il valore che aveva quell'elemento non è più disponibile. È come se quell'elemento non sia mai stato referenziato oppure come se non gli sia mai stato assegnato un valore. Il seguente è un esempio di eliminazione di elementi da un vettore:

```
for (i in frequenze)
    delete frequenze[i]
```

Quest'esempio rimuove tutti gli elementi dal vettore `frequenze`. Una volta che un elemento è stato eliminato, una successiva istruzione `for` che visiti il vettore non troverà quell'elemento, e l'uso dell'operatore `in` per controllare la presenza di quell'elemento restituisce zero (cioè falso):

```
delete pippo[4]
if (4 in pippo)
    print "Questo non verrà mai stampato"
```

È importante notare che eliminare un elemento *non* è la stessa cosa che assegnargli un valore nullo (la stringa vuota, ""). Per esempio:

```
pippo[4] = ""
if (4 in pippo)
    print "Questo viene stampato, anche se pippo[4] è vuoto"
```

Non è un errore eliminare un elemento che non esiste. Tuttavia, se viene data l'opzione `--lint` sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), `gawk` emette un messaggio di avvertimento quando viene eliminato un elemento che non è presente in un vettore.

Tutti gli elementi di un vettore possono essere eliminati con una singola istruzione omettendo l'indice nell'istruzione `delete`, in questo modo:

```
delete vettore
```

L'uso di questa versione dell'istruzione `delete` è circa tre volte più efficiente dell'equivalente ciclo che elimina gli elementi uno alla volta.

Questa forma dell'istruzione `delete` è ammessa anche da BWK `awk` e da `mawk`, e anche da diverse altre implementazioni.

**NOTA:** Per molti anni, l'uso di `delete` senza un indice era un'estensione comune. A settembre 2012 si è deciso di includerla nello standard POSIX. Si veda [il sito dell'Austin Group](#).

La seguente istruzione fornisce un modo portabile, anche se non evidente, per svuotare un vettore:<sup>3</sup>

```
split("", vettore)
```

<sup>3</sup> Un ringraziamento a Michael Brennan per la segnalazione.

La funzione `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)) dapprima svuota il vettore indicato. La chiamata chiede di dividere la stringa nulla. Poiché non c'è nulla da dividere, la funzione si limita a svuotare il vettore e poi termina.

**ATTENZIONE:** L'eliminazione di tutti gli elementi di un vettore non cambia il suo tipo; non si può svuotare un vettore e poi usare il nome del vettore come scalare (cioè, come una variabile semplice). Per esempio, questo non è consentito:

```
a[1] = 3
delete a
a = 3
```

## 8.5 Vettori multidimensionali

Un *vettore multidimensionale* è un vettore in cui un elemento è identificato da un insieme di indici invece che da un indice singolo. Per esempio, un vettore bidimensionale richiede due indici. Il modo consueto (in molti linguaggi, compreso `awk`) per far riferimento a un elemento di un vettore multidimensionale chiamato *griglia* è con `griglia[x,y]`.

I vettori multidimensionali sono resi disponibili in `awk` attraverso la concatenazione di più indici in una stringa; `awk` converte gli indici in stringhe (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), [pagina 121](#)) e le concatena assieme, inserendo un separatore tra ognuna di loro. Ne risulta una sola stringa che include i valori di ogni indice. La stringa così composta viene usata come un singolo indice in un vettore ordinario monodimensionale. Il separatore usato è il valore della variabile predefinita `SUBSEP`.

Per esempio, supponiamo di valutare l'espressione `'pippo[5,12] = "valore"'` quando il valore di `SUBSEP` è `"@"`. I numeri 5 e 12 vengono convertiti in stringhe che sono poi concatenate con un `'@'` tra di essi, dando `"5@12"`; di conseguenza, l'elemento del vettore `pippo["5@12"]` è impostato a `"valore"`.

Una volta che il valore dell'elemento è memorizzato, `awk` ignora se sia stato memorizzato con un solo indice o con una serie di indici. Le due espressioni `'pippo[5,12]'` e `'pippo[5 SUBSEP 12]'` sono sempre equivalenti.

Il valore di default di `SUBSEP` è la stringa `"\034"`, che contiene un carattere non stampabile che difficilmente appare in un programma di `awk` e nella maggior parte dei dati di input. Il vantaggio di scegliere un carattere improbabile discende dal fatto che i valori degli indici che contengono una stringa corrispondente a `SUBSEP` possono portare a stringhe risultanti ambigue. Supponendo che `SUBSEP` valga `"@"`, `'pippo["a@b", "c"]'` e `'pippo["a", "b@c"]'` risultano indistinguibili perché entrambi sarebbero in realtà memorizzati come `'pippo["a@b@c"]'`.

Per verificare se una determinata sequenza di indici esiste in un vettore multidimensionale, si usa lo stesso operatore (`in`) che viene usato per i vettori monodimensionali. Si scrive l'intera sequenza di indici tra parentesi, separati da virgole, come operando di sinistra:

```
if ((indice1, indice2, ...) in vettore)
    ...
```

Qui vediamo un esempio, avendo in input un vettore bidimensionale di campi, ruota questo vettore di 90 gradi in senso orario e stampa il risultato. Si suppone che tutte le righe in input contengano lo stesso numero di elementi:

```

{
    if (max_nf < NF)
        max_nf = NF
    max_nr = NR
    for (x = 1; x <= NF; x++)
        vettore[x, NR] = $x
}

END {
    for (x = 1; x <= max_nf; x++) {
        for (y = max_nr; y >= 1; --y)
            printf("%s ", vettore[x, y])
        printf("\n")
    }
}

```

Dato l'input:

```

1 2 3 4 5 6
2 3 4 5 6 1
3 4 5 6 1 2
4 5 6 1 2 3

```

il programma produce il seguente output:

```

4 3 2 1
5 4 3 2
6 5 4 3
1 6 5 4
2 1 6 5
3 2 1 6

```

### 8.5.1 Visitare vettori multidimensionali

Non c'è un'istruzione `for` particolare per visitare un vettore “multidimensionale”. Non ce ne può essere una, perché `awk` in realtà non ha vettori o elementi multidimensionali: c'è solo una modalità multidimensionale per *accedere* a un vettore.

Comunque, se un programma ha un vettore al quale si accede sempre in modalità multidimensionale, si può ottenere il risultato di visitarlo combinando l'istruzione di visita `for` (si veda la [Sezione 8.1.5 \[Visitare tutti gli elementi di un vettore\]](#), pagina 181) con la funzione interna `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Si procede nel seguente modo:

```

for (indice_combinato in vettore) {
    split(indice_combinato, indici_separati, SUBSEP)
    ...
}

```

Questo imposta la variabile `indice_combinato` a ogni concatenazione di indici contenuta nel vettore, e la suddivide nei singoli indici separandoli in corrispondenza del valore di `SUBSEP`. I singoli indici diventano poi gli elementi del vettore `indici_separati`.

Perciò, se un valore è stato precedentemente memorizzato in `vettore[1, "pippo"]`, esiste in `vettore` un elemento con indice `"1\034pippo"` (ricordare che il valore di default di `SUBSEP` è il carattere con codice ottale 034). Prima o poi, l'istruzione `for` trova quell'indice e fa un'iterazione con la variabile `indice_combinato` impostata a `"1\034pippo"`. Poi viene chiamata la funzione `split()` in questo modo:

```
split("1\034pippo", indici_separati, "\034")
```

Il risultato è quello di impostare `indici_separati[1]` a `"1"` e `indici_separati[2]` a `"pippo"`. Ecco fatto! La sequenza originale degli indici separati è ripristinata.

## 8.6 Vettori di vettori

`gawk` migliora l'accesso ai vettori multidimensionali di `awk` standard e mette a disposizione dei veri vettori di vettori. Agli elementi di un sottovettore si fa riferimento tramite il loro indice racchiuso tra parentesi quadre, proprio come gli elementi del vettore principale. Per esempio, quel che segue crea un sottovettore con due elementi all'indice 1 del vettore principale `a`:

```
a[1][1] = 1
a[1][2] = 2
```

Questo simula un vero vettore bidimensionale. Ogni elemento di un sottovettore può contenere un altro sottovettore come valore, che a sua volta può contenere anche ulteriori vettori. In questo modo, si possono creare vettori di tre o più dimensioni. Gli indici possono essere costituiti da qualunque espressione di `awk`, compresi dei valori scalari separati da virgole (cioè, un indice multidimensionale simulato di `awk`). Quindi, la seguente espressione è valida in `gawk`:

```
a[1][3][1, "nome"] = "barney"
```

Ogni sottovettore e il vettore principale possono essere di diversa lunghezza. Di fatto, gli elementi di un vettore o un suo sottovettore non devono essere necessariamente tutti dello stesso tipo. Ciò significa che il vettore principale come anche uno qualsiasi dei suoi sottovettori può essere non rettangolare, o avere una struttura frastagliata. Si può assegnare un valore scalare all'indice 4 del vettore principale `a`, anche se `a[1]` è esso stesso un vettore e non uno scalare:

```
a[4] = "Un elemento in un vettore frastagliato"
```

I termini *dimensione*, *riga* e *colonna* sono privi di significato quando sono applicati a questo tipo di vettore, ma d'ora in poi useremo “dimensione” per indicare il numero massimo di indici necessario per far riferimento a un elemento esistente. Il tipo di ogni elemento che è già stato assegnato non può essere cambiato assegnando un valore di tipo diverso. Prima si deve eliminare l'elemento corrente, per togliere completamente dalla memoria di `gawk` ogni riferimento a quell'indice:

```
delete a[4]
a[4][5][6][7] = "Un elemento in un vettore quadridimensionale"
```

Le due istruzioni rimuovono il valore scalare dall'indice 4 e inseriscono poi un sottovettore interno a tre indici contenente uno scalare. Si può anche eliminare un intero sottovettore o un sottovettore di sottovettori:

```
delete a[4][5]
a[4][5] = "Un elemento nel sottovettore a[4]"
```

Si deve però ricordare che non è consentito eliminare il vettore principale `a` e poi usarlo come scalare.

Le funzioni predefinite che accettano come argomenti dei vettori possono essere usate anche con i sottovettori. Per esempio, il seguente frammento di codice usa `length()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)) per determinare il numero di elementi nel vettore principale `a` e nei suoi sottovettori:

```
print length(a), length(a[1]), length(a[1][3])
```

Il risultato per il nostro vettore principale `a` è il seguente:

```
2, 3, 1
```

L'espressione '*indice in vettore*' (si veda la [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\], pagina 179](#)) funziona allo stesso modo sia per i vettori regolari in stile `awk` che per i vettori di vettori. Per esempio, le espressioni '`1 in a`', '`3 in a[1]`' e '`(1, "nome") in a[1][3]`' risultano tutte di valore uno (vero) per il nostro vettore `a`.

L'istruzione '*for (elemento in vettore)*' (si veda la [Sezione 8.1.5 \[Visitare tutti gli elementi di un vettore\], pagina 181](#)) può essere nidificata per visitare tutti gli elementi di un vettore di vettori che abbia una struttura rettangolare. Per stampare il contenuto (valori scalari) di un vettore di vettori bidimensionale (cioè nel quale ogni elemento di primo livello è esso stesso un vettore, non necessariamente di lunghezza uguale agli altri) si può usare il seguente codice:

```
for (i in vettore)
    for (j in vettore[i])
        print vettore[i][j]
```

La funzione `isarray()` (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\], pagina 223](#)) permette di verificare se un elemento di un vettore è esso stesso un vettore:

```
for (i in vettore) {
    if (isarray(vettore[i])) {
        for (j in vettore[i]) {
            print vettore[i][j]
        }
    }
    else
        print vettore[i]
}
```

Se la struttura di un vettore di vettori frastagliato è nota in anticipo, si può spesso trovare il modo per visitarlo usando istruzioni di controllo. Per esempio, il seguente codice stampa gli elementi del nostro vettore principale `a`:

```
for (i in a) {
    for (j in a[i]) {
        if (j == 3) {
            for (k in a[i][j])
                print a[i][j][k]
        } else
            print a[i][j]
    }
}
```

```
    }
}
```

Si veda la [Sezione 10.7 \[Attraversare vettori di vettori\]](#), pagina 277, per una funzione definita dall'utente che "visita" un vettore di vettori di dimensioni arbitrarie.

Si ricordi che un riferimento a un elemento di un vettore non inizializzato genera un elemento con valore uguale a "", la stringa nulla. Questo ha un'importante implicazione quando s'intende usare un sottovettore come argomento di una funzione, come illustrato nel seguente esempio:

```
$ gawk 'BEGIN { split("a b c d", b[1]); print b[1][1] }'
error gawk: riga com.:1: fatale: split: secondo argomento
error non-vettoriale
```

Il modo per aggirare quest'ostacolo è quello di definire prima `b[1]` come vettore creando un indice arbitrario:

```
$ gawk 'BEGIN { b[1][1] = ""; split("a b c d", b[1]); print b[1][1] }'
+ a
```

## 8.7 Sommario

- `awk` standard dispone di vettori associativi monodimensionali (vettori indicizzati da valori di tipo stringa). Tutti i vettori sono associativi; gli indici numerici vengono convertiti automaticamente in stringhe.
- Agli elementi dei vettori si fa riferimento come `vettore[indice]`. Fare riferimento a un elemento lo crea se questo non esiste ancora.
- Il modo corretto per vedere se un vettore ha un elemento con un dato indice è quello di usare l'operatore `in`: `'indice in vettore'`.
- Si usa `'for (indice in vettore) ...'` per visitare ogni singolo elemento di un vettore. Nel corpo del ciclo, `indice` assume via via il valore dell'indice di ogni elemento del vettore.
- L'ordine in cui il ciclo `'for (indice in vettore)'` attraversa un vettore non è definito in POSIX `awk` e varia a seconda dell'implementazione. `gawk` consente di controllare l'ordinamento di visita assegnando speciali valori predefiniti a `PROCINFO["sorted_in"]`.
- Si usa `'delete vettore[indice]'` per eliminare un singolo elemento di un vettore. Per eliminare tutti gli elementi di un vettore, si usa `'delete vettore'`. Quest'ultima funzionalità è stata per molti anni un'estensione comune e ora è standard, ma potrebbe non essere disponibile in tutte le versioni commerciali di `awk`.
- `awk` standard simula vettori multidimensionali ammettendo più indici separati da virgole. I loro valori sono concatenati in un'unica stringa, separati dal valore di `SUBSEP`. Il modo di creazione dell'indice non viene immagazzinato; così, cambiare `SUBSEP` potrebbe avere conseguenze inaspettate. Si può usare `'(sub1, sub2, ...) in vettore'` per vedere se un certo indice multidimensionale esiste in `vettore`.
- `gawk` consente di avere a disposizione veri vettori di vettori. Si usa una coppia di parentesi quadre per ogni dimensione in tali vettori: `dati[riga][colonna]`, per esempio. Gli elementi del vettore possono poi essere valori scalari (numeri o stringhe) o altri vettori.

- Si usa la funzione predefinita `isarray()` per determinare se un elemento di un vettore è esso stesso un sottovettore.



## 9 Funzioni

Questo capitolo descrive le funzioni predefinite di **awk**, che sono di tre tipi: numeriche, di stringa, e di I/O. **gawk** mette a disposizione ulteriori tipi di funzioni per gestire valori che rappresentano marcature temporali, per manipolare bit, per ordinare vettori, per fornire informazioni sui tipi di variabile, per internazionalizzare e localizzare i programmi.<sup>1</sup>

Oltre alle funzioni predefinite, **awk** consente di scrivere nuove funzioni utilizzabili all'interno di un programma. La seconda metà di questo capitolo descrive le funzioni *definite dall'utente*. Vengono infine descritte le chiamate indirette a una funzione, un'estensione specifica di **gawk** che consente di stabilire durante l'esecuzione del programma quale funzione chiamare.

### 9.1 Funzioni predefinite

Le funzioni *predefinite* sono sempre disponibili per essere chiamate da un programma **awk**. Questa sezione definisce tutte le funzioni predefinite di **awk**; di alcune di queste si fa menzione in altre sezioni, ma sono comunque riassunte anche qui per comodità.

#### 9.1.1 Chiamare funzioni predefinite

Per chiamare una delle funzioni predefinite di **awk**, si scrive il nome della funzione seguito dai suoi argomenti racchiusi tra parentesi. Per esempio, `'atan2(y + z, 1)'` è una chiamata alla funzione `atan2()` e ha due argomenti.

La presenza di spazi bianchi tra il nome della funzione predefinita e la parentesi aperta è consentita, ma è buona norma quella di evitare di inserire spazi bianchi in quella posizione. Le funzioni definite dall'utente non consentono che vi siano spazi bianchi fra nome funzione e aperta parentesi, ed è più semplice evitare errori seguendo una semplice convenzione che resta sempre valida: non inserire spazi dopo il nome di una funzione.

Ogni funzione predefinita accetta un certo numero di argomenti. In alcuni casi, gli argomenti possono essere omessi. I valori di default per gli argomenti omessi variano da funzione a funzione e sono descritti insieme a ciascuna funzione. In alcune implementazioni di **awk**, gli eventuali argomenti in più specificati per le funzioni predefinite sono ignorati. Tuttavia, in **gawk**, è un errore fatale fornire argomenti in più a una funzione predefinita.

Quando si richiama una funzione viene calcolato, prima di effettuare la chiamata, il valore assunto dalle espressioni che descrivono i parametri da passare alla funzione. Per esempio, nel seguente frammento di codice:

```
i = 4
j = sqrt(i++)
```

la variabile `i` è incrementata al valore cinque prima di chiamare la funzione `sqrt()` alla quale viene fornito come parametro il valore quattro. L'ordine di valutazione delle espressioni usate come parametri per la funzione è indefinito. Per questo motivo, si deve evitare di scrivere programmi che presuppongono che i parametri siano valutati da sinistra a destra o da destra a sinistra. Per esempio:

```
i = 5
```

---

<sup>1</sup> Per un'introduzione alle tematiche suddette, si può consultare l'articolo "Localizzazione dei programmi" nel [sito pluto.it](http://pluto.it).

```
j = atan2(++i, i *= 2)
```

Se l'ordine di valutazione è da sinistra a destra, `i` assume dapprima il valore 6, e quindi il valore 12, e la funzione `atan2()` è chiamata con i due argomenti 6 e 12. Ma se l'ordine di valutazione è da destra a sinistra, `i` assume dapprima il valore 10, e poi il valore 11, e la funzione `atan2()` è chiamata con i due argomenti 11 e 10.

### 9.1.2 Funzioni numeriche

La seguente lista descrive tutte le funzioni predefinite che hanno a che fare con i numeri. I parametri facoltativi sono racchiusi tra parentesi quadre ([ ]):

- atan2(y, x)** Restituisce l'arcotangente di `y / x` in radianti. Si può usare `'pi = atan2(0, -1)'` per ottenere il valore di  $\pi$  greco.
- cos(x)** Restituisce il coseno di `x`, con `x` in radianti.
- exp(x)** Restituisce l'esponenziale di `x` ( $e^x$ ) o un messaggio di errore se `x` è fuori dall'intervallo consentito. L'intervallo entro il quale può variare `x` dipende dalla rappresentazione dei numeri in virgola mobile nella macchina in uso.
- int(x)** Restituisce l'intero più vicino a `x`, situato tra `x` e zero, troncato togliendo i decimali. Per esempio, `int(3)` è 3, `int(3.9)` è 3, `int(-3.9)` è -3, e `int(-3)` è ancora -3.
- intdiv(numeratore, denominatore, risultato)** Esegue una divisione tra numeri interi, simile alla funzione standard C che ha lo stesso nome. Dapprima, il `numeratore` e il `denominatore` vengono troncati, eliminando la parte decimale, per trasformarli in numeri interi. Il vettore `risultato` viene dapprima svuotato, e poi viene impostato l'elemento `risultato["quotient"]` al risultato della divisione `'numeratore / denominatore'`, troncato a numero intero mediante l'eliminazione dei decimali, e viene impostato l'elemento `risultato["remainder"]` al risultato dell'operazione `'numeratore % denominatore'`, troncato a numero intero allo stesso modo del risultato. Questa funzione è rivolta principalmente a chi usa numeri interi di lunghezza arbitraria; consente di evitare la creazione di numeri in virgola mobile di precisione arbitraria usando la funzionalità MPFR (si veda la [Sezione 15.5 \[Aritmetica dei numeri interi a precisione arbitraria con gawk\]](#), [pagina 389](#)).  
Questa funzione è un'estensione **gawk**. Non è disponibile in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)).
- log(x)** Restituisce il logaritmo naturale di `x`, se `x` è positivo; altrimenti, restituisce NaN ("not a number") sui sistemi che implementano lo standard IEEE 754. Inoltre, **gawk** stampa un messaggio di avvertimento qualora `x` sia negativo.
- rand()** Restituisce un numero casuale. I valori di `rand()` sono uniformemente distribuiti tra zero e uno. Il valore potrebbe essere zero ma non è mai uno.<sup>2</sup>

<sup>2</sup> La versione C di `rand()` in molti sistemi Unix produce notoriamente delle sequenze piuttosto mediocri di numeri casuali. Tuttavia, non è prescritto che un'implementazione di **awk** debba usare la funzione `rand()` del linguaggio C per implementare la versione **awk** di `rand()`. In effetti, **gawk** usa, per generare numeri casuali, la funzione `random()` di BSD, che è notevolmente migliore di `rand()`

Spesso servono dei numeri casuali interi invece che frazionari. La seguente funzione definita dall'utente può essere usata per ottenere un numero casuale non negativo inferiore a  $n$ :

```
function randint(n)
{
    return int(n * rand())
}
```

La moltiplicazione produce un numero casuale maggiore o uguale a zero e minore di  $n$ . Tramite `int()`, questo risultato diventa un intero tra zero e  $n - 1$ , estremi inclusi.

Il seguente esempio usa una funzione simile per generare interi casuali fra uno e  $n$ . Il programma stampa un numero casuale per ogni record in input:

```
# funzione per simulare un tiro di dado.
function roll(n) { return 1 + int(rand() * n) }

# Tira 3 dadi a sei facce e
# stampa il numero di punti.
{
    printf("%d punteggio\n", roll(6) + roll(6) + roll(6))
}
```

**ATTENZIONE:** Nella maggior parte delle implementazioni di `awk`, compreso `gawk`, `rand()` inizia a generare numeri casuali partendo sempre dallo stesso numero, o *seme*, per ogni invocazione di `awk`.<sup>3</sup> È per questo motivo che un programma genera sempre gli stessi risultati ogni volta che lo si esegue. I numeri sono casuali all'interno di una singola esecuzione di `awk` ma "prevedibili" in ogni successiva esecuzione. Ciò torna utile in fase di test, ma se si desidera che un programma generi sequenze differenti di numeri casuali ogni volta che è chiamato, occorre impostare il seme a un valore che cambi per ogni esecuzione. Per fare questo, è prevista la funzione `srand()`.

**sin(x)** Restituisce il seno di  $x$ , con  $x$  espresso in radianti.

**sqrt(x)** Restituisce la radice quadrata positiva di  $x$ . `gawk` stampa un messaggio di avvertimento se  $x$  è un numero negativo. Quindi, `sqrt(4)` vale 2.

**srand([x])** Imposta al valore  $x$  il numero di partenza, o seme, utilizzato per generare numeri casuali.

Ogni seme genera una sequenza particolare di numeri casuali.<sup>4</sup> Quindi, impostando il seme allo stesso valore una seconda volta, viene prodotta ancora la stessa sequenza di numeri casuali.

<sup>3</sup> `mawk` usa un seme differente ogni volta.

<sup>4</sup> I numeri casuali generati da un computer non sono veramente casuali. Tecnicamente sono conosciuti come numeri *pseudo-casuali*. Ciò vuol dire che, anche se i numeri in una sequenza sembrano casuali, è possibile in realtà generare la stessa sequenza di numeri casuali più e più volte.

**ATTENZIONE:** Differenti implementazioni di `awk` usano internamente differenti generatori di numeri casuali. Non si deve dare per scontato che lo stesso programma `awk` generi la stessa serie di numeri casuali se viene eseguito da differenti versioni di `awk`.

Se si omette l'argomento `x`, scrivendo `'srand()'`, viene usato come seme la data e ora corrente. È questo il modo per ottenere numeri casuali che sono veramente imprevedibili.

Il valore restituito da `srand()` è quello del seme precedente. Questo per facilitare il monitoraggio dei semi, nel caso occorra riprodurre in maniera coerente delle sequenze di numeri casuali.

POSIX non specifica quale debba essere il seme iniziale, che quindi varia a seconda delle implementazioni `awk`.

### 9.1.3 Funzioni di manipolazione di stringhe

Le funzioni in questa sezione leggono o modificano il testo di una o più stringhe.

`gawk` implementa la localizzazione (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), [pagina 141](#)) ed effettua ogni manipolazione di stringhe trattando ogni singolo *carattere*, non ogni singolo *byte*. Questa distinzione è particolarmente importante da comprendere per quelle localizzazioni in cui un singolo carattere può essere rappresentato da più di un byte. Quindi, per esempio, la funzione `length()` restituisce il numero di caratteri in una stringa, e non il numero di byte usato per rappresentare quei caratteri. Allo stesso modo, `index()` restituisce indici di caratteri, e non indici di byte.

**ATTENZIONE:** Un certo numero di funzioni riguarda indici all'interno di stringhe. Per queste funzioni, il primo carattere di una stringa è alla posizione (all'indice) uno. Questo comportamento è differente da quello del C e dei linguaggi che da esso discendono, nei quali il primo carattere è alla posizione zero. È importante ricordarlo quando si fanno calcoli sugli indici, in particolare se si ha familiarità con il linguaggio C.

Nella lista seguente, i parametri facoltativi sono racchiusi tra parentesi quadre (`[ ]`). Parecchie funzioni operano sostituzioni in una stringa; la spiegazione completa di ciò è contenuta nella descrizione della funzione `sub()`, che si trova quasi alla fine di questa lista, ordinata alfabeticamente.

Le funzioni specifiche di `gawk` sono contrassegnate col simbolo del cancelletto (`#`). Tali funzioni non sono disponibili in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)):

```
asort(sorgente [, destinazione [, come ]]) #
asorti(sorgente [, destinazione [, come ]]) #
```

Queste due funzioni sono abbastanza simili, e quindi sono descritte insieme.

**NOTA:** La seguente descrizione ignora il terzo argomento, *come*, perché richiede la conoscenza di funzionalità di cui non si è ancora parlato. Per questo motivo la seguente trattazione è volutamente semplificata. (In seguito l'argomento verrà trattato in maniera più esauriente; si veda [Sezione 12.2.2 \[Ordinare valori e indici di un vettore con gawk\]](#), [pagina 336](#), per la descrizione completa.)

Entrambe le funzioni restituiscono il numero di elementi nel vettore *sorgente*. Con `asort()`, `gawk` ordina i valori di *sorgente* e rimpiazza gli indici dei valori ordinati di *sorgente* con numeri interi sequenziali, a partire da uno. Se si specifica il vettore opzionale *destinazione*, *sorgente* è copiato in *destinazione*. *destinazione* viene quindi ordinato, lasciando immutati gli indici di *sorgente*. Nel confronto tra stringhe, la variabile `IGNORECASE` influenza l'ordinamento (si veda la [Sezione 12.2.2 \[Ordinare valori e indici di un vettore con gawk\]](#), [pagina 336](#)). Se il vettore *sorgente* contiene sottovettori come valori (si veda la [Sezione 8.6 \[Vettori di vettori\]](#), [pagina 190](#)), questi saranno alla fine, dopo tutti i valori scalari. I sottovettori *non* vengono ordinati ricorsivamente.

Per esempio, se i contenuti del vettore `a` sono i seguenti:

```
a["ultimo"] = "de"
a["primo"] = "sac"
a["mediano"] = "cul"
```

Una chiamata a `asort()`:

```
asort(a)
```

genera i seguenti contenuti di `a`:

```
a[1] = "cul"
a[2] = "de"
a[3] = "sac"
```

La funzione `asorti()` si comporta in maniera simile ad `asort()`; tuttavia l'ordinamento avviene in base agli *indici*, e non in base ai valori. Quindi, nell'esempio seguente, a partire dallo stesso insieme iniziale di indici e valori nel vettore `a`, la chiamata di `'asorti(a)'` produrrebbe:

```
a[1] = "mediano"
a[2] = "primo"
a[3] = "ultimo"
```

`gensub(regex, rimpiazzo, come [, obiettivo]) #`

Ricerca nella stringa *obiettivo* delle corrispondenze all'espressione regolare *regex*. Se *come* è una stringa che inizia con `'g'` o `'G'` (abbreviazione di "global"), sostituisce ogni occorrenza di *regex* con la stringa *rimpiazzo*. Altrimenti, *come* è visto come un numero che indica quale corrispondenza di *regex* va rimpiazzata. Se non si specifica il nome dell'*obiettivo*, si opera su `$0`. La funzione restituisce come risultato la stringa modificata, e la stringa originale di partenza *non* viene modificata.

`gensub()` è una funzione generale di sostituzione. Mira a fornire più funzionalità rispetto alle funzioni standard `sub()` e `gsub()`.

`gensub()` prevede una funzionalità ulteriore, non disponibile in `sub()` o `gsub()`: la possibilità di specificare componenti di una *regex* nel testo da sostituire. Questo è fatto utilizzando delle parentesi nella *regex* per designare i componenti, e quindi inserendo `'\N'` nel testo di rimpiazzo, dove *N* è una cifra da 1 a 9. Per esempio:

```
$ gawk '
> BEGIN {
```

```

>      a = "abc def"
>      b = gensub(/(.+) (.+)/, "\\2 \\1", "g", a)
>      print b
> }'
└─ def abc

```

Come con `sub()`, occorre battere due barre inverse, per ottenerne una come componente della stringa. Nel testo di rimpiazzo, la sequenza `'\0'` rappresenta l'intero testo corrispondente, e lo stesso vale per il carattere `'&'`.

Il seguente esempio mostra come è possibile usare il terzo argomento per controllare quale corrispondenza della *regexp* sia da modificare:

```

$ echo a b c a b c |
> gawk '{ print gensub(/a/, "AA", 2) }'
└─ a b c AA b c

```

In questo caso, `$0` è la stringa obiettivo di default. `gensub()` restituisce la nuova stringa come risultato, e questa è passata direttamente a `print` per essere stampata.

Se l'argomento *come* è una stringa che non inizia con `'g'` o `'G'`, o se è un numero minore o uguale a zero, si effettua solo una sostituzione. Se *come* è zero, `gawk` emette un messaggio di avvertimento.

Se *regexp* non viene trovata in *obiettivo*, il valore restituito da `gensub()` è il valore originale e non modificato di *obiettivo*.

**gsub(*regexp*, *rimpiazzo* [, *obiettivo*])**

Ricerca in *obiettivo* tutte le sottostringhe corrispondenti al criterio di ricerca, le più lunghe possibili partendo da sinistra, *non sovrapposte tra loro*, e le sostituisce con *rimpiazzo*. La lettera `'g'` in `gsub()` significa "global", e richiede di sostituire dappertutto. Per esempio:

```
{ gsub(/Inghilterra/, "Regno Unito"); print }
```

sostituisce tutte le occorrenze della stringa `'Inghilterra'` con `'Regno Unito'` in tutti i record in input.

La funzione `gsub()` restituisce il numero di sostituzioni effettuate. Se la variabile da cercare e modificare (*obiettivo*) è omessa, viene usato l'intero record in input. Come in `sub()`, i caratteri `'&'` e `'\'` sono speciali, e il terzo argomento dev'essere modificabile.

**index(*dove*, *cosa*)**

Ricerca nella stringa *dove* la prima occorrenza della stringa *cosa*, e restituisce la posizione in caratteri dell'inizio di quest'occorrenza nella stringa *dove*. Si consideri il seguente esempio:

```

$ awk 'BEGIN { print index("noccioline", "oli") }'
└─ 6

```

Se *cosa* non viene trovato, `index()` restituisce zero.

In BWK `awk` e `gawk`, è un errore fatale usare una costante *regexp* per *cosa*. Altre implementazioni lo consentono, considerando semplicemente la costante *regexp* come un'espressione che significa `'$0 ~ /regexp/'`.



`length([stringa])`

Restituisce il numero di caratteri in *stringa*. Se *stringa* è un numero, viene restituita la lunghezza della stringa di cifre che rappresenta quel numero. Per esempio, `length("abcde")` è cinque. Invece, `length(15 * 35)` restituisce tre. In questo esempio,  $15 \cdot 35 = 525$ , e 525 è quindi convertito alla stringa "525", che è composta da tre caratteri.

Se non si specifica alcun argomento, `length()` restituisce la lunghezza di `$0`.

**NOTA:** In alcune delle prime versioni di `awk`, la funzione `length()` poteva essere richiamata senza alcuna parentesi. Farlo è considerata una cattiva abitudine, sebbene il POSIX standard 2008 lo consenta esplicitamente, per compatibilità con la vecchia prassi. Per garantire la massima portabilità ai programmi, è meglio mettere sempre le parentesi.

Se `length()` è chiamata con una variabile che non è stata usata, `gawk` considera la variabile come uno scalare. Altre implementazioni di `awk` non assegnano nessun tipo alla variabile. Si consideri:

```
$ gawk 'BEGIN { print length(x) ; x[1] = 1 }'
+ 0
error gawk: riga com.:1: fatale: tentativo di usare
error scalare 'x' come vettore

$ nawk 'BEGIN { print length(x) ; x[1] = 1 }'
+ 0
```

Se `--lint` è stato specificato sulla riga di comando, `gawk` emette un avvertimento a questo riguardo.

In `gawk` e in parecchie altre implementazioni `awk`, se l'argomento è un vettore, la funzione `length()` restituisce il numero di elementi nel vettore. (e.c.) Ciò è meno utile di quel che sembra a prima vista, in quanto non è affatto detto che il vettore abbia come indici i numeri da 1 al numero di elementi che contiene. Se `--lint` è stato specificato sulla riga di comando, (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), `gawk` avvisa che l'uso di un vettore come argomento non è portabile. Se si specifica l'opzione `--posix`, l'uso di un vettore come argomento genera un errore fatale (si veda il [Capitolo 8 \[Vettori in awk\]](#), pagina 177).

`match(stringa, regexp [, vettore])`

Ricerca in *stringa* la sottostringa più lunga, a partire da sinistra, che corrisponde all'espressione regolare *regexp* e restituisce la posizione del carattere (indice) con cui inizia la sottostringa (uno, se la corrispondenza parte dall'inizio di *stringa*). Se non viene trovata alcuna corrispondenza, restituisce zero.

L'argomento *regexp* può essere sia una costante *regexp* (`/.../`) che una costante stringa (`"..."`). In quest'ultimo caso, la stringa è trattata come una *regexp* per la quale cercare una corrispondenza. Si veda la [Sezione 3.6 \[Usare regexp dinamiche\]](#), pagina 57, per una spiegazione sulla differenza tra le due forme e sulle loro implicazioni riguardo al modo per scrivere correttamente un programma.



L'ordine dei primi due argomenti è l'opposto di molte altre funzioni che trattano stringhe e che hanno a che fare con espressioni regolari, come `sub()` e `gsub()`. Potrebbe essere di aiuto ricordare che per `match()`, l'ordine è lo stesso che per l'operatore `'~'`: `'stringa ~ regexp'`.

La funzione `match()` imposta la variabile predefinita `RSTART` all'indice. Imposta anche la variabile predefinita `RLENGTH` alla lunghezza in caratteri della sottostringa individuata. Se non viene trovata alcuna corrispondenza, `RSTART` è impostata a zero, e `RLENGTH` a `-1`.

Per esempio:

```
{
    if ($1 == "TROVA")
        regexp = $2
    else {
        dove = match($0, regexp)
        if (dove != 0)
            print "Corrispondenza di", regexp, "alla posiz.", \
                dove, "in", $0
    }
}
```

Questo programma ricerca delle righe che corrispondono all'espressione regolare contenuta nella variabile `regexp`. Quest'espressione regolare può essere modificata. Se la prima parola in una riga è `'TROVA'`, `regexp` diventa la seconda parola su quella riga. Quindi, dato:

```
TROVA or+e
Il mio programma corre
ma non troppo velocemente
TROVA Melvin
JF+KM
Questa riga appartiene a Reality Engineering Co.
Melvin è passato da qui.
```

`awk` stampa:

```
Corrispondenza di or+e alla posiz. 19 in Il mio programma corre
Corrispondenza di Melvin alla posiz. 1 in Melvin è passato da qui.
```

Se `vettore` esiste già, viene cancellato, e quindi l'elemento numero zero di `vettore` è impostato all'intera parte di `stringa` individuata da `regexp`. Se `regexp` contiene parentesi, gli elementi aventi per indici numeri interi in `vettore` sono impostati per contenere ognuno la parte di `stringa` individuata dalla corrispondente sottoespressione delimitata da parentesi. Per esempio:

```
$ echo pippooooooperpluttttttt |
> gawk '{ match($0, /(pippo+).(plut*)/, vett)
>         print vett[1], vett[2] }'
+ pippooooo pluttttttt
```

Inoltre, sono disponibili indici multidimensionali che contengono la posizione di partenza e la lunghezza di ogni sottoespressione individuata:

```
$ echo pippooooooperpluttttttt |
```

```

> gawk '{ match($0, /(pippo+).(plut*)/, vett)
>         print vett[1], vett[2]
>         print vett[1, "start"], vett[1, "length"]
>         print vett[2, "start"], vett[2, "length"]
> }'
+ pippooooo pluttttttt
+ 1 8
+ 14 10

```

Possono non esserci indici che individuino inizio e posizione per ogni sottoespressione fra parentesi, perché non tutte potrebbero aver individuato del testo; quindi, andrebbero esaminati usando l'operatore `in` (si veda la [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\]](#), pagina 179).

L'argomento `vettore` di `match()` è un'estensione `gawk`. In modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33), l'impiego di un terzo argomento causa un errore fatale.

```

patsplit(stringa, vettore [, regexprdelim [, separatori ]]) #

```

Divide *stringa* in parti definite da *regexprdelim* e memorizza i pezzi in *vettore* e le stringhe di separazione nel vettore *separatori*. Il primo pezzo è memorizzato in *vettore*[1], il secondo pezzo in *vettore*[2], e così via. Il terzo argomento, *regexprdelim*, è una *regexp* che descrive i campi in *stringa* (allo stesso modo in cui `FPAT` è una *regexp* che descrive i campi nei record in input). Può essere una costante *regexp* o una stringa. Se *regexprdelim* è omesso, viene usato il valore di `FPAT`. `patsplit()` restituisce il numero di elementi creati. *separatori*[*i*] è la stringa che separa l'elemento *vettore*[*i*] e *vettore*[*i*+1]. Ogni separatore iniziale sarà in *separatori*[0].

La funzione `patsplit()` divide delle stringhe in pezzi in modo simile a quello con cui le righe in input vengono divise in campi usando `FPAT` (si veda la [Sezione 4.7 \[Definire i campi in base al contenuto\]](#), pagina 79).

Prima di dividere la stringa, `patsplit()` cancella ogni elemento che fosse eventualmente presente nei vettori *vettore* e *separatori*.

```

split(stringa, vettore [, separacampo [, separatori ]])

```

Divide *stringa* in pezzi separati da *separacampo* e memorizza i pezzi in *vettore* e le stringhe di separazione nel vettore *separatori*. Il primo pezzo è memorizzato in *vettore*[1], il secondo pezzo in *vettore*[2], e così via. Il valore della stringa specificata nel terzo argomento, *separacampo*, è una *regexp* che indica come dividere *stringa* (analogamente a come `FS` può essere un *regexp* che indica dove dividere i record in input). Se *separacampo* è omesso, si usa il valore di `FS`. `split()` restituisce il numero di elementi creati. *separatori* è un'estensione `gawk`, in cui *separatori*[*i*] è la stringa che separa *vettore*[*i*] e *vettore*[*i*+1]. Se *separacampo* è uno spazio bianco, ogni eventuale spazio bianco a inizio stringa viene messo in *separatori*[0] e ogni eventuale spazio bianco a fine stringa viene messo in *separatori*[*n*], dove *n* è il valore restituito da `split()` (cioè il numero di elementi in *vettore*).

La funzione `split()` divide le stringhe in pezzi in modo simile a quello con cui le righe in input sono divise in campi. Per esempio:

```
split("cul-de-sac", a, "-", separatori)
```

divide la stringa "cul-de-sac" in tre campi usando '-' come separatore. Il vettore *a* ha i seguenti contenuti:

```
a[1] = "cul"
a[2] = "de"
a[3] = "sac"
```

e imposta il contenuto del vettore *separatori* come segue:

```
seps[1] = "-"
seps[2] = "-"
```

Il valore restituito da questa chiamata a `split()` è tre.

Come nella divisione in campi dei record in input, quando il valore di *separacampo* è " ", gli spazi bianchi a inizio e fine stringa vengono ignorati nell'assegnare valori agli elementi di *vettore* ma non nel vettore *separatori*, e gli elementi sono separati da uno o più spazi bianchi. Inoltre, come nel caso della divisione dei record in input, se *separacampo* è la stringa nulla, ogni singolo carattere nella stringa costituisce un elemento del vettore. (e.c.)

Si noti, tuttavia, che *RS* non influisce sul comportamento di `split()`. Anche se '*RS* = ""' fa sì che il carattere di ritorno a capo sia un separatore di campo, questo non influenza il modo in cui `split()` divide le stringhe.

Recenti implementazioni di *awk*, incluso *gawk*, consentono che il terzo argomento sia una costante *regexp* (`/.../`) o anche una stringa. Anche lo standard POSIX permette questo. Si veda la [Sezione 3.6 \[Usare \*regexp\* dinamiche\], pagina 57](#), per la spiegazione della differenza tra l'uso di una costante stringa e l'uso di una costante *regexp*, sulle loro implicazioni riguardo a come scrivere correttamente un programma.

Prima di dividere la stringa, `split()` cancella ogni elemento eventualmente già presente nei vettori *vettore* e *separatori*.

Se *stringa* è la stringa nulla, il vettore non ha elementi. (Quindi, in questo modo si può cancellare un intero vettore con una sola istruzione). Si veda la [Sezione 8.4 \[L'istruzione \*delete\*\], pagina 187](#).)

Se in *stringa* non viene trovato *separacampo* (ma la stringa non è la stringa nulla), *vettore* ha solo un elemento. Il valore di quell'elemento è la *stringa* originale.

In modalità POSIX (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), il quarto argomento non è disponibile.

```
sprintf(formato, espressione1, ...)
```

Restituisce (senza stamparla) la stringa che `printf` avrebbe stampato con gli stessi argomenti (si veda la [Sezione 5.5 \[Usare l'istruzione \*printf\* per stampe sofisticate\], pagina 98](#)). Per esempio:

```
pival = sprintf("pi = %.2f (approx.)", 22/7)
```

assegna la stringa 'pi = 3.14 (approx.)' alla variabile *pival*.

```
strtonum(stringa) #
```

Esamina *stringa* e restituisce il suo valore numerico. Se *stringa* inizia con la cifra '0', `strtonum()` presuppone che *stringa* sia un numero ottale. Se *stringa* inizia



con '0x' o '0X', `strtonum()` presuppone che *stringa* sia un numero esadecimale. Per esempio:

```
$ echo 0x11 |
> gawk '{ printf "%d\n", strtonum($1) }'
+ 17
```

Usare la funzione `strtonum()` *non* è lo stesso che aggiungere zero al valore di una stringa; la conversione automatica di stringhe in numeri si applica solo a dati decimali, non a quelli ottali o esadecimali.<sup>5</sup>

Si noti anche che `strtonum()` usa il separatore decimale della localizzazione corrente per riconoscere i numeri (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), [pagina 141](#)).

`sub(regex, rimpiazzo [, obiettivo])`

Ricerca in *obiettivo*, che è visto come una stringa, la prima sottostringa più lunga possibile, a partire da sinistra, che corrisponde all'espressione regolare *regex*. Modifica l'intera stringa sostituendo il testo individuato con *rimpiazzo*. La stringa così modificata diventa il nuovo valore di *obiettivo*. Restituisce il numero di sostituzioni fatte (zero o uno).

L'argomento *regex* può essere o una costante *regex* (`/.../`) o una costante stringa (`"..."`). In quest'ultimo caso, la stringa è trattata come una *regex* da individuare. Si veda la [Sezione 3.6 \[Usare regex dinamiche\]](#), [pagina 57](#), per la spiegazione della differenza tra le due forme, delle loro implicazioni riguardo al modo di scrivere correttamente un programma.

Questa funzione è particolare perché *obiettivo* non è semplicemente usato per calcolare un valore, e non basta che sia un'espressione qualsiasi: dev'essere una variabile, un campo, o un elemento di vettore in cui `sub()` possa memorizzare un valore modificato. Se questo argomento è omissso, il comportamento di default è quello di usare e modificare `$0`.<sup>6</sup> Per esempio:

```
str = "acqua, acqua dappertutto"
sub(/cqu/, "vari", str)
```

modifica *stringa* facendola divenire 'avaria, acqua dappertutto', rimpiazzando l'occorrenza più lunga, a partire da sinistra, di 'cqu' con 'vari'.

Se il carattere speciale '&' compare in *rimpiazzo*, designa l'esatta sottostringa individuata da *regex*. (Se *regex* può individuare più di una stringa, questa sottostringa può assumere valori diversi.) Per esempio:

```
{ sub(/candidato/, "& e sua moglie"); print }
```

cambia la prima occorrenza di 'candidato' a 'candidato e sua moglie' in ogni riga in input. Ecco un altro esempio:

```
$ awk 'BEGIN {
>         str = "daabaaa"
```

<sup>5</sup> Tranne nel caso si usi l'opzione `--non-decimal-data`, il che non è consigliato. Si veda la [Sezione 12.1 \[Consentire dati di input non decimali\]](#), [pagina 331](#), per ulteriori informazioni.

<sup>6</sup> Si noti che questo significa che il record sarà dapprima ricostruito, usando il valore di `OFS` se qualche campo è stato cambiato, e che i campi saranno aggiornati dopo la sostituzione, anche se l'operazione in sé non cambia il record (è una "no-op") come `sub(/~/, "")`.

```

>      sub(/a+/, "C&C", str)
>      print str
> }'
+ dCaaCbaaa

```

questo mostra come ‘&’ possa rappresentare una stringa variabile e illustra anche la regola “a partire da sinistra, la più lunga” nell’individuazione di *regex*p (si veda la [Sezione 3.5 \[Quanto è lungo il testo individuato?\]](#), pagina 57).

L’effetto di questo carattere speciale (‘&’) può essere neutralizzato antepoendogli una barra inversa nella stringa. Come al solito, per inserire una barra inversa nella stringa, occorre scrivere due barre inverse. Quindi, occorre scrivere ‘\\&’ in una costante stringa per includere un carattere ‘&’ nel rimpiazzo. Per esempio, quanto segue mostra come rimpiazzare il primo ‘|’ su ogni riga con un ‘&’:

```
{ sub(/\\|/, "\\&"); print }
```

Come già accennato, il terzo argomento di `sub()` dev’essere una variabile, un campo, o un elemento di vettore. Alcune versioni di **awk** accettano come terzo argomento un’espressione che non è un *lvalue*. In tal caso, `sub()` cerca ugualmente l’espressione e restituisce zero o uno, ma il risultato della sostituzione (se ce n’è uno) viene scartato perché non c’è un posto dove memorizzarlo. Tali versioni di **awk** accettano espressioni come le seguenti:

```
sub(/USA/, "Stati Uniti", "gli USA e il Canada")
```

Per compatibilità storica, **gawk** accetta un tale codice erroneo. Tuttavia, l’uso di qualsiasi altra espressione non modificabile come terzo parametro causa un errore fatale, e il programma non viene portato a termine.

Infine, se la *regex*p non è una costante *regex*p, è convertita in una stringa, e quindi il valore di quella stringa è trattato come la *regex*p da individuare.

**substr(stringa, inizio [, lunghezza])**

Restituisce una sottostringa di *stringa* lunga *lunghezza* caratteri, iniziando dal carattere numero *inizio*. Il primo carattere di una stringa è il carattere numero uno.<sup>7</sup> Per esempio, `substr("Washington", 5, 3)` restituisce "ing".

Se *lunghezza* non è presente, `substr()` restituisce l’intero suffisso di *stringa* a partire dal carattere numero *inizio*. Per esempio, `substr("Washington", 5)` restituisce "ington". L’intero suffisso è restituito anche se *lunghezza* è maggiore del numero di caratteri disponibili nella stringa, a partire dal carattere *inizio*.

Se *inizio* è minore di uno, `substr()` lo tratta come se fosse uno. (POSIX non specifica cosa fare in questo caso: BWK **awk** si comporta così, e quindi **gawk** fa lo stesso.) Se *inizio* è maggiore del numero di caratteri nella stringa, `substr()` restituisce la stringa nulla. Analogamente, se *lunghezza* è presente ma minore o uguale a zero, viene restituita la stringa nulla.

La stringa restituita da `substr()` *non può* essere assegnata. Quindi, è un errore tentare di modificare una porzione di una stringa, come si vede nel seguente esempio:

```
stringa = "abcdef"
```

<sup>7</sup> Questo è differente da C e C++, in cui il primo carattere ha il numero zero.

```
# tentare di ottenere "abCDEf", non è possibile
substr(stringa, 3, 3) = "CDE"
```

È anche un errore usare `substr()` come terzo argomento di `sub()` o `gsub()`:

```
gsub(/xyz/, "pdq", substr($0, 5, 20)) # SBAGLIATO
```

(Alcune versioni commerciali di `awk` consentono un tale uso di `substr()`, ma un tale codice non è portabile.)

Se si devono sostituire pezzi di una stringa, si combini `substr()` con una concatenazione di stringa, nel modo seguente:

```
stringa = "abcdef"
...
stringa = substr(stringa, 1, 2) "CDE" substr(stringa, 6)
```

`tolower(stringa)`

Restituisce una copia di *stringa*, con ogni carattere maiuscolo nella stringa rimpiazzato dal suo corrispondente carattere minuscolo. I caratteri non alfabetici non vengono modificati. Per esempio, `tolower("MaIuSc010 MiNuSc010 123")` restituisce "maiuscolo minuscolo 123".

`toupper(stringa)`

Restituisce una copia di *stringa*, con ogni carattere minuscolo nella stringa rimpiazzato dal suo corrispondente carattere maiuscolo. I caratteri non alfabetici non vengono modificati. Per esempio, `tolower("MaIuSc010 MiNuSc010 123")` restituisce "MAIUSCOLO MINUSCOLO 123".

#### Individuare la stringa nulla

In `awk`, l'operatore `'*'` può individuare la stringa nulla. Questo è particolarmente importante per le funzioni `sub()`, `gsub()` e `gensub()`. Per esempio:

```
$ echo abc | awk '{ gsub(/m*/, "X"); print }'
- XaXbXcX
```

Sebbene questo sia abbastanza sensato, può suscitare una certa sorpresa.

### 9.1.3.1 Ulteriori dettagli su `'\'` e `'&'` con `sub()`, `gsub()` e `gensub()`

**ATTENZIONE:** Si dice che questa sottosezione possa causare dei mal di testa.

In prima lettura può essere benissimo saltata.

Quando si usa `sub()`, `gsub()` o `gensub()`, e si desidera includere delle barre inverse e delle "e commerciali" (`&`) nel testo da sostituire è necessario ricordare che ci sono parecchi livelli di *protezione caratteri* in gioco.

Anzitutto, vi è il livello *lessicale*, quello in cui `awk` legge un programma e ne costruisce una copia interna da eseguire. Poi c'è il momento dell'esecuzione, quello in cui `awk` esamina effettivamente la stringa da sostituire, per determinare cosa fare.

In entrambi i livelli, `awk` ricerca un dato insieme di caratteri che possono venire dopo una barra inversa. A livello lessicale, cerca le sequenze di protezione elencate in [Sezione 3.2 \[Sequenze di protezione\]](#), [pagina 50](#). Quindi, per ogni `'\'` che `awk` elabora al momento dell'esecuzione, occorre immetterne due a livello lessicale. Quando un carattere che non

ha necessità di una sequenza di protezione segue una ‘\’, sia BWK `awk` che `gawk` semplicemente rimuovono la ‘\’ stessa e mettono il carattere seguente nella stringa. Quindi, per esempio, `"a\qb"` è trattato come se si fosse scritto `"aqb"`.

Al momento dell’esecuzione, le varie funzioni gestiscono sequenze di ‘\’ e ‘&’ in maniera differente. La situazione è (purtroppo) piuttosto complessa. Storicamente, le funzioni `sub()` e `gsub()` trattavano la sequenza di due caratteri ‘\&’ in maniera speciale; questa sequenza era rimpiazzata nel testo generato da un singolo carattere ‘&’. Ogni altra ‘\’ contenuta nella stringa *rimpiazzo* che non era posta prima di una ‘&’ era lasciata passare senza modifiche. Questo è illustrato nella [Tabella 9.1](#).

Immissione	<code>sub()</code> vede	<code>sub()</code> genera
<code>\&amp;</code>	<code>&amp;</code>	Il testo individuato
<code>\\&amp;</code>	<code>\&amp;</code>	Il carattere ‘&’
<code>\\\&amp;</code>	<code>\&amp;</code>	Il carattere ‘&’
<code>\\\\&amp;</code>	<code>\\&amp;</code>	I caratteri ‘\&’
<code>\\\&amp;</code>	<code>\\&amp;</code>	I caratteri ‘\&’
<code>\\\\\&amp;</code>	<code>\\\&amp;</code>	I caratteri ‘\\&’
<code>\\q</code>	<code>\q</code>	I caratteri ‘\q’

Tabella 9.1: Elaborazione storica delle sequenze di protezione per `sub()` e `gsub()`

Questa tabella mostra l’elaborazione a livello lessicale, in cui un numero dispari di barre inverse diventa un numero pari al momento dell’esecuzione, e mostra anche l’elaborazione in fase di esecuzione fatta da `sub()`. (Per amor di semplicità le tavole che ancora seguono mostrano solo il caso di un numero pari di barre inverse immesso a livello lessicale.)

Il problema con l’approccio storico è che non c’è modo di ottenere un carattere ‘\’ seguito dal testo individuato.

Parecchie edizioni dello standard POSIX hanno provato a risolvere questo problema, senza riuscirci. I dettagli sono irrilevanti in questo contesto.

A un certo punto, il manutentore di `gawk` ha presentato una proposta per una revisione dello standard per tornare a regole che corrispondano più da vicino alla prassi originalmente seguita. Le regole proposte hanno dei casi speciali che rendono possibile produrre una ‘\’ prima del testo individuato. Questo si può vedere nella [Tabella 9.2](#).

Immissione	<code>sub()</code> vede	<code>sub()</code> genera
<code>\\\&amp;</code>	<code>\\&amp;</code>	I caratteri ‘\&’
<code>\\&amp;</code>	<code>\&amp;</code>	Il carattere ‘\’, seguito dal testo individuato
<code>\&amp;</code>	<code>\&amp;</code>	Il carattere ‘&’
<code>\q</code>	<code>\q</code>	I caratteri ‘\q’
<code>\\</code>	<code>\\</code>	<code>\\</code>

Tabella 9.2: Regole `gawk` per `sub()` e barra inversa

In breve, al momento dell'esecuzione, ci sono ora tre sequenze speciali di caratteri ('\\&', '\\&', e '&') mentre tradizionalmente ce n'era una sola. Tuttavia, come nel caso storico, ogni '\\' che non fa parte di una di queste tre sequenze non è speciale e appare nell'output così come è scritto.

**gawk** 3.0 e 3.1 seguono queste regole per `sub()` e `gsub()`. La revisione dello standard POSIX ha richiesto molto più tempo di quel che ci si attendeva. Inoltre, la proposta del manutentore di **gawk** è andata persa durante il processo di standardizzazione. Le regole finali risultanti sono un po' più semplici. I risultati sono simili, tranne che in un caso.

Le regole POSIX stabiliscono che '\\&' nella stringa di rimpiazzo produca il carattere '&', '\\ produce il carattere '\\', e che '\\' seguito da qualsiasi carattere non è speciale; la '\\' è messa direttamente nell'output. Queste regole sono presentate nella [Tabella 9.3](#).

Immissione	<code>sub()</code> vede	<code>sub()</code> genera
\\\\\\\\&	\\\\&	I caratteri '\\&'
\\\\\\\\	\\\\	Il carattere '\\', seguito dal testo individuato
\\\\&	\\\\&	Il carattere '&'
\\\\q	\\\\q	I caratteri '\\q'
\\\\	\\\\	\\

Tabella 9.3: Regole POSIX per `sub()` e `gsub()`

Il solo caso in cui la differenza è rilevante è l'ultimo: '\\\\\\' è visto come '\\\\' e produce '\\' invece che '\\\\'.

A partire dalla versione 3.1.4, **gawk** ha seguito le regole POSIX quando si specifica `--posix` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)). Altrimenti, ha continuato a seguire le regole proposte [a POSIX], poiché questa è stato il comportamento seguito per parecchi anni.

Quando la versione 4.0.0 è stata rilasciata, il manutentore di **gawk** ha stabilito come default le regole POSIX, interrompendo così oltre un decennio di compatibilità all'indietro.<sup>8</sup> Inutile dire che questa non è stata una buona idea, e quindi dalla versione 4.0.1, **gawk** ha ripreso il suo comportamento tradizionale, seguendo le regole POSIX solo quando si specifica l'opzione `--posix`.

Le regole per `gensub()` sono molto più semplici. Al momento dell'esecuzione, quando **gawk** vede una '\\', se il carattere seguente è una cifra, il testo individuato dalla corrispondente sottoespressione tra parentesi è inserito nell'output generato. Altrimenti, qualsiasi carattere segua la '\\' viene inserito nel testo generato, mentre la '\\' va persa, come si vede nella [Tabella 9.4](#).

<sup>8</sup> Questa decisione si è dimostrata piuttosto avventata, anche se una nota in questa sezione avvertiva che la successiva versione principale di **gawk** avrebbe adottato le regole POSIX.

Immissione	<code>gensub()</code> vede	<code>gensub()</code> genera
<code>&amp;</code>	<code>&amp;</code>	Il testo individuato
<code>\\&amp;</code>	<code>\\&amp;</code>	Il carattere <code>'&amp;'</code>
<code>\\\\</code>	<code>\\\\</code>	Il carattere <code>'\\'</code>
<code>\\\\&amp;</code>	<code>\\\\&amp;</code>	Il carattere <code>'\\'</code> , seguito dal testo individuato
<code>\\\\\\\\&amp;</code>	<code>\\\\\\\\&amp;</code>	I caratteri <code>'\\&amp;'</code>
<code>\\\\q</code>	<code>\\\\q</code>	Il carattere <code>'q'</code>

Tabella 9.4: Elaborazione sequenze di protezione in `gensub()`

A causa della complessità dell'elaborazione a livello lessicale e in fase di esecuzione, e dei casi speciali di `sub()` e `gsub()`, si raccomanda l'uso di `gawk` e di `gensub()` quando ci siano da fare delle sostituzioni.

### 9.1.4 Funzioni di Input/Output

Le seguenti funzioni riguardano l'input/output (I/O). I parametri facoltativi sono racchiusi tra parentesi quadre ([ ]):

`close(nome_file [, come])`

Chiude il file *nome\_file* in input o in output. Alternativamente, l'argomento può essere un comando della shell usato per creare un coprocesso, o per ridirigere verso o da una *pipe*; questo coprocesso o *pipe* viene chiuso. Si veda la [Sezione 5.9 \[Chiudere ridirezioni in input e in output\]](#), pagina 109, per ulteriori informazioni.

Quando si chiude un coprocesso, può talora essere utile chiudere dapprima un lato della *pipe* bidirezionale e quindi chiudere l'altro. Questo si può fare fornendo un secondo argomento a `close()`. Questo secondo argomento (*come*) dovrebbe essere una delle due stringhe `"to"` o `"from"`, che indicano quale lato della *pipe* chiudere. La stringa può essere scritta indifferentemente in maiuscolo o in minuscolo. Si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339, che tratta questa funzionalità con maggior dettaglio e mostra un esempio.

Si noti che il secondo argomento di `close()` è un'estensione `gawk`; non è disponibile in modalità compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

`fflush([nome_file])`

Scriva su disco ogni output contenuto in memoria, associato con *nome\_file*, che è o un file aperto in scrittura o un comando della shell che ridirige output a una *pipe* o a un coprocesso.

Molti programmi di utilità *bufferizzano* il loro output (cioè, accumulano in memoria record da scrivere in un file su disco o sullo schermo, fin quando non arriva il momento giusto per inviare i dati al dispositivo di output). Questo è spesso più efficiente che scrivere ogni particella di informazione non appena diventa disponibile. Tuttavia, qualche volta è necessario forzare un programma a *svuotare* i suoi buffer (cioè, inviare l'informazione alla sua destinazione, anche

se un buffer non è pieno). Questo è lo scopo della funzione `fflush()`; anche `gawk` scrive il suo output in un buffer, e la funzione `fflush()` forza `gawk` a svuotare i suoi buffer.

Brian Kernighan ha aggiunto `fflush()` al suo `awk` nell'aprile 1992. Per due decenni è rimasta un'estensione comune. A Dicembre 2012 è stata accettata e inclusa nello standard POSIX. Si veda [il sito Web dell'Austin Group](#).

POSIX standardizza `fflush()` come segue: se non c'è alcun argomento, o se l'argomento è la stringa nulla (`""`), `awk` svuota i buffer di *tutti* i file in output e di *tutte* le *pipe*.

**NOTA:** Prima della versione 4.0.2, `gawk` avrebbe svuotato solo i buffer dello standard output se non era specificato alcun argomento, e svuotato tutti i buffer dei file in output e delle *pipe* se l'argomento era la stringa nulla. Questo è stato modificato per essere compatibile con l'`awk` di Kernighan, nella speranza che standardizzare questa funzionalità in POSIX sarebbe stato più agevole (come poi è effettivamente successo).

Con `gawk`, si può usare `'fflush("/dev/stdout")'` se si desidera solo svuotare i buffer dello standard output.

`fflush()` restituisce zero se il buffer è svuotato con successo; altrimenti, restituisce un valore diverso da zero. (`gawk` restituisce `-1`.) Nel caso in cui tutti i buffer vadano svuotati, il valore restituito è zero solo se tutti i buffer sono stati svuotati con successo. Altrimenti, è `-1`, e `gawk` avvisa riguardo al *nome\_file* che ha problemi.

`gawk` invia anche un messaggio di avvertimento se si tenta di svuotare i buffer di un file o *pipe* che era stato aperto in lettura (p.es. con `getline`), o se *nome\_file* non è un file, una *pipe*, o un coprocesso aperto. in tal caso, `fflush()` restituisce ancora `-1`.

**Bufferizzazione interattiva e non interattiva**

A complicare ulteriormente le cose, i problemi di bufferizzazione possono peggiorare se il programma eseguito è *interattivo* (cioè, se comunica con un utente seduto davanti a una tastiera).<sup>9</sup>

I programmi interattivi normalmente *bufferizzano per riga* il loro output (cioè, scrivono in output una riga alla volta). I programmi non-interattivi attendono di aver riempito un buffer, il che può voler dire anche parecchie righe di output. Ecco un esempio della differenza:

```
$ awk '{ print $1 + $2 }'
1 1
+ 2
2 3
+ 5
Ctrl-d
```

Ogni riga di output è stampata immediatamente. Si confronti questo comportamento con quello di questo esempio:

```
$ awk '{ print $1 + $2 }' | cat
1 1
2 3
Ctrl-d
+ 2
+ 5
```

In questo caso, nessun output viene stampato finché non è stato battuto il *Ctrl-d*, perché l'output è bufferizzato e inviato tramite *pipe* al comando *cat* in un colpo solo.

`system(comando)`

Esegue il comando del sistema operativo *comando* e quindi ritorna al programma *awk*. Restituisce il codice ritorno di *comando*.

Per esempio, inserendo il seguente frammento di codice in un programma *awk*:

```
END {
    system("date | mail -s 'awk completato' root")
}
```

all'amministratore di sistema viene inviato un messaggio di posta quando il programma *awk* termina di elaborare l'input e inizia l'elaborazione da eseguire alla fine dell'input.

Si noti che la ridirezione di `print` o `printf` in una *pipe* è spesso sufficiente per ottenere lo stesso risultato. Se è necessario eseguire parecchi comandi, è più efficiente stamparli verso una *pipe* diretta alla shell:

```
while (ancora lavoro da fare)
    print comando | "/bin/sh"
close("/bin/sh")
```

<sup>9</sup> Un programma è interattivo se il suo standard output è connesso a un dispositivo terminale. Ai giorni nostri, questo vuol dire davanti a uno schermo e a una tastiera.

Tuttavia, nel caso che il programma `awk` sia interattivo, `system()` è utile per eseguire grossi programmi autonomi, come ad esempio la shell o un programma di modifica testi. Alcuni sistemi operativi non consentono di implementare la funzione `system()`. Richiamare `system()` in sistemi in cui non è disponibile provoca un errore fatale.

**NOTA:** Quando si specifica l'opzione `--sandbox`, la funzione `system()` è disabilitata (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

Nei sistemi aderenti allo standard POSIX, il codice di ritorno di un comando è un numero contenuto in 16 bit. Il valore del codice di ritorno passato alla funzione C `exit()` alla fine del programma è contenuto negli 8 bit di valore più alto dei 16 bit (la metà sinistra) che compongono il numero. I bit di valore più basso (la metà destra) indicano se il processo è stato terminato da un segnale (bit 7), e, se questo è il caso, il numero del segnale che ha provocato la terminazione (bit 0–6).

Tradizionalmente, la funzione `system()` di `awk` si è semplicemente limitata a restituire il valore del codice di ritorno diviso per 256 (ossia la metà sinistra del numero di 16 bit, spostata a destra). In una situazione normale questo equivale a utilizzare il codice di ritorno di `system()`, ma nel caso in cui il programma sia stato terminato da un segnale, il valore diventa un numero frazionario a virgola mobile.<sup>10</sup> POSIX stabilisce che la chiamata a `system()` dall'interno di `awk` dovrebbe restituire l'intero valore a 16 bit.

`gawk` si trova in qualche modo a metà strada. I valori del codice di ritorno sono descritti nella [Tabella 9.5](#).

Situazione	Valore codice di ritorno da <code>system()</code>
<code>--traditional</code>	Valore dalla funzione C <code>system()</code> /256
<code>--posix</code>	Valore dalla funzione C <code>system()</code>
Uscita normale dal comando	Codice di ritorno del comando
Terminazione da un segnale	256 + numero segnale "assassino"
Terminazione da un segnale con dump memoria	512 + numero segnale "assassino"
Qualsiasi tipo di errore	-1

Tabella 9.5: Valori codici di ritorno da chiamata a `system()`

<sup>10</sup> In uno scambio di messaggi privato il Dr. Kernighan mi ha comunicato che questo modo di procedere è probabilmente errato.

**Controllare la bufferizzazione dell'output con system()**

La funzione `fflush()` consente un controllo esplicito sulla bufferizzazione dell'output per singoli file e *pipe*. Tuttavia, il suo utilizzo non è portabile su molte delle meno recenti implementazioni di `awk`. Un metodo alternativo per forzare la scrittura dell'output è una chiamata a `system()` che abbia come argomento la stringa nulla:

```
system("")    # scrive l'output su disco
```

`gawk` tratta questo uso della funzione `system()` come un caso speciale, e si guarda bene dall'invocare la shell (o un altro interprete di comandi) con un comando nullo. Quindi, con `gawk`, questa maniera di procedere non è solo utile, ma è anche efficiente. Questo metodo dovrebbe funzionare anche con altre implementazioni di `awk`, ma non è detto che eviti una invocazione non necessaria della shell. (Altre implementazioni potrebbero limitarsi a forzare la scrittura del buffer associato con lo standard output, e non necessariamente di tutto l'output bufferizzato.)

Avendo in mente le attese di un programmatore, sarebbe sensato che `system()` forzi la scrittura su disco di tutto l'output disponibile. Il programma seguente:

```
BEGIN {
    print "prima riga stampata"
    system("echo system echo")
    print "seconda riga stampata"
}
```

deve stampare:

```
prima riga stampata
system echo
seconda riga stampata
```

e non:

```
system echo
prima riga stampata
seconda riga stampata
```

Se `awk` non forzasse la scrittura dei suoi buffer prima di invocare `system()`, l'output sarebbe quest'ultimo (quello non voluto).

**9.1.5 Funzioni per gestire marcature temporali**

I programmi `awk` sono frequentemente usati per elaborare file di registro [file con estensione `.log`], che contengono l'informazione sulla data e l'ora (marcatura temporale) in cui un particolare record è stato registrato sul log. Molti programmi registrano questa informazione nel formato restituito dalla chiamata di sistema `time()`, la quale misura il numero di secondi trascorsi a partire da una certa data iniziale (Epoca). Nei sistemi aderenti allo standard POSIX, questo è il numero di secondi a partire dal primo gennaio 1970, ora di Greenwich (1970-01-01 00:00:00 UTC), senza includere i secondi intercalari.<sup>11</sup> Tutti i sistemi noti aderenti allo standard POSIX gestiscono le marcature temporali da 0 fino a  $2^{31} - 1$ , il che è sufficiente per rappresentare date e ore fino a inizio 2038 (2038-01-19 03:14:07 UTC). Molti sistemi supportano una maggiore estensione di date, compresi dei valori negativi per rappresentare delle date anteriori all'Epoca.

<sup>11</sup> Si veda il [Glossario], pagina 515, in particolare le voci "Epoca" e "UTC."

Per facilitare l'elaborazione di tali file di registro, e per produrre dei rapporti utili, **gawk** prevede le seguenti funzioni per lavorare con le marcature temporali. Si tratta di estensioni **gawk**; non sono previste nello standard POSIX.<sup>12</sup> Tuttavia, anche versioni recenti di **mawk** (si veda la [Sezione B.5 \[Altre implementazioni di \*\*awk\*\* liberamente disponibili\]](#), pagina 494) prevedono queste funzioni. I parametri facoltativi sono racchiusi tra parentesi quadre ([ ]):

**mktime**(*specifiche\_data* [, *utc-flag* ])

Trasforma *specifiche\_data* in una marcatura temporale nello stesso formato restituito da **systemtime**(). È simile alla funzione omonima in ISO C. L'argomento, *specifiche\_data*, è una stringa della forma "**AAAA MM GG HH MM SS [DST]**". La stringa consiste di sei o sette numeri che rappresentano, rispettivamente, l'anno in quattro cifre, il mese da 1 a 12, il giorno del mese da 1 a 31, l'ora del giorno da 0 a 23, il minuto da 0 a 59, il secondo da 0 a 60,<sup>13</sup> e un'indicazione opzionale relativa all'ora legale.

I valori di questi numeri possono non essere negli intervalli specificati; per esempio, un'ora di  $-1$  sta a indicare 1 ora prima di mezzanotte. Viene adottato il calendario gregoriano con l'origine posta all'anno zero, con l'anno 0 che viene prima dell'anno 1 e l'anno  $-1$  che viene prima dell'anno 0. Se il flag *utc-flag* è specificato ed è diverso da zero e dalla stringa nulla, si suppone che l'ora sia quella del fuso orario UTC; altrimenti l'ora è considerata essere quella del fuso orario locale. Se l'indicatore dell'ora legale è positivo, si presuppone che l'ora sia quella legale; se è 0, l'ora considerata è quella di Greenwich (standard time); se invece è negativo (questo è il default), **mktime**() tenta di determinare se è in vigore l'ora legale o no, nel momento specificato.

Se *specifiche\_data* non contiene elementi in numero sufficiente, o se la data e ora risultante sono fuori dall'intervallo previsto, **mktime**() restituisce  $-1$ .

**strftime**([*formato* [, *data\_e\_ora* [, *utc*] ] ])

Formatta la data e ora specificata da *data\_e\_ora* in base alle indicazioni contenute nella stringa *formato* e restituisce il risultato. È simile alla funzione omonima in ISO C. Se *utc* è presente ed è diverso da zero o dalla stringa nulla, il valore è formattato come UTC (Tempo Coordinato Universale, già noto come GMT o Tempo Medio di Greenwich). Altrimenti, il valore è formattato per il fuso orario locale. La stringa *data\_e\_ora* è nello stesso formato del valore restituito dalla funzione **systemtime**(). Se non si specifica l'argomento *data\_e\_ora*, **gawk** usa l'ora del giorno corrente per la formattazione. Omettendo l'argomento *formato*, **strftime**() usa il valore di **PROCINFO["strftime"]** come stringa di formattazione (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162). Il valore di default della stringa è "**%a %b %e %H:%M:%S %Z %Y**". Questa stringa di formattazione produce lo stesso output del programma di utilità equivalente **date**. Si può assegnare un nuovo valore a **PROCINFO["strftime"]** per modificare la formattazione di default; si veda la lista che segue per le varie direttive di formattazione.

<sup>12</sup> Il comando di utilità GNU **date** può fare anche molte delle cose qui descritte. Può essere preferibile usarlo per semplici operazioni relative a data e ora in semplici script della shell.

<sup>13</sup> Occasionalmente ci sono dei minuti in un anno con un secondo intercalare, il che spiega perché i secondi possono arrivare fino a 60.

**systeme()**

Restituisce l'ora corrente come numero di secondi a partire dall'Epoca del sistema. Sui sistemi aderenti allo standard POSIX, questo è il numero di secondi trascorsi a partire dal primo gennaio 1970, ora di Greenwich (1970-01-01 00:00:00 UTC), senza includere i secondi intercalari.

La funzione **systeme()** consente di confrontare una marcatura temporale in un file di registro con la data e ora correnti. In particolare, è facile determinare quanto tempo prima un particolare record è stato registrato. È anche possibile produrre record di registro usando il formato “secondi a partire dall'Epoca”.

La funzione **mktime()** consente di convertire una rappresentazione in forma testuale di una data e ora in una marcatura temporale. Questo semplifica i confronti prima/dopo tra differenti date e ore, in particolare quando si abbia a che fare con date e ore provenienti da una fonte esterna, come un file di registro.

La funzione **strftime()** permette di trasformare facilmente una marcatura temporale in un'informazione intelligibile. È analoga come tipo alla funzione **sprintf()** (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)), nel senso che copia letteralmente ciò che non è una specifica di formato nella stringa che viene restituita, mentre sostituisce i valori di data e ora a seconda delle specifiche di formato contenute nella stringa *formato*.

Per **strftime()** lo standard 1999 ISO C<sup>14</sup> consente le seguenti specifiche di formattazione delle date:

%a	Il nome abbreviato del giorno della settimana nella lingua locale.
%A	Il nome completo del giorno della settimana nella lingua locale.
%b	Il nome abbreviato del mese dell'anno nella lingua locale.
%B	Il nome completo del mese dell'anno nella lingua locale.
%c	Il formato “appropriato” della rappresentazione della data e ora nella lingua locale. (Questo è ‘%A %B %d %T %Y’ per la localizzazione “C”.)
%C	La parte che designa il secolo nell'anno corrente. Si ottiene dividendo per 100 l'anno, e troncando verso il basso all'intero più vicino.
%d	Il giorno del mese come numero decimale (01–31).
%D	Equivale a specificare ‘%m/%d/%y’.
%e	Il giorno del mese, preceduto da uno spazio se di tratta di una cifra sola.
%F	Equivale a specificare ‘%Y-%m-%d’. Questo è il formato ISO 8601 della data.
%g	L'anno (ultime due cifre) ricavato prendendo il resto della divisione per 100 dell'anno a cui appartiene la settimana, secondo ISO 8601, come numero decimale (00–99). Per esempio, il primo gennaio 2012, fa parte della settimana 53 del 2011. Quindi, l'anno relativo al numero di settimana ISO di quella data è 2011 (ossia 11), anche se la data in sé è nel 2012. Analogamente, il 31 dicembre

<sup>14</sup> Sfortunatamente, non tutte le funzioni **strftime()** dei vari sistemi operativi ammettono tutte le conversioni qui elencate.

2012, è nella prima settimana del 2013. Quindi, l'anno relativo al numero di settimana ISO di quella data è 2013 (ossia 13), anche se la data in sé è nel 2012.

%G	L'anno intero relativo al numero di settimana ISO, come numero decimale.
%h	Equivalente a '%b'.
%H	L'ora (in un orologio a 24 ore) come numero decimale (00–23).
%I	L'ora (in un orologio a 12 ore) come numero decimale (01–12).
%j	Il giorno dell'anno come numero decimale (001–366).
%m	Il mese come numero decimale (01–12).
%M	Il minuto come numero decimale (00–59).
%n	Un carattere di ritorno a capo (ASCII LF).
%p	L'equivalente nella lingua locale delle designazioni AM/PM (mattino/pomeriggio) associate a un orologio a 12 ore.
%r	L'ora locale nel formato a 12 ore. (Questo è '%I:%M:%S %p' nella localizzazione "C".)
%R	Equivalente a specificare '%H:%M'.
%S	Il secondo come numero decimale (00–60).
%t	Un carattere di tabulazione [TAB].
%T	Equivalente a specificare '%H:%M:%S'.
%u	Il numero del giorno della settimana come numero decimale (1–7). Lunedì è il giorno numero 1.
%U	Il numero di settimana dell'anno (con la prima domenica dell'anno presa come primo giorno della prima settimana) come numero decimale (00–53).
%V	Il numero di settimana dell'anno (con il primo lunedì dell'anno preso come primo giorno della prima settimana) come numero decimale (01–53). Il metodo per determinare il numero di settimana è quello specificato dallo standard ISO 8601. (In pratica: se la settimana che contiene il primo gennaio ha quattro o più giorni nel nuovo anno, allora è la settimana numero uno; altrimenti è l'ultima settimana [52 o 53] dell'anno precedente, e la settimana successiva è la settimana numero uno.)
%w	Il giorno della settimana come numero decimale (0–6). Domenica è il giorno zero.
%W	Il numero di settimana dell'anno (con il primo lunedì come primo giorno della settimana numero uno) come numero decimale (00–53).
%x	Il formato “appropriato” della rappresentazione della data nella lingua locale. (Questo è '%A %B %d %Y' nella localizzazione "C".)
%X	Il formato “appropriato” della rappresentazione della data. (Questo è '%T' nella localizzazione "C".)

%y	L'anno modulo 100 (le ultime due cifre) come numero decimale (00–99).
%Y	L'anno come numero decimale (p.es., 2015).
%z	La differenza di fuso orario [rispetto all'ora di Greenwich] in formato '+00MM' (p.es., il formato necessario per produrre intestazioni di data conformi agli standard RFC 822/RFC 1036).
%Z	Il nome o l'abbreviazione della zona di fuso orario ( <i>time zone</i> ); se il fuso orario non è determinabile, è impostata alla stringa nulla.
%Ec %EC %Ex %EX %Ey %EY %Od %Oe %OH	
%OI %Om %OM %OS %Ou %OU %OV %Ow %OW %Oy	
	“notazioni alternative” di specifica in cui solo la seconda lettera ('%c', '%C' e così via) è significativa. <sup>15</sup> (Queste facilitano la compatibilità con il programma di utilità POSIX <code>date</code> .)
%%	Un singolo carattere '%’.

Se uno specificatore di conversione non è tra quelli elencati sopra, il comportamento è indefinito.<sup>16</sup>

Per sistemi che non aderiscono completamente agli standard `gawk` utilizza una copia di `strftime()` dalla libreria C di GNU. Sono disponibili tutte le specifiche di formato sopra elencate. Se la detta versione è usata per compilare `gawk` (si veda la [Appendice B \[Installare gawk\]](#), [pagina 479](#)), sono disponibili anche le seguenti ulteriori specifiche di formato:

%k	L'ora (in un orologio a 24 ore) come numero decimale (0–23). I numeri di una sola cifra sono preceduti da uno spazio bianco.
%l	L'ora (in un orologio a 12 ore) come numero decimale (1–12). I numeri di una sola cifra sono preceduti da uno spazio bianco.
%s	L'ora espressa in numero di secondi a partire dall'Epoca.

In aggiunta a ciò, le notazioni alternative sono riconosciute, ma al loro posto sono usate quelle normali.

Il seguente esempio è un'implementazione `awk` del programma di utilità POSIX `date`. Normalmente, il programma di utilità `date` stampa la data e l'ora corrente nel formato ben noto. Tuttavia, se si specifica al comando un argomento che inizia con un '+', `date` copia i caratteri che non sono specifiche di formato nello standard output e interpreta l'ora corrente secondo gli specificatori di formato contenuti nella stringa. Per esempio:

```
$ date '+Oggi è %A, %d %B %Y.'
- Oggi è lunedì, 22 settembre 2014.
```

Ecco la versione `gawk` del programma di utilità `date`. È all'interno di uno script di shell per gestire l'opzione `-u`, che richiede che `date` sia eseguito come se il fuso orario fosse impostato a UTC:

```
#!/bin/sh
```

<sup>15</sup> Se questo risulta incomprensibile, non è il caso di preoccuparsi; queste notazioni hanno lo scopo di facilitare la “internazionalizzazione” dei programmi. Altre funzionalità di internazionalizzazione sono descritte in [Capitolo 13 \[Internazionalizzazione con gawk\]](#), [pagina 349](#).

<sup>16</sup> Questo è perché ISO C lascia indefinito il comportamento della versione C di `strftime()` e `gawk` usa la versione di sistema di `strftime()`, se disponibile. Tipicamente, lo specificatore di conversione “non previsto” non appare nella stringa risultante, o appare così come è scritto.

```

#
# date --- simula il comando POSIX 'date'

case $1 in
-u) TZ=UTC0      # usare UTC
    export TZ
    shift ;;
esac

gawk 'BEGIN {
    formato = PROCINFO["strftime"]
    codice_di_ritorno = 0

    if (ARGC > 2)
        codice_di_ritorno = 1
    else if (ARGC == 2) {
        formato = ARGV[1]
        if (formato ~ /\^\/)
            formato = substr(formato, 2)  # toglie il + iniziale
    }
    print strftime(formato)
    exit codice_di_ritorno
}' "$@"

```

### 9.1.6 Funzioni per operazioni di manipolazione bit

*Io posso spiegarlo per te, ma non posso capirlo per te.*

—Anonimo

Molti linguaggi consentono di eseguire operazioni *bit a bit* su due numeri interi. In altre parole, l'operazione è eseguita su ogni successiva coppia di bit presi da ognuno dei due operandi. Tre operazioni comuni sono AND, OR e XOR bit a bit. Queste operazioni sono descritte nella [Tabella 9.6](#).

Operandi	Operatore booleano					
	AND		OR		XOR	
	0	1	0	1	0	1
0	0	0	0	1	0	1
1	0	1	1	1	1	0

Tabella 9.6: Operazioni a livello di bit

Come si vede, il risultato di un'operazione di AND è 1 solo quando *entrambi* i bit sono 1. Il risultato di un'operazione di OR è 1 se *almeno un* bit è 1. Il risultato di un'operazione di XOR è 1 se l'uno o l'altro bit è 1, ma non tutti e due. La successiva operazione è il *complemento*; il complemento di 1 è 0 e il complemento di 0 è 1. Quindi, quest'operazione “inverte” tutti i bit di un dato valore.

Infine, due altre operazioni comuni consistono nello spostare i bit a sinistra o a destra. Per esempio, se si ha una stringa di bit ‘10111001’ e la si sposta a destra di tre bit, si ottiene ‘00010111’.<sup>17</sup> Partendo nuovamente da ‘10111001’ e spostandolo a sinistra di tre bit, si ottiene ‘11001000’. La lista seguente descrive le funzioni predefinite di **gawk** che rendono disponibili le operazioni a livello di bit. I parametri facoltativi sono racchiusi tra parentesi quadre ([ ]):

`and(v1, v2 [, ...])`

Restituisce l’AND bit a bit degli argomenti. Gli argomenti devono essere almeno due.

`compl(val)`

Restituisce il complemento bit a bit di *val*.

`lshift(val, contatore)`

Restituisce il valore di *val*, spostato a sinistra di *contatore* bit.

`or(v1, v2 [, ...])`

Restituisce l’OR bit a bit degli argomenti. Gli argomenti devono essere almeno due.

`rshift(val, contatore)`

Restituisce il valore di *val*, spostato a destra di *contatore* bit.

`xor(v1, v2 [, ...])`

Restituisce il XOR bit a bit degli argomenti. Gli argomenti devono essere almeno due.

**ATTENZIONE:** A partire dalla versione di **gawk** versione 4.2, gli operandi negativi non sono consentiti per nessuna di queste funzioni. Un operando negativo produce un errore fatale. Si veda la nota a lato “Attenzione. Non è tutto oro quel che luccica!” per maggiori informazioni sul perché.

Ecco una funzione definita dall’utente (si veda la [Sezione 9.2 \[Funzioni definite dall’utente\]](#), [pagina 224](#)) che illustra l’uso di queste funzioni:

---

<sup>17</sup> Questo esempio presuppone che degli zeri riempiano le posizioni a sinistra. Per **gawk**, è sempre così, ma in alcuni linguaggi è possibile che le posizioni a sinistra siano riempite con degli uno.

```
# bits2str --- decodifica un byte in una serie di 0/1 leggibili

function bits2str(byte,      dati, maschera)
{
    if (byte == 0)
        return "0"

    maschera = 1
    for (; byte != 0; stringa = rshift(stringa, 1))
        dati = (and(byte, maschera) ? "1" : "0") dati

    while ((length(dati) % 8) != 0)
        dati = "0" dati

    return dati
}

BEGIN {
    printf "123 = %s\n", bits2str(123)
    printf "0123 = %s\n", bits2str(0123)
    printf "0x99 = %s\n", bits2str(0x99)
    comp = compl(0x99)
    printf "compl(0x99) = %#x = %s\n", comp, bits2str(comp)
    shift = lshift(0x99, 2)
    printf "lshift(0x99, 2) = %#x = %s\n", shift, bits2str(shift)
    shift = rshift(0x99, 2)
    printf "rshift(0x99, 2) = %#x = %s\n", shift, bits2str(shift)
}
```

Questo programma produce il seguente output quando viene eseguito:

```
$ gawk -f testbits.awk
+ 123 = 01111011
+ 0123 = 01010011
+ 0x99 = 10011001
+ compl(0x99) = 0x3fffffff66 = 00111111111111111111111111111111
+ 111111111111111101100110
+ lshift(0x99, 2) = 0x264 = 0000001001100100
+ rshift(0x99, 2) = 0x26 = 00100110
```

La funzione `bits2str()` trasforma un numero binario in una stringa. Inizializzando `maschera` a uno otteniamo un valore binario in cui il bit più a destra è impostato a uno. Usando questa maschera, la funzione continua a controllare il bit più a destra. l'operazione di AND tra la maschera e il valore indica se il bit più a destra è uno oppure no. Se questo è il caso, un "1" è concatenato all'inizio della stringa. Altrimenti, è concatenato uno "0". Il valore è quindi spostato a destra di un bit e il ciclo continua finché non ci sono più bit.

Se il valore iniziale è zero, viene restituito semplicemente uno "0". Altrimenti, alla fine, al valore ottenuto vengono aggiunti degli zeri a sinistra, per arrivare a stringhe di lunghezza

multipla di 8, ossia contenenti un numero intero di byte. Questo è tipico dei computer moderni.

Il codice principale nella regola **BEGIN** mostra la differenza tra i valori decimale e ottale dello stesso numero. (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115), e poi mostra i risultati delle funzioni `compl()`, `lshift()` e `rshift()`.

**Attenzione. Non è tutto oro quel che luccica!**

In altri linguaggi, le operazioni "bit a bit" sono eseguite su valori interi, non su valori a virgola mobile. Come regola generale, tali operazioni funzionano meglio se eseguite su interi senza segno.

**gawk** tenta di trattare gli argomenti delle funzioni "bit a bit" come interi senza segno. Per questo motivo, gli argomenti negativi provocano un errore fatale.

In una normale operazione, per tutte queste funzioni, prima il valore a virgola mobile a doppia precisione viene convertito nel tipo intero senza segno di C più ampio, poi viene eseguita l'operazione "bit a bit". Se il risultato non può essere rappresentato esattamente come un tipo `double` di C, vengono rimossi i bit iniziali diversi da zero uno alla volta finché non sono rappresentati esattamente. Il risultato è poi nuovamente convertito in un tipo `double` di C.<sup>18</sup>

Comunque, quando si usa il calcolo con precisione arbitraria con l'opzione `-M` (si veda la [Capitolo 15 \[Calcolo con precisione arbitraria con gawk\]](#), pagina 379), il risultato può essere diverso. Questo è particolarmente evidente con la funzione `compl()`:

```
$ gawk 'BEGIN { print compl(42) }'
+ 9007199254740949
$ gawk -M 'BEGIN { print compl(42) }'
+ -43
```

Quel che avviene diventa chiaro quando si stampano i risultati in notazione esadecimale:

```
$ gawk 'BEGIN { printf "%#x\n", compl(42) }'
+ 0xffffffffffffd5
$ gawk -M 'BEGIN { printf "%#x\n", compl(42) }'
+ 0xffffffffffffd5
```

Quando si usa l'opzione `-M`, nel dettaglio, **gawk** usa gli interi a precisione arbitraria di GNU MP che hanno almeno 64 bit di precisione. Quando non si usa l'opzione `-M`, **gawk** memorizza i valori interi come regolari valori a virgola mobile con doppia precisione, che mantengono solo 53 bit di precisione. Inoltre, la libreria GNU MP tratta (o almeno sembra che tratti) il bit iniziale come un bit con segno; così il risultato con `-M` in questo caso è un numero negativo.

In breve, usare **gawk** per qualsiasi tipo di operazione "bit a bit", tranne le più semplici, probabilmente è una cattiva idea; caveat emptor!

<sup>18</sup> Per essere più chiari, la conseguenza è che **gawk** può memorizzare solo un determinato intervallo di valori interi; i numeri al di fuori di questo intervallo vengono ridotti per rientrare all'interno dell'intervallo.

### 9.1.7 Funzioni per conoscere il tipo di una variabile

`gawk` prevede due funzioni che permettono di conoscere il tipo di una variabile. Questo è necessario per scrivere del codice che visiti ogni elemento di un vettore di vettori (si veda la [Sezione 8.6 \[Vettori di vettori\]](#), pagina 190) e in altri contesti.

`isarray(x)`

Restituisce il valore 'vero' se `x` è un vettore. Altrimenti, restituisce 'falso'.

`typeof(x)`

Restituisce una delle stringhe seguenti, a seconda del tipo di `x`:

"array" `x` è un vettore.

"regexp" `x` è una *regexp* fortemente tipizzata (si veda la [Sezione 6.1.2.2 \[Costanti regexp fortemente tipizzate\]](#), pagina 118).

"number" `x` è un numero.

"string" `x` è una stringa.

"strnum" `x` è un numero che ha avuto origine da un input dell'utente, come un campo o il risultato di una chiamata a `split()`. (Cioè, `x` ha l'attributo *strnum*; si veda la [Sezione 6.3.2.1 \[Tipo stringa rispetto a tipo numero\]](#), pagina 131.)

"unassigned"

`x` è una variabile scalare a cui non è ancora stato assegnato un valore. Per esempio:

```
BEGIN {
    # crea a[1] ma non gli attribuisce alcun valore
    a[1]
    print typeof(a[1]) # unassigned
}
```

"untyped"

`x` non è stata usata per nulla; può diventare uno scalare o un vettore. Per esempio:

```
BEGIN {
    print typeof(x)      # x non è mai stato usato --> untyped
    mk_arr(x)
    print typeof(x)      # x ora è un vettore      --> array
}
```

```
function mk_arr(a) { a[1] = 1 }
```

`isarray()` torna utile in due occasioni. La prima è quando si visita un vettore multidimensionale: si può stabilire se un elemento è un vettore oppure no. La seconda è all'interno del corpo di una funzione definita dall'utente (argomento non ancora trattato; si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224), per determinare se un parametro è un vettore oppure no.

**NOTA:** Usare `isarray()` a livello globale per controllare le variabili non ha alcun senso. Si suppone infatti che chi scrive il programma sappia se una variabile

è un vettore oppure no. E in effetti, per come funziona **gawk**, se si passa una variabile che non sia stata usata in precedenza a **isarray()**, **gawk** la crea al volo, assegnandole il tipo scalare.

La funzione **typeof()** è generale; consente di determinare se una variabile o un parametro di funzione è uno scalare, un vettore, o una *regex* fortemente tipizzata.

L'uso di **isarray()** è deprecato; si dovrebbe usare **typeof()** al suo posto. Si dovrebbe sostituire ogni uso esistente di **'isarray(var)'** nei programmi esistenti con **'typeof(var) == "array"'**.

### 9.1.8 Funzioni per tradurre stringhe

**gawk** prevede strumenti per internazionalizzare i programmi **awk**. Questi sono costituiti dalle funzioni descritte nella lista seguente. Le descrizioni sono volutamente concise. Si veda la [Capitolo 13 \[Internazionalizzazione con gawk\], pagina 349](#), per un'esposizione completa. I parametri facoltativi sono racchiusi tra parentesi quadre ([ ]):

**bindtextdomain(directory [, dominio])**

Imposta la *directory* in cui **gawk** trova i file di traduzione dei messaggi, nel caso in cui non siano o non possano essere messi nelle *directory* “standard” (p.es., durante la fase di test di un programma). Restituisce la *directory* alla quale *dominio* è “connesso.”

Il default per *dominio* è il valore di **TEXTDOMAIN**. Se *directory* è la stringa nulla (“”), **bindtextdomain()** restituisce la connessione corrente per il *dominio* dato.

**dcgettext(stringa [, dominio [, categoria] ])**

Restituisce la traduzione di *stringa* nel dominio linguistico *dominio* per la categoria di localizzazione *categoria*. Il valore di default per *dominio* è il valore corrente di **TEXTDOMAIN**. Il valore di default per *categoria* è **"LC\_MESSAGES"**.

**dcngettext(stringa1, stringa2, numero [, dominio [, categoria] ])**

Restituisce la forma plurale usata per *numero* nella traduzione di *stringa1* e *stringa2* nel dominio di testo *dominio* per la categoria di localizzazione *categoria*. *stringa1* è la variante al singolare in inglese di un messaggio e *stringa2* è la variante al plurale in inglese dello stesso messaggio. Il valore di default per *dominio* è il valore corrente di **TEXTDOMAIN**. Il valore di default per *categoria* è **"LC\_MESSAGES"**.

## 9.2 Funzioni definite dall'utente

Programmi **awk** complessi spesso possono essere semplificati definendo delle apposite funzioni personali. Le funzioni definite dall'utente sono richiamate allo stesso modo di quelle predefinite (si veda la [Sezione 6.4 \[Chiamate di funzione\], pagina 138](#)), ma dipende dall'utente la loro definizione (cioè, dire ad **awk** cosa dovrebbero fare queste funzioni).

### 9.2.1 Come scrivere definizioni e cosa significano

*Risponde al vero affermare che la sintassi di awk per la definizione di variabili locali è semplicemente atroce.*

—Brian Kernighan

Definizioni di funzioni possono stare in una posizione qualsiasi tra le regole di un programma `awk`. Quindi, la forma generale di un programma `awk` è estesa per permettere l'inclusione di regole e la definizione di funzioni create dall'utente. Non è necessario che la definizione di una funzione sia posta prima del richiamo della stessa. Questo dipende dal fatto che `awk` legge l'intero programma, prima di iniziare ad eseguirlo.

La definizione di una funzione chiamata *nome* è simile a questa:

```
function nome([lista-parametri])
{
    corpo-della-funzione
}
```

Qui, *nome* è il nome della funzione da definire. Un nome di funzione valido è come un nome di variabile valido: una sequenza di lettere, cifre e trattini bassi che non inizia con una cifra. Anche qui, solo le 52 lettere inglesi maiuscole e minuscole possono essere usate in un nome di funzione. All'interno di un singolo programma `awk`, un dato nome può essere usato una sola volta: per una variabile, o per un vettore, o per una funzione.

*lista-parametri* è una lista opzionale degli argomenti della funzione e dei nomi delle variabili locali, separati da virgole. Quando la funzione viene chiamata, i nomi degli argomenti sono usati per contenere il valore degli argomenti passati con la chiamata.

Una funzione non può avere due parametri con lo stesso nome, e neanche un parametro con lo stesso nome della funzione stessa.

**ATTENZIONE:** Secondo lo standard POSIX, i parametri di funzione non possono avere lo stesso nome di una delle speciali variabili predefinite (si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)), e un parametro di funzione non può avere lo stesso nome di un'altra funzione. Non tutte le versioni di `awk` applicano queste limitazioni. `gawk` applica solo la prima di queste restrizioni. Se viene specificata l'opzione `--posix` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), anche la seconda restrizione viene applicata.

Le variabili locali si comportano come la stringa vuota se vengono utilizzate dove è richiesto il valore di una stringa, e valgono zero se utilizzate dove è richiesto un valore numerico. Questo è lo stesso comportamento delle variabili regolari a cui non sia stato ancora assegnato un valore. (Ci sono ulteriori informazioni riguardo alle variabili locali; si veda la [Sezione 9.2.5 \[Funzioni e loro effetti sul tipo di una variabile\], pagina 234](#).)

Il *corpo-della-funzione* è composto da istruzioni `awk`. Questa è la parte più importante della definizione, perché dice quello che la funzione dovrebbe realmente *fare*. I nomi di argomento esistono per consentire al corpo della funzione di gestire gli argomenti; le variabili locali esistono per consentire al corpo della funzione di memorizzare dei valori temporanei.

I nomi di argomento non sono sintatticamente distinti da quelli delle variabili locali. Invece, il numero di argomenti forniti quando la funzione viene chiamata determina quanti degli argomenti passati sono delle variabili. Quindi, se tre valori di argomento sono specificati, i primi tre nomi in *lista-parametri* sono degli argomenti e i rimanenti sono delle variabili locali.

Ne consegue che se il numero di argomenti richiesto non è lo stesso in tutte le chiamate alla funzione, alcuni dei nomi in *lista-parametri* possono essere in alcuni casi degli argomenti e in altri casi delle variabili locali. Un'altra angolatura da cui guardare questo fatto è che gli argomenti omessi assumono come valore di default la stringa nulla.

Solitamente, quando si scrive una funzione, si sa quanti nomi si intendono usare per gli argomenti e quanti si vogliono usare come variabili locali. È una convenzione in uso quella di aggiungere alcuni spazi extra tra gli argomenti e le variabili locali, per documentare come va utilizzata quella funzione.

Durante l'esecuzione del corpo della funzione, gli argomenti e i valori delle variabili locali nascondono, o *oscurano*, qualsiasi variabile dello stesso nome usata nel resto del programma. Le variabili oscurate non sono accessibili nel corpo della funzione, perché non c'è modo di accedere a esse mentre i loro nomi sono stati "occupati" dagli argomenti e dalle variabili locali. Tutte le altre variabili usate nel programma **awk** possono essere accedute o impostate normalmente nel corpo della funzione.

Gli argomenti e le variabili locali esistono solo finché il corpo della funzione è in esecuzione. Una volta che l'esecuzione è terminata, ritornano accessibili le variabili che erano oscurate durante l'esecuzione della funzione.

Il corpo della funzione può contenere espressioni che chiamano altre funzioni. Tali espressioni possono perfino chiamare direttamente, o indirettamente tramite un'altra funzione, la funzione stessa. Quando questo succede, la funzione è detta *ricorsiva*. Il fatto che una funzione richiami se stessa è detto *ricorsione*.

Tutte le funzioni predefinite restituiscono un valore al loro chiamante. Anche le funzioni definite dall'utente possono farlo, usando l'istruzione **return**, che è descritta in dettaglio nella [Sezione 9.2.4 \[L'istruzione \*\*return\*\*\], pagina 232](#). Molti dei successivi esempi in questa sezione usano l'istruzione **return**.

In molte implementazioni di **awk**, compreso **gawk**, la parola chiave **function** può essere abbreviata come **func**. (e.c.) Tuttavia, POSIX specifica solo l'uso della parola chiave **function**. Questo ha alcune implicazioni di carattere pratico. Se **gawk** è in modalità POSIX-compatibile (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)), la seguente istruzione *non* definisce una funzione:

```
func foo() { a = sqrt($1) ; print a }
```

Invece, definisce una regola che, per ogni record, concatena il valore della variabile '**func**' con il valore restituito dalla funzione '**foo**'. Se la stringa risultante è diversa dalla stringa nulla, l'azione viene eseguita. Questo non è con ogni probabilità quello che si desidera. (**awk** accetta questo input come sintatticamente valido, perché le funzioni, nei programmi **awk** possono essere usate prima che siano state definite.<sup>19</sup>)

Per essere certi che un programma **awk** sia portabile, va sempre usata la parola chiave **function** per definire una funzione.

### 9.2.2 Un esempio di definizione di funzione

Ecco un esempio di funzione definita dall'utente, di nome **stampa\_num()**, che ha come input un numero e lo stampa in un formato specifico:

```
function stampa_num(numero)
{
    printf "%6.3g\n", numero
}
```

<sup>19</sup> Questo programma in realtà non verrà eseguito, perché **foo()** risulterà essere una funzione non definita.

Per comprenderne il funzionamento, ecco una regola `awk` che usa la funzione `stampa_num()`:

```
$3 > 0      { stampa_num($3) }
```

Questo programma stampa, nel nostro formato speciale, tutti i terzi campi nei record in input che contengono un numero positivo. Quindi, dato il seguente input:

```
1.2  3.4  5.6  7.8
9.10 11.12 -13.14 15.16
17.18 19.20 21.22 23.24
```

questo programma, usando la nostra funzione per formattare i risultati, stampa:

```
5.6
21.2
```

La funzione seguente cancella tutti gli elementi in un vettore (si ricordi che gli spazi bianchi in soprannumero stanno a indicare l'inizio della lista delle variabili locali):

```
function cancella_vettore(a, i)
{
    for (i in a)
        delete a[i]
}
```

Quando si lavora con vettori, è spesso necessario cancellare tutti gli elementi in un vettore e ripartire con una nuova lista di elementi (si veda la [Sezione 8.4 \[L'istruzione delete, pagina 187\]](#)). Invece di dover ripetere questo ciclo ogni volta che si deve cancellare un vettore, un programma può limitarsi a effettuare una chiamata a `cancella_vettore()`. (Questo garantisce la portabilità. L'uso di `'delete vettore'` per cancellare il contenuto di un intero vettore è un'aggiunta relativamente recente<sup>20</sup> allo standard POSIX.)

Quello che segue è un esempio di una funzione ricorsiva. Prende come parametro di input una stringa e restituisce la stringa in ordine inverso. Le funzioni ricorsive devono sempre avere un test che interrompa la ricorsione. In questo caso, la ricorsione termina quando la stringa in input è già vuota:

```
function rev(stringa)
{
    if (stringa == "")
        return ""

    return (rev(substr(stringa, 2)) substr(stringa, 1, 1))
}
```

Se questa funzione è in un file di nome `rev.awk`, si può provare così:

```
$ echo "Non v'allarmate!" |
> gawk -e '{ print rev($0) }' -f rev.awk
⇩ !etamralla'v noN
```

La funzione C `ctime()` prende una marcatura temporale e la restituisce come una stringa, formattata come già sappiamo. Il seguente esempio usa la funzione predefinita `strftime()` (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali, pagina 214\]](#)) per creare una versione `awk` di `ctime()`:

```
# ctime.awk
```

---

<sup>20</sup> Verso la fine del 2012.

```
#
# versione awk della funzione C ctime(3)

function ctime(ts,    format)
{
    format = "%a %e %b %Y, %H.%M.%S, %Z"

    if (ts == 0)
        ts = systime()      # usare data e ora correnti per default
    return strftime(format, ts)
}
```

Si potrebbe pensare che la funzione `ctime()` possa usare `PROCINFO["strftime"]` come stringa di formato. Sarebbe un errore, perché si suppone che `ctime()` restituisca data e ora formattati in maniera standard, e qualche codice a livello utente potrebbe aver modificato in precedenza `PROCINFO["strftime"]`.

### 9.2.3 Chiamare funzioni definite dall'utente

*Chiamare una funzione* significa richiedere l'esecuzione di una funzione, la quale svolge il compito per cui è stata scritta. La chiamata di una funzione è un'espressione e il suo valore è quello restituito dalla funzione.

#### 9.2.3.1 Scrivere una chiamata di funzione

Una chiamata di funzione consiste nel nome della funzione seguito dagli argomenti racchiusi tra parentesi. Gli argomenti specificati nella chiamata sono costituiti da espressioni `awk`. Ogni volta che si esegue una chiamata queste espressioni vengono ricalcolate, e i loro valori diventano gli argomenti passati alla funzione. Per esempio, ecco una chiamata a `pippo()` con tre argomenti (il primo dei quali è una concatenazione di stringhe):

```
pippo(x y, "perdere", 4 * z)
```

**ATTENZIONE:** Caratteri bianchi (spazi e TAB) non sono permessi tra il nome della funzione e la parentesi aperta che apre la lista degli argomenti. Se per errore si lasciano dei caratteri bianchi, `awk` li interpreterebbe come se s'intendesse concatenare una variabile con un'espressione tra parentesi. Tuttavia, poiché si è usato un nome di funzione e non un nome di variabile, verrebbe emesso un messaggio di errore.

#### 9.2.3.2 Variabili locali e globali.

Diversamente da molti altri linguaggi, non c'è modo di rendere locale una variabile in un blocco `{ ... }` di `awk`, ma si può rendere locale una variabile di una funzione. È buona norma farlo quando una variabile serve solo all'interno di quella particolare funzione.

Per rendere locale una variabile per una funzione, basta dichiarare la variabile come argomento della funzione dopo gli argomenti richiesti dalla funzione (si veda la [Sezione 9.2.1 \[Come scrivere definizioni e cosa significano\]](#), pagina 224). Si consideri il seguente esempio, dove la variabile `i` è una variabile globale usata sia dalla funzione `pippo()` che dalla funzione `pluto()`:

```
function pluto()
```

```

{
    for (i = 0; i < 3; i++)
        print "in pluto i=" i
}

function pippo(j)
{
    i = j + 1
    print "in pippo i=" i
    pluto()
    print "in pippo i=" i
}

BEGIN {
    i = 10
    print "in BEGIN i=" i
    pippo(0)
    print "in BEGIN i=" i
}

```

L'esecuzione di questo script produce quanto segue, perché la stessa variabile `i` è usata sia nelle funzioni `pippo()` e `pluto()` sia a livello della regola `BEGIN`:

```

in BEGIN i=10
in pippo i=1
in pluto i=0
in pluto i=1
in pluto i=2
in pippo i=3
in BEGIN i=3

```

Se si vuole che `i` sia una variabile locale sia per `pippo()` che per `pluto()`, occorre procedere in questo modo (gli spazi extra prima della `i` sono una convenzione di codifica che serve a ricordare che `i` è una variabile locale, non un argomento):

```

function pluto(    i)
{
    for (i = 0; i < 3; i++)
        print "in pluto i=" i
}

function pippo(j,    i)
{
    i = j + 1
    print "in pippo i=" i
    pluto()
    print "in pippo i=" i
}

BEGIN {

```

```

        i = 10
        print "in BEGIN i=" i
        pippo(0)
        print "in BEGIN i=" i
    }

```

L'esecuzione della versione corretta dello script produce il seguente output:

```

in BEGIN i=10
in pippo i=1
in pluto i=0
in pluto i=1
in pluto i=2
in pippo i=1
in BEGIN i=10

```

Oltre a valori scalari (stringhe e numeri), si possono usare anche vettori locali. Usando come parametro il nome di un vettore, `awk` lo considera come tale, e lo tratta come locale alla funzione. Inoltre, chiamate ricorsive creano nuovi vettori. Si consideri questo esempio:

```

function qualche_funz(p1,      a)
{
    if (p1++ > 3)
        return

    a[p1] = p1

    qualche_funz(p1)

    printf("Al livello %d, indice %d %s trova in a\n",
           p1, (p1 - 1), (p1 - 1) in a ? "si" : "non si")
    printf("Al livello %d, indice %d %s trova in a\n",
           p1, p1, p1 in a ? "si" : "non si")
    print ""
}

BEGIN {
    qualche_funz(1)
}

```

Quando viene eseguito, questo programma produce il seguente output:

```

Al livello 4, indice 3 non si trova in a
Al livello 4, indice 4 si trova in a

Al livello 3, indice 2 non si trova in a
Al livello 3, indice 3 si trova in a

Al livello 2, indice 1 non si trova in a
Al livello 2, indice 2 si trova in a

```

### 9.2.3.3 Passare parametri di funzione per valore o per riferimento

In `awk`, quando si definisce una funzione, non c'è modo di dichiarare esplicitamente se gli argomenti sono passati *per valore* o *per riferimento*.

Invece, il modo con cui i parametri sono passati è determinato durante l'esecuzione del programma, quando la funzione è chiamata, nel rispetto della regola seguente: se l'argomento è una variabile di tipo vettoriale, questa è passata per riferimento. Altrimenti, l'argomento è passato per valore.

Passare un argomento per valore significa che quando una funzione è chiamata, le viene fornita una *copia* del valore di quell'argomento. Il chiamante può usare una variabile il cui valore calcolato viene passato come argomento, ma la funzione chiamata non la riconosce come variabile; riconosce solo il valore assunto dall'argomento. Per esempio, scrivendo il seguente codice:

```
pippo = "pluto"
z = mia_funzione(pippo)
```

non si deve pensare che l'argomento passato a `mia_funzione()` sia “la variabile `pippo`.” Invece, è corretto considerare l'argomento come la stringa il cui valore è “`pluto`”. Se la funzione `mia_funzione()` altera i valori delle sue variabili locali, ciò non influisce su nessun'altra variabile. Quindi, se `mia_funzione()` fa questo:

```
function mia_funzione(stringa)
{
    print stringa
    stringa = "zzz"
    print stringa
}
```

cambiando così il valore della variabile che è il suo primo argomento, ossia `stringa`, il valore di `pippo` per il chiamante *non* viene modificato. Il ruolo svolto da `pippo` nella chiamata di `mia_funzione()` termina quando il suo valore (“`pluto`”) viene calcolato. Se la variabile `stringa` esiste anche al di fuori di `mia_funzione()`, il corpo della funzione non può modificare questo valore esterno, perché esso rimane oscurato durante l'esecuzione di `mia_funzione()` e non può quindi essere visto o modificato.

Tuttavia, quando sono dei vettori a fungere da parametri alle funzioni, questi *non* vengono copiati. Invece, il vettore stesso è reso disponibile per essere manipolato direttamente dalla funzione. Questo è quel che si dice solitamente una *chiamata per riferimento*. Le modifiche effettuate su un vettore passato come parametro all'interno del corpo di una funzione *sono* visibili all'esterno della funzione.

**NOTA:** Modificare un vettore passato come parametro all'interno di una funzione può essere molto pericoloso se non si sta attenti a quel che si sta facendo.

Per esempio:

```
function cambialo(vettore, ind, nvalore)
{
    vettore[ind] = nvalore
}
```

```
BEGIN {
```

```

        a[1] = 1; a[2] = 2; a[3] = 3
        cambialo(a, 2, "due")
        printf "a[1] = %s, a[2] = %s, a[3] = %s\n",
            a[1], a[2], a[3]
    }

    stampa 'a[1] = 1, a[2] = due, a[3] = 3', perché cambialo() memorizza
    "due" nel secondo elemento di a.

```

Alcune implementazioni di `awk` consentono di chiamare una funzione che non è stata definita. Viene solo emesso un messaggio che descrive il problema al momento dell'esecuzione, se il programma tenta di chiamare quella funzione. Per esempio:

```

BEGIN {
    if (0)
        pippo()
    else
        pluto()
}
function pluto() { ... }
# si noti che 'pippo' non è definito

```

Poiché la condizione dell'istruzione `'if'` non risulterà mai verificata in questo caso, non è un problema reale il fatto che `pippo()` non sia stato definito. Solitamente, tuttavia, è un problema se un programma chiama una funzione indefinita.

Se si specifica l'opzione `--lint` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)), `gawk` elenca le chiamate a funzioni indefinite.

Alcune implementazioni di `awk` emettono un messaggio di errore se si usa l'istruzione `next` o `nextfile` (si veda la [Sezione 7.4.8 \[L'istruzione next\]](#), [pagina 159](#), e si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), [pagina 160](#)) all'interno di una funzione definita dall'utente. `gawk` non ha questa limitazione.

### 9.2.4 L'istruzione `return`

Come visto in parecchi esempi precedenti, il corpo di una funzione definita dall'utente può contenere un'istruzione `return`. Quest'istruzione restituisce il controllo a quella parte del del programma `awk` che ha effettuato la chiamata. Può anche essere usata per restituire un valore da usare nel resto del programma `awk`. Questo è un esempio:

```

return [espressione]

```

La parte *espressione* è facoltativa. Probabilmente per una svista, POSIX non definisce qual è il valore restituito, se si omette *espressione*. Tecnicamente parlando, questo rende il valore restituito indefinito, e quindi, indeterminato. In pratica, tuttavia, tutte le versioni di `awk` restituiscono semplicemente la stringa nulla, che vale zero se usata in un contesto che richiede un numero.

Un'istruzione `return` senza una *espressione* è considerata presente alla fine di ogni definizione di funzione. Quindi, se il flusso di esecuzione raggiunge la fine del corpo della funzione, tecnicamente la funzione restituisce un valore indeterminato. In pratica, restituisce la stringa nulla. `awk` non emette alcun messaggio di avvertimento se si usa il valore restituito di una tale funzione.

Talvolta può capitare di scrivere una funzione per quello che fa, non per quello che restituisce. Una tale funzione corrisponde a una funzione `void` in C, C++, o Java, o a una *procedure* in Ada. Quindi, può essere corretto non restituire alcun valore; basta fare attenzione a non usare poi il valore restituito da una tale funzione.

Quello che segue è un esempio di una funzione definita dall'utente che restituisce un valore che è il numero più alto presente tra gli elementi di un vettore:

```
function massimo(vettore, i, max)
{
    for (i in vettore) {
        if (max == "" || vettore[i] > max)
            max = vettore[i]
    }
    return max
}
```

La chiamata a `massimo()` ha un solo argomento, che è il nome di un vettore. Le variabili locali `i` e `max` non vanno intese come argomenti; nulla vieta di passare più di un argomento a `massimo()` ma i risultati sarebbero strani. Gli spazi extra prima di `i` nella lista dei parametri della funzione indicano che `i` e `max` sono variabili locali. È consigliabile seguire questa convenzione quando si definiscono delle funzioni.

Il programma seguente usa la funzione `massimo()`. Carica un vettore, richiama `massimo()`, e quindi elenca il numero massimo contenuto in quel vettore:

```
function massimo(vettore, i, max)
{
    for (i in vettore) {
        if (max == "" || vettore[i] > max)
            max = vettore[i]
    }
    return max
}

# Carica tutti i campi di ogni record in numeri.
{
    for (i = 1; i <= NF; i++)
        numeri[NR, i] = $i
}

END {
    print massimo(numeri)
}
```

Dato il seguente input:

```
1 5 23 8 16
44 3 5 2 8 26
256 291 1396 2962 100
-6 467 998 1101
99385 11 0 225
```

il programma trova (come si può immaginare) che 99.385 è il valore più alto contenuto nel vettore.

### 9.2.5 Funzioni e loro effetti sul tipo di una variabile

`awk` è un linguaggio molto fluido. È possibile che `awk` non sia in grado di stabilire se un identificativo rappresenta una variabile scalare o un vettore, prima dell'effettiva esecuzione di un programma. Ecco un esempio di programma commentato:

```
function pippo(a)
{
    a[1] = 1 # il parametro è un vettore
}

BEGIN {
    b = 1
    pippo(b) # non valido: errore fatale, tipi variabile in conflitto

    pippo(x) # x non inizializzato, diventa un vettore dinamicamente
    x = 1    # a questo punto, non permesso: errore in esecuzione
}
```

In questo esempio, la prima chiamata a `pippo()` genera un errore fatale, quindi `awk` non arriverà a segnalare il secondo errore. Se si commenta la prima chiamata e si riesegue il programma, a quel punto `awk` terminerà con un messaggio relativo al secondo errore. Solitamente queste cose non causano grossi problemi, ma è bene esserne a conoscenza.

## 9.3 Chiamate indirette di funzione

Questa sezione descrive un'estensione avanzata, specifica di `gawk`.

Spesso può essere utile ritardare la scelta della funzione da chiamare fino al momento in cui il programma viene eseguito. Per esempio, potrebbero esserci diversi tipi di record in input, ciascuno dei quali dovrebbe essere elaborato in maniera differente.

Solitamente, si userebbe una serie di istruzioni `if-else` per decidere quale funzione chiamare. Usando la chiamata *indiretta* a una funzione, si può assegnare il nome della funzione da chiamare a una variabile di tipo stringa, e usarla per chiamare la funzione. Vediamo un esempio.

Si supponga di avere un file con i punteggi ottenuti negli esami per i corsi che si stanno seguendo, e che si desideri ottenere la somma e la media dei punteggi ottenuti. Il primo campo è il nome del corso. I campi seguenti sono i nomi delle funzioni da chiamare per elaborare i dati, fino a un campo "separatore" `'dati:.'`. Dopo il separatore, fino alla fine del record, ci sono i vari risultati numerici di ogni test.

Ecco il file iniziale:

```
Biologia_101 somma media dati: 87.0 92.4 78.5 94.9
Chimica_305 somma media dati: 75.2 98.3 94.7 88.2
Inglese_401 somma media dati: 100.0 95.6 87.1 93.4
```

Per elaborare i dati, si potrebbe iniziare a scrivere:

```
{
```

```

corso = $1
for (i = 2; $i != "dati: "; i++) {
    if ($i == "somma")
        somma()    # elabora l'intero record
    else if ($i == "media")
        media()
    ...            # e così via
}
}

```

Questo stile di programmazione funziona, ma può essere scomodo. Con la chiamata *indiretta* di funzione, si può richiedere a **gawk** di usare il *valore* di una variabile come *nome* della funzione da chiamare.

La sintassi è simile a quella di una normale chiamata di funzione: un identificativo, seguito immediatamente da una parentesi aperta, qualche argomento, e una parentesi chiusa, con l'aggiunta di un carattere '@' all'inizio:

```

quale_funzione = "somma"
risultato = @quale_funzione()    # chiamata della funzione somma()

```

Ecco un intero programma che elabora i dati mostrati sopra, usando la chiamata indiretta di funzioni:

```

# chiamataindiretta.awk --- esempio di chiamata indiretta di funzioni

# media --- calcola la media dei valori dei campi $primo - $ultimo

function media(primo, ultimo,    somma, i)
{
    somma = 0;
    for (i = primo; i <= ultimo; i++)
        somma += $i

    return somma / (ultimo - primo + 1)
}

# somma --- restituisce la somma dei valori dei campi $primo - $ultimo

function somma(primo, ultimo,    totale, i)
{
    max = 0;
    for (i = primo; i <= ultimo; i++)
        totale += $i

    return totale
}

```

Queste due funzioni presuppongono che si lavori con dei campi; quindi, i parametri **primo** e **ultimo** indicano da quale campo iniziare e fino a quale arrivare. Per il resto, eseguono i calcoli richiesti, che sono i soliti:

```

# Per ogni record,
# stampa il nome del corso e le statistiche richieste
{
    nome_corso = $1
    gsub(/_/, " ", nome_corso) # Rimpiazza _ con spazi

    # trova campo da cui iniziare
    for (i = 1; i <= NF; i++) {
        if ($i == "dati:") {
            inizio = i + 1
            break
        }
    }

    printf("%s:\n", nome_corso)
    for (i = 2; $i != "dati:"; i++) {
        quale_funzione = $i
        printf("\t%s: <%s>\n", $i, @quale_funzione(inizio, NF) "")
    }
    print ""
}

```

Questo è il ciclo principale eseguito per ogni record. Stampa il nome del corso (con le linee basse sostituite da spazi). Trova poi l'inizio dei dati veri e propri, salvandolo in `inizio`. L'ultima parte del codice esegue un ciclo per ogni nome di funzione (da `$2` fino al separatore, `'dati:'`), chiamando la funzione il cui nome è specificato nel campo. La chiamata di funzione indiretta compare come parametro nella chiamata a `printf`. (La stringa di formattazione di `printf` usa `'%s'` come specificatore di formato, affinché sia possibile usare funzioni che restituiscano sia stringhe che numeri. Si noti che il risultato della chiamata indiretta è concatenato con la stringa nulla, in modo da farlo considerare un valore di tipo stringa).

Ecco il risultato dell'esecuzione del programma:

```

$ gawk -f chiamataindiretta.awk dati_dei_corsi
+ Biologia 101:
+   somma: <352.8>
+   media: <88.2>
+
+ Chimica 305:
+   somma: <356.4>
+   media: <89.1>
+
+ Inglese 401:
+   somma: <376.1>
+   media: <94.025>

```

La possibilità di usare la chiamata indiretta di funzioni è più potente di quel che si possa pensare inizialmente. I linguaggi C e C++ forniscono “puntatori di funzione” che sono un metodo per chiamare una funzione scelta al momento dell'esecuzione. Uno dei più noti usi

di questa funzionalità è la funzione C `qsort()`, che ordina un vettore usando il famoso algoritmo noto come “quicksort” (si veda [l’articolo di Wikipedia](#) per ulteriori informazioni). Per usare questa funzione, si specifica un puntatore a una funzione di confronto. Questo meccanismo consente di ordinare dei dati arbitrari in una maniera arbitraria.

Si può fare qualcosa di simile usando `gawk`, così:

```
# quicksort.awk --- Algoritmo di quicksort, con funzione di confronto
#                               fornita dall'utente

# quicksort --- Algoritmo di quicksort di C.A.R. Hoare.
#               Si veda Wikipedia o quasi ogni libro
#               che tratta di algoritmi o di informatica.

function quicksort(dati, sinistra, destra, minore_di, i, ultimo)
{
    if (sinistra >= destra) # non fa nulla se il vettore contiene
        return            # meno di due elementi

    quicksort_scambia(dati, sinistra, int((sinistra + destra) / 2))
    ultimo = sinistra
    for (i = sinistra + 1; i <= destra; i++)
        if (@minore_di(dati[i], dati[sinistra]))
            quicksort_scambia(dati, ++ultimo, i)
    quicksort_scambia(dati, sinistra, ultimo)
    quicksort(dati, sinistra, ultimo - 1, minore_di)
    quicksort(dati, ultimo + 1, destra, minore_di)
}

# quicksort_scambia --- funzione ausiliaria per quicksort,
#                       sarebbe meglio fosse nel programma principale

function quicksort_scambia(dati, i, j, salva)
{
    salva = dati[i]
    dati[i] = dati[j]
    dati[j] = salva
}
```

La funzione `quicksort()` riceve il vettore `dati`, gli indici iniziali e finali da ordinare (`sinistra` e `destra`), e il nome di una funzione che esegue un confronto “minore di”. Viene quindi eseguito l’algoritmo di `quicksort`.

Per fare uso della funzione di ordinamento, torniamo all’esempio precedente. La prima cosa da fare è di scrivere qualche funzione di confronto:

```
# num_min --- confronto numerico per minore di

function num_min(sinistra, destra)
{
    return ((sinistra + 0) < (destra + 0))
}
```

```

}

# num_magg_o_ug --- confronto numerico per maggiore o uguale

function num_magg_o_ug(sinistra, destra)
{
    return ((sinistra + 0) >= (destra + 0))
}

```

La funzione `num_magg_o_ug()` serve per ottenere un ordinamento decrescente (dal numero più alto al più basso); quando è usato per eseguire un test per “minore di”, in realtà fa l’opposto (maggiore o uguale a), il che conduce a ottenere dati ordinati in ordine decrescente.

Poi serve una funzione di ordinamento. Come parametri ha i numeri del campo iniziale e di quello finale, e il nome della funzione di confronto. Costruisce un vettore con i dati e richiama appropriatamente `quicksort()`; quindi formatta i risultati mettendoli in un’unica stringa:

```

# ordina --- ordina i dati a seconda di ‘confronta’
#                e li restituisce come un’unica stringa

function ordina(primo, ultimo, confronta,      dati, i, risultato)
{
    delete dati
    for (i = 1; primo <= ultimo; primo++) {
        dati[i] = $primo
        i++
    }

    quicksort(dati, 1, i-1, confronta)

    risultato = dati[1]
    for (i = 2; i in dati; i++)
        risultato = risultato " " dati[i]

    return risultato
}

```

Per finire, le due funzioni di ordinamento chiamano la funzione `ordina()`, passandole i nomi delle due funzioni di confronto:

```

# ascendente --- ordina i dati in ordine crescente
#                e li restituisce sotto forma di stringa

function ascendente(primo, ultimo)
{
    return ordina(primo, ultimo, "num_min")
}

# discendente --- ordina i dati in ordine decrescente
#                e li restituisce sotto forma di stringa

```

```
function discendente(primo, ultimo)
{
    return ordina(primo, ultimo, "num_magg_o_ug")
}
```

Ecco una versione estesa del file-dati:

```
Biologia_101 somma media ordina discendente dati: 87.0 92.4 78.5 94.9
Chimica_305 somma media ordina discendente dati: 75.2 98.3 94.7 88.2
Inglese_401 somma media ordina discendente dati: 100.0 95.6 87.1 93.4
```

Per finire, questi sono i risultati quando si esegue il programma in questa versione migliorata:

```
$ gawk -f quicksort.awk -f indirettacall.awk class_data2
+ Biologia 101:
+   somma: <352.8>
+   media: <88.2>
+   ascendente: <78.5 87.0 92.4 94.9>
+   discendente: <94.9 92.4 87.0 78.5>
+
+ Chimica 305:
+   somma: <356.4>
+   media: <89.1>
+   ascendente: <75.2 88.2 94.7 98.3>
+   discendente: <98.3 94.7 88.2 75.2>
+
+ Inglese 401:
+   somma: <376.1>
+   media: <94.025>
+   ascendente: <87.1 93.4 95.6 100.0>
+   discendente: <100.0 95.6 93.4 87.1>
```

Un altro esempio in cui le chiamate indirette di funzione sono utili è costituito dall'elaborazione di vettori. La descrizione si può trovare [Sezione 10.7 \[Attraversare vettori di vettori\]](#), pagina 277.

Occorre ricordarsi di anteporre il carattere '@' prima di una chiamata indiretta di funzione.

A partire dalla versione 4.1.2 di **gawk**, le chiamate indirette di funzione possono anche essere usate per chiamare funzioni predefinite e con funzioni di estensione (si veda la [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395). Ci sono alcune limitazioni nel richiamare in maniera indiretta delle funzioni predefinite, come qui dettagliato:

- Non si può passare una costante *regexp* a una funzione predefinita effettuando una chiamata di funzione indiretta.<sup>21</sup> Quanto sopra vale per le funzioni `sub()`, `gsub()`, `gensub()`, `match()`, `split()` e `patsplit()`.

<sup>21</sup> Questa limitazione potrebbe cambiare in una futura versione; per appurarlo, si controlli la documentazione che accompagna la versione in uso di **gawk**.

- Nel chiamare `sub()` o `gsub()`, sono accettati solo due argomenti, poiché queste funzioni sono atipiche, in quanto aggiornano il loro terzo argomento. Questo significa che verrà sempre aggiornato l'argomento di default, `$0`.

`gawk` fa del suo meglio per rendere efficiente la chiamata indiretta di funzioni. Per esempio, nel ciclo seguente:

```
for (i = 1; i <= n; i++)
    @quale_funzione()
```

`gawk` ricerca solo una volta quale funzione chiamare.

## 9.4 Sommario

- `awk` include delle funzioni predefinite e consente all'utente di definire le sue proprie funzioni.
- POSIX `awk` include tre tipi di funzioni predefinite: numeriche, di stringa, e di I/O. `gawk` prevede funzioni per ordinare vettori, per lavorare con valori che rappresentano marcature temporali, per la manipolazione di bit, per determinare il tipo di una variabile (vettoriale piuttosto che scalare), e programmi per l'internazionalizzazione e la localizzazione. `gawk` prevede anche parecchie estensioni ad alcune funzioni standard, tipicamente nella forma di ulteriori argomenti.
- Le funzioni accettano zero o più argomenti e restituiscono un valore. Le espressioni che specificano il valore di ogni argomento sono valutate completamente prima della chiamata a una funzione. L'ordine di valutazione di questi argomenti non è definito. Il valore restituito dalla funzione può essere ignorato.
- La gestione delle barre inverse in `sub()` e `gsub()` non è semplice. È più semplice nella funzione di `gawk` `gensub()`, ma anche questa funzione richiede attenzione quando la si usa.
- Le funzioni definite dall'utente consentono importanti funzionalità ma hanno anche alcune ineleganze sintattiche. In una chiamata di funzione non si può inserire alcuno spazio tra il nome della funzione e la parentesi sinistra aperta che inizia la lista degli argomenti. Inoltre, non c'è nessuna prescrizione per le variabili locali, e per questo la convenzione in uso è di aggiungere parametri extra, e di separarli visivamente dai parametri veri e propri inserendo degli spazi bianchi prima di essi.
- Le funzioni definite dall'utente possono chiamare altre funzioni definite dall'utente (oltre a quelle predefinite) e possono chiamare se stesse ricorsivamente. I parametri di funzione "nascondono" qualsiasi variabile globale che abbia lo stesso nome. Non si può usare il nome di una variabile riservata (p.es. `ARGC`) come nome di un parametro in funzioni definite dall'utente.
- I valori scalari sono passati alle funzioni definite dall'utente per valore. I parametri che sono dei vettori sono passati alle funzioni per riferimento; ogni modifica fatta dalla funzione a un parametro che sia un vettore è quindi visibile dopo aver eseguito quella funzione.
- L'istruzione `return` serve per tornare indietro da una funzione definita dall'utente. Un'espressione opzionale diviene il valore restituito dalla funzione. Una funzione può solo restituire valori di tipo scalare.

- Se una variabile che non è stata mai usata è passata a una funzione definita dall'utente, il modo con cui quella funzione elabora la variabile ne può determinare il tipo: o scalare o vettoriale.
- `gawk` consente la chiamata indiretta di funzioni usando una sintassi speciale. Impostando una variabile al nome di una funzione, si può determinare al momento dell'esecuzione che funzione sarà chiamata in un certo punto del programma. Questo equivale a usare un puntatore a una funzione nei linguaggi C e C++.



**Parte II:**

**Risoluzione di problemi con awk**



## 10 Una libreria di funzioni awk

La [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224, descrive come scrivere le proprie funzioni `awk` personali. Scrivere funzioni è importante, perché consente di incapsulare in un unico contenitore algoritmi e azioni di programma. Semplifica la programmazione, rendendo lo sviluppo di un programma più gestibile, e rendendo i programmi più leggibili.

Nel loro autorevole libro del 1976, *Software Tools*,<sup>1</sup> Brian Kernighan e P.J. Plauger hanno scritto:

A programmare bene non s'impara dai concetti generali, ma vedendo come programmi complessi possono essere resi puliti, facili da leggere, facili da mantenere e modificare, strutturati in modo comprensibile, efficienti e affidabili, applicando il buon senso e delle buone pratiche di programmazione. Lo studio attento e l'imitazione di buoni programmi conduce a una migliore scrittura.

In effetti, loro reputavano quest'idea tanto importante da mettere questa frase sulla copertina del libro. Poiché credo fermamente che la loro affermazione sia corretta, questo capitolo e il [Capitolo 11 \[Programmi utili scritti in awk\]](#), pagina 281, forniscono una corposa raccolta di codice da leggere e, si spera, da cui imparare.

Questo capitolo illustra una libreria di utili funzioni `awk`. Molti dei programmi descritti nel seguito di questo libro usano queste funzioni. Le funzioni sono illustrate progressivamente, dalla più semplice alla più complessa.

La [Sezione 11.3.7 \[Estrarre programmi da un file sorgente Texinfo\]](#), pagina 312, illustra un programma che si può usare per estrarre il codice sorgente degli esempi di funzioni di libreria e di programmi dal sorgente Texinfo di questo libro. (Questo è già stato fatto durante la preparazione della distribuzione di `gawk`.)

Chi avesse scritto una o più funzioni `awk` utili e di uso generale, e volesse metterle a disposizione della comunità degli utenti di `awk`, può leggere le informazioni contenute in [\[Come collaborare\]](#), pagina 11.

I programmi contenuti in questo capitolo e in [Capitolo 11 \[Programmi utili scritti in awk\]](#), pagina 281, utilizzano anche le funzionalità specifiche di `gawk`. Riscrivere questi programmi per implementazioni di `awk` diverse è piuttosto semplice:

- I messaggi di errore diagnostici sono inviati a `/dev/stderr`. Usare `'| "cat 1>&2"'` al posto di `'> "/dev/stderr"'` se il sistema in uso non ha un `/dev/stderr`, o se non è possibile usare `gawk`.
- Alcuni programmi usano `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160) per evitare di leggere gli input ancora non letti dal file in input corrente.
- Infine, alcuni dei programmi scelgono di ignorare la distinzione tra maiuscolo e minuscolo nei loro input, assegnando il valore uno a `IGNORECASE`. Si può ottenere quasi lo stesso effetto<sup>2</sup> aggiungendo la seguente regola all'inizio del programma:

```
# ignora maiuscolo/minuscolo
{ $0 = tolower($0) }
```

<sup>1</sup> Purtroppo, a distanza di oltre 35 anni, molte delle lezioni impartite da questo libro devono ancora essere apprese da un gran numero di programmatori professionisti.

<sup>2</sup> I risultati non sono identici. L'output del record trasformato sarà tutto in minuscolo, mentre `IGNORECASE` preserva il contenuto originale del record in input.

Inoltre, si verifichi che tutte le *regexp* e le costanti di tipo stringa usate nei confronti utilizzano solo lettere minuscole.

## 10.1 Dare un nome a variabili globali in funzioni di libreria

Per come si è sviluppato il linguaggio **awk**, le variabili sono o *globali* (usabili dall'intero programma) o *locali* (usabili solo in una specifica funzione). Non c'è uno stato intermedio analogo alle variabili **statiche** in C.

Le funzioni di libreria hanno spesso necessità di avere variabili globali da usare per conservare informazioni di stato tra successive chiamate alla funzione; per esempio, la variabile di `getopt()` `_opti` (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), [pagina 263](#)). Tali variabili vengono dette *private*, poiché le sole funzioni che devono usarle sono quelle della libreria.

Quando si scrive una funzione di libreria, si dovrebbe cercare di scegliere per le variabili private dei nomi che non entrano in conflitto con nessuna delle variabili usate da un'altra funzione di libreria o dal programma principale di un utente. Per esempio, un nome come `i` o `j` non è una buona scelta, perché i programmi a livello utente usano spesso nomi di variabile come questi per le proprie elaborazioni.

I programmi di esempio mostrati in questo capitolo usano per le loro variabili private nomi che iniziano con un trattino basso (`'_'`). Generalmente gli utenti non usano trattini bassi iniziali nei nomi di variabile, così questa convenzione riduce le possibilità che il nome di variabile coincida con un nome usato nel programma dell'utente.

Inoltre, parecchie funzioni di libreria usano un prefisso che suggerisce quale funzione o gruppo di funzioni usa quelle variabili; per esempio, `_pw_byname()` nelle routine che consultano la lista degli utenti (si veda la [Sezione 10.5 \[Leggere la lista degli utenti\]](#), [pagina 268](#)). L'uso di questa convenzione viene raccomandata, poiché riduce ulteriormente la possibilità di conflitti accidentali tra nomi di variabile. Si noti che questa convenzione può anche essere usata per i nomi di variabile e per i nomi delle funzioni private.<sup>3</sup>

Come nota finale sui nomi delle variabili, se una funzione rende disponibile una variabile globale per essere usata da un programma principale, è una buona convenzione quella di far iniziare i nomi di queste variabili con una lettera maiuscola; per esempio, `Opterr` e `Optind` di `getopt()` (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), [pagina 263](#)). La lettera maiuscola iniziale indica che la variabile è globale, mentre il fatto che il nome della variabile non è tutto in lettere maiuscole indica che la variabile non è una delle variabili predefinite di **awk**, come `FS`.

È importante anche che *tutte* le variabili nelle funzioni di libreria che non abbiano la necessità di essere conservate per tutta la durata del programma siano, di fatto, dichiarate come locali.<sup>4</sup> Se ciò non viene fatto, la variabile potrebbe essere usata accidentalmente nel programma dell'utente, conducendo a errori che sono molto difficili da scoprire:

```
function lib_func(x, y,    l1, l2)
{
```

<sup>3</sup> Sebbene tutte le routine di libreria si sarebbero potute riscrivere usando questa convenzione, ciò non è stato fatto, per far vedere come lo stile di programmazione in **awk** si è evoluto e per fornire alcuni spunti per questa spiegazione.

<sup>4</sup> L'opzione da riga di comando di **gawk** `--dump-variables` è utile per verificare questo.

```

...
# qualche_var dovrebbe essere locale ma per una svista non lo è
uso della variabile qualche_var
...
}

```

Una differente convenzione, comune nella comunità Tcl, è quella di usare un solo vettore associativo che contiene i valori necessari alle funzioni di libreria, o “package.” Questo riduce significativamente il numero degli effettivi nomi globali in uso. Per esempio, le funzioni descritte in [Sezione 10.5 \[Leggere la lista degli utenti\], pagina 268](#), potrebbero aver usato gli elementi di vettore `PW_data["inizializzato"]`, `PW_data["totale"]`, `PW_data["contatore"]`, e `PW_data["awklib"]`, al posto di `_pw_inizializzato`, `_pw_totale`, `_pw_awklib` e `_pw_contatore`.

Le convenzioni illustrate in questa sezione sono esattamente quello che indica il termine: convenzioni. Non si è obbligati a scrivere i propri programmi in questo modo: è solo auspicabile che lo si faccia.

## 10.2 Programmazione di tipo generale

Questa sezione illustra diverse funzioni che sono di uso generale nella programmazione.

### 10.2.1 Conversione di stringhe in numeri

La funzione `strtonum()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)) è un'estensione gawk. La seguente funzione fornisce un'implementazione per altre versioni di awk:

```

# mystrtonum --- converte stringhe in numeri

function mystrtonum(str,          ret, n, i, k, c)
{
    if (str ~ /^0[0-7]*$/) {
        # ottale
        n = length(str)
        ret = 0
        for (i = 1; i <= n; i++) {
            c = substr(str, i, 1)
            # index() restituisce 0 se c non è nella stringa,
            # e anche se c == "0"
            k = index("1234567", c)

            ret = ret * 8 + k
        }
    } else if (str ~ /^0[xX][[:xdigit:]]+$/) {
        # esadecimale
        str = substr(str, 3)      # via 0x iniziale
        n = length(str)
        ret = 0
        for (i = 1; i <= n; i++) {

```

```

        c = substr(str, i, 1)
        c = tolower(c)
        # index() restituisce 0 se c non è nella stringa,
        # e anche se c == "0"
        k = index("123456789abcdef", c)

        ret = ret * 16 + k
    }
} else if (str ~ \
/^[+-]?([0-9]+([.][0-9]*([Ee][0-9]+)?)?|([.][0-9]+([Ee][+-]?[0-9]+)?))$/ ) {
    # numero decimale, eventualmente in virgola mobile
    ret = str + 0
} else
    ret = "NON-UN-NUMERO"

return ret
}

# BEGIN {      # dati per un test
#     a[1] = "25"
#     a[2] = ".31"
#     a[3] = "0123"
#     a[4] = "0xdeadBEEF"
#     a[5] = "123.45"
#     a[6] = "1.e3"
#     a[7] = "1.32"
#     a[8] = "1.32E2"
#
#     for (i = 1; i in a; i++)
#         print a[i], strtonum(a[i]), mystrtonum(a[i])
# }

```

La funzione cerca dapprima numeri ottali in stile C (base 8). Se la stringa in input corrisponde all'espressione regolare che descrive i numeri ottali, `mystrtonum()` esegue il ciclo per ogni carattere presente nella stringa. Imposta `k` all'indice in "1234567" della cifra ottale corrente. Il valore di ritorno sarà lo stesso numero della cifra, o zero se il carattere non c'è, il che succederà per ogni cifra '0'. Questo si può fare, perché il test di *regex* nell'istruzione `if` assicura che vengano scelti per essere convertiti solo dei numeri ottali.

Una logica simile si applica al codice che ricerca e converte un valore esadecimale, che inizia con '0x' o '0X'. L'uso di `tolower()` semplifica il calcolo per trovare il valore numerico corretto per ogni cifra esadecimale.

Infine, se la stringa corrisponde alla (piuttosto complicata) *regex* per un intero decimale regolare o per un numero in virgola mobile, il calcolo '`ret = str + 0`' fa sì che `awk` converta il valore in un numero.

È incluso un programma di verifica commentato, in modo che la funzione possa essere verificata con `gawk` e il risultato confrontato con la funzione predefinita `strtonum()`.

### 10.2.2 Asserzioni

Quando si scrivono grossi programmi, spesso è utile sapere se una condizione o una serie di condizioni è verificata oppure no. Prima di procedere con un determinato calcolo, si fa un'affermazione su cosa si crede sia vero. Tale affermazione è nota come *asserzione*. Il linguaggio C fornisce un file di intestazione `<assert.h>` e una corrispondente macro `assert()` che un programmatore può utilizzare per fare asserzioni. Se l'asserzione risulta falsa, la macro `assert()` predispone la stampa di un messaggio diagnostico che descrive la condizione che sarebbe dovuta essere vera ma che non lo era, e poi fa terminare il programma. In C, l'uso di `assert()` è simile a questo:

```
#include <assert.h>

int myfunc(int a, double b)
{
    assert(a <= 5 && b >= 17.1);
    ...
}
```

Se l'asserzione è falsa, il programma stampa un messaggio simile a questo:

```
prog.c:5: asserzione falsa: 'a <= 5 && b >= 17.1'
```

Il linguaggio C rende possibile trasformare questa condizione in una stringa da usare per stampare il messaggio di diagnosi. Ciò in `awk` non è possibile, per cui la funzione `assert()` scritta in `awk` richiede anche una descrizione della condizione da verificare, in formato stringa. La funzione è la seguente:

```
# assert --- Verifica una condizione. Se questa è falsa esce.

function assert(condizione, stringa)
{
    if (! condizione) {
        printf("%s:%d: asserzione falsa: %s\n",
            FILENAME, FNR, stringa) > "/dev/stderr"
        _assert_exit = 1
        exit 1
    }
}

END {
    if (_assert_exit)
        exit 1
}
```

La funzione `assert()` verifica il parametro `condizione`. Se è falso, stampa un messaggio sullo standard error, usando il parametro `stringa` per descrivere la condizione non verificata. Poi imposta la variabile `_assert_exit` a uno ed esegue l'istruzione `exit`. L'istruzione `exit` salta alla regola `END`. Se la regola `END` trova vera la variabile `_assert_exit`, esce immediatamente.

Lo scopo della verifica nella regola `END` è quello di evitare che venga eseguita qualsiasi altra eventuale regola `END`. Quando un'asserzione non è verificata, il programma dovrebbe

uscire immediatamente. Se nessuna asserzione fallisce, `_assert_exit` è ancora falso quando la regola `END` è eseguita normalmente, e le eventuali altre regole `END` del programma vengono eseguite. Affinché tutto questo funzioni correttamente, `assert.awk` dev'essere il primo file sorgente che viene letto da `awk`. La funzione può essere usata in un programma nel seguente modo:

```
function miafunz(a, b)
{
    assert(a <= 5 && b >= 17.1, "a <= 5 && b >= 17.1")
    ...
}
```

Se l'asserzione non è verificata, si vedrà un messaggio simile a questo:

```
mydata:1357: asserzione falsa: a <= 5 && b >= 17.1
```

C'è un piccolo problema con questa versione di `assert()`. Come visto, una regola `END` viene automaticamente aggiunta al programma che chiama `assert()`. Normalmente, se un programma consiste solo di una regola `BEGIN`, i file in input e/o lo standard input non vengono letti. Tuttavia, ora che il programma ha una regola `END`, `awk` tenta di leggere i file-dati in input o lo standard input (si veda la [Sezione 7.1.4.1 \[Azioni di inizializzazione e pulizia\], pagina 148](#)), provocando molto probabilmente la sospensione del programma come se rimanesse in attesa di input.

C'è un modo per aggirare questo problema: assicurarsi che la regola `BEGIN` termini sempre con un'istruzione `exit`.

### 10.2.3 Arrotondamento di numeri

Il modo in cui `printf` e `sprintf()` (si veda la [Sezione 5.5 \[Usare l'istruzione printf per stampe sofisticate\], pagina 98](#)) effettuano l'arrotondamento spesso dipende dalla subroutine `C sprintf()` del sistema. Su molte macchine, l'arrotondamento di `sprintf()` è *statistico*, il che significa che non sempre arrotonda un .5 finale per eccesso, contrariamente alle normali aspettative. Nell'arrotondamento statistico, .5 arrotonda alla cifra pari, anziché sempre per eccesso, così 1.5 arrotonda a 2 e 4.5 arrotonda a 4. Ciò significa che se si sta usando un formato che fa arrotondamenti (p.es. `%.0f`), si dovrebbe controllare quello che fa il sistema che si sta usando. La seguente funzione esegue un arrotondamento tradizionale; potrebbe essere utile nel caso in cui l'istruzione `printf` di `awk` che si sta usando faccia degli arrotondamenti statistici:

```
# round.awk --- effettua arrotondamento tradizionale

function round(x, ival, aval, frazione)
{
    ival = int(x)    # parte intera, int() fa un troncamento

    # vedere se c'è la parte frazionale
    if (ival == x)   # nessuna parte frazionale
        return ival # nessun decimale

    if (x < 0) {
        aval = -x   # valore assoluto
```

```

    ival = int(aval)
    frazione = aval - ival
    if (frazione >= .5)
        return int(x) - 1    # -2.5 --> -3
    else
        return int(x)        # -2.3 --> -2
} else {
    frazione = x - ival
    if (frazione >= .5)
        return ival + 1
    else
        return ival
}
}

# codice per testare, commentato
# { print $0, round($0) }

```

#### 10.2.4 Il generatore di numeri casuali Cliff

Il **generatore di numeri casuali Cliff** è un generatore di numeri casuali molto semplice che “passa il test della sfera del rumore per la casualità non mostrando di avere alcuna struttura.” È programmato in modo molto semplice, in meno di 10 righe di codice `awk`:

```

# cliff_rand.awk --- generare numeri casuali con algoritmo di Cliff

BEGIN { _cliff_seme = 0.1 }

function cliff_rand()
{
    _cliff_seme = (100 * log(_cliff_seme)) % 1
    if (_cliff_seme < 0)
        _cliff_seme = - _cliff_seme
    return _cliff_seme
}

```

Questo algoritmo richiede un “seme” iniziale di 0,1. Ogni nuovo valore usa il seme corrente come input per il calcolo. Se la funzione predefinita `rand()` (si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), pagina 196) non è abbastanza casuale, si può tentare di usare al suo posto questa funzione.

#### 10.2.5 Tradurre tra caratteri e numeri

Un’implementazione commerciale di `awk` fornisce una funzione predefinita `ord()`, che prende un carattere e restituisce il valore numerico per quel carattere nella rappresentazione dei caratteri di quella particolare macchina. Se la stringa passata a `ord()` ha più di un carattere, viene usato solo il primo.

L’inverso di questa funzione è `chr()` (dalla funzione con lo stesso nome in Pascal), che, dato un numero, restituisce il corrispondente carattere. Entrambe le funzioni si possono

scrivere molto bene usando `awk`; non vi è nessun reale motivo per inglobarle come funzioni predefinite `awk`:

```
# ord.awk --- implementa ord e chr

# Identificatori globali:
#   _ord_:      valori numerici indicizzati da caratteri
#   _ord_init:  funzione per inizializzare _ord_

BEGIN    { _ord_init() }

function _ord_init(    basso, alto, i, t)
{
    basso = sprintf("%c", 7) # BEL è ascii 7
    if (basso == "\a") {    # ascii regolare
        basso = 0
        alto = 127
    } else if (sprintf("%c", 128 + 7) == "\a") {
        # ascii, con il primo bit a 1 (mark)
        basso = 128
        alto = 255
    } else {                # ebcdic(!)
        basso = 0
        alto = 255
    }

    for (i = basso; i <= alto; i++) {
        t = sprintf("%c", i)
        _ord_[t] = i
    }
}
```

Alcune spiegazioni riguardo ai numeri usati da `_ord_init()` non guastano. La serie di caratteri più importante oggi in uso è nota come ASCII.<sup>5</sup> Sebbene un byte a 8 bit possa contenere 256 valori distinti (da 0 a 255), ASCII definisce solo i caratteri che usano i valori da 0 a 127.<sup>6</sup> Nel lontano passato, almeno un produttore di microcomputer ha usato ASCII, ma con una parità di tipo *mark*, cioè con il bit più a sinistra sempre a 1. Questo significa che su questi sistemi i caratteri ASCII hanno valori numerici da 128 a 255. Infine, i grandi elaboratori centrali usano la serie di caratteri EBCDIC, che prevede tutti i 256 valori. Ci sono altre serie di caratteri in uso su alcuni sistemi più vecchi, ma non vale la pena di considerarli:

```
function ord(str,    c)
```

<sup>5</sup> La situazione sta però cambiando: molti sistemi usano Unicode, una serie di caratteri molto ampia che comprende ASCII al suo interno. Nei sistemi che supportano interamente Unicode, un carattere può occupare fino a 32 bit, facendo diventare i semplici test usati qui eccessivamente complessi.

<sup>6</sup> ASCII è stato esteso in molti paesi per usare i valori da 128 a 255 includendo i caratteri specifici del paese. Se il sistema in uso si avvale di queste estensioni, si può semplificare `_ord_init()` per eseguire un ciclo da 0 a 255.

```

{
    # solo il primo carattere è d'interesse
    c = substr(str, 1, 1)
    return _ord_[c]
}

function chr(c)
{
    # trasforma c in un numero aggiungendo uno 0
    return sprintf("%c", c + 0)
}

#### programma di verifica ####
# BEGIN {
#     for (;;) {
#         printf("immetti un carattere: ")
#         if (getline var <= 0)
#             break
#         printf("ord(%s) = %d\n", var, ord(var))
#     }
# }

```

Un ovvio miglioramento a queste funzioni è quello di spostare il codice per la funzione `_ord_init` nel corpo della regola `BEGIN`. Il programma è stato scritto inizialmente in questo modo per comodità di sviluppo. C'è un “programma di verifica” in una regola `BEGIN`, per verificare la funzione. È commentato, per poter essere eventualmente usato in produzione.

### 10.2.6 Trasformare un vettore in una sola stringa

Quando si fanno elaborazioni su stringhe, spesso è utile poter unire tutte le stringhe di un vettore in una lunga stringa. La funzione seguente, `join()`, svolge questo compito. Verrà utilizzata nel seguito in diversi programmi applicativi (si veda il [Capitolo 11 \[Programmi utili scritti in awk\]](#), pagina 281).

La buona progettazione di una funzione è importante; la funzione dev'essere generale, ma potrebbe anche avere un ragionevole comportamento di default. Viene chiamata con un vettore e anche con gli indici iniziale e finale degli elementi del vettore da riunire. Questo presuppone che gli indici del vettore siano numerici—una supposizione logica, dato che il vettore probabilmente è stato creato con `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198):

```

# join.awk --- trasforma un vettore in una stringa

function join(vettore, iniz, fine, separ, risultato, i)
{
    if (separ == "")
        separ = " "
    else if (separ == SUBSEP) # valore magico
        separ = ""
    risultato = vettore[iniz]

```

```

    for (i = iniz + 1; i <= fine; i++)
        risultato = risultato separ vettore[i]
    return risultato
}

```

Un ulteriore argomento opzionale è il separatore da usare quando si uniscono nuovamente le stringhe. Se il chiamante fornisce un valore non nullo, `join()` usa quello; se non viene fornito, per default ha un valore nullo. In questo caso, `join()` usa uno spazio singolo come separatore di default per le stringhe. Se il valore è uguale a `SUBSEP`, `join()` unisce le stringhe senza un separatore tra di esse. `SUBSEP` serve come valore “magico” per indicare che potrebbe non esserci un separatore tra le stringhe componenti.<sup>7</sup>

### 10.2.7 Gestione dell’ora del giorno

Le funzioni `systemtime()` e `strftime()` descritte nella [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), [pagina 214](#), forniscono la funzionalità minima necessaria per visualizzare l’ora del giorno in una forma intelligibile. Sebbene `strftime()` offra un’ampia gamma di formattazioni, i formati di controllo non sono facili da ricordare o intuitivamente ovvii quando si legge un programma.

La seguente funzione, `getlocaltime()`, riempie un vettore fornito dall’utente con informazioni sul tempo preformattate. Restituisce una stringa con data e ora corrente formattata come nel programma di utilità `date`:

```

# getlocaltime.awk --- ottiene l’ora del giorno in un formato usabile

# Restituisce una stringa nel formato dell’output di date(1)
# Riempie l’argomento del vettore time con valori individuali:
#   time["second"]      -- secondi (0 - 59)
#   time["minute"]     -- minuti (0 - 59)
#   time["hour"]       -- ore (0 - 23)
#   time["althour"]    -- ore (0 - 12)
#   time["monthday"]   -- giorno del mese (1 - 31)
#   time["month"]      -- mese dell’anno (1 - 12)
#   time["monthname"]  -- nome del mese
#   time["shortmonth"] -- nome breve del mese
#   time["year"]       -- anno modulo 100 (0 - 99)
#   time["fullyear"]   -- anno completo
#   time["weekday"]    -- giorno della settimana (domenica = 0)
#   time["altweekday"] -- giorno della settimana (lunedì = 0)
#   time["dayname"]    -- nome del giorno della settimana
#   time["shortdayname"] -- nome breve del giorno della settimana
#   time["yearday"]    -- giorno dell’anno (0 - 365)
#   time["timezone"]   -- abbreviazione del nome della zona di fuso orario
#   time["ampm"]       -- designazione di AM o PM
#   time["weeknum"]    -- numero della settimana, domenica primo giorno
#   time["altweeknum"] -- numero della settimana, lunedì primo giorno

```

<sup>7</sup> Sarebbe bello se `awk` avesse un operatore di assegnamento per la concatenazione. La mancanza di un esplicito operatore per la concatenazione rende le operazioni sulle stringhe più difficili di quanto potrebbero essere.

```

function getlocaltime(ora,    ret, adesso, i)
{
    # ottiene data e ora una volta sola,
    # evitando chiamate di sistema non necessarie
    adesso = systime()

    # restituisce l'output in stile date(1)
    ret = strftime("%a %e %b %Y, %H.%M.%S, %Z", adesso)

    # clear out target array
    delete time

    # immette i valori, forzando i valori numerici
    # a essere numerici aggiungendo uno 0
    time["second"]      = strftime("%S", adesso) + 0
    time["minute"]      = strftime("%M", adesso) + 0
    time["hour"]        = strftime("%H", adesso) + 0
    time["althour"]     = strftime("%I", adesso) + 0
    time["monthday"]    = strftime("%d", adesso) + 0
    time["month"]       = strftime("%m", adesso) + 0
    time["monthname"]   = strftime("%B", adesso)
    time["shortmonth"]  = strftime("%b", adesso)
    time["year"]        = strftime("%y", adesso) + 0
    time["fullyear"]    = strftime("%Y", adesso) + 0
    time["weekday"]     = strftime("%w", adesso) + 0
    time["altweekday"]  = strftime("%u", adesso) + 0
    time["dayname"]     = strftime("%A", adesso)
    time["shortdayname"] = strftime("%a", adesso)
    time["yearday"]     = strftime("%j", adesso) + 0
    time["timezone"]    = strftime("%Z", adesso)
    time["ampm"]        = strftime("%p", adesso)
    time["weeknum"]     = strftime("%U", adesso) + 0
    time["altweeknum"]  = strftime("%W", adesso) + 0

    return ret
}

```

Gli indici di stringa sono più facili da usare e leggere rispetto ai vari formati richiesti da `strftime()`. Il programma `alarm` illustrato in [Sezione 11.3.2 \[Un programma di sveglia\], pagina 303](#), usa questa funzione. Una progettazione più generica della funzione `getlocaltime()` avrebbe permesso all'utente di fornire un valore di data e ora opzionale da usare al posto della data/ora corrente.

### 10.2.8 Leggere un intero file in una sola volta

Spesso è conveniente avere il contenuto di un intero file disponibile in memoria, visto come un'unica stringa. Un modo chiaro e semplice per far ciò potrebbe essere questo:

```
function readfile(file,    temp, contenuto)
{
    if ((getline temp < file) < 0)
        return

    contenuto = temp
    while (getline temp < file) > 0)
        contenuto = contenuto RT tmp

    close(file)
    return contenuto
}
```

Questa funzione legge da `file` un record alla volta, ricostruendo l'intero contenuto del file nella variabile locale `contenuto`. Funziona, ma non è detto che sia efficiente.

La funzione seguente, basata su un suggerimento di Denis Shirokov, legge l'intero contenuto del file in un colpo solo:

```
# readfile.awk --- legge un intero file in un colpo solo

function readfile(file,    temp, salva_rs)
{
    salva_rs = RS
    RS = "^$"
    getline temp < file
    close(file)
    RS = salva_rs

    return temp
}
```

Funziona impostando `RS` a `^$`, un'espressione regolare che non trova nessuna corrispondenza se il file ha un contenuto. `gawk` legge i dati dal file contenuto in `temp`, tentando di trovare una corrispondenza con `RS`. La ricerca dopo ogni lettura non ha mai successo, ma termina rapidamente, e quindi `gawk` inserisce in `temp` l'intero contenuto del file. (Si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63, per informazioni su `RT` e `RS`.)

Se `file` è vuoto, il valore di ritorno è la stringa vuota. Quindi, il codice chiamante può usare qualcosa simile a questo:

```
contenuto = readfile("/qualche/percorso")
if (length(contenuto) == 0)
    # file vuoto ...
```

La verifica serve a determinare se il file è vuoto o no. Una verifica equivalente potrebbe essere `contenuto == ""`.

Si veda la [Sezione 16.7.10 \[Leggere un intero file in una stringa\]](#), pagina 454, per una funzione di estensione anch'essa finalizzata a leggere un intero file in memoria.

### 10.2.9 Stringhe con apici da passare alla shell

Michael Brennan propone il seguente modello di programma, da lui usato spesso:

```
#!/bin/sh
```

```
awkp='
...
;
```

```
specifica_programma_da_eseguire | awk "$awkp" | /bin/sh
```

Per esempio, un suo programma chiamato `flac-edit`<sup>8</sup> ha questa forma:

```
$ flac-edit -song="Whoope! That's Great" file.flac
```

`flac-edit` genera in output il seguente script, da passare alla shell (`/bin/sh`) per essere eseguito:

```
chmod +w file.flac
metaflac --remove-tag=TITLE file.flac
LANG=en_US.88591 metaflac --set-tag=TITLE='Whoope! That''''s Great' file.flac
chmod -w file.flac
```

Si noti la necessità di gestire gli apici nello script da passare alla shell. La funzione `shell_quote()` li prepara nel formato richiesto. `SINGLE` è la stringa di un solo carattere `'''` e `QSINGLE` è la stringa di tre caratteri `\"'\"'`:

```
# shell_quote --- pone tra apici un argomento da passare alla shell
```

```
function shell_quote(s,                # parametro
    SINGLE, QSINGLE, i, X, n, ret)    # variabili locali
{
    if (s == "")
        return "\"\""

    SINGLE = "\"x27"                # apice singolo
    QSINGLE = "\"\"x27\""            # apice singolo incapsulato
    n = split(s, X, SINGLE)

    ret = SINGLE X[1] SINGLE
    for (i = 2; i <= n; i++)
        ret = ret QSINGLE SINGLE X[i] SINGLE

    return ret
}
```

## 10.3 Gestione di file-dati

Questa sezione presenta funzioni utili per gestire file-dati da riga di comando.

<sup>8</sup> I file con suffisso *flac* contengono normalmente dei brani musicali. `metaflac` è un programma che permette di modificare le informazioni [*metadati*] contenute all'inizio di un file di tipo *flac*.

### 10.3.1 Trovare i limiti dei file-dati

Ognuna delle regole `BEGIN` ed `END` viene eseguita esattamente solo una volta, rispettivamente all'inizio e alla fine del programma `awk` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali `BEGIN` ed `END`\], pagina 148](#)). Una volta noi (gli autori di `gawk`) siamo venuti in contatto con un utente che erroneamente pensava che le regole `BEGIN` venissero eseguite all'inizio di ogni file-dati e le regole `END` alla fine di ogni file-dati.

Quando lo abbiamo informato che non era così, ci ha chiesto di aggiungere un nuovo criterio di ricerca speciale a `gawk`, chiamato `BEGIN_FILE` e `END_FILE`, che avesse il comportamento desiderato. Ci ha fornito anche il codice per far questo.

Non è stato necessario aggiungere a `gawk` questi criteri di ricerca speciali; il lavoro si può fare tranquillamente usando `awk`, come illustrato nel seguente programma di libreria. È strutturato in modo da chiamare due funzioni fornite dall'utente, `a_inizio_file()` e `a_fine_file()`, all'inizio e alla fine di ogni file-dati. Oltre a risolvere il problema in sole nove(!) righe di codice, questa soluzione è *portabile*; il programma funziona con qualsiasi implementazione di `awk`:

```
# transfile.awk
#
# Dare all'utente un aggancio per il passaggio
# da un file in input a quello successivo
#
# L'utente deve fornire le funzioni a_inizio_file() ed a_fine_file()
# ciascuna delle quali è invocata
# quando il file, rispettivamente,
# inizia e finisce.

FILENAME != _nome_file_vecchio {
    if (_nome_file_vecchio != "")
        a_fine_file(_nome_file_vecchio)
    _nome_file_vecchio = FILENAME
    a_inizio_file(FILENAME)
}

END { a_fine_file(FILENAME) }
```

Questo file [`transfile.awk`] dev'essere caricato prima del programma “principale” dell'utente, in modo che la regola ivi contenuta venga eseguita per prima.

Questa regola dipende dalla variabile di `awk` `FILENAME`, che cambia automaticamente per ogni nuovo file-dati. Il nome-file corrente viene salvato in una variabile privata, `_nome_file_vecchio`. Se `FILENAME` non è uguale a `_nome_file_vecchio`, inizia l'elaborazione di un nuovo file-dati ed è necessario chiamare `a_fine_file()` per il vecchio file. Poiché `a_fine_file()` dovrebbe essere chiamato solo se un file è stato elaborato, il programma esegue prima un controllo per assicurarsi che `_nome_file_vecchio` non sia la stringa nulla. Il programma assegna poi il valore corrente di nome-file a `_nome_file_vecchio` e chiama `a_inizio_file()` per il file. Poiché, come tutte le variabili di `awk`, `_nome_file_vecchio` è inizializzato alla stringa nulla, questa regola viene eseguita correttamente anche per il primo file-dati.

Il programma contiene anche una regola `END` per completare l'elaborazione per l'ultimo file. Poiché questa regola `END` viene prima di qualsiasi regola `END` contenuta nel programma “principale”, `a_fine_file()` viene chiamata per prima. Ancora una volta, l'utilità di poter avere più regole `BEGIN` ed `END` dovrebbe risultare chiara.

Se lo stesso file-dati compare due volte di fila sulla riga di comando, `a_fine_file()` e `a_inizio_file()` non vengono eseguite alla fine del primo passaggio e all'inizio del secondo passaggio. La versione seguente risolve il problema:

```
# ftrans.awk --- gestisce il passaggio da un file dati al successivo
#
# L'utente deve fornire le funzioni a_inizio_file() ed a_fine_file()

FNR == 1 {
    if (_filename_ != "")
        a_fine_file(_filename_)
    _filename_ = FILENAME
    a_inizio_file(FILENAME)
}

END { a_fine_file(_filename_) }
```

La [Sezione 11.2.7 \[Contare cose\]](#), [pagina 300](#), mostra come utilizzare questa funzione di libreria e come ciò semplifichi la scrittura del programma principale.

#### Allora perché gawk ha BEGINFILE e ENDFILE?

Ci si chiederà, probabilmente: perché, se le funzioni `a_inizio_file()` e `a_fine_file()` possono eseguire il compito, `gawk` prevede i criteri di ricerca `BEGINFILE` e `ENDFILE`?

Buona domanda. Normalmente, se `awk` non riesce ad aprire un file, questo fatto provoca un errore fatale immediato. In tal caso, per una funzione definita dall'utente non vi è alcun modo di affrontare il problema, giacché la chiamata verrebbe effettuata solo dopo aver aperto il file e letto il primo record. Quindi, la ragione principale di `BEGINFILE` è quella di dare un “aggancio” per gestire i file che non posso essere elaborati. `ENDFILE` esiste per simmetria, e perché consente facilmente una pulizia “file per file”. Per maggiori informazioni si faccia riferimento alla [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), [pagina 150](#).

### 10.3.2 Rileggere il file corrente

Un'altra richiesta per una nuova funzione predefinita è stata per una funzione per rileggere il file corrente. L'utente che l'ha richiesta non voleva dover usare `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), [pagina 83](#)) all'interno di un ciclo.

Comunque, se non si è nella regola `END`, è piuttosto facile fare in modo di chiudere il corrente file in input immediatamente e ricominciare a leggerlo dall'inizio. In mancanza di un nome migliore, chiameremo la funzione `rewind()`:

```
# rewind.awk --- ricarica il file corrente e ricomincia a leggerlo

function rewind(    i)
{
```

```

    # sposta in alto i rimanenti argomenti
    for (i = ARGC; i > ARGIND; i--)
        ARGV[i] = ARGV[i-1]

    # assicurarsi che gawk sappia raggiungerli
    ARGC++

    # fa sì che il file corrente sia il prossimo a essere letto
    ARGV[ARGIND+1] = FILENAME

    # do it
    nextfile
}

```

La funzione `rewind()` dipende dalla variabile `ARGIND` (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui `awk` fornisce informazioni\]](#), pagina 165), che è specifica di `gawk`. Dipende anche dalla parola chiave `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione `nextfile`\]](#), pagina 160). Perciò, non si dovrebbe chiamarla da una regola `ENDFILE`. (Non sarebbe peraltro necessario, perché `gawk` legge il file successivo non appena la regola `ENDFILE` finisce!)

Occorre prestare attenzione quando si chiama `rewind()`. Si può provocare una ricorsione infinita se non si sta attenti. Ecco un esempio di uso:

```

$ cat dati
→ a
→ b
→ c
→ d
→ e

$ cat test.awk
→ FNR == 3 && ! riavvolto {
→   riavvolto = 1
→   rewind()
→ }
→
→ { print FILENAME, FNR, $0 }

$ gawk -f rewind.awk -f test.awk dati
→ data 1 a
→ data 2 b
→ data 1 a
→ data 2 b
→ data 3 c
→ data 4 d
→ data 5 e

```

### 10.3.3 Controllare che i file-dati siano leggibili

Normalmente, se si fornisce ad **awk** un file-dati che non è leggibile, il programma si arresta con un errore fatale. Ci sono casi in cui sarebbe preferibile ignorare semplicemente questi file e proseguire.<sup>9</sup> Si può far questo facendo precedere il proprio programma **awk** dal seguente programma:

```
# readable.awk --- file di libreria per saltare file non leggibili

BEGIN {
    for (i = 1; i < ARGV; i++) {
        if (ARGV[i] ~ /^[a-zA-Z_][a-zA-Z0-9_]*=.*\/ \
            || ARGV[i] == "-" || ARGV[i] == "/dev/stdin")
            continue # assegnamento di variabile o standard input
        else if ((getline aperedere < ARGV[i]) < 0) # file non leggibile
            delete ARGV[i]
        else
            close(ARGV[i])
    }
}
```

Questo codice funziona, perché l'errore di **getline** non è fatale. Rimuovendo l'elemento da **ARGV** con **delete** si tralascia il file (perché non è più nella lista). Si veda anche [Sezione 7.5.3 \[Usare ARGV e ARGV\], pagina 172](#).

Poiché per i nomi delle variabili **awk** si possono usare solo lettere dell'alfabeto inglese, di proposito il controllo con espressioni regolari non usa classi di carattere come `[:alpha:]` e `[:alnum:]` (si veda la [Sezione 3.4 \[Usare espressioni tra parentesi quadre\], pagina 55](#)).

### 10.3.4 Ricerca di file di lunghezza zero

Tutte le implementazioni note di **awk** ignorano senza mandare alcun messaggio i file di lunghezza zero. Questo è un effetto collaterale del ciclo implicito di **awk** "leggi un record e confrontalo con le regole": quando **awk** cerca di leggere un record da un file vuoto, riceve immediatamente un'indicazione di fine-file [*end-of-file*], chiude il file, e prosegue con il successivo file-dati presente nella riga di comando, *senza* eseguire alcun codice di programma **awk** a livello di utente.

Usando la variabile **ARGIND** di **gawk** (si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)), è possibile accorgersi quando un file-dati è stato saltato. Simile al file di libreria illustrato in [Sezione 10.3.1 \[Trovare i limiti dei file-dati\], pagina 258](#), il seguente file di libreria chiama una funzione di nome **zerofile()** che l'utente deve fornire. Gli argomenti passati sono il nome-file e la posizione del file in **ARGV**:

```
# zerofile.awk --- file di libreria per elaborare file in input vuoti

BEGIN { Argind = 0 }

ARGIND > Argind + 1 {
```

<sup>9</sup> Il criterio di ricerca speciale **BEGINFILE** (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\], pagina 150](#)) fornisce un meccanismo alternativo per trattare i file che non sono leggibili. Tuttavia, il codice qui proposto fornisce una soluzione portabile.

```

        for (Argind++; Argind < ARGIND; Argind++)
            zerofile(ARGV[Argind], Argind)
    }

    ARGIND != Argind { Argind = ARGIND }

    END {
        if (ARGIND > Argind)
            for (Argind++; Argind <= ARGIND; Argind++)
                zerofile(ARGV[Argind], Argind)
    }

```

La variabile definita dall'utente `Argind` permette al programma `awk` di tracciare il suo percorso all'interno di `ARGV`. Ogniqualvolta il programma rileva che `ARGIND` è maggiore di `'Argind + 1'`, vuol dire che uno o più file vuoti sono stati tralasciati. L'azione chiama poi `zerofile()` per ogni file che è stato saltato, incrementando ogni volta `Argind`.

La regola `'Argind != ARGIND'` tiene semplicemente aggiornato `Argind` nel caso che non ci siano file vuoti.

Infine, la regola `END` prende in considerazione il caso di un qualsiasi file vuoto alla fine degli argomenti nella riga di comando. Si noti che nella condizione del ciclo `for`, la verifica usa l'operatore `'<='`, non `'<'`.

### 10.3.5 Trattare assegnamenti di variabile come nomi-file

Occasionalmente, potrebbe essere più opportuno che `awk` non elabori gli assegnamenti di variabile presenti sulla riga di comando (si veda la [Sezione 6.1.3.2 \[Assegnare una variabile dalla riga di comando\]](#), pagina 120). In particolare, se si ha un nome-file che contiene un carattere `'='`, `awk` tratta il nome-file come un assegnamento e non lo elabora.

Alcuni utenti hanno suggerito un'opzione aggiuntiva da riga di comando per `gawk` per disabilitare gli assegnamenti dati sulla riga di comando. Comunque, poche righe di codice di programmazione in un file di libreria hanno lo stesso effetto:

```

# noassign.awk --- file di libreria per evitare la necessità
# di una speciale opzione per disabilitare gli assegnamenti da
# riga di comando

function disable_assigns(argc, argv,    i)
{
    for (i = 1; i < argc; i++)
        if (argv[i] ~ /^[a-zA-Z_][a-zA-Z0-9_]*=.*/)
            argv[i] = ("./" argv[i])
}

BEGIN {
    if (Disabilita_variabili)
        disable_assigns(ARGC, ARGV)
}

```

Il programma va poi eseguito in questo modo:

```
awk -v Disabilita_variabili=1 -f noassign.awk -f vostro_programma.awk *
```

La funzione esegue un ciclo che esamina ogni argomento. Antepone ‘./’ a qualsiasi argomento che abbia la forma di un assegnamento di variabile, trasformando così quell’argomento in un nome-file.

L’uso di `Disabilita_variabili` consente di disabilitare assegnamenti di variabile dati sulla riga di comando al momento dell’invocazione, assegnando alla variabile un valore *vero*. Se non viene impostata la variabile è inizializzata a zero (cioè *falso*), e gli argomenti sulla riga di comando non vengono modificati.

## 10.4 Elaborare opzioni specificate sulla riga di comando

La maggior parte dei programmi di utilità su sistemi compatibili con POSIX prevedono opzioni presenti sulla riga di comando che possono essere usate per cambiare il modo in cui un programma si comporta. `awk` è un esempio di tali programmi (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). Spesso le opzioni hanno degli *argomenti* (cioè, dati che servono al programma per eseguire correttamente le opzioni specificate sulla riga di comando). Per esempio, l’opzione `-F` di `awk` richiede di usare la stringa specificata come separatore di campo. La prima occorrenza, sulla riga di comando, di `--` o di una stringa che non inizia con ‘-’ segnala la fine delle opzioni.

I moderni sistemi Unix hanno una funzione C chiamata `getopt()` per elaborare gli argomenti presenti sulla riga di comando. Il programmatore fornisce una stringa che descrive le opzioni, ognuna delle quali consiste di una sola lettera. Se un’opzione richiede un argomento, nella stringa l’opzione è seguita da due punti. A `getopt()` vengono anche passati il numero e i valori degli argomenti presenti sulla riga di comando e viene chiamata in un ciclo. `getopt()` scandisce gli argomenti della riga di comando cercando le lettere delle opzioni. A ogni passaggio del ciclo restituisce un carattere singolo che rappresenta la successiva lettera di opzione trovata, o ‘?’ se viene trovata un’opzione non prevista. Quando restituisce `-1`, non ci sono ulteriori opzioni da trattare sulla riga di comando.

Quando si usa `getopt()`, le opzioni che non prevedono argomenti possono essere raggruppate. Inoltre, le opzioni che hanno argomenti richiedono obbligatoriamente che l’argomento sia specificato. L’argomento può seguire immediatamente la lettera dell’opzione, o può costituire un argomento separato sulla riga di comando.

Dato un ipotetico programma che ha tre opzioni sulla riga di comando, `-a`, `-b` e `-c`, dove `-b` richiede un argomento, tutti i seguenti sono modi validi per invocare il programma:

```
programma -a -b pippo -c dati1 dati2 dati3
programma -ac -bpippo -- dati1 dati2 dati3
programma -acbpippo dati1 dati2 dati3
```

Si noti che quando l’argomento è raggruppato con la sua opzione, la parte rimanente dell’argomento è considerato come argomento dell’opzione. In quest’esempio, `-acbpippo` indica che tutte le opzioni `-a`, `-b` e `-c` sono presenti, e che ‘`pippo`’ è l’argomento dell’opzione `-b`.

`getopt()` fornisce quattro variabili esterne a disposizione del programmatore:

<code>optind</code>	L’indice nel vettore dei valori degli argomenti ( <code>argv</code> ) dove si può trovare il primo argomento sulla riga di comando che non sia un’opzione.
<code>optarg</code>	Il valore (di tipo stringa) dell’argomento di un’opzione.

**opterr** Solitamente `getopt()` stampa un messaggio di errore quando trova un'opzione non valida. Impostando **opterr** a zero si disabilita questa funzionalità. (un'applicazione potrebbe voler stampare un proprio messaggio di errore.)

**optopt** La lettera che rappresenta l'opzione sulla riga di comando.

Il seguente frammento di codice C mostra come `getopt()` potrebbe elaborare gli argomenti della riga di comando per `awk`:

```
int
main(int argc, char *argv[])
{
    ...
    /* stampa un appropriato messaggio */
    opterr = 0;
    while ((c = getopt(argc, argv, "v:f:F:W:")) != -1) {
        switch (c) {
            case 'f':    /* file */
                ...
                break;
            case 'F':    /* separatore di campo */
                ...
                break;
            case 'v':    /* assegnamento di variabile */
                ...
                break;
            case 'W':    /* estensione */
                ...
                break;
            case '?':
            default:
                messaggio_di_aiuto();
                break;
        }
    }
    ...
}
```

Incidentalmente, `gawk` al suo interno usa la funzione GNU `getopt_long()` per elaborare sia le normali opzioni che quelle lunghe in stile GNU (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)).

L'astrazione fornita da `getopt()` è molto utile ed è piuttosto comoda anche nei programmi `awk`. Di seguito si riporta una versione `awk` di `getopt()`. Questa funzione mette in evidenza uno dei maggiori punti deboli di `awk`, che è quello di essere molto carente nella manipolazione di caratteri singoli. Sono necessarie ripetute chiamate a `substr()` per accedere a caratteri singoli. (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)).<sup>10</sup>

<sup>10</sup> Questa funzione è stata scritta prima che `gawk` acquisisse la capacità di dividere le stringhe in caratteri singoli usando `"` come separatore. È stata lasciata così, poiché l'uso di `substr()` è più portabile.

La spiegazione della funzione viene data man mano che si elencano i pezzi di codice che la compongono:

```
# getopt.awk --- imita in awk la funzione di libreria C getopt(3)

# Variabili esterne:
#   Optind -- indice in ARGV del primo argomento che non è un'opzione
#   Optarg -- valore di tipo stringa dell'argomento dell'opzione corrente
#   Opterr -- se diverso da zero, viene stampato un messaggio diagnostico
#   Optopt -- lettera dell'opzione corrente

# Restituisce:
#   -1      alla fine delle opzioni
#   "?"     per un'opzione non riconosciuta
#   <c>     un carattere che rappresenta l'opzione corrente

# Dati privati:
#   _opti -- indice in un'opzione multipla, p.es., -abc
```

La funzione inizia con commenti che elencano e descrivono le variabili globali utilizzate, spiegano quali sono i valori di ritorno, il loro significato, e ogni altra variabile che è “esclusiva” a questa funzione di libreria. Tale documentazione è essenziale per qualsiasi programma, e in modo particolare per le funzioni di libreria.

La funzione `getopt()` dapprima controlla che sia stata effettivamente chiamata con una stringa di opzioni (il parametro `opzioni`). Se `opzioni` ha lunghezza zero, `getopt()` restituisce immediatamente `-1`:

```
function getopt(argc, argv, opzioni,   unaopz, i)
{
    if (length(opzioni) == 0)      # nessuna opzione specificata
        return -1

    if (argv[Optind] == "--") {   # fatto tutto
        Optind++
        _opti = 0
        return -1
    } else if (argv[Optind] !~ /^-[^:space:]/) {
        _opti = 0
        return -1
    }
}
```

Il successivo controllo cerca la fine delle opzioni. Due trattini (`--`) marcano la fine delle opzioni da riga di comando, e lo stesso fa qualsiasi argomento sulla riga di comando che non inizi con `'-'`. `Optind` è usato per scorrere il vettore degli argomenti presenti sulla riga di comando; mantiene il suo valore attraverso chiamate successive a `getopt()`, perché è una variabile globale.

L'espressione regolare che viene usata, `/^-[^:space:]/`, chiede di cercare un `'-'` seguito da qualsiasi cosa che non sia uno spazio vuoto o un carattere di due punti. Se l'argomento corrente sulla riga di comando non corrisponde a quest'espressione regolare, vuol dire che

non si tratta di un'opzione, e quindi viene terminata l'elaborazione delle opzioni. Continuando:

```

    if (_opti == 0)
        _opti = 2
    unaopz = substr(argv[Optind], _opti, 1)
    Optopt = unaopz
    i = index(opzioni, unaopz)
    if (i == 0) {
        if (Opterr)
            printf("%c -- opzione non ammessa\n", unaopz) > "/dev/stderr"
        if (_opti >= length(argv[Optind])) {
            Optind++
            _opti = 0
        } else
            _opti++
        return "?"
    }

```

La variabile `_opti` tiene traccia della posizione nell'argomento della riga di comando correntemente in esame (`argv[Optind]`). Se opzioni multiple sono raggruppate con un '-' (p.es., `-abx`), è necessario restituirle all'utente una per volta.

Se `_opti` è uguale a zero, viene impostato a due, ossia all'indice nella stringa del successivo carattere da esaminare ('-', che è alla posizione uno viene ignorato). La variabile `unaopz` contiene il carattere, ottenuto con `substr()`. Questo è salvato in `Optopt` per essere usato dal programma principale.

Se `unaopz` non è nella stringa delle opzioni `opzioni`, si tratta di un'opzione non valida. Se `Opterr` è diverso da zero, `getopt()` stampa un messaggio di errore sullo *standard error* che è simile al messaggio emesso dalla versione C di `getopt()`.

Poiché l'opzione non è valida, è necessario tralasciarla e passare al successivo carattere di opzione. Se `_opti` è maggiore o uguale alla lunghezza dell'argomento corrente della riga di comando, è necessario passare al successivo argomento, in modo che `Optind` venga incrementato e `_opti` sia reimpostato a zero. In caso contrario, `Optind` viene lasciato com'è e `_opti` viene soltanto incrementato.

In ogni caso, poiché l'opzione non è valida, `getopt()` restituisce "?". Il programma principale può esaminare `Optopt` se serve conoscere quale lettera di opzione è quella non valida. Proseguendo:

```

    if (substr(opzioni, i + 1, 1) == ":") {
        # ottiene un argomento di opzione
        if (length(substr(argv[Optind], _opti + 1)) > 0)
            Optarg = substr(argv[Optind], _opti + 1)
        else
            Optarg = argv[++Optind]
        _opti = 0
    } else
        Optarg = ""

```

Se l'opzione richiede un argomento, la lettera di opzione è seguita da due punti nella stringa `opzioni`. Se rimangono altri caratteri nell'argomento corrente sulla riga di comando (`argv[Optind]`), il resto di quella stringa viene assegnato a `Optarg`. Altrimenti, viene usato il successivo argomento sulla riga di comando (`'-xFOO'` piuttosto che `'-x FOO'`). In entrambi i casi, `_opti` viene reimpostato a zero, perché non ci sono altri caratteri da esaminare nell'argomento corrente sulla riga di comando. Continuando:

```
    if (_opti == 0 || _opti >= length(argv[Optind])) {
        Optind++
        _opti = 0
    } else
        _opti++
    return unaopz
}
```

Infine, se `_opti` è zero o maggiore della lunghezza dell'argomento corrente sulla riga di comando, significa che l'elaborazione di quest'elemento in `argv` è terminata, quindi `Optind` è incrementato per puntare al successivo elemento in `argv`. Se nessuna delle condizioni è vera, viene incrementato solo `_opti`, cosicché la successiva lettera di opzione può essere elaborata con la successiva chiamata a `getopt()`.

La regola `BEGIN` inizializza sia `Opterr` che `Optind` a uno. `Opterr` viene impostato a uno, perché il comportamento di default per `getopt()` è quello di stampare un messaggio diagnostico dopo aver visto un'opzione non valida. `Optind` è impostato a uno, perché non c'è alcun motivo per considerare il nome del programma, che è in `ARGV[0]`:

```
BEGIN {
    Opterr = 1    # il default è eseguire una diagnosi
    Optind = 1    # salta ARGV[0]

    # programma di controllo
    if (_getopt_test) {
        while ((_go_c = getopt(ARGC, ARGV, "ab:cd")) != -1)
            printf("c = <%c>, Optarg = <%s>\n",
                    _go_c, Optarg)
        printf("argomenti che non sono opzioni:\n")
        for (; Optind < ARGC; Optind++)
            printf("\tARGV[%d] = <%s>\n",
                    Optind, ARGV[Optind])
    }
}
```

Il resto della regola `BEGIN` è un semplice programma di controllo. Qui sotto si riportano i risultati di due esecuzioni di prova del programma di controllo:

```
$ awk -f getopt.awk -v _getopt_test=1 -- -a -cbARG bax -x
+ c = <a>, Optarg = <>
+ c = <c>, Optarg = <>
+ c = <b>, Optarg = <ARG>
+ argomenti che non sono opzioni:
+     ARGV[3] = <bax>
```

```

+          ARGV[4] = <-x>

$ awk -f getopt.awk -v _getopt_test=1 -- -a -x -- xyz abc
+ c = <a>, Optarg = <>
+ error x -- opzione non ammessa
+ c = <?>, Optarg = <>
+ argomenti che non sono opzioni:
+          ARGV[4] = <xyz>
+          ARGV[5] = <abc>

```

In entrambe le esecuzioni, il primo `--` fa terminare gli argomenti dati ad `awk`, in modo che `awk` non tenti di interpretare le opzioni `-a`, etc. come sue opzioni.

**NOTA:** Dopo che `getopt()` è terminato, il codice a livello utente deve eliminare tutti gli elementi di `ARGV` da 1 a `Optind`, in modo che `awk` non tenti di elaborare le opzioni sulla riga di comando come nomi-file.

Usare `#!` con l'opzione `-E` può essere d'aiuto per evitare conflitti tra le opzioni del proprio programma e quelle di `gawk`, poiché l'opzione `-E` fa sì che `gawk` abbandoni l'elaborazione di ulteriori opzioni. (si veda la [Sezione 1.1.4 \[Programmi awk da eseguire come script\]](#), pagina 19, e si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

Molti degli esempi presentati in [Capitolo 11 \[Programmi utili scritti in awk\]](#), pagina 281, usano `getopt()` per elaborare i propri argomenti.

## 10.5 Leggere la lista degli utenti

Il vettore `PROCINFO` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162) dà accesso ai numeri ID reale ed effettivo dell'utente e del gruppo e, se disponibili, alla serie di gruppi ulteriori a cui l'utente appartiene. Comunque, poiché questi sono numeri, non forniscono informazioni molto utili per l'utente medio. Bisogna trovare un modo per reperire informazioni sull'utente associate con i numeri ID dell'utente e del gruppo. Questa sezione illustra una raccolta di funzioni per ottenere le informazioni dalla lista gli utenti. Si veda la [Sezione 10.6 \[Leggere la lista dei gruppi\]](#), pagina 272, per una raccolta di funzioni simili per ottenere informazioni dalla lista dei gruppi.

Lo standard POSIX non definisce il file dove sono mantenute le informazioni degli utenti. Invece, fornisce il file d'intestazione `<pwd.h>` e diverse *subroutine* del linguaggio C per ottenere informazioni sugli utenti. La funzione primaria è `getpwent()`, che sta per “get password entry”. La “password” proviene dal file originale della lista degli utenti, `/etc/passwd`, che contiene le informazioni sugli utenti assieme alle password criptate (da cui il nome).<sup>11</sup>

Sebbene un programma `awk` possa semplicemente leggere `/etc/passwd` direttamente, questo file può non contenere tutte le informazioni su tutti gli utenti del sistema.<sup>12</sup> Per essere sicuri di poter produrre una versione leggibile e completa della banca dati degli utenti, è necessario scrivere un piccolo programma in C che chiama `getpwent()`. `getpwent()` viene definita in modo da restituire un puntatore a una `struct passwd`. Ogni volta che

<sup>11</sup> Questo è vero per le versioni più antiche di Unix. In quelle più recenti, la *password* di ogni utente è stata trasferita nel file `/etc/shadow`, un file non accessibile dall'utente normale. La struttura del file `/etc/passwd` è rimasta la stessa, ma al posto del campo *password* c'è una *x*.

<sup>12</sup> Capita spesso che le informazioni sulla password siano memorizzate in una lista in rete.

viene chiamata, restituisce l'elemento successivo della lista. Quando non ci sono più elementi, restituisce NULL, il puntatore nullo. Quando accade ciò, il programma C dovrebbe chiamare `endpwent()` per chiudere la lista.. Quel che segue è `pwcat`, un programma in C che “concatena” la lista delle password:

```
/*
 * pwcat.c
 *
 * Genera una versione stampabile della lista delle password.
 */
#include <stdio.h>
#include <pwd.h>

int
main(int argc, char **argv)
{
    struct passwd *p;

    while ((p = getpwent()) != NULL)
        printf("%s:%s:%ld:%ld:%s:%s:%s\n",
            p->pw_name, p->pw_passwd, (long) p->pw_uid,
            (long) p->pw_gid, p->pw_gecos, p->pw_dir, p->pw_shell);

    endpwent();
    return 0;
}
```

Se non si conosce il linguaggio C, non è il caso di preoccuparsi. L'output di `pwcat` è la lista degli utenti, nel formato tradizionale del file `/etc/passwd` con campi separati da due punti. I campi sono:

Login name

Il nome di login dell'utente.

Encrypted password

La password criptata dell'utente. Può non essere disponibile su alcuni sistemi.

User-ID

L'ID numerico dell'utente. (Su alcuni sistemi, è un numero di formato `long` [32bit] del linguaggio C, e non nel formato `int` [16bit]. Quindi, lo cambieremo in `long` per sicurezza.)

Group-ID

L'ID di gruppo numerico dell'utente. (Valgono le stesse considerazioni su `long` al posto di `int`.)

Full name

Il nome completo dell'utente, e talora altre informazioni associate all'utente.

Home directory

La directory di login (o “home”) (nota ai programmatori di shell come `$HOME`).

Login shell

Il programma che viene eseguito quando l'utente effettua l'accesso. Questo è comunemente una shell, come Bash.

Di seguito si riportano alcune righe di un possibile output di `pwcat`:

```
$ pwcat
+ root:x:0:1:Operator:/:/bin/sh
+ nobody:x:65534:65534:/:/
+ daemon:x:1:1:/:/
+ sys:x:2:2:/:/bin/csh
+ bin:x:3:3:/:bin:
+ arnold:x:2076:10:Arnold Robbins:/home/arnold:/bin/sh
+ miriam:x:112:10:Miriam Robbins:/home/miriam:/bin/sh
+ andy:x:113:10:Andy Jacobs:/home/andy:/bin/sh
...
```

Dopo quest'introduzione, di seguito si riporta un gruppo di funzioni per ottenere informazioni sugli utenti. Ci sono diverse funzioni, che corrispondono alle omonime funzioni C:

```
# passwd.awk --- accedere alle informazioni del file delle password

BEGIN {
    # modificare per adattarlo al sistema in uso
    _pw_awklib = "/usr/local/libexec/awk/"
}

function _pw_init(    oldfs, oldrs, olddol0, pwcat, using_fw, using_fpat)
{
    if (_pw_inizializzato)
        return

    oldfs = FS
    oldrs = RS
    olddol0 = $0
    using_fw = (PROCINFO["FS"] == "FIELDWIDTHS")
    using_fpat = (PROCINFO["FS"] == "FPAT")
    FS = ":"
    RS = "\n"

    pwcat = _pw_awklib "pwcat"
    while ((pwcat | getline) > 0) {
        _pw_byname[$1] = $0
        _pw_byuid[$3] = $0
        _pw_bycount[++_pw_totale] = $0
    }
    close(pwcat)
    _pw_contatore = 0
    _pw_inizializzato = 1
    FS = oldfs
    if (using_fw)
        FIELDWIDTHS = FIELDWIDTHS
}
```

```

    else if (using_fpat)
        FPAT = FPAT
    RS = oldrs
    $0 = olddolo
}

```

La regola `BEGIN` imposta una variabile privata col nome della directory in cui si trova `pwcat`. Poiché è destinata a essere usata da una routine di libreria di `awk`, si è scelto di metterla in `/usr/local/libexec/awk`; comunque, in un altro sistema potrebbe essere messa in una directory differente.

La funzione `_pw_init()` mette tre copie delle informazioni sull'utente in tre vettori associativi. I vettori sono indicizzati per nome-utente (`_pw_byname`), per numero di ID-utente (`_pw_byuid`), e per ordine di occorrenza (`_pw_bycount`). La variabile `_pw_inizializzato` è usata per efficienza, poiché in questo modo `_pw_init()` viene chiamata solo una volta.

Poiché questa funzione usa `getline` per leggere informazioni da `pwcat`, dapprima salva i valori di `FS`, `RS` e `$0`. Annota nella variabile `using_fw` se la suddivisione in campi usando `FIELDWIDTHS` è attiva o no. Far questo è necessario, poiché queste funzioni potrebbero essere chiamate da qualsiasi parte all'interno di un programma dell'utente, e l'utente può suddividere i record in campi a suo piacimento. Ciò rende possibile ripristinare il corretto meccanismo di suddivisione dei campi in un secondo momento. La verifica può restituire solo `vero` per `gawk`. Il risultato può essere `falso` se si usa `FS` o `FPAT`, o in qualche altra implementazione di `awk`.

Il codice che controlla se si sta usando `FPAT`, utilizzando `using_fpat` e `PROCINFO["FS"]`, è simile.

La parte principale della funzione usa un ciclo per leggere le righe della lista, suddividere le righe in campi, e poi memorizzare la riga all'interno di ogni vettore a seconda delle necessità. Quando il ciclo è completato, `_pw_init()` fa pulizia chiudendo la *pipe*, impostando `_pw_inizializzato` a uno, e ripristinando `FS` (e `FIELDWIDTHS` o `FPAT` se necessario), `RS` e `$0`. L'uso di `_pw_contatore` verrà spiegato a breve.

La funzione `getpwnam()` ha un nome utente come argomento di tipo stringa. Se quell'utente è presente nella lista, restituisce la riga appropriata. Altrimenti, il riferimento a un elemento inesistente del vettore aggiunge al vettore stesso un elemento il cui valore è la stringa nulla:

```

function getpwnam(nome)
{
    _pw_init()
    return _pw_byname[nome]
}

```

In modo simile, la funzione `getpwuid()` ha per argomento il numero ID di un utente. Se un utente con quel numero si trova nella lista, restituisce la riga appropriata. Altrimenti restituisce la stringa nulla:

```

function getpwuid(uid)
{
    _pw_init()
    return _pw_byuid[uid]
}

```

La funzione `getpwent()` scorre semplicemente la lista, un elemento alla volta. Usa `_pw_contatore` per tener traccia della posizione corrente nel vettore `_pw_bycount`:

```
function getpwent()
{
    _pw_init()
    if (_pw_contatore < _pw_totale)
        return _pw_bycount[++_pw_contatore]
    return ""
}
```

La funzione `endpwent()` reimposta `_pw_contatore` a zero, in modo che chiamate successive a `getpwent()` ricomincino da capo:

```
function endpwent()
{
    _pw_contatore = 0
}
```

In questa serie di funzioni, il fatto che ogni subroutine chiami `_pw_init()` per inizializzare il vettore della lista utenti risponde a una precisa scelta progettuale. Il lavoro necessario per eseguire un processo separato che generi la lista degli utenti, e l'I/O per esaminarla, si ha solo se il programma principale dell'utente chiama effettivamente una di queste funzioni. Se questo file di libreria viene caricato assieme a un programma dell'utente, ma non viene mai chiamata nessuna delle routine, non c'è nessun lavoro aggiuntivo richiesto in fase di esecuzione. (L'alternativa è quella di spostare il corpo di `_pw_init()` all'interno di una regola `BEGIN`, che esegua sempre `pwcat`. Questo semplifica il codice ma richiede di eseguire un processo extra il cui risultato potrebbe non essere mai utilizzato dal programma.)

A sua volta, chiamare ripetutamente `_pw_init()` non è troppo dispendioso, perché la variabile `_pw_inizializzato` permette di evitare di leggere i dati relativi agli utenti più di una volta. Se la preoccupazione è quella di minimizzare il tempo di esecuzione del programma `awk`, il controllo di `_pw_inizializzato` potrebbe essere spostato al di fuori di `_pw_init()` e duplicato in tutte le altre funzioni. In pratica, questo non è necessario, poiché la maggior parte dei programmi di `awk` è I/O-bound<sup>13</sup>, e una tale modifica complicherebbe inutilmente il codice.

Il programma `id` in [Sezione 11.2.3 \[Stampare informazioni sull'utente\]](#), pagina 290, usa queste funzioni.

## 10.6 Leggere la lista dei gruppi

Molto di quel che è stato detto nella [Sezione 10.5 \[Leggere la lista degli utenti\]](#), pagina 268, vale anche per la lista dei gruppi. Sebbene questa sia tradizionalmente contenuta in un file ben noto (`/etc/group`) in un altrettanto noto formato, lo standard POSIX prevede solo una serie di routine della libreria C (`<grp.h>` e `getgrent()`) per accedere a tali informazioni. Anche se il file suddetto è disponibile, potrebbe non contenere delle informazioni complete. Perciò, come per la lista degli utenti, è necessario avere un piccolo programma in C che

<sup>13</sup> I programmi si distinguono tradizionalmente in CPU-bound e I/O-bound. Quelli CPU-bound effettuano elaborazioni che non richiedono molta attività di I/O, come ad esempio la preparazione di una tavola di numeri primi. Quelli I/O bound leggono dei file, ma richiedono poca attività di elaborazione per ogni record letto.

genera la lista dei gruppi come suo output. `grcat`, un programma in C che fornisce la lista dei gruppi, è il seguente:

```
/*
 * grcat.c
 *
 * Genera una versione stampabile della lista dei gruppi.
 */
#include <stdio.h>
#include <grp.h>

int
main(int argc, char **argv)
{
    struct group *g;
    int i;

    while ((g = getgrent()) != NULL) {
        printf("%s:%s:%ld:", g->gr_name, g->gr_passwd,
                (long) g->gr_gid);
        for (i = 0; g->gr_mem[i] != NULL; i++) {
            printf("%s", g->gr_mem[i]);
            if (g->gr_mem[i+1] != NULL)
                putchar(',');
        }
        putchar('\n');
    }
    endgrent();
    return 0;
}
```

Ciascuna riga nella lista dei gruppi rappresenta un gruppo. I campi sono separati da due punti e rappresentano le seguenti informazioni:

Nome del gruppo

Il nome del gruppo.

Password del gruppo

La password del gruppo criptata. In pratica, questo campo non viene mai usato; normalmente è vuoto o impostato a 'x'.

Numero ID del gruppo

Il numero ID del gruppo in formato numerico; l'associazione del nome al numero dev'essere univoca all'interno di questo file. (Su alcuni sistemi, è un numero nel formato `long` [32bit] del linguaggio C, e non nel formato `int` [16bit]. Quindi, lo cambieremo in `long` per sicurezza.)

Lista dei membri del gruppo

Una lista di nomi utente separati da virgole. Questi utenti sono i membri del gruppo. I sistemi Unix moderni consentono agli utenti di appartenere a diversi gruppi simultaneamente. Se il sistema in uso è uno di questi, ci sono elementi

in PROCINFO che vanno da "group1" fino a "groupN" per quei numeri di ID di gruppo. (Si noti che PROCINFO è un'estensione gawk; si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162.)

Di seguito si riporta quel che grcat potrebbe produrre:

```
$ grcat
+ wheel:x:0:arnold
+ nogroup:x:65534:
+ daemon:x:1:
+ kmem:x:2:
+ staff:x:10:arnold,miriam,andy
+ other:x:20:
...
```

Qui ci sono le funzioni per ottenere informazioni relative alla lista dei gruppi. Ce ne sono diverse, costruite sul modello delle omonime funzioni della libreria C:

```
# group.awk --- funzioni per il trattamento del file dei gruppi
```

```
BEGIN {
    # Modificare in base alla struttura del proprio sistema
    _gr_awklib = "/usr/local/libexec/awk/"
}
```

```
function _gr_init(    oldfs, oldrs, olddol0, grcat,
                    using_fw, using_fpat, n, a, i)
```

```
{
    if (_gr_inizializzato)
        return

    oldfs = FS
    oldrs = RS
    olddol0 = $0
    using_fw = (PROCINFO["FS"] == "FIELDWIDTHS")
    using_fpat = (PROCINFO["FS"] == "FPAT")
    FS = ":"
    RS = "\n"

    grcat = _gr_awklib "grcat"
    while ((grcat | getline) > 0) {
        if ($1 in _gr_byname)
            _gr_byname[$1] = _gr_byname[$1] "," $4
        else
            _gr_byname[$1] = $0
        if ($3 in _gr_bygid)
            _gr_bygid[$3] = _gr_bygid[$3] "," $4
        else
            _gr_bygid[$3] = $0
    }
```

```

        n = split($4, a, "[ \\t]*,[ \\t]*")
        for (i = 1; i <= n; i++)
            if (a[i] in _gr_groupsbyuser)
                _gr_groupsbyuser[a[i]] = _gr_groupsbyuser[a[i]] " " $1
            else
                _gr_groupsbyuser[a[i]] = $1

        _gr_bycount[++_gr_contatore] = $0
    }
    close(grcat)
    _gr_contatore = 0
    _gr_inizializzato++
    FS = oldfs
    if (using_fw)
        FIELDWIDTHS = FIELDWIDTHS
    else if (using_fpat)
        FPAT = FPAT
    RS = oldrs
    $0 = olddol0
}

```

La regola `BEGIN` imposta una variabile privata con il nome della directory in cui si trova `grcat`. Poiché è destinata a essere usata da una routine di libreria di `awk`, si è scelto di metterla in `/usr/local/libexec/awk`; comunque, in un altro sistema potrebbe essere messa in una directory differente.

Queste routine seguono le stesse linee generali delle routine per formare la lista degli utenti (si veda la [Sezione 10.5 \[Leggere la lista degli utenti\], pagina 268](#)). La variabile `_gr_inizializzato` è usata per essere sicuri che la lista venga letta una volta sola. La funzione `_gr_init()` dapprima salva `FS`, `RS` e `$0`, e poi imposta `FS` e `RS` ai valori da usare nel passare in rassegna le informazioni di gruppo. Inoltre viene annotato se si stanno usando `FIELDWIDTHS` o `FPAT`, per poter poi ripristinare il meccanismo di suddivisione in campi appropriato.

Le informazioni sui gruppi sono memorizzate in diversi vettori associativi. I vettori sono indicizzati per nome di gruppo (`_gr_byname`), per numero ID del gruppo (`_gr_bygid`), e per posizione nella lista (`_gr_bycount`). C'è un vettore aggiuntivo indicizzato per nome utente (`_gr_groupsbyuser`), che è una lista, separata da spazi, dei gruppi ai quali ciascun utente appartiene.

Diversamente dalla lista degli utenti, è possibile avere più record nella lista per lo stesso gruppo. Questo è frequente quando un gruppo ha un gran numero di membri. Un paio di tali voci potrebbero essere come queste:

```

tvpeople:x:101:johnny,jay,arsenio
tvpeople:x:101:david,conan,tom,joan

```

Per questo motivo, `_gr_init()` controlla se un nome di gruppo o un numero di ID di gruppo è stato già visto. Se così fosse, i nomi utente vanno semplicemente concatenati con la precedente lista di utenti.<sup>14</sup>

Infine, `_gr_init()` chiude la *pipe* a `grcat`, ripristina `FS` (e `FIELDWIDTHS` o `FPAT`, se necessario), `RS` e `$0`, inizializza `_gr_contatore` a zero (per essere usato più tardi), e rende `_gr_inizializzato` diverso da zero.

La funzione `getgrnam()` ha come argomento un nome di gruppo, e se quel gruppo esiste, viene restituito.

Altrimenti, il riferimento a un elemento inesistente del vettore aggiunge al vettore stesso un elemento il cui valore è la stringa nulla:

```
function getgrnam(group)
{
    _gr_init()
    return _gr_byname[group]
}
```

La funzione `getgrgid()` è simile; ha come argomento un numero ID di gruppo e controlla le informazioni associate con quell'ID di gruppo:

```
function getgrgid(gid)
{
    _gr_init()
    return _gr_bygid[gid]
}
```

La funzione `getgruser()` non ha un equivalente in C. Ha come argomento un nome utente e restituisce l'elenco dei gruppi di cui l'utente è membro:

```
function getgruser(user)
{
    _gr_init()
    return _gr_groupsbyuser[user]
}
```

La funzione `getgrent()` scorre la lista un elemento alla volta. Usa `_gr_contatore` per ricordare la posizione corrente nella lista:

```
function getgrent()
{
    _gr_init()
    if (++_gr_contatore in _gr_bycount)
        return _gr_bycount[_gr_contatore]
    return ""
}
```

La funzione `endgrent()` reimposta `_gr_contatore` a zero in modo che `getgrent()` possa ricominciare da capo:

```
function endgrent()
{
```

---

<sup>14</sup> C'è un piccolo problema col codice appena illustrato. Supponiamo che la prima volta non ci siano nomi. Questo codice aggiunge i nomi con una virgola iniziale. Inoltre non controlla che ci sia un `$4`.

```

    _gr_contatore = 0
}

```

Come con le routine per la lista degli utenti, ogni funzione chiama `_gr_init()` per inizializzare i vettori. Così facendo si avrà il solo lavoro aggiuntivo di eseguire `grcat` se queste funzioni vengono usate (rispetto a spostare il corpo di `_gr_init()` all'interno della regola `BEGIN`).

La maggior parte del lavoro consiste nell'ispezionare la lista e nel costruire i vari vettori associativi. Le funzioni che l'utente chiama sono di per sé molto semplici, poiché si appoggiano sui vettori associativi di `awk` per fare il lavoro.

Il programma `id` in [Sezione 11.2.3 \[Stampare informazioni sull'utente\], pagina 290](#), usa queste funzioni.

## 10.7 Attraversare vettori di vettori

La [Sezione 8.6 \[Vettori di vettori\], pagina 190](#), trattava come `gawk` avere a disposizione vettori di vettori. In particolare, qualsiasi elemento di un vettore può essere uno scalare o un altro vettore. La funzione `isarray()` (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\], pagina 223](#)) permette di distinguere un vettore da uno scalare. La seguente funzione, `walk_array()`, attraversa ricorsivamente un vettore, stampando gli indici e i valori di ogni elemento. Viene chiamata col vettore e con una stringa che contiene il nome del vettore:

```

function walk_array(vett, nome,      i)
{
    for (i in vett) {
        if (isarray(vett[i]))
            walk_array(vett[i], (nome "[" i "]))
        else
            printf("%s[%s] = %s\n", nome, i, vett[i])
    }
}

```

Funziona eseguendo un ciclo su ogni elemento del vettore. Se un dato elemento è esso stesso un vettore, la funzione chiama sé stessa ricorsivamente, passando il sottovettore e una nuova stringa che rappresenta l'indice corrente. In caso contrario, la funzione stampa semplicemente il nome, l'indice e il valore dell'elemento. Qui di seguito si riporta un programma principale che ne mostra l'uso:

```

BEGIN {
    a[1] = 1
    a[2][1] = 21
    a[2][2] = 22
    a[3] = 3
    a[4][1][1] = 411
    a[4][2] = 42

    walk_array(a, "a")
}

```

Quando viene eseguito, il programma produce il seguente output:

```
$ gawk -f walk_array.awk
+ a[1] = 1
+ a[2][1] = 21
+ a[2][2] = 22
+ a[3] = 3
+ a[4][1][1] = 411
+ a[4][2] = 42
```

La funzione appena illustrata stampa semplicemente il nome e il valore di ogni elemento costituito da un vettore scalare. Comunque è facile generalizzarla, passandole il nome di una funzione da chiamare quando si attraversa un vettore. La funzione modificata è simile a questa:

```
function process_array(vett, nome, elab, do_arrays, i, nuovo_nome)
{
    for (i in vett) {
        nuovo_nome = (nome "[" i "]")
        if (isarray(vett[i])) {
            if (do_arrays)
                @elab(nuovo_nome, vett[i])
            process_array(vett[i], nuovo_nome, elab, do_arrays)
        } else
            @elab(nuovo_nome, vett[i])
    }
}
```

Gli argomenti sono i seguenti:

**vett**        Il vettore.

**nome**       Il nome del vettore (una stringa).

**elab**       Il nome della funzione da chiamare.

**do\_arrays**  
              Se vale vero, la funzione può gestire elementi che sono sottovettori.

Se devono essere elaborati sottovettori, questo vien fatto prima di attraversarne altri.

Quando viene eseguita con la seguente struttura, la funzione produce lo stesso risultato della precedente versione di `walk_array()`:

```
BEGIN {
    a[1] = 1
    a[2][1] = 21
    a[2][2] = 22
    a[3] = 3
    a[4][1][1] = 411
    a[4][2] = 42

    process_array(a, "a", "do_print", 0)
}
```

```
function do_print(nome, elemento)
{
    printf "%s = %s\n", nome, elemento
}
```

## 10.8 Riassunto

- Leggere i programmi è un eccellente metodo per imparare la "buona programmazione". Le funzioni e i programmi contenuti in questo capitolo e nel successivo si propongono questo obiettivo.
- Quando si scrivono funzioni di libreria di uso generale, si deve stare attenti ai nomi da dare alle variabili globali, facendo in modo che non entrino in conflitto con le variabili di un programma dell'utente.
- Le funzioni descritte qui appartengono alle seguenti categorie:

### Problemi generali

Conversione di numeri in stringhe, verifica delle asserzioni, arrotondamenti, generazione di numeri casuali, conversione di caratteri in numeri, unione di stringhe, ottenimento di informazioni su data e ora facilmente usabili, e lettura di un intero file in una volta sola

### Gestione dei file-dati

Annotazione dei limiti di un file-dati, rilettura del file corrente, ricerca di file leggibili, ricerca di file di lunghezza zero, e trattamento degli assegnamenti di variabili fatti sulla riga comando come nomi-file

### Elaborazione di opzioni sulla riga di comando

Una versione **awk** della funzione del C standard `getopt()`

### Lettura dei file degli utenti e dei gruppi

Due serie di routine equivalenti alle versioni disponibili nella libreria del linguaggio C

### Attraversamento di vettori di vettori

Due funzioni che attraversano un vettore di vettori fino in fondo

## 10.9 Esercizi

1. Nella [Sezione 10.3.4 \[Ricerca di file di lunghezza zero\]](#), [pagina 261](#), abbiamo illustrato il programma `zerofile.awk`, che fa uso della variabile di **gawk** `ARGIND`. Questo problema può essere risolto senza dipendere da `ARGIND`? Se sì, come?
2. Come esercizio collegato, rivedere quel codice per gestire il caso in cui un valore contenuto in `ARGV` sia un assegnamento di variabile.



## 11 Programmi utili scritti in awk

Il Capitolo 10 [Una libreria di funzioni **awk**], pagina 245, ha prospettato l'idea che la lettura di programmi scritti in un certo linguaggio possa aiutare a imparare quel linguaggio. Questo capitolo ripropone lo stesso tema, presentando una miscellanea di programmi **awk** per il piacere di leggerli. Ci sono tre sezioni. La prima spiega come eseguire i programmi descritti in questo capitolo.

La seconda illustra la versione **awk** di parecchi comuni programmi di utilità disponibili in POSIX. Si presuppone che si abbia già una certa familiarità con questi programmi, e che quindi i problemi a loro legati siano facilmente comprensibili. Riscrivendo questi programmi in **awk**, ci si può focalizzare sulle particolarità di **awk** nella risoluzione dei problemi di programmazione.

La terza sezione è una collezione di programmi interessanti. Essi mirano a risolvere un certo numero di differenti problemi di manipolazione e di gestione dati. Molti dei programmi sono brevi, per evidenziare la capacità di **awk** di fare molte cose usando solo poche righe di codice.

Molti di questi programmi usano le funzioni di libreria che sono state presentate nel Capitolo 10 [Una libreria di funzioni **awk**], pagina 245.

### 11.1 Come eseguire i programmi di esempio.

Per eseguire un dato programma, si procederebbe tipicamente così:

```
awk -f programma -- opzioni file
```

Qui, *programma* è il nome del programma **awk** (p.es. *cut.awk*), *opzioni* sono le opzioni sulla riga di comando per il programma che iniziano con un '-', e *file* sono i file-dati in input.

Se il sistema prevede il meccanismo '#!' di specifica di un *interprete* (si veda la Sezione 1.1.4 [Programmi **awk** da eseguire come *script*], pagina 19), si può invece eseguire direttamente un programma:

```
cut.awk -c1-8 i_miei_file > risultati
```

Se **awk** non è **gawk**, può invece essere necessario usare:

```
cut.awk -- -c1-8 i_miei_file > risultati
```

### 11.2 Reinventare la ruota per divertimento e profitto

Questa sezione presenta un certo numero di programmi di utilità POSIX implementati in **awk**. Riscrivere questi programmi in **awk** è spesso divertente, perché gli algoritmi possono essere espressi molto chiaramente, e il codice è normalmente molto semplice e conciso. Ciò è possibile perché **awk** facilita molto le cose al programmatore.

Va precisato che questi programmi non sono necessariamente scritti per sostituire le versioni installate sul sistema in uso. Inoltre, nessuno di questi programmi è del tutto aderente ai più recenti standard POSIX. Questo non è un problema; il loro scopo è di illustrare la programmazione in linguaggio **awk** che serve nel "mondo reale".

I programmi sono presentati in ordine alfabetico.

### 11.2.1 Ritagliare campi e colonne

Il programma di utilità `cut` seleziona, o “taglia” (*cut*), caratteri o campi dal suo standard input e li spedisce al suo standard output. I campi sono separati da caratteri TAB per default, ma è possibile fornire un’opzione dalla riga di comando per cambiare il campo *delimitatore* (cioè, il carattere che separa i campi). La definizione di campo di `cut` è meno generale di quella di `awk`.

Un uso comune del comando `cut` potrebbe essere quello di estrarre i nomi degli utenti correntemente collegati al sistema, a partire dall’output del comando `who`. Per esempio, la seguente pipeline genera una lista in ordine alfabetico, senza doppioni, degli utenti correntemente collegati al sistema:

```
who | cut -c1-8 | sort | uniq
```

Le opzioni per `cut` sono:

- c *lista*** Usare *lista* come lista di caratteri da ritagliare. Elementi all’interno della lista possono essere separati da virgole, e intervalli di caratteri possono essere separati da trattini. La lista ‘1-8,15,22-35’ specifica i caratteri da 1 a 8, 15, e da 22 a 35.
- f *lista*** Usare *lista* come lista di campi da ritagliare.
- d *delimitatore*** Usare *delimitatore* come carattere che separa i campi invece del carattere TAB.
- s** Evita la stampa di righe che non contengono il delimitatore di campo.

L’implementazione `awk` del comando `cut` usa la funzione di libreria `getopt()` (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\], pagina 263](#)) e la funzione di libreria `join()` (si veda la [Sezione 10.2.6 \[Trasformare un vettore in una sola stringa\], pagina 253](#)).

Il programma inizia con un commento che descrive le opzioni, le funzioni di libreria necessarie, e una funzione `sintassi()` che stampa un messaggio ed esce. `sintassi()` è chiamato se si specificano degli argomenti non validi:

```
# cut.awk --- implementa cut in awk

# Opzioni:
#   -f lista    Ritagliare campi
#   -d c        Carattere di delimitazione di campo
#   -c lista    Ritagliare caratteri
#
#   -s          Sopprimere righe che non contengono il delimitatore
#
# Richiede le funzioni di libreria getopt() e join()

function sintassi()
{
    print("sintassi: cut [-f lista] [-d c] [-s] [file...]" > "/dev/stderr"
    print("sintassi: cut [-c lista] [file...]" > "/dev/stderr"
    exit 1
}
```

Subito dopo c'è una regola `BEGIN` che analizza le opzioni della riga di comando. Questa regola imposta `FS` a un solo carattere `TAB`, perché quello è il separatore di campo di `cut` per default. La regola poi imposta il separatore di campo in output allo stesso valore del separatore di campo in input. Un ciclo che usa `getopt()` esamina le opzioni della riga di comando. Una e una sola delle variabili `per_campi` o `per_caratteri` è impostata a "vero", per indicare che l'elaborazione sarà fatta per campi o per caratteri, rispettivamente. Quando si ritaglia per caratteri, il separatore di campo in output è impostato alla stringa nulla:

```
BEGIN {
    FS = "\t"      # default
    OFS = FS
    while ((c = getopt(ARGC, ARGV, "sf:c:d:")) != -1) {
        if (c == "f") {
            per_campi = 1
            lista_campi = Optarg
        } else if (c == "c") {
            per_caratteri = 1
            lista_campi = Optarg
            OFS = ""
        } else if (c == "d") {
            if (length(Optarg) > 1) {
                printf("cut: usa il primo carattere di %s \
                    " come delimitatore\n", Optarg) > "/dev/stderr"
                Optarg = substr(Optarg, 1, 1)
            }
            fs = FS = Optarg
            OFS = FS
            if (FS == " ")      # mette specifica in formato awk
                FS = "[ ]"
        } else if (c == "s")
            sopprimi = 1
        else
            sintassi()
    }

    # Toglie opzioni da riga di comando
    for (i = 1; i < Optind; i++)
        ARGV[i] = ""
}
```

Nella scrittura del codice si deve porre particolare attenzione quando il delimitatore di campo è uno spazio. Usare un semplice spazio (" ") come valore per `FS` è sbagliato: `awk` separerebbe i campi con serie di spazi, `TAB`, e/o ritorni a capo, mentre devono essere separati solo da uno spazio. Per far questo, salviamo il carattere di spazio originale nella variabile `fs` per un uso futuro; dopo aver impostato `FS` a "[ ]" non è possibile usarlo direttamente per vedere se il carattere delimitatore di campo è nella stringa.

Si ricordi anche che dopo che si è finito di usare `getopt()` (come descritto nella Sezione 10.4 [Elaborare opzioni specificate sulla riga di comando], pagina 263), è necessario

eliminare tutti gli elementi del vettore `ARGV` da 1 a `Optind`, in modo che `awk` non tenti di elaborare le opzioni della riga di comando come nomi-file.

Dopo aver elaborato le opzioni della riga di comando, il programma verifica che le opzioni siano coerenti. Solo una tra le opzioni `-c` e `-f` dovrebbe essere presente, ed entrambe richiedono una lista di campi. Poi il programma chiama `prepara_lista_campi()` oppure `prepara_lista_caratteri()` per preparare la lista dei campi o dei caratteri:

```

    if (per_campi && per_caratteri)
        sintassi()

    if (per_campi == 0 && per_caratteri == 0)
        per_campi = 1    # default

    if (lista_campi == "") {
        print "cut: specificare lista per -c o -f" > "/dev/stderr"
        exit 1
    }

    if (per_campi)
        prepara_lista_campi()
    else
        prepara_lista_caratteri()
}

```

`prepara_lista_campi()` pone la lista campi, usando la virgola come separatore, in un vettore. Poi, per ogni elemento del vettore, controlla che esso non sia un intervallo. Se è un intervallo, lo fa diventare un elenco. La funzione controlla l'intervallo specificato, per assicurarsi che il primo numero sia minore del secondo. Ogni numero nella lista è aggiunto al vettore `lista_c`, che semplicemente elenca i campi che saranno stampati. Viene usata la normale separazione in campi di `awk`. Il programma lascia ad `awk` il compito di separare i campi:

```

function prepara_lista_campi(      n, m, i, j, k, f, g)
{
    n = split(lista_campi, f, ",")
    j = 1    # indice in lista_c
    for (i = 1; i <= n; i++) {
        if (index(f[i], "-") != 0) { # un intervallo
            m = split(f[i], g, "-")
            if (m != 2 || g[1] >= g[2]) {
                printf("cut: lista campi errata: %s\n",
                       f[i]) > "/dev/stderr"
                exit 1
            }
            for (k = g[1]; k <= g[2]; k++)
                lista_c[j++] = k
        } else
            lista_c[j++] = f[i]
    }
}

```

```

    ncampi = j - 1
}

```

La funzione `prepara_lista_caratteri()` è più complicata di `prepara_lista_campi()`. L'idea qui è di usare la variabile di gawk `FIELDWIDTHS` (si veda la [Sezione 4.6 \[Leggere campi di larghezza costante\]](#), [pagina 77](#)), che descrive input a larghezza costante. Quando si usa una lista di caratteri questo è proprio il nostro caso.

Impostare `FIELDWIDTHS` è più complicato che semplicemente elencare i campi da stampare. Si deve tener traccia dei campi da stampare e anche dei caratteri che li separano, che vanno saltati. Per esempio, supponiamo che si vogliano i caratteri da 1 a 8, 15, e da 22 a 35. Per questo si specifica `'-c 1-8,15,22-35'`. Il valore che corrisponde a questo nella variabile `FIELDWIDTHS` è `"8 6 1 6 14"`. Questi sono cinque campi, e quelli da stampare sono `$1`, `$3`, e `$5`. I campi intermedi sono *riempitivo* (*filler*), ossia è ciò che separa i dati che si desidera estrarre. `lista_c` lista i campi da stampare, e `t` traccia l'elenco completo dei campi, inclusi i riempitivi:

```

function prepara_lista_caratteri(    campo, i, j, f, g, n, m, t,
                                   filler, ultimo, lungo)
{
    campo = 1    # contatore totale campi
    n = split(lista_campi, f, ",")
    j = 1        # indice in lista_c
    for (i = 1; i <= n; i++) {
        if (index(f[i], "-") != 0) { # intervallo
            m = split(f[i], g, "-")
            if (m != 2 || g[1] >= g[2]) {
                printf("cut: lista caratteri errata: %s\n",
                       f[i]) > "/dev/stderr"
                exit 1
            }
            lungo = g[2] - g[1] + 1
            if (g[1] > 1) # calcola lunghezza del riempitivo
                filler = g[1] - ultimo - 1
            else
                filler = 0
            if (filler)
                t[campo++] = filler
            t[campo++] = lungo # lunghezza del campo
            ultimo = g[2]
            lista_c[j++] = campo - 1
        } else {
            if (f[i] > 1)
                filler = f[i] - ultimo - 1
            else
                filler = 0
            if (filler)
                t[campo++] = filler
            t[campo++] = 1
        }
    }
}

```

```

        ultimo = f[i]
        lista_c[j++] = campo - 1
    }
}
FIELDWIDTHS = join(t, 1, campo - 1)
ncampi = j - 1
}

```

Poi viene la regola che elabora i dati. Se l'opzione `-s` è stata specificata, il flag `sopprimi` è vero. La prima istruzione `if` accerta che il record in input abbia il separatore di campo. Se `cut` sta elaborando dei campi, e `sopprimi` è vero, e il carattere di separazione dei campi non è presente nel record, il record è ignorato.

Se il record è valido, `gawk` ha già separato i dati in campi, usando il carattere in `FS` o usando campi a lunghezza fissa e `FIELDWIDTHS`. Il ciclo scorre attraverso la lista di campi che si dovrebbero stampare. Il campo corrispondente è stampato se contiene dati. Se il campo successivo contiene pure dei dati, il carattere di separazione è scritto tra i due campi:

```

{
    if (per_campi && sopprimi && index($0, fs) == 0)
        next

    for (i = 1; i <= ncampi; i++) {
        if ($lista_c[i] != "") {
            printf "%s", $lista_c[i]
            if (i < ncampi && $lista_c[i+1] != "")
                printf "%s", OFS
        }
    }
    print ""
}

```

Questa versione di `cut` utilizza la variabile `FIELDWIDTHS` di `gawk` per ritagliare in base alla posizione dei caratteri. È possibile, in altre implementazioni di `awk` usare `substr()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)), ma la cosa è molto più complessa. La variabile `FIELDWIDTHS` fornisce una soluzione elegante al problema di suddividere la riga in input in singoli caratteri.

### 11.2.2 Ricercare espressioni regolari nei file

Il programma di utilità `egrep` ricerca occorrenze di espressioni regolari all'interno di file. Usa espressioni regolari che sono quasi identiche a quelle disponibili in `awk` (si veda il [Capitolo 3 \[Espressioni regolari\], pagina 49](#)). Si richiama così:

```
egrep [opzioni] 'espressione' file ...
```

*espressione* è un'espressione regolare. Normalmente, l'espressione regolare è protetta da apici per impedire alla shell di espandere ogni carattere speciale come nome-file. Normalmente, `egrep` stampa le righe per cui è stata trovata una corrispondenza. Se nella riga di comando si richiede di operare su più di un nome-file, ogni riga in output è preceduta dal nome del file, e dal segno due punti.

Le opzioni di `egrep` sono le seguenti:

- c       Stampa un contatore delle righe che corrispondono al criterio di ricerca, e non le righe stesse.
- s       Funziona in silenzio. Non si produce alcun output ma il codice di ritorno indica se il criterio di ricerca ha trovato almeno una corrispondenza.
- v       Inverte il senso del test. **egrep** stampa le righe che *non* soddisfano il criterio di ricerca ed esce con successo se il criterio di ricerca non è soddisfatto.
- i       Ignora maiuscolo/minuscolo sia nel criterio di ricerca che nei dati in input.
- l       Stampa (elenca) solo i nomi dei file che corrispondono, e non le righe trovate.
- e *espressione*  
       Usa *espressione* come *regex* da ricercare. Il motivo per cui è prevista l'opzione -e è di permettere dei criteri di ricerca che inizino con un '-'.

Questa versione usa la funzione di libreria `getopt()` (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), pagina 263) e il programma di libreria che gestisce il passaggio da un file dati al successivo (si veda la [Sezione 10.3.1 \[Trovare i limiti dei file-dati\]](#), pagina 258).

Il programma inizia con un commento descrittivo e poi c'è una regola `BEGIN` che elabora gli argomenti della riga di comando usando `getopt()`. L'opzione `-i` (ignora maiuscolo/minuscolo) è particolarmente facile da implementare con **gawk**; basta usare la variabile predefinita `IGNORECASE` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162):

```
# egrep.awk --- simula egrep in awk
#
# Opzioni:
#   -c   conta le righe trovate
#   -s   silenziosa: genera solo il codice di ritorno
#   -v   inverte test, successo se regex non presente
#   -i   ignora maiuscolo/minuscolo
#   -l   stampa solo nomi file
#   -e   espressione da ricercare
#
# Richiede la funzione getopt() e il programma di libreria
#   che gestisce il passaggio da un file dati al successivo

BEGIN {
    while ((c = getopt(ARGC, ARGV, "ce:svil")) != -1) {
        if (c == "c")
            conta_e_basta++
        else if (c == "s")
            non_stampare++
        else if (c == "v")
            inverti_test++
        else if (c == "i")
            IGNORECASE = 1
        else if (c == "l")
            solo_nomi_file++
    }
}
```

```

        else if (c == "e")
            criterio_di_ricerca = Optarg
        else
            sintassi()
    }

```

Nel seguito c'è il codice che gestisce il comportamento specifico di **egrep**. Se non è fornito esplicitamente alcun criterio di ricerca tramite l'opzione **-e**, si usa il primo argomento sulla riga di comando che non sia un'opzione. Gli argomenti della riga di comando di **awk** fino ad **ARGV[Optind]** vengono cancellati, in modo che **awk** non tenti di elaborarli come file. Se non è stato specificato alcun nome di file, si usa lo standard input, e se è presente più di un nome di file, lo si annota, in modo che i nomi-file vengano scritti prima di ogni riga di output corrispondente:

```

    if (criterio_di_ricerca == "")
        criterio_di_ricerca = ARGV[Optind++]

    for (i = 1; i < Optind; i++)
        ARGV[i] = ""
    if (Optind >= ARGC) {
        ARGV[1] = "--"
        ARGC = 2
    } else if (ARGC - Optind > 1)
        servono_nomi_file++

    #   if (IGNORECASE)
    #       criterio_di_ricerca = tolower(criterio_di_ricerca)
    #

```

Le ultime due righe sono solo dei commenti, in quanto non necessarie in **gawk**. Per altre versioni di **awk**, potrebbe essere necessario utilizzarle come istruzioni effettive (togliendo il "#").

Il prossimo insieme di righe dovrebbe essere decommentato se non si sta usando **gawk**. Questa regola converte in minuscolo tutti i caratteri della riga in input, se è stata specificata l'opzione **-i**.<sup>1</sup> La regola è commentata perché non è necessaria se si usa **gawk**:

```

#{
#   if (IGNORECASE)
#       $0 = tolower($0)
#}

```

La funzione **a\_inizio\_file()** è chiamata dalla regola in **ftrans.awk** quando ogni nuovo file viene elaborato. In questo caso, non c'è molto da fare; ci si limita a inizializzare una variabile **contatore\_file** a zero. **contatore\_file** serve a ricordare quante righe nel file corrente corrispondono al criterio di ricerca. Scegliere come nome di parametro **da\_buttare** indica che sappiamo che **a\_inizio\_file()** è chiamata con un parametro, ma che noi non siamo interessati al suo valore:

```

function a_inizio_file(da_buttare)

```

---

<sup>1</sup> Inoltre, qui si introduce un errore subdolo; se una corrispondenza viene trovata, viene inviata in output la riga tradotta, non quella originale.

```
{
    contatore_file = 0
}
```

La funzione `endfile()` viene chiamata dopo l'elaborazione di ogni file. Ha influenza sull'output solo quando l'utente desidera un contatore del numero di righe che sono state individuate. `non_stampare` è vero nel caso si desideri solo il codice di ritorno. `conta_e_basta` è vero se si desiderano solo i contatori delle righe trovate. `egrep` quindi stampa i contatori solo se sia la stampa che il conteggio delle righe sono stati abilitati. Il formato di output deve tenere conto del numero di file sui quali si opera. Per finire, `contatore_file` è aggiunto a `totale`, in modo da stabilire qual è il numero totale di righe che ha soddisfatto il criterio di ricerca:

```
function endfile(file)
{
    if (! non_stampare && conta_e_basta) {
        if (servono_nomi_file)
            print file ":" contatore_file
        else
            print contatore_file
    }

    totale += contatore_file
}
```

Si potrebbero usare i criteri di ricerca speciali `BEGINFILE` ed `ENDFILE` (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\], pagina 150](#)), ma in quel caso il programma funzionerebbe solo usando `gawk`. Inoltre, questo esempio è stato scritto prima che a `gawk` venissero aggiunti i criteri speciali `BEGINFILE` ed `ENDFILE`.

La regola seguente fa il grosso del lavoro per trovare righe corrispondenti al criterio di ricerca fornito. La variabile `corrisponde` è vera se la riga è individuata dal criterio di ricerca. Se l'utente chiede invece le righe che non corrispondono, il senso di `corrisponde` è invertito, usando l'operatore `!`. `contatore_file` è incrementato con il valore di `corrisponde`, che vale uno o zero, a seconda che la corrispondenza sia stata trovata oppure no. Se la riga non corrisponde, l'istruzione `next` passa ad esaminare il record successivo.

Vengono effettuati anche altri controlli, ma soltanto se non si sceglie di contare le righe. Prima di tutto, se l'utente desidera solo il codice di ritorno (`non_stampare` è vero), è sufficiente sapere che *una* riga nel file corrisponde, e si può passare al file successivo usando `nextfile`. Analogamente, se stiamo solo stampando nomi-file, possiamo stampare il nome-file, e quindi saltare al file successivo con `nextfile`. Infine, ogni riga viene stampata, preceduta, se necessario, dal nome-file e dai due punti:

```
{
    corrisponde = ($0 ~ criterio_di_ricerca)
    if (inverti_test)
        corrisponde = ! corrisponde

    contatore_file += corrisponde    # 1 o 0
```

```

    if (! corrisponde)
        next

    if (! conta_e_basta) {
        if (non_stampare)
            nextfile

        if (solo_nomi_file) {
            print nome_file
            nextfile
        }

        if (servono_nomi_file)
            print nome_file ":" $0
        else
            print
    }
}

```

La regola **END** serve a produrre il codice di ritorno corretto. Se non ci sono corrispondenze, il codice di ritorno è uno; altrimenti, è zero:

```

END {
    exit (totale == 0)
}

```

La funzione `sintassi()` stampa un messaggio per l'utente, nel caso siano state specificate opzioni non valide, e quindi esce:

```

function sintassi()
{
    print("sintassi: egrep [-csvil] [-e criterio_di_ricerca] [file ...]")\
    > "/dev/stderr"
    print("\n\tegrep [-csvil] criterio_di_ricerca [file ...]") > "/dev/stderr"
    exit 1
}

```

### 11.2.3 Stampare informazioni sull'utente

Il programma di utilità `id` elenca i numeri identificativi (ID) reali ed effettivi di un utente, e l'insieme dei gruppi a cui l'utente appartiene, se ve ne sono. `id` stampa i numeri identificativi di utente e di gruppo solo se questi sono differenti da quelli reali. Se possibile, `id` elenca anche i corrispondenti nomi di utente e di gruppo. L'output potrebbe essere simile a questo:

```

$ id
+ uid=1000(arnold) gid=1000(arnold) groups=1000(arnold),4(adm),7(lp),27(sudo)

```

Questa informazione è parte di ciò che è reso disponibile dal vettore `PROCINFO` di **gawk** (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162). Comunque, il programma di utilità `id` fornisce un output più comprensibile che non una semplice lista di numeri.

Ecco una versione semplice di `id` scritta in **awk**. Usa le funzioni di libreria che riguardano il database degli utenti (si veda la [Sezione 10.5 \[Leggere la lista degli utenti\]](#), pagina 268) e

le funzioni di libreria che riguardano il database dei gruppi (si veda la [Sezione 10.6 \[Leggere la lista dei gruppi\]](#), [pagina 272](#)) contenute nel [Capitolo 10 \[Una libreria di funzioni awk\]](#), [pagina 245](#).

Il programma è abbastanza semplice. Tutto il lavoro è svolto nella regola `BEGIN`. I numeri ID di utente e di gruppo sono ottenuti da `PROCINFO`. Il codice è ripetitivo. La riga nel database degli utenti che descrive l'ID reale dell'utente è divisa in parti, separate tra loro da `:`. Il nome è il primo campo. Un codice analogo è usato per l'ID effettivo, e per i numeri che descrivono i gruppi:

```
# id.awk --- implement id in awk
#
# Richiede funzioni di libreria per utente e gruppo
# l'output è:
# uid=12(pippo) euid=34(pluto) gid=3(paperino) \
# egid=5(paperina) groups=9(nove),2(due),1(un)

BEGIN {
    uid = PROCINFO["uid"]
    euid = PROCINFO["euid"]
    gid = PROCINFO["gid"]
    egid = PROCINFO["egid"]

    printf("uid=%d", uid)
    pw = getpwuid(uid)
    stampa_primo_campo(pw)

    if (euid != uid) {
        printf(" euid=%d", euid)
        pw = getpwuid(euid)
        stampa_primo_campo(pw)
    }

    printf(" gid=%d", gid)
    pw = getgrgid(gid)
    stampa_primo_campo(pw)

    if (egid != gid) {
        printf(" egid=%d", egid)
        pw = getgrgid(egid)
        stampa_primo_campo(pw)
    }

    for (i = 1; ("group" i) in PROCINFO; i++) {
        if (i == 1)
            printf(" gruppi=")
        group = PROCINFO["group" i]
        printf("%d", group)
    }
}
```

```

        pw = getgrgid(group)
        stampa_primo_campo(pw)
        if (("group" (i+1)) in PROCINFO)
            printf(",")
    }

    print ""
}

function stampa_primo_campo(str, a)
{
    if (str != "") {
        split(str, a, ":")
        printf("(%s)", a[1])
    }
}

```

Il test incluso nel ciclo `for` è degno di nota. Ogni ulteriore gruppo nel vettore `PROCINFO` ha come indice da `"group1"` a `"groupN"` dove il numero *N* è il numero totale di gruppi ulteriori). Tuttavia, non si sa quanti di questi gruppi ci siano per un dato utente.

Questo ciclo inizia da uno, concatena il valore di ogni iterazione con `"group"`, e poi usando l'istruzione `in` verifica se quella chiave è nel vettore (si veda la [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\]](#), pagina 179). Quando *i* è incrementato oltre l'ultimo gruppo presente nel vettore, il ciclo termina.

Il ciclo funziona correttamente anche se *non* ci sono ulteriori gruppi; in quel caso la condizione risulta falsa fin dal primo controllo, e il corpo del ciclo non viene mai eseguito.

La funzione `stampa_primo_campo()` semplicemente incapsula quelle parti di codice che vengono usate ripetutamente, rendendo il programma più conciso e ordinato. In particolare, inserendo in questa funzione il test per la stringa nulla consente di risparmiare parecchie righe di programma.

### 11.2.4 Suddividere in pezzi un file grosso

Il programma `split` divide grossi file di testo in pezzi più piccoli. La sua sintassi è la seguente:<sup>2</sup>

```
split [-contatore] [file] [prefisso]
```

Per default, i file di output avranno nome `xaa`, `xab`, e così via. Ogni file contiene 1.000 righe, con la probabile eccezione dell'ultimo file. Per cambiare il numero di righe in ogni file, va indicato un numero sulla riga di comando, preceduto da un segno meno (p.es., `'-500'` per file con 500 righe ognuno invece che 1.000). Per modificare i nomi dei file di output in qualcosa del tipo `miofileaa`, `miofileab`, e così via, va indicato un argomento ulteriore che specifica il prefisso del nome-file.

Ecco una versione di `split` in `awk`. Usa le funzioni `ord()` e `chr()` descritte nella [Sezione 10.2.5 \[Tradurre tra caratteri e numeri\]](#), pagina 251.

<sup>2</sup> Questo è la sintassi tradizionale. La versione POSIX del comando ha una sintassi differente, ma per lo scopo di questo programma `awk` la cosa non ha importanza.

Il programma dapprima imposta i suoi valori di default, e poi controlla che non siano stati specificati troppi argomenti. Quindi esamina gli argomenti uno alla volta. Il primo argomento potrebbe essere un segno meno seguito da un numero. Poiché il numero in questione può apparire negativo, lo si fa diventare positivo, e viene usato per contare le righe. Il nome del file-dati è per ora ignorato e l'ultimo argomento è usato come prefisso per i nomi-file in output:

```
# split.awk --- comando split scritto in awk
#
# Richiede le funzioni di libreria ord() e chr()
# sintassi: split [-contatore] [file] [nome_in_output]

BEGIN {
    outfile = "x"      # default
    contatore = 1000
    if (ARGC > 4)
        sintassi()

    i = 1
    if (i in ARGV && ARGV[i] ~ /^-[[[:digit:]]+$/ ) {
        contatore = -ARGV[i]
        ARGV[i] = ""
        i++
    }
    # testa argv nel caso che si legga da stdin invece che da file
    if (i in ARGV)
        i++      # salta nome file-dati
    if (i in ARGV) {
        outfile = ARGV[i]
        ARGV[i] = ""
    }

    s1 = s2 = "a"
    out = (outfile s1 s2)
}
```

La regola seguente fa il grosso del lavoro. `contatore_t` (contatore temporaneo) tiene conto di quante righe sono state stampate sul file di output finora. Se questo numero supera il valore di `contatore`, è ora di chiudere il file corrente e di iniziare a scriverne uno nuovo. Le variabili `s1` e `s2` sono usate per creare i suffissi da apporre a nome-file. Se entrambi arrivano al valore 'z', il file è troppo grosso. Altrimenti, `s1` passa alla successiva lettera dell'alfabeto e `s2` ricomincia da 'a':

```
{
    if (++contatore_t > contatore) {
        close(out)
        if (s2 == "z") {
            if (s1 == "z") {
                printf("split: %s è troppo grosso da suddividere\n",
```

```

                                nome_file) > "/dev/stderr"
                                exit 1
                                }
                                s1 = chr(ord(s1) + 1)
                                s2 = "a"
                                }
                                else
                                    s2 = chr(ord(s2) + 1)
                                out = (outfile s1 s2)
                                contatore_t = 1
                                }
                                print > out
                                }

```

La funzione `sintassi()` stampa solo un messaggio di errore ed esce:

```

function sintassi()
{
    print("sintassi: split [-num] [file] [nome_in_output]") > "/dev/stderr"
    exit 1
}

```

Questo programma è un po' approssimativo; conta sul fatto che **awk** chiuda automaticamente l'ultimo file invece di farlo in una regola **END**. Un altro presupposto del programma è che le lettere dell'alfabeto siano in posizioni consecutive nella codifica in uso, il che non è vero per i sistemi che usano la codifica EBCDIC.

### 11.2.5 Inviare l'output su più di un file

Il programma **tee** è noto come *pipe fitting* (tubo secondario). **tee** copia il suo standard input al suo standard output e inoltre lo duplica scrivendo sui file indicati nella riga di comando. La sua sintassi è la seguente:

```
tee [-a] file ...
```

L'opzione **-a** chiede a **tee** di aggiungere in fondo al file indicato, invece che riscriverlo dall'inizio.

La regola **BEGIN** dapprima fa una copia di tutti gli argomenti presenti sulla riga di comando, in un vettore di nome **copia**. **ARGV[0]** non serve, e quindi non viene copiato. **tee** non può usare **ARGV** direttamente, perché **awk** tenta di elaborare ogni nome-file in **ARGV** come dati in input.

Se il primo argomento è **-a**, la variabile flag **append** viene impostata a vero, e sia **ARGV[1]** che **copia[1]** vengono cancellati. Se **ARGC** è minore di due, nessun nome-file è stato fornito, e **tee** stampa un messaggio di sintassi ed esce. Infine, **awk** viene obbligato a leggere lo standard input impostando **ARGV[1]** al valore **"-"** e **ARGC** a due:

```

# tee.awk --- tee in awk
#
# Copia lo standard input a tutti i file di output indicati.
# Aggiunge in fondo se viene data l'opzione -a.
#
BEGIN {

```

```

for (i = 1; i < ARGV; i++)
    copia[i] = ARGV[i]

if (ARGV[1] == "-a") {
    append = 1
    delete ARGV[1]
    delete copia[1]
    ARGV--
}
if (ARGC < 2) {
    print "sintassi: tee [-a] file ..." > "/dev/stderr"
    exit 1
}
ARGV[1] = "-"
ARGC = 2
}

```

La seguente regola è sufficiente da sola a eseguire il lavoro. Poiché non è presente alcun criterio di ricerca, la regola è eseguita per ogni riga di input. Il corpo della regola si limita a stampare la riga su ogni file indicato nella riga di comando, e poi sullo standard output:

```

{
    # spostare l'if fuori dal ciclo ne velocizza l'esecuzione
    if (append)
        for (i in copia)
            print >> copia[i]
    else
        for (i in copia)
            print > copia[i]
    print
}

```

È anche possibile scrivere il ciclo così:

```

for (i in copia)
    if (append)
        print >> copia[i]
    else
        print > copia[i]

```

Questa forma è più concisa, ma anche meno efficiente. L'«if» è eseguito per ogni record e per ogni file di output. Duplicando il corpo del ciclo, l'«if» è eseguito solo una volta per ogni record in input. Se ci sono  $N$  record in input e  $M$  file di output, il primo metodo esegue solo  $N$  istruzioni «if», mentre il secondo esegue  $N*M$  istruzioni «if».

Infine, la regola END fa pulizia, chiudendo tutti i file di output:

```

END {
    for (i in copia)
        close(copia[i])
}

```

### 11.2.6 Stampare righe di testo non duplicate

Il programma di utilità **uniq** legge righe di dati ordinati sul suo standard input, e per default rimuove righe duplicate. In altre parole, stampa solo righe uniche; da cui il nome. **uniq** ha diverse opzioni. La sintassi è la seguente:

```
uniq [-udc [-n]] [+n] [file_input [file_output]]
```

Le opzioni per **uniq** sono:

- d**            Stampa solo righe ripetute (duplicate).
- u**            Stampa solo righe non ripetute (uniche).
- c**            Contatore righe. Quest'opzione annulla le opzioni **-d** e **-u**. Sia le righe ripetute che quelle non ripetute vengono contate.
- n**            Salta *n* campi prima di confrontare le righe. La definizione di campo è simile al default di **awk**: caratteri non bianchi, separati da sequenze di spazi e/o TAB.
- +n**            Salta *n* caratteri prima di confrontare le righe. Eventuali campi specificati con **'-n'** sono saltati prima.

#### *file\_input*

I dati sono letti dal file in input specificato sulla riga di comando, invece che dallo standard input.

#### *file\_output*

L'output generato è scritto sul file di output specificato, invece che sullo standard output.

Normalmente **uniq** si comporta come se siano state specificate entrambe le opzioni **-d** e **-u**.

**uniq** usa la funzione di libreria **getopt()** (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\], pagina 263](#)) e la funzione di libreria **join()** (si veda la [Sezione 10.2.6 \[Trasformare un vettore in una sola stringa\], pagina 253](#)).

Il programma inizia con una funzione **sintassi()** e poi con una breve spiegazione delle opzioni e del loro significato, sotto forma di commenti. La regola **BEGIN** elabora gli argomenti della riga di comando e le opzioni. Viene usato un artificio per poter impiegare **getopt()** con opzioni della forma **'-25'**, trattando quest'opzione come la lettera di opzione **'2'** con l'argomento **'5'**. Se si specificano due o più cifre (**Optarg** sembra essere numerico), **Optarg** è concatenato con la cifra che costituisce l'opzione e poi al risultato è addizionato zero, per trasformarlo in un numero. Se c'è solo una cifra nell'opzione, **Optarg** non è necessario. In tal caso, **Optind** dev'essere decrementata, in modo che **getopt()** la elabori quando viene nuovamente richiamato. Questo codice è sicuramente un po' intricato.

Se non sono specificate opzioni, per default si stampano sia le righe ripetute che quelle non ripetute. Il file di output, se specificato, è assegnato a **file\_output**. In precedenza, **file\_output** è inizializzato allo standard output, **/dev/stdout**:

```
# uniq.awk --- implementa uniq in awk
#
# Richiede le funzioni di libreria getopt() e join()

function sintassi()
```

```

{
    print("sintassi: uniq [-udc [-n]] [+n] [ in [ out ]]") > "/dev/stderr"
    exit 1
}

# -c    contatore di righe. prevale su -d e -u
# -d    solo righe ripetute
# -u    solo righe non ripetute
# -n    salta n campi
# +n    salta n caratteri, salta prima eventuali campi

BEGIN {
    contatore = 1
    file_output = "/dev/stdout"
    opts = "udc0:1:2:3:4:5:6:7:8:9:"
    while ((c = getopt(ARGC, ARGV, opts)) != -1) {
        if (c == "u")
            solo_non_ripetute++
        else if (c == "d")
            solo_ripetute++
        else if (c == "c")
            conta_record++
        else if (index("0123456789", c) != 0) {
            # getopt() richiede argomenti per le opzioni
            # questo consente di gestire cose come -5
            if (Optarg ~ /^[[:digit:]]+$/ )
                contatore_file = (c Optarg) + 0
            else {
                contatore_file = c + 0
                Optind--
            }
        } else
            sintassi()
    }

    if (ARGV[Optind] ~ /\^[[:digit:]]+$/ ) {
        conta_caratteri = substr(ARGV[Optind], 2) + 0
        Optind++
    }

    for (i = 1; i < Optind; i++)
        ARGV[i] = ""

    if (solo_ripetute == 0 && solo_non_ripetute == 0)
        solo_ripetute = solo_non_ripetute = 1

    if (ARGC - Optind == 2) {

```

```

        file_output = ARGV[ARGC - 1]
        ARGV[ARGC - 1] = ""
    }
}

```

La funzione seguente, `se_sono_uguali()`, confronta la riga corrente, `$0`, con la riga precedente, `ultima`. Gestisce il salto di campi e caratteri. Se non sono stati richiesti né contatori di campo né contatori di carattere, `se_sono_uguali()` restituisce uno o zero a seconda del risultato di un semplice confronto tra le stringhe `ultima` e `$0`.

In caso contrario, le cose si complicano. Se devono essere saltati dei campi, ogni riga viene suddivisa in un vettore, usando `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\], pagina 198](#)); i campi desiderati sono poi nuovamente uniti in un'unica riga usando `join()`. Le righe ricongiunte vengono immagazzinate in `campi_ultima` e `campi_corrente`. Se non ci sono campi da saltare, `campi_ultima` e `campi_corrente` sono impostati a `ultima` e `$0`, rispettivamente. Infine, se occorre saltare dei caratteri, si usa `substr()` per eliminare i primi `conta_caratteri` caratteri in `campi_ultima` e `campi_corrente`. Le due stringhe sono poi confrontate e `se_sono_uguali()` restituisce il risultato del confronto:

```

function se_sono_uguali(    n, m, campi_ultima, campi_corrente,\
vettore_ultima, vettore_corrente)
{
    if (contatore_file == 0 && conta_caratteri == 0)
        return (ultima == $0)

    if (contatore_file > 0) {
        n = split(ultima, vettore_ultima)
        m = split($0, vettore_corrente)
        campi_ultima = join(vettore_ultima, contatore_file+1, n)
        campi_corrente = join(vettore_corrente, contatore_file+1, m)
    } else {
        campi_ultima = ultima
        campi_corrente = $0
    }
    if (conta_caratteri) {
        campi_ultima = substr(campi_ultima, conta_caratteri + 1)
        campi_corrente = substr(campi_corrente, conta_caratteri + 1)
    }

    return (campi_ultima == campi_corrente)
}

```

Le due regole seguenti sono il corpo del programma. La prima è eseguita solo per la prima riga dei dati. Imposta `ultima` al record corrente `$0`, in modo che le righe di testo successive abbiano qualcosa con cui essere confrontate.

La seconda regola fa il lavoro. La variabile `uguale` vale uno o zero, a seconda del risultato del confronto effettuato in `se_sono_uguali()`. Se `uniq` sta contando le righe ripetute, e le righe sono uguali, viene incrementata la variabile `contatore`. Altrimenti, viene stampata la riga e azzerato `contatore`, perché le due righe non sono uguali.

Se `uniq` non sta contando, e se le righe sono uguali, `contatore` è incrementato. Non viene stampato niente, perché l'obiettivo è quello di rimuovere i duplicati. Altrimenti, se `uniq` sta contando le righe ripetute e viene trovata più di una riga, o se `uniq` sta contando le righe non ripetute e viene trovata solo una riga, questa riga viene stampata, e `contatore` è azzerato.

Infine, una logica simile è usata nella regola `END` per stampare l'ultima riga di dati in input:

```
NR == 1 {
    ultima = $0
    next
}

{
    uguale = se_sono_uguali()

    if (conta_record) {    # prevale su -d e -u
        if (uguale)
            contatore++
        else {
            printf("%4d %s\n", contatore, ultima) > file_output
            ultima = $0
            contatore = 1    # reset
        }
        next
    }

    if (uguale)
        contatore++
    else {
        if ((solo_ripetute && contatore > 1) ||
            (solo_non_ripetute && contatore == 1))
            print ultima > file_output
        ultima = $0
        contatore = 1
    }
}

END {
    if (conta_record)
        printf("%4d %s\n", contatore, ultima) > file_output
    else if ((solo_ripetute && contatore > 1) ||
             (solo_non_ripetute && contatore == 1))
        print ultima > file_output
    close(file_output)
}
```

### 11.2.7 Contare cose

Il programma di utilità `wc` (*word count*, contatore di parole) conta righe, parole, e caratteri in uno o più file in input. La sua sintassi è la seguente:

```
wc [-lwc] [file ...]
```

Se nessun file è specificato sulla riga di comando, `wc` legge il suo standard input. Se ci sono più file, stampa anche il contatore totale di tutti i file. Le opzioni e il loro significato sono i seguenti:

- l            Conta solo le righe.
- w            Conta solo le parole. Una “parola” è una sequenza contigua di caratteri non bianchi, separata da spazi e/o TAB. Fortunatamente, questo è il modo normale in cui `awk` separa i campi nei suoi record in input.
- c            Conta solo i caratteri.

Implementare `wc` in `awk` è particolarmente elegante, perché `awk` fa molto lavoro al posto nostro; divide le righe in parole (cioè, campi) e le conta, conta le righe (cioè, i record), e può facilmente dire quanto è lunga una riga.

Questo programma usa la funzione di libreria `getopt()` (si veda la [Sezione 10.4 \[Elaborare opzioni specificate sulla riga di comando\]](#), pagina 263) e le funzioni di passaggio da un file all'altro (si veda la [Sezione 10.3.1 \[Trovare i limiti dei file-dati\]](#), pagina 258).

Questa versione ha una differenza significativa rispetto alle versioni tradizionali di `wc`: stampa sempre i contatori rispettando l'ordine righe, parole e caratteri. Le versioni tradizionali rilevano l'ordine in cui sono specificate le opzioni `-l`, `-w` e `-c` sulla riga di comando, e stampano i contatori in quell'ordine.

La regola `BEGIN` si occupa degli argomenti. La variabile `stampa_totale` è vera se più di un file è presente sulla riga di comando:

```
# wc.awk --- conta righe, parole, caratteri

# Opzioni:
#   -l   conta solo righe
#   -w   conta solo parole
#   -c   conta solo caratteri
#
# Il default è di contare righe, parole, caratteri
#
# Richiede le funzioni di libreria getopt()
# e il programma di libreria che gestisce
# il passaggio da un file dati al successivo

BEGIN {
    # consente a getopt() di stampare un messaggio se si specificano
    # opzioni non valide. Noi le ignoriamo
    while ((c = getopt(ARGC, ARGV, "lwc")) != -1) {
        if (c == "l")
            conta_righe = 1
```

```

        else if (c == "w")
            conta_parole = 1
        else if (c == "c")
            conta_caratteri = 1
    }
    for (i = 1; i < Optind; i++)
        ARGV[i] = ""

    # se nessuna opzione è specificata, conta tutto
    if (! conta_righe && ! conta_parole && ! conta_caratteri)
        conta_righe = conta_parole = conta_caratteri = 1

    stampa_totale = (ARGC - i > 1)
}

```

La funzione `a_inizio_file()` è semplice; si limita ad azzerare i contatori di righe, parole e caratteri, e salva il valore corrente di nome-file in `nome_file`:

```

function a_inizio_file(file)
{
    righe = parole = caratteri = 0
    nome_file = FILENAME
}

```

La funzione `a_fine_file()` aggiunge i numeri del file corrente al totale di righe, parole, e caratteri. Poi stampa i numeri relativi al file appena letto. La funzione `a_inizio_file()` azzeri i numeri relativi al file-dati seguente:

```

function a_fine_file(file)
{
    totale_righe += righe
    totale_parole += parole
    totale_caratteri += caratteri
    if (conta_righe)
        printf "\t%d", righe
    if (conta_parole)
        printf "\t%d", parole
    if (conta_caratteri)
        printf "\t%d", caratteri
    printf "\t%s\n", nome_file
}

```

C'è una regola che viene eseguita per ogni riga. Aggiunge la lunghezza del record più uno, a `caratteri`.<sup>3</sup> Aggiungere uno alla lunghezza del record è necessario, perché il carattere di ritorno a capo, che separa i record (il valore di `RS`) non è parte del record stesso, e quindi non è incluso nella sua lunghezza. Poi, `righe` è incrementata per ogni riga letta, e `parole` è incrementato con il valore `NF`, che è il numero di “parole” su questa riga:

```

# per ogni riga...

```

---

<sup>3</sup> Poiché `gawk` gestisce le localizzazioni in cui un carattere può occupare più di un byte, questo codice conta i caratteri, non i byte.

```

{
    caratteri += length($0) + 1    # aggiunge un ritorno a capo
    righe++
    parole += NF
}

```

Infine, la regola END si limita a stampare i totali per tutti i file:

```

END {
    if (stampa_totale) {
        if (conta_righe)
            printf "\t%d", totale_righe
        if (conta_parole)
            printf "\t%d", totale_parole
        if (conta_caratteri)
            printf "\t%d", totale_caratteri
        print "\ttotale"
    }
}

```

## 11.3 Un paniere di programmi awk

Questa sezione è un “paniere” che contiene vari programmi. Si spera che siano interessanti e divertenti.

### 11.3.1 Trovare parole duplicate in un documento

Un errore comune quando si scrive un testo lungo è quello di ripetere accidentalmente delle parole. Tipicamente lo si può vedere in testi del tipo “questo questo programma fa quanto segue. . .” Quando il testo è pubblicato in rete, spesso le parole duplicate sono poste tra il termine di una riga e l’inizio di un’altra, il che rende difficile scoprirle.

Questo programma, `dupword.awk`, legge un file una riga alla volta e cerca le occorrenze adiacenti della stessa parola. Conserva anche l’ultima parola di ogni riga (nella variabile `precedente`) per confrontarla con la prima parola sulla riga successiva.

Le prime due istruzioni fanno sì che la riga sia tutta in minuscolo, in modo che, per esempio, “Il” e “il” risultino essere la stessa parola. L’istruzione successiva sostituisce i caratteri che sono non alfanumerici e diversi dagli spazi bianchi con degli spazi, in modo che neppure la punteggiatura influenzi i confronti. I caratteri sono rimpiazzati da spazi in modo che i controlli di formattazione non creino parole prive di senso (p.es., l’espressione Texinfo ‘@code{NF}’ diventa ‘codeNF’, se ci si limita a eliminare la punteggiatura). Il record è poi suddiviso di nuovo in campi, producendo così solo la lista delle parole presenti sulla riga, esclusi eventuali campi nulli.

Se, dopo aver rimosso tutta la punteggiatura, non rimane alcun campo, il record corrente è saltato. In caso contrario, il programma esegue il ciclo per ogni parola, confrontandola con quella che la precede:

```

# dupword.awk --- trova parole duplicate in un testo
{
    $0 = tolower($0)
    gsub(/[^\[:alnum:][:blank:]]/, " ");

```

```

$0 = $0          # divide di nuovo in campi
if (NF == 0)
    next
if ($1 == prec)
    printf("%s:%d: duplicato %s\n",
           nome_file, FNR, $1)
for (i = 2; i <= NF; i++)
    if ($i == $(i-1))
        printf("%s:%d: duplicato %s\n",
               nome_file, FNR, $i)
prec = $NF
}

```

### 11.3.2 Un programma di sveglia

*Nessuna cura contro l'insonnia è efficace quanto una sveglia che suona.*

—Arnold Robbins

*Il sonno è per sviluppatori web.*

—Erik Quantstrom

Il seguente programma è un semplice programma di “sveglia”. Si può specificare un’ora del giorno e un messaggio opzionale. All’ora specificata, il programma stampa il messaggio sullo standard output. Inoltre, si può specificare il numero di volte in cui il messaggio va ripetuto, e anche un intervallo di tempo (ritardo) tra ogni ripetizione.

Questo programma usa la funzione `getlocaltime()` dalla [Sezione 10.2.7 \[Gestione dell’ora del giorno\]](#), pagina 254.

Tutto il lavoro è svolto nella regola `BEGIN`. La prima parte è il controllo degli argomenti e l’impostazione dei valori di default: l’intervallo prima di ripetere, il contatore, e il messaggio da stampare. Se l’utente ha fornito un messaggio che non contiene il carattere ASCII BEL (noto come carattere “campanello”, `"\a"`), questo viene aggiunto al messaggio. (Su molti sistemi, stampare il carattere ASCII BEL genera un suono udibile. Quindi, quando la sveglia suona, il sistema richiama l’attenzione su di sé nel caso che l’utente non stia guardando il computer.) Per amor di varietà, questo programma usa un’istruzione `switch` (si veda la [Sezione 7.4.5 \[L’istruzione switch\]](#), pagina 156), ma l’elaborazione potrebbe anche essere fatta con una serie di istruzioni `if-else`. Ecco il programma:

```

# alarm.awk --- impostare una sveglia
#
# Richiede la funzione di libreria getlocaltime()
# sintassi: alarm a_che_ora [ "messaggio" [ contatore [ ritardo ] ] ]

BEGIN {
    # Controllo iniziale congruità argomenti
    sintassi1 = "sintassi: alarm a_che_ora ['messaggio' [contatore [ritardo]]]"
    sintassi2 = sprintf("\t(%s) formato ora: ::= hh:mm", ARGV[1])

    if (ARGC < 2) {
        print sintassi1 > "/dev/stderr"
        print sintassi2 > "/dev/stderr"
    }
}

```

```

        exit 1
    }
    switch (ARGC) {
    case 5:
        ritardo = ARGV[4] + 0
        # vai al caso seguente
    case 4:
        contatore = ARGV[3] + 0
        # vai al caso seguente
    case 3:
        messaggio = ARGV[2]
        break
    default:
        if (ARGV[1] !~ /[[:digit:]]?[[:digit:]]:[[:digit:]]{2}/) {
            print sintassi1 > "/dev/stderr"
            print sintassi2 > "/dev/stderr"
            exit 1
        }
        break
    }
}

# imposta i valori di default per quando arriva l'ora desiderata
if (ritardo == 0)
    ritardo = 180    # 3 minuti
if (contatore == 0)
    contatore = 5
if (messaggio == "")
    messaggio = sprintf("\aAdesso sono le %s!\a", ARGV[1])
else if (index(message, "\a") == 0)
    messaggio = "\a" messaggio "\a"

```

La successiva sezione di codice scompone l'ora specificata in ore e minuti, la converte (se è il caso) al formato 24-ore, e poi calcola il relativo numero di secondi dalla mezzanotte. Poi trasforma l'ora corrente in un contatore dei secondi dalla mezzanotte. La differenza tra i due è il tempo di attesa che deve passare prima di far scattare la sveglia:

```

# scomponi ora della sveglia
split(ARGV[1], ore_minuti, ":")
ora = ore_minuti[1] + 0    # trasforma in numero
minuto = ore_minuti[2] + 0 # trasforma in numero

# ottiene ora corrente divisa in campi
getlocaltime(adesso)

# se l'ora desiderata è in formato 12-ore ed è nel pomeriggio
# (p.es., impostare 'alarm 5:30' alle 9 del mattino
# vuol dire far suonare la sveglia alle 5:30 pomeridiane)
# aggiungere 12 all'ora richiesta

```

```

if (hour < 12 && adesso["hour"] > ora)
    ora += 12

# imposta l'ora in secondi dalla mezzanotte
sveglia = (ora * 60 * 60) + (minuto * 60)

# ottieni l'ora corrente in secondi dalla mezzanotte
corrente = (now["hour"] * 60 * 60) + \
            (now["minute"] * 60) + now["second"]

# quanto restare appisolati
sonno = sveglia - corrente
if (sonno <= 0) {
    print "alarm: l'ora è nel passato!" > "/dev/stderr"
    exit 1
}

```

Infine, il programma usa la funzione `system()` (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), [pagina 210](#)) per chiamare il programma di utilità `sleep`. Il programma di utilità `sleep` non fa altro che aspettare per il numero di secondi specificato. Se il codice di ritorno restituito è diverso da zero, il programma suppone che `sleep` sia stato interrotto ed esce. Se `sleep` è terminato con un codice di ritorno corretto, (zero), il programma stampa il messaggio in un ciclo, utilizzando ancora `sleep` per ritardare per il numero di secondi necessario:

```

# zzzzzzz..... esci se sleep è interrotto
if (system(sprintf("sleep %d", sonno)) != 0)
    exit 1

# è ora di avvisare!
command = sprintf("sleep %d", ritardo)
for (i = 1; i <= contatore; i++) {
    print messaggio
    # se il comando sleep è interrotto, esci
    if (system(command) != 0)
        break
}

exit 0
}

```

### 11.3.3 Rimpiazzare o eliminare caratteri

Il programma di utilità di sistema `tr` rimpiazza caratteri. Per esempio, è spesso usato per trasformare lettere maiuscole in lettere minuscole in vista di ulteriori elaborazioni:

```
generare dei dati | tr 'A-Z' 'a-z' | elaborare dei dati ...
```

`tr` richiede due liste di caratteri.<sup>4</sup> Quando si elabora l'input, il primo carattere della prima lista è rimpiazzato con il primo carattere della seconda lista, il secondo carattere della prima lista è rimpiazzato con il secondo carattere della seconda lista, e così via. Se ci sono più caratteri nella lista “da” che in quella “a”, l'ultimo carattere della lista “a” è usato per i restanti caratteri della lista “da”.

In un lontano passato, un utente propose di aggiungere una funzione di traslitterazione a `gawk`. Il programma seguente è stato scritto per dimostrare che la traslitterazione di caratteri poteva essere fatta con una funzione definita dall'utente. Questo programma non è così completo come il programma di utilità di sistema `tr`, ma svolge buona parte dello stesso lavoro.

Il programma `translate` è stato scritto molto prima che `gawk` fosse in grado di separare ciascun carattere di una stringa in elementi distinti di un vettore. Questo è il motivo per cui usa ripetutamente le funzioni predefinite `substr()`, `index()`, e `gsub()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Ci sono due funzioni. La prima, `traduci_stringa()`, richiede tre argomenti:

`da`            Una lista di caratteri da cui traslitterare  
`a`             Una lista di caratteri a cui traslitterare  
`stringa`      La stringa su cui effettuare la traslitterazione

I vettori associativi facilitano molto la parte di traslitterazione. `vettore_trad` contiene i caratteri “a”, indicizzato dai caratteri “da”. Poi un semplice ciclo scandisce `da`, un carattere alla volta. Per ogni carattere in `da`, se il carattere compare in `stringa` è rimpiazzato con il corrispondente carattere `a`.

La funzione `traducilo()` chiama `traduci_stringa()`, usando `$0` come stringa. Il programma principale imposta due variabili globali, `DA` e `A`, dalla riga di comando, e poi modifica `ARGV` in modo che `awk` legga dallo standard input.

Infine, la regola di elaborazione si limita a chiamare `traducilo()` per ogni record:

```
# translate.awk --- fa cose simili al comando tr
# Bug: non gestisce cose del tipo tr A-Z a-z; deve essere
# descritto carattere per carattere.
# Tuttavia, se 'a' è più corto di 'da',
# l'ultimo carattere in 'a' è usato per il resto di 'da'.

function traduci_stringa(da, a, stringa,      lf, lt, lstringa, vettore_trad,
                        i, c, risultato)
{
    lf = length(da)
    lt = length(a)
    lstringa = length(stringa)
    for (i = 1; i <= lt; i++)
        vettore_trad[substr(da, i, 1)] = substr(a, i, 1)
    if (lt < lf)
```

<sup>4</sup> Su alcuni sistemi più datati, incluso Solaris, la versione di sistema di `tr` può richiedere che le liste siano scritte come espressioni di intervallo, racchiuse in parentesi quadre (“[a-z]”) e tra apici, per evitare che la shell effettui espansioni di nome-file. Questo non è un miglioramento.

```

        for (; i <= lf; i++)
            vettore_trad[substr(da, i, 1)] = substr(a, lt, 1)
    for (i = 1; i <= lstringa; i++) {
        c = substr(stringa, i, 1)
        if (c in vettore_trad)
            c = vettore_trad[c]
        risultato = risultato c
    }
    return risultato
}

function traducilo(da, a)
{
    return $0 = traduci_stringa(da, a, $0)
}

# programma principale
BEGIN {
    if (ARGC < 3) {
        print "sintassi: translate da a" > "/dev/stderr"
        exit
    }
    DA = ARGV[1]
    A = ARGV[2]
    ARGC = 2
    ARGV[1] = "-"
}

{
    traducilo(DA, A)
    print
}

```

È possibile effettuare la traslitterazione di caratteri in una funzione a livello utente, ma non è detto che sia efficiente, e noi (sviluppatori di **gawk**) abbiamo iniziato a prendere in considerazione l'aggiunta di una funzione. Tuttavia, poco dopo aver scritto questo programma, abbiamo saputo che Brian Kernighan aveva aggiunto le funzioni **toupper()** e **tolower()** alla sua versione di **awk** (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)). Queste funzioni gestiscono la maggior parte dei casi in cui serva la traslitterazione di caratteri, e quindi abbiamo deciso di limitarci ad aggiungere le stesse funzioni a **gawk**, e di disinteressarci del resto.

Un miglioramento ovvio a questo programma sarebbe di impostare il vettore **vettore\_trad** solo una volta, in una regola **BEGIN**. Tuttavia, ciò presuppone che le liste “da” e “a” non cambino mai durante tutta l'esecuzione del programma.

Un altro miglioramento ovvio è di consentire l'uso di intervalli, come ‘a-z’, come consentito dal programma di utilità **tr**. Si può trarre ispirazione dal codice di **cut.awk** (si veda la [Sezione 11.2.1 \[Ritagliare campi e colonne\]](#), [pagina 282](#)).

### 11.3.4 Stampare etichette per lettere

Ecco un programma “del mondo-reale”<sup>5</sup>. Questo script legge elenchi di nomi e indirizzi, e genera etichette per lettera. Ogni pagina di etichette contiene 20 etichette, su due file da 10 etichette l’una. Gli indirizzi non possono contenere più di cinque righe di dati. Ogni indirizzo è separato dal successivo da una riga bianca.

L’idea di base è di leggere dati per 20 etichette. Ogni riga di ogni etichetta è immagazzinata nel vettore `riga`. L’unica regola si occupa di riempire il vettore `riga` e di stampare la pagina dopo che sono state lette 20 etichette.

La regola `BEGIN` si limita a impostare `RS` alla stringa vuota, in modo che `awk` divida un record dal successivo quando incontra una riga bianca. (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\], pagina 63](#)). Inoltre imposta `LIMITE_LINEE` a 100, perché 100 è il massimo numero di righe sulla pagina ( $20 \cdot 5 = 100$ ).

Il grosso del lavoro è svolto nella funzione `stampa_pagina()`. Le righe che compongono le etichette sono immagazzinate sequenzialmente nel vettore `riga`. Ma occorre stamparle in orizzontale: `riga[1]` a fianco di `riga[6]`, `riga[2]` a fianco di `riga[7]`, e così via. Questo si può fare utilizzando due cicli. Quello più esterno, controllato dalla variabile `i`, gestisce 10 righe di dati, ovvero la stampa di due etichette una a fianco dell’altra. Il ciclo più interno controllato dalla variabile `j`, gestisce le singole righe che compongono ognuno degli indirizzi. Poiché `j` varia da 0 a 4, `‘i+j’` è la riga `j`-esima dell’indirizzo di sinistra, e `‘i+j+5’` è quella stampata alla sua destra. L’output è simile a quello mostrato qui sotto:

```

riga 1          riga 6
riga 2          riga 7
riga 3          riga 8
riga 4          riga 9
riga 5          riga 10
...

```

La stringa di formato per `printf ‘%-41s’` allinea a sinistra i dati, e li stampa in un campo di lunghezza fissa.

Come nota finale, un’ulteriore riga bianca extra viene stampata alle righe 21 e 61, per mantenere entro i bordi l’output sulle etichette. Ciò dipende dalla particolare marca di etichette in uso quando il programma è stato scritto. Si noti anche che ci sono due righe bianche a inizio pagina e due righe bianche a fine pagina.

La regola `END` si occupa di stampare l’ultima pagina di etichette; è improbabile che il numero di indirizzi da stampare sia un multiplo esatto di 20:

```

# labels.awk --- stampare etichette per lettera

# Ogni etichetta è 5 righe di dati, qualcuna delle quali può essere bianca.
# I fogli con le etichetta hanno 2 righe bianche in cima alla pagina e altre 2
# a fine pagina.

BEGIN    { RS = "" ; LIMITE_LINEE = 100 }

function stampa_pagina(    i, j)

```

<sup>5</sup> “Del mondo-reale” è definito come “un programma effettivamente usato per realizzare qualcosa”.

```

{
    if (NUMEROorighe <= 0)
        return

    printf "\n\n"          # in cima

    for (i = 1; i <= NUMEROorighe; i += 10) {
        if (i == 21 || i == 61)
            print ""
        for (j = 0; j < 5; j++) {
            if (i + j > LIMITE_LINEE)
                break
            printf "    %-41s %s\n", riga[i+j], riga[i+j+5]
        }
        print ""
    }

    printf "\n\n"          # in fondo

    delete riga
}

# regola principale
{
    if (contatore >= 20) {
        stampa_pagina()
        contatore = 0
        NUMEROorighe = 0
    }
    n = split($0, a, "\n")
    for (i = 1; i <= n; i++)
        riga[++NUMEROorighe] = a[i]
    for (; i <= 5; i++)
        riga[++NUMEROorighe] = ""
    contatore++
}

END {
    stampa_pagina()
}

```

### 11.3.5 Generare statistiche sulla frequenza d'uso delle parole

Quando si lavora con una grande quantità di testo, può essere interessante sapere quanto spesso ricorrono le diverse parole. Per esempio, un autore può fare un uso eccessivo di certe parole, e in questo caso si potrebbero trovare sinonimi da sostituire a parole che appaiono

troppo spesso. Questa sottosezione spiega come scrivere un programma per contare le parole e presentare in un formato utile le informazioni relative alla loro frequenza.

A prima vista, un programma come questo sembrerebbe essere sufficiente:

```
# wordfreq-first-try.awk --- stampa lista frequenze utilizzo parole

{
    for (i = 1; i <= NF; i++)
        freq[$i]++
}

END {
    for (word in freq)
        printf "%s\t%d\n", word, freq[word]
}
```

Il programma si affida al meccanismo con cui **awk** divide i campi per default, per suddividere ogni riga in “parole” e usa un vettore associativo di nome **freq**, che ha per indici le singole parole, per contare il numero di volte che ogni parola viene usata. Nella regola **END**, stampa i contatori.

Questo programma ha parecchi problemi che lo rendono praticamente inutile su file di testo reali:

- Il linguaggio **awk** considera i caratteri maiuscoli e minuscoli come distinti (non equivalenti). Quindi, “barista” e “Barista” sono considerate parole differenti. Questo non è un comportamento auspicabile, perché le parole iniziano con la lettera maiuscola se sono a inizio frase in un testo normale, e un analizzatore di frequenze dovrebbe ignorare la distinzione maiuscolo/minuscolo.
- Le parole sono individuate usando la convenzione **awk** secondo cui i campi sono separati solo da spazi bianchi. Altri caratteri nell’input (tranne il ritorno a capo) non hanno alcun particolare significato per **awk**. Questo significa che i segni di interpunzione sono visti come parte di una parola.
- L’output non è scritto in alcun ordine utile. Si è probabilmente più interessati a sapere quali parole ricorrono più di frequente, o ad avere una tabella in ordine alfabetico che mostra quante volte ricorre ogni parola.

Il primo problema si può risolvere usando **tolower()** per rimuovere la distinzione maiuscolo/minuscolo. Il secondo problema si può risolvere usando **gsub()** per rimuovere i caratteri di interpunzione. Infine, per risolvere il terzo problema si può usare il programma di utilità **sort** per elaborare l’output dello script **awk**. Ecco la nuova versione del programma:

```
# wordfreq.awk --- stampa la lista con la frequenza delle parole

{
    $0 = tolower($0)      # toglia maiuscolo/minuscolo
    # toglia interpunzione
    gsub(/[^\[:alnum:]\_[:blank:]]/, "", $0)
    for (i = 1; i <= NF; i++)
        freq[$i]++
}
```

```

END {
    for (word in freq)
        printf "%s\t%d\n", word, freq[word]
}

```

La *regex* `/[^\[:alnum:]\_[:blank:]]/` si poteva scrivere come `/[[:punct:]]/`, ma in questo modo il caratteri trattino basso sarebbe stato rimosso, mentre si desidera conservarlo.

Supponendo di aver salvato questo programma in un file di nome `wordfreq.awk`, e che i dati siano in `file1`, il seguente comando con *pipeline*:

```
awk -f wordfreq.awk file1 | sort -k 2nr
```

produce una tabella delle parole che appaiono in `file1` in ordine decrescente di frequenza.

Il programma `awk` da solo gestisce adeguatamente i dati e produce una tabella delle frequenze che non è ordinata. L'output di `awk` è poi messo in ordine dal programma di utilità `sort` e stampato sullo schermo.

Le opzioni passate a `sort` richiedono un ordinamento che usi come chiave il secondo campo di ogni riga in input (saltando il primo campo), che le chiavi di ordinamento siano trattate come quantità numeriche (altrimenti '15' sarebbe stampato prima di '5'), e che l'ordinamento sia fatto in ordine decrescente (inverso).

Il comando `sort` potrebbe anche essere richiamato dall'interno del programma, cambiando l'azione da fare nella regola `END` a:

```

END {
    sort = "sort -k 2nr"
    for (word in freq)
        printf "%s\t%d\n", word, freq[word] | sort
    close(sort)
}

```

Questa maniera di ordinare dev'essere usata su sistemi che non hanno delle vere e proprie *pipe* a livello di riga di comando (o di procedura di comandi). Si veda la documentazione generale riguardo al sistema operativo per maggiori informazioni su come usare il programma `sort`.

### 11.3.6 Eliminare duplicati da un file non ordinato

Il programma `uniq` (si veda la [Sezione 11.2.6 \[Stampare righe di testo non duplicate\], pagina 296](#)) rimuove righe duplicate da dati *ordinati*.

Si supponga, tuttavia, di dover rimuovere righe duplicate da un file-dati, ma di voler conservare l'ordine in cui le righe sono state scritte. Un buon esempio di questo tipo potrebbe essere un file della cronologia dei comandi della shell. Il file della cronologia dei comandi mantiene copia di tutti i comandi che sono stati dati, e non è insolito ripetere un comando molte volte di fila. Occasionalmente si potrebbe voler compattare la cronologia togliendo le righe duplicate. Tuttavia sarebbe opportuno mantenere l'ordine originale dei comandi.

Questo semplice programma fa questo. Usa due vettori. Il vettore `dati` ha come indice il testo di ogni riga. Per ogni riga, `dati[$0]` è incrementato di uno. Se una particolare riga non è stata ancora vista, `dati[$0]` è zero. In tal caso, il testo della riga è immagazzinato in `righe[contatore]`. Ogni elemento del vettore `righe` è un comando unico, e gli indici

di `righe` indicano l'ordine in cui quelle righe sono state incontrate. La regola `END` stampa semplicemente le righe, in ordine:

```
# histsort.awk --- compatta un file della cronologia dei comandi della shell
# Grazie a Byron Rakitzis per l'idea generale

{
    if (dati[$0]++ == 0)
        righe[++contatore] = $0
}

END {
    for (i = 1; i <= contatore; i++)
        print righe[i]
}
```

Questo programma può essere un punto di partenza per generare altre informazioni utili. Per esempio, usando la seguente istruzione `print` nella regola `END` permette di sapere quante volte viene usato un certo comando:

```
print dati[righe[i]], righe[i]
```

Questo si può fare perché `dati[$0]` è incrementato ogni volta che una riga è stata trovata.

### 11.3.7 Estrarre programmi da un file sorgente Texinfo

Sia questo capitolo che il precedente ([Capitolo 10 \[Una libreria di funzioni awk\]](#), pagina 245) presentano un numero elevato di programmi `awk`. Se si vuole fare pratica con questi programmi, è fastidioso doverli digitare di nuovo manualmente. È per questo che abbiamo pensato a un programma in grado di estrarre parti di un file in input Texinfo e metterli in file separati.

Questo libro è scritto in [Texinfo](#), il programma di formattazione di documenti del progetto GNU. Un solo file sorgente Texinfo può essere usato per produrre sia la documentazione stampata, usando `TeX`, sia quella online. (Texinfo è esaurientemente documentato nel libro *Texinfo—The GNU Documentation Format*, disponibile alla Free Software Foundation, e anche [online](#).)

Per quel che ci riguarda, è sufficiente sapere tre cose riguardo ai file di input Texinfo:

- Il simbolo “chiocciola” (`@`) è speciale per Texinfo, proprio come la barra inversa (`\`) lo è per il linguaggio C o per `awk`. I simboli `@` sono rappresentati nel sorgente Texinfo come `@@`.
- I commenti iniziano con `@c` o con `@comment`. Il programma di estrazione file funziona usando dei commenti speciali che sono posti all'inizio di una riga.
- Righe contenenti comandi `@group` e `@end group` racchiudono testi di esempio che non dovrebbero andare a cavallo di due pagine. (Sfortunatamente, `TeX` non è sempre in grado di fare le cose in maniera esatta, e quindi va un po' aiutato).

Il programma seguente, `extract.awk`, legge un file sorgente Texinfo e fa due cose, basandosi sui commenti speciali. Dopo aver visto il commento `@c system ...`, esegue un comando, usando il testo del comando contenuto nella riga di controllo e passandolo alla funzione `system()` (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210).

Dopo aver trovato il commento '@c file *nome\_file*', ogni riga successiva è spedita al file *nome\_file*, fino a che si trova un commento '@c endfile'. Le regole in `extract.awk` sono soddisfatte sia quando incontrano '@c' che quando incontrano '@comment' e quindi la parte '@comment' è opzionale. Le righe che contengono '@group' e '@end group' sono semplicemente ignorate. `extract.awk` usa la funzione di libreria `join()` (si veda la [Sezione 10.2.6 \[Trasformare un vettore in una sola stringa\]](#), pagina 253).

I programmi di esempio nel sorgente Texinfo online di *GAWK: Programmare efficacemente in AWK* (`gawktexi.in`) sono stati tutti inseriti tra righe 'file' e righe 'endfile'. La distribuzione di `gawk` usa una copia di `extract.awk` per estrarre i programmi di esempio e per installarne molti in una particolare directory dove `gawk` li può trovare. Il file Texinfo ha un aspetto simile a questo:

```
...
Questo programma ha una regola @code{BEGIN}
che stampa un messaggio scherzoso:
```

```
@example
@c file esempi/messages.awk
BEGIN @ { print "Non v'allarmate!" @}
@c endfile
@end example
```

Stampa anche qualche avviso conclusivo:

```
@example
@c file esempi/messages.awk
END @ { print "Evitate sempre gli archeologi annoiati!" @}
@c endfile
@end example
...
```

Il programma `extract.awk` inizia con l'impostare `IGNORECASE` a uno, in modo che un miscuglio di lettere maiuscole e minuscole nelle direttive non faccia differenza.

La prima regola gestisce le chiamate a `system()`, controllando che sia stato fornito un comando (NF dev'essere almeno tre) e controllando anche che il comando termini con un codice di ritorno uguale a zero, che sta a significare che tutto è andato bene:

```
# extract.awk --- estrae file ed esegue programmi dal file Texinfo
```

```
BEGIN    { IGNORECASE = 1 }

/~@c(omment)?[ \t]+system/ {
    if (NF < 3) {
        e = ("extract: " FILENAME ":" FNR)
        e = (e " ": riga 'system' con formato errato")
        print e > "/dev/stderr"
        next
    }
    $1 = ""
```

```

$2 = ""
stat = system($0)
if (stat != 0) {
    e = ("extract: " FILENAME ":" FNR)
    e = (e ": attenzione: system ha restituito " stat)
    print e > "/dev/stderr"
}
}

```

La variabile `e` è stata usata per far sì che la regola sia agevolmente contenuta nella pagina.

La seconda regola gestisce il trasferimento di dati in un file. Verifica che nella direttiva sia stato fornito un nome-file. Se il nome del file non è quello del file corrente, il file corrente viene chiuso. Mantenere aperto il file corrente finché non si trova un nuovo nome file permette di usare la ridirezione `>` per stampare i contenuti nel file, semplificando la gestione dei file aperti.

Il ciclo `for` esegue il lavoro. Legge le righe usando `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83). Se si raggiunge una fine-file inattesa, viene chiamata la funzione `fine_file_inattesa()`. Se la riga è una riga “endfile”, il ciclo viene abbandonato. Se la riga inizia con `@group` o `@end group`, la riga viene ignorata, e si passa a quella seguente. Allo stesso modo, eventuali commenti all’interno degli esempi vengono ignorati.

Il grosso del lavoro è nelle poche righe che seguono. Se la riga non ha simboli `@`, il programma la può stampare così com’è. Altrimenti, ogni `@` a inizio parola dev’essere eliminato. Per rimuovere i simboli `@`, la riga viene divisa nei singoli elementi del vettore `a`, usando la funzione `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Il simbolo `@` è usato come carattere di separazione. Ogni elemento del vettore `a` che risulti vuoto indica due caratteri `@` contigui nella riga originale. Per ogni due elementi vuoti (`@@` nel file originale), va inserito un solo simbolo `@` nel file in output.

Una volta terminato di esaminare il vettore, viene chiamata la funzione `join()` specificando nella chiamata il valore di `SUBSEP` (si veda la [Sezione 8.5 \[Vettori multidimensionali\]](#), pagina 188), per riunire nuovamente i pezzi in una riga sola. La riga è poi stampata nel file di output:

```

/^@c(omment)?[ \t]+file/ {
    if (NF != 3) {
        e = ("extract: " FILENAME ":" FNR ": riga 'file' con formato errato")
        print e > "/dev/stderr"
        next
    }
    if ($3 != file_corrente) {
        if (file_corrente != "")
            close(file_corrente)
        file_corrente = $3
    }

    for (;;) {
        if ((getline riga) <= 0)

```

```

        fine_file_inattesa()
    if (riga ~ /^@c(omment)?[ \t]+endfile/)
        break
    else if (riga ~ /^@(end[ \t]+)?group/)
        continue
    else if (riga ~ /^@c(omment+)?[ \t]+/)
        continue
    if (index(riga, "@") == 0) {
        print riga > file_corrente
        continue
    }
    n = split(riga, a, "@")
    # if a[1] == "", vuol dire riga che inizia per @,
    # non salvare un @
    for (i = 2; i <= n; i++) {
        if (a[i] == "") { # era un @@
            a[i] = "@"
            if (a[i+1] == "")
                i++
        }
    }
    print join(a, 1, n, SUBSEP) > file_corrente
}
}

```

È importante notare l'uso della ridirezione '>'. L'output fatto usando '>' apre il file solo la prima volta; il file resta poi aperto, e ogni scrittura successiva è aggiunta in fondo al file. (si veda la [Sezione 5.6 \[Ridirigere l'output di print e printf\]](#), pagina 104). Ciò rende possibile mischiare testo del program e commenti esplicativi (come è stato fatto qui) nello stesso file sorgente, senza nessun problema. Il file viene chiuso solo quando viene trovato un nuovo nome di file-dati oppure alla fine del file in input.

Per finire, la funzione `fine_file_inattesa()` stampa un appropriato messaggio di errore ed esce. La regola `END` gestisce la pulizia finale, chiudendo il file aperto:

```

function fine_file_inattesa()
{
    printf("extract: %s:%d: fine-file inattesa, o errore\n",
          FILENAME, FNR) > "/dev/stderr"
    exit 1
}

END {
    if (file_corrente)
        close(file_corrente)
}

```

### 11.3.8 Un semplice editor di flusso

Il programma di utilità `sed` è un *editore di flusso*, ovvero un programma che legge un flusso di dati, lo modifica, e scrive il file così modificato. È spesso usato per fare modifiche generalizzate a un grosso file, o a un flusso di dati generato da una *pipeline* di comandi. Sebbene `sed` sia un programma piuttosto complesso di suo, l'uso che se ne fa solitamente è di effettuare delle sostituzioni globali attraverso una *pipeline*:

```
comando1 < dati.originali | sed 's/vecchio/nuovo/g' | comando2 > risultato
```

Qui, `'s/vecchio/nuovo/g'` chiede a `sed` di ricercare la *regexp* `'vecchio'` in ogni riga di input e di sostituirla dappertutto con il testo `'nuovo'` (cioè, in tutte le occorrenze di ciascuna riga). Questo è simile a quello che fa la funzione di `awk` `gsub()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).

Il programma seguente, `awksed.awk`, accetta almeno due argomenti dalla riga di comando: l'espressione da ricercare e il testo con cui rimpiazzarla. Ogni ulteriore argomento è considerato come un nome di file-dati da elaborare. Se non ne viene fornito alcuno, si usa lo standard input:

```
# awksed.awk --- fa s/pippo/pluto/g usando solo print
#   Ringraziamenti a Michael Brennan per l'idea

function sintassi()
{
    print "sintassi: awksed espressione rimpiazzo [file...]" > "/dev/stderr"
    exit 1
}

BEGIN {
    # valida argomenti
    if (ARGC < 3)
        sintassi()

    RS = ARGV[1]
    ORS = ARGV[2]

    # non usare argomenti come nomi di file
    ARGV[1] = ARGV[2] = ""
}

# guarda, mamma, senza mani!
{
    if (RT == "")
        printf "%s", $0
    else
        print
}
```

Il programma fa assegnamento sulla capacità di **gawk** di avere come **RS** una *regexp*, e anche sul fatto che **RT** viene impostato al testo che effettivamente delimita il record (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63).

L'idea è di usare **RS** come espressione da ricercare. **gawk** automaticamente imposta **\$0** al testo che compare tra due corrispondenze all'espressione di ricerca. Questo è appunto il testo che vogliamo conservare inalterato. Quindi, impostando **ORS** al testo che si vuole sostituire, una semplice istruzione **print** scrive il testo che si vuole mantenere, seguito dal testo che si vuole invece sostituire.

C'è un problema in questo schema, ossia cosa fare se l'ultimo record non termina con un testo che corrisponde a **RS**. Usando un'istruzione **print** incondizionatamente stampa il testo da sostituire, il che non è corretto. Tuttavia, se il file non termina con del testo che corrisponde a **RS**, **RT** è impostata alla stringa nulla. In tal caso, si può stampare **\$0** usando **printf** (si veda la [Sezione 5.5 \[Usare l'istruzione printf per stampe sofisticate\]](#), pagina 98).

La regola **BEGIN** gestisce la preparazione, controllando che ci sia il numero giusto di argomenti e chiamando **sintassi()** se c'è un problema. Poi imposta **RS** e **ORS** dagli argomenti della riga di comando e imposta **ARGV[1]** e **ARGV[2]** alla stringa nulla, per impedire che vengano considerati dei nomi-file (si veda la [Sezione 7.5.3 \[Usare ARGV e ARGV\]](#), pagina 172).

La funzione **sintassi()** stampa un messaggio di errore ed esce. Per finire, l'unica regola gestisce lo schema di stampa delineato più sopra, usando **print** o **printf** come richiesto, a seconda del valore di **RT**.

### 11.3.9 Una maniera facile per usare funzioni di libreria

Nella [Sezione 2.7 \[Come includere altri file nel proprio programma\]](#), pagina 45, abbiamo visto come **gawk** preveda la possibilità di includere file. Tuttavia, questa è un'estensione **gawk**. Questa sezione evidenzia l'utilità di rendere l'inclusione di file disponibile per **awk** standard, e mostra come farlo utilizzando una combinazione di programmazione di shell e di **awk**.

Usare funzioni di libreria in **awk** può presentare molti vantaggi. Incoraggia il riutilizzo di codice e la scrittura di funzioni di tipo generale. I programmi sono più snelli e quindi più comprensibili. Tuttavia, usare funzioni di libreria è facile solo in fase di scrittura di programmi **awk**; è invece complicato al momento di eseguirli, rendendo necessario specificare molte opzioni **-f**. Se **gawk** non è disponibile, non lo sono neppure la variabile d'ambiente **AWKPATH** e la possibilità di conservare funzioni **awk** in una directory di libreria (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33). Sarebbe bello poter scrivere programmi nel modo seguente:

```
# funzioni di libreria
@include getopt.awk
@include join.awk
...

# programma principale
BEGIN {
    while ((c = getopt(ARGC, ARGV, "a:b:cde")) != -1)
        ...
    ...
}
```

```
}

```

Il programma seguente, `igawk.sh`, fornisce questo servizio. Simula la ricerca da parte di `gawk` della variabile d'ambiente `AWKPATH` e permette anche delle inclusioni *nidificate* (cioè, un file che è stato incluso tramite `@include` può contenere ulteriori istruzioni `@include`). `igawk` tenta di includere ogni file una volta sola, in modo che delle inclusioni nidificate non contengano accidentalmente una funzione di libreria più di una volta.

`igawk` dovrebbe comportarsi esternamente proprio come `gawk`. Questo vuol dire che dovrebbe accettare sulla riga di comando tutti gli argomenti di `gawk`, compresa la capacità di specificare più file sorgenti tramite l'opzione `-f` e la capacità di mescolare istruzioni da riga di comando e file di sorgenti di libreria.

Il programma è scritto usando il linguaggio della Shell POSIX (`sh`).<sup>6</sup> Il funzionamento è il seguente:

1. Esegue un ciclo attraverso gli argomenti, salvando tutto ciò che non si presenta come codice sorgente `awk`, per quando il programma espanso sarà eseguito.
2. Per ogni argomento che rappresenta del codice `awk`, mette l'argomento in una variabile di shell che verrà espansa. Ci sono due casi:
  - a. Un testo letterale, fornito con l'opzione `-e` o `--source`. Questo testo viene aggiunto direttamente in fondo.
  - b. nomi-file sorgenti, forniti con l'opzione `-f`. Usiamo il truccetto di aggiungere `'@include nome_file'` in fondo ai contenuti della variabile di shell. Poiché il programma di inclusione dei file funziona allo stesso modo in cui funziona `gawk`, ne risulta che il file viene incluso nel programma al punto giusto.
3. Esegue un programma (naturalmente `awk`) sui contenuti della variabile di shell per espandere le istruzioni `@include`. Il programma espanso è messo in una seconda variabile di shell.
4. Esegue il programma espanso richiamando `gawk` e tutti gli altri argomenti originalmente forniti dall'utente sulla riga di comando (come p.es. dei nomi di file-dati).

Questo programma usa variabili di shell in quantità: per immagazzinare argomenti della riga di comando e il testo del programma `awk` che espanderà il programma dell'utente, per il programma originale dell'utente e per il programma espanso. Questo modo di procedere risolve potenziali problemi che potrebbero presentarsi se si usassero invece dei file temporanei, ma rende lo script un po' più complicato.

La parte iniziale del programma attiva il tracciamento della shell se il primo argomento è `'debug'`.

La parte successiva esegue un ciclo che esamina ogni argomento della riga di comando. Ci sono parecchi casi da esaminare:

--            Quest'opzione termina gli argomenti per `igawk`. Tutto quel che segue dovrebbe essere passato al programma `awk` dell'utente senza essere preso in considerazione.

---

<sup>6</sup> Una spiegazione dettagliata del linguaggio della `sh` non rientra negli intenti di questo libro. Qualche spiegazione sommaria viene fornita, ma se si desidera una comprensione più dettagliata, si dovrebbe consultare un buon libro sulla programmazione della shell.

**-W** Questo indica che l'opzione successiva è propria di **gawk**. Per facilitare l'elaborazione degli argomenti, l'opzione **-W** è aggiunta davanti agli argomenti rimanenti, e il ciclo continua. (Questo è un trucco di programmazione della **sh**. Non è il caso di preoccuparsene se non si ha familiarità con il comando **sh**.)

**-v, -F** Queste opzioni sono conservate e lasciate da gestire a **gawk**.

**-f, --file, --file=, -Wfile=**

Il nome-file è aggiunto alla variabile di shell **programma**, insieme a un'istruzione **@include**. Il programma di utilità **expr** è usato per eliminare la parte iniziale dell'argomento (p.es., **--file=**). (La sintassi tipica di **sh** richiederebbe di usare il comando **echo** e il programma di utilità **sed** per far questo. Sfortunatamente, alcune versioni di **echo** valutano le sequenze di protezione contenute nei loro argomenti, e questo potrebbe finire per alterare il testo del programma. L'uso di **expr** evita questo problema.)

**--source, --source=, -Wsource=**

Il testo sorgente è aggiunto in fondo a **programma**.

**--version, -Wversion**

**igawk** stampa il proprio numero di versione, esegue **'gawk --version'** per ottenere l'informazione relativa alla versione di **gawk**, ed esce.

Se nessuno degli argomenti **-f, --file, -Wfile, --source, o -Wsource** è stato fornito, il primo argomento che non è un'opzione dovrebbe essere il programma **awk**. Se non ci sono argomenti rimasti sulla riga di comando, **igawk** stampa un messaggio di errore ed esce. Altrimenti, il primo argomento è aggiunto in fondo a **programma**. In qualsiasi caso, dopo che gli argomenti sono stati elaborati, la variabile di shell **programma** contiene il testo completo del programma originale **awk**.

Il programma è il seguente:

```
#!/bin/sh
# igawk --- come gawk ma abilita l'uso di @include

if [ "$1" = debug ]
then
    set -x
    shift
fi

# Un ritorno a capo letterale,
# per formattare correttamente il testo del programma
n='
'

# Inizializza delle variabili alla stringa nulla
programma=
opts=

while [ $# -ne 0 ] # ciclo sugli argomenti
```

```

do
    case $1 in
        --)      shift
                break ;;

        -W)      shift
                # Il costrutto ${x?'messaggio qui'} stampa un
                # messaggio diagnostico se $x è la stringa nulla
                set -- -W"${@?'manca operando'}"
                continue ;;

        -[vF])   opts="$opts $1 '${2?'manca operando'}'"
                shift ;;

        -[vF]*)  opts="$opts '$1'" ;;

        -f)      programma="$programma$n@include ${2?'manca operando'}"
                shift ;;

        -f*)     f=$(expr "$1" : '-f\(.*\)\')
                programma="$programma$n@include $f" ;;

        -[W-]file=*)
                f=$(expr "$1" : '-.file=\(.*\)\')
                programma="$programma$n@include $f" ;;

        -[W-]file)
                programma="$programma$n@include ${2?'manca operando'}"
                shift ;;

        -[W-]source=*)
                t=$(expr "$1" : '-.source=\(.*\)\')
                programma="$programma$n$t" ;;

        -[W-]source)
                programma="$programma$n${2?'manca operando'}"
                shift ;;

        -[W-]version)
                echo igawk: version 3.0 1>&2
                gawk --version
                exit 0 ;;

        -[W-]*)  opts="$opts '$1'" ;;

        *)      break ;;
    esac

```

```

        shift
done

if [ -z "$programma" ]
then
    programma=${1?'manca programma'}
    shift
fi

# A questo punto, 'programma' contiene il programma.

```

Il programma `awk` che elabora le direttive `@include` è immagazzinato nella variabile di shell `progr_che_espande`. Ciò serve a mantenere leggibile lo script. Questo programma `awk` legge tutto il programma dell'utente, una riga per volta, usando `getline` (si veda la [Sezione 4.9 \[Richiedere input usando `getline`\], pagina 83](#)). I nomi-file in input e le istruzioni `@include` sono gestiti usando una pila. Man mano che viene trovata una `@include`, il valore corrente di nome-file è “spinto” sulla pila e il file menzionato nella direttiva `@include` diventa il nome-file corrente. Man mano che un file è finito, la pila viene “disfatta”, e il precedente file in input diventa nuovamente il file in input corrente. Il processo viene iniziato ponendo il file originale come primo file sulla pila.

La funzione `percorso()` trova qual è il percorso completo di un file. Simula il comportamento di `gawk` quando utilizza la variabile d'ambiente `AWKPATH` (si veda la [Sezione 2.5.1 \[Ricerca di programmi `awk` in una lista di directory.\], pagina 42](#)). Se un nome-file contiene una `/`, non viene effettuata la ricerca del percorso. Analogamente, se il nome-file è `-`, viene usato senza alcuna modifica. Altrimenti, il nome-file è concatenato col nome di ogni directory nella lista dei percorsi, e vien fatto un tentativo per aprire il nome-file così generato. Il solo modo di controllare se un file è leggibile da `awk` è di andare avanti e tentare di leggerlo con `getline`; questo è quel che `percorso()` fa.<sup>7</sup> Se il file può essere letto, viene chiuso e viene restituito il valore di nome-file:

```

progr_che_espande='

function percorso(file,    i, t, da_buttare)
{
    if (index(file, "/") != 0)
        return file

    if (file == "-")
        return file

    for (i = 1; i <= n_dir; i++) {
        t = (lista_percorsi[i] "/" file)

```

---

<sup>7</sup> In alcune versioni molto datate di `awk`, il test `'getline da_buttare < t'` può ripetersi in un ciclo infinito se il file esiste ma è vuoto.

```

        if ((getline da_buttare < t) > 0) {
            # found it
            close(t)
            return t
        }
    }
    return ""
}

```

Il programma principale è contenuto all'interno di una regola **BEGIN**. La prima cosa che fa è di impostare il vettore `lista_percorsi` usato dalla funzione `percorso()`. Dopo aver diviso la lista usando come delimitatore `:`, gli elementi nulli sono sostituiti da `."`, che rappresenta la directory corrente:

```

BEGIN {
    percorsi = ENVIRON["AWKPATH"]
    n_dir = split(percorsi, lista_percorsi, ":")
    for (i = 1; i <= n_dir; i++) {
        if (lista_percorsi[i] == "")
            lista_percorsi[i] = "."
    }
}

```

La pila è inizializzata con `ARGV[1]`, che sarà `"/dev/stdin"`. Il ciclo principale viene subito dopo. Le righe in input sono lette una dopo l'altra. Righe che non iniziano con `@include` sono stampate così come sono. Se la riga inizia con `@include`, il nome-file è in `$2`. La funzione `percorso()` è chiamata per generare il percorso completo. Se questo non riesce, il programma stampa un messaggio di errore e continua.

Subito dopo occorre controllare se il file sia già stato incluso. Il vettore `gia_fatto` è indicizzato dal nome completo di ogni nome-file incluso e tiene traccia per noi di questa informazione. Se un file viene visto più volte, viene stampato un messaggio di avvertimento. Altrimenti il nuovo nome-file è aggiunto alla pila e l'elaborazione continua.

Infine, quando `getline` giunge alla fine del file in input, il file viene chiuso, e la pila viene elaborata. Quando `indice_pila` è minore di zero, il programma è terminato:

```

indice_pila = 0
input[indice_pila] = ARGV[1] # ARGV[1] è il primo file

for (; indice_pila >= 0; indice_pila--) {
    while ((getline < input[indice_pila]) > 0) {
        if (tolower($1) != "@include") {
            print
            continue
        }
        cammino = percorso($2)
        if (cammino == "") {
            printf("igawk: %s:%d: non riesco a trovare %s\n",
                input[indice_pila], FNR, $2) > "/dev/stderr"
            continue
        }
        if (! (cammino in gia_fatto)) {

```

```

        gia_fatto[cammino] = input[indice_pila]
        input[++indice_pila] = cammino # aggiungilo alla pila
    } else
        print $2, "incluso in", input[indice_pila],
            "era già incluso in",
            gia_fatto[cammino] > "/dev/stderr"
    }
    close(input[indice_pila])
}
}' # l'apice chiude la variabile 'progr_che_espande'

programma_elaborato=$(gawk -- "$progr_che_espande" /dev/stdin << EOF
$programma
EOF
)

```

Il costrutto di shell *'comando << marcatore'* è chiamato *here document* (*documento sul posto*). Ogni riga presente nello script di shell fino al *marcatore* è passato in input a *comando*. La shell elabora i contenuti dell'*here document* sostituendo, dove serve, variabili e comandi (ed eventualmente altre cose, a seconda della shell in uso).

Il costrutto di shell *'\$(...)'* è chiamato *sostituzione di comando*. L'output del comando posto all'interno delle parentesi è sostituito nella riga di comando. Poiché il risultato è usato in un assegnamento di variabile, viene salvato come un'unica stringa di caratteri, anche se il risultato contiene degli spazi bianchi.

Il programma espanso è salvato nella variabile `programma_elaborato`. Il tutto avviene secondo le fasi seguenti:

1. Si esegue `gawk` con il programma che gestisce le `@include` (il valore della variabile di shell `progr_che_espande`) leggendo lo standard input.
2. Lo standard input contiene il programma dell'utente, nella variabile di shell `programma`. L'input è passato a `gawk` tramite un *here document*.
3. I risultati di questo processo sono salvati nella variabile di shell `programma_elaborato` usando la sostituzione di comando.

L'ultima fase è la chiamata a `gawk` con il programma espanso, insieme alle opzioni originali e agli argomenti della riga di comando che l'utente aveva fornito:

```
eval gawk $opts -- "$programma_elaborato" "$@"
```

Il comando `eval` è una struttura della shell che riesegue l'elaborazione dei parametri della riga di comando. Gli apici proteggono le parti restanti.

Questa versione di `igawk` è la quinta versione di questo programma. Ci sono quattro semplificazioni migliorative:

- L'uso di `@include` anche per i file specificati tramite l'opzione `-f` consente di semplificare di molto la preparazione del programma iniziale `awk`; tutta l'elaborazione delle istruzioni `@include` può essere svolta in una sola volta.
- Non tentare di salvare la riga letta tramite `getline` all'interno della funzione `percorso()` quando si controlla se il file è accessibile per il successivo uso nel programma principale semplifica notevolmente le cose.

- Usare un ciclo di `getline` nella regola `BEGIN` rende possibile fare tutto in un solo posto. Non è necessario programmare un ulteriore ciclo per elaborare le istruzioni `@include` nidificate.
- Invece di salvare il programma espanso in un file temporaneo, assegnarlo a una variabile di shell evita alcuni potenziali problemi di sicurezza. Ciò però ha lo svantaggio di basare lo script su funzionalità del linguaggio `sh`, il che rende più difficile la comprensione a chi non abbia familiarità con il comando `sh`.

Inoltre, questo programma dimostra come spesso valga la pena di utilizzare insieme la programmazione della `sh` e quella di `awk`. Solitamente, si può fare parecchio senza dover ricorrere alla programmazione di basso livello in C o C++, ed è spesso più facile fare certi tipi di manipolazioni di stringhe e argomenti usando la shell, piuttosto che `awk`.

Infine, `igawk` dimostra che non è sempre necessario aggiungere nuove funzionalità a un programma; queste possono spesso essere aggiunte in cima.<sup>8</sup>

### 11.3.10 Trovare anagrammi da una lista di parole

Un'interessante sfida per il programmatore è quella di cercare *anagrammi* in una lista di parole (come `/usr/share/dict/italian` presente in molti sistemi GNU/Linux). Una parola è un anagramma di un'altra se entrambe le parole contengono le stesse lettere (p.es., "branzino" e "bronzina").

La Colonna 2, Problema C, della seconda edizione del libro di Jon Bentley *Programming Pearls*, presenta un algoritmo elegante. L'idea è di assegnare a parole che sono anagrammi l'una dell'altra una firma comune, e poi di ordinare tutte le parole in base alla loro firma e di stamparle. Il Dr. Bentley fa notare che prendere tutte le lettere di ogni parola ed elencarle in ordine alfabetico produce queste firme comuni.

Il programma seguente usa vettori di vettori per riunire parole con la stessa firma, e l'ordinamento di vettori per stampare le parole trovate in ordine alfabetico:

```
# anagram.awk --- Un'implementazione dell'algoritmo per trovare anagrammi
#
# dalla seconda edizione
# del libro di Jon Bentley "Programming Pearls".
# Addison Wesley, 2000, ISBN 0-201-65788-0.
# Colonna 2, Problema C, sezione 2.8, pp 18-20.

/'s$/ { next }      # Salta i genitivi sassoni
```

Il programma inizia con un'intestazione, e poi una regola per saltare i genitivi sassoni eventualmente contenuti nel file che contiene la lista di parole. La regola successiva costruisce la struttura dei dati. Il primo indice del vettore è rappresentato dalla firma; il secondo è la parola stessa:

```
{
    chiave = da_parola_a_chiave($1) # costruisce la firma
    data[chiave][$1] = $1 # Immagazzina parola con questa firma
}
```

---

<sup>8</sup> `gawk` è in grado di elaborare istruzioni `@include` al suo stesso interno, per permettere l'uso di programmi `awk` come script Web CGI.

La funzione `da_parola_a_chiave()` crea la firma. Divide la parola in lettere singole, mette in ordine alfabetico le lettere, e poi le rimette ancora insieme:

```
# da_parola_a_chiave --- divide parole in lettere, ordina e riunisce

function da_parola_a_chiave(parola,      a, i, n, risultato)
{
    n = split(parola, a, "")
    asort(a)

    for (i = 1; i <= n; i++)
        risultato = risultato a[i]

    return risultato
}
```

Infine, la regola `END` percorre tutto il vettore e stampa le liste degli anagrammi. L'output è poi passato al comando di sistema `sort` perché altrimenti gli anagrammi sarebbero elencati in ordine arbitrario:

```
END {
    sort = "sort"
    for (chiave in data) {
        # ordina parole con la stessa chiave
        n_parole = asorti(data[chiave], parole)
        if (n_parole == 1)
            continue

        # e stampa. Problema minore: uno spazio extra a fine di ogni riga
        for (j = 1; j <= n_parole; j++)
            printf("%s ", parole[j]) | sort
        print "" | sort
    }
    close(sort)
}
```

Ecco una piccola parte dell'output quando il programma è eseguito:

```
$ gawk -f anagram.awk /usr/share/dict/italian | grep '^b'
...
baraste bastare serbata
barasti basarti
baratro tabarro
barattoli ribaltato tribolata
barbieri birberia
barche brache
barcollerei corbelleria
bare erba
bareremmo brameremo
barili librai
...
```

### 11.3.11 E ora per qualcosa di completamente differente

Il programma seguente è stato scritto da Davide Brini ed è pubblicato sul [suo sito web](#). Serve come sua firma nel gruppo Usenet `comp.lang.awk`. Questi sono i termini da lui stabiliti per il copyright:

Copyright © 2008 Davide Brini

Copying and distribution of the code published in this page, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

Ecco il programma:

```
awk 'BEGIN{0="~"~"~";o="=="=="=="";o+=+o;x=0"0";while(X++<=x+o+o)c=c"%c";
printf c,(x-0)*(x-0),x*(x-o)-o,x*(x-0)+x-0-o,+x*(x-0)-x+o,X*(o+o+0)+x-0,
X*(X-x)-o*o,(x+X)*o*o+o,x*(X-x)-0-0,x-0+(0+o+X+x)*(o+0),X*X-X*(x-0)-x+0,
0+X*(o*(o+0)+0),+x+0+X*o,x*(x-o),(o+X+x)*o*o-(x-0-0),0+(X-x)*(X+0),x-0}'
```

Viene lasciato al lettore il piacere di stabilire cosa fa il programma. (Se si è sull'orlo della disperazione nel tentativo di comprensione, si veda la spiegazione di Chris Johansen, che è contenuta nel file sorgente `Texinfo` di questo libro.)

## 11.4 Sommario

- I programmi illustrati in questo capitolo ripropongo la tesi secondo cui leggere programmi è una maniera eccellente per imparare a fare della buona programmazione.
- Usare `#!` per rendere i programmi `awk` direttamente eseguibili ne rende più semplice l'uso. In alternativa, si può invocare un programma usando `awk -f ...`.
- Reimplementare programmi POSIX standard in `awk` è un esercizio piacevole; il potere espressivo di `awk` consente di scrivere tali programmi usando relativamente poche righe di codice, nonostante i programmi risultanti siano funzionalmente completi e utilizzabili.
- Una delle debolezze della versione standard di `awk` riguarda il lavorare con singoli caratteri. La possibilità di usare `split()` con la stringa nulla come separatore può semplificare considerevolmente tale compito.
- Gli esempi proposti dimostrano l'utilità delle funzioni di libreria introdotte nel [Capitolo 10 \[Una libreria di funzioni awk\]](#), [pagina 245](#), per un numero (sia pur piccolo) di programmi reali.
- Oltre a reinventare la ruota POSIX, altri programmi risolvono una serie di problemi interessanti, come trovare delle parole duplicate in un testo, stampare etichette per lettere, e trovare anagrammi.

## 11.5 Esercizi

1. Riscrivere `cut.awk` (si veda la [Sezione 11.2.1 \[Ritagliare campi e colonne\]](#), [pagina 282](#)) usando `split()` con `" "` come separatore.
2. Nella [Sezione 11.2.2 \[Ricerca espressioni regolari nei file\]](#), [pagina 286](#), è detto che `'egrep -i'` potrebbe essere simulato in versioni di `awk` che non prevedono `IGNORECASE` usando `tolower()` sulla riga e nei criteri di ricerca. In una nota a piè di pagina è anche

detto che questa soluzione ha un problema: in output viene scritta la riga tradotta (a lettere minuscole), e non quella originale. Risolvere questo problema.

3. La versione POSIX di `id` accetta opzioni che controllano quali informazioni stampare. Modificare la versione `awk` (si veda la [Sezione 11.2.3 \[Stampare informazioni sull'utente\]](#), [pagina 290](#)) per accettare gli stessi argomenti e funzionare allo stesso modo.
4. Il programma `split.awk` (si veda la [Sezione 11.2.4 \[Suddividere in pezzi un file grosso\]](#), [pagina 292](#)) presuppone che le lettere siano contigue nella codifica dei caratteri, il che non è vero per sistemi che usano la codifica EBCDIC. Risolvere questo problema. (Suggerimento: Considerare un modo diverso di analizzare l'alfabeto, senza appoggiarsi sulle funzioni `ord()` e `chr()`.)
5. Nel programma `uniq.awk` (si veda la [Sezione 11.2.6 \[Stampare righe di testo non duplicate\]](#), [pagina 296](#), la logica per scegliere quali righe stampare rappresenta una *macchina a stati*, ossia “un dispositivo che può essere in uno di un insieme di stati stabili, a seconda dello stato in cui si trovava in precedenza, e del valore corrente dei suoi input.”<sup>9</sup> Brian Kernighan suggerisce che “un approccio alternativo alle macchine a stati è di leggere tutto l'input e metterlo in un vettore, e quindi usare gli indici. È quasi sempre più semplice da programmare, e per la maggior parte degli input in cui si può usare, altrettanto veloce in esecuzione.” Riscrivere la logica del programma seguendo questa indicazione.
6. Perché il programma `wc.awk` (si veda la [Sezione 11.2.7 \[Contare cose\]](#), [pagina 300](#)) non può limitarsi a usare il valore di `FNR` nella funzione `a_fine_file()`? Suggerimento: Esaminare il codice nella [Sezione 10.3.1 \[Trovare i limiti dei file-dati\]](#), [pagina 258](#).
7. La manipolazione di singoli caratteri nel programma `translate` (si veda la [Sezione 11.3.3 \[Rimpiazzare o eliminare caratteri\]](#), [pagina 305](#)) è farraginosa usando le funzioni standard `awk`. Poiché `gawk` può dividere stringhe in caratteri singoli usando come separatore `"`, come si potrebbe usare questa funzionalità per semplificare il programma?
8. Il programma `extract.awk` (si veda la [Sezione 11.3.7 \[Estrarre programmi da un file sorgente Texinfo\]](#), [pagina 312](#)) è stato scritto prima che `gawk` avesse a disposizione la funzione `gsub()`. Usarla per semplificare il codice.
9. Si confronti la velocità di esecuzione del programma `awksed.awk` (si veda la [Sezione 11.3.8 \[Un semplice editor di flusso\]](#), [pagina 316](#)) con il più diretto:

```
BEGIN {
    stringa = ARGV[1]
    rimpiazzo = ARGV[2]
    ARGV[1] = ARGV[2] = ""
}

{ gsub(stringa, rimpiazzo); print }
```

10. Quali sono vantaggi e svantaggi di `awksed.awk` rispetto al vero programma di utilità `sed`?
11. Nella [Sezione 11.3.9 \[Una maniera facile per usare funzioni di libreria\]](#), [pagina 317](#), si è detto che non tentando di salvare la riga letta con `getline` nella funzione `percorso()`,

<sup>9</sup> Questo è la definizione trovata usando `define: state machine` come chiave di ricerca in Google.

mentre si controlla l'accessibilità del file da usare nel programma principale, semplifica notevolmente le cose. Quale problema è peraltro generato così facendo?

12. Come ulteriore esempio dell'idea che non sempre è necessario aggiungere nuove funzionalità a un programma, si consideri l'idea di avere due file in una directory presente nel percorso di ricerca:

**default.awk**

Questo file contiene un insieme di funzioni di libreria di default, come `getopt()` e `assert()`.

**sito.awk**

Questo file contiene funzioni di libreria che sono specifiche di un sito o di un'installazione; cioè, funzioni sviluppate localmente. Mantenere due file separati consente a **default.awk** di essere modificato in seguito a nuove versioni di **gawk**, senza che l'amministratore di sistema debba ogni volta aggiornarlo aggiungendo le funzioni locali.

Un utente ha suggerito che **gawk** venga modificato per leggere automaticamente questi file alla partenza. Piuttosto, sarebbe molto semplice modificare **igawk** per farlo. Poiché **igawk** è capace di elaborare direttive `@include` nidificate, **default.awk** potrebbe contenere semplicemente la lista di direttive `@include` con le funzioni di libreria desiderate. Fare questa modifica.

13. Modificare **anagram.awk** (si veda la [Sezione 11.3.10 \[Trovare anagrammi da una lista di parole\]](#), [pagina 324](#)), per evitare di usare il programma di utilità esterno **sort**.

**Parte III:**

**Andare oltre awk con gawk**



## 12 Funzionalità avanzate di gawk

*Scrivete la documentazione supponendo che chiunque la leggerà sia uno psicotico violento, che conosce il vostro indirizzo di casa.*

—Steve English, citato da Peter Langston

Questo capitolo tratta delle funzionalità avanzate in **gawk**. È un po' come un "pacco sorpresa" di argomenti che non sono collegati tra di loro in altro modo. Per prima cosa, vediamo un'opzione da riga di comando che consente a **gawk** di riconoscere i numeri non decimali nei dati in input, e non soltanto nei programmi **awk**. Poi vengono illustrate delle funzionalità speciali di **gawk** per l'ordinamento di vettori. Quindi viene trattato dettagliatamente l'I/O bidirezionale, di cui si è fatto cenno in precedenti parti di questo libro, assieme ai fondamenti sulle reti TCP/IP. Infine, vediamo come **gawk** può tracciare il *profilo* di un programma **awk**, così che si possa ritoccarlo per migliorarne le prestazioni.

Altre funzionalità avanzate vengono trattate separatamente dedicando un capitolo per ciascuna di esse:

- Il [Capitolo 13 \[Internazionalizzazione con gawk\]](#), pagina 349, parla di come internazionalizzare i propri programmi **awk**, in modo che parlino più lingue nazionali.
- Il [Capitolo 14 \[Effettuare il debug dei programmi awk\]](#), pagina 361, descrive il debugger dalla riga di comando disponibile all'interno di **gawk** per individuare errori nei programmi **awk**.
- Il [Capitolo 15 \[Calcolo con precisione arbitraria con gawk\]](#), pagina 379, illustra come si può usare **gawk** per eseguire calcoli con precisione arbitraria.
- Il [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395, tratta della capacità di aggiungere dinamicamente nuove funzioni predefinite a **gawk**.

### 12.1 Consentire dati di input non decimali

Se si esegue **gawk** con l'opzione `--non-decimal-data`, si possono avere valori in base diversa da dieci nei dati di input:

```
$ echo 0123 123 0x123 |
> gawk --non-decimal-data '{ printf "%d, %d, %d\n", $1, $2, $3 }'
+ 83, 123, 291
```

Affinché questa funzionalità sia disponibile, i programmi devono essere scritti in modo che **gawk** tratti i dati come valori numerici:

```
$ echo 0123 123 0x123 | gawk '{ print $1, $2, $3 }'
+ 0123 123 0x123
```

L'istruzione **print** tratta le sue espressioni come se fossero stringhe. Sebbene i campi possano comportarsi come numeri, quando necessario, essi rimangono sempre stringhe, per cui **print** non cerca di elaborarli come se fossero numeri. Si deve aggiungere zero a un campo affinché venga considerato come un numero. Per esempio:

```
$ echo 0123 123 0x123 | gawk --non-decimal-data '
> { print $1, $2, $3
>   print $1 + 0, $2 + 0, $3 + 0 }'
+ 0123 123 0x123
+ 83 123 291
```

Poiché capita comunemente di avere dati di tipo decimale con degli zeri iniziali, e poiché l'uso di questa funzionalità può portare a risultati inattesi, il comportamento di default è quello lasciarla disabilitata. Se si vuole, la si deve richiedere esplicitamente.

**ATTENZIONE:** *L'uso di questa opzione non è consigliata.* Può provocare errori molto seri eseguendo vecchi programmi. Al suo posto è meglio usare la funzione `strtonum()` per convertire i dati (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Questo rende i programmi più facili da scrivere e più facili da leggere, e porta a risultati meno inattesi.

Quest'opzione potrebbe sparire dalle versioni future di `gawk`.

## 12.2 Controllare la visita di un vettore e il suo ordinamento

`gawk` permette di controllare l'ordine con cui un ciclo `'for (indice in vettore)'` attraversa un vettore.

Inoltre, due funzioni predefinite, `asort()` e `asorti()`, permettono di mettere in ordine i vettori sulla base, rispettivamente, dei valori e degli indici del vettore. Queste due funzioni danno anche il controllo sui criteri in base ai quali riordinare gli elementi del vettore.

### 12.2.1 Controllare visita vettori

Per default, l'ordine secondo il quale un ciclo `'for (indice in vettore)'` scorre un vettore non è definito; in genere si basa sull'implementazione interna dei vettori all'interno di `awk`.

Spesso, tuttavia, si vorrebbe poter eseguire un ciclo sugli elementi in un determinato ordine scelto dall'utente programmatore. Con `gawk` si può fare.

La [Sezione 8.1.6 \[Visita di vettori in ordine predefinito con `gawk`\]](#), pagina 182, parla di come si possono assegnare valori speciali predefiniti a `PROCINFO["sorted_in"]` per controllare l'ordine secondo il quale `gawk` attraversa un vettore durante un ciclo `for`.

Inoltre, il valore di `PROCINFO["sorted_in"]` può essere un nome di funzione.<sup>1</sup> Questo consente di scorrere un vettore sulla base di un qualsiasi criterio personalizzato. Gli elementi del vettore vengono ordinati in accordo col valore ritornato da questa funzione. La funzione che fa il confronto dovrebbe essere definita con almeno quattro argomenti:

```
function confronta(i1, v1, i2, v2)
{
    confronta gli elementi 1 e 2 in qualche modo
    return < 0; 0; o > 0
}
```

Qui, `i1` e `i2` sono gli indici, e `v1` e `v2` sono i corrispondenti valori dei due elementi che si stanno confrontando. `v1` oppure `v2`, o entrambi, possono essere vettori se il vettore che si sta visitando contiene sottovettori come valori. (Si veda la [Sezione 8.6 \[Vettori di vettori\]](#), pagina 190, per maggiori informazioni sui sottovettori.) I tre possibili valori di ritorno sono interpretati nel seguente modo:

```
confronta(i1, v1, i2, v2) < 0
```

L'indice `i1` viene prima dell'indice `i2` durante l'avanzamento del ciclo.

<sup>1</sup> Questo è il motivo per cui gli ordinamenti predefiniti iniziano con il carattere '@', che non può essere usato in un identificatore.

```
confronta(i1, v1, i2, v2) == 0
```

Gli indici i1 e i2 sono equivalenti, ma l'ordine tra loro non è definito.

```
confronta(i1, v1, i2, v2) > 0
```

L'indice i1 viene dopo l'indice i2 durante l'avanzamento del ciclo.

La prima funzione di confronto può essere usata per scorrere un vettore secondo l'ordine numerico degli indici:

```
function cfr_ind_num(i1, v1, i2, v2)
{
    # confronto di indici numerici, ordine crescente
    return (i1 - i2)
}
```

La seconda funzione scorre un vettore secondo l'ordine delle stringhe dei valori degli elementi piuttosto che secondo gli indici:

```
function cfr_val_str(i1, v1, i2, v2)
{
    # confronto di valori di stringa, ordine crescente
    v1 = v1 ""
    v2 = v2 ""
    if (v1 < v2)
        return -1
    return (v1 != v2)
}
```

La terza funzione di confronto restituisce dapprima tutti i numeri, e dopo questi le stringhe numeriche senza spazi iniziali o finali, durante l'avanzamento del ciclo:

```
function cfr_val_num_str(i1, v1, i2, v2, n1, n2)
{
    # confronto mettendo i numeri prima dei valori di stringa,
    # ordine crescente
    n1 = v1 + 0
    n2 = v2 + 0
    if (n1 == v1)
        return (n2 == v2) ? (n1 - n2) : -1
    else if (n2 == v2)
        return 1
    return (v1 < v2) ? -1 : (v1 != v2)
}
```

Qui vediamo un programma principale che mostra come gawk si comporta usando ciascuna delle funzioni precedenti:

```
BEGIN {
    data["uno"] = 10
    data["due"] = 20
    data[10] = "uno"
    data[100] = 100
    data[20] = "due"
```

```

f[1] = "cfr_ind_num"
f[2] = "cfr_val_str"
f[3] = "cfr_val_num_str"
for (i = 1; i <= 3; i++) {
    printf("Funzione di ordinamento: %s\n", f[i])
    PROCINFO["sorted_in"] = f[i]
    for (j in data)
        printf("\tdata[%s] = %s\n", j, data[j])
    print ""
}
}

```

I risultati dell'esecuzione del programma sono questi:

```

$ gawk -f compdemo.awk
+ Funzione di ordinamento: cfr_ind_num Ordinamento per indice numerico
+   data[uno] = 10
+   data[due] = 20      Entrambe le stringhe sono numericamente zero
+   data[10] = uno
+   data[20] = due
+   data[100] = 100
+
+ Funzione di ordinamento: cfr_val_str Ordinamento per valore degli
+                                       elementi come stringhe
+   data[uno] = 10
+   data[100] = 100     La stringa 100 è minore della stringa 20
+   data[due] = 20
+   data[20] = due
+   data[10] = uno
+
+ Funzione di ordinamento: cfr_val_num_str Ordinamento con tutti i
+                                           valori numerici prima di tutte le stringhe
+   data[uno] = 10
+   data[due] = 20
+   data[100] = 100
+   data[20] = due
+   data[10] = uno

```

Si provi a ordinare gli elementi di un file delle password del sistema GNU/Linux in base al nome d'accesso dell'utente. Il seguente programma ordina i record secondo una specifica posizione del campo e può essere usato per questo scopo:

```

# passwd-sort.awk --- semplice programma per ordinare in base alla
# posizione del campo
# la posizione del campo è specificata dalla variabile globale POS

function per_campo(i1, v1, i2, v2)
{
    # confronto per valore, come stringa, e in ordine crescente

```

```

    return v1[POS] < v2[POS] ? -1 : (v1[POS] != v2[POS])
}

{
    for (i = 1; i <= NF; i++)
        a[NR][i] = $i
}

END {
    PROCINFO["sorted_in"] = "per_campo"
    if (POS < 1 || POS > NF)
        POS = 1
    for (i in a) {
        for (j = 1; j <= NF; j++)
            printf("%s%c", a[i][j], j < NF ? ":" : "")
        print ""
    }
}

```

Il primo campo di ogni elemento del file delle password è il nome d'accesso dell'utente, e i campi sono separati tra loro da due punti. Ogni record definisce un sottovettore, con ogni campo come elemento nel sottovettore. L'esecuzione del programma produce il seguente output:

```

$ gawk -v POS=1 -F: -f sort.awk /etc/passwd
+ adm:x:3:4:adm:/var/adm:/sbin/nologin
+ apache:x:48:48:Apache:/var/www:/sbin/nologin
+ avahi:x:70:70:Avahi daemon:/:/sbin/nologin
...

```

Il confronto normalmente dovrebbe restituire sempre lo stesso valore quando vien dato come argomento un preciso paio di elementi del vettore. Se viene restituito un risultato non coerente, l'ordine è indefinito. Questo comportamento può essere sfruttato per introdurre un ordinamento casuale in dati apparentemente ordinati:

```

function ordina_a_caso(i1, v1, i2, v2)
{
    # ordine casuale (attenzione: potrebbe non finire mai!)
    return (2 - 4 * rand())
}

```

Come già accennato, l'ordine degli indici è arbitrario se due elementi risultano uguali. Normalmente questo non è un problema, ma lasciare che elementi di uguale valore compaiano in ordine arbitrario può essere un problema, specialmente quando si confrontano valori di elementi di un elenco. L'ordine parziale di elementi uguali può cambiare quando il vettore viene visitato di nuovo, se nel vettore vengono aggiunti o rimossi elementi. Un modo per superare l'ostacolo quando si confrontano elementi con valori uguali è quello di includere gli indici nelle regole di confronto. Si noti che questo potrebbe rendere meno efficiente l'attraversamento del ciclo, per cui si consiglia di farlo solo se necessario. Le seguenti funzioni di confronto impongono un ordine deterministico, e si basano sul fatto che gli indici (di stringa) di due elementi non sono mai uguali:

```

function per_numero(i1, v1, i2, v2)
{
    # confronto di valori numerici (e indici), ordine decrescente
    return (v1 != v2) ? (v2 - v1) : (i2 - i1)
}

function per_stringa(i1, v1, i2, v2)
{
    # confronto di valori di stringa (e indici), ordine decrescente
    v1 = v1 i1
    v2 = v2 i2
    return (v1 > v2) ? -1 : (v1 != v2)
}

```

Una funzione di confronto personalizzata spesso può semplificare l'attraversamento del ciclo ordinato, e il solo limite è il cielo, quando si va a progettare una funzione di questo tipo.

Quando i confronti tra stringhe son fatti durante un'operazione di ordinamento, per valori di elementi che, uno o entrambi, non sono numeri, o per indici di elementi gestiti come stringhe, il valore di `IGNORECASE` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), [pagina 162](#)) controlla se i confronti trattano corrispondenti lettere maiuscole e minuscole come equivalenti o come distinte.

Un'altra cosa da tenere a mente è che nel caso di sottovettori, i valori degli elementi possono essere a loro volta dei vettori; una funzione di confronto in produzione dovrebbe usare la funzione `isarray()` (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), [pagina 223](#)) per controllare ciò, e scegliere un ordinamento preciso per i sottovettori.

Tutti gli ordinamenti basati su `PROCINFO["sorted_in"]` sono disabilitati in modalità `POSIX`, perché il vettore `PROCINFO` in questo caso non è speciale.

Come nota a margine, si è visto che ordinare gli indici del vettore prima di scorrere il vettore porta a un incremento variabile dal 15% al 20% del tempo di esecuzione dei programmi `awk`. Per questo motivo l'attraversamento ordinato di vettori non è il default.

### 12.2.2 Ordinare valori e indici di un vettore con `gawk`

Nella maggior parte delle implementazioni di `awk`, ordinare un vettore richiede una funzione `sort()`. Questo può essere istruttivo per provare diversi algoritmi di ordinamento, ma normalmente non è questo lo scopo del programma. In `gawk` ci sono le funzioni predefinite `asort()` e `asorti()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), [pagina 198](#)) per i vettori ordinati. Per esempio:

```

riempire il vettore dati
n = asort(dati)
for (i = 1; i <= n; i++)
    fare qualcosa con dati[i]

```

Dopo la chiamata ad `asort()`, il vettore `dati` è indicizzato da 1 a `n`, il numero totale di elementi in `dati`. (Questo conteggio è il valore di ritorno di `asort()`). `dati[1] ≤ dati[2] ≤ dati[3]`, e così via. Il confronto di default è basato sul tipo di elementi (si

veda la [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), pagina 130). Tutti i valori numerici vengono prima dei valori di stringa, che a loro volta vengono prima di tutti i sottovettori.

Un effetto collaterale rilevante nel chiamare `asort()` è che *gli indici originali del vettore vengono persi irrimediabilmente*. Poiché questo non sempre è opportuno, `asort()` accetta un secondo argomento:

```
populate the array orig
n = asort(orig, dest)
for (i = 1; i <= n; i++)
    fai qualcosaa con dest[i]
```

In questo caso, `gawk` copia il vettore `orig` nel vettore `dest` e ordina `dest`, distruggendo i suoi indici. Tuttavia il vettore `orig` non viene modificato.

Spesso, ciò di cui si ha bisogno è di ordinare per i valori degli *indici* invece che per i valori degli elementi. Per far questo si usa la funzione `asorti()`. L'interfaccia e il comportamento sono identici a quelli di `asort()`, solo che per l'ordinamento vengono usati i valori degli indici, che diventano i valori del vettore risultato:

```
{ orig[$0] = una_funz($0) }

END {
    n = asorti(orig, dest)
    for (i = 1; i <= n; i++) {
        Lavora direttamente con gli indici ordinati:
        fa qualcosa con dest[i]
        ...
        Accede al vettore originale attraverso gli indici ordinati:
        fa qualcosa con orig[dest[i]]
    }
}
```

Fin qui, tutto bene. Ora inizia la parte interessante. Sia `asort()` che `asorti()` accettano un terzo argomento di stringa per controllare il confronto di elementi del vettore. Quando abbiamo introdotto `asort()` e `asorti()` nella [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198, abbiamo ignorato questo terzo argomento; comunque, è giunto il momento di descrivere come questo argomento influenza queste due funzioni.

Fondamentalmente, il terzo argomento specifica come dev'essere ordinato il vettore. Ci sono due possibilità. Come per `PROCINFO["sorted_in"]`, quest'argomento può essere uno degli argomenti predefiniti che `gawk` fornisce (si veda la [Sezione 8.1.6 \[Visita di vettori in ordine predefinito con gawk\]](#), pagina 182), o può essere il nome di una funzione definita dall'utente (si veda la [Sezione 12.2.1 \[Controllare visita vettori\]](#), pagina 332).

Nell'ultimo caso, *la funzione può confrontare gli elementi in qualunque modo si voglia*, prendendo in considerazione solo gli indici, solo i valori, o entrambi. Questo è estremamente potente.

Una volta che il vettore è ordinato, `asort()` prende i *valori* nel loro ordine finale e li usa per riempire il vettore risultato, mentre `asorti()` prende gli *indici* nel loro ordine finale e li usa per riempire il vettore risultato.

**NOTA:** Copiare indici ed elementi non è dispendioso in termini di memoria. Internamente, `gawk` mantiene un *conteggio dei riferimenti* ai dati. Per esempio, dopo che `asort()` copia il primo vettore nel secondo, in memoria c'è ancora una sola copia dei dati degli elementi del vettore originale, ed entrambi i vettori accedono all'unica copia di valori che esiste in memoria.

Poiché `IGNORECASE` influenza i confronti tra stringhe, il valore di `IGNORECASE` influisce anche sull'ordinamento sia con `asort()` che con `asorti()`. Si noti inoltre che l'ordinamento della localizzazione *non* entra in gioco; i confronti sono basati solamente sul valore dei caratteri.<sup>2</sup>

L'esempio seguente mostra l'uso di una funzione di confronto usata con `asort()`. La funzione di confronto, `confronta_in_minuscolo()`, trasforma gli elementi da confrontare in lettere minuscole, in modo da avere confronti che non dipendono da maiuscolo/minuscolo.

```
# confronta_in_minuscolo --- confronta stringhe ignorando maiuscolo/minuscolo

function confronta_in_minuscolo(i1, v1, i2, v2,    l, r)
{
    l = tolower(v1)
    r = tolower(v2)

    if (l < r)
        return -1
    else if (l == r)
        return 0
    else
        return 1
}
```

E questo programma può essere usato per provarla:

```
# programma di test

BEGIN {
    Letters = "abcdefghijklmnopqrstuvwxyz" \
              "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    split(Letters, data, "")

    asort(data, risultato, "confronta_in_minuscolo")

    j = length(risultato) # numero elementi del vettore "risultato"
    for (i = 1; i <= j; i++) {
        printf("%s", risultato[i])
        if (i % (j/2) == 0)
            # a metà, la stampa del vettore va a capo
            printf("\n")
    }
}
```

---

<sup>2</sup> Ciò è vero perché il confronto basato sulla localizzazione avviene solo quando si è in modalità POSIX-compatibile, e poiché `asort()` e `asorti()` sono estensioni di `gawk`, esse non sono disponibili in quel caso.

```

        else
            printf(" ")
    }
}

```

Se si esegue il programma, si ottiene:

```

$ gawk -f confronta_in_minuscolo.awk
├ A a B b c C D d e E F f g G H h i I J j k K l L M m
├ n N O o p P Q q r R S s t T u U V v w W X x y Y z Z

```

## 12.3 Comunicazioni bidirezionali con un altro processo

Spesso è utile poter inviare dati a un programma separato che li elabori e in seguito leggere il risultato. Questo può essere sempre fatto con file temporanei:

```

# Scrivere i dati per l'elaborazione
filetemp = ("mieidati." PROCINFO["pid"])
while (non dipendente dai dati)
    print dati | ("sottoprogramma > " filetemp)
close("sottoprogramma > " filetemp)

# Legge il risultato, rimuove filetemp quando ha finito
while ((getline nuovidati < filetemp) > 0)
    elabora nuovidati secondo le esigenze
close(filetemp)
system("rm " filetemp)

```

Questo funziona, ma non è elegante. Tra le altre cose, richiede che il programma venga eseguito in una directory che non può essere condivisa tra gli utenti; per esempio, `/tmp` non può esserlo, poiché potrebbe accadere che un altro utente stia usando un file temporaneo con lo stesso nome.<sup>3</sup>

Comunque, con **gawk**, è possibile aprire una *pipe bidirezionale* verso un altro processo. Il secondo processo è chiamato *coprocesso*, poiché viene eseguito in parallelo con **gawk**. La connessione bidirezionale viene creata usando l'operatore `'|&'` (preso in prestito dalla shell Korn, **ksh**):<sup>4</sup>

```

do {
    print dati |& "sottoprogramma"
    "sottoprogramma" |& getline risultato
} while (ci sono ancora dati da elaborare)
close("sottoprogramma")

```

La prima volta che viene eseguita un'operazione I/O usando l'operatore `'|&'`, **gawk** crea una *pipeline* bidirezionale verso un processo figlio che esegue l'altro programma. L'output creato con `print` o con `printf` viene scritto nello standard input del programma, e il contenuto dello standard output del programma può essere letto dal programma **gawk** usando

<sup>3</sup> Michael Brennan suggerisce l'uso di `rand()` per generare nomi-file unici. Questo è un punto valido; tuttavia, i file temporanei rimangono più difficili da usare delle *pipe* bidirezionali.

<sup>4</sup> Questo è molto diverso dallo stesso operatore nella C shell e in Bash.

**getline.** Come accade coi processi avviati con '|', il sottoprogramma può essere un qualsiasi programma, o una *pipeline* di programmi, che può essere avviato dalla shell.

Ci sono alcune avvertenze da tenere presenti:

- Per come funziona internamente **gawk**, lo standard error dei coprocessi va nello stesso posto dove va lo standard error del genitore **gawk**. Non è possibile leggere lo standard error del figlio separatamente.
- La permanenza in memoria (bufferizzazione) dell'I/O del sottoprocesso potrebbe essere un problema. **gawk** automaticamente scrive su disco tutto l'output spedito tramite la *pipe* al coprocesso. Tuttavia, se il coprocesso non scrive su disco il suo output, **gawk** potrebbe bloccarsi mentre esegue una **getline** per leggere il risultato del coprocesso. Questo può portare a una situazione conosciuta come *stallo* (*deadlock*, abbraccio mortale), in cui ciascun processo rimane in attesa che l'altro processo faccia qualcosa.

È possibile chiudere una *pipe* bidirezionale con un coprocesso solo in una direzione, fornendo un secondo argomento, "to" o "from", alla funzione **close()** (si veda la [Sezione 5.9 \[Chiudere ridirezioni in input e in output\], pagina 109](#)). Queste stringhe dicono a **gawk** di chiudere la *pipe*, rispettivamente nella direzione che invia i dati al coprocesso e nella direzione che legge da esso.

Questo è particolarmente necessario per usare il programma di utilità di sistema **sort** come parte di un coprocesso; **sort** deve leggere *tutti* i dati di input prima di poter produrre un qualsiasi output. Il programma **sort** non riceve un'indicazione di fine-file (end-of-file) finché **gawk** non chiude l'estremità in scrittura della *pipe*.

Una volta terminata la scrittura dei dati sul programma **sort**, si può chiudere il lato "to" della *pipe*, e quindi iniziare a leggere i dati ordinati via **getline**. Per esempio:

```
BEGIN {
    comando = "LC_ALL=C sort"
    n = split("abcdefghijklmnopqrstuvwxyz", a, "")

    for (i = n; i > 0; i--)
        print a[i] |& comando
    close(comando, "to")

    while ((comando |& getline line) > 0)
        print "ricevuto", line
    close(comando)
}
```

Questo programma scrive le lettere dell'alfabeto in ordine inverso, uno per riga, attraverso la *pipe* bidirezionale verso **sort**. Poi chiude la direzione di scrittura della *pipe*, in modo che **sort** riceva un'indicazione di fine-file. Questo fa in modo che **sort** ordini i dati e scriva i dati ordinati nel programma **gawk**. Una volta che tutti i dati sono stati letti, **gawk** termina il coprocesso ed esce.

Come nota a margine, l'assegnamento 'LC\_ALL=C' nel comando **sort** assicura che **sort** usi l'ordinamento tradizionale di Unix (ASCII). Ciò non è strettamente necessario in questo caso, ma è bene sapere come farlo.

Occorre prestare attenzione quando si chiude il lato "from" di una *pipe* bidirezionale; in tal caso **gawk** attende che il processo-figlio termini, il che può causare lo stallo del programma

**awk** in esecuzione. (Per questo motivo, questa particolare funzionalità è molto meno usata, in pratica, di quella che consente la possibilità di chiudere il lato "to" della *pipe*.)

**ATTENZIONE:** Normalmente, è un errore fatale (che fa terminare il programma **awk**) scrivere verso il lato "to" di una *pipe* bidirezionale che è stata chiusa, e lo stesso vale se si legge dal lato "from" di una *pipe* bidirezionale che sia stata chiusa.

È possibile impostare `PROCINFO["comando", "NONFATAL"]` per far sì che tali operazioni non provochino la fine del programma **awk**. Se lo si fa, è necessario controllare il valore di `ERRNO` dopo ogni istruzione `print`, `printf`, o `getline`. Si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112, per ulteriori informazioni.

Per le comunicazioni bidirezionali si possono anche usare delle pseudo *tty* (*pty*) al posto delle *pipe*, se il sistema in uso le prevede. Questo vien fatto, a seconda del comando da usare, impostando un elemento speciale nel vettore `PROCINFO` (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui awk fornisce informazioni\]](#), pagina 165), in questo modo:

```
comando = "sort -nr"           # comando, salvato in una variabile
PROCINFO[comando, "pty"] = 1  # aggiorna PROCINFO
print ... |& comando           # avvia la pipe bidirezionale
...
```

Se il sistema in uso non ha le *pty*, o se tutte le *pty* del sistema sono in uso, **gawk** automaticamente torna a usare le *pipe* regolari.

Usare le *pty* in genere evita i problemi di stallo del buffer descritti precedentemente, in cambio di un piccolo calo di prestazioni. Ciò dipende dal fatto che la gestione delle *tty* è fatta una riga per volta. Su sistemi che hanno il comando `stdbuf` (parte del pacchetto **GNU Coreutils**), si può usare tale programma, invece delle *pty*.

Si noti anche che le *pty* non sono completamente trasparenti. Alcuni codici di controllo binari, come `Ctrl-d` per indicare la condizione di file-file, sono interpretati dal gestore di *tty* e non sono passati all'applicazione.

**ATTENZIONE:** In ultima analisi, i coprocessi danno adito alla possibilità di uno *stallo* (deadlock) tra **gawk** e il programma in esecuzione nel coprocesso. Ciò può succedere se si inviano "troppi" dati al coprocesso, prima di leggere dati inviati dallo stesso; entrambi i processi sono bloccati sulla scrittura dei dati, e nessuno dei due è disponibile a leggere quelli che sono già stati scritti dall'altro. Non c'è modo di evitare completamente una tale situazione; occorre una programmazione attenta, insieme alla conoscenza del comportamento del coprocesso.

## 12.4 Usare gawk per la programmazione di rete

EMRED:

*A host is a host from coast to coast,  
and nobody talks to a host that's close,  
unless the host that isn't close  
is busy, hung, or dead.*

EMRED:

*Un computer è un computer lontano o vicino,*

*e nessuno parla con un computer vicino,  
a meno che il computer lontano  
sia occupato, fuori linea, o spento.*

—Mike O'Brien (noto anche come Mr. Protocol)

Oltre a poter aprire una *pipeline* bidirezionale verso un coprocesso sullo stesso sistema (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339), è possibile attivare una connessione bidirezionale verso un altro processo o verso un altro sistema attraverso una connessione di rete IP.

Si può pensare a questo semplicemente come a una *pipeline* bidirezionale *molto lunga* verso un coprocesso. Il modo in cui **gawk** stabilisce quando usare una rete TCP/IP è mediante il riconoscimento di speciali nomi-file che iniziano con `‘/inet/’`, con `‘/inet4/’` o con `‘/inet6/’`.

La sintassi completa del nome-file speciale è `/tipo-rete/protocollo/porta-locale/host-remoto/porta-remota`. I componenti sono:

*tipo-rete* Specifica il tipo di connessione a internet da stabilire. Si usa `‘/inet4/’` per usare solo IPv4, e `‘/inet6/’` per usare solo IPv6. Il semplice `‘/inet/’` (che usualmente è l'unica opzione) usa il tipo di default del sistema, quasi certamente IPv4.

*protocollo* Il protocollo da usare sull'IP. Questo dev'essere o `‘tcp’` o `‘udp’`, per una connessione IP rispettivamente TCP o UDP. TCP dovrebbe venir usato per la maggior parte delle applicazioni.

*porta-locale*

Il numero di porta TCP o UDP da usare. Si usa un numero di porta di valore `‘0’` quando si vuole che sia il sistema a scegliere una porta. Questo è quel che si dovrebbe fare quando si scrive un'applicazione cliente TCP o UDP. Si può usare anche un nome di un servizio noto, come `‘smtp’` o `‘http’`, nel qual caso **gawk** tenta di determinare il numero di porta predefinito usando la funzione C `getaddrinfo()`.

*host-remoto*

L'indirizzo IP o il nome di dominio completamente qualificato dell'host internet al quale ci si vuol connettere.

*porta-remota*

Il numero di porta TCP o UDP da usare sul particolare *host remoto*. Anche in questo caso, si usi `‘0’` se non ci sono preferenze, o alternativamente, un nome di servizio comunemente noto.

**NOTA:** Un insuccesso nell'apertura di un socket bidirezionale darà luogo alla segnalazione di un errore non fatale al codice chiamante. Il valore di `ERRNO` indica l'errore (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\]](#), pagina 165).

Si consideri il seguente esempio molto semplice:

```
BEGIN {
  Servizio = "/inet/tcp/0/localhost/daytime"
  Servizio |& getline
  print $0
```

```

        close(Servizio)
    }

```

Questo programma legge la data e l'ora corrente dal server `daytime` TCP del sistema locale. Stampa poi il risultato e chiude la connessione.

Poiché questo tema è molto ampio, l'uso di `gawk` per la programmazione TCP/IP viene documentato separatamente. Si veda *TCP/IP Internetworking with gawk*, che fa parte della distribuzione `gawk`, per una introduzione e trattazione molto più completa e con molti esempi.

**NOTA:** `gawk` può aprire solo socket diretti. Al momento non c'è alcun modo per accedere ai servizi disponibili su Secure Socket Layer (SSL); questo comprende qualsiasi servizio web il cui URL inizia con `'https://'`.

## 12.5 Profilare i propri programmi awk

È possibile tener traccia dell'esecuzione dei propri programmi `awk`. Ciò si può fare passando l'opzione `--profile` a `gawk`. Al termine dell'esecuzione, `gawk` crea un profilo del programma in un file chiamato `awkprof.out`. A causa dell'attività di profilazione l'esecuzione del programma è più lenta fino al 45% rispetto al normale.

Come mostrato nel seguente esempio, l'opzione `--profile` può essere usata per cambiare il nome del file su cui `gawk` scriverà il profilo:

```
gawk --profile=mioprof.prof -f mioprof.awk dati1 dati2
```

Nell'esempio precedente, `gawk` mette il profilo in `mioprof.prof` anziché in `awkprof.out`.

Vediamo ora una sessione d'esempio che mostra un semplice programma `awk`, i suoi dati in input, e il risultato dell'esecuzione di `gawk` con l'opzione `--profile`. Innanzitutto, il programma `awk`:

```

BEGIN { print "Prima regola BEGIN" }

END { print "Prima regola END" }

/kippo/ {
    print "trovato /kippo/, perbacco"
    for (i = 1; i <= 3; i++)
        sing()
}

{
    if (/kippo/)
        print "l'if è vero"
    else
        print "l'else è vero"
}

BEGIN { print "Seconda regola BEGIN" }

END { print "Seconda regola END" }

```

```
function sing(    ignora)
{
    print "Devo essere io!"
}

```

Questi sono i dati in input:

```

pippo
pluto
paperino
pippo
cianfrusaglie

```

E questo è il file `awkprof.out` che è il risultato dell'esecuzione del profilatore di `gawk` su questo programma e sui dati (quest'esempio dimostra anche che i programmatori di `awk` a volte si alzano molto presto al mattino per lavorare):

```

# profilo gawk, creato Mon Sep 29 05:16:21 2014

# BEGIN regola(e)

BEGIN {
1  print "Prima regola BEGIN"
}

BEGIN {
1  print "Seconda regola BEGIN"
}

# Regola(e)

5  /pippo/ { # 2
2  print "trovato /pippo/, perbacco"
6  for (i = 1; i <= 3; i++) {
6  sing()
    }
}

5  {
5  if (/pippo/) { # 2
2  print "l'if è vero"
3  } else {
3  print "l'else è vero"
    }
}

# END regola(e)

END {

```

```

1  print "Prima regola END"
    }

    END {
1  print "Seconda regola END"
    }

    # Funzioni, in ordine alfabetico

6  function sing(ignora)
    {
6  print "Devo essere io!"
    }

```

Quest'esempio illustra molte caratteristiche fondamentali dell'output della profilazione. Queste sono:

- Il programma viene stampato nell'ordine: regole **BEGIN**, regole **BEGINFILE**, regole criterio di ricerca-azione, regole **ENDFILE**, regole **END**, e funzioni, elencate in ordine alfabetico. Le regole **BEGIN** ed **END** multiple conservano le loro distinte identità, così come le regole **BEGINFILE** ed **ENDFILE** multiple.
- Le regole criterio di ricerca-azione hanno due conteggi. Il primo conteggio, a sinistra della regola, mostra quante volte il criterio di ricerca della regola è stato *testato*. Il secondo conteggio, alla destra della parentesi graffa aperta, all'interno di un commento, mostra quante volte l'azione della regola è stata *eseguita*. La differenza tra i due indica quante volte il criterio di ricerca della regola è stato valutato come falso.
- Analogamente, il conteggio per un'istruzione **if-else** mostra quante volte la condizione è stata testata. Alla destra della parentesi graffa sinistra aperta per il corpo di **if** c'è un conteggio che mostra quante volte la condizione è stata trovata vera. Il conteggio per **else** indica quante volte la verifica non ha avuto successo.
- Il conteggio per un ciclo (come **for** o **while**) mostra quante volte il test del ciclo è stato eseguito. (Per questo motivo, non si può solamente guardare il conteggio sulla prima istruzione in una regola per determinare quante volte la regola è stata eseguita. Se la prima istruzione è un ciclo, il conteggio è ingannevole.)
- Per le funzioni definite dall'utente, il conteggio vicino alla parola chiave **function** indica quante volte la funzione è stata chiamata. I conteggi vicino alle istruzioni nel corpo mostrano quante volte quelle istruzioni sono state eseguite.
- L'impaginazione usa lo stile "K&R" con le tabulazioni. Le parentesi graffe sono usate dappertutto, anche dove il corpo di un **if**, di un **else** o di un ciclo è formato da un'unica istruzione.
- Le parentesi vengono usate solo dov'è necessario, come si rileva dalla struttura del programma e dalle regole di precedenza. Per esempio, `'(3 + 5) * 4'` significa sommare tre e cinque, quindi moltiplicare il totale per quattro. Di contro, `'3 + 5 * 4'` non ha parentesi, e significa `'3 + (5 * 4)'`.
- Le parentesi vengono usate attorno agli argomenti di **print** e **printf** solo quando

l'istruzione `print` o `printf` è seguita da una ridirezione. Similarmente, se l'oggetto di una ridirezione non è uno scalare, viene messo tra parentesi.

- **gawk** mette dei commenti iniziali davanti alle regole **BEGIN** ed **END**, alle regole **BEGINFILE** ed **ENDFILE**, alle regole **criterio\_di\_ricerca-azione** e alle funzioni.

La versione profilata del proprio programma potrebbe non apparire esattamente come quella scritta durante la stesura del programma. Questo perché **gawk** crea la versione profilata facendo una “stampa elegante” della sua rappresentazione interna del programma. Un vantaggio di ciò è che **gawk** può produrre una rappresentazione standard. Inoltre, cose come:

```
/pippo/
```

appaiono come:

```
/pippo/ {
    print $0
}
```

che è corretto, ma probabilmente inatteso.

Oltre a creare profili una volta completato il programma, **gawk** può generare un profilo mentre è in esecuzione. Questo è utile se il proprio programma **awk** entra in un ciclo infinito e si vuol vedere cosa è stato eseguito. Per usare questa funzionalità, bisogna eseguire **gawk** con l'opzione `--profile` in background:

```
$ gawk --profile -f mioprogram &
[1] 13992
```

La shell stampa un numero di job e il numero di ID del relativo processo; in questo caso, 13992. Si usi il comando `kill` per inviare il segnale **USR1** a **gawk**:

```
$ kill -USR1 13992
```

Come al solito, la versione profilata del programma è scritta nel file `awkprof.out`, o in un file differente se ne viene specificato uno con l'opzione `--profile`.

Assieme al profilo regolare, come mostrato in precedenza, il file del profilo include una traccia di ogni funzione attiva:

```
# 'Stack' (Pila) Chiamate Funzione:

# 3. paperino
# 2. pluto
# 1. pippo
# -- main --
```

Si può inviare a **gawk** il segnale **USR1** quante volte si vuole. Ogni volta, il profilo e la traccia della chiamata alla funzione vengono aggiunte in fondo al file di profilo creato.

Se si usa il segnale **HUP** invece del segnale **USR1**, **gawk** genera il profilo e la traccia della chiamata alla funzione ed esce.

Quando **gawk** viene eseguito sui sistemi MS-Windows, usa i segnali **INT** e **QUIT** per generare il profilo, e nel caso del segnale **INT**, **gawk** esce. Questo perché questi sistemi non prevedono il comando `kill`, per cui gli unici segnali che si possono trasmettere a un programma sono quelli generati dalla tastiera. Il segnale **INT** è generato dalle combinazioni

di tasti *Ctrl-c* o *Ctrl-BREAK*, mentre il segnale *QUIT* è generato dalla combinazione di tasti *Ctrl-\*.

Infine, *gawk* accetta anche un'altra opzione, *--pretty-print*. Quando viene chiamato in questo modo, *gawk* fa una “stampa elegante” del programma nel file *awkprof.out*, senza conteggi sull'esecuzione.

**NOTA:** Una volta, l'opzione *--pretty-print* eseguiva anche il programma. Ora non più.

C'è una differenza significativa tra l'output creato durante la profilazione, e quello creato durante la stampa elegante. L'output della stampa elegante preserva i commenti originali che erano nel programma, anche se la loro posizione può non corrispondere esattamente alle posizioni originali che avevano nel codice sorgente.<sup>5</sup>

Comunque, per una precisa scelta progettuale, l'output della profilazione *omette* i commenti del programma originale. Questo permette di concentrarsi sui dati del conteggio di esecuzione ed evita la tentazione di usare il profilatore per creare una stampa elegante.

Oltre a ciò, l'output stampato in modo elegante non ha l'indentazione iniziale che ha l'output della profilazione. Questo rende agevole la stampa elegante del proprio codice una volta completato lo sviluppo, usando poi il risultato come versione finale del programma.

Poiché la rappresentazione interna del programma è formattata per essere aderente al programma *awk* in questione, la profilatura e la formattazione graziosa (opzione *--pretty-print*) disabilitano automaticamente le ottimizzazioni di default di *gawk*.

La formattazione elegante mantiene anche il formato originale delle costanti numeriche; se sono stati usati dei valori ottali o esadecimali nel codice sorgente, questi compariranno nell'output nello stesso formato con cui sono stati inseriti.

## 12.6 Sommario

- L'opzione *--non-decimal-data* fa sì che *gawk* tratti i dati di input che hanno l'aspetto ottale ed esadecimale come valori ottali ed esadecimali. L'opzione dovrebbe essere usata con prudenza o non usata affatto; è preferibile l'uso di *strtonum()*. Si noti che quest'opzione potrebbe sparire nelle prossime versioni di *gawk*.
- Si può prendere il completo controllo dell'ordinamento nello scorrere il vettore con *'for (indice in vettore)'*, impostando *PROCINFO["sorted\_in"]* al nome di una funzione definita dall'utente che fa il confronto tra elementi del vettore basandosi su indice e valore.
- Analogamente, si può fornire il nome di una funzione di confronto definita dall'utente come terzo argomento di *asort()* o di *asorti()* per controllare come queste funzioni ordinano i vettori. O si può fornire una delle stringhe di controllo predefinite che funzionano per *PROCINFO["sorted\_in"]*.
- Si può usare l'operatore *'|&'* per creare una *pipe* bidirezionale verso un coprocesso. Si legge dal coprocesso con *getline*, ci si scrive sopra con *print* o con *printf*. Usare

<sup>5</sup> *gawk* fa del suo meglio per mantenere la distinzione tra commenti posti dopo delle istruzioni e commenti su righe a sé stanti. Per limiti insiti nell'implementazione, non sempre questo può avvenire in maniera corretta, in particolare nel caso di istruzioni *switch*. I manutentori di *gawk* sperano di poter migliorare la situazione in una futura versione.

`close()` per bloccare il coprocesso completamente o, se necessario, chiudere le comunicazioni bidirezionali in una direzione.

- Usando degli speciali nomi-file con l'operatore `'|&'`, si può aprire una connessione TCP/IP (o UDP/IP) verso host remoti su internet. `gawk` supporta sia IPv4 che IPv6.
- Si possono generare profili del proprio programma con i conteggi del numero di esecuzione di ogni singola istruzione. Questo può essere d'aiuto nel determinare quali parti del programma potrebbero portar via la maggior parte del tempo, consentendo così di aggiustarli più agevolmente. Inviando il segnale `USR1` durante la profilazione `gawk` scrive il profilo, includendo la stack della chiamata alla funzione e prosegue nell'elaborazione.
- Si può anche fare solo una "stampa elegante" del programma.

## 13 Internazionalizzazione con gawk

Tanto tempo fa i produttori di computer scrivevano software che comunicava solo in inglese. Col passare del tempo, i venditori di hardware e di software si sono resi conto che se i loro sistemi avessero comunicato anche nelle lingue materne di paesi dove non si parlava inglese, ciò avrebbe avuto come risultato un incremento delle vendite. Per questo motivo, l'internazionalizzazione e la localizzazione di programmi e sistemi software è divenuta una pratica comune.

Per molti anni la possibilità di fornire l'internazionalizzazione era sostanzialmente limitata ai programmi scritti in C e C++. Questo capitolo descrive la libreria *ad hoc* utilizzata da **gawk** per l'internazionalizzazione e anche il modo in cui le funzionalità che consentono l'internazionalizzazione sono rese disponibili da **gawk** a ogni programma scritto in **awk**. La disponibilità dell'internazionalizzazione a livello di programma **awk** offre ulteriore flessibilità agli sviluppatori di software: non sono più obbligati a scrivere in C o C++ quando l'internazionalizzazione è necessaria in un programma.

### 13.1 Internazionalizzazione e localizzazione

*Internazionalizzazione* significa scrivere (o modificare) un programma una volta sola, in maniera tale che possa usare più di una lingua senza bisogno di ulteriori modifiche al file sorgente. *Localizzazione* significa fornire i dati necessari perché un programma internazionalizzato sia in grado di funzionare con una data lingua. Questi termini si riferiscono comunemente a funzionalità quali la lingua usata per stampare messaggi di errore, quella usata per leggere risposte, e alle informazioni relative al modo di leggere e di stampare dati di tipo numerico o valutarario.

### 13.2 Il comando GNU gettext

**gawk** usa il comando GNU **gettext** per rendere disponibili le proprie funzionalità di internazionalizzazione. L'attenzione del comando GNU **gettext** è rivolta principalmente ai messaggi: stringhe di caratteri stampate da un programma, sia direttamente sia usando la formattazione prevista dalle istruzioni **printf** o **sprintf()**.<sup>1</sup>

Quando si usa il comando GNU **gettext**, ogni applicazione ha il proprio *dominio di testo*. Questo è un nome unico come, p.es., **'kpilot'** o **'gawk'**, che identifica l'applicazione. Un'applicazione completa può avere più componenti: programmi scritti in C o C++, come pure script di **sh** o di **awk**. Tutti i componenti usano lo stesso dominio di testo.

Per andare sul concreto, si supponga di scrivere un'applicazione chiamata **guide**. L'internazionalizzazione per quest'applicazione può essere implementata seguendo nell'ordine i passi qui delineati:

1. Il programmatore esamina i sorgenti di tutti i componenti dell'applicazione **guide** e prende nota di ogni stringa che potrebbe aver bisogno di traduzione. Per esempio, **"'-F': option required"** è una stringa che sicuramente necessita di una traduzione. Una tabella che contenga stringhe che sono nomi di opzioni *non* necessita di traduzione.

---

<sup>1</sup> Per alcuni sistemi operativi, la relativa versione di **gawk** non supporta il comando GNU **gettext**. Per questo motivo, queste funzionalità non sono disponibili nel caso si stia lavorando con uno di questi sistemi operativi. Siamo spiacenti.

(P.es., l'opzione di `gawk --profile` dovrebbe restare immutata, a prescindere dalla lingua locale).

2. Il programmatore indica il dominio di testo dell'applicazione ("`guide`") alla libreria `gettext`, chiamando la funzione `textdomain()`.
3. I messaggi dell'applicazione che vanno tradotti sono estratti dal codice sorgente e messi in un file di tipo *portable object template* [modello di oggetto portabile] di nome `guide.pot`, che elenca le stringhe e le relative traduzioni. Le traduzioni sono inizialmente vuote (esiste la struttura che definisce la stringa tradotta, ma la stringa tradotta è una stringa nulla). Il messaggio originale (normalmente in inglese) è utilizzato come chiave di riferimento per le traduzioni.
4. Per ogni lingua per cui sia disponibile un traduttore, il file `guide.pot` è copiato in un file di tipo *portable object* [oggetto portabile] (dal suffisso `.po`) e le traduzioni sono effettuate su quel file, che viene distribuito con l'applicazione. Per esempio, potrebbe esserci un file `it.po` per la traduzione italiana.
5. Il file `.po` di ogni lingua è convertito in un formato binario, detto *message object* (file `.gmo`). Un file di tipo *message object* contiene i messaggi originali e le loro traduzioni in un formato binario che facilita il ritrovamento delle traduzioni quando l'applicazione viene eseguita.
6. Quando `guide` è compilato e installato, i file binari contenenti le traduzioni sono installati in una directory standard.
7. Durante la fase di prova e sviluppo, è possibile chiedere a `gettext` di usare un file `.gmo` in una directory diversa da quella standard, usando la funzione `bindtextdomain()`.
8. Quando viene eseguito, il programma `awk guide` cerca ogni stringa da tradurre facendo una chiamata a `gettext()`. La stringa ricevuta in ritorno è la stringa tradotta, se è stata trovata, o la stringa originale, se una traduzione non è disponibile.
9. Se necessario, è possibile procurarsi dei messaggi tradotti da un dominio di testo diverso da quello proprio dell'applicazione, senza dover altalenare fra questo secondo dominio e quello dell'applicazione.

In C (o C++), la marcatura della stringa la ricerca dinamica della traduzione si fanno inserendo ogni stringa da tradurre in una chiamata a `gettext()`:

```
printf("%s", gettext("Don't Panic!\n"));
```

Gli strumenti software che estraggono messaggi dal codice sorgente individuano tutte le stringhe racchiuse nelle chiamate a `gettext()`.

Gli sviluppatori del comando GNU `gettext`, riconoscendo che continuare a immettere `'gettext(...)` è sia faticoso che poco elegante da vedere, usano la macro `'_'` (un trattino basso) per facilitare la cosa:

```
/* Nel file di intestazione standard: */
#define _(str) gettext(str)

/* Nel testo del programma: */
printf("%s", _("Don't Panic!\n"));
```

Questo permette di ridurre la digitazione extra a solo tre caratteri per ogni stringa da tradurre e inoltre migliora di molto la leggibilità.

Ci sono *categorie* di localizzazione per tipi diversi di informazioni legate a una particolare localizzazione. Le categorie di localizzazione note a `gettext` sono:

#### LC\_MESSAGES

Testo dei messaggi. Questa è la categoria di default usata all'interno di `gettext`, ma è possibile specificarne esplicitamente una differente, se necessario. (Questo non è quasi mai necessario.)

#### LC\_COLLATE

Informazioni sull'ordinamento alfabetico (cioè, come caratteri diversi e/o gruppi di carattere sono ordinati in un dato linguaggio).

**LC\_CTYPE** Informazioni sui singoli caratteri (alfabetico, numerico, maiuscolo o minuscolo, etc.), come pure sulla codifica dei caratteri. Quest'informazione è utilizzata per stabilire le classi di caratteri come definite nello standard POSIX, nelle espressioni regolari, come p. es. `/[[[:alnum:]]/` (si veda la [Sezione 3.4 \[Usare espressioni tra parentesi quadre\]](#), pagina 55).

#### LC\_MONETARY

Le informazioni di tipo monetario, quali il simbolo della moneta, e se il simbolo va prima o dopo il valore numerico.

#### LC\_NUMERIC

Informazioni di tipo numerico, quali il carattere da usare per separare le cifre decimali e quello per separare le migliaia.<sup>2</sup>

**LC\_TIME** Informazioni relative alle date e alle ore, come l'uso di ore nel formato a 12 ore oppure a 24 ore, il mese stampato prima o dopo il giorno in una data, le abbreviazioni dei mesi nella lingua locale, e così via.

**LC\_ALL** Tutte le categorie viste sopra. (Non molto utile nel contesto del comando `gettext`.)

**NOTA:** Come descritto in [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141, le variabili d'ambiente che hanno lo stesso nome delle categorie di localizzazione (`LC_CTYPE`, `LC_ALL`, etc.) influenzano il comportamento di `gawk` (e quello di altri programmi di utilità).

Solitamente, queste variabili influenzano anche il modo con cui la libreria `gettext` trova le traduzioni. Tuttavia, la variabile d'ambiente `LANGUAGE` prevale sulle variabili della famiglia `LC_XXX`. Molti sistemi GNU/Linux possono aver definito questa variabile senza esplicitamente notificarlo all'utente, e questo potrebbe far sì che `gawk` non riesca a trovare le traduzioni corrette. Se si incontra questa situazione, occorre controllare se la variabile d'ambiente `LANGUAGE` è definita, e, in questo caso, va usato il comando `unset` per rimuoverla.

Per il test di traduzioni dei messaggi inviati da `gawk` stesso, si può impostare la variabile d'ambiente `GAWK_LOCALE_DIR`. Si veda la documentazione per la funzione `C bindtextdomain()`, e si veda anche [Sezione 2.5.3 \[Le variabili d'ambiente.\]](#), pagina 43.

<sup>2</sup> Gli americani usano una virgola ogni tre cifre decimali, e un punto per separare la parte decimale di un numero, mentre molti europei (fra cui gli italiani) fanno esattamente l'opposto: 1,234.56 invece che 1.234,56.

### 13.3 Internazionalizzare programmi awk

`gawk` prevede le seguenti variabili per l'internazionalizzazione:

#### TEXTDOMAIN

Questa variabile indica il dominio di testo dell'applicazione. Per compatibilità con il comando GNU `gettext`, il valore di default è `"messages"`.

`_"questo è un messaggio da tradurre"`

Costanti di tipo stringa marcate con un trattino basso iniziale sono candidate per essere tradotte al momento dell'esecuzione del programma `gawk`. Costanti di tipo stringa non precedute da un trattino basso non verranno tradotte.

`gawk` fornisce le seguenti funzioni al servizio dell'internazionalizzazione:

`dcgettext(string [, dominio [, categoria]])`

Restituisce la traduzione di *stringa* nel dominio di testo *dominio* per la categoria di localizzazione *categoria*. Il valore di default per *dominio* è il valore corrente di `TEXTDOMAIN`. Il valore di default per *categoria* è `"LC_MESSAGES"`.

Se si assegna un valore a *categoria*, dev'essere una stringa uguale a una delle categorie di localizzazione note, descritte nella precedente sezione. Si deve anche specificare un dominio di testo. Si usi `TEXTDOMAIN` se si desidera usare il dominio corrente.

**ATTENZIONE:** L'ordine degli argomenti per la versione `awk` della funzione `dcgettext()` è differente, per una scelta di progetto, dall'ordine degli argomenti passati alla funzione C che ha lo stesso nome. L'ordine della versione `awk` è stato scelto per amore di semplicità e per consentire di avere dei valori di default per gli argomenti che fossero il più possibile simili, come stile, a quello di `awk`.

`dcngettext(stringa1, stringa2, numero [, dominio [, categoria]])`

Restituisce la forma, singolare o plurale, da usare a seconda del valore di *numero* per la traduzione di *stringa1* e *stringa2* nel dominio di testo *dominio* per la categoria di localizzazione *categoria*. *stringa1* è la variante al singolare in inglese di un messaggio, e *stringa2* è la variante al plurale in inglese dello stesso messaggio. Il valore di default per *dominio* è il valore corrente di `TEXTDOMAIN`. Il valore di default per *categoria* è `"LC_MESSAGES"`.

Valgono le stesse osservazioni riguardo all'ordine degli argomenti fatte a proposito della funzione `dcgettext()`.

`bindtextdomain(directory [, dominio])`

Cambia la *directory* nella quale `gettext` va a cercare i file `.gmo`, per il caso in cui questi non possano risiedere nelle posizioni standard (p.es., in fase di test). Restituisce la *directory* alla quale *dominio* è "collegato".

Il *dominio* di default è il valore di `TEXTDOMAIN`. Se l'argomento *directory* è impostato alla stringa nulla (`"`), `bindtextdomain()` restituisce il collegamento corrente applicabile al *dominio* specificato.

Per usare queste funzionalità in un programma `awk`, va seguita la procedura qui indicata:

1. Impostare la variabile `TEXTDOMAIN` al dominio di testo del programma. È meglio fare ciò all'interno di una regola `BEGIN` (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali `BEGIN` ed `END`\]](#), pagina 148), ma si può anche fare dalla riga di comando, usando l'opzione `-v` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):

```
BEGIN {
    TEXTDOMAIN = "guide"
    ...
}
```

2. Marcare tutte le stringhe traducibili antepoendo loro un trattino basso ('\_'). Il trattino *deve* essere adiacente ai doppi apici di apertura della stringa. Per esempio:

```
print _"hello, world"
x = _"you goofed"
printf(_"Number of users is %d\n", nusers)
```

3. Se le stringhe da visualizzare sono create dinamicamente, è ancora possibile tradurle, usando la funzione predefinita `dcgettext()`:<sup>3</sup>

```
if (assonnato)
    messaggio = dcgettext("%d clienti mi scocciano\n", "adminprog")
else
    messaggio = dcgettext("mi diverto con %d clienti\n", "adminprog")
printf(messaggio, numero_clienti)
```

In questo esempio, la chiamata a `dcgettext()` specifica un diverso dominio di testo ("adminprog") in cui trovare il messaggio, ma usa la categoria di default "LC\_MESSAGES".

Il precedente esempio funziona solo se `numero_clienti` è un numero maggiore di uno. Per questo esempio sarebbe più appropriato usare la funzione `dcngettext()`:

```
if (assonnato)
    messaggio = dcngettext("%d cliente mi scoccia\n",
        "%d clienti mi scocciano\n",
        numero_clienti, "adminprog")
else
    messaggio = dcngettext("mi diverto con %d cliente\n",
        "mi diverto con %d clienti\n",
        numero_clienti, "adminprog")
printf(messaggio, numero_clienti)
```

4. In fase di sviluppo, si può scegliere di tenere il file `.gmo` in una directory a parte, solo per provarlo. Ciò si fa con la funzione predefinita `bindtextdomain()`:

```
BEGIN {
    TEXTDOMAIN = "guide"    # dominio di testo regolare
    if (Testing) {
        # dove trovare il file in prova
        bindtextdomain("testdir")
        # joe si occupa del programma adminprog
        bindtextdomain("../joe/testdir", "adminprog")
    }
```

---

<sup>3</sup> I miei ringraziamenti a Bruno Haible per questo esempio.

```

    }
    ...
}

```

Si veda la [Sezione 13.5 \[Un semplice esempio di internazionalizzazione.\]](#), pagina 357, per un programma di esempio che illustra i passi da seguire per creare e usare traduzioni nei programmi `awk`.

## 13.4 Traduzione dei programmi `awk`

Dopo aver marcato le stringhe che si desidera tradurre in un programma, queste vanno estratte per creare il file iniziale `.pot`. Durante la traduzione, è spesso utile modificare l'ordine nel quale gli argomenti passati a `printf` vengono stampati.

L'opzione da riga di comando `--gen-pot` di `gawk` serve a estrarre i messaggi, ed è esposta qui di seguito. Dopo di che, verrà illustrata la possibilità di modificare l'ordine in cui l'istruzione `printf` stampa gli argomenti che le sono passati in fase di esecuzione.

### 13.4.1 Estrarre stringhe marcate

Una volta che il programma `awk` funziona, e tutte le stringhe sono state marcate ed è stato impostato (e forse fissato) il dominio di testo, è ora di preparare le traduzioni. Per prima cosa, usando l'opzione di riga di comando `--gen-pot`, si crea il file iniziale `.pot`:

```
gawk --gen-pot -f guide.awk > guide.pot
```

Quando viene chiamato specificando `--gen-pot`, `gawk` non esegue il programma. Il programma viene esaminato come al solito, e tutte le stringhe che sono state marcate per essere tradotte vengono scritte nello standard output, nel formato di un file Portable Object di GNU `gettext`. L'output comprende anche quelle stringhe costanti che appaiono come primo argomento della funzione `dcgettext()` o come primo e secondo argomento della funzione `dcngettext()`.<sup>4</sup> Il file `.pot` così generato andrebbe distribuito insieme al programma `awk`; i traduttori potranno eventualmente utilizzarlo per preparare delle traduzioni, le quali potranno a loro volta essere distribuite. Si veda la [Sezione 13.5 \[Un semplice esempio di internazionalizzazione.\]](#), pagina 357, per una lista esauriente dei passi necessari per creare e testare traduzioni per il programma `guide`.

### 13.4.2 Riordinare argomenti di `printf`

Le stringhe di formattazione per `printf` e `sprintf()` (si veda la [Sezione 5.5 \[Usare l'istruzione `printf` per stampe sofisticate\]](#), pagina 98) hanno un problema speciale con le traduzioni. Si consideri il seguente esempio:<sup>5</sup>

```
printf(_("String '%s' has %d characters\n",
        string, length(string)))
```

Una possibile traduzione in italiano di questo messaggio potrebbe essere:

```
"%d è la lunghezza della stringa '%s'\n"
```

<sup>4</sup> Il comando di utilità `xgettext` che fa parte del pacchetto distribuito come `gettext` è in grado di gestire i file di tipo `.awk`.

<sup>5</sup> Questo esempio è preso in prestito dal manuale del comando GNU `gettext`.

Il problema dovrebbe essere ovvio: l'ordine delle specifiche di formattazione è differente da quello originale! `gettext()`, che pure può restituire la stringa tradotta in fase di esecuzione, non è in grado di cambiare l'ordine degli argomenti nella chiamata a `printf`.

Per risolvere questo problema, gli specificatori di formato di `printf` possono avere un elemento in più, facoltativo, detto *specificatore posizionale*. Per esempio:

```
"%2$d è la lunghezza della stringa '%1$s'\n"
```

Qui, lo specificatore posizionale consiste in un numero intero, che indica quale argomento utilizzare, seguito da un carattere '\$'. I numeri partono da uno, e la stringa di formattazione vera e propria *non* è inclusa. Quindi, nell'esempio seguente, `'stringa'` è il primo argomento e `'length(stringa)'` è il secondo:

```
$ gawk 'BEGIN {
>     stringa = "Non v\47allarmate!"
>     printf "%2$d caratteri compongono \"%1$s\"\n",
>           stringa, length(stringa)
> }'
+ 16 caratteri compongono "Non v\47allarmate!"
```

Se presenti, gli specificatori posizionali precedono, nella specifica di formato, i flag, la larghezza del campo e/o la precisione.

Gli specificatori posizionali possono essere usati anche se si specifica una larghezza dinamica dei campi, e della capacità di precisione:

```
$ gawk 'BEGIN {
>     printf("%*.*s\n", 10, 20, "hello")
>     printf("%3$*2$.*1$s\n", 20, 10, "hello")
> }'
+      hello
+      hello
```

**NOTA:** Se si usa '\*' con uno specificatore posizionale, il carattere '\*' viene per primo, seguito dal numero che indica la posizione, a sua volta seguito dal '\$'. Ciò può parere poco intuitivo.

`gawk` non consente di mischiare specificatori di formato standard con altri contenenti degli specificatori posizionali in una stessa stringa di formato:

```
$ gawk 'BEGIN { printf "%d %3$s\n", 1, 2, "ciao" }'
[error] gawk: riga com.:1: fatale: 'count$' va usato per tutti
[error] i formati o per nessuno
```

**NOTA:** Ci sono alcuni casi patologici in cui `gawk` potrebbe non inviare alcun messaggio diagnostico, anche quando sarebbe necessario. In tali casi, l'output può non essere quello atteso. Rimane sempre una pessima idea quella di tentare di mischiare i formati, anche se `gawk` non riesce ad accorgersene.

Sebbene gli specificatori posizionali possano essere usati direttamente nei programmi `awk`, il motivo per cui sono stati introdotti è perché siano d'aiuto nel produrre traduzioni corrette della stringa di formattazione in lingue differenti da quella nella quale il programma è stato originariamente scritto.

### 13.4.3 Problemi di portabilità a livello di awk

Le funzionalità di internazionalizzazione di **gawk** sono state appositamente implementate per avere il minimo impatto possibile sulla portabilità, verso altre versioni di **awk**, dei programmi **awk** che ne fanno uso. Si consideri questo programma:

```
BEGIN {
    TEXTDOMAIN = "guide"
    if (Test_Guide) # da impostare tramite -v
        bindtextdomain("/test/guide/messages")
    print _"don't panic!"
}
```

Per il modo in cui è scritto, non funzionerà con altre versioni di **awk**. Tuttavia, è in realtà quasi portabile, e richiede modifiche minime:

- Le assegnazioni di valori a **TEXTDOMAIN** non avranno effetto alcuno, perché **TEXTDOMAIN** non è una variabile speciale in altre implementazioni di **awk**.
- Versioni Non-GNU di **awk** considerano le stringhe marcate come la concatenazione di una variabile di nome **\_** con la stringa che viene subito dopo.<sup>6</sup> Tipicamente, la variabile **\_** ha come valore la stringa nulla (**"**), il che produce come risultato la stringa originale.
- Definendo delle funzioni “fittizie” per sostituire **dcgettext()**, **dcngettext()** e **bindtextdomain()**, il programma **awk** può essere reso eseguibile, ma tutti i messaggi verranno inviati nella lingua originale del programma. Per esempio:

```
function bindtextdomain(dir, domain)
{
    return dir
}

function dcgettext(string, domain, category)
{
    return string
}

function dcngettext(string1, string2, number, domain, category)
{
    return (number == 1 ? string1 : string2)
}
```

- L'uso di specificazioni posizionali in **printf** o **sprintf()** *non* è portabile. Per supportare **gettext()** nella programmazione in linguaggio C, molte versioni C di **sprintf()** supportano specificatori posizionali. Ma la cosa funziona solo se nella chiamata di funzione sono stati specificati argomenti a sufficienza. Molte versioni di **awk** passano i formati e gli argomenti di **printf**, senza modificarli, alla funzione di libreria in linguaggio C **sprintf()**, ma solo un formato e un argomento alla volta. Quel che succede se si usa una specificazione posizionale resta indeterminato. Tuttavia, poiché le specificazioni posizionali sono usate principalmente per le stringhe di formattazione *tradotte*, e poiché le versioni non-GNU di **awk** non utilizzano mai le stringhe tradotte, ciò non dovrebbe, in pratica, causare problemi.

<sup>6</sup> Questo è un buon materiale per una gara di “Oscurità **awk**”.

### 13.5 Un semplice esempio di internazionalizzazione.

Vediamo ora un esempio dettagliato di come internazionalizzare e localizzare un semplice programma `awk`, usando come nostro programma sorgente originale il file `guide.awk`:

```
BEGIN {
    TEXTDOMAIN = "guide"
    bindtextdomain(".") # per la fase di test
    print _"Don't Panic"
    print _"The Answer Is", 42
    print "Pardon me, Zaphod who?"
}
```

Eseguire ‘`gawk --gen-pot`’ per creare il file `.pot`:

```
$ gawk --gen-pot -f guide.awk > guide.pot
```

Questo produce:

```
#: guide.awk:4
msgid "Don't Panic"
msgstr ""

#: guide.awk:5
msgid "The Answer Is"
msgstr ""
```

Questo modello di *portable object* va salvato e riutilizzato per ogni lingua in cui l'applicazione viene tradotta. La stringa `msgid` è seguita dalla stringa originale da tradurre, e la stringa `msgstr` conterrà la traduzione.

**NOTA:** Le stringhe non aventi come prefisso un trattino basso non sono inserite nel file `guide.pot`.

Successivamente, i messaggi devono essere tradotti. Questa è una traduzione in un ipotetico dialetto dell'inglese, chiamato “Mellow”:<sup>7</sup>

```
$ cp guide.pot guide-mellow.po
Aggiungere traduzioni al file guide-mellow.po ...
```

Ecco le traduzioni:

```
#: guide.awk:4
msgid "Don't Panic"
msgstr "Hey man, relax!"

#: guide.awk:5
msgid "The Answer Is"
msgstr "Like, the scoop is"
```

Il passo successivo è di creare la directory che contenga il file binario con le traduzioni dei messaggi (file `.mo` [message object]) e creare in quella directory il file `guide.mo`. Si presume che il file in questione debba essere usato nella localizzazione `en_US.UTF-8`, perché

<sup>7</sup> Forse sarebbe meglio chiamarlo “Hippy.” Meglio non indagare oltre.

si deve usare un nome di localizzazione che sia noto alle routine del comando C `gettext`. La disposizione delle directory qui utilizzata è standard per il comando GNU `gettext` sui sistemi GNU/Linux. Altre versioni di `gettext` possono usare una disposizione differente:

```
$ mkdir en_US.UTF-8 en_US.UTF-8/LC_MESSAGES
```

Il programma di utilità `msgfmt` effettua la conversione dal file leggibile, in formato testo, `.po` nel file, in formato binario, `.mo`. Per default, `msgfmt` crea un file di nome `messages`. A questo file dev'essere assegnato un nome appropriato, e va messo nella directory predisposta (usando l'opzione `-o`) in modo che `gawk` sia in grado di trovarlo:

```
$ msgfmt guide-mellow.po -o en_US.UTF-8/LC_MESSAGES/guide.mo
```

Infine, eseguiamo il programma per provare se funziona:

```
$ gawk -f guide.awk
+ Hey man, relax!
+ Like, the scoop is 42
+ Pardon me, Zaphod who?
```

Se le tre funzioni che rimpiazzano `dcgettext()`, `dcngettext()`, e `bindtextdomain()` (si veda la [Sezione 13.4.3 \[Problemi di portabilità a livello di awk\]](#), pagina 356) sono contenute in un file di nome `libintl.awk`, è possibile eseguire `guide.awk` senza modificarlo, nel modo seguente:

```
$ gawk --posix -f guide.awk -f libintl.awk
+ Don't Panic
+ The Answer Is 42
+ Pardon me, Zaphod who?
```

## 13.6 gawk stesso è internazionalizzato

Il comando `gawk` stesso è stato internazionalizzato usando il pacchetto GNU `gettext`. (GNU `gettext` è descritto in maniera esauriente in [GNU gettext utilities](#).) Al momento in cui questo libro è stato scritto, la versione più recente di GNU `gettext` è [versione 0.19.4](#).

Se esiste una traduzione dei messaggi di `gawk`, `gawk` invia messaggi, avvertimenti, ed errori fatali utilizzando la lingua locale.

## 13.7 Sommario

- Internazionalizzazione significa scrivere un programma in modo che possa interagire in molte lingue senza che sia necessario cambiare il codice sorgente. Localizzazione significa fornire i dati necessari perché un programma internazionalizzato possa interagire usando una determinata lingua.
- `gawk` usa il comando GNU `gettext` per consentire l'internazionalizzazione e la localizzazione di programmi `awk`. Un dominio di testo di un programma identifica il programma, e consente di raggruppare tutti i messaggi e gli altri dati del programma in un solo posto.
- Si marcano le stringhe in un programma da tradurre preponendo loro un trattino basso. Una volta fatto questo, queste stringhe sono estratte in un file `.pot`. Questo file è copiato, per ogni lingua, in un file `.po` e i file `.po` sono compilati in file `.gmo` che saranno usati in fase di esecuzione del programma.

- È possibile usare specificazioni posizionali con le istruzioni `sprintf()` e `printf` per modificare la posizione del valore degli argomenti nelle stringhe di formato e nell'output. Ciò è utile nella traduzione di stringhe di formattazione dei messaggi.
- Le funzionalità di internazionalizzazione sono state progettate in modo da poter essere facilmente gestite in un programma `awk` standard.
- Anche il comando `gawk` è stato internazionalizzato e viene distribuito con traduzioni in molte lingue dei messaggi inviati in fase di esecuzione.



## 14 Effettuare il debug dei programmi awk

Sarebbe bello se i programmi per il calcolatore funzionassero perfettamente la prima volta che vengono eseguiti, ma nella vita reale questo accade raramente, qualunque sia la complessità dei programmi. Perciò la maggior parte dei linguaggi di programmazione hanno a disposizione degli strumenti che facilitano la ricerca di errori (*debugging*) nei programmi, e `awk` non fa eccezione.

Il *debugger* di `gawk` è di proposito costruito sul modello del debugger da riga di comando GNU Debugger (GDB). Se si ha familiarità con GDB, sarà facile imparare come usare `gawk` per eseguire il debug dei propri programmi.

### 14.1 Introduzione al debugger di gawk

Questa sezione, dopo un'introduzione sul debug in generale, inizia la trattazione del debug in `gawk`.

#### 14.1.1 Generalità sul debug

(Se si sono usati dei debugger in altri linguaggi, si può andare direttamente alla [Sezione 14.1.3 \[Il debug di awk\], pagina 362.](#))

Naturalmente, un programma di debug non può correggere gli errori al posto del programmatore, perché non può sapere quello che il programmatore o gli utenti considerano un “bug” e non una “funzionalità”. (Talvolta, anche noi umani abbiamo difficoltà nel determinarlo.) In quel caso, cosa ci si può aspettare da un tale strumento? La risposta dipende dal linguaggio su cui si effettua il debug, comunque in generale ci si può attendere almeno questo:

- La possibilità di osservare l'esecuzione delle istruzioni di un programma una per una, dando al programmatore l'opportunità di pensare a quel che accade a una scala temporale di secondi, minuti od ore, piuttosto che alla scala dei nanosecondi alla quale normalmente viene eseguito il codice.
- L'opportunità, non solo di osservare passivamente le operazioni del programma, ma di controllarlo e tentare diversi percorsi di esecuzione, senza dover modificare i file sorgenti.
- La possibilità di vedere i valori o i dati nel programma in qualsiasi punto dell'esecuzione, e anche di cambiare i dati al volo, per vedere come questo influisca su ciò che accade dopo. (Questo include spesso la capacità di esaminare le strutture interne dei dati oltre alle variabili che sono state effettivamente definite nel codice del programma.)
- La possibilità di ottenere ulteriori informazioni sullo stato del programma o anche sulle sue strutture interne.

Tutti questi strumenti sono di grande aiuto e permettono di usare l'abilità che si possiede e la comprensione che si ha degli obiettivi del programma per trovare dove si verificano i problemi (o, in alternativa, per comprendere meglio la logica di un programma funzionante, di cui si sia l'autore, o anche di un programma scritto da altri).

#### 14.1.2 Concetti fondamentali sul debug

Prima di entrare nei dettagli, dobbiamo introdurre diversi importanti concetti che valgono per tutti i debugger. La seguente lista definisce i termini usati nel resto di questo capitolo:

*Stack frame*

Durante la loro esecuzione i programmi normalmente chiamano delle funzioni. Una funzione può a sua volta chiamarne un'altra, o può richiamare se stessa (ricorsione). La catena di funzioni chiamate (il programma principale chiama A, che chiama B, che chiama C) può essere vista come una pila di funzioni in esecuzione: la funzione correntemente in esecuzione è quella in cima alla pila, e quando questa finisce (ritorna al chiamante), quella immediatamente sotto diventa la funzione attiva. Tale pila (*stack*) è chiamata *call stack* (pila delle chiamate).

Per ciascuna funzione della pila delle chiamate (*call stack*), il sistema mantiene un'area di dati che contiene i parametri della funzione, le variabili locali e i valori di ritorno, e anche ogni altra informazione “contabile” necessaria per gestire la pila delle chiamate. Quest'area di dati è chiamata *stack frame*.

Anche **gawk** segue questo modello, e permette l'accesso alla pila delle chiamate e a ogni *stack frame*. È possibile esaminare la pila delle chiamate, e anche sapere da dove ciascuna funzione sulla pila è stata invocata. I comandi che stampano la pila delle chiamate stampano anche le informazioni su ogni *stack frame* (come vedremo più avanti in dettaglio).

*Punto d'interruzione*

Durante le operazioni di debug, spesso si preferisce lasciare che il programma venga eseguito finché non raggiunge un certo punto, e da quel punto in poi si continua l'esecuzione un'istruzione alla volta. Il modo per farlo è quello di impostare un *punto d'interruzione* all'interno del programma. Un punto d'interruzione è il punto dove l'esecuzione del programma dovrebbe interrompersi (fermarsi), in modo da assumere il controllo dell'esecuzione del programma. Si possono aggiungere e togliere quanti punti d'interruzione si vogliono.

*Punto d'osservazione*

Un punto d'osservazione è simile a un punto d'interruzione. La differenza è che i punti d'interruzione sono orientati attorno al codice; fermano il programma quando viene raggiunto un certo punto nel codice. Un punto d'osservazione, invece, fa fermare il programma quando è stato cambiato il *valore di un dato*. Questo è utile, poiché a volte succede che una variabile riceva un valore errato, ed è difficile rintracciare il punto dove ciò accade solo leggendo il codice sorgente. Usando un punto d'osservazione, si può fermare il programma in qualunque punto vi sia un'assegnazione di variabile, e di solito si individua il codice che genera l'errore abbastanza velocemente.

**14.1.3 Il debug di awk**

Il debug di un programma **awk** ha delle particolarità proprie, che non sono presenti in programmi scritti in altri linguaggi.

Prima di tutto, il fatto che i programmi **awk** ricevano generalmente l'input riga per riga da uno o più file e operino su tali righe usando regole specifiche, rende particolarmente agevole organizzare l'esame dell'esecuzione del programma facendo riferimento a tali regole. Come vedremo, ogni regola **awk** viene trattata quasi come una chiamata di funzione, col proprio specifico blocco di istruzioni.

Inoltre, poiché **awk** è un linguaggio deliberatamente molto conciso, è facile perdere di vista tutto ciò che avviene “dentro” ogni riga di codice **awk**. Il debugger dà l’opportunità di guardare le singole istruzioni primitive la cui esecuzione è innescata dai comandi di alto livello di **awk**.

## 14.2 Esempio di sessione di debug di gawk

Per illustrare l’uso di **gawk** come debugger, vediamo un esempio di sessione di debug. Come esempio verrà usata l’implementazione **awk** del comando POSIX **uniq** descritta in precedenza (si veda la [Sezione 11.2.6 \[Stampare righe di testo non duplicate\]](#), pagina 296).

### 14.2.1 Come avviare il debugger

Per avviare il debugger in **gawk** si richiama il comando esattamente come al solito, specificando solo un’opzione aggiuntiva, **--debug**, o la corrispondente opzione breve **-D**. I file (o il file) che contengono il programma e ogni codice ulteriore sono immessi sulla riga di comando come argomenti a una o più opzioni **-f**. (**gawk** non è progettato per eseguire il debug di programmi scritti sulla riga di comando, ma solo per quello di programmi che risiedono su file.) Nel nostro caso, il debugger verrà invocato in questo modo:

```
$ gawk -D -f getopt.awk -f join.awk -f uniq.awk -1 file_di_input
```

dove entrambi i file **getopt.awk** e **uniq.awk** sono in **\$AWKPATH**. (Gli utenti esperti di GDB o debugger simili dovrebbero tener presente che questa sintassi è leggermente differente da quello che sono abituati a usare. Col debugger di **gawk**, si danno gli argomenti per eseguire il programma nella riga di comando al debugger piuttosto che come parte del comando **run** al prompt del debugger.) L’opzione **-1** è un’opzione per **uniq.awk**.

Invece di eseguire direttamente il programma sul **file\_di\_input**, come **gawk** farebbe normalmente, il debugger semplicemente carica i file sorgenti del programma, li compila internamente, e poi mostra la riga d’invito:

```
gawk>
```

da dove si possono impartire i comandi al debugger. Sin qui non è stato ancora eseguito nessun codice.

### 14.2.2 Trovare il bug

Poniamo di avere un problema usando (una versione difettosa di) **uniq.awk** nella modalità “salta-campi”, perché sembra che non catturi le righe che dovrebbero essere identiche dopo aver saltato il primo campo, come:

```
awk, ecco un programma meraviglioso!
gawk, ecco un programma meraviglioso!
```

Questo potrebbe accadere se noi pensassimo (come in C) che i campi in un record siano numerati prendendo come base lo zero, per cui, invece di scrivere:

```
campi_ultima = join(vettore_ultima, contatore_file+1, n)
campi_corrente = join(vettore_corrente, contatore_file+1, m)
```

abbiamo scritto:

```
campi_ultima = join(vettore_ultima, contatore_file, n)
campi_corrente = join(vettore_corrente, contatore_file, m)
```

La prima cosa da fare quando si tenta di indagare su un problema come questo è quella di mettere un punto d'interruzione (*breakpoint*) nel programma, in modo da poterlo vedere al lavoro e catturare quello che non va. Una posizione ragionevole per un punto d'interruzione in `uniq.awk` è all'inizio della funzione `se_sono_uguali()`, che confronta la riga corrente con la precedente. Per impostare il punto d'interruzione, usare il comando `b` (*breakpoint*):

```
gawk> b se_sono_uguali
  └ Breakpoint 1 impostato al file 'uniq.awk', riga 63
```

Il debugger mostra il file e il numero di riga dove si trova il punto d'interruzione. Ora bisogna immettere `'r'` o `'run'` e il programma viene eseguito fino al primo punto d'interruzione:

```
gawk> r
  └ Partenza del programma:
  └ Mi fermo in Rule ...
  └ Breakpoint 1, se_sono_uguali(n, m, campi_ultima, campi_corrente,
  └                                     vettore_ultima, vettore_corrente)
  └ a 'uniq.awk':63
  └ 63          if (contatore_file == 0 && conta_caratteri == 0)
gawk>
```

Ora possiamo osservare cosa accade all'interno del nostro programma. Prima di tutto, vediamo come siamo arrivati a questo punto. Sulla riga di comando battiamo `'bt'` (che sta per "backtrace"), e il debugger risponde con un listato degli *stack frame* correnti:

```
gawk> bt
  └ #0 se_sono_uguali(n, m, campi_ultima, campi_corrente,
  └                                     vettore_ultima, vettore_corrente)
  └ a 'uniq.awk':63
  └ #1 in main() a 'uniq.awk':88
```

Questo ci dice che la funzione `se_sono_uguali()` è stata chiamata dal programma principale alla riga 88 del file `uniq.awk`. (Questo non sorprende, perché è questa l'unica chiamata a `se_sono_uguali()` nel programma, però in programmi più complessi, sapere chi ha chiamato una funzione e con quali parametri può essere la chiave per trovare l'origine del problema.)

Ora che siamo in `se_sono_uguali()`, possiamo iniziare a guardare i valori di alcune variabili. Immaginiamo di battere `'p n'` (`p` sta per *print* [stampa]). Ci aspetteremo di vedere il valore di `n`, un parametro di `se_sono_uguali()`. In realtà, il debugger ci dà:

```
gawk> p n
  └ n = untyped variable
```

In questo caso, `n` è una variabile locale non inizializzata, perché la funzione è stata chiamata senza argomenti (si veda la [Sezione 6.4 \[Chiamate di funzione\]](#), pagina 138).

Una variabile più utile da visualizzare potrebbe essere la seguente:

```
gawk> p $0
  └ $0 = "gawk, ecco un programma meraviglioso!"
```

All'inizio questo potrebbe lasciare un tantino perplessi, perché è la seconda riga dell'input del test. Vediamo `NR`:

```
gawk> p NR
  └ NR = 2
```

Come si può vedere, `se_sono_uguali()` è stata chiamata solo per la seconda riga del file. Naturalmente, ciò accade perché il nostro programma contiene una regola per `'NR == 1'`:

```
NR == 1 {
    ultima = $0
    next
}
```

Bene, controlliamo che questa funzioni correttamente:

```
gawk> p ultima
+ ultima = "awk, ecco un programma meraviglioso!"
```

Tutto ciò che è stato fatto fin qui ha verificato che il programma funziona come previsto fino alla chiamata a `se_sono_uguali()` compresa; quindi il problema dev'essere all'interno di questa funzione. Per indagare ulteriormente, iniziamo a “scorrere una ad una” le righe di `se_sono_uguali()`. Cominciamo col battere ‘n’ (per “next” [successivo]):

```
gawk> n
+ 66          if (contatore_file > 0) {
```

Questo ci dice che `gawk` ora è pronto per eseguire la riga 66, che decide se assegnare alle righe il trattamento speciale “salta-campi” indicato dall'opzione sulla riga di comando `-1`. (Si noti che abbiamo saltato da dov'eravamo prima, alla riga 63, a qui, perché la condizione nella riga 63, `'if (contatore_file == 0 && conta_caratteri == 0)'`, era falsa.)

Continuando a scorrere le righe, ora raggiungiamo la divisione del record corrente e dell'ultimo:

```
gawk> n
+ 67          n = split(ultima, vettore_ultima)
gawk> n
+ 68          m = split($0, vettore_corrente)
```

A questo punto, potremmo stare a vedere in quante parti il nostro record è stato suddiviso, quindi proviamo a osservare:

```
gawk> p n m vettore_ultima vettore_corrente
+ n = 5
+ m = untyped variable
+ vettore_ultima = array, 5 elements
+ vettore_corrente = untyped variable
```

(Il comando `p` può accettare più argomenti, analogamente all'istruzione di `awk print`.)

Questo ci lascia piuttosto perplessi. Tutto ciò che abbiamo trovato è che ci sono cinque elementi in `vettore_ultima`; `m` e `vettore_corrente` non hanno valori perché siamo alla riga 68 che non è ancora stata eseguita. Questa informazione è abbastanza utile (ora sappiamo che nessuna delle parole è stata lasciata fuori accidentalmente), ma sarebbe desiderabile vedere i valori del vettore.

Una prima possibilità è quella di usare degli indici:

```
gawk> p vettore_ultima[0]
+ "0" non presente nel vettore 'vettore_ultima'
```

Oops!

```
gawk> p vettore_ultima[1]
```

```
    └ vettore_ultima["1"] = "awk,"
```

Questo metodo sarebbe piuttosto lento per un vettore con 100 elementi, per cui **gawk** fornisce una scorciatoia (che fa venire in mente un altro linguaggio che non nominiamo):

```
gawk> p @vettore_ultima
└ vettore_ultima["1"] = "awk,"
└ vettore_ultima["2"] = "ecco"
└ vettore_ultima["3"] = "un"
└ vettore_ultima["4"] = "programma"
└ vettore_ultima["5"] = "meraviglioso!"
```

Finora, sembra che tutto vada bene. Facciamo un altro passo, o anche due:

```
gawk> n
└ 69                campi_ultima = join(vettore_ultima, contatore_file, n)
gawk> n
└ 70                campi_corrente = join(vettore_corrente, contatore_file, m)
```

Bene, eccoci arrivati al nostro errore (ci spiace di aver rovinato la sorpresa). Quel che avevamo in mente era di unire i campi a partire dal secondo per creare il record virtuale da confrontare, e se il primo campo aveva il numero zero, questo avrebbe funzionato. Vediamo quel che abbiamo finora:

```
gawk> p campi_ultima campi_corrente
└ campi_ultima = "awk, ecco un programma meraviglioso!"
└ campi_corrente = "gawk, ecco un programma meraviglioso!"
```

Ehi! queste frasi suonano piuttosto familiari! Sono esattamente i nostri record di input originali, inalterati. Pensandoci un po' (il cervello umano è ancora il miglior strumento di debug), ci si rende conto che eravamo fuori di uno!

Usciamo dal debugger:

```
gawk> q
└ Il programma è in esecuzione. Esco comunque (y/n)? y
```

Quindi modifichiamo con un editore di testo:

```
campi_ultima = join(vettore_ultima, contatore_file+1, n)
campi_corrente = join(vettore_corrente, contatore_file+1, m)
```

e il problema è risolto!

## 14.3 I principali comandi di debug

L'insieme dei comandi del debugger di **gawk** può essere diviso nelle seguenti categorie:

- Controllo di punti d'interruzione
- Controllo di esecuzione
- Vedere e modificare dati
- Lavorare con le pile
- Ottenere informazioni
- Comandi vari

Ciascuna di esse è trattata nelle sottosezioni che seguono. Nelle descrizioni seguenti, i comandi che possono essere abbreviati mostrano l'abbreviazione su una seconda riga di

descrizione. Un nome di comando del debugger può essere anche troncato se la parte già scritta non è ambigua. Il debugger ha la capacità predefinita di ripetere automaticamente il precedente comando semplicemente battendo *Invio*. Questo vale per i comandi **list**, **next**, **nexti**, **step**, **stepi** e **continue** quando sono eseguiti senza argomenti.

### 14.3.1 Controllo dei punti d'interruzione

Come abbiamo già visto, la prima cosa che si dovrebbe fare in una sessione di debug è quella di definire dei punti d'interruzione, poiché altrimenti il programma verrà eseguito come se non fosse sotto il debugger. I comandi per controllare i punti d'interruzione sono:

**break** *[[nome-file:]n | funzione] ["espressione"]*

**b** *[[nome-file:]n | funzione] ["espressione"]*

Senza argomenti, imposta un punto d'interruzione alla prossima istruzione da eseguire nello *stack frame* selezionato. Gli argomenti possono essere uno dei seguenti:

*n*                   Imposta un punto d'interruzione alla riga numero *n* nel file sorgente corrente.

*nome-file:n*           Imposta un punto d'interruzione alla riga numero *n* nel file sorgente *nome-file*.

*funzione*           Imposta un punto d'interruzione all'ingresso (la prima istruzione eseguibile) della funzione *funzione*.

A ogni punto d'interruzione è assegnato un numero che può essere usato per cancellarlo dalla lista dei punti d'interruzione usando il comando **delete**.

Specificando un punto d'interruzione, si può fornire anche una condizione. Questa è un'espressione **awk** (racchiusa tra doppi apici) che il debugger valuta ogni volta che viene raggiunto quel punto d'interruzione. Se la condizione è vera, il debugger ferma l'esecuzione e rimane in attesa di un comando. Altrimenti, continua l'esecuzione del programma.

**clear** *[[nome-file:]n | funzione]*

Senza argomenti, cancella ogni eventuale punto d'interruzione all'istruzione successiva da eseguirsi nello *stack frame* selezionato. Se il programma si ferma in un punto d'interruzione, quel punto d'interruzione viene cancellato in modo che il programma non si fermi più in quel punto. Gli argomenti possono essere uno tra i seguenti:

*n*                   Cancella il punto (o i punti) d'interruzione impostato/i alla riga *n* nel file sorgente corrente.

*nome-file:n*           Cancella il punto (o i punti) d'interruzione impostato/i alla riga *n* nel file sorgente *nome-file*.

*funzione*           Cancella il punto (o i punti) d'interruzione impostato/i all'ingresso della funzione *funzione*.

**condition** *n* "*espressione*"

Aggiunge una condizione al punto d'interruzione o al punto d'osservazione esistente *n*. La condizione è un'espressione **awk** *racchiusa tra doppi apici* che il debugger valuta ogni volta che viene raggiunto il punto d'interruzione o il punto d'osservazione. Se la condizione è vera, il debugger ferma l'esecuzione e attende l'immissione di un comando. Altrimenti, il debugger continua l'esecuzione del programma. Se l'espressione della condizione non viene specificata, tutte le condizioni esistenti vengono rimosse (cioè, il punto d'interruzione o di osservazione viene considerato incondizionato).

**delete** [*n1 n2 ...*] [*n-m*]

**d** [*n1 n2 ...*] [*n-m*]

Cancella i punti d'interruzione specificati o un intervallo di punti d'interruzione. Se non vengono forniti argomenti, cancella tutti i punti d'interruzione esistenti.

**disable** [*n1 n2 ...* | *n-m*]

Disabilita punti d'interruzione specificati o un intervallo di essi. Senza argomenti, disabilita tutti i punti d'interruzione.

**enable** [*del* | *once*] [*n1 n2 ...*] [*n-m*]

**e** [*del* | *once*] [*n1 n2 ...*] [*n-m*]

Abilita specifici punti d'interruzione o un intervallo di essi. Senza argomenti, abilita tutti i punti d'interruzione. Opzionalmente, si può specificare come abilitare i punti d'interruzione:

**del**            Abilita dei punti d'interruzione *una tantum*, poi li cancella quando il programma si ferma in quel punto.

**once**          Abilita dei punti d'interruzione *una tantum*, poi li cancella quando il programma si ferma in quel punto.

**ignore** *n contatore*

Ignora il punto d'interruzione numero *n* le successive *contatore* volte in cui viene raggiunto.

**tbreak** [[*nome-file:*]*n* | *funzione*]

**t** [[*nome-file:*]*n* | *funzione*]

Imposta un punto d'interruzione temporaneo (abilitato solo per la prima volta che viene raggiunto). Gli argomenti sono gli stessi di **break**.

### 14.3.2 Controllo di esecuzione

Dopo che i punti d'interruzione sono pronti, si può iniziare l'esecuzione del programma, osservando il suo comportamento. Ci sono più comandi per controllare l'esecuzione del programma di quelli visti nei precedenti esempi:

**commands** [*n*]

**silent**

...

**end**

Imposta una lista di comandi da eseguire subito dopo l'arresto del programma in un punto d'interruzione o di osservazione. *n* è il numero del punto d'interruzione o di osservazione. Se non si specifica un numero, viene usato l'ultimo numero

che è stato specificato. I comandi veri e propri seguono, a cominciare dalla riga successiva, e hanno termine col comando **end**. Se il comando **silent** è nella lista, i consueti messaggi sull'arresto del programma a un punto d'interruzione e la riga sorgente non vengono stampati. Qualsiasi comando nella lista che riprende l'esecuzione (p.es., **continue**) pone fine alla lista (un **end** implicito), e i comandi successivi vengono ignorati. Per esempio:

```
gawk> commands
> silent
> printf "Un punto d'interruzione silenzioso; i = %d\n", i
> info locals
> set i = 10
> continue
> end
gawk>
```

**continue** [*contatore*]

**c** [*contatore*]

Riprende l'esecuzione del programma. Se si riparte da un punto d'interruzione e viene specificato *contatore*, il punto d'interruzione in quella posizione viene ignorato per le prossime *contatore* volte prima di fermarsi nuovamente.

**finish** Esegue fino a quando lo stack frame selezionato completa l'esecuzione. Stampa il valore restituito.

**next** [*contatore*]

**n** [*contatore*]

Continua l'esecuzione alla successiva riga sorgente, saltando le chiamate di funzione. L'argomento *contatore* controlla il numero di ripetizioni dell'azione, come in **step**.

**nexti** [*contatore*]

**ni** [*contatore*]

Esegue una o *contatore* istruzioni, comprese le chiamate di funzione.

**return** [*valore*]

Cancella l'esecuzione di una chiamata di funzione. Se *valore* (una stringa o un numero) viene specificato, è usato come valore di ritorno della funzione. Se usato in un frame diverso da quello più interno (la funzione correntemente in esecuzione; cioè, il frame numero 0), ignora tutti i frame più interni di quello selezionato, e il chiamante del frame selezionato diventa il frame più interno.

**run**

**r**

Avvia/riavvia l'esecuzione del programma. Quando il programma viene riavviato, il debugger mantiene i punti d'interruzione e di osservazione, la cronologia dei comandi, la visualizzazione automatica di variabili, e le opzioni del debugger.

**step** [*contatore*]

**s** [*contatore*]

Continua l'esecuzione finché il controllo non raggiunge una diversa riga del sorgente nello *stack frame* corrente, eseguendo ogni funzione chiamata all'interno

della riga. Se viene fornito l'argomento *contatore*, esegue il numero di istruzioni specificate prima di fermarsi, a meno che non s'imbatta in un punto d'interruzione o di osservazione.

```
stepi [contatore]
si [contatore]
```

Esegue una o *contatore* istruzioni, comprese le chiamate di funzione. (Per una spiegazione su cosa s'intende per "istruzione" in **gawk**, si veda l'output mostrato sotto **dump** nella [Sezione 14.3.6 \[Comandi vari del debugger\]](#), pagina 374.)

```
until [[nome-file:]n | funzione]
u [[nome-file:]n | funzione]
```

Senza argomenti, prosegue l'esecuzione finché non viene raggiunta una riga dopo la riga corrente nello *stack frame* corrente. Se viene specificato un argomento, prosegue l'esecuzione finché non viene raggiunto il punto specificato, o lo *stack frame* corrente non termina l'esecuzione.

### 14.3.3 Vedere e modificare dati

I comandi per vedere e modificare variabili all'interno di **gawk** sono:

```
display [var | $n]
```

Aggiunge la variabile *var* (o il campo *\$n*) alla lista di visualizzazione. Il valore della variabile o del campo è visualizzato ogni volta che il programma s'interrompe. Ogni variabile aggiunta alla lista è identificata da un numero univoco:

```
gawk> display x
- 10: x = 1
```

La riga qui sopra mostra il numero di elemento assegnato, il nome della variabile e il suo valore corrente. Se la variabile di *display* fa riferimento a un parametro di funzione, è cancellata silenziosamente dalla lista non appena l'esecuzione raggiunge un contesto dove la variabile con quel nome non esiste più. Senza argomenti, **display** mostra i valori correnti degli elementi della lista.

```
eval "istruzioni awk"
```

Valuta *istruzioni awk* nel contesto del programma in esecuzione. Si può fare qualsiasi cosa che un programma **awk** farebbe: assegnare valori a variabili, chiamare funzioni, e così via.

```
eval param, ...
istruzioni awk
```

**end** Questa forma di **eval** è simile alla precedente, solo che permette di definire "variabili locali" che esistono nel contesto delle *istruzioni awk*, invece di usare variabili o parametri di funzione già definiti nel programma.

```
print var1[, var2 ...]
p var1[, var2 ...]
```

Stampa i valori di una o più variabili o campi di **gawk**. I campi devono essere indicizzati usando delle costanti:

```
gawk> print $3
```

Questo stampa il terzo campo del record di input (se il campo specificato non esiste, stampa il ‘campo nullo’). Una variabile può essere un elemento di un vettore, avente come indice una stringa di valore costante. Per stampare il contenuto di un vettore, si deve anteporre il simbolo ‘@’ al nome del vettore:

```
gawk> print @a
```

L'esempio stampa gli indici e i corrispondenti valori di tutti gli elementi del vettore `a`.

```
printf formato [, arg ...]
```

Stampa un testo formattato. Il *formato* può includere sequenze di protezione, come ‘\n’ (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50). Non viene stampato nessun ritorno a capo che non sia stato specificato esplicitamente.

```
set var=valore
```

Assegna un valore costante (numero o stringa) a una variabile o a un campo di `awk`. I valori di stringa devono essere racchiusi tra doppi apici ("...").

Si possono impostare anche delle variabili speciali di `awk`, come `FS`, `NF`, `NR`, e così via.

```
watch var | $n ["espressione"]
```

```
w var | $n ["espressione"]
```

Aggiunge la variabile `var` (o il campo `$n`) alla lista dei punti d'osservazione. Il debugger quindi interrompe il programma ogni volta che il valore della variabile o del campo cambia. A ogni elemento osservato viene assegnato un numero che può essere usato per cancellarlo dalla lista usando il comando `unwatch` [non-osservare più].

Definendo un punto d'osservazione, si può anche porre una condizione, che è un'espressione `awk` (racchiusa tra doppi apici) che il debugger valuta ogni volta che viene raggiunto il punto d'osservazione. Se la condizione è vera, il debugger interrompe l'esecuzione e rimane in attesa di un comando. Altrimenti, `gawk` prosegue nell'esecuzione del programma.

```
undisplay [n]
```

Rimuove l'elemento numero `n` (o tutti gli elementi, se non vi sono argomenti) dalla lista delle visualizzazioni automatiche.

```
unwatch [n]
```

Rimuove l'elemento numero `n` (o tutti gli elementi, se non vi sono argomenti) dalla lista dei punti d'osservazione.

### 14.3.4 Lavorare con lo stack

Ogni volta che si esegue un programma che contiene chiamate di funzione, `gawk` mantiene una pila contenente la lista delle chiamate di funzione che hanno portato al punto in cui il programma si trova in ogni momento. È possibile vedere a che punto si trova il programma, e anche muoversi all'interno della pila per vedere qual era lo stato delle cose nelle funzioni che hanno chiamato quella in cui ci si trova. I comandi per far questo sono:

**backtrace** [*contatore*]

**bt** [*contatore*]

**where** [*contatore*]

Stampa a ritroso una traccia di tutte le chiamate di funzione (stack frame), o i dei *contatore* frame più interni se *contatore* > 0. Stampa i *contatore* frame più esterni se *contatore* < 0. La tracciatura a ritroso mostra il nome e gli argomenti di ciascuna funzione, il sorgente nome-file, e il numero di riga. L'alias **where** per **backtrace** viene mantenuto per i vecchi utenti di GDB che potrebbero essere abituati a quel comando.

**down** [*contatore*]

Sposta *contatore* (default 1) frame sotto la pila verso il frame più interno. Poi seleziona e stampa il frame.

**frame** [*n*]

**f** [*n*]

Seleziona e stampa lo *stack frame* *n*. Il frame 0 è quello correntemente in esecuzione, o il frame *più interno*, (chiamata di funzione); il frame 1 è il frame che ha chiamato quello più interno. Il frame col numero più alto è quello per il programma principale. Le informazioni stampate comprendono il numero di frame, i nomi delle funzioni e degli argomenti, i file sorgenti e le righe sorgenti.

**up** [*contatore*]

Sposta *contatore* (default 1) frame sopra la pila verso il frame più esterno. Poi seleziona e stampa il frame.

### 14.3.5 Ottenere informazioni sullo stato del programma e del debugger

Oltre che vedere i valori delle variabili, spesso si ha necessità di ottenere informazioni di altro tipo sullo stato del programma e dello stesso ambiente di debug. Il debugger di **gawk** ha un comando che fornisce quest'informazione, chiamato convenientemente **info**. **info** è usato con uno dei tanti argomenti che dicono esattamente quel che si vuol sapere:

**info** *cosa*

**i** *cosa* Il valore di *cosa* dovrebbe essere uno dei seguenti:

**args** Elenca gli argomenti del frame selezionato.

**break** Elenca tutti i punti d'interruzione attualmente impostati.

**display** Elenca tutti gli elementi della lista delle visualizzazioni automatiche.

**frame** Dà una descrizione degli *stack frame* selezionati.

**functions**

Elenca tutte le definizioni delle funzioni compresi i nomi-file e i numeri di riga.

**locals** Elenca le variabili locali dei frame selezionati.

**source** Stampa il nome del file sorgente corrente. Ogni volta che il programma si interrompe, il file sorgente corrente è il file che contiene

l'istruzione corrente. Quando il debugger viene avviato per la prima volta, il file sorgente corrente è il primo file incluso attraverso l'opzione `-f`. Il comando `'list nome-file:numero-riga'` può essere usato in qualsiasi momento per cambiare il sorgente corrente.

**sources** Elenca tutti i sorgenti del programma.

**variables**  
Elenca tutte le variabili locali.

**watch** Elenca tutti gli elementi della lista dei punti d'osservazione.

Ulteriori comandi permettono di avere il controllo sul debugger, la capacità di salvare lo stato del debugger e la capacità di eseguire comandi del debugger da un file. I comandi sono:

**option** [*nome*[=*valore*]]

o [*nome*[=*valore*]]

Senza argomenti, visualizza le opzioni del debugger disponibili e i loro valori correnti. `'option nome'` mostra il valore corrente dell'opzione così denominata. `'option nome=valore'` assegna un nuovo valore all'opzione. Le opzioni disponibili sono:

**history\_size**

Imposta il numero massimo di righe da mantenere nel file della cronologia `./gawk_history`. Il valore di default è 100.

**listsize** Specifica il numero di righe che `list` deve stampare. Il valore di default è 15.

**outfile** Invia l'output di `gawk` in un file; l'output del debugger è visualizzato comunque anche nello standard output. Assegnare come valore stringa vuota (`""`) reimposta l'output solo allo standard output.

**prompt** Cambia la riga per l'immissione dei comandi del debugger. Il valore di default è `'gawk> '`.

**save\_history** [*on* | *off*]

Salva la cronologia dei comandi nel file `./gawk_history`. L'impostazione di default è *on*.

**save\_options** [*on* | *off*]

Salva le opzioni correnti nel file `./gawkrc` all'uscita. L'impostazione di default è *on*. Le opzioni sono lette di nuovo all'avvio della sessione successiva.

**trace** [*on* | *off*]

Attiva o disattiva il tracciamento delle istruzioni. L'impostazione di default è *off*.

**save nome-file**

Salva i comandi eseguiti nella sessione corrente nel nome-file indicato, in modo da poterli ripetere in seguito usando il comando `source`.

`source nome-file`

Esegue comandi contenuti in un file; un errore in un comando non impedisce l'esecuzione dei comandi successivi. In un file di comandi sono consentiti i commenti (righe che iniziano con '#'). Le righe vuote vengono ignorate; esse *non* ripetono l'ultimo comando. Non si può riavviare il programma mettendo più di un comando `run` nel file. Inoltre, la lista dei comandi può includere altri comandi `source`; in ogni caso, il debugger di `gawk` non richiama lo stesso file più di una volta per evitare ricorsioni infinite.

Oltre al comando `source`, o al posto di esso, si possono usare le opzioni sulla riga di comando `-D file` o `--debug=file` per eseguire comandi da un file in maniera non interattiva (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), [pagina 33](#)).

### 14.3.6 Comandi vari del debugger

Ci sono alcuni altri comandi che non rientrano nelle precedenti categorie, come i seguenti:

`dump [nome-file]`

Riversa il *byte code* del programma nello standard output o nel file definito in *nome-file*. Questo stampa una rappresentazione delle istruzioni interne che `gawk` esegue per implementare i comandi `awk` in un programma. Ciò può essere molto istruttivo, come dimostra il seguente riversamento parziale del codice offuscato di Davide Brini (si veda la [Sezione 11.3.11 \[E ora per qualcosa di completamente differente\]](#), [pagina 326](#)):

```
gawk> dump
+          # BEGIN
+
+ [ 1:0xfcd340] Op_rule      : [in_rule = BEGIN] [source_file = brini.awk]
+ [ 1:0xfcc240] Op_push_i   : "" [MALLOC|STRING|STRCUR]
+ [ 1:0xfcc2a0] Op_push_i   : "" [MALLOC|STRING|STRCUR]
+ [ 1:0xfcc280] Op_match    :
+ [ 1:0xfcc1e0] Op_store_var : 0
+ [ 1:0xfcc2e0] Op_push_i   : "" [MALLOC|STRING|STRCUR]
+ [ 1:0xfcc340] Op_push_i   : "" [MALLOC|STRING|STRCUR]
+ [ 1:0xfcc320] Op_equal    :
+ [ 1:0xfcc200] Op_store_var : o
+ [ 1:0xfcc380] Op_push     : o
+ [ 1:0xfcc360] Op_plus_i   : 0 [MALLOC|NUMCUR|NUMBER]
+ [ 1:0xfcc220] Op_push_lhs : o [do_reference = true]
+ [ 1:0xfcc300] Op_assign_plus :
+ [  :0xfcc2c0] Op_pop      :
+ [ 1:0xfcc400] Op_push     : 0
+ [ 1:0xfcc420] Op_push_i   : "" [MALLOC|STRING|STRCUR]
+ [  :0xfcc4a0] Op_no_op    :
+ [ 1:0xfcc480] Op_push     : 0
+ [  :0xfcc4c0] Op_concat   : [expr_count = 3] [concat_flag = 0]
+ [ 1:0xfcc3c0] Op_store_var : x
+ [ 1:0xfcc440] Op_push_lhs : X [do_reference = true]
+ [ 1:0xfcc3a0] Op_postincrement :
+ [ 1:0xfcc4e0] Op_push     : x
+ [ 1:0xfcc540] Op_push     : o
+ [ 1:0xfcc500] Op_plus     :
+ [ 1:0xfcc580] Op_push     : o
+ [ 1:0xfcc560] Op_plus     :
```

```

+ [ 1:0xfcc460] Op_leq          :
+ [   :0xfcc5c0] Op_jump_false : [target_jump = 0xfcc5e0]
+ [ 1:0xfcc600] Op_push_i      : "%c" [MALLOC|STRING|STRCUR]
+ [   :0xfcc660] Op_no_op       :
+ [ 1:0xfcc520] Op_assign_concat : c
+ [   :0xfcc620] Op_jump        : [target_jump = 0xfcc440]
+
...
+
+ [      2:0xfcc5a0] Op_K_printf      : [expr_count = 17] [redir_type = ""]
+ [      :0xfcc140] Op_no_op          :
+ [      :0xfcc1c0] Op_atexit         :
+ [      :0xfcc640] Op_stop           :
+ [      :0xfcc180] Op_no_op          :
+ [      :0xfcd150] Op_after_beginfile :
+ [      :0xfcc160] Op_no_op          :
+ [      :0xfcc1a0] Op_after_endfile  :
gawk>

```

**exit** Esce dal debugger. Si veda la voce ‘quit’, più avanti in quest’elenco.

**help**

**h** Stampa una lista di tutti i comandi del debugger di **gawk** con un breve sommario su come usarli. ‘**help comando**’ stampa l’informazione sul comando *comando*.

**list** [- | + | *n* | *nome-file:n* | *n-m* | *funzione*]

**l** [- | + | *n* | *nome-file:n* | *n-m* | *funzione*]

Stampa le righe specificate (per default 15) dal file sorgente corrente o il file chiamato *nome-file*. I possibili argomenti di **list** sono i seguenti:

- (Meno) Stampa righe prima delle ultime righe stampate.

+

Stampa righe dopo le ultime righe stampate. **list** senza argomenti fa la stessa cosa.

*n* Stampa righe centrate attorno alla riga numero *n*.

*n-m* Stampa righe dalla numero *n* alla numero *m*.

*nome-file:n*

Stampa righe centrate attorno alla riga numero *n* nel file sorgente *nome-file*. Questo comando può cambiare il file sorgente corrente.

*funzione* Stampa righe centrate attorno all’inizio della funzione *function*. Questo comando può cambiare il file sorgente corrente.

**quit**

**q** Esce dal debugger. Fare il debug è divertente, ma noi tutti a volte dobbiamo far fronte ad altri impegni nella vita, e talvolta troviamo il bug e possiamo tranquillamente passare a quello successivo! Come abbiamo visto prima, se si sta eseguendo un programma, il debugger avverte quando si batte ‘q’ o ‘quit’, in modo da essere sicuri di voler realmente abbandonare il debug.

**trace** [on | off]

Abilita o disabilita la stampa continua delle istruzioni che si stanno per eseguire, assieme alle righe di **awk** che implementano. L’impostazione di default è **off**.

È auspicabile che la maggior parte dei “codici operativi” (o “opcode”) in queste istruzioni siano sufficientemente autoesplicativi, e l’uso di `stepi` e `nexti` mentre `trace` è abilitato li renderà familiari.

## 14.4 Supporto per Readline

Se `gawk` è compilato con la libreria **GNU Readline**, ci si può avvantaggiare delle sue funzionalità riguardanti il completamento dei comandi della libreria e l’espansione della cronologia. Sono disponibili i seguenti tipi di completamento:

Completamento dei comandi

Nomi dei comandi.

Completamento del nome-file del sorgente

Nomi-file dei sorgenti. I relativi comandi sono `break`, `clear`, `list`, `tbreak` e `until`.

Completamento di argomento

Argomenti di un comando non numerici. I relativi comandi sono `enable` e `info`.

Completamento del nome di variabile

Interessa i nomi delle variabili globali, e gli argomenti di funzione nel contesto corrente se il programma è in esecuzione. I relativi comandi sono `display`, `print`, `set` e `watch`.

## 14.5 Limitazioni

Si spera che il lettore trovi il debugger di `gawk` utile e piacevole da usare, ma come accade per ogni programma, specialmente nelle sue prime versioni, ha ancora delle limitazioni. Quelle di cui è bene essere al corrente sono:

- Nella versione presente, il debugger non dà una spiegazione dettagliata dell’errore che si è commesso quando si immette qualcosa che il debugger ritiene sbagliato. La risposta invece è solamente ‘`syntax error`’. Quando si arriva a capire l’errore commesso, tuttavia, ci si sentirà come un vero guru.
- Se si studiano i “dump” dei codici operativi nella **Sezione 14.3.6 [Comandi vari del debugger]**, **pagina 374**, (o se si ha già familiarità con i comandi interni di `gawk`), ci si renderà conto che gran parte della manipolazione interna di dati in `gawk`, così come in molti interpreti, è fatta su di una pila. `Op_push`, `Op_pop`, e simili sono il pane quotidiano di gran parte del codice di `gawk`.

Sfortunatamente, al momento, il debugger di `gawk` non consente di esaminare i contenuti della pila. Cioè, i risultati intermedi della valutazione delle espressioni sono sulla pila, ma non è possibile stamparli. Invece, possono essere stampate solo quelle variabili che sono state definite nel programma. Naturalmente, un espediente per cercare di rimediare è di usare più variabili esplicite in fase di debug e poi cambiarle di nuovo per ottenere un codice forse più difficile da comprendere, ma più ottimizzato.

- Non c’è alcun modo per guardare “dentro” al processo della compilazione delle espressioni regolari per vedere se corrispondono a quel che si intendeva. Come programmatore di `awk`, ci si aspetta che chi legge conosca il significato di `/[^[[:alnum:]][:blank:]]/`.

- Il debugger di **gawk** è progettato per essere usato eseguendo un programma (con tutti i suoi parametri) dalla riga di comando, come descritto nella [Sezione 14.2.1 \[Come avviare il debugger\]](#), [pagina 363](#). Non c'è alcun modo (al momento) di modificare o di “entrare dentro” l'esecuzione di un programma. Questo sembra ragionevole per un linguaggio che è usato principalmente per eseguire programmi piccoli e che non richiedono molto tempo di esecuzione.
- Il debugger di **gawk** accetta solo codice sorgente fornito con l'opzione **-f**.

## 14.6 Sommario

- Raramente i programmi funzionano bene al primo colpo. Trovare gli errori che contengono viene chiamato *debugging*, e un programma che aiuta a trovarli è un *debugger*. **gawk** ha un debugger incorporato che funziona in modo molto simile al debugger GNU, GDB.
- I debugger possono eseguire il programma un'istruzione per volta, esaminare e cambiare i valori delle variabili e dei vettori, e fanno tante altre cose per permettere di comprendere cosa sta facendo effettivamente il programma in un dato momento (a differenza del comportamento atteso).
- Come la maggior parte dei debugger, il debugger di **gawk** funziona in termini di stack frame, e si possono inserire sia punti d'interruzione (interruzioni a un certo punto del codice) sia punti d'osservazione (interruzioni quando il valore di un dato cambia).
- La serie di comandi del debugger è abbastanza completa, e permette di monitorare i punti d'interruzione, l'esecuzione, la visualizzazione e la modifica dei dati, di lavorare con le pile, ottenere informazioni, e di svolgere altri compiti.
- Se la libreria GNU Readline è disponibile al momento della compilazione di **gawk**, viene usata dal debugger per fornire la cronologia della riga di comando e delle modifiche apportate durante il debug.
- Normalmente, il debugger non influenza il programma che sta controllando, ma questo può succedere occasionalmente.



## 15 Calcolo con precisione arbitraria con gawk

In questo capitolo si introducono alcuni concetti base su come i computer eseguono i calcoli e si definiscono alcuni termini importanti. Si continua poi descrivendo il calcolo in virgola mobile, che è quello che **awk** usa per tutte le sue operazioni aritmetiche, e si prosegue con una trattazione del calcolo in virgola mobile con precisione arbitraria, una funzionalità disponibile solo in **gawk**. Si passa poi a illustrare i numeri interi a precisione arbitraria e si conclude con una descrizione di alcuni punti sui quali **gawk** e lo standard POSIX non sono esattamente in accordo.

**NOTA:** La maggior parte degli utenti di **gawk** può saltare senza patemi d'animo questo capitolo. Tuttavia, se si vogliono eseguire calcoli scientifici con **gawk**, questo è il luogo adatto per imparare a farlo.

### 15.1 Una descrizione generale dell'aritmetica del computer

Sinora, abbiamo avuto a che fare con dati come numeri o stringhe. Ultimamente, comunque, i computer rappresentano ogni cosa in termini di *cifre binarie*, o *bit*. Una cifra decimale può assumere uno di 10 valori: da zero a nove. Una cifra binaria può assumere uno di due valori: zero o uno. Usando il sistema binario, i computer (e i programmi per computer) possono rappresentare e manipolare dati numerici e dati costituiti da caratteri. In generale, tanti più bit è possibile usare per rappresentare una determinata cosa, tanto maggiore sarà l'intervallo dei possibili valori che essa potrà assumere.

I moderni calcolatori possono eseguire calcoli numerici in almeno due modi, e spesso anche di più. Ogni tipo di calcolo usa una diversa rappresentazione (organizzazione dei bit) dei numeri. Le modalità di calcolo che ci interessano sono:

#### Calcolo decimale

Questo è il tipo di calcolo che s'impara alle scuole elementari, usando carta e penna (o anche una calcolatrice). In teoria, i numeri possono avere un numero arbitrario di cifre su ambo i lati del separatore decimale, e il risultato di un'operazione è sempre esatto.

Alcuni sistemi moderni possono eseguire calcoli decimali direttamente, tramite apposite istruzioni disponibili nell'hardware dell'elaboratore, ma normalmente si ha necessità di una speciale libreria software che consenta di effettuare le operazioni desiderate. Ci sono anche librerie che svolgono i calcoli decimali interamente per via software.

Anche se alcuni utenti si aspettano che **gawk** effettui delle operazioni usando numeri in base decimale,<sup>1</sup> non è questo quello che succede.

#### La matematica coi numeri interi

A scuola ci hanno insegnato che i valori interi erano quei numeri privi di una parte frazionaria, come 1, 42, o -17. Il vantaggio dei numeri interi è che essi rappresentano dei valori in maniera esatta. Lo svantaggio è che i numeri rappresentabili sono limitati.

---

<sup>1</sup> Non sappiamo perché se lo aspettino, ma è così.

Nei calcolatori, i valori interi sono di due tipi: *con segno* e *senza segno*. I valori con segno possono essere negativi o positivi, mentre i valori senza segno sono sempre maggiori o uguali a zero.

Nei sistemi informatici, il calcolo con valori interi è esatto, ma il possibile campo di variazione dei valori è limitato. L'elaborazione con numeri interi è più veloce di quella con numeri a virgola mobile.

La matematica coi numeri a virgola mobile

I numeri a virgola mobile rappresentano quelli che a scuola sono chiamati numeri “reali” (cioè, quelli che hanno una parte frazionaria, come 3.1415927). Il vantaggio dei numeri a virgola mobile è che essi possono rappresentare uno spettro di valori molto più ampio di quello rappresentato dai numeri interi. Lo svantaggio è che ci sono numeri che essi non possono rappresentare in modo esatto.

I computer moderni possono eseguire calcoli su valori a virgola mobile nell'hardware dell'elaboratore, entro un intervallo di valori limitato. Ci sono inoltre librerie di programmi che consentono calcoli, usando numeri a virgola mobile, di precisione arbitraria.

POSIX **awk** usa numeri a virgola mobile a *doppia precisione*, che possono gestire più cifre rispetto ai numeri a virgola mobile a *singola precisione*. **gawk** ha inoltre funzionalità, descritte in dettaglio più sotto, che lo mettono in grado di eseguire calcoli con i numeri a virgola mobile con precisione arbitraria.

I calcolatori operano con valori interi e a virgola mobile su diversi intervalli. I valori interi normalmente hanno una dimensione di 32 bit o 64 bit. I valori a virgola mobile a singola precisione occupano 32 bit, mentre i valori a virgola mobile a doppia precisione occupano 64 bit. I valori a virgola mobile sono sempre con segno. Il possibile campo di variazione dei valori è mostrato in [Tabella 15.1](#).

Rappresentazione numerica	Valore minimo	Valore massimo
Interi con segno a 32-bit	-2.147.483.648	2.147.483.647
Interi senza segno a 32-bit	0	4.294.967.295
Interi con segno a 64-bit	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
Interi senza segno a 64-bit	0	18.446.744.073.709.551.615
Virgola mobile, singola precisione (circa)	$1,175494^{-38}$	$3,402823^{38}$
Virgola mobile, doppia precisione (circa)	$2,225074^{-308}$	$1,797693^{308}$

Tabella 15.1: Intervalli dei valori per diverse rappresentazioni numeriche

## 15.2 Altre cose da sapere

Il resto di questo capitolo usa un certo numero di termini. Di seguito vengono date alcune definizioni informali che dovrebbero essere utili per la lettura di questo documento:

#### Accuratezza

L'accuratezza del calcolo sui numeri a virgola mobile indica di quanto si avvicina il calcolo al valore reale (calcolato con carta e penna).

**Errore** La differenza tra quello che il risultato di un calcolo “dovrebbe dare” e quello che effettivamente dà. È meglio minimizzare l'errore quanto più possibile.

**Esponente** L'ordine di grandezza di un valore; alcuni bit in un valore a virgola mobile contengono l'esponente.

**Inf** Un valore speciale che rappresenta l'infinito. Le operazioni tra un qualsiasi numero e l'infinito danno infinito.

**Mantissa** Un valore a virgola mobile è formato dalla mantissa moltiplicata per 10 alla potenza dell'esponente. Per esempio, in `1,2345e67`, la mantissa è `1,2345`.

#### Modalità di arrotondamento

Come i numeri vanno arrotondati, per eccesso o per difetto, quando necessario. Maggiori dettagli verranno forniti in seguito.

**NaN** “Not a number” (Non un Numero).<sup>2</sup> Un valore speciale che risulta da un calcolo che non ha risposta come numero reale. In tal caso, i programmi possono o ricevere un'eccezione di virgola mobile, o restituire NaN come risultato. Lo standard IEEE 754 consiglia che i sistemi restituiscano NaN. Alcuni esempi:

`sqrt(-1)` La radice quadrata di  $-1$  ha senso nell'insieme dei numeri complessi, ma non nell'insieme dei numeri reali, per cui il risultato è NaN.

`log(-8)` Il logaritmo di  $-8$  è fuori dal dominio di `log()`, per cui il risultato è NaN.

#### Normalizzato (formato)

Come la mantissa (vedi oltre in questa lista) è usualmente memorizzata. Il valore viene aggiustato in modo che il primo bit sia sempre uno, e in questo modo l'uno iniziale è supposto presente (per come viene generato il numero), ma non è memorizzato fisicamente. Questo fornisce un bit di precisione in più.

**Precisione** Il numero di bit usati per rappresentare un numero a virgola mobile. Più sono i bit, e maggiore è l'intervallo di cifre che si possono rappresentare. Le precisioni binaria e decimale sono legate in modo approssimativo, secondo la formula:

$$prec = 3.322 \cdot dps$$

Qui, *prec* indica la precisione binaria (misurata in bit) e *dps* (abbreviazione di "decimal places") indica le cifre decimali.

**Stabilità** Dall'[articolo di Wikipedia sulla stabilità numerica](#): “I calcoli per i quali si può dimostrare che non amplificano gli errori di approssimazione sono chiamati *numericamente stabili*.”

<sup>2</sup> Grazie a Michael Brennan per questa descrizione, che abbiamo parafrasato, e per gli esempi.

Si veda [l'articolo di Wikipedia su accuratezza e precisione](#) per maggiori informazioni su questi due termini.

Sui computer moderni, l'unità di calcolo in virgola mobile usa la rappresentazione e le operazioni definite dallo standard IEEE 754. Tre dei tipi definiti nello standard IEEE 754 sono: 32-bit singola precisione, 64-bit doppia precisione e 128-bit quadrupla precisione. Lo standard specifica anche formati a precisione estesa per consentire una maggiore precisione e campi di variazione degli esponenti più ampi. (`awk` usa solo il formato a 64-bit doppia precisione.)

**Tabella 15.2** elenca la precisione e i valori di campo dell'esponente per i principali formati binari IEEE 754.

Nome	Bit totali	Precisione	Esponente minimo	Esponente massimo
Singola	32	24	−126	+127
Doppia	64	53	−1022	+1023
Quadrupla	128	113	−16382	+16383

Tabella 15.2: Valori per i principali formati IEEE

**NOTA:** I numeri che descrivono la precisione includono la cifra 1 iniziale implicita, il che equivale ad avere un bit in più nella mantissa.

### 15.3 Funzionalità per il calcolo a precisione arbitraria in `gawk`

Per default, `gawk` usa i valori in virgola mobile a doppia precisione disponibili nell'hardware del sistema su cui viene eseguito. Tuttavia, se è stato compilato in modo da includere questa funzionalità ed è stata specificata l'opzione da riga di comando `-M`, `gawk` usa le librerie **GNU MPFR** e **GNU MP** (GMP) per effettuare calcoli sui numeri con una precisione arbitraria. Si può verificare se il supporto a MPFR è disponibile in questo modo:

```
$ gawk --version
+ GNU Awk 4.1.2, API: 1.1 (GNU MPFR 3.1.0-p3, GNU MP 5.0.2)
+ Copyright (C) 1989, 1991-2015 Free Software Foundation.
...
```

(I numeri di versione visualizzati possono essere diversi. Non importa; l'importante è che siano presenti GNU MPFR e GNU MP nel testo restituito.)

Inoltre, ci sono alcuni elementi disponibili nel vettore `PROCINFO` per fornire informazioni sulle librerie MPFR e GMP (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui `awk` fornisce informazioni\]](#), pagina 165).

La libreria MPFR dà un controllo accurato sulle precisioni e sulle modalità di arrotondamento, e dà risultati correttamente arrotondati, riproducibili e indipendenti dalla piattaforma. Con l'opzione da riga di comando `-M`, tutti gli operatori aritmetici e le funzioni in virgola mobile possono produrre risultati a ogni livello di precisione supportato da MPFR.

Due variabili predefinite, `PREC` e `ROUNDMODE`, danno il controllo sulla precisione di elaborazione e sulla modalità di arrotondamento. La precisione e la modalità di arrotondamento sono impostate a livello globale per ogni operazione da eseguire. Si veda la [Sezione 15.4.4](#)

[Impostare la precisione], pagina 386, e la Sezione 15.4.5 [Impostare la modalità di arrotondamento], pagina 387, per maggiori informazioni.

## 15.4 Calcolo in virgola mobile: *Caveat Emptor!*

*L'ora di matematica è ostica!*

—Teen Talk Barbie, luglio 1992

Questa sezione fornisce un quadro dettagliato dei problemi che si presentano quando si eseguono molti calcoli in virgola mobile.<sup>3</sup> Le spiegazioni fornite valgono sia per il calcolo in virgola mobile effettuato direttamente dall'hardware del computer, sia per quello ottenuto tramite il software per la precisione arbitraria.

**ATTENZIONE:** Le informazioni fornite in questa sede sono deliberatamente di tipo generale. Se si devono eseguire calcoli complessi col computer, si dovrebbero prima ottenere ulteriori informazioni, e non basarsi solo su quanto qui detto.

### 15.4.1 La matematica in virgola mobile non è esatta

Le rappresentazioni e i calcoli con numeri a virgola mobile binari sono inesatti. Semplici valori come 0,1 non possono essere rappresentati in modo preciso usando numeri a virgola mobile binari, e la limitata precisione dei numeri a virgola mobile significa che piccoli cambiamenti nell'ordine delle operazioni o la precisione di memorizzazione di operazioni intermedie può cambiare il risultato. Per rendere la situazione più difficile, nel calcolo in virgola mobile con precisione arbitraria, si può impostare la precisione prima di eseguire un calcolo, però non si può sapere con certezza quale sarà il numero di cifre decimali esatte nel risultato finale.

#### 15.4.1.1 Molti numeri non possono essere rappresentati esattamente

Perciò, prima di iniziare a scrivere del codice, si dovrebbe pensare al risultato che si vuole effettivamente ottenere e a cosa realmente accade. Si considerino i due numeri nel seguente esempio:

```
x = 0.875          # 1/2 + 1/4 + 1/8
y = 0.425
```

Diversamente dal numero in `y`, il numero memorizzato in `x` è rappresentabile esattamente nel formato binario, perché può essere scritto come somma finita di una o più frazioni i cui denominatori sono tutti multipli di due. Quando `gawk` legge un numero a virgola mobile dal sorgente di un programma, arrotonda automaticamente quel numero alla precisione, quale che sia, supportata dal computer in uso. Se si tenta di stampare il contenuto numerico di una variabile usando una stringa di formato in uscita di `"%.17g"`, il valore restituito può non essere lo stesso numero assegnato a quella variabile:

```
$ gawk 'BEGIN { x = 0.875; y = 0.425
>               printf("%.17g", x, y) }'
+ 0.875, 0.42499999999999999
```

<sup>3</sup> C'è un saggio molto bello *sul calcolo in virgola mobile* di David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys* **23**, 1 (1991-03): 5-48. Vale la pena di leggerlo, se si è interessati a scendere nei dettagli, però richiede delle conoscenze informatiche.

Spesso l'errore è talmente piccolo da non essere neppure notato, e se è stato notato, si può sempre specificare il grado di precisione si vuole nell'output. In genere questo è una stringa di formato come "%.15g" che, se usata nell'esempio precedente, dà luogo a un output identico all'input.

#### 15.4.1.2 Fare attenzione quando si confrontano valori

Poiché la rappresentazione interna del computer può discostarsi, sia pur di poco, dal valore esatto, confrontare dei valori a virgola mobile per vedere se sono esattamente uguali è generalmente una pessima idea. Questo è un esempio in cui tale confronto non funziona come dovrebbe:

```
$ gawk 'BEGIN { print (0.1 + 12.2 == 12.3) }'
+ 0
```

Il metodo generalmente seguito per confrontare valori a virgola mobile consiste nel controllare se la differenza tra loro è minore di un certo valore (chiamato *delta*, o *tolleranza*). Quel che si deve decidere è qual è il valore minimo di delta adeguato. Il codice per far ciò è qualcosa del genere:

```
delta = 0.00001          # per esempio
differenza = abs(a) - abs(b) # sottrazione dei due valori
if (differenza < delta)
    # va bene
else
    # non va bene
```

(Si presuppone che sia stata definita in qualche parte del programma una semplice funzione che restituisce il valore assoluto di un numero, chiamata *abs()*.)

#### 15.4.1.3 Gli errori diventano sempre maggiori

La perdita di accuratezza in un singolo calcolo con numeri a virgola mobile generalmente non dovrebbe destare preoccupazione. Tuttavia, se si calcola un valore che è una sequenza di operazioni in virgola mobile, l'errore si può accumulare e influire sensibilmente sul risultato del calcolo stesso. Qui sotto vediamo un tentativo di calcolare il valore di  $\pi$  usando una delle sue rappresentazioni come somma di una serie di numeri:

```
BEGIN {
    x = 1.0 / sqrt(3.0)
    n = 6
    for (i = 1; i < 30; i++) {
        n = n * 2.0
        x = (sqrt(x * x + 1) - 1) / x
        printf("%.15f\n", n * x)
    }
}
```

Quando viene eseguito, gli errori iniziali si propagano nei calcoli successivi, facendo terminare il ciclo prematuramente dopo un tentativo di divisione per zero:

```
$ gawk -f pi.awk
+ 3.215390309173475
+ 3.159659942097510
```

```

+ 3.146086215131467
+ 3.142714599645573
...
+ 3.224515243534819
+ 2.791117213058638
+ 0.0000000000000000
[error] gawk: pi.awk:6: fatale: tentativo di dividere per zero

```

Ecco un altro esempio in cui l'inaccuratezza nelle rappresentazioni interne porta a un risultato inatteso:

```

$ gawk 'BEGIN {
>   for (d = 1.1; d <= 1.5; d += 0.1) # esegue il ciclo cinque volte (?)
>       i++
>   print i
> }'
+ 4

```

### 15.4.2 Ottenere la precisione voluta

Può il calcolo con precisione arbitraria dare risultati esatti? Non ci sono risposte facili. Le regole standard dell'algebra spesso non valgono nei calcoli con precisione arbitraria. Tra le altre cose, le leggi distributiva e associativa non sono rispettate completamente, e l'ordine dell'operazione può essere importante per il calcolo. Errori di arrotondamento, perdite di precisione che si accumulano, e valori molto vicini allo zero sono spesso causa di problemi.

Quando **gawk** verifica l'eguaglianza delle espressioni `'0.1 + 12.2'` e `'12.3'` usando l'aritmetica a doppia precisione della macchina, decide che non sono uguali! (Si veda la [Sezione 15.4.1.2 \[Fare attenzione quando si confrontano valori\], pagina 384.](#)) Si può ottenere il risultato cercato aumentando la precisione; 56 bit in questo caso sono sufficienti:

```

$ gawk -M -v PREC=56 'BEGIN { print (0.1 + 12.2 == 12.3) }'
+ 1

```

Se aggiungere più bit è una buona cosa, aggiungerne ancora di più è meglio? Ecco cosa succede se si usa un valore di `PREC` ancora più alto:

```

$ gawk -M -v PREC=201 'BEGIN { print (0.1 + 12.2 == 12.3) }'
+ 0

```

Non è un bug di **gawk** o della libreria MPFR. È facile dimenticare che il numero finito di bit usato per memorizzare il valore spesso è solo un'approssimazione dopo un opportuno arrotondamento. Il test di uguaglianza riesce se e solo se *tutti* i bit dei due operandi sono esattamente gli stessi. Poiché questo non è necessariamente vero dopo un calcolo in virgola mobile con una determinata precisione e con una modalità di arrotondamento valida, un test di uguaglianza convenzionale potrebbe non riuscire. Invece, il test riesce confrontando i due numeri per vedere se la differenza tra di loro rientra in un delta accettabile.

In applicazioni dove sono sufficienti fino a 15 cifre decimali, il calcolo in doppia precisione eseguito dall'hardware del computer può essere una buona soluzione, e in genere è più veloce. Però bisogna tener presente che ogni operazione in virgola mobile può subire un nuovo errore di arrotondamento con conseguenze catastrofiche, come si è visto nel precedente tentativo di calcolare il valore di  $\pi$ . In tali casi una precisione supplementare può aumentare la stabilità e l'accuratezza del calcolo.

Oltre a ciò, bisogna tenere conto del fatto che addizioni ripetute non sono necessariamente equivalenti a una moltiplicazione nell'aritmetica in virgola mobile. Nell'esempio visto in [Sezione 15.4.1.3 \[Gli errori diventano sempre maggiori\]](#), [pagina 384](#):

```
$ gawk 'BEGIN {
>   for (d = 1.1; d <= 1.5; d += 0.1) # ciclo eseguito cinque volte (?)
>       i++
>   print i
> }'
```

└ 4

non è detto che, scegliendo per `PREC` un valore arbitrariamente alto, si riesca a ottenere il risultato corretto. La riformulazione del problema in questione è spesso il modo corretto di comportarsi in tali situazioni.

### 15.4.3 Tentare di aggiungere bit di precisione e arrotondare

Invece dell'aritmetica in virgola mobile con precisione arbitraria, spesso tutto ciò di cui si ha bisogno è un aggiustamento della logica o di un diverso ordine delle operazioni nei calcoli. La stabilità e l'accuratezza del calcolo di  $\pi$  nel primo esempio possono essere migliorata usando la seguente semplice trasformazione algebrica:

$$(\sqrt{x * x + 1} - 1) / x \equiv x / (\sqrt{x * x + 1} + 1)$$

Dopo aver fatto questo cambiamento, il programma converge verso  $\pi$  in meno di 30 iterazioni:

```
$ gawk -f pi2.awk
└ 3.215390309173473
└ 3.159659942097501
└ 3.146086215131436
└ 3.142714599645370
└ 3.141873049979825
...
└ 3.141592653589797
└ 3.141592653589797
```

### 15.4.4 Impostare la precisione

`gawk` usa una precisione di lavoro a livello globale; non tiene traccia della precisione e accuratezza dei singoli numeri. Eseguendo un'operazione aritmetica o chiamando una funzione predefinita, il risultato viene arrotondato alla precisione di lavoro. La precisione di lavoro di default è di 53 bit, modificabile usando la variabile predefinita `PREC`. Si può anche impostare il valore a una delle stringhe predefinite (non importa se scritte in maiuscolo o minuscolo) elencate in [Tabella 15.3](#), per emulare un formato binario che segue lo standard IEEE 754.

PREC	<b>formato binario IEEE 754</b>
"half"	16-bit mezza precisione
"single"	32-bit singole precisione di base
"double"	64-bit doppia precisione di base
"quad"	128-bit quadrupla precisione di base
"oct"	256-bit ottupla precisione

Tabella 15.3: Stringhe di precisione predefinita per PREC

Il seguente esempio illustra gli effetti del cambiamento di precisione sulle operazioni aritmetiche:

```
$ gawk -M -v PREC=100 'BEGIN { x = 1.0e-400; print x + 0
> PREC = "double"; print x + 0 }'
-| 1e-400
-| 0
```

**ATTENZIONE:** Diffidare delle costanti in virgola mobile! Quando si legge una costante in virgola mobile dal codice sorgente di un programma, **gawk** usa la precisione di default (quella del formato **double** di C), a meno che non venga richiesto, tramite la variabile speciale **PREC** fornita sulla riga di comando, di memorizzarla internamente come un numero MPFR. Cambiare la precisione tramite **PREC** nel testo del programma *non* cambia la precisione di una costante.

Se si deve rappresentare una costante in virgola mobile con una precisione maggiore di quella di default e non è possibile usare un assegnamento a `PREC` da riga di comando, si dovrebbe definire la costante o come stringa, o come numero razionale, ove possibile. L'esempio seguente illustra le differenze tra i diversi modi di stampare una costante in virgola mobile:

```
$ gawk -M 'BEGIN { PREC = 113; printf("%.25f\n", 0.1) }'
└─ 0.10000000000000000055511151
$ gawk -M -v PREC=113 'BEGIN { printf("%.25f\n", 0.1) }'
└─ 0.1000000000000000000000000000
$ gawk -M 'BEGIN { PREC = 113; printf("%.25f\n", "0.1") }'
└─ 0.1000000000000000000000000000
$ gawk -M 'BEGIN { PREC = 113; printf("%.25f\n", 1/10) }'
└─ 0.1000000000000000000000000000
```

### 15.4.5 Impostare la modalità di arrotondamento

La variabile `ROUNDMODE` permette di controllare a livello di programma la modalità di arrotondamento. La corrispondenza tra `ROUNDMODE` e le modalità di arrotondamento IEEE è mostrata in [Tabella 15.4](#).

Modalità di arrotondamento	Nome IEEE	ROUNDMODE
Arrotonda al più vicino, o a un numero pari	<code>roundTiesToEven</code>	"N" o "n"
Arrotonda verso infinito	<code>roundTowardPositive</code>	"U" o "u"
Arrotonda verso meno infinito	<code>roundTowardNegative</code>	"D" o "d"
Arrotonda verso zero (troncamento)	<code>roundTowardZero</code>	"Z" o "z"
Arrotonda al più vicino, o per eccesso	<code>roundTiesToAway</code>	"A" o "a"

Tabella 15.4: Modalità di arrotondamento in `gawk`

`ROUNDMODE` ha "N" come valore di default, ovvero si usa la modalità di arrotondamento IEEE 754 `roundTiesToEven`. In Tabella 15.4, il valore "A" seleziona `roundTiesToAway`. Questo è applicabile solo se la versione in uso della libreria MPFR lo supporta; altrimenti, l'impostazione di `ROUNDMODE` ad "A" non ha alcun effetto.

La modalità di default `roundTiesToEven` è la più preferita, ma allo stesso tempo la meno intuitiva. Questo metodo fa la cosa ovvia per la maggior parte dei valori, arrotondandoli per eccesso o per difetto alla cifra più prossima. Per esempio, arrotondando 1.132 alle due cifre decimali si ottiene 1.13, e 1.157 viene arrotondato a 1.16.

Tuttavia, se si deve arrotondare un valore posto esattamente a metà strada, le cose non funzionano come probabilmente si insegna a scuola. In questo caso, il numero è arrotondato alla cifra *pari* più prossima. Così arrotondando 0.125 alle due cifre si arrotonda per difetto a 0.12, ma arrotondando 0.6875 alle tre cifre si arrotonda per eccesso a 0.688. Probabilmente ci si è già imbattuti in questa modalità di arrotondamento usando `printf` per formattare numeri a virgola mobile. Per esempio:

```
BEGIN {
    x = -4.5
    for (i = 1; i < 10; i++) {
        x += 1.0
        printf("%4.1f => %2.0f\n", x, x)
    }
}
```

produce il seguente output quando viene eseguito sul sistema dell'autore:<sup>4</sup>

```
-3.5 => -4
-2.5 => -2
-1.5 => -2
-0.5 => 0
0.5 => 0
1.5 => 2
2.5 => 2
3.5 => 4
4.5 => 4
```

<sup>4</sup> È possibile che l'output sia completamente diverso, se la libreria C presente nel sistema in uso non si conforma, per `printf`, alla regola IEEE 754 di arrotondamento al valore pari in caso di equidistanza.

La teoria che sta dietro alla regola `roundTiesToEven` è che gli arrotondamenti di valori equidistanti in eccesso e in difetto si distribuiscono più o meno uniformemente, con la possibile conseguenza che errori di arrotondamento ripetuti tendono ad annullarsi a vicenda. Questa è la modalità di arrotondamento di default per funzioni e operatori di calcolo secondo IEEE 754.

Le altre modalità di arrotondamento sono usate raramente. Gli arrotondamenti verso l'infinito (`roundTowardPositive`) e verso il meno infinito (`roundTowardNegative`) vengono spesso usati per eseguire calcoli su intervalli, dove si adotta questa modalità di arrotondamento per calcolare i limiti superiore e inferiore per l'intervallo di valori in uscita. La modalità `roundTowardZero` può essere usata per convertire numeri a virgola mobile in numeri interi. La modalità di arrotondamento `roundTiesToAway` arrotonda il risultato al numero più vicino, e in caso di equidistanza arrotonda per eccesso.

Qualche esperto di analisi numerica dirà che la scelta dello stile di arrotondamento ha un grandissimo impatto sul risultato finale, e consiglierà di attendere sino al risultato finale dopo ogni arrotondamento. Invece, spesso si possono evitare problemi legati a errori di arrotondamento impostando all'inizio la precisione a un valore sufficientemente maggiore della precisione desiderata, in modo che il cumulo degli errori di arrotondamento non influisca sul risultato finale. Se si ha il dubbio che i risultati del calcolo contengano un'accumulazione di errori di arrotondamento, occorre, per accertare la cosa, controllare se si verifica una differenza significativa nell'output cambiando la modalità di arrotondamento.

## 15.5 Aritmetica dei numeri interi a precisione arbitraria con gawk

Quando viene specificata l'opzione `-M`, `gawk` esegue tutti i calcoli sui numeri interi usando gli interi a precisione arbitraria della libreria GMP. Qualsiasi numero che appaia come un intero in un sorgente o in un file-dati è memorizzato come intero a precisione arbitraria. La dimensione del numero intero ha come limite solo la memoria disponibile. Per esempio, il seguente programma calcola  $5^{4^{3^2}}$ , il cui risultato è oltre i limiti degli ordinari valori a virgola mobile a doppia precisione dei processori:

```
$ gawk -M 'BEGIN {
>   x = 5^4^3^2
>   print "numero di cifre =", length(x)
>   print substr(x, 1, 20), "...", substr(x, length(x) - 19, 20)
> }'
+ numero di cifre = 183231
+ 62060698786608744707 ... 92256259918212890625
```

Se invece si dovesse calcolare lo stesso valore usando valori a virgola mobile con precisione arbitraria, la precisione necessaria per il risultato corretto (usando la formula  $prec = 3.322 \cdot dps$ ) sarebbe  $3.322 \cdot 183231$ , o 608693.

Il risultato di un'operazione aritmetica tra un intero e un valore a virgola mobile è un valore a virgola mobile con precisione uguale alla precisione di lavoro. Il seguente programma calcola l'ottavo termine nella successione di Sylvester<sup>5</sup> usando una ricorrenza:

```
$ gawk -M 'BEGIN {
```

---

<sup>5</sup> Weisstein, Eric W. *Sylvester's Sequence*. From MathWorld—A Wolfram Web Resource (<http://mathworld.wolfram.com/SylvestersSequence.html>).

```
> s = 2.0
> for (i = 1; i <= 7; i++)
>     s = s * (s - 1) + 1
> print s
> }'
+ 113423713055421845118910464
```

Il risultato mostrato differisce dal numero effettivo, 113.423.713.055.421.844.361.000.443, perché la precisione di default di 53 bit non è sufficiente per rappresentare esattamente il risultato in virgola mobile. Si può o aumentare la precisione (in questo caso bastano 100 bit), o sostituire la costante in virgola mobile '2.0' con un intero, per eseguire tutti i calcoli usando l'aritmetica con gli interi per ottenere l'output corretto.

A volte `gawk` deve convertire implicitamente un intero con precisione arbitraria in un valore a virgola mobile con precisione arbitraria. Ciò si rende necessario principalmente perché la libreria MPFR non sempre prevede l'interfaccia necessaria per elaborare interi a precisione arbitraria o numeri di tipo eterogeneo come richiesto da un'operazione o funzione. In tal caso, la precisione viene impostata al minimo valore necessario per una conversione esatta, e non viene usata la precisione di lavoro. Se questo non è quello di cui si ha bisogno o che si vuole, si può ricorrere a un sotterfugio e convertire preventivamente l'intero in un valore a virgola mobile, come qui di seguito:

```
gawk -M 'BEGIN { n = 13; print (n + 0.0) % 2.0 }'
```

Si può evitare completamente questo passaggio specificando il numero come valore a virgola mobile fin dall'inizio:

```
gawk -M 'BEGIN { n = 13.0; print n % 2.0 }'
```

Si noti che, per questo specifico esempio, probabilmente è meglio semplicemente specificare:

```
gawk -M 'BEGIN { n = 13; print n % 2 }'
```

Dividendo due interi a precisione arbitraria con '/' o con '%', il risultato è tipicamente un valore a virgola mobile con precisione arbitraria (a meno che il risultato non sia un numero intero esatto). Per eseguire divisioni intere o calcolare moduli con interi a precisione arbitraria, usare la funzione predefinita `intdiv()` (si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), [pagina 196](#)).

Si può simulare la funzione `intdiv()` in `awk` standard usando questa funzione definita dall'utente:

```
# intdiv --- fa una divisione intera

function intdiv(numerator, denominator, result)
{
    split("", result)

    numerator = int(numerator)
    denominator = int(denominator)
    result["quotient"] = int(numerator / denominator)
    result["remainder"] = int(numerator % denominator)

    return 0.0
}
```

```
}
```

Il seguente programma d'esempio, proposto da Katie Wasserman, usa `intdiv()` per calcolare le cifre di  $\pi$  al numero di cifre significative che si è scelto di impostare:

```
# pi.awk --- calcola le cifre di pi

BEGIN {
    cifre = 100000
    due = 2 * 10 ^ cifre
    pi = due
    for (m = cifre * 4; m > 0; --m) {
        d = m * 2 + 1
        x = pi * m
        intdiv(x, d, risultato)
        pi = risultato["quotient"]
        pi = pi + due
    }
    print pi
}
```

Quando gli fu chiesto dell'algoritmo usato, Katie rispose:

Non è quello più noto ma nemmeno quello più incomprensibile. È la variante di Eulero al metodo di Newton per il calcolo del Pi greco. Si vedano le righe (23) - (25) nel sito: <http://mathworld.wolfram.com/PiFormulas.html>.

L'algoritmo che ho scritto semplicemente espande il moltiplicare per 2 e lavora dall'espressione più interna verso l'esterno. Ho usato questo per programmare delle calcolatrici HP perché è piuttosto facile da adattare ai dispositivi di scarsa memoria con dimensioni di parola piuttosto piccole. Si veda <http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/articles.cgi?read=899>.

## 15.6 Confronto tra standard e uso corrente

Per diverso tempo, `awk` ha convertito le stringhe dall'aspetto non numerico nel valore numerico zero, quando richiesto. Per di più, la definizione originaria del linguaggio e lo standard POSIX originale prevedevano che `awk` riconoscesse solo i numeri decimali (base 10), e non i numeri ottali (base 8) o esadecimali (base 16).

Le modifiche nel linguaggio degli standard POSIX 2001 e 2004 possono essere interpretate nel senso che `awk` debba fornire delle funzionalità aggiuntive. Queste sono:

- Interpretazione del valore dei dati a virgola mobile specificati in notazione esadecimale (p.es., `0xDEADBEEF`). (Da notare: valore dei dati letti, *non* costanti facenti parte del codice sorgente.)
- Supporto per i valori a virgola mobile speciali IEEE 754 “not a number” (NaN), più infinito (“inf”) e meno infinito (“-inf”). In particolare, il formato per questi valori è quello specificato dallo standard C ISO 1999, che non distingue maiuscole/minuscole e può consentire caratteri aggiuntivi dipendenti dall'implementazione dopo il ‘`nan`’, e consentire o ‘`inf`’ o ‘`infinity`’.

Il primo problema è che entrambe le modifiche sono deviazioni evidenti dalla prassi consolidata:

- Il manutentore di **gawk** crede che supportare i valori a virgola mobile esadecimale, nello specifico, sia sbagliato, e che non sia mai stata intenzione dell'autore originale di introdurlo nel linguaggio.
- Consentire che stringhe completamente alfabetiche abbiano valori numerici validi è anch'essa una deviazione molto marcata dalla prassi consolidata.

Il secondo problema è che il manutentore di **gawk** crede che questa interpretazione dello standard, che richiede una certa dimestichezza col linguaggio giuridico per essere compresa, non sempre è stata colta dai normali sviluppatori. In altre parole, “Sappiamo come siete arrivati sin qui, ma non pensiamo che questo sia il posto dove volete essere.”

Recependo queste argomentazioni, e cercando nel contempo di assicurare la compatibilità con le versioni precedenti dello standard, lo standard POSIX 2008 ha aggiunto delle formulazioni esplicite per consentire l'uso da parte di **awk**, solo a richiesta, dei valori a virgola mobile esadecimale e dei valori speciali “*not a number*” e infinito.

Sebbene il manutentore di **gawk** continui a credere che introdurre queste funzionalità sia sconsigliabile, ciò nonostante, sui sistemi che supportano i valori in virgola mobile IEEE, sembra giusto fornire *qualche* possibilità di usare i valori NaN e infinito. La soluzione implementata in **gawk** è questa:

- Se è stata specificata l'opzione da riga di comando `--posix`, **gawk** non interviene. I valori di stringa sono passati direttamente alla funzione `strtod()` della libreria di sistema, e se quest'ultima restituisce senza errori un valore numerico, esso viene usato.<sup>6</sup> Per definizione, i risultati non sono portabili su diversi sistemi; e sono anche piuttosto sorprendenti:

```
$ echo nanny | gawk --posix '{ print $1 + 0 }'
+ nan
$ echo 0xDeadBeef | gawk --posix '{ print $1 + 0 }'
+ 3735928559
```

- Senza l'opzione `--posix`, **gawk** interpreta i quattro valori di stringa ‘+inf’, ‘-inf’, ‘+nan’ e ‘-nan’ in modo speciale, producendo i corrispondenti valori numerici speciali. Il segno iniziale serve per segnalare a **gawk** (e all'utente) che il valore è realmente numerico. I numeri a virgola mobile esadecimale non sono consentiti (a meno di non usare anche `--non-decimal-data`, che *non* è consigliabile). Per esempio:

```
$ echo nanny | gawk '{ print $1 + 0 }'
+ 0
$ echo +nan | gawk '{ print $1 + 0 }'
+ nan
$ echo 0xDeadBeef | gawk '{ print $1 + 0 }'
+ 0
```

**gawk** ignora la distinzione maiuscole/minuscole nei quattro valori speciali. Così, ‘+nan’ e ‘+NaN’ sono la stessa cosa.

---

<sup>6</sup> L'avete voluto, tenetevelo.

## 15.7 Sommario

- La maggior parte dell'aritmetica al calcolatore è fatta usando numeri interi oppure valori a virgola mobile. L'**awk** standard usa valori a virgola mobile a doppia precisione.
- Nei primi anni '90 Barbie disse erroneamente, “L'ora di matematica è ostica!” Sebbene la matematica non sia ostica, l'aritmetica a virgola mobile non è proprio come la matematica “carta e penna”, e bisogna prestare attenzione:
  - Non tutti i numeri possono essere rappresentati in modo esatto.
  - Per confrontare dei valori bisognerebbe usare un delta, invece di farlo direttamente con ‘==’ e ‘!=’.
  - Gli errori si accumulano.
  - Le operazioni non sempre sono esattamente associative o distributive.
- Aumentare l'accuratezza può essere d'aiuto, ma non è una panacea.
- Spesso, aumentare la precisione e poi arrotondare al numero di cifre desiderato produce risultati soddisfacenti.
- Specificare l'opzione **-M** (o **--bignum**) per abilitare il calcolo MPFR. Usare **PREC** per impostare la precisione in bit, e **ROUNDMODE** per impostare la modalità di arrotondamento tra quelle previste nello standard IEEE 754.
- Specificando l'opzione **-M**, **gawk** esegue calcoli su interi a precisione arbitraria usando la libreria GMP. In tal modo si ha una maggiore velocità e una più efficiente allocazione dello spazio rispetto all'uso di MPFR per eseguire gli stessi calcoli.
- Ci sono diverse aree per quanto attiene ai numeri a virgola mobile in cui **gawk** è in disaccordo con lo standard POSIX. È importante averlo ben presente.
- In generale, non vi è alcun bisogno di essere eccessivamente diffidenti verso i risultati del calcolo in virgola mobile. La lezione da ricordare è che il calcolo in virgola mobile è sempre più complesso di quello che si fa con carta e penna. Per trarre vantaggio dalla potenza del calcolo in virgola mobile, bisogna conoscere i suoi limiti e stare all'interno di essi. Per la maggior parte degli usi occasionali del calcolo in virgola mobile, si possono ottenere i risultati attesi semplicemente arrotondando la visualizzazione dei risultati finali al giusto numero di cifre decimali significative.
- Come consiglio generale, evitare di rappresentare dati numerici in maniera tale da far sembrare che la precisione sia maggiore di quella effettivamente necessaria.



## 16 Scrivere estensioni per gawk

È possibile aggiungere nuove funzioni, scritte in C o C++, a **gawk** usando librerie caricate dinamicamente. Questa funzionalità è disponibile su sistemi che supportano le funzioni C `dlopen()` e `dlsym()`. Questo capitolo descrive come creare estensioni usando codice scritto in C o C++.

Chi è completamente digiuno di programmazione in C può tranquillamente saltare questo capitolo, ma potrebbe valer la pena di dare un'occhiata alla documentazione sulle estensioni che sono installate insieme a **gawk** (si veda la [Sezione 16.7 \[Le estensioni di esempio incluse nella distribuzione gawk\]](#), pagina 445), e alle informazioni sul progetto **gawkextlib** (si veda la [Sezione 16.8 \[Il progetto gawkextlib\]](#), pagina 455). Gli esempi di estensione sono automaticamente compilati e installati quando si installa **gawk**.

**NOTA:** Se si specifica l'opzione `--sandbox`, le estensioni non sono disponibili (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

### 16.1 Cos'è un'estensione

Un'estensione (talora chiamata *plug-in*) è un frammento di codice compilato esternamente che **gawk** può caricare in fase di esecuzione per ottenere funzionalità ulteriori, che vanno ad aggiungersi a quelle di **gawk** descritte nel resto di questo libro.

Le estensioni sono utili perché consentono (ovviamente) di estendere le funzionalità di **gawk**. Per esempio, possono permettere l'uso di *chiamate di sistema* (come `chdir()` per cambiare directory) e di altre routine di libreria C potenzialmente utili. Come per la maggior parte del software, “il cielo è il limite”; se si riesce a immaginare qualcosa che si vuol fare e che è possibile programmare in C o C++, si può scrivere un'estensione che lo faccia!

Le estensioni sono scritte in C o C++, usando l'API (*Application Programming Interface*) definita per questo scopo dagli sviluppatori di **gawk**. Il resto di questo capitolo descrive le possibilità offerte dall'API e come usarle, e illustra una piccola estensione di esempio. Inoltre, sono documentati gli esempi di estensione inclusi nella distribuzione di **gawk** e viene descritto il progetto **gawkextlib**. Si veda la [Sezione C.5 \[Note di progetto dell'estensione API\]](#), pagina 506, per una disamina degli obiettivi e del progetto del meccanismo delle estensioni.

### 16.2 Tipo di licenza delle estensioni

Ogni estensione dinamica dev'essere distribuita in base a una licenza che sia compatibile con la licenza GNU GPL (si veda la [\[Licenza Pubblica Generale GNU \(GPL\)\]](#), pagina 529).

Per far sapere a **gawk** che la licenza è quella corretta, l'estensione deve definire il simbolo globale `plugin_is_GPL_compatible`. Se tale simbolo non è stato definito, **gawk** termina con un messaggio di errore fatale al momento del caricamento dell'estensione.

Il tipo dichiarato per il suddetto simbolo dev'essere `int`. Esso non deve tuttavia essere presente in ogni sezione allocata. Il controllo in essere si limita a constatare che quel simbolo esiste a livello globale. Qualcosa del genere può essere sufficiente:

```
int plugin_is_GPL_compatible;
```

### 16.3 Una panoramica sul funzionamento ad alto livello

La comunicazione tra **gawk** e un'estensione è bidirezionale. Dapprima, quando un'estensione è caricata, **gawk** le passa un puntatore a una struttura (**struct**) i cui campi sono dei puntatori di funzione. Questo si può vedere in [Figura 16.1](#).

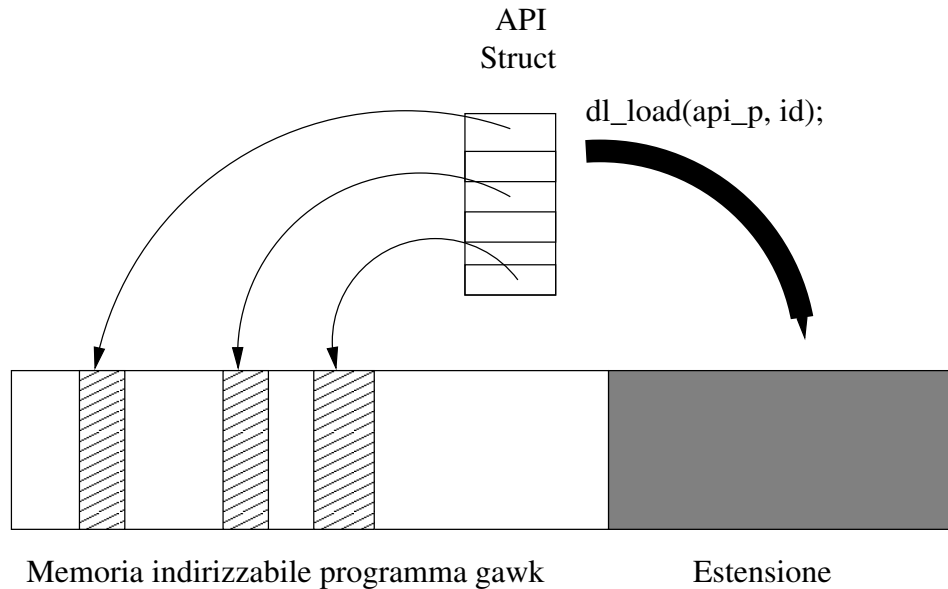


Figura 16.1: Caricamento dell'estensione

L'estensione è in grado di chiamare funzioni all'interno di **gawk** utilizzando questi puntatori a funzione, in fase di esecuzione, senza aver bisogno di accedere (in fase di compilazione), ai simboli di **gawk**. Uno di questi puntatori a funzione punta a una funzione che serve per "registrare" nuove funzioni. Questo è mostrato in [Figura 16.2](#).

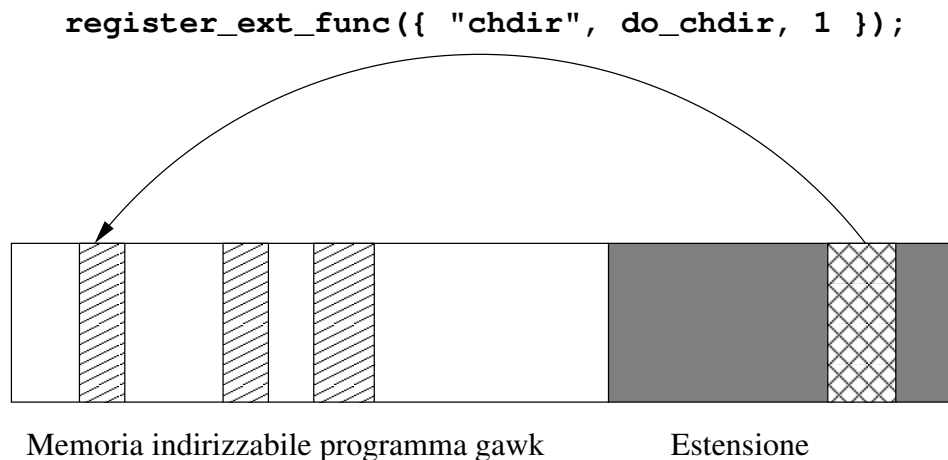


Figura 16.2: Registrare una nuova funzione

Nella direzione opposta, l'estensione registra le sue nuove funzioni con `gawk` passando dei puntatori che puntano alle funzioni che implementano la nuova funzionalità, (p.es. `do_chdir()`). `gawk` associa il puntatore a funzione con un nome ed è in grado di chiamarlo in seguito, usando una convenzione di chiamata predefinita. Questo è mostrato in [Figura 16.3](#).

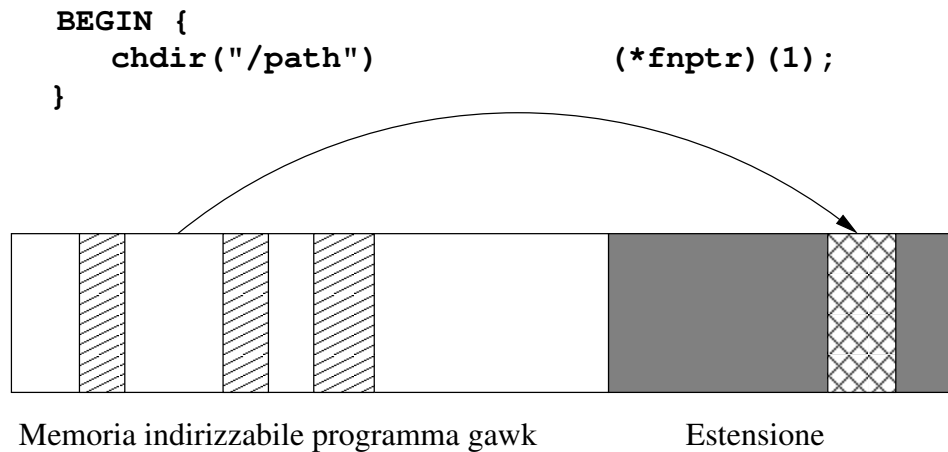


Figura 16.3: Chiamata della nuova funzione

La funzione `do_xxx()`, a sua volta, utilizza i puntatori a funzione nella struttura (`struct`) API per svolgere il proprio compito, come aggiornare variabili o vettori, stampare messaggi, impostare la variabile `ERRNO`, e così via.

Delle macro di servizio rendono la chiamata effettuata utilizzando i puntatori simile a quella delle funzioni normali, in modo che il codice sorgente delle estensioni rimanga sufficientemente leggibile e comprensibile.

Sebbene tutto ciò possa sembrare piuttosto complesso, il risultato è che il codice sorgente dell'estensione è abbastanza intuitivo da scrivere e da leggere. Lo si può constatare nell'estensione di esempio `filefuncs.c` (si veda la [Sezione 16.6 \[Esempio: alcune funzioni per i file\], pagina 434](#)), come pure nel codice `testext.c`, che testa l'interfaccia di programmazione (API).

Ecco alcuni ulteriori dettagli:

- L'API fornisce accesso ai valori delle variabili `gawk do_xxx`, che memorizzano opzioni della riga di comando come `do_lint`, `do_profiling`, e così via (si veda la [Sezione 16.4.13 \[Variabili fornite dall'API\], pagina 431](#)). Questi valori sono solo informativi: un'estensione non può modificarli all'interno di `gawk`. Oltre a ciò, il tentativo di assegnare loro dei valori produce un errore quando l'estensione viene compilata.
- L'API fornisce anche i numeri che identificano la specifica versione di `gawk`, in modo che un'estensione possa controllare se il comando `gawk` che l'ha caricata è in grado di supportare le funzionalità utilizzate nell'estensione. (Discrepanze tra le versioni “non dovrebbero” accadere, ma si sa come vanno *queste* cose.) Si veda la [Sezione 16.4.13.1 \[Costanti e variabili della versione dell'API\], pagina 431](#), per ulteriori dettagli.

## 16.4 Una descrizione completa dell'API

Il codice sorgente scritto in C o C++ per un'estensione deve includere il file di intestazione `gawkapi.h`, che dichiara le funzioni e definisce i tipi di dati usati per comunicare con `gawk`. Questa (non breve) sezione descrive l'API in dettaglio.

### 16.4.1 Introduzione alle funzioni dell'API

L'accesso a funzionalità interne a `gawk` è effettuato con una chiamata che usa i puntatori a funzione resi disponibili all'estensione.

Puntatori a funzioni API sono previsti per i seguenti tipi di operazioni:

- Allocare, riallocare e liberare memoria.
- Registrare funzioni. Si possono registrare:
  - Funzioni di estensione
  - Funzioni ausiliarie di pulizia (*callbacks*)
  - Una stringa di caratteri che identifica la versione
  - Funzioni per analizzare l'input
  - Funzioni per modificare l'output
  - Processori bidirezionali

Tutti questi elementi sono spiegati dettagliatamente nel resto di questo capitolo.

- Stampare messaggi fatali, di avvertimento e quelli generati dall'opzione "lint".
- Modificare `ERRNO` o annullarne il valore.
- Accedere a parametri, compresa la possibilità di definire come vettore un parametro ancora indefinito.
- Accedere alla "tabella dei simboli": procurarsi il valore di una variabile globale, crearne una nuova o modificarne una già esistente.
- Creare ed eliminare valori nascosti; ciò rende possibile usare un particolare valore per più di una variabile, e può migliorare parecchio le prestazioni.
- Manipolare vettori:
  - Ritrovare il valore di elementi del vettore, aggiungerne di nuovi, cancellare e modificare elementi esistenti.
  - Ottenere il numero di elementi presenti in un vettore
  - Creare un nuovo vettore
  - Cancellare un intero vettore
  - Appiattare un vettore per poter facilmente eseguire un ciclo, in stile C, su tutti i suoi indici ed elementi
- Accedere a ridirezioni e manipolarle.

Alcune osservazioni riguardo all'uso dell'API:

- I seguenti tipi di variabili, macro e/o funzioni sono resi disponibili nel file `gawkapi.h`. Perché siano utilizzabili, i rispettivi file di intestazione standard indicati devono essere stati specificati *prima* di includere `gawkapi.h`:

Elemento C	File d'intestazione
EOF	<stdio.h>
valori di <code>errno</code>	<errno.h>
FILE	<stdio.h>
NULL	<stddef.h>
<code>memcpy()</code>	<string.h>
<code>memset()</code>	<string.h>
<code>size_t</code>	<sys/types.h>
<code>struct stat</code>	<sys/stat.h>

Per ragioni di portabilità, specialmente per sistemi che non sono interamente aderenti agli standard, occorre assicurarsi di includere i file corretti nel modo corretto. Questa richiesta mira a mantenere il file `gawkapi.h` ordinato, invece che farlo diventare un'accozzaglia di problemi di portabilità, quale si può vedere in alcune parti del codice sorgente di `gawk`.

- Il file `gawkapi.h` può essere incluso più volte, senza conseguenze negative. Tuttavia sarebbe meglio evitare di farlo, per uno stile di programmazione migliore.
- Sebbene l'API usi solo funzionalità ISO C 90, c'è un'eccezione; le funzioni “costruttrici” usano la parola chiave `inline`. Se il compilatore in uso non supporta questa parola chiave, si dovrebbe specificare sulla riga di comando il parametro `-Dinline=` oppure usare gli strumenti Autotools GNU e includere un file `config.h` nel codice sorgente delle estensioni.
- Tutti i puntatori messi a disposizione da `gawk` puntano ad aree di memoria gestite da `gawk` e dovrebbero essere trattati dall'estensione come in sola lettura. Le aree di memoria che contengono *tutte* le stringhe passate a `gawk` dall'estensione *devono* provenire da una chiamata a `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`, e sono gestite da `gawk` da quel punto in avanti.
- L'API definisce parecchie semplici `struct` che mappano dei valori come sono visti da `awk`. Un valore può essere un numero `double` (a virgola mobile, in doppia precisione), una stringa o un vettore (come è il caso per i vettori multidimensionali o nella creazione di un nuovo vettore).

I valori di tipo stringa sono costituiti da un puntatore e da una lunghezza, poiché nella stringa possono essere presenti dei caratteri NUL (zeri binari, che normalmente marcano la fine di una stringa).

**NOTA:** Di proposito, `gawk` immagazzina le stringhe usando la codifica multibyte correntemente in uso (come definita dalle variabili d'ambiente `LC_XXX`) e non usando dei caratteri larghi (ovvero due byte per ogni carattere). Ciò riflette il modo con cui `gawk` memorizza le stringhe internamente, e anche il modo in cui i caratteri sono verosimilmente letti dai file in input e scritti nei file in output.

**NOTA:** I valori di una stringa passati a un'estensione da `gawk` hanno sempre un carattere NUL alla fine (come delimitatore). Quindi è possibile usare senza inconvenienti tali valori di stringa per chiamare funzioni di libreria standard e routine di sistema. Tuttavia, poiché `gawk` consente che all'interno di una stringa di dati possano essere presenti caratteri NUL, si dovrebbe

controllare che la lunghezza di ogni stringa passata un'estensione coincida con il valore restituito dalla funzione `strlen()` per la stringa stessa.

- Per ottenere un valore (p.es. quello di un parametro o quello di una variabile globale, oppure di un elemento di un vettore), l'estensione chiede un tipo specifico di variabile (numero, stringa, scalare, *value cookie* [si veda più avanti], vettore o “undefined”). Quando la richiesta è “undefined,” il valore restituito sarà quello originale della variabile in questione.

In ogni caso, se la richiesta e il tipo effettivo della variabile non corrispondono, la funzione di accesso restituisce “false” e fornisce il tipo proprio della variabile, in modo che l'estensione possa, p.es., stampare un messaggio di errore (del tipo “ricevuto uno scalare, invece del vettore previsto”).

Si possono chiamare le funzioni dell'API usando i puntatori a funzione direttamente, ma l'interfaccia non è molto elegante. Per permettere al codice sorgente delle estensioni di assomigliare di più a un codice normale, il file di intestazione `gawkapi.h` definisce parecchie macro da usare nel codice sorgente dell'estensione. Questa sezione presenta le macro come se si trattasse di funzioni.

### 16.4.2 I tipi di dati di impiego generale

*Ho un vero rapporto di amore/odio con le unioni.*

—Arnold Robbins

*Questo è ciò che contraddistingue le unioni: il compilatore è in grado di accomodare le cose in modo da far coesistere amore e odio.*

—Chet Ramey

L'estensione API definisce un certo numero di semplici tipi di dato e strutture di uso generale. Ulteriori strutture di dati, più specializzate, saranno introdotte nelle successive sezioni, insieme alle funzioni che ne fanno uso.

I tipi di dati e le strutture di uso generale sono le seguenti:

```
typedef void *awk_ext_id_t;
```

Un valore di questo tipo è trasmesso da **gawk** a un'estensione nel momento in cui viene caricata. Tale valore dev'essere restituito a **gawk** come primo parametro di ogni funzione API.

```
#define awk_const ...
```

Questa macro genera delle ‘costanti’ nel momento in cui si compila un'estensione, e non genera nulla quando si compila il comando **gawk** vero e proprio. Ciò rende alcuni campi nelle strutture dei dati dell'API non alterabili dal codice sorgente dell'estensione, ma consente al comando **gawk** di usarle secondo necessità.

```
typedef enum awk_bool {
```

```
    awk_false = 0,
```

```
    awk_true
```

```
} awk_bool_t;
```

Un semplice tipo di variabile booleana.

```
typedef struct awk_string {
    char *str;      /* dati veri e propri */
    size_t len;     /* lunghezza degli stessi, in caratteri */
} awk_string_t;
```

Questo rappresenta una stringa modificabile. `gawk` è responsabile per la gestione della memoria utilizzata, se ha fornito il valore della stringa. Altrimenti, assume il possesso della memoria in questione. *Questa memoria dev'essere resa disponibile chiamando una delle funzioni `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`!*

Come già detto, la rappresentazione delle stringhe in memoria usa la codifica multibyte corrente.

```
typedef enum {
    AWK_UNDEFINED,
    AWK_NUMBER,
    AWK_STRING,
    AWK_REGEX,
    AWK_STRNUM,
    AWK_ARRAY,
    AWK_SCALAR,      /* accesso opaco a una variabile */
    AWK_VALUE_COOKIE /* per aggiornare un valore
                       già creato */
} awk_valtype_t;
```

L'elenco `enum` indica di che tipo è un certo valore. È usato nella seguente struttura `struct`.

```
typedef struct awk_value {
    awk_valtype_t val_type;
    union {
        awk_string_t      s;
        double             d;
        awk_array_t        a;
        awk_scalar_t       scl;
        awk_value_cookie_t vc;
    } u;
} awk_value_t;
```

Un “valore di `awk`”. Il campo `val_type` indica che tipo di valore `union` contiene, e ogni campo è del tipo appropriato.

```
#define str_value      u.s
#define strnum_value   str_value
#define regex_value    str_value
#define num_value      u.d
#define array_cookie   u.a
#define scalar_cookie  u.scl
#define value_cookie   u.vc
```

L'uso di queste macro rende più facile da seguire l'accesso ai campi di `awk_value_t`.

```
typedef void *awk_scalar_t;
```

La variabili scalari possono essere rappresentate da un tipo opaco. Questi valori sono ottenuti da **gawk** e in seguito gli vengono restituiti. Questo argomento è discusso in maniera generale nel testo che segue questa lista, e più in dettaglio nella [Sezione 16.4.10.2 \[Accedere alle variabili per “cookie” e aggiornarle\]](#), [pagina 417](#).

```
typedef void *awk_value_cookie_t;
```

Un “*value cookie*” è un tipo di variabile opaca, e rappresenta un valore nascosto. Anche questo argomento è discusso in maniera generale nel testo che segue questa lista, e più in dettaglio nella [Sezione 16.4.10.3 \[Creare e usare valori nascosti\]](#), [pagina 419](#).

I valori di tipo scalare in **awk** sono numeri, stringhe, *strnum* o *regex* fortemente tipizzate. La struttura `awk_value_t` rappresenta valori. Il campo `val_type` indica cosa contiene `union`.

Rappresentare numeri è facile: l’API usa una variabile C di tipo `double`. Le stringhe richiedono uno sforzo maggiore. Poiché **gawk** consente che le stringhe contengano dei byte NUL (a zeri binari) nel valore di una stringa, una stringa dev’essere rappresentata da una coppia di campi che contengono il puntatore al dato vero e proprio e la lunghezza della stringa. È questo è il tipo `awk_string_t`.

Un valore di tipo *strnum* (stringa numerica) è rappresentato come una stringa e consiste di dati in input forniti dall’utente che appaiono essere numerici. Quando una funzione di estensione crea un valore di tipo *strnum*, il risultato è una stringa che viene marcata come immessa dall’utente. La successiva analisi da parte di **gawk** servirà poi a determinare se la stringa appare essere un numero, e va quindi trattata come *strnum*, invece che come una normale stringa di caratteri.

Ciò è utile nei casi in cui una funzione di estensione desideri fare qualcosa di paragonabile alla funzione `split`, la quale imposta l’attributo di *strnum* agli elementi di vettore che crea. Per esempio, un’estensione che implementi la divisione di record CSV (Comma Separated Values, i cui elementi sono delimitati da virgole) potrebbe voler usare questa funzionalità. Un’altra situazione in cui ciò è utile è quello di una funzione che richieda campi-dati ad una banca di dati. La funzione `PQgetvalue()` della banca dati PostgreSQL, per esempio, restituisce una stringa che può essere numerica o di tipo carattere, a seconda del contesto.

I valori di *regex* fortemente tipizzate (si veda la [Sezione 6.1.2.2 \[Costanti regex fortemente tipizzate\]](#), [pagina 118](#), non sono molto utili nelle funzioni di estensione. Le funzioni di estensione possono stabilire di averli ricevuti, e crearne, attribuendo valori di tipo scalare. In alternativa, è possibile esaminare il testo della *regex* utilizzando campi `regex_value.str` e `regex_value.len`.

Identificativi (cioè, nomi di variabili globali) possono essere associati sia a valori scalari che a vettori. Inoltre, **gawk** consente veri vettori di vettori, in cui ogni singolo elemento di un vettore può a sua volta essere un vettore. La spiegazione dei vettori è rinviata alla [Sezione 16.4.11 \[Manipolazione di vettori\]](#), [pagina 421](#).

La varie macro sopra elencate facilitano l’uso degli elementi delle `union` come se fossero campi in una `struct`; è questa una pratica comunemente adottata nella scrittura di programmi in C. Questo tipo di codice è più semplice da scrivere e da leggere, ma resta una

responsabilità *del programmatore* assicurarsi che il campo `val_type` rifletta correttamente il tipo del valore contenuto nella struttura `awk_value_t`.

Dal punti di vista concettuale, i primi tre campi dell'`union` (numero, stringa, e vettore) sono sufficienti per lavorare con i valori `awk`. Tuttavia, poiché l'API fornisce routine per ottenere e modificare il valore di una variabile scalare globale usando solo il nome della variabile, si ha qui una perdita di efficienza: `gawk` deve cercare la variabile ogni volta che questa è utilizzata e modificata. Questo è un problema reale, non solo un problema teorico.

Per questo motivo, se si sa che una certa estensione passerà molto tempo a leggere e/o modificare il valore di una o più variabili scalari, si può ottenere uno *scalar cookie*<sup>1</sup> per quella variabile, e poi usare il *cookie* per ottenere il valore della variabile o per modificarne il valore. Il tipo `awk_scalar_t` contiene uno *scalar cookie*, e la macro `scalar_cookie` fornisce accesso al valore di quel tipo nella struttura `awk_value_t`. Dato uno *scalar cookie*, `gawk` può trovare o modificare direttamente il valore, come richiesto, senza bisogno di andarlo a cercare ogni volta.

Il tipo `awk_value_cookie_t` e la macro `value_cookie` sono simili. Se si pensa di dover usare lo stesso *valore* numerico o la stessa *stringa* per una o più variabili, si può creare il valore una volta per tutte, mettendo da parte un *value cookie* per quel valore, e in seguito specificare quel *value cookie* quando si desidera impostare il valore di una variabile. Ciò consente di risparmiare spazio in memoria all'interno del processo di `gawk` e riduce il tempo richiesto per creare il valore.

### 16.4.3 Funzioni per allocare memoria e macro di servizio

L'API fornisce alcune funzioni per effettuare *allocazioni di memoria* che possono essere passate a `gawk`, e anche un certo numero di macro che possono tornare utili. Questa sottosezione le presenta come prototipi di funzione, nel modo con cui il codice dell'estensione potrebbe usarle:

```
void *gawk_malloc(size_t size);
```

Chiama la versione corretta di `malloc()` per allocare memoria, che può in seguito essere messa a disposizione di `gawk`.

```
void *gawk_calloc(size_t nmemb, size_t size);
```

Chiama la versione corretta di `calloc()` per allocare memoria che può in seguito essere messa a disposizione di `gawk`.

```
void *gawk_realloc(void *ptr, size_t size);
```

Chiama la versione corretta di `realloc()` per allocare memoria che può in seguito essere messa a disposizione di `gawk`.

```
void gawk_free(void *ptr);
```

Chiama la versione corretta di `free()` per liberare memoria che era stata allocata con `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`.

L'API deve fornire queste funzioni perché è possibile che un'estensione sia stata compilata e costruita usando una versione diversa della libreria C rispetto a quella usata per il

<sup>1</sup> Si veda la voce “*cookie*” nello *Jargon file* per una definizione di *cookie*, e la voce “*magic cookie*” sempre nello *Jargon file* per un bell'esempio. Si veda anche la voce “*Cookie*” nel [Glossario], pagina 515. [È disponibile in rete anche una traduzione italiana dello *Jargon file*]

programma eseguibile **gawk**.<sup>2</sup> Se **gawk** usasse la propria versione di **free()** per liberare della memoria acquisita tramite una differente versione di **malloc()**, il risultato sarebbe molto probabilmente differente da quello atteso.

Due macro di utilità possono essere usate per allocare memoria tramite **gawk\_malloc()** e **gawk\_realloc()**. Se l'allocazione non riesce, **gawk** termina l'esecuzione con un messaggio di errore fatale. Queste macro dovrebbero essere usate come se fossero dei richiami a procedure che non restituiscono un codice di ritorno:

```
#define emalloc(pointer, type, size, message) ...
```

Gli argomenti per questa macro sono i seguenti:

<b>pointer</b>	La variabile di tipo puntatore che punterà alla memoria allocata.
<b>type</b>	Il tipo della variabile puntatore. Questo è usato per definire il tipo quando si chiama <b>gawk_malloc()</b> .
<b>size</b>	Il numero totale di byte da allocare.
<b>message</b>	Un messaggio da anteporre all'eventuale messaggio di errore fatale. Questo è solitamente il nome della funzione che sta usando la macro.

Per esempio, si potrebbe allocare il valore di una stringa così:

```
awk_value_t risultato;
char *message;
const char greet[] = "non v'allarmate!";

emalloc(message, char *, sizeof(greet), "myfunc");
strcpy(message, greet);
make_malloced_string(message, strlen(message), & risultato);
```

```
#define erealloc(pointer, type, size, message) ...
```

Questo è simile a **emalloc()**, ma chiama **gawk\_realloc()** invece che **gawk\_malloc()**. Gli argomenti sono gli stessi della macro **emalloc()**.

#### 16.4.4 Funzioni per creare valori

L'API fornisce varie funzioni di *costruzione* per creare valori di tipo stringa e di tipo numerico, e anche varie macro di utilità. Questa sottosezione le presenta tutte come prototipi di funzione, nel modo in cui il codice sorgente di un'estensione le userebbe:

```
static inline awk_value_t *
make_const_string(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione mette il valore di una stringa nella variabile **awk\_value\_t** puntata da **risultato**. La funzione presuppone che **stringa** sia una costante stringa C (o altri dati che formano una stringa), e automaticamente crea una *copia* dei dati che sarà immagazzinata in **risultato**. Viene restituito il puntatore **risultato**.

---

<sup>2</sup> Questo succede più spesso nei sistemi MS-Windows, ma può capitare anche in sistemi di tipo Unix.

```
static inline awk_value_t *
```

```
make_malloced_string(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione mette il valore di una stringa nella variabile `awk_value_t` puntata da `risultato`. Si presuppone che `stringa` sia un valore `'char *'` che punta a dati ottenuti in precedenza per mezzo di `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`. L'idea è che questi dati siano comunicati direttamente a `gawk`, che se ne assume la responsabilità. Viene restituito il puntatore `risultato`.

```
static inline awk_value_t *
```

```
make_null_string(awk_value_t *risultato);
```

Questa funzione specializzata crea una stringa nulla (il valore "undefined") nella variabile `awk_value_t` puntata da `risultato`. Viene restituito il puntatore `risultato`.

```
static inline awk_value_t *
```

```
make_number(double num, awk_value_t *risultato);
```

Questa funzione crea semplicemente un valore numerico nella variabile `awk_value_t`, puntata da `risultato`.

```
static inline awk_value_t *
```

```
make_const_user_input(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione è identica a `make_const_string()`, ma la stringa è marcata come input dell'utente, che dovrà essere trattata come *strnum* se il contenuto della stringa appare essere numerico.

```
static inline awk_value_t *
```

```
make_malloced_user_input(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione è identica a `make_malloced_string()`, ma la stringa è marcata come input dell'utente, che dovrà essere trattata come *strnum* se il contenuto della stringa appare essere numerico.

```
static inline awk_value_t *
```

```
make_const_regex(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione crea un valore di *regex* fortemente tipizzata, allocando una copia della stringa. `stringa` è l'espressione regolare, di lunghezza `lunghezza`.

```
static inline awk_value_t *
```

```
make_malloced_regex(const char *stringa, size_t lunghezza, awk_value_t
*risultato);
```

Questa funzione crea un valore di *regex* fortemente tipizzata. `stringa` è l'espressione regolare, di lunghezza `lunghezza`. Si aspetta che `stringa` sia un valore di tipo `'char *'` che punta a dati ottenuti in precedenza tramite una chiamata a `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`.

### 16.4.5 Funzioni di registrazione

Questa sezione descrive le funzioni dell'API per registrare parti di un'estensione con `gawk`.

### 16.4.5.1 Registrare funzioni di estensione

Le funzioni di estensione sono descritte dal seguente tracciato record:

```
typedef struct awk_ext_func {
    const char *name;
    awk_value_t *(*const function)(int num_actual_args,
                                   awk_value_t *risultato,
                                   struct awk_ext_func *finfo);

    const size_t max_expected_args;
    const size_t min_required_args;
    awk_bool_t suppress_lint;
    void *data;          /* puntatore di tipo opaco
                           a ogni informazione ulteriore */
} awk_ext_func_t;
```

I campi sono:

`const char *name;`

Il nome della nuova funzione. Il codice sorgente a livello di `awk` richiama la funzione usando questo nome. Il nome è una normale stringa di caratteri del linguaggio C.

I nomi di funzione devono rispettare le stesse regole che valgono per gli identificativi `awk`. Cioè, devono iniziare o con una lettera dell'alfabeto inglese o con un trattino basso, che possono essere seguiti da un numero qualsiasi di lettere, cifre o trattini bassi. L'uso di maiuscolo/minuscolo è significativo.

`awk_value_t *(*const function)(int num_actual_args,  
 awk_value_t *risultato,  
 struct awk_ext_func *finfo);`

Questo è un puntatore alla funzione C che fornisce la funzionalità per cui è stata scritta l'estensione. La funzione deve riempire l'area di memoria puntata da `*risultato` con un numero, con una stringa, oppure con una *regex*. `gawk` diventa il proprietario di tutte le stringhe di memoria. Come già detto sopra, la stringa di memoria *deve* essere stata ottenuta usando `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`.

L'argomento `num_actual_args` dice alla funzione scritta in C quanti parametri sono stati effettivamente passati dal codice chiamante all'interno di `awk`.

La funzione deve restituire il valore di `risultato`. Questo è per utilità del codice chiamante all'interno di `gawk`.

`size_t max_expected_args;`

Questo è il massimo numero di argomenti che la funzione si aspetta di ricevere. Se chiamata con un numero di argomenti maggiore di questo, e se è stato richiesto il controllo *lint*, `gawk` stampa un messaggio di avvertimento. Per ulteriori informazioni, si veda la descrizione di `suppress_lint`, più avanti in questa lista.

```
const size_t min_required_args;
```

Questo è il minimo numero di argomenti che la funzione si aspetta di ricevere. Se è chiamata con un numero inferiore di argomenti, **gawk** stampa un messaggio di errore fatale ed esce.

```
awk_bool_t suppress_lint;
```

Questo *flag* dice a **gawk** di non stampare un messaggio *lint* se è stato richiesto un controllo *lint* e se sono stati forniti più argomenti di quelli attesi. Una funzione di estensione può stabilire se **gawk** ha già stampato almeno uno di tali messaggi controllando se `num_actual_args > finfo->max_expected_args`. In tal caso, se la funzione non desidera la stampa di ulteriori messaggi, dovrebbe impostare `finfo->suppress_lint` a `awk_true`.

```
void *data;
```

Questo è un puntatore di tipo opaco a tutti quei dati che una funzione di estensione desidera avere disponibili al momento della chiamata. Passando alla funzione di estensione la struttura `awk_ext_func_t` e avendo al suo interno questo puntatore disponibile, rende possibile scrivere un'unica funzione C o C++ che implementa più di una funzione di estensione a livello **awk**.

Una volta preparato un record che descrive l'estensione, la funzione di estensione va registrata con **gawk** usando questa funzione dell'API:

```
awk_bool_t add_ext_func(const char *namespace, awk_ext_func_t *func);
```

Questa funzione restituisce il valore *true* se ha successo, oppure *false* in caso contrario. Il parametro `namespace` non è usato per ora; dovrebbe puntare a una stringa vuota (`""`). Il puntatore `func` è l'indirizzo di una `struct` che rappresenta la funzione stessa, come descritto sopra.

**gawk** non modifica ciò che è puntato da `func`, ma la funzione di estensione stessa riceve questo puntatore e può modificarlo e farlo puntare altrove, quindi il puntatore non è stato dichiarato di tipo costante (`const`).

La combinazione di `min_required_args`, `max_expected_args`, e `suppress_lint` può ingenerare confusione. Ecco delle linee-guida sul da farsi.

Un numero qualsiasi di argomenti è valido

Impostare `min_required_args` and `max_expected_args` a zero e impostare `suppress_lint` ad `awk_true`.

Un numero minimo di argomenti è richiesto, ma non c'è un limite al numero massimo

Impostare `min_required_args` al minimo richiesto. Impostare `max_expected_args` a zero e impostare `suppress_lint` ad `awk_true`.

Un numero minimo di argomenti è richiesto, ma c'è un limite al numero massimo

Impostare `min_required_args` al minimo richiesto. Impostare `max_expected_args` al massimo atteso. Impostare `suppress_lint` ad `awk_false`.

Un numero minimo di argomenti è richiesto, ma c'è un numero massimo non superabile

Impostare `min_required_args` al minimo richiesto. Impostare `max_expected_args` al massimo atteso. Impostare `suppress_lint` ad `awk_false`. Nella funzione di estensione, controllare che `num_actual_args` non ecceda `f->max_expected_args`. Se il massimo è superato, stampare un messaggio di errore fatale.

### 16.4.5.2 Registrare una funzione *exit callback*

Una funzione *exit callback* è una funzione che **gawk** invoca prima di completare l'esecuzione del programma. Siffatte funzioni sono utili se ci sono dei compiti generali di “pulizia” che dovrebbero essere effettuati nell'estensione (come chiudere connessioni a un *database* o rilasciare altre risorse). Si può registrare una tale funzione con **gawk** per mezzo della seguente funzione:

```
void awk_atexit(void (*funcp)(void *data, int exit_status),
               void *arg0);
```

I parametri sono:

- |              |   |
|--------------|---|
| <b>funcp</b> | Un puntatore alla funzione da chiamare prima che <b>gawk</b> completi l'esecuzione. Il parametro <b>data</b> sarà il valore originale di <b>arg0</b> . Il parametro <b>exit_status</b> è il valore del codice di ritorno che <b>gawk</b> intende passare alla chiamata di sistema <b>exit()</b> (che termina l'esecuzione del programma). |
| <b>arg0</b>  | Un puntatore a un'area dati privata che <b>gawk</b> mantiene perché sia poi passata alla funzione puntata da <b>funcp</b> .   |

Le funzioni *exit callback* sono chiamate in ordine inverso rispetto a quello con cui è stata fatta la registrazione con **gawk** (LIFO: Last In, First Out).

### 16.4.5.3 Registrare una stringa di versione per un'estensione

Si può registrare una stringa di versione che indica il nome e la versione di una data estensione a **gawk**, come segue:

```
void register_ext_version(const char *version);
```

Registra la stringa puntata da **version** con **gawk**. Si noti che **gawk** *non* copia la stringa **version**, e quindi questa stringa non dovrebbe essere modificata.

**gawk** stampa tutte le stringhe con le versioni di estensione registrate, quando viene invocato specificando l'opzione **--version**.

### 16.4.5.4 Analizzatori di input personalizzati

Per default, **gawk** legge file di testo come input. Il valore della variabile **RS** è usato per determinare la fine di un record, e subito dopo la variabile **FS** (o **FIELDWIDTHS** o **FPAT**) viene usata per suddividerlo in campi (si veda il [Capitolo 4 \[Leggere file in input\]](#), pagina 63). Viene inoltre impostato il valore di **RT** (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162).

Se lo si desidera, è possibile fornire un analizzatore di input personalizzato. Il compito di un analizzatore di input è di restituire un record al codice di **gawk**, che poi lo elaborerà, accompagnato, se necessario, da indicatori del valore e della lunghezza dei dati da usare per **RT**.

Per fornire un analizzatore personalizzato di input, occorre innanzitutto rendere disponibili due funzioni (dove **XXX** è un nome che fa da prefisso all'estensione intera):

```
awk_bool_t XXX_can_take_file(const awk_input_buf_t *iobuf);
```

Questa funzione esamina l'informazione disponibile in **iobuf** (che vedremo tra poco). Basandosi su tale informazione, decide se l'analizzatore di input personalizzato andrà usato per questo file. Se questo è il caso, dovrebbe restituire *true*.

Altrimenti, restituirà *false*. Nessuno stato (valori di variabili, etc.) dovrebbe venire modificato all'interno di gawk.

```
awk_bool_t XXX_take_control_of(awk_input_buf_t *iobuf);
```

Quando gawk decide di passare il controllo del file a questo analizzatore di input, richiamerà questa funzione. Questa funzione a sua volta deve assegnare un valore ad alcuni campi nella struttura `awk_input_buf_t` e assicurarsi che alcune condizioni siano verificate. Dovrebbe poi restituire *true*. Se si verifica un errore di qualche tipo, i campi in questione non dovrebbero venire riempiti, e la funzione dovrebbe restituire *false*; in questo caso gawk non utilizzerà più l'analizzatore personalizzato di input. I dettagli sono descritti più avanti.

L'estensione dovrebbe raccogliere queste funzioni all'interno di una struttura `awk_input_parser_t`, simile a questa:

```
typedef struct awk_input_parser {
    const char *name;      /* nome dell'analizzatore */
    awk_bool_t (*can_take_file)(const awk_input_buf_t *iobuf);
    awk_bool_t (*take_control_of)(awk_input_buf_t *iobuf);
    awk_const struct awk_input_parser *awk_const next; /* per uso
                                                         di gawk */
} awk_input_parser_t;
```

I campi sono:

```
const char *name;
```

Il nome dell'analizzatore di input. Questa è una normale stringa di caratteri del linguaggio C.

```
awk_bool_t (*can_take_file)(const awk_input_buf_t *iobuf);
```

Un puntatore alla funzione `XXX_can_take_file()`.

```
awk_bool_t (*take_control_of)(awk_input_buf_t *iobuf);
```

Un puntatore alla funzione `XXX_take_control_of()`.

```
awk_const struct input_parser *awk_const next;
```

Questa struttura è per uso di gawk; per questo motivo è marcata `awk_const` in modo che l'estensione non possa modificarla.

I passi da seguire sono i seguenti:

1. Creare una variabile `static awk_input_parser_t` e inizializzarla adeguatamente.
2. Quando l'estensione è caricata, registrare l'analizzatore personalizzato di input con gawk usando la funzione API `register_input_parser()` (descritta più sotto).

La definizione di una struttura `awk_input_buf_t` è simile a questa:

```
typedef struct awk_input {
    const char *name;      /* nome file */
    int fd;                /* descrittore di file */
#define INVALID_HANDLE (-1)
    void *opaque;          /* area dati privata
                           per l'analizzatore di input */
    int (*get_record)(char **out, struct awk_input *iobuf,
```

```

        int *errcode, char **rt_start, size_t *rt_len);
ssize_t (*read_func)();
void (*close_func)(struct awk_input *iobuf);
struct stat sbuf;      /* buffer per stat */
} awk_input_buf_t;

```

I campi si possono dividere in due categorie: quelli che sono usati (almeno inizialmente) da `XXX_can_take_file()`, e quelli che sono usati da `XXX_take_control_of()`. Il primo gruppo di campi, e il loro uso, è così definito:

```

const char *name;
    Il nome del file.

int fd;    Un descrittore di file per il file. Se gawk riesce ad aprire il file, il valore di fd
    non sarà uguale a INVALID_HANDLE [descrittore non valido]. In caso contrario,
    quello sarà il valore.

struct stat sbuf;
    Se il descrittore di file è valido, gawk avrà riempito i campi di questa struttura
    invocando la chiamata di sistema fstat().

```

La funzione `XXX_can_take_file()` dovrebbe esaminare i campi di cui sopra e decidere se l'analizzatore di input vada usato per il file. La decisione può dipendere da uno stato di **gawk** (il valore di una variabile definita in precedenza dall'estensione e impostata dal codice **awk**), dal nome del file, dal fatto che il descrittore di file sia valido o no, dalle informazioni contenute in `struct stat` o da una qualsiasi combinazione di questi fattori.

Una volta che `XXX_can_take_file()` restituisce *true*, e **gawk** ha deciso di usare l'analizzatore personalizzato, viene chiamata la funzione `XXX_take_control_of()`. Tale funzione si occupa di riempire il campo `get_record` oppure il campo `read_func` nella struttura `awk_input_buf_t`. La funzione si assicura inoltre che `fd` *not* sia impostato al valore `INVALID_HANDLE`. L'elenco seguente descrive i campi che possono essere riempiti da `XXX_take_control_of()`:

```

void *opaque;
    Questo campo è usato per contenere qualsiasi informazione di stato sia necessaria
    per l'analizzatore di input riguardo a questo file. Il campo è "opaco" per gawk.
    L'analizzatore di input non è obbligato a usare questo puntatore.

```

```

int (*get_record)(char **out,
    struct awk_input *iobuf,
    int *errcode,
    char **rt_start,
    size_t *rt_len);

```

Questo puntatore a funzione dovrebbe puntare a una funzione che crea i record in input. Tale funzione è il nucleo centrale dell'analizzatore di input. Il suo modo di operare è descritto nel testo che segue questo elenco.

```

ssize_t (*read_func)();
    Questo puntatore a funzione dovrebbe puntare a una funzione che ha lo stesso
    comportamento della chiamata di sistema standard POSIX read(). È in alter-
    nativa al puntatore a get_record. Il relativo comportamento è pure descritto
    nel testo che segue quest'elenco.

```

```
void (*close_func)(struct awk_input *iobuf);
```

Questo puntatore a funzione dovrebbe puntare a una funzione che fa la “pulizia finale”. Dovrebbe liberare ogni risorsa allocata da `XXX_take_control_of()`. Può anche chiudere il file. Se lo fa, dovrebbe impostare il campo `fd` a `INVALID_HANDLE`.

Se `fd` è ancora diverso da `INVALID_HANDLE` dopo la chiamata a questa funzione, `gawk` invoca la normale chiamata di sistema `close()`.

Avere una funzione di “pulizia” è facoltativo. Se l’analizzatore di input non ne ha bisogno, basta non impostare questo campo. In questo caso, `gawk` invoca la normale chiamata di sistema `close()` per il descrittore di file, che, quindi, dovrebbe essere valido.

La funzione `XXX_get_record()` svolge il lavoro di creazione dei record in input. I parametri sono i seguenti:

```
char **out
```

Questo è un puntatore a una variabile `char *` che è impostata in modo da puntare al record. `gawk` usa una sua copia locale dei dati, quindi l’estensione deve gestire la relativa area di memoria.

```
struct awk_input *iobuf
```

Questa è la struttura `awk_input_buf_t` per il file. I campi dovrebbero essere usati per leggere i dati (`fd`) e per gestire lo stato privato (`opaque`), se necessario.

```
int *errcode
```

Se si verifica un errore, `*errcode` dovrebbe essere impostato a un valore appropriato tra quelli contenuti in `<errno.h>`.

```
char **rt_start
```

```
size_t *rt_len
```

Se il concetto “fine record” è applicabile, `*rt_start` dovrebbe essere impostato per puntare ai dati da usare come RT, e `*rt_len` dovrebbe essere impostata alla lunghezza di quel campo. In caso contrario, `*rt_len` dovrebbe essere impostata a zero. `gawk` usa una sua copia di questi dati, quindi l’estensione deve gestire tale memoria.

Il codice di ritorno è la lunghezza del buffer puntato da `*out` oppure `EOF`, se è stata raggiunta la fine del file o se si è verificato un errore.

Poiché `errcode` è sicuramente un puntatore valido, non c’è bisogno di controllare che il valore sia `NULL`. `gawk` imposta `*errcode` a zero, quindi non c’è bisogno di impostarlo, a meno che non si verifichi un errore.

In presenza di un errore, la funzione dovrebbe restituire `EOF` e impostare `*errcode` a un valore maggiore di zero. In questo caso, se `*errcode` non è uguale a zero, `gawk` automaticamente aggiorna la variabile `ERRNO` usando il valore contenuto in `*errcode`. (In generale, impostare `*errcode = errno` dovrebbe essere la cosa giusta da fare.)

Invece di fornire una funzione che restituisce un record in input, è possibile fornirne una che semplicemente legge dei byte, e lascia che sia `gawk` ad analizzare i dati per farne dei record. In questo caso, i dati dovrebbero essere restituiti nella codifica multibyte propria della localizzazione corrente. Una siffatta funzione dovrebbe imitare il comportamento della

chiamata di sistema `read()`, e riempire il puntatore `read_func` con il proprio indirizzo nella struttura `awk_input_buf_t`.

Per default, `gawk` imposta il puntatore `read_func` in modo che punti alla chiamata di sistema `read()`. In questo modo l'estensione non deve preoccuparsi di impostare esplicitamente questo campo.

**NOTA:** Occorre decidere per l'uno o per l'altro metodo: o una funzione che restituisce un record o una che restituisce dei dati grezzi. Nel dettaglio, se si fornisce una funzione che prepara un record, `gawk` la invocherà, e non chiamerà mai la funzione che fa una lettura grezza.

`gawk` viene distribuito con un'estensione di esempio che legge delle directory, restituendo un record per ogni elemento contenuto nella directory (si veda la [Sezione 16.7.6 \[Leggere directory\]](#), pagina 451). Questo codice sorgente può essere usato come modello per scrivere un analizzatore di input personalizzato.

Quando si scrive un analizzatore di input, si dovrebbe progettare (e documentare) il modo con cui si suppone che interagisca con il codice `awk`. Si può scegliere di utilizzarlo per tutte le letture, e intervenire solo quando è necessario, (come fa l'estensione di esempio `readdir`). Oppure lo si può utilizzare a seconda del valore preso da una variabile `awk`, come fa l'estensione XML inclusa nel progetto `gawkextlib` (si veda la [Sezione 16.8 \[Il progetto gawkextlib\]](#), pagina 455). In quest'ultimo caso, il codice in una regola `BEGINFILE` può controllare `FILENAME` ed `ERRNO` per decidere se attivare un analizzatore di input (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), pagina 150) oppure no.

Un analizzatore di input va registrato usando la seguente funzione:

```
void register_input_parser(awk_input_parser_t *input_parser);
    Registra l'analizzatore di input puntato da input_parser con gawk.
```

#### 16.4.5.5 Registrare un processore di output

Un *processore di output* è l'immagine riflessa di un analizzatore di input. Consente a un'estensione di prendere il controllo dell'output indirizzato verso un file che sia stato aperto con gli operatori di ridirezione di I/O '`>`' o '`>>`' (si veda la [Sezione 5.6 \[Ridirigere l'output di print e printf\]](#), pagina 104).

Il processore di output è molto simile, come struttura, all'analizzatore di input:

```
typedef struct awk_output_wrapper {
    const char *name;    /* nome del processore */
    awk_bool_t (*can_take_file)(const awk_output_buf_t *outbuf);
    awk_bool_t (*take_control_of)(awk_output_buf_t *outbuf);
    awk_const struct awk_output_wrapper *awk_const next; /* per gawk */
} awk_output_wrapper_t;
```

I campi sono i seguenti:

```
const char *name;
```

Questo è il nome del processore di output.

```
awk_bool_t (*can_take_file)(const awk_output_buf_t *outbuf);
```

Questo è il puntatore a una funzione che esamina l'informazione contenuta nella struttura `awk_output_buf_t` puntata da `outbuf`. Dovrebbe restituire `true` se il

processore di output vuole elaborare il file, e *false* in caso contrario. Nessuno stato (valori di variabili, etc.) dovrebbe essere modificato all'interno di **gawk**.

```
awk_bool_t (*take_control_of)(awk_output_buf_t *outbuf);
```

La funzione puntata da questo campo viene chiamata quando **gawk** decide di consentire al processore di output di prendere il controllo del file. Dovrebbe riempire in maniera appropriata dei campi nella struttura **awk\_output\_buf\_t**, come descritto sotto, e restituire *true* se ha successo, *false* in caso contrario.

```
awk_const struct output_wrapper *awk_const next;
```

Questa struttura è per uso di **gawk**; per questo motivo è marcata **awk\_const** in modo che l'estensione non possa modificarlo.

La struttura **awk\_output\_buf\_t** è simile a questa:

```
typedef struct awk_output_buf {
    const char *name;    /* nome del file in output */
    const char *mode;    /* argomento mode per fopen */
    FILE *fp;           /* puntatore stdio file */
    awk_bool_t redirected; /* true se un processore è attivo */
    void *opaque;        /* per uso del processore di output */
    size_t (*gawk_fwrite)(const void *buf, size_t size, size_t count,
                          FILE *fp, void *opaque);
    int (*gawk_fflush)(FILE *fp, void *opaque);
    int (*gawk_ferror)(FILE *fp, void *opaque);
    int (*gawk_fclose)(FILE *fp, void *opaque);
} awk_output_buf_t;
```

Anche qui, l'estensione definirà le funzioni **XXX\_can\_take\_file()** e **XXX\_take\_control\_of()** che esaminano e aggiornano campi dati in **awk\_output\_buf\_t**. I campi dati sono i seguenti:

```
const char *name;
```

Il nome del file in output.

```
const char *mode;
```

La stringa *mode* (come sarebbe usata nel secondo argomento della chiamata di sistema **fopen()**) con cui il file era stato aperto.

```
FILE *fp;
```

Il puntatore **FILE** da **<stdio.h>**. **gawk** apre il file prima di controllare se esiste un processore di output.

```
awk_bool_t redirected;
```

Questo campo dev'essere impostato a *true* dalla funzione **XXX\_take\_control\_of()**.

```
void *opaque;
```

Questo puntatore è opaco per **gawk**. L'estensione dovrebbe usarlo per contenere un puntatore a qualsiasi dato privato associato al file.

```

size_t (*gawk_fwrite)(const void *buf, size_t size, size_t count,
                      FILE *fp, void *opaque);
int (*gawk_fflush)(FILE *fp, void *opaque);
int (*gawk_ferror)(FILE *fp, void *opaque);
int (*gawk_fclose)(FILE *fp, void *opaque);

```

Questi puntatori dovrebbero essere impostati per puntare a funzioni la cui azione sia equivalente a quella delle funzioni di `<stdio.h>`, se questo è ciò che si desidera. `gawk` usa questi puntatori a funzione per *tutti* gli output. `gawk` inietta i puntatori per puntare a funzioni interne “di passaggio” che si limitano a chiamare le funzioni normali di `<stdio.h>`, e quindi un’estensione deve ridefinire solo le funzioni appropriate per fare il lavoro richiesto.

La funzione `XXX_can_take_file()` dovrebbe decidere in base ai campi `name` e `mode`, e a ogni altro ulteriore indicatore di stato (p.es., valori di variabili `awk`) adatto allo scopo.

Quando `gawk` chiama `XXX_take_control_of()`, la funzione dovrebbe riempire i rimanenti campi in modo opportuno, tranne che per `fp`, che dovrebbe essere usato normalmente.

Il processore di output va registrato usando la seguente funzione:

```

void register_output_wrapper(awk_output_wrapper_t *output_wrapper);

```

Registra il processore di output puntato da `output_wrapper` con `gawk`.

#### 16.4.5.6 Registrare un processore bidirezionale

Un *processore bidirezionale* combina un analizzatore di input e un processore di output per un I/O bidirezionale usando l’operatore ‘|&’ (si veda la [Sezione 5.6 \[Ridirigere l’output di print e printf\], pagina 104](#)). Le strutture `awk_input_parser_t` e `awk_output_buf_t` sono usate nella maniera già descritta precedentemente.

Un processore bidirezionale è rappresentato dalla struttura seguente:

```

typedef struct awk_two_way_processor {
    const char *name; /* nome del processore bidirezionale */
    awk_bool_t (*can_take_two_way)(const char *name);
    awk_bool_t (*take_control_of)(const char *name,
                                awk_input_buf_t *inbuf,
                                awk_output_buf_t *outbuf);
    awk_const struct awk_two_way_processor *awk_const next; /* per gawk */
} awk_two_way_processor_t;

```

I campi sono i seguenti:

```

const char *name;

```

Il nome del processore bidirezionale.

```

awk_bool_t (*can_take_two_way)(const char *name);

```

La funzione puntata da questo campo dovrebbe restituire *true* se vuole gestire l’I/O bidirezionale per questo nome-file. La funzione non dovrebbe modificare alcuno stato (valori di variabili, etc.) all’interno di `gawk`.

```
awk_bool_t (*take_control_of)(const char *name,
                              awk_input_buf_t *inbuf,
                              awk_output_buf_t *outbuf);
```

La funzione puntata da questo campo dovrebbe riempire le strutture `awk_input_buf_t` e `awk_output_buf_t` puntate da `inbuf` e `outbuf`, rispettivamente. Queste strutture sono già state descritte in precedenza.

```
awk_const struct two_way_processor *awk_const next;
```

Questa struttura è per uso di `gawk`; per questo motivo è marcata `awk_const` in modo che l'estensione non possa modificarla.

Come per l'analizzatore di input e il processore di output, vanno fornite le funzione “sì, ci penso io” e “per questo, fai tu”, `XXX_can_take_two_way()` e `XXX_take_control_of()`.

Il processore bidirezionale va registrato usando la seguente funzione:

```
void register_two_way_processor(awk_two_way_processor_t *two_way_processor);
```

Registra il processore bidirezionale puntato da `two_way_processor` con `gawk`.

### 16.4.6 Stampare messaggi dalle estensioni

È possibile stampare diversi tipi di messaggi di avvertimento da un'estensione, come qui spiegato. Si noti che, per queste funzioni, si deve fornire l'ID di estensione ricevuto da `gawk` al momento in cui l'estensione è stata caricata:<sup>3</sup>

```
void fatal(awk_ext_id_t id, const char *format, ...);
```

Stampa un messaggio e poi `gawk` termina immediatamente l'esecuzione.

```
void nonfatal(awk_ext_id_t id, const char *format, ...);
```

Stampa un messaggio di errore non-fatale.

```
void warning(awk_ext_id_t id, const char *format, ...);
```

Stampa un messaggio di avvertimento.

```
void lintwarn(awk_ext_id_t id, const char *format, ...);
```

Stampa un messaggio di avvertimento “lint”. Normalmente questo equivale a stampare un messaggio di avvertimento, ma se `gawk` era stato invocato specificando l'opzione ‘`--lint=fatal`’, gli avvertimenti di *lint* diventano messaggi di errore fatali.

Tutte queste funzioni sono per il resto simili alla famiglia di funzioni `printf()` del linguaggio C, dove il parametro `format` è una stringa contenente dei caratteri normali e delle istruzioni di formattazione, mischiati tra loro.

### 16.4.7 Funzioni per aggiornare ERRNO

Le seguenti funzioni consentono l'aggiornamento della variabile `ERRNO`:

```
void update_ERRNO_int(int errno_val);
```

Imposta `ERRNO` alla stringa equivalente del codice di errore in `errno_val`. Il valore dovrebbe essere uno dei codici di errore definiti in `<errno.h>`, e `gawk`

---

<sup>3</sup> Poiché l'API usa solo funzionalità previste dal compilatore ISO C 90, non è possibile usare le macro di tipo variadico (che accettano un numero variabile di argomenti) disponibili nel compilatore ISO C 99, che nasconderebbero quel parametro. Un vero peccato!

lo trasforma in una stringa (qualora possibile, tradotta) usando la funzione `C strerror()`.

```
void update_ERRNO_string(const char *string);
```

Imposta `ERRNO` direttamente usando il valore della stringa specificata. `gawk` fa una copia del valore di `stringa`.

```
void unset_ERRNO(void);
```

Annulla il valore di `ERRNO`.

### 16.4.8 Richiedere valori

Tutte le funzioni che restituiscono valori da `gawk` funzionano allo stesso modo. Si fornisce un campo `awk_valtype_t` per indicare il tipo di valore che ci si aspetta. Se il valore disponibile corrisponde a quello richiesto, la funzione restituisce *true* e riempie il campo del risultato `awk_value_t`. Altrimenti, la funzione restituisce *false*, e il campo `val_type` indica il tipo di valore disponibile. A quel punto si può, a seconda di quel che richiede la situazione, stampare un messaggio di errore oppure ripetere la richiesta specificando il tipo di valore che risulta disponibile. Questo comportamento è riassunto nella [Tabella 16.1](#).

		Tipo di valore reale					
Tipo Richiesto	Stringa	Stringa	Stringa	Stringa	Stringa	false	false
	Strnum	false	Strnum	Strnum	false	false	false
	Numero	Numero	Numero	Numero	false	false	false
	Regexp	false	false	false	Regexp	false	false
	Vettore	false	false	false	false	Vettore	false
	Scalar	Scalar	Scalar	Scalar	Scalar	false	false
	Indefinito	Stringa	Strnum	Numero	Regexp	Vettore	Indefinito
	Value cookie	false	false	false	false	false	false

Tabella 16.1: Tipi di valori restituiti dall'API

### 16.4.9 Accedere ai parametri e aggiornarli

Due funzioni consentono di accedere agli argomenti (parametri) passati all'estensione. Esse sono:

```
awk_bool_t get_argument(size_t count,
                        awk_valtype_t wanted,
                        awk_value_t *risultato);
```

Riempie la struttura `awk_value_t` puntata da `risultato` con l'argomento numero `count`. Restituisce *true* se il tipo dell'argomento corrisponde a quello specificato in `wanted`, e *false* in caso contrario. In quest'ultimo caso, `risultato->val_type` indica il tipo effettivo dell'argomento (si veda la [Tabella 16.1](#)). La numerazione degli argomenti parte da zero: il primo argomento è il numero zero, il secondo è il numero uno, e così via. `wanted` indica il tipo di valore atteso.

```
awk_bool_t set_argument(size_t count, awk_array_t array);
```

Converte un parametro di tipo indefinito in un vettore; ciò permette la chiamata per riferimento per i vettori. Restituisce *false* se *count* è troppo elevato, o se il tipo di argomento è diverso da *undefined*. Si veda la [Sezione 16.4.11 \[Manipolazione di vettori\]](#), pagina 421, per ulteriori informazioni riguardo alla creazione di vettori.

### 16.4.10 Accedere alla Tabella dei simboli

Due insiemi di routine permettono di accedere alle variabili globali, e un insieme consente di creare e rilasciare dei valori nascosti.

#### 16.4.10.1 Accedere alle variabili per nome e aggiornarle

Le routine che seguono permettono di raggiungere e aggiornare le variabili globali a livello di *awk* per nome. Nel gergo dei compilatori, gli identificativi di vario tipo sono noti come *simboli*, da cui il prefisso “sym” nei nomi delle routine. La struttura di dati che contiene informazioni sui simboli è chiamata *Tabella dei simboli* (*Symbol table*). Le funzioni sono le seguenti:

```
awk_bool_t sym_lookup(const char *name,
                      awk_valtype_t wanted,
                      awk_value_t *risultato);
```

Riempie la struttura *awk\_value\_t* puntata da *risultato* con il valore della variabile il cui nome è nella stringa *name*, che è una normale stringa di caratteri C. *wanted* indica il tipo di valore atteso. La funzione restituisce *true* se il tipo effettivo della variabile è quello specificato in *wanted*, e *false* in caso contrario. In quest’ultimo caso, *risultato>val\_type* indica il tipo effettivo della variabile (si veda la [Tabella 16.1](#)).

```
awk_bool_t sym_update(const char *name, awk_value_t *valore);
```

Aggiorna la variabile il cui nome è contenuto nella stringa *name*, che è una normale stringa di caratteri C. La variabile è aggiunta alla Tabella dei simboli di *gawk*, se non è già presente. Restituisce *true* se tutto è andato bene, e *false* in caso contrario.

La modifica del tipo (da scalare a vettoriale o viceversa) di una variabile già esistente *non* è consentito, e questa routine non può neppure essere usata per aggiornare un vettore. Questa routine non può essere usata per modificare nessuna delle variabili predefinite (come *ARGC* o *NF*).

Un’estensione può andare a cercare il valore delle variabili speciali di *gawk*. Tuttavia, con l’eccezione del vettore *PROCINFO*, un’estensione non può cambiare alcuna di queste variabili.

#### 16.4.10.2 Accedere alle variabili per “cookie” e aggiornarle

Uno *scalar cookie* è un puntatore nascosto (*opaque handle*) che fornisce accesso a una variabile globale o a un vettore. Si tratta di un’ottimizzazione, per evitare di ricercare variabili nella Tabella dei simboli di *gawk* ogni volta che un accesso è necessario. Questo argomento è già stato trattato in precedenza, nella [Sezione 16.4.2 \[I tipi di dati di impiego generale\]](#), pagina 400.

Le funzioni seguenti servono per gestire gli *scalar cookie*:

```
awk_bool_t sym_lookup_scalar(awk_scalar_t cookie,
                             awk_valtype_t wanted,
                             awk_value_t *risultato);
```

Ottiene il valore corrente di uno *scalar cookie*. Una volta ottenuto lo *scalar cookie* usando `sym_lookup()`, si può usare questa funzione per accedere al valore della variabile in modo più efficiente. Restituisce *false* se il valore non è disponibile.

```
awk_bool_t sym_update_scalar(awk_scalar_t cookie, awk_value_t *valore);
```

Aggiorna il valore associato con uno *scalar cookie*. Restituisce *false* se il nuovo valore non è del tipo `AWK_STRING`, `AWK_STRNUM`, `AWK_REGEX` o `AWK_NUMBER`. Anche in questo caso, le variabili predefinite non possono essere aggiornate.

Non è immediatamente evidente come si lavora con gli *scalar cookie* o quale sia la loro vera *raison d'être*. In teoria, le routine `sym_lookup()` e `sym_update()` sono tutto ciò che occorre per lavorare con le variabili. Per esempio, ci potrebbe essere un codice che ricerca il valore di una variabile, valuta una condizione, e potrebbe poi cambiare il valore della variabile a seconda dei risultati della valutazione in modo simile a questo:

```
/* do_magic --- fai qualcosa di veramente grande */

static awk_value_t *
do_magic(int nargs, awk_value_t *risultato)
{
    awk_value_t valore;

    if ( sym_lookup("MAGIC_VAR", AWK_NUMBER, & valore)
        && qualche_condizione(valore.num_valore)) {
        valore.num_valore += 42;
        sym_update("MAGIC_VAR", & valore);
    }

    return make_number(0.0, risultato);
}
```

Questo codice sembra (ed è) semplice e immediato. Qual è il problema?

Beh, si consideri cosa succede se un qualche codice a livello di `awk` associato con l'estensione richiama la funzione `magic()` (implementata in linguaggio C da `do_magic()`), una volta per ogni record, mentre si stanno elaborando file contenenti migliaia o milioni di record. La variabile `MAGIC_VAR` viene ricercata nella Tabella dei simboli una o due volte per ogni richiamo della funzione!

La ricerca all'interno della Tabella dei simboli è in realtà una pura perdita di tempo; è molto più efficiente ottenere un *value cookie* che rappresenta la variabile, e usarlo per ottenere il valore della variabile e aggiornarlo a seconda della necessità.<sup>4</sup>

---

<sup>4</sup> La differenza è misurabile e indubbiamente reale. Fidatevi.

Quindi, la maniera per usare i valori-cookie è la seguente. Per prima cosa, la variabile di estensione va messa nella Tabella dei simboli di `gawk` usando `sym_update()`, come al solito. Poi si deve ottenere uno *scalar cookie* per la variabile usando `sym_lookup()`:

```
static awk_scalar_t magic_var_cookie;    /* cookie per MAGIC_VAR */

static void
inizializza_estensione()
{
    awk_value_t valore;

    /* immettere il valore iniziale */
    sym_update("MAGIC_VAR", make_number(42.0, & valore));

    /* ottenere il value cookie */
    sym_lookup("MAGIC_VAR", AWK_SCALAR, & valore);

    /* salvarlo per dopo */
    magic_var_cookie = valore.scalar_cookie;
    ...
}
```

Dopo aver fatto questo, si usino le routine descritte in questa sezione per ottenere e modificare il valore usando il *value cookie*. Quindi, `do_magic()` diviene ora qualcosa del tipo:

```
/* do_magic --- fai qualcosa di veramente grande */

static awk_value_t *
do_magic(int nargs, awk_value_t *risultato)
{
    awk_value_t valore;

    if ( sym_lookup_scalar(magic_var_cookie, AWK_NUMBER, & valore)
        && qualche_condizione(valore.num_valore)) {
        valore.num_valore += 42;
        sym_update_scalar(magic_var_cookie, & valore);
    }
    ...

    return make_number(0.0, risultato);
}
```

**NOTA:** Il codice appena visto omette il controllo di eventuali errori, per amor di semplicità. Il codice dell'estensione dovrebbe essere più complesso e controllare attentamente i valori restituiti dalle funzioni dell'API.

### 16.4.10.3 Creare e usare valori nascosti

Le routine in questa sezione permettono di creare e rilasciare valori nascosti. Come gli *scalar cookie*, in teoria i valori nascosti non sono necessari. Si possono creare numeri e stringhe

usando le funzioni descritte nella [Sezione 16.4.4 \[Funzioni per creare valori\]](#), pagina 404. Si possono poi assegnare quei valori a delle variabili usando `sym_update()` o `sym_update_scalar()`, come si preferisce.

Tuttavia, si può comprendere l'utilità di avere dei valori nascosti se si pone mente al fatto che la memoria di *ogni* valore di stringa *deve* essere ottenuta tramite `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`. Se ci sono 20 variabili, e tutte hanno per valore la stessa stringa, si devono creare 20 copie identiche della stringa.<sup>5</sup>

Chiaramente è più efficiente, se possibile, creare il valore una sola volta, e fare in modo che `gawk` utilizzi quell'unico valore per molte variabili. Questo è ciò che la routine in questa sezione permette di fare. Le funzioni sono le seguenti:

```
awk_bool_t create_value(awk_value_t *valore, awk_value_cookie_t *risultato);
```

Crea una stringa o un valore numerico nascosti, da `valore`, in vista di un successivo assegnamento di valore. Sono consentiti solo valori di tipo `AWK_NUMBER`, `AWK_REGEX` ed `AWK_STRING`. Ogni altro tipo è rifiutato. Il tipo `AWK_UNDEFINED` potrebbe essere consentito, ma in questo caso l'efficienza del programma ne soffrirebbe.

```
awk_bool_t release_value(awk_value_cookie_t vc);
```

Libera la memoria associata con un *value cookie* ottenuto mediante `create_value()`.

Si usano i *value cookie* in modo simile a quello con cui si usano gli *scalar cookie*. Nella routine di inizializzazione dell'estensione, si crea il *value cookie*:

```
static awk_value_cookie_t answer_cookie; /* static value cookie */

static void
inizializza_estensione()
{
    awk_value_t value;
    char *long_string;
    size_t long_string_len;

    /* codice precedente */
    ...
    /* ... riempire long_string e long_string_len ... */
    make_malloced_string(long_string, long_string_len, & value);
    create_value(& value, & answer_cookie); /* creare cookie */
    ...
}
```

Una volta che il valore è creato, si può usare come valore per un numero qualsiasi di variabili:

```
static awk_value_t *
do_magic(int nargs, awk_value_t *risultato)
{
```

---

<sup>5</sup> I valori numerici creano molti meno problemi, in quanto richiedono solo una variabile C `double` (8 byte) per contenerli.

```

awk_value_t new_value;

...    /* come in precedenza */

value.val_type = AWK_VALUE_COOKIE;
value.value_cookie = answer_cookie;
sym_update("VAR1", & value);
sym_update("VAR2", & value);
...
sym_update("VAR100", & value);
...
}

```

Usare *value cookie* in questo modo permette di risparmiare parecchia memoria, poiché tutte le variabili da VAR1 a VAR100 condividono lo stesso valore.

Ci si potrebbe chiedere, “Questa condivisione crea problemi? Cosa succede se il codice **awk** assegna un nuovo valore a VAR1; sono modificate anche tutte le altre variabili?”

Buona domanda! La risposta è che no, non è un problema. Internamente, **gawk** usa *un contatore dei riferimenti alle stringhe*. Questo significa che molte variabili possono condividere lo stesso valore di tipo stringa, e **gawk** mantiene traccia del loro uso. Quando il valore di una variabile viene modificato, **gawk** semplicemente diminuisce di uno il contatore dei riferimenti del vecchio valore, e aggiorna la variabile perché usi il nuovo valore.

Infine, come parte della pulizia al termine del programma (si veda la [Sezione 16.4.5.2 \[Registrare una funzione exit callback\], pagina 408](#)) si deve liberare ogni valore nascosto che era stato creato, usando la funzione `release_value()`.

### 16.4.11 Manipolazione di vettori

La struttura di dati primaria<sup>6</sup> in **awk** è il vettore associativo (si veda il [Capitolo 8 \[Vettori in awk\], pagina 177](#)). Le estensioni devono essere in grado di manipolare vettori **awk**. L’API fornisce varie strutture di dati per lavorare con vettori, funzioni per lavorare con singoli elementi di un vettore, e funzioni per lavorare con interi vettori. È prevista anche la possibilità di “appiattire” un vettore in modo da rendere facile a un programma scritto in C la “visita” di tutti gli elementi del vettore. Le strutture dati per i vettori sono facilmente integrabili con le strutture dati per variabili scalari, per facilitare sia l’elaborazione, sia la creazione di veri vettori di vettori (si veda la [Sezione 16.4.2 \[I tipi di dati di impiego generale\], pagina 400](#)).

#### 16.4.11.1 Tipi di dati per i vettori

I tipi di dato associati con i vettori sono i seguenti:

```
typedef void *awk_array_t;
```

Se si richiede il valore di una variabile contenuta in un vettore, si ottiene un valore del tipo `awk_array_t`. Questo valore è *opaco*<sup>7</sup> per l’estensione; identifica in maniera univoca il vettore ma può solo essere usato come parametro di una funzione dell’API, o essere ricevuto da una funzione dell’API. Questo è molto

<sup>6</sup> D’accordo, l’unica struttura di dati.

<sup>7</sup> È anche un “cookie,” ma gli sviluppatori di **gawk** hanno preferito non abusare di questo termine.

simile al modo in cui i valori 'FILE \*' sono usati con le routine di libreria di `<stdio.h>`.

```
typedef struct awk_element {
    /* puntatore di servizio
    a lista collegata, non usato da gawk */
    struct awk_element *next;
    enum {
        AWK_ELEMENT_DEFAULT = 0, /* impostato da gawk */
        AWK_ELEMENT_DELETE = 1  /* impostato dall'estensione */
    } flags;
    awk_value_t index;
    awk_value_t value;
} awk_element_t;
```

`awk_element_t` è un elemento di vettore “appiattito”. `awk` produce un vettore di questo tipo all'interno della struttura `awk_flat_array_t` (si veda poco più avanti). Singoli elementi di vettore possono essere marcati per essere cancellati. Nuovi elementi del vettore devono essere aggiunti individualmente, uno per volta, usando una funzione API apposita. I campi sono i seguenti:

`struct awk_element *next;`

Questo puntatore è presente come ausilio a chi scrive un'estensione. Permette a un'estensione di creare una lista collegata (*linked list*) di nuovi elementi che possono essere aggiunti a un vettore con un singolo ciclo che percorre tutta la lista.

`enum { ... } flags;`

Un insieme di valori di flag che passano informazione tra l'estensione e `gawk`. Per ora c'è solo un flag disponibile: `AWK_ELEMENT_DELETE`. Se lo si imposta, `gawk` elimina l'elemento in questione dal vettore originale, dopo che il vettore “appiattito” è stato rilasciato.

`index`

`value` L'indice e il valore di un elemento, rispettivamente. *Tutta* la memoria puntata da `index` e `valore` appartiene a `gawk`.

```
typedef struct awk_flat_array {
    awk_const void *awk_const opaque1; /* per uso di gawk */
    awk_const void *awk_const opaque2; /* per uso di gawk */
    awk_const size_t count; /* quanti elementi nel vettore */
    awk_element_t elements[1]; /* saranno ‘‘appiattiti’’ */
} awk_flat_array_t;
```

Questo è un vettore appiattito. Quando un'estensione ottiene da `gawk` questa struttura, il vettore `elements` ha una dimensione reale di `count` elementi. I puntatori `opaque1` e `opaque2` sono per uso di `gawk`; come tali, sono marcati come `awk_const` in modo che l'estensione non possa modificarli.

### 16.4.11.2 Funzioni per lavorare coi vettori

Le funzioni seguenti permettono di gestire singoli elementi di un vettore:

```
awk_bool_t get_element_count(awk_array_t a_cookie, size_t *count);
```

Per il vettore rappresentato da `a_cookie`, restituisce in `*count` il numero di elementi in esso contenuti. Ogni sottovettore è conteggiato come se fosse un solo elemento. Restituisce *false* se si verifica un errore.

```
awk_bool_t get_array_element(awk_array_t a_cookie,
                             const awk_value_t *const index,
                             awk_valtype_t wanted,
                             awk_value_t *risultato);
```

Per il vettore rappresentato da `a_cookie`, restituisce in `*risultato` il valore dell'elemento il cui indice è `index`. `wanted` specifica il tipo di valore che si vuole ritrovare. Restituisce *false* se `wanted` non coincide con il tipo di dato o se `index` non è nel vettore (si veda la [Tabella 16.1](#)).

Il valore per `index` può essere numerico, nel qual caso `gawk` lo converte in una stringa. Usare valori non interi è possibile, ma richiede di comprendere il modo con cui tali valori sono convertiti in stringhe (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121); per questo motivo, è meglio usare numeri interi.

Come per *tutte* le stringhe passate a `gawk` da un'estensione, la memoria che contiene il valore della stringa con chiave `index` deve essere stata acquisita utilizzando le funzioni `gawk_malloc()`, `gawk_calloc()` o `gawk_realloc()`, e `gawk` rilascerà (al momento opportuno) la relativa memoria.

```
awk_bool_t set_array_element(awk_array_t a_cookie,
                             const awk_value_t *const index,
                             const awk_value_t *const value);
```

Nel vettore rappresentato da `a_cookie`, crea o modifica l'elemento il cui indice è contenuto in `index`. I vettori `ARGV` ed `ENVIRON` non possono essere modificati, mentre il vettore `PROCINFO` è modificabile.

```
awk_bool_t set_array_element_by_elem(awk_array_t a_cookie,
                                     awk_element_t element);
```

Come `set_array_element()`, ma prende l'indice `index` e il valore `value` da `element`. Questa è una macro di utilità.

```
awk_bool_t del_array_element(awk_array_t a_cookie,
                             const awk_value_t* const index);
```

Elimina dal vettore, rappresentato da `a_cookie`, l'elemento con l'indice specificato. Restituisce *true* se l'elemento è stato rimosso o *false* se l'elemento non era presente nel vettore.

Le seguenti funzioni operano sull'intero vettore:

```
awk_array_t create_array(void);
```

Crea un nuovo vettore a cui si possono aggiungere elementi. Si veda la [Sezione 16.4.11.4 \[Come creare e popolare vettori\]](#), pagina 427, per una trattazione su come creare un nuovo vettore e aggiungervi elementi.

```
awk_bool_t clear_array(awk_array_t a_cookie);
```

Svuota il vettore rappresentato da `a_cookie`. Restituisce *false* in presenza di qualche tipo di problema, *true* in caso contrario. Il vettore non viene eliminato

ma, dopo aver chiamato questa funzione, resta privo di elementi. Questo è equivalente a usare l'istruzione `delete` (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187).

```
awk_bool_t flatten_array_typed(awk_array_t a_cookie, awk_flat_array_t
**data, awk_valtype_t index_type, awk_valtype_t value_type);
```

Per il vettore rappresentato da `a_cookie`, crea una struttura `awk_flat_array_t` e la riempi con indici e valori del tipo richiesto. Imposta il puntatore il cui indirizzo è passato in `data` per puntare a questa struttura. Restituisce *true* se tutto va bene o *false* in caso contrario. Si veda la [Sezione 16.4.11.3 \[Lavorare con tutti gli elementi di un vettore\]](#), pagina 424, per una trattazione su come appiattare un vettore per poterci lavorare.

```
awk_bool_t flatten_array(awk_array_t a_cookie, awk_flat_array_t **data);
```

Per il vettore rappresentato da `a_cookie`, crea una struttura `awk_flat_array_t` e la riempi con indici di tipo `AWK_STRING` e valori `AWK_UNDEFINED`. Questa funzione è resa obsoleta da `flatten_array_typed()`. È fornita come macro, e mantenuta per convenienza e per compatibilità a livello di codice sorgente con la precedente versione dell'API.

```
awk_bool_t release_flattened_array(awk_array_t a_cookie,
                                   awk_flat_array_t *data);
```

Quando si è finito di lavorare con un vettore appiattito, si liberi la memoria usando questa funzione. Occorre fornire sia il cookie del vettore originale, sia l'indirizzo della struttura da liberare, `awk_flat_array_t`. La funzione restituisce *true* se tutto va bene, *false* in caso contrario.

### 16.4.11.3 Lavorare con tutti gli elementi di un vettore

*Appiattare* un vettore vuol dire creare una struttura che rappresenta l'intero vettore in modo da facilitare la visita dell'intero vettore da parte del codice in C. Parte del codice in `extension/testtext.c` fa questo, ed è anche un bell'esempio di come utilizzare l'API.

Questa parte del codice sorgente sarà descritta un po' per volta. Ecco, per iniziare, lo script `gawk` che richiama l'estensione di test:

```
@load "testtext"
BEGIN {
    n = split("blacky rusty sophie raincloud lucky", pets)
    printf("pets ha %d elementi\n", length(pets))
    ret = dump_array_and_delete("pets", "3")
    printf("dump_array_and_delete(pets) ha restituito %d\n", ret)
    if ("3" in pets)
        printf("dump_array_and_delete() NON ha rimosso l'indice \"3\"!\n")
    else
        printf("dump_array_and_delete() ha rimosso l'indice \"3\"!\n")
    print ""
}
```

Questo codice crea un vettore usando la funzione `split()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198) e poi chiama `dump_array_and_delete()`. Questa funzione ricerca il vettore il cui nome è passato come primo argomento, ed elimina

l'elemento il cui indice è passato come secondo argomento. Il codice `awk` stampa poi il valore restituito e controlla che l'elemento sia stato effettivamente cancellato. Ecco il codice C che costituisce la funzione `dump_array_and_delete()`. È stato leggermente modificato per facilitare l'esposizione.

La prima parte dichiara variabili, imposta il valore di ritorno di default in `risultato`, e controlla che la funzione sia stata chiamata con il numero corretto di argomenti:

```
static awk_value_t *
dump_array_and_delete(int nargs, awk_value_t *risultato)
{
    awk_value_t valore, valore2, valore3;
    awk_flat_array_t *flat_array;
    size_t count;
    char *name;
    int i;

    assert(risultato != NULL);
    make_number(0.0, risultato);

    if (nargs != 2) {
        printf("dump_array_and_delete: nargs errato "
              "(%d dovrebbe essere 2)\n", nargs);
        goto out;
    }
}
```

La funzione poi prosegue un passo per volta, come segue. Il primo passo è recuperare il nome del vettore, passato come primo argomento, seguito dal vettore stesso. Se una di queste operazioni non riesce, viene stampato un messaggio di errore e si ritorna al chiamante:

```
/* trasforma in un vettore piatto il vettore
passato come argomento e lo stampa */
if (get_argument(0, AWK_STRING, &value)) {
    name = valore.str_value.str;
    if (sym_lookup(name, AWK_array, &value2))
        printf("dump_array_and_delete: sym_lookup di %s effettuato\n",
              name);
    else {
        printf("dump_array_and_delete: sym_lookup di %s non riuscito\n",
              name);
        goto out;
    }
} else {
    printf("dump_array_and_delete: get_argument(0) non riuscito\n");
    goto out;
}
```

Per controllo, e per assicurarsi che il codice C veda lo stesso numero di elementi del codice `awk`, il secondo passo è quello di ottenere il numero di elementi nel vettore e stamparlo:

```
if (! get_element_count(valore2.array_cookie, &count)) {
```

```

        printf("dump_array_and_delete: get_element_count non riuscito\n");
        goto out;
    }

```

```

        printf("dump_array_and_delete: il vettore in input ha %lu elementi\n",
               (unsigned long) count);

```

Il terzo passo è quello di appiattire il vettore, e quindi controllare che il numero di elementi nella struttura `awk_flat_array_t` sia uguale a quello appena trovato:

```

        if (! flatten_array_typed(valore2.array_cookie, & flat_array,
                                  AWK_STRING, AWK_UNDEFINED)) {
            printf("dump_array_and_delete: non sono riuscito ad appiattire \
il vettore\n");
            goto out;
        }

```

```

        if (flat_array->count != count) {
            printf("dump_array_and_delete: flat_array->count (%lu)"
                   " != count (%lu)\n",
                   (unsigned long) flat_array->count,
                   (unsigned long) count);
            goto out;
        }

```

Il quarto passo è ritrovare l'indice dell'elemento da eliminare, che era stato passato come secondo argomento. Va tenuto presente che i contatori di argomenti passati a `get_argument()` partono da zero, e che quindi il secondo argomento è quello numero uno:

```

        if (! get_argument(1, AWK_STRING, & value3)) {
            printf("dump_array_and_delete: get_argument(1) non riuscito\n");
            goto out;
        }

```

Il quinto passo è quello in cui si fa il “vero lavoro”. La funzione esegue un ciclo su ogni elemento nel vettore, stampando i valori degli indici e degli elementi. Inoltre, dopo aver trovato, tramite l'indice, l'elemento che si vorrebbe eliminare, la funzione imposta il *bit* `AWK_ELEMENT_DELETE` nel campo `flags` dell'elemento. Quando il vettore è stato interamente percorso, `gawk` visita il vettore appiattito, ed elimina ogni elemento in cui il relativo *bit* della flag sia impostato:

```

        for (i = 0; i < flat_array->count; i++) {
            printf("\t%s[\"%.*s\"] = %s\n",
                   name,
                   (int) flat_array->elements[i].index.str_value.len,
                   flat_array->elements[i].index.str_value.str,
                   valrep2str(& flat_array->elements[i].valore));

            if (strcmp(valore3.str_value.str,
                      flat_array->elements[i].index.str_value.str) == 0) {
                flat_array->elements[i].flags |= AWK_ELEMENT_DELETE;
            }
        }

```

```

        printf("dump_array_and_delete: ho marcato l'elemento \"%s\" "
               "per eliminazione\n",
               flat_array->elements[i].index.str_value.str);
    }
}

```

Il sesto passo è liberare il vettore appiattito. Questo segnala a **gawk** che l'estensione non sta più usando il vettore, e che dovrebbe eliminare gli elementi marcati per l'eliminazione. **gawk** libera anche ogni area di memoria che era stata allocata, e quindi non si dovrebbe più usare il puntatore (`flat_array` in questo codice) dopo aver chiamato `release_flattened_array()`:

```

    if (! release_flattened_array(valore2.array_cookie, flat_array)) {
        printf("dump_array_and_delete: non riesco a liberare \
il vettore appiattito\n");
        goto out;
    }

```

Infine, poiché tutto è andato bene, la funzione imposta il codice di ritorno a "successo", e lo restituisce quando esce:

```

    make_number(1.0, risultato);
out:
    return risultato;
}

```

Ecco l'output ottenuto eseguendo questa parte del test:

```

pets ha 5 elementi
dump_array_and_delete: sym_lookup di pets effettuato
dump_array_and_delete: il vettore in input ha 5 elementi
    pets["1"] = "blacky"
    pets["2"] = "rusty"
    pets["3"] = "sophie"
dump_array_and_delete: ho marcato l'elemento "3" per eliminazione
    pets["4"] = "raincloud"
    pets["5"] = "lucky"
dump_array_and_delete(pets) ha restituito 1
dump_array_and_delete() ha rimosso l'indice "3"!

```

#### 16.4.11.4 Come creare e popolare vettori

Oltre a lavorare con vettori creati da codice **awk**, si possono creare vettori a cui aggiungere elementi secondo le esigenze, che poi il codice **awk** può utilizzare e manipolare.

Ci sono due punti importanti da tener presente quando di creano vettori dal codice di un'estensione:

- Il vettore appena creato deve essere subito messo nella Tabella dei simboli di **gawk**. Solo dopo aver fatto questo è possibile aggiungere elementi al vettore.

Analogamente, se si installa un nuovo vettore come sottovettore di un vettore già esistente, occorre prima aggiungere il nuovo vettore al suo "genitore" per poter poi aggiungere degli elementi allo stesso.

Quindi, il modo giusto per costruire un vettore è di lavorare “dall’alto verso il basso”. Creare il vettore, e subito aggiungerlo alla Tabella dei simboli di **gawk** usando **sym\_update()**, o installarlo come elemento in un vettore già esistente usando **set\_array\_element()**. Un esempio di codice è fornito più sotto.

- Per come funziona internamente **gawk**, dopo aver usato **sym\_update()** per definire un vettore in **gawk**, si deve innanzitutto recuperare il *cookie* del vettore dal valore passato a **sym\_update()**, in questo modo:

```
awk_value_t val;
awk_array_t new_array;

new_array = create_array();
val.val_type = AWK_ARRAY;
val.array_cookie = new_array;

/* aggiunge il vettore alla Tabella dei simboli */
sym_update("array", & val);

new_array = val.array_cookie;    /* QUESTO È OBBLIGATORIO */
```

Se si sta installando un vettore come sottovettore, occorre anche recuperare il *cookie* del vettore dopo aver chiamato **set\_element()**.

Il seguente codice C è una semplice estensione di test per creare un vettore con due elementi normali e con un sottovettore. Le direttive iniziali **#include** e le solite dichiarazione di variabili sono state omesse per amor di brevità (si veda la [Sezione 16.4.14 \[Codice predefinito di interfaccia API\], pagina 432](#)). Il primo passo è creare un nuovo vettore e poi aggiungerlo alla Tabella dei simboli:

```
/* create_new_array --- creare un vettore denominato */

static void
create_new_array()
{
    awk_array_t a_cookie;
    awk_array_t sottovettore;
    awk_value_t index, valore;

    a_cookie = create_array();
    valore.val_type = AWK_array;
    valore.array_cookie = a_cookie;

    if (! sym_update("new_array", & valore))
        printf("create_new_array: sym_update(\"nuovo_vettore\") \
non riuscito!\n");
    a_cookie = valore.array_cookie;
```

Si noti come **a\_cookie** è reimpostato dal campo **array\_cookie** nella struttura **valore**.

Il secondo passo aggiunge due elementi normali a **nuovo\_vettore**:

```
(void) make_const_string("salve", 5, & index);
```

```

(void) make_const_string("mondo", 5, & value);
if (! set_array_element(a_cookie, & index, & value)) {
    printf("fill_in_array: set_array_element non riuscito\n");
    return;
}

(void) make_const_string("risposta", 8, & index);
(void) make_number(42.0, & value);
if (! set_array_element(a_cookie, & index, & value)) {
    printf("fill_in_array: set_array_element non riuscito\n");
    return;
}

```

Il terzo passo è creare il sottovettore e aggiungerlo al vettore:

```

(void) make_const_string("sottovettore", 12, & index);
sottovettore = create_array();
valore.val_type = AWK_array;
valore.array_cookie = subarray;
if (! set_array_element(a_cookie, & index, & value)) {
    printf("fill_in_array: set_array_element non riuscito\n");
    return;
}
sottovettore = valore.array_cookie;

```

Il passo finale è di aggiungere al sottovettore un suo proprio elemento:

```

(void) make_const_string("pippo", 5, & index);
(void) make_const_string("pluto", 5, & value);
if (! set_array_element(sottovettore, & index, & value)) {
    printf("fill_in_array: set_array_element non riuscito\n");
    return;
}
}

```

Ecco uno script di esempio che carica l'estensione e quindi stampa il valore di tutti gli elementi del vettore, invocando nuovamente se stessa nel caso che un particolare elemento sia a sua volta un vettore:

```

@load "subarray"

function dumparray(name, vettore, i)
{
    for (i in vettore)
        if (isarray(vettore[i]))
            dumparray(name "[" i "]", vettore[i])
        else
            printf("%s[%s] = %s\n", name, i, vettore[i])
}

BEGIN {

```

```
    dumparray("new_array", new_array);
}
```

Ecco il risultato dell'esecuzione dello script:

```
$ AWKLIBPATH=$PWD ./gawk -f subarray.awk
+ new_array["sottovettore"]["pippo"] = pluto
+ new_array["salve"] = mondo
+ new_array["risposta"] = 42
```

(Si veda la [Sezione 16.5 \[Come gawk trova le estensioni compilate\]](#), pagina 434, per ulteriori dettagli sulla variabile d'ambiente AWKLIBPATH.)

### 16.4.12 Accedere alle ridirezioni e modificarle

La seguente funzione consente alle estensioni di accedere e di manipolare delle ridirezioni.

```
awk_bool_t get_file(const char *name,
                    size_t name_len,
                    const char *tipofile,
                    int fd,
                    const awk_input_buf_t **ibufp,
                    const awk_output_buf_t **obufp);
```

Ricerca il file `name` nella tabella interna di ridirezione di `gawk`. Se `name` è `NULL` o `name_len` è zero, restituisce i dati del file in input correntemente aperto il cui nome è memorizzato in `FILENAME`. (Questa chiamata non usa l'argomento `filetype`, che, quindi, può essere lasciato indefinito). Se il file non è già aperto, tenta di aprirlo. L'argomento `filetype` deve terminare con uno zero binario, e dovrebbe avere uno di questi valori:

">"	Un file aperto in output.
">>"	Un file aperto in output, record aggiunti a fine file, dopo quelli già esistenti [ <i>append</i> ].
"<"	Un file aperto in input.
" >"	Una <i>pipe</i> aperta in output.
" <"	Una <i>pipe</i> aperta in input.
" &"	Un coprocesso bidirezionale.

In caso di errore, restituisce il valore `awk_false`. Altrimenti, restituisce `awk_true`, insieme a ulteriori informazioni sulla ridirezione nei puntatori `ibufp` e `obufp`. Per ridirezioni di input il valore `*ibufp` non dovrebbe essere `NULL`, mentre `*obufp` dovrebbe essere `NULL`. Per ridirezioni di output, il valore di `*obufp` non dovrebbe essere `NULL`, e `*ibufp` dovrebbe essere `NULL`. Per coprocessi bidirezionali, nessuno dei due valori dovrebbe essere `NULL`.

Normalmente, l'estensione è interessata a `(*ibufp)->fd` e/o `fileno((*obufp)->fp)`. Se il file non è già aperto, e l'argomento `fd` non è negativo, `gawk` userà quel descrittore di file invece di aprire il file nella maniera solita. Se l'`fd` non è negativo, ma il file esiste già, `gawk` ignora l'`fd` e restituisce il file esistente. È responsabilità del chiamante notare che né l'`fd`

nella struttura restituita `awk_input_buf_t`, né `fd` nella struttura restituita `awk_output_buf_t` contiene il valore richiesto.

Si noti che fornire un descrittore di file *non* è al momento supportato per le *pipe*. Tuttavia, l'utilizzo di un descrittore di file dovrebbe essere possibile per *socket* in input, output, aggiunta-a-fine-file (append), e bidirezionale (coprocessi). Se `filetype` è bidirezionale, **gawk** presuppone che sia un *socket*! Si noti che nel caso bidirezionale i descrittori di file in input e output possono essere differenti. Per essere sicuri che tutto sia andato bene, si deve controllare che uno dei due corrisponda alla richiesta.

Si prevede che questa funzione API verrà usata per parallelizzare l'I/O e rendere disponibile una libreria per i *socket*.

### 16.4.13 Variabili fornite dall'API

L'API fornisce due insiemi di variabili. Il primo insieme contiene informazioni sulla versione dell'API (sia la versione dell'estensione compilata, che quella di **gawk**). Il secondo insieme contiene informazioni su come **gawk** è stato invocato.

#### 16.4.13.1 Costanti e variabili della versione dell'API

L'API fornisce sia un numero di versione “principale” che uno “secondario”. Le versioni dell'API sono disponibili al momento della compilazione, come definizioni per il preprocessore C, a supporto della compilazione condizionale, e come elencazione di costanti per facilitare il debug:

versione API	Definiz. Preprocessore C	Costante di elenco
Major	<code>gawk_api_major_version</code>	<code>GAWK_API_MAJOR_VERSION</code>
Minor	<code>gawk_api_minor_version</code>	<code>GAWK_API_MINOR_VERSION</code>

Tabella 16.2: Costanti delle versioni API gawk

La versione secondaria aumenta quando nuove funzioni sono aggiunte all'API. Tali nuove funzioni sono sempre aggiunte alla fine della `struct` dell'API.

La versione principale aumenta (e la versione secondaria torna a zero) se qualche tipo di dati cambia dimensione o si modifica l'ordine dei campi, o se qualcuna delle funzioni esistenti cambia il livello di versione.

Può capitare che un'estensione sia stata compilata con una versione dell'API ma caricata da una versione di **gawk** che ne usa una differente. Per questo motivo, la versione principale e quella secondaria dell'API della versione in uso di **gawk** sono incluse nella `struct` dell'API come costanti intere in sola lettura:

```
api->major_version
```

La versione principale di **gawk** in esecuzione.

```
api->minor_version
```

La versione secondaria di **gawk** in esecuzione.

Dipende dall'estensione decidere se ci sono incompatibilità con l'API. Tipicamente, basta un controllo di questo tipo:

```
if (api->major_version != GAWK_API_MAJOR_VERSION
```

```

    || api->minor_version < GAWK_API_MINOR_VERSION) {
        fprintf(stderr, "estensione_pippo: discordanza di versione \
con gawk!\n");
        fprintf(stderr, "\tLa mia versione (%d, %d), versione gawk \
(%d, %d)\n",
                GAWK_API_MAJOR_VERSION, GAWK_API_MINOR_VERSION,
                api->major_version, api->minor_version);
        exit(1);
    }

```

Questo codice è incluso nella macro generica `dl_load_func()` presente in `gawkapi.h` (trattata nella [Sezione 16.4.14 \[Codice predefinito di interfaccia API\]](#), pagina 432).

### 16.4.13.2 Variabili informative

L'API fornisce accesso a parecchie variabili che descrivono se le opzioni della riga di comando corrispondenti sono state specificate quando `gawk` è stato chiamato. Le variabili sono:

`do_debug` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--debug`.

`do_lint` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--lint`.

`do_mpfr` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--bignum`.

`do_profile` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--profile`.

`do_sandbox` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--sandbox`.

`do_traditional` Questa variabile è *true* se `gawk` è stato invocato con l'opzione `--traditional`.

Il valore di `do_lint` può cambiare se il codice `awk` modifica la variabile predefinita `LINT` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162). Gli altri valori non dovrebbero cambiare durante l'esecuzione.

### 16.4.14 Codice predefinito di interfaccia API

Come già detto (si veda la [Sezione 16.3 \[Una panoramica sul funzionamento ad alto livello\]](#), pagina 396), le definizioni di funzioni qui presentate sono in realtà delle macro. Per usare queste macro, l'estensione deve fornire una piccola quantità di codice predefinito (variabili e funzioni) nella parte iniziale del file sorgente, usando dei nomi standard, come descritto qui sotto. Il codice predefinito in questione è anche descritto nel file di intestazione `gawkapi.h`:

```

/* Codice predefinito: */
int plugin_is_GPL_compatible;

static gawk_api_t *const api;
static awk_ext_id_t ext_id;
static const char *ext_version = NULL; /* o ... = "qualche stringa" */

static awk_ext_func_t func_table[] = {
    { "name", do_name, 1, 0, awk_false, NULL },

```

```

    /* ... */
};

/* 0: */

static awk_bool_t (*init_func)(void) = NULL;

/* OPPURE: */

static awk_bool_t
init_mia_estensione(void)
{
    ...
}

static awk_bool_t (*init_func)(void) = init_mia_estensione;

dl_load_func(func_table, qualche_nome, "name_space_in_quotes")

```

Queste variabili e funzioni sono:

```
int plugin_is_GPL_compatible;
```

Qui si dichiara che l'estensione è compatibile con la licenza GNU GPL (si veda la [Licenza Pubblica Generale GNU \(GPL\)](#), pagina 529).

Se l'estensione non ha questa variabile, non verrà caricata da gawk (si veda la [Sezione 16.2 \[Tipo di licenza delle estensioni\]](#), pagina 395).

```
static gawk_api_t *const api;
```

Questa variabile globale **static** dovrebbe essere impostata per puntare al puntatore **gawk\_api\_t** che gawk passa alla funzione (dell'estensione) **dl\_load()**. Questa variabile è usata da tutte le macro.

```
static awk_ext_id_t ext_id;
```

Questa variabile globale **static** dovrebbe essere impostata al valore **awk\_ext\_id\_t** che gawk passa alla funzione **dl\_load()**. Questa variabile è usata da tutte le macro.

```
static const char *ext_version = NULL; /* o ... = "qualche stringa" */
```

Questa variabile globale **static** dovrebbe essere impostata a **NULL** oppure puntare a una stringa che contiene il nome e la versione dell'estensione.

```
static awk_ext_func_t func_table[] = { ... };
```

Questo è un vettore di una o più strutture **awk\_ext\_func\_t**, come descritto in precedenza (si veda la [Sezione 16.4.5.1 \[Registrare funzioni di estensione\]](#), pagina 406). Può essere usato in seguito per più chiamate a **add\_ext\_func()**.

```
static awk_bool_t (*init_func)(void) = NULL;
```

*OR*

```
static awk_bool_t init_mia_estensione(void) { ... }
static awk_bool_t (*init_func)(void) = init_mia_estensione;
```

Se qualche lavoro di inizializzazione è necessario, si dovrebbe definire una funzione all'uopo (crea variabili, apre file, etc.) e poi definire il puntatore `init_func` che punti alla funzione stessa. La funzione dovrebbe restituire `awk_false` se non va a buon fine o `awktrue` se tutto va bene.

Se un'inizializzazione non è necessaria, si definisca il puntatore e lo si inizializzi a `NULL`.

```
dl_load_func(func_table, qualche_nome, "nome_spazio_tra_doppi_apici")
```

Questa macro genera una funzione `dl_load()` che farà tutte le inizializzazioni necessarie.

Lo scopo di tutte le variabili e dei vettori è di far sì che la funzione `dl_load()` (richiamata dalla macro `dl_load_func()`) faccia tutto il lavoro standard necessario, qui descritto:

1. Controlla le versioni dell'API. Se la versione principale dell'estensione non corrisponde a quella di `gawk` o se la versione secondaria dell'estensione è maggiore di quella di `gawk`, stampa un messaggio di errore fatale ed esce.
2. Carica le funzioni definite in `func_table`. Se qualche caricamento non riesce, stampa un messaggio di avvertimento ma continua l'esecuzione.
3. Se il puntatore `init_func` non è `NULL`, chiama la funzione da esso puntata. Se questa restituisce `awk_false`, stampa un messaggio di avvertimento.
4. Se `ext_version` non è `NULL`, registra la stringa di versione con `gawk`.

### 16.4.15 Modifiche dalla versione 1 dell'API

La versione API corrente *non* è compatibile a livello binario con la versione 1 dell'API. Le funzioni di estensione vanno ricomilate per poterle usare con la versione corrente di `gawk`.

Fortunatamente, fatti salvi alcuni possibili avvertimenti a livello di compilazione, l'API rimane compatibile a livello di codice sorgente con la precedente versione API. Le differenze più rilevanti sono gli ulteriori campi nella struttura `awk_ext_func_t`, e l'aggiunta del terzo argomento nella funzione di implementazione in linguaggio C.

## 16.5 Come gawk trova le estensioni compilate

Le estensioni compilate vanno installate in una directory dove `gawk` possa trovarle. Se `gawk` è configurato e installato nella maniera di default, la directory dove trovare le estensioni è `/usr/local/lib/gawk`. Si può anche specificare un percorso di ricerca contenente una lista di directory da esaminare per la ricerca di estensioni compilate. Si veda la [Sezione 2.5.2 \[Ricerca di librerie condivise awk su varie directory.\]](#), pagina 43, per ulteriori dettagli.

## 16.6 Esempio: alcune funzioni per i file

*In qualunque posto vai, là tu sei.*

—Buckaroo Banzai

Due utili funzioni che non sono in `awk` sono `chdir()` (per permettere a un programma `awk` di cambiare directory di lavoro) e `stat()` (per far sì che un programma `awk` possa

raccogliere informazioni su un dato file). Per illustrare l'azione dell'API, questa sezione fornisce queste funzioni a **gawk** in un'estensione.

### 16.6.1 Usare `chdir()` e `stat()`

Questa sezione mostra come usare le nuove funzioni a livello di **awk** una volta che siano state integrate nell'interprete del programma **gawk** in esecuzione. Usare `chdir()` è molto semplice. Richiede un solo argomento, la nuova directory su cui posizionarsi:

```
@load "filefuncs"
...
newdir = "/home/arnold/funstuff"
ret = chdir(newdir)
if (ret < 0) {
    printf("non riesco a passare a %s: %s\n", newdir, ERRNO) > "/dev/stderr"
    exit 1
}
...
```

Il valore restituito è negativo se la chiamata a `chdir()` non è riuscita, ed `ERRNO` (si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)) è impostato a una stringa che descrive l'errore.

Usare `stat()` è un po' più complicato. La funzione scritta in C `stat()` riempie una struttura che ha una certa quantità di informazioni. La maniera corretta per immagazzinarle in **awk** è quella di riempire un vettore associativo con le informazioni appropriate:

```
file = "/home/arnold/.profile"
ret = stat(file, fdata)
if (ret < 0) {
    printf("non è stato possibile eseguire stat per %s: %s\n",
           file, ERRNO) > "/dev/stderr"
    exit 1
}
printf("dimensione di %s è %d byte\n", file, fdata["size"])
```

La funzione `stat()` svuota sempre il vettore che contiene i dati, anche nel caso che la chiamata a `stat()` non riesca. I seguenti elementi vengono restituiti dalla funzione:

"name"	Il nome del file oggetto della chiamata a <code>stat()</code> .
"dev"	
"ino"	I numeri di <i>device</i> e di <i>inode</i> , rispettivamente.
"mode"	Il modo del file, in formato numerico. Questo include sia il tipo di file che i suoi permessi di accesso.
"nlink"	Il numero di collegamenti fisici del file (stesso file con diversi nomi).
"uid"	
"gid"	Gli identificativi di utente e di gruppo del possessore del file.
"size"	La dimensione in byte del file.
"blocks"	Il numero di blocchi su disco realmente occupati dal file. Questo può non essere proporzionale alla dimensione del file se il file ha delle lacune [ossia se solo alcune parti del file esistono veramente, il resto non è ancora stato riempito].

"atime"	
"mtime"	
"ctime"	La data e ora dell'ultimo accesso, modifica, e aggiornamento dell' <i>inode</i> , rispettivamente. Questi sono delle marcature temporali numeriche (misurate in secondi dal 01 gennaio 1970), che possono essere formattate dalla funzione <code>strftime()</code> (si veda la <a href="#">Sezione 9.1.5 [Funzioni per gestire marcature temporali]</a> , pagina 214).
"pmode"	La modalità stampabile ("printable mode") del file. Questo è una stringa che rappresenta il tipo del file e i permessi di accesso, come sono visualizzati da <code>'ls -l'</code> —per esempio, <code>"drwxr-xr-x"</code> .
"type"	Una stringa stampabile che descrive il tipo di file. Il valore è uno dei seguenti: <ul style="list-style-type: none"> <li>"blockdev" "chardev" Il file è un dispositivo a blocchi o a caratteri ("file speciale").</li> <li>"directory" Il file è una directory.</li> <li>"fifo" Il file è una <i>pipe</i> denominata (nota anche come FIFO [First In First Out]).</li> <li>"file" Il file è un file normale.</li> <li>"socket" Il file è un <i>socket</i> AF_UNIX ("Unix domain") nel filesystem.</li> <li>"symlink" Il file è un collegamento simbolico.</li> </ul>
"devbsize"	La dimensione di un blocco per l'elemento indicizzato da <code>"blocks"</code> . Questa informazione è derivata dalla costante <code>DEV_BSIZE</code> definita in <code>&lt;sys/param.h&gt;</code> nella maggior parte dei sistemi, o dalla costante <code>S_BLKSIZE</code> in <code>&lt;sys/stat.h&gt;</code> nei sistemi BSD. Per alcuni altri sistemi il valore si basa su una conoscenza <i>a priori</i> delle caratteristiche di un particolare sistema. Se non si riesce a determinare il valore, viene restituito quello di default, che è 512.
Possono essere presenti diversi altri elementi, a seconda del sistema operativo e del tipo di file. Si può controllarne la presenza dal programma <code>awk</code> per mezzo dell'operatore <code>in</code> (si veda la <a href="#">Sezione 8.1.2 [Come esaminare un elemento di un vettore]</a> , pagina 179):	
"blksize"	La dimensione preferita di un blocco per effettuare operazioni di I/O sul file. Questo campo non è presente nella struttura C <code>stat</code> di tutti i sistemi che rispettano lo standard POSIX.
"linkval"	Se il file è un collegamento simbolico, questo elemento è il nome del file puntato dal collegamento simbolico (cioè, il valore del collegamento).
"rdev"	
"major"	
"minor"	Se il file è un dispositivo a blocchi o a caratteri, questi valori rappresentano il numero del dispositivo e, rispettivamente, le componenti principale e secondaria di quel numero.

### 16.6.2 Codice C per eseguire `chdir()` e `stat()`

Questo è il codice C per queste estensioni.<sup>8</sup>

Il file include alcuni file di intestazione standard, e poi il file di intestazione `gawkapi.h`, che fornisce le definizioni dell'API. A queste seguono le dichiarazioni di variabili, necessarie per usare le macro dell'API e il codice predefinito (si veda la [Sezione 16.4.14 \[Codice predefinito di interfaccia API\]](#), pagina 432):

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <assert.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>

#include "gawkapi.h"

#include "gettext.h"
#define _(msgid) gettext(msgid)
#define N_(msgid) msgid

#include "gawkfts.h"
#include "stack.h"

static const gawk_api_t *api;      /* per consentire il funzionamento
                                   delle macro di utilità */

static awk_ext_id_t *ext_id;
static awk_bool_t init_filefuncs(void);
static awk_bool_t (*init_func)(void) = init_filefuncs;
static const char *ext_version = "filefuncs extension: version 1.0";

int plugin_is_GPL_compatible;

/* do_chdir --- fornisce funzione chdir()
               caricata dinamicamente per gawk */
```

---

<sup>8</sup> La versione qui presentata è lievemente modificata per amor di semplicità. Si veda `extension/filefuncs.c` nella distribuzione `gawk` per la versione completa.

```
static awk_value_t *
do_chdir(int nargs, awk_value_t *risultato, struct awk_ext_func *non_usata)
{
    awk_value_t newdir;
    int ret = -1;

    assert(risultato != NULL);
```

La variabile `newdir` rappresenta la nuova directory nella quale cambiare, che è ottenuta tramite la funzione `get_argument()`. Si noti che il primo argomento è quello numero zero.

Se l'argomento è stato trovato con successo, la funzione invoca la chiamata di sistema `chdir()`. In caso contrario, se la `chdir()` non riesce, viene aggiornata la variabile `ERRNO`:

```
if (get_argument(0, AWK_STRING, & newdir)) {
    ret = chdir(newdir.str_value.str);
    if (ret < 0)
        update_ERRNO_int(errno);
}
```

Infine, la funzione restituisce il codice di ritorno da `chdir` a livello di `awk`:

```
return make_number(ret, risultato);
}
```

L'estensione `stat()` è più impegnativa. Dapprima abbiamo una funzione che trasforma la stringa di autorizzazione numerica (*mode*) in una rappresentazione stampabile (p.es., il codice ottale 0644 diviene `'-rw-r--r--'`). Questa parte è qui omessa per brevità.

```
/* format_mode --- trasforma il campo mode di stat
   in qualcosa di leggibile */
```

```
static char *
format_mode(unsigned long fmode)
{
    ...
}
```

Viene poi una funzione per leggere dei collegamenti simbolici, anche questa omessa per brevità:

```
/* read_symlink --- legge un collegamento simbolico in un buffer
   allocato.
   ... */
```

```
static char *
read_symlink(const char *fname, size_t bufsize, ssize_t *linksize)
{
    ...
}
```

Due funzioni ausiliarie semplificano l'immissione di valori nel vettore che conterrà il risultato della chiamata a `stat()`:

```
/* array_set --- imposta un elemento di un vettore */
```

```
static void
array_set(awk_array_t vettore, const char *sub, awk_value_t *valore)
{
    awk_value_t index;

    set_array_element(vettore,
                      make_const_string(sub, strlen(sub), & index),
                      valore);
}
```

```
/* array_set_numeric --- imposta un elemento di un vettore con un
   numero */
```

```
static void
array_set_numeric(awk_array_t vettore, const char *sub, double num)
{
    awk_value_t tmp;

    array_set(vettore, sub, make_number(num, & tmp));
}
```

La seguente funzione fa il grosso del lavoro per riempire il vettore dei risultati `awk_array_t` con valori ottenuti da una `struct stat` valida. Questo lavoro è fatto in una funzione separata per supportare sia la funzione `stat()` per `gawk`, che l'estensione `fts()`, che è inclusa nello stesso file, ma non è mostrata qui (si veda la [Sezione 16.7.1 \[Funzioni relative ai file\]](#), pagina 445).

La prima parte della funzione è la dichiarazione delle variabili, compresa una tabella per tradurre i tipi di file in stringhe:

```
/* fill_stat_array --- fa il lavoro di riempire un
   vettore con informazioni da stat */

static int
fill_stat_array(const char *name, awk_array_t vettore, struct stat *sbuf)
{
    char *pmode;    /* mode stampabile */
    const char *type = "unknown";
    awk_value_t tmp;
    static struct ftype_map {
        unsigned int mask;
        const char *type;
    } ftype_map[] = {
        { S_IFREG, "file" },
        { S_IFBLK, "blockdev" },
        { S_IFCHR, "chardev" },
        { S_IFDIR, "directory" },
```

```

#ifdef S_IFSOCK
    { S_IFSOCK, "socket" },
#endif
#ifdef S_IFIFO
    { S_IFIFO, "fifo" },
#endif
#ifdef S_IFLNK
    { S_IFLNK, "symlink" },
#endif
#ifdef S_IFDOOR /* Stranezza Solaris */
    { S_IFDOOR, "door" },
#endif /* S_IFDOOR */
};
int j, k;

```

Il vettore di destinazione è svuotato di elementi, e poi il codice riempie i vari elementi prendendoli dai valori presenti in `struct stat`:

```

/* svuota il vettore */
clear_array(vettore);

/* riempie il vettore */
array_set(vettore, "name", make_const_string(name, strlen(name),
                                              & tmp));
array_set_numeric(vettore, "dev", sbuf->st_dev);
array_set_numeric(vettore, "ino", sbuf->st_ino);
array_set_numeric(vettore, "mode", sbuf->st_mode);
array_set_numeric(vettore, "nlink", sbuf->st_nlink);
array_set_numeric(vettore, "uid", sbuf->st_uid);
array_set_numeric(vettore, "gid", sbuf->st_gid);
array_set_numeric(vettore, "size", sbuf->st_size);
array_set_numeric(vettore, "blocks", sbuf->st_blocks);
array_set_numeric(vettore, "atime", sbuf->st_atime);
array_set_numeric(vettore, "mtime", sbuf->st_mtime);
array_set_numeric(vettore, "ctime", sbuf->st_ctime);

/* per dispositivi a blocchi o carattere, aggiunge rdev,
   e il numero principale e secondario */
if (S_ISBLK(sbuf->st_mode) || S_ISCHR(sbuf->st_mode)) {
    array_set_numeric(vettore, "rdev", sbuf->st_rdev);
    array_set_numeric(vettore, "major", major(sbuf->st_rdev));
    array_set_numeric(vettore, "minor", minor(sbuf->st_rdev));
}

```

L'ultima parte della funzione fa alcune aggiunte selettive al vettore di destinazione, a seconda che siano disponibili o no certi campi e/o il tipo del file. Viene poi restituito zero, per indicare che tutto è andato bene:

```

#ifdef HAVE_STRUCT_STAT_ST_BLKSIZE
    array_set_numeric(vettore, "blksize", sbuf->st_blksize);

```

```

#endif /* HAVE_STRUCT_STAT_ST_BLKSIZE */

pmode = format_mode(sbuf->st_mode);
array_set(vettore, "pmode", make_const_string(pmode, strlen(pmode),
                                              & tmp));

/* per collegamenti simbolici, si aggiunge un campo linkval */
if (S_ISLNK(sbuf->st_mode)) {
    char *buf;
    ssize_t linksize;

    if ((buf = read_symlink(name, sbuf->st_size,
                          & linksize)) != NULL)
        array_set(vettore, "linkval",
                  make_malloced_string(buf, linksize, & tmp));
    else
        warning(ext_id, _("stat: non riesco a leggere il \
collegamento simbolico '%s'"),
              name);
}

/* aggiunge il tipo di campo */
type = "unknown"; /* non dovrebbe succedere */
for (j = 0, k = sizeof(ftype_map)/sizeof(ftype_map[0]); j < k; j++) {
    if ((sbuf->st_mode & S_IFMT) == ftype_map[j].mask) {
        type = ftype_map[j].type;
        break;
    }
}

array_set(vettore, "type", make_const_string(type, strlen(type), & tmp));

return 0;
}

```

Del terzo argomento passato a `stat()` non si era ancora parlato. Questo argomento è facoltativo. Se presente, dice a `do_stat()` di usare la chiamata di sistema `stat()` invece della chiamata di sistema `lstat()`. Questo avviene attraverso un puntatore a funzione: `statfunc`. `statfunc` è inizializzato per puntare a `lstat()` (invece che a `stat()`) per ottenere le informazioni relative al file, nel caso che il file in questione sia un collegamento simbolico. Tuttavia, se il terzo argomento è specificato, `statfunc` viene modificato in modo da puntare a `stat()`.

Ecco la funzione `do_stat()`, che inizia con la dichiarazione delle variabili e un controllo degli argomenti passati dal chiamante:

```

/* do_stat --- fornisce una funzione stat() per gawk */

static awk_value_t *

```

```

do_stat(int nargs, awk_value_t *risultato, struct awk_ext_func *non_usata)
{
    awk_value_t file_param, array_param;
    char *name;
    awk_array_t vettore;
    int ret;
    struct stat sbuf;
    /* per default si usa lstat() */
    int (*statfunc)(const char *path, struct stat *sbuf) = lstat;

    assert(risultato != NULL);

```

A questo punto inizia l'elaborazione vera e propria. Per prima cosa, la funzione esamina gli argomenti. Poi, ottiene le informazioni relative al file. Se la funzione chiamata (`lstat()` o `stat()`) restituisce un errore, il codice imposta `ERRNO` e torna al chiamante:

```

    /* file è il primo argomento,
       il vettore per contenere i risultati è il secondo */
    if ( ! get_argument(0, AWK_STRING, &file_param)
        || ! get_argument(1, AWK_ARRAY, &array_param)) {
        warning(ext_id, _("stat: parametri errati"));
        return make_number(-1, risultato);
    }

    if (nargs == 3) {
        statfunc = stat;
    }

    name = file_param.str_value.str;
    vettore = array_param.array_cookie;

    /* svuota sempre il vettore all'inizio */
    clear_array(vettore);

    /* chiama stat per il file;
       in caso di errore,
       imposta ERRNO e ritorna */
    ret = statfunc(name, &sbuf);
    if (ret < 0) {
        update_ERRNO_int(errno);
        return make_number(ret, risultato);
    }

```

Il lavoro noioso è svolto da `fill_stat_array()`, visto in precedenza. Alla fine, la funzione restituisce il codice di ritorno impostato da `fill_stat_array()`:

```

    ret = fill_stat_array(name, vettore, &sbuf);

    return make_number(ret, risultato);
}

```

Infine, è necessario fornire la “colla” che aggrega le nuove funzioni a `gawk`.

L'estensione `filefuncs` comprende anche una funzione `fts()`, qui omessa (si veda la [Sezione 16.7.1 \[Funzioni relative ai file\], pagina 445](#)). È anche prevista una funzione di inizializzazione:

```
/* init_filefuncs --- routine di inizializzazione */

static awk_bool_t
init_filefuncs(void)
{
    ...
}
```

Siamo quasi alla fine. Serve un vettore di strutture `awk_ext_func_t` per caricare ogni funzione in `gawk`:

```
static awk_ext_func_t func_table[] = {
    { "chdir", do_chdir, 1, 1, awk_false, NULL },
    { "stat",  do_stat, 3, 2, awk_false, NULL },
    ...
};
```

Ogni estensione deve avere una routine di nome `dl_load()` per caricare tutto ciò che occorre caricare. La cosa più semplice è di usare la macro `dl_load_func()` in `gawkapi.h`:

```
/* definizione della funzione dl_load()
   usando la macro standard */

dl_load_func(func_table, filefuncs, "")
```

Abbiamo finito!

### 16.6.3 Integrare le estensioni

Dopo aver scritto il codice, dev'essere possibile aggiungerlo in fase di esecuzione all'interprete `gawk`. Per prima cosa, il codice va compilato. Supponendo che le funzioni siano in un file di nome `filefuncs.c`, e che `idir` sia la posizione del file di intestazione `gawkapi.h`, i seguenti passi<sup>9</sup> creano una libreria condivisa GNU/Linux:

```
$ gcc -fPIC -shared -DHAVE_CONFIG_H -c -O -g -Iidir filefuncs.c
$ gcc -o filefuncs.so -shared filefuncs.o
```

Una volta creata la libreria, questa viene caricata usando la parola chiave `@load`:

```
# file testff.awk
@load "filefuncs"

BEGIN {
    "pwd" | getline curdir # salva la directory corrente
    close("pwd")
```

<sup>9</sup> In pratica, si potrebbe decidere di usare i comandi GNU Autotools (Automake, Autoconf, Libtool, e `gettext`) per configurare e costruire le librerie necessarie. L'esposizione di come ciò può essere fatto esula dal tema di questo libro. Si veda la [Sezione 16.8 \[Il progetto gawkextlib\], pagina 455](#), per i puntatori a siti Internet che permettono di accedere a questi strumenti.

```

chdir("/tmp")
system("pwd")    # verifica l'avvenuto cambio di directory
chdir(curdir)    # torna indietro

print "Info per testff.awk"
ret = stat("testff.awk", data)
print "ret =", ret
for (i in data)
    printf "data[\"%s\"] = %s\n", i, data[i]
print "testff.awk modified:",
    strftime("%m %d %Y %H:%M:%S", data["mtime"])

print "\nInfo per JUNK"
ret = stat("JUNK", data)
print "ret =", ret
for (i in data)
    printf "data[\"%s\"] = %s\n", i, data[i]
print "JUNK modified:", strftime("%m %d %Y %H:%M:%S", data["mtime"])
}

```

La variabile d'ambiente `AWKLIBPATH` dice a `gawk` dove è possibile trovare le estensioni (si veda la [Sezione 16.5 \[Come gawk trova le estensioni compilate\]](#), pagina 434). La variabile viene impostata alla directory corrente, e quindi viene eseguito il programma:

```

$ AWKLIBPATH=$PWD gawk -f testff.awk
+ /tmp
+ Info per testff.awk
+ ret = 0
+ data["blksize"] = 4096
+ data["devbsize"] = 512
+ data["mtime"] = 1412004710
+ data["mode"] = 33204
+ data["type"] = file
+ data["dev"] = 2053
+ data["gid"] = 1000
+ data["ino"] = 10358899
+ data["ctime"] = 1412004710
+ data["blocks"] = 8
+ data["nlink"] = 1
+ data["name"] = testff.awk
+ data["atime"] = 1412004716
+ data["pmode"] = -rw-rw-r--
+ data["size"] = 666
+ data["uid"] = 1000
+ testff.awk modified: 09 29 2014 18:31:50
+
+ Info per JUNK

```

```

+ ret = -1
+ JUNK modified: 01 01 1970 02:00:00

```

## 16.7 Le estensioni di esempio incluse nella distribuzione gawk

Questa sezione fornisce una breve panoramica degli esempi di estensione inclusi nella distribuzione di **gawk**. Alcune di esse sono destinate per l'uso in produzione (p.es., le estensioni **filefuncs**, **readdir**, e **inplace**). Altre sono state scritte principalmente per mostrare come si usa l'estensione API.

### 16.7.1 Funzioni relative ai file

L'estensione **filefuncs** include tre funzioni diverse, come descritto sotto. L'uso è il seguente:

```
@load "filefuncs"
```

Questo è il modo per caricare l'estensione.

```
risultato = chdir("/qualche/directory")
```

La funzione **chdir()** invoca a sua volta la chiamata di sistema **chdir()** per cambiare la directory corrente. Restituisce zero se tutto va bene o un valore minore di zero in caso di errore. In quest'ultimo caso, viene aggiornata la variabile **ERRNO**.

```
risultato = stat("/qualche/percorso", statdata [, follow])
```

La funzione **stat()** invoca a sua volta la chiamata di sistema **stat()**. Restituisce zero se tutto va bene o un valore minore di zero in caso di errore. In quest'ultimo caso, viene aggiornata la variabile **ERRNO**.

Per default, viene usata la chiamata di sistema **lstat()**. Tuttavia, se alla funzione viene passato un terzo argomento, questa invoca invece **stat()**.

In tutti i casi, il vettore **statdata** viene preventivamente svuotato. Quando la chiamata a **stat()** riesce, viene riempito il vettore **statdata** con le informazioni ottenute dal filesystem, come segue:

Indice	Campo in struct stat	Tipo file
"name"	Il nome-file	Tutti
"dev"	st_dev	Tutti
"ino"	st_ino	Tutti
"mode"	st_mode	Tutti
"nlink"	st_nlink	Tutti
"uid"	st_uid	Tutti
"gid"	st_gid	Tutti
"size"	st_size	Tutti
"atime"	st_atime	Tutti
"mtime"	st_mtime	Tutti
"ctime"	st_ctime	Tutti
"rdev"	st_rdev	Dispositivi
"major"	st_major	Dispositivi
"minor"	st_minor	Dispositivi
"blksize"	st_blksize	Tutti

"pmode"	Una versione leggibile del valore dell'autorizzazione, come quello stampato dal comando <code>ls</code> (per esempio, <code>"-rwxr-xr-x"</code> )	Tutti
"linkval"	Il valore del collegamento simbolico	Collegamenti simbolici
"type"	Il tipo del file in formato stringa—può essere <code>"file"</code> , <code>"blockdev"</code> , <code>"chardev"</code> , <code>"directory"</code> , <code>"socket"</code> , <code>"fifo"</code> , <code>"symlink"</code> , <code>"door"</code> o <code>"unknown"</code> (non tutti i sistemi supportano tutti i tipi file)	Tutti

```
flags = or(FTS_PHYSICAL, ...)
```

```
risultato = fts(pathlist, flags, filedata)
```

Percorre gli alberi di file elencati in `pathlist` e riempie il vettore `filedata`, come descritto qui di seguito. `flags` è l'operazione *OR bit a bit* di parecchi valori predefiniti, pure descritti più sotto. Restituisce zero in assenza di errori, in caso contrario restituisce `-1`.

La funzione `fts()` invoca a sua volta la routine di libreria C `fts()` per percorrere gerarchie di file. Invece di restituire i dati relativi ai file uno per volta in sequenza, riempie un vettore multidimensionale con i dati di ogni file e directory che risiedono nelle gerarchie richieste.

Gli argomenti sono i seguenti:

**pathlist** Un vettore di nomi-file. Sono usati i valori dei singoli elementi; gli indici che puntano a tali valori vengono ignorati.

**flags** Questo dovrebbe essere l'*OR bit a bit* di uno o più dei seguenti valori dei flag costanti predefiniti. Almeno uno dei due flag `FTS_LOGICAL` o `FTS_PHYSICAL` dev'essere impostato; in caso contrario `fts()` restituisce una segnalazione di errore e imposta `ERRNO`. I flag sono:

#### `FTS_LOGICAL`

Passa in rassegna i file in modo "logico", e quindi l'informazione restituita per un collegamento simbolico è quella relativa al file puntato, e non al collegamento simbolico stesso. Questo flag è mutuamente esclusivo con `FTS_PHYSICAL`.

#### `FTS_PHYSICAL`

Passa in rassegna i file in modo "fisico", e quindi l'informazione restituita per un collegamento simbolico è quella relativa al collegamento simbolico stesso. Questo flag è mutuamente esclusivo con `FTS_LOGICAL`.

#### `FTS_NOCHDIR`

Per migliorare le prestazioni, la routine di libreria C `fts()` cambia directory mentre percorre una gerarchia di file. Questo flag disabilita quell'ottimizzazione.

**FTS\_COMFOLLOW**

Si accede al file puntato da un collegamento simbolico esistente in `pathlist`, anche se `FTS_LOGICAL` non è stato impostato.

**FTS\_SEEDOT**

Per default, la routine di libreria C `fts()` non restituisce informazioni per i file `"."` (punto) e `".."` (punto-punto). Quest'opzione richiede l'informazione anche per `".."`. (L'estensione ritorna sempre l'informazione per `"."`; maggiori dettagli più sotto.)

**FTS\_XDEV** Mentre si percorre un filesystem, non passare mai a un filesystem montato diverso da quello in cui si opera.

**filedata** Il vettore `filedata` contiene i risultati. La funzione `fts()` lo svuota all'inizio. In seguito viene creato un elemento in `filedata` per ogni elemento in `pathlist`. L'indice è il nome della directory o del file specificato in `pathlist`. L'elemento puntato da questo indice è a sua volta un vettore. Ci sono due casi:

*Il percorso è un file*

In questo caso, il vettore contiene due o tre elementi:

- "path"** Il percorso completo di questo file, a partire dalla directory radice ("root") indicata nel vettore `pathlist`.
- "stat"** Questo elemento è esso stesso un vettore, contenente le stesse informazioni fornite dalla funzione `stat()` vista in precedenza a proposito del suo argomento `statdata`. L'elemento può non essere presente se la chiamata di sistema `stat()` per il file non è riuscita.
- "error"** Se qualche tipo di errore si verifica durante l'elaborazione, il vettore conterrà anche un elemento con chiave **"error"**, che è una stringa che descrive l'errore.

*Il percorso è una directory*

In questo caso, nel vettore viene creato un elemento per ogni elemento contenuto nella directory. Se un elemento della directory è un file, l'azione del programma è la stessa descritta sopra per un file. Se invece la directory contiene delle sottodirectory, l'elemento creato nel vettore è (ricorsivamente) a sua volta un vettore che descrive la sottodirectory. Se fra i flag è stato specificato il flag `FTS_SEEDOT`, ci sarà anche un elemento di nome `".."`. Questo elemento sarà un vettore contenente i dati restituiti da un'invocazione di `stat()`.

Inoltre, ci sarà un elemento il cui indice è `"."`. Questo elemento è un vettore contenente gli stessi due o tre elementi che sono messi a disposizione per un file: **"path"**, **"stat"**, ed **"error"**.

La funzione `fts()` restituisce zero in assenza di errori. in caso contrario, restituisce `-1`.

**NOTA:** L'estensione `fts()` non imita esattamente l'interfaccia fornita dalla routine di libreria C `fts()`, ma sceglie di fornire un'interfaccia basata sui vettori

associativi, che è più adeguata per l'uso da parte di un programma **awk**. Questo implica la mancanza di una funzione di confronto, poiché **gawk** già prevede la possibilità di mettere facilmente nell'ordine desiderato gli elementi di un vettore. Anche se un'interfaccia modellata su **fts\_read()** avrebbe potuto essere fornita, è sembrato più naturale mettere a disposizione un vettore multidimensionale, che rappresenta la gerarchia dei file e le informazioni relative a ognuno di essi.

Si veda il file **test/fts.awk** nella distribuzione di **gawk** per un esempio di uso dell'estensione **fts()**.

### 16.7.2 Un'interfaccia a **fnmatch()**

Quest'estensione fornisce un'interfaccia per utilizzare la funzione di libreria C **fnmatch()**. Si usa così:

```
@load "fnmatch"
    È questo il modo per caricare l'estensione.
```

```
risultato = fnmatch(pattern, stringa, flags)
    Il valore restituito è zero se tutto va bene, oppure FNM_NOMATCH se la funzione
    non ha trovato alcuna corrispondenza, o un valore differente, diverso da zero,
    se si è verificato un errore.
```

Oltre a rendere disponibile la funzione **fnmatch()**, l'estensione di **fnmatch** definisce una costante (**FNM\_NOMATCH**), e un vettore con dei valori di flag, di nome **FNM**.

Gli argomenti per **fnmatch()** sono:

**pattern**    L'espressione regolare con cui confrontare nome-file

**stringa**    La stringa nome-file

**flags**       Può valere zero o essere l'*OR bit a bit* di uno o più flag nel vettore **FNM**

I flag sono i seguenti:

Elemento del vettore	Flag corrispondente definito da <b>fnmatch()</b>
<b>FNM["CASEFOLD"]</b>	<b>FNM_CASEFOLD</b>
<b>FNM["FILE_NAME"]</b>	<b>FNM_FILE_NAME</b>
<b>FNM["LEADING_DIR"]</b>	<b>FNM_LEADING_DIR</b>
<b>FNM["NOESCAPE"]</b>	<b>FNM_NOESCAPE</b>
<b>FNM["PATHNAME"]</b>	<b>FNM_PATHNAME</b>
<b>FNM["PERIOD"]</b>	<b>FNM_PERIOD</b>

Ecco un esempio:

```
@load "fnmatch"
...
flags = or(FNM["PERIOD"], FNM["NOESCAPE"])
if (fnmatch("*.a", "pippo.c", flags) == FNM_NOMATCH)
    print "nessuna corrispondenza"
```

### 16.7.3 Un'interfaccia a `fork()`, `wait()`, e `waitpid()`

L'estensione `fork` mette a disposizione tre funzioni, come segue:

```
@load "fork"
```

Questo è il modo per caricare l'estensione.

```
pid = fork()
```

Questa funzione crea un nuovo processo. Il valore restituito è zero nel processo “figlio” e il numero che identifica il nuovo processo (*pid*) nel processo “padre”, o `-1` in caso di errore. In quest'ultimo caso, `ERRNO` indica il problema. Nel processo figlio, gli elementi `PROCINFO["pid"]` e `PROCINFO["ppid"]` vengono aggiornati per riflettere i valori corretti.

```
ret = waitpid(pid)
```

Questa funzione ha un unico argomento numerico, l'identificativo del processo di cui aspettare l'esito. Il valore di ritorno è quello restituito dalla chiamata di sistema `waitpid()`.

```
ret = wait()
```

Questa funzione attende che il primo processo “figlio” termini. Il valore restituito è quello della chiamata di sistema `wait()`.

Non c'è una funzione corrispondente alla chiamata di sistema `exec()`.

Ecco un esempio:

```
@load "fork"
...
if ((pid = fork()) == 0)
    print "salve dal processo figlio"
else
    print "salve dal processo padre"
```

### 16.7.4 Consentire la modifica in loco dei file

L'estensione `inplace` svolge un lavoro simile a quello dell'opzione `-i` nel programma di utilità GNU `sed`, che svolge delle funzioni di modifica “al volo” su ogni file in input. Viene usato il file `inplace.awk`, caricato dinamicamente, per richiamare l'estensione in maniera corretta:

```

# inplace --- carica e richiama l'estensione inplace.

@load "inplace"

# È buona cosa impostare INPLACE_SUFFIX in modo da fare
# una copia di backup.
# Per esempio, si potrebbe impostare INPLACE_SUFFIX a .bak
# sulla riga di comando, o in una regola BEGIN.

# Per default, ogni file specificato sulla riga di comando
# verrà modificato sovrascrivendo il file originale.
# Ma è possibile evitarlo specificando l'argomento inplace=0
# davanti al nome del file che non si desidera elaborare in questo modo.
# Si può poi abilitare di nuovo l'aggiornamento diretto del file
# sulla riga di comando, specificando inplace=1 prima del file
# che si vuole modificare direttamente.

# N.B. La funzione inplace_end() è invocata nelle regole
# BEGINFILE ed END, in modo che ogni eventuale azione
# in una regola ENDFILE sarà ridiretta come previsto.

BEGIN {
    inplace = 1 # abilitato per default
}

BEGINFILE {
    if (_inplace_filename != "")
        inplace_end(_inplace_filename, INPLACE_SUFFIX)
    if (inplace)
        inplace_begin(_inplace_filename = FILENAME, INPLACE_SUFFIX)
    else
        _inplace_filename = ""
}

END {
    if (_inplace_filename != "")
        inplace_end(_inplace_filename, INPLACE_SUFFIX)
}

```

Per ogni file elaborato, l'estensione ridirige lo standard output verso un file temporaneo definito in modo da avere lo stesso proprietario e le stesse autorizzazioni del file originale. Dopo che il file è stato elaborato, l'estensione riporta lo standard output alla sua destinazione originale. Se `INPLACE_SUFFIX` non è una stringa vuota, il file originale è collegato a un nome-file di backup, creato aggiungendo il suffisso al nome originale. Infine, il file temporaneo è rinominato in modo da essere lo stesso del nome-file originario.

Si noti che l'uso di questa funzionalità può essere controllato specificando `'inplace=0'` sulla riga di comando, prima del nome del file che non dovrebbe essere elaborato come

appena descritto. Si può richiedere ancora l'aggiornamento diretto di un file, specificando l'argomento `'inplace=1'` davanti al nome del file da elaborare in maniera diretta.

La variabile `_inplace_filename` serve per tener traccia del nome del file corrente, in modo da non eseguire la funzione `inplace_end()` prima di aver elaborato il primo file.

Se si verifica un errore, l'estensione emette un messaggio di errore fatale per terminare l'elaborazione immediatamente, senza danneggiare il file originale.

Ecco alcuni semplici esempi:

```
$ gawk -i inplace '{ gsub(/pippo/, "pluto") }; { print }' file1 file2 file3
```

Per mantenere una copia di backup del file originale, si provi a fare così:

```
$ gawk -i inplace -v INPLACE_SUFFIX=.bak '{ gsub(/pippo/, "pluto") }
> { print }' file1 file2 file3
```

Si noti che, anche se l'estensione tenta di mantenere il proprietario e i permessi di accesso del file originario, non viene tentata la copia degli ulteriori permessi di accesso (*ACL - Access Control Lists*) del file originale.

Se il programma termina prima del previsto, come potrebbe succedere se riceve dal sistema un segnale non gestito, può lasciare come residuo un file temporaneo.

### 16.7.5 Caratteri e valori numerici: `ord()` e `chr()`

L'estensione `ordchr` aggiunge due funzioni, di nome `ord()` e `chr()`, come segue:

```
@load "ordchr"
```

Questo è il modo per caricare l'estensione.

```
numero = ord(stringa)
```

Restituisce il valore numerico del primo carattere in `stringa`.

```
char = chr(number)
```

Restituisce una stringa il cui primo carattere è quello rappresentato da `number`.

Queste funzioni sono ispirate alle funzioni del linguaggio Pascal dallo stesso nome. Ecco un esempio:

```
@load "ordchr"
...
printf("Il valore numerico di 'A' è %d\n", ord("A"))
printf("Il valore come stringa di 65 è %s\n", chr(65))
```

### 16.7.6 Leggere directory

L'estensione `readdir` aggiunge un analizzatore di input per esaminare directory. L'uso è il seguente:

```
@load "readdir"
```

Quando quest'estensione è in uso, invece che saltare le directory presenti sulla riga di comando, (o accedute tramite `getline`), queste sono lette, e ogni elemento della directory è restituito come un record.

Il record consiste di tre campi. I primi due sono il numero di *inode* e il nome-file, separati fra loro da una barra. Nei sistemi in cui l'elemento di directory contiene il tipo del file, il record ha un terzo campo (pure separato da una barra), composto da una sola lettera,

che indica il tipo del file. Le lettere e i tipi di file a cui corrispondono sono mostrate in [Tabella 16.3](#).

Lettera	Tipo di file
b	Dispositivo a blocchi
c	Dispositivo a caratteri
d	Directory
f	File normale
l	Collegamento simbolico
p	<i>pipe</i> con nome (FIFO)
s	<i>socket</i>
u	Tutto il resto (sconosciuto)

Tabella 16.3: Tipi file restituiti dall'estensione `readdir`

Nei sistemi che non contengono l'informazione sul tipo del file, il terzo campo è sempre 'u'.

**NOTA:** Nei sistemi GNU/Linux, ci sono filesystem che non supportano il campo `d_type` (si veda la pagina di manuale *readdir(3)*), e in questo caso il tipo di file è sempre 'u'. Si può usare l'estensione `filefuncs` per chiamare `stat()` e ottenere l'informazione corretta sul tipo di file.

Ecco un esempio:

```
@load "readdir"
...
BEGIN { FS = "/" }
{ print "nome-file è", $2 }
```

### 16.7.7 Invertire la stringa in output

L'estensione `revoutput` aggiunge un semplice processore di output che inverte i caratteri di ogni riga in output. Serve a dimostrare come è possibile scrivere un processore di output, anche se può essere a prima vista vagamente divertente. Ecco un esempio:

```
@load "revoutput"

BEGIN {
    REVOUT = 1
    print "non v'allarmate" > "/dev/stdout"
}
```

L'output di questo programma è 'etamralla'v non'.

### 16.7.8 Esempio di I/O bidirezionale

L'estensione `revtwoaway` aggiunge un semplice processore bidirezionale che inverte i caratteri di ogni riga che riceve, per farla poi rileggere dal programma `awk`. Il motivo per cui è stata scritta è quello di mostrare come si scrive un processore bidirezionale, anche se può sembrare un programma vagamente divertente. Il seguente esempio mostra come usarlo:

```
@load "revtwoaway"
```

```

BEGIN {
    cmd = "/specchio/magico"
    print "non v'allarmate" |& cmd
    cmd |& getline risultato
    print risultato
    close(cmd)
}

```

L'output di questo programma anche in questo caso è: 'etamralla'v non'.

### 16.7.9 Scaricare e ricaricare un vettore

L'estensione `rwarray` aggiunge due funzioni, di nome `writea()` e `reada()`, come segue:

```
@load "rwarray"
```

Questo è il modo per caricare l'estensione.

```
ret = writea(file, vettore)
```

Questa funzione ha come argomento una stringa, che è il nome del file sul quale scaricare il vettore, e il vettore stesso è il secondo argomento. `writea()` è in grado di gestire vettori di vettori. Restituisce il valore uno se completa il lavoro o zero se non va a buon fine.

```
ret = reada(file, vettore)
```

`reada()` è la funzione inversa di `writea()`; legge il file il cui nome è fornito come primo argomento, riempiendo il vettore il cui nome è il secondo argomento. Il vettore viene preventivamente svuotato. Anche in questo caso, il valore restituito è uno se tutto va bene o zero se la funzione non va a buon fine.

Il vettore creato da `reada()` è identico a quello scritto da `writea()` nel senso che i contenuti sono gli stessi. Tuttavia, per come è strutturata la funzione, l'ordine di attraversamento del vettore ricreato è quasi certamente differente da quello del vettore originale. Poiché l'ordine di attraversamento di un vettore è, per default, indefinito in `awk`, questo non è (tecnicamente) un problema. Se serve che l'attraversamento del vettore avvenga in un ordine preciso, si possono usare le funzionalità di ordinamento di un vettore disponibili in `gawk` (si veda la [Sezione 12.2 \[Controllare la visita di un vettore e il suo ordinamento\]](#), [pagina 332](#)).

Il file contiene dati in formato binario. Tutti i valori interi sono scritti in *network byte order*<sup>10</sup>. Tuttavia, i valori in virgola mobile a doppia precisione sono scritti come dati binari nativi. Quindi, vettori che contengono solo dati in formato stringa possono essere scaricati da un sistema con un certo ordine di byte e ripristinati su un sistema con un ordine di byte differente, anche se un test al riguardo non è mai stato fatto.

Ecco un esempio:

```

@load "rwarray"
...
ret = writea("scaricato.bin", vettore)

```

<sup>10</sup> Cioè, nella maniera con cui sarebbero normalmente scritti in un testo, con le cifre più significative del numero contenute nella parte sinistra, e quelle meno significative nella parte destra della rappresentazione binaria del numero.

```
...
ret = reada("scaricato.bin", vettore)
```

### 16.7.10 Leggere un intero file in una stringa

L'estensione `readfile` aggiunge una sola funzione di nome `readfile()`, e un analizzatore di input:

```
@load "readfile"
```

Questo è il modo per caricare l'estensione.

```
risultato = readfile("/qualche/persorso")
```

L'argomento è il nome del file da leggere. Il valore restituito è una stringa contenente l'intero contenuto del file richiesto. In caso di errore, la funzione restituisce la stringa vuota e imposta `ERRNO`.

```
BEGIN { PROCINFO["readfile"] = 1 }
```

Inoltre, l'estensione aggiunge un analizzatore di input che è attivato se l'elemento `PROCINFO["readfile"]` esiste. Quando l'analizzatore è attivato, ogni file in input è restituito interamente come `$0`. La variabile `RT` è impostata alla stringa nulla.

Ecco un esempio:

```
@load "readfile"
...
contents = readfile("/percorso/del/file");
if (contents == "" && ERRNO != "") {
    print("problema in lettura file", ERRNO) > "/dev/stderr"
    ...
}
```

### 16.7.11 Funzioni dell'estensione time

L'estensione `time` aggiunge due funzioni, di nome `gettimeofday()` e `sleep()`, come segue:

```
@load "time"
```

Questo è il modo per caricare l'estensione.

```
ora_corrente = gettimeofday()
```

Restituisce il numero di secondi trascorsi dalle ore 00:00 del giorno 01/01/1970 UTC come valore a virgola mobile. Se questa informazione non è disponibile nella piattaforma in uso, restituisce `-1` e imposta `ERRNO`. Il valore fornito dovrebbe avere la precisione di una frazione di secondo, ma la precisione effettiva può variare a seconda della piattaforma. Se la chiamata di sistema standard C `gettimeofday()` è disponibile nella piattaforma in uso, questo è il valore restituito. In caso contrario, se si sta lavorando con MS-Windows, la chiamata di sistema è fatta a `GetSystemTimeAsFileTime()`.

```
risultato = sleep(secondi)
```

Il programma `gawk` resta inattivo (dorme) per i *secondi* specificati. Se *secondi* ha un valore negativo, o la chiamata di sistema non riesce, restituisce `-1` e imposta `ERRNO`. In caso contrario, restituisce zero dopo aver lasciato trascorrere

la quantità di tempo indicata. Si noti che *secondi* può essere un numero a virgola mobile (non solo un numero intero). Dettagli di implementazione: a seconda della disponibilità nel sistema in uso, questa funzione tenta di usare `nanosleep()` o `select()` per ottenere il tempo di attesa richiesto.

### 16.7.12 Test per la API

L'estensione `testtext` controlla la funzionalità di parti dell'API delle estensioni che non sono utilizzate negli altri esempi. Il file `extension/testtext.c` contiene sia il codice C per l'estensione che il codice `awk` (tra i commenti del codice C) per eseguire i test. L'ambiente di test estrae il codice sorgente `awk` ed esegue i test. Si veda il file sorgente per maggiori informazioni.

## 16.8 Il progetto gawkextlib

Il progetto `gawkextlib` fornisce varie estensioni per `gawk`, compresa una per l'elaborazione dei file XML. Questa è un'evoluzione del progetto noto come `xgawk` (XML `gawk`).

Al momento della stesura di questo testo, ci sono otto estensioni:

- Estensione `errno`
- Estensione GD graphics library
- Estensione libreria MPFR (fornisce l'accesso a varie funzioni MPFR non previste dal supporto nativo di MPFR disponibile in `gawk`)
- Estensione PDF
- Estensione PostgreSQL
- Estensione Redis
- Estensione Select
- Estensione analizzatore XML, usando la libreria di analisi XML `Expat`

Si può scaricare il codice del progetto `gawkextlib` usando il codice sorgente mantenuto tramite `Git`. Il comando per farlo è il seguente:

```
git clone git://git.code.sf.net/p/gawkextlib/code gawkextlib-code
```

Per poter compilare e usare l'estensione XML, è necessario installare la libreria di analisi XML `Expat`.

Inoltre, è necessario installare gli strumenti GNU Autotools (`Autoconf`, `Automake`, `Libtool` e `GNU gettext`).

La semplice procedura per compilare e testare `gawkextlib` è la seguente. Dapprima, occorre compilare e installare `gawk`:

```
cd .../percorso/del/sorgente/gawk
./configure --prefix=/tmp/newgawk    Installa in /tmp/newgawk per ora
make && make check                  Compila e controlla che tutto sia a posto
make install                        Installa gawk
```

Poi, dal sito <http://sourceforge.net/projects/gawkextlib/files> si deve scaricare `gawkextlib` e le estensioni che si vogliono installare. Il file `README` del sito spiega come compilare il codice. Se si è installato `gawk` in una posizione non-standard, occorre specificare `./configure --with-gawk=/percorso/del/programma/gawk` per far sì che venga

trovato. Può essere necessario usare il programma di utilità `sudo` per installare sia `gawk` che `gawkextlib`, a seconda di come funziona il sistema su cui si lavora.

Chi scrive un'estensione e desidera condividerla con altri utenti `gawk`, può prendere in considerazione l'idea di farlo attraverso il progetto `gawkextlib`. Si veda il sito web del progetto per maggiori informazioni.

## 16.9 Sommario

- Si possono scrivere estensioni (dette anche *plug-in*) per `gawk` nel linguaggio C o C++ usando l'interfaccia di programmazione applicativa (API) definita dagli sviluppatori di `gawk`.
- Le estensioni devono avere una licenza compatibile con la GNU General Public License (GPL), e devono dichiararlo definendo un'apposita variabile di nome `plugin_is_GPL_compatible`.
- La comunicazione tra `gawk` e un'estensione è bidirezionale. `gawk` passa all'estensione una struttura (`struct`) che contiene vari campi di dati e puntatori a funzione. L'estensione può poi chiamare funzioni all'interno di `gawk` tramite dei puntatori a funzioni per svolgere alcuni compiti.
- Uno di questi compiti è di “registrare” il nome e l'implementazione di nuove funzioni a livello di `awk` con `gawk`. L'implementazione ha la forma di un puntatore del linguaggio C, cui è associato un dato livello di versione. Per convenzione, le funzioni di implementazione hanno nome `do_XXXX()` per una funzione a livello di `awk` di nome `XXXX()`.
- L'API è definita in un file di intestazione di nome `gawkapi.h`. Occorre includere alcuni file di intestazione standard *prima* di includere tale intestazione nel codice sorgente.
- Vengono forniti dei puntatori a funzioni dell'API per i seguenti tipi di operazioni:
  - Allocare, riallocare, e liberare memoria
  - Registrare funzioni (si possono registrare funzioni di estensione, funzioni ausiliarie di pulizia (*callbacks*), una stringa di versione, degli analizzatori di input, dei processori di output, e dei processori bidirezionali)
  - Stampare messaggi fatali, non fatali, di avvertimento, e avvertimenti “lint”
  - Aggiornare `ERRNO` o annullarlo
  - Accedere a parametri, come pure convertire un parametro di tipo non definito in un vettore
  - Accedere alla tabella dei simboli (ricuperare il valore di una variabile globale, crearne una nuova o modificarne una esistente)
  - Creare e rilasciare valori nascosti; questo consente di usare in modo efficiente lo stesso valore per più variabili e può migliorare di molto le prestazioni del programma
  - Manipolare vettori (ricuperare, aggiungere, cancellare e modificare elementi; ottenere il numero di elementi in un vettore; creare un nuovo vettore; svuotare un vettore; e appiattire un vettore per poterlo percorrere facilmente con un ciclo in stile C, visitando tutti i suoi indici ed elementi)
- L'API definisce diversi tipi di dati standard per rappresentare valori di variabili, elementi di vettore e vettori presenti in `awk`.

- L'API fornisce funzioni di servizio per definire dei valori. Sono anche disponibili funzioni di gestione della memoria, per assicurare la compatibilità fra memoria allocata da **gawk** e memoria allocata da un'estensione.
- *Tutta* la memoria passata da **gawk** a un'estensione dev'essere considerata come in sola lettura dall'estensione.
- *Tutta* la memoria passata da un'estensione a **gawk** deve essere ottenuta dalle funzioni di allocazione della memoria previste dall'API. **gawk** è responsabile per la gestione di quella memoria e la libera quando è il momento per farlo.
- L'API fornisce informazioni sulla versione di **gawk** in esecuzione, in modo che un'estensione possa verificare la propria compatibilità con la versione di **gawk** da cui è stata caricata.
- È più facile iniziare a programmare una nuova estensione usando il codice predefinito descritto in questo capitolo. Alcune macro nel file di intestazione **gawkapi.h** rendono la cosa più agevole.
- La distribuzione di **gawk** comprende un numero di piccoli ma utili esempi di estensione. Il progetto **gawkextlib** include diverse altre estensioni, di maggiori dimensioni. Per chi desideri scrivere un'estensione e metterla a disposizione della comunità degli utenti di **gawk**, il progetto **gawkextlib** è il posto adatto per farlo.

## 16.10 Esercizi

1. Aggiungere funzioni per rendere disponibili chiamate di sistema come **chown()**, **chmod()** e **umask()** nelle estensioni che operano con i file viste nella [Sezione 16.6.2 \[Codice C per eseguire \*\*chdir\(\)\*\* e \*\*stat\(\)\*\*\]](#), pagina 437.
2. Scrivere un analizzatore di input che stampi un prompt se l'input proviene da un dispositivo che sia un "terminale". Si può usare la funzione **isatty()** per sapere se il file in input è un terminale. (Suggerimento: questa funzione normalmente usa parecchie risorse quando è richiamata; si tenti di chiamarla una volta sola.) Il contenuto del prompt dovrebbe provenire da una variabile che sia possibile impostare a livello di codice **awk**. Si può inviare il prompt allo standard error. Tuttavia, per ottenere risultati migliori, è meglio aprire un nuovo descrittore di file (o puntatore a un file) sul file **/dev/tty** e stampare il prompt su quel file, nel caso in cui lo standard error sia stato ridiretto.  
  
Perché lo standard error è una scelta migliore dello standard output per scrivere il prompt? Quale meccanismo di lettura andrebbe sostituito, quello che legge un record o quello che legge dei semplici byte?
3. (Difficile.) Come si potrebbero gestire degli insiemi di nomi (*namespaces*) in **gawk**, in modo che i nomi di funzione presenti in estensioni differenti non siano in conflitto tra loro? Chi riesce a trovare uno schema di buona qualità è pregato di contattare il manutentore di **gawk**, per metterlo al corrente.
4. Si scriva uno script di shell che funga da interfaccia per l'estensione "inplace", vista nella [Sezione 16.7.4 \[Consentire la modifica in loco dei file\]](#), pagina 449, in modo che il comportamento sia simile a quello del comando **'sed -i'**.



## Parte IV:

### Appendici



## Appendice A L'evoluzione del linguaggio awk

Questo libro descrive l'implementazione GNU di `awk` conforme alle specifiche POSIX. Molti degli utenti di lunga data di `awk` hanno imparato a programmare in `awk` usando l'implementazione originale di `awk` presente nella versione 7 di Unix. (Questa versione è servita da base per la versione Berkeley Unix di `awk`, attraverso la versione 4.3BSD-Reno. Successive versioni di Berkeley Unix e, per un certo periodo, alcuni sistemi derivati da 4.4BSD-Lite, hanno usato varie versioni di `gawk` come loro `awk`.) Questo capitolo descrive in breve l'evoluzione del linguaggio `awk`, facendo riferimento ad altre parti del libro dove si possono trovare ulteriori informazioni.

### A.1 Differenze importanti tra V7 e System V Release 3.1

Il linguaggio `awk` si è evoluto considerevolmente tra Unix versione 7 (1978) e la nuova implementazione disponibile a partire da Unix System V Release 3.1 (1987). Questa sezione riassume le differenze e indica dove è possibile trovare ulteriori dettagli:

- La necessità di inserire `;` per separare più regole su una riga (si veda la [Sezione 1.6 \[Istruzioni e righe in awk\]](#), pagina 28)
- Funzioni definite dall'utente e istruzione `return` (si veda la [Sezione 9.2 \[Funzioni definite dall'utente\]](#), pagina 224)
- L'istruzione `delete` (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187)
- L'istruzione `do-while` (si veda la [Sezione 7.4.3 \[L'istruzione do-while\]](#), pagina 154)
- Le funzioni predefinite `atan2()`, `cos()`, `sin()`, `rand()` e `srand()` (si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), pagina 196)
- Le funzioni predefinite `gsub()`, `sub()` e `match()` (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
- Le funzioni predefinite `close()` e `system()` (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210)
- Le variabili predefinite `ARGC`, `ARGV`, `FNR`, `RLENGTH`, `RSTART` e `SUBSEP` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162)
- Possibilità di modificare `$0` (si veda la [Sezione 4.4 \[Cambiare il contenuto di un campo\]](#), pagina 69)
- L'espressione condizionale che fa uso dell'operatore ternario `?:` (si veda la [Sezione 6.3.4 \[Espressioni condizionali\]](#), pagina 137)
- L'espressione `'indice in vettore'` esterna alle istruzioni `for` (si veda la [Sezione 8.1.2 \[Come esaminare un elemento di un vettore\]](#), pagina 179)
- L'operatore esponenziale `^` (si veda la [Sezione 6.2.1 \[Operatori aritmetici\]](#), pagina 123) e il relativo operatore di assegnamento `^=` (si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126)
- Precedenze tra operatori compatibili con quelle del linguaggio C, che rendono non funzionanti alcuni vecchi programmi `awk` (si veda la [Sezione 6.5 \[Precedenza degli operatori \(Come si nidificano gli operatori\)\]](#), pagina 140)
- La possibilità di usare `regexp` come valori di `FS` (si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\]](#), pagina 71) e come terzo argomento per la funzione `split()`

(si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198), invece di usare solo il primo carattere di FS

- *Regex* dinamiche come operandi degli operatori ‘~’ e ‘!~’ (si veda la [Sezione 3.6 \[Usare regex dinamiche\]](#), pagina 57)
- Le sequenze di protezione ‘\b’, ‘\f’ e ‘\r’ (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50)
- La ridirezione dell’input per la funzione `getline` (si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83)
- La possibilità di avere più regole BEGIN ed END (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148)
- Vettori multidimensionali (si veda la [Sezione 8.5 \[Vettori multidimensionali\]](#), pagina 188)

## A.2 Differenze tra le versioni System V Release 3.1 e SVR4

La versione per Unix System V Release 4 (1989) di `awk` ha aggiunto queste funzionalità (alcune delle quali introdotte da `gawk`):

- Il vettore `ENVIRON` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162)
- La possibilità di specificare più opzioni `-f` sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
- L’opzione `-v` per assegnare variabili prima di iniziare l’esecuzione del programma (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
- La notazione `--` per indicare la fine delle opzioni sulla riga di comando
- Le sequenze di protezione ‘\a’, ‘\v’ e ‘\x’ (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50)
- Un valore di ritorno definito per la funzione predefinita `srand()` (si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), pagina 196)
- Le funzioni predefinite per stringhe `toupper()` e `tolower()` per la conversione maiuscolo/minuscolo (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
- Una specificazione più accurata per la lettera ‘%c’ di controllo del formato nella funzione `printf` (si veda la [Sezione 5.5.2 \[Lettere di controllo del formato\]](#), pagina 99)
- La capacità di decidere dinamicamente la larghezza di un campo e la precisione da usare (“%\*.d”) nella lista degli argomenti passati a `printf` e `sprintf()` (si veda la [Sezione 5.5.2 \[Lettere di controllo del formato\]](#), pagina 99)
- L’uso di costanti *regex*, p.es. `/pippo/`, come espressioni, che equivalgono a usare l’operatore di ricerca di una corrispondenza, p.es. `$0 ~ /pippo/` (si veda la [Sezione 6.1.2 \[Usare espressioni regolari come costanti\]](#), pagina 117)
- Gestione di sequenze di protezione nell’assegnamento di variabili effettuato tramite la riga di comando (si veda la [Sezione 6.1.3.2 \[Assegnare una variabile dalla riga di comando\]](#), pagina 120)

### A.3 Differenze tra versione SVR4 e POSIX di awk

Lo standard POSIX Command Language and Utilities per `awk` (1992) ha introdotto le seguenti modifiche al linguaggio:

- L'uso dell'opzione `-W` per opzioni specifiche a una data implementazione (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
- L'uso di `CONVFMT` per controllare la conversione di numeri in stringhe (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121)
- Il concetto di stringa numerica e regole di confronto più precise da seguire al riguardo (si veda la [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), pagina 130)
- L'uso di variabili predefinite come nomi di parametri delle funzioni è vietato (si veda la [Sezione 9.2.1 \[Come scrivere definizioni e cosa significano\]](#), pagina 224)
- Una documentazione più completa di molte tra le funzionalità del linguaggio precedentemente non documentate

Nel 2012, un certo numero di estensioni che erano già comunemente disponibili da parecchi anni sono state finalmente aggiunte allo standard POSIX. Ecco l'elenco:

- La funzione predefinita `fflush()` per forzare la scrittura dei buffer in output (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210)
- L'istruzione `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160)
- La possibilità di eliminare completamente un vettore con l'istruzione `delete vettore` (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187)

Si veda la [Sezione A.7 \[Sommario Estensioni Comuni\]](#), pagina 473, per una lista delle estensioni comuni non previste nello standard POSIX.

Lo standard POSIX 2008 è reperibile online a: <http://www.opengroup.org/onlinepubs/9699919799/>.

### A.4 Estensioni nell'awk di Brian Kernighan

Brian Kernighan ha reso disponibile la sua versione nel suo sito. (si veda la [Sezione B.5 \[Altre implementazioni di awk liberamente disponibili\]](#), pagina 494).

Questa sezione descrive estensioni comuni disponibili per la prima volta nella sua versione di `awk`:

- Gli operatori `**` e `**=` (si veda la [Sezione 6.2.1 \[Operatori aritmetici\]](#), pagina 123, e [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126)
- L'uso di `func` come abbreviazione di `function` (si veda la [Sezione 9.2.1 \[Come scrivere definizioni e cosa significano\]](#), pagina 224)
- La funzione predefinita `fflush()` per forzare la scrittura dei buffer in output (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210)

Si veda la [Sezione A.7 \[Sommario Estensioni Comuni\]](#), pagina 473, per una lista completa delle estensioni disponibile nel suo `awk`.

## A.5 Estensioni di gawk non in POSIX awk

L'implementazione GNU di **gawk** aggiunge molte funzionalità. Queste possono essere disabilitate completamente sia con l'opzione `--traditional` che con l'opzione `--posix` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

Alcune funzionalità sono state introdotte e successivamente tolte con il passare del tempo. Questa sezione sintetizza le ulteriori funzionalità rispetto a POSIX **awk** che sono presenti nella versione corrente di **gawk**.

- Ulteriori variabili predefinite:
  - Le variabili `ARGIND`, `BINMODE`, `ERRNO`, `FIELDWIDTHS`, `FPAT`, `IGNORECASE`, `LINT`, `PROCINFO`, `RT` e `TEXTDOMAIN` (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162)
- File speciali verso cui ridirigere l'I/O:
  - I file `/dev/stdin`, `/dev/stdout`, `/dev/stderr` e i nomi-file speciali `/dev/fd/N` (si veda la [Sezione 5.8 \[Nomi-file speciali in gawk\]](#), pagina 108)
  - I file speciali `/inet`, `/inet4` e `/inet6` per interagire con la rete TCP/IP usando `'|&'` per specificare quale versione usare del protocollo IP (si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341)
- Differenze e/o aggiunte al linguaggio:
  - La sequenza di protezione `'\x'` (si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50)
  - Supporto completo per *regex* sia POSIX che GNU (si veda il [Capitolo 3 \[Espressioni regolari\]](#), pagina 49)
  - La possibilità che `FS` e il terzo argomento di `split()` siano la stringa nulla (si veda la [Sezione 4.5.3 \[Fare di ogni carattere un campo separato\]](#), pagina 73)
  - La possibilità che `RS` sia una *regex* (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63)
  - La possibilità di usare costanti ottali ed esadecimali nei programmi scritti in **awk** (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115)
  - L'operatore `'|&'` per poter effettuare I/O bidirezionale verso un coprocesso (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339)
  - Chiamate indirette di funzione (si veda la [Sezione 9.3 \[Chiamate indirette di funzione\]](#), pagina 234)
  - La possibilità di ignorare *directory* specificate sulla riga di comando, emettendo un messaggio di avvertimento (si veda la [Sezione 4.12 \[Directory sulla riga di comando\]](#), pagina 92)
  - Errori in output usando `print` e `printf` non provocano necessariamente la fine del programma (si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112)
- Nuove parole chiave:
  - I criteri di ricerca speciali `BEGINFILE` ed `ENDFILE` (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), pagina 150)
  - L'istruzione `switch` (si veda la [Sezione 7.4.5 \[L'istruzione switch\]](#), pagina 156)

- Differenze in funzioni standard di **awk**:
  - Il secondo argomento opzionale di **close()** che consente di chiudere un solo lato dell'I/O di una *pipe* bidirezionale aperta verso un coprocesso (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339)
  - Aderenza allo standard POSIX per le funzioni **gsub()** e **sub()** se è stata specificata l'opzione **--posix**
  - La funzione **length()** accetta come argomento il nome di un vettore e restituisce il numero di elementi nel vettore (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
  - Il terzo argomento opzionale della funzione **match()** per contenere eventuali sottoespressioni individuate all'interno di una *regex* (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
  - Specificatori posizionali nei formati di **printf** per facilitare le traduzioni di messaggi (si veda la [Sezione 13.4.2 \[Riordinare argomenti di printf\]](#), pagina 354)
  - L'aggiunta di un quarto argomento opzionale alla funzione **split()**, per designare un vettore che contenga il testo dei separatori di campo (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
- Ulteriori funzioni presenti solo in **gawk**:
  - Le funzioni **gensub()**, **patsplit()** e **strtonum()** per una gestione di testi più potente (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198)
  - Le funzioni **asort()** e **asorti()** per l'ordinamento di vettori (si veda la [Sezione 12.2 \[Controllare la visita di un vettore e il suo ordinamento\]](#), pagina 332)
  - Le funzioni **mktime()**, **sysstime()** e **strftime()** per lavorare con date e ore (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214)
  - Le funzioni **and()**, **compl()**, **lshift()**, **or()**, **rshift()** e **xor()** per la manipolazione a livello di bit (si veda la [Sezione 9.1.6 \[Funzioni per operazioni di manipolazione bit\]](#), pagina 219)
  - La funzione **isarray()** per controllare se una variabile è un vettore oppure no (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), pagina 223)
  - Le funzioni **bindtextdomain()**, **dcgettext()** e **dcngettext()** per l'internazionalizzazione (si veda la [Sezione 13.3 \[Internazionalizzare programmi awk\]](#), pagina 352)
  - La funzione **intdiv()** per effettuare divisioni a numeri interi e ottenere il resto della divisione (si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), pagina 196)
- Modifiche e/o aggiunte alle opzioni della riga di comando:
  - La variabile d'ambiente **AWKPATH** per specificare un percorso di ricerca per l'opzione **-f** della riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
  - La variabile d'ambiente **AWKLIBPATH** per specificare un percorso di ricerca per l'opzione **-l** della riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
  - Le opzioni brevi **-b**, **-c**, **-C**, **-d**, **-D**, **-e**, **-E**, **-g**, **-h**, **-i**, **-l**, **-L**, **-M**, **-n**, **-N**, **-o**, **-O**, **-p**, **-P**, **-r**, **-s**, **-S**, **-t** e **-V**. Inoltre, la possibilità di usare opzioni in

formato lungo (stile GNU) che iniziano con `--` e le opzioni lunghe `--assign`, `--bignum`, `--characters-as-bytes`, `--copyright`, `--debug`, `--dump-variables`, `--exec`, `--field-separator`, `--file`, `--gen-pot`, `--help`, `--include`, `--lint`, `--lint-old`, `--load`, `--non-decimal-data`, `--optimize`, `--no-optimize`, `--posix`, `--pretty-print`, `--profile`, `--re-interval`, `--sandbox`, `--source`, `--traditional`, `--use-lc-numeric`, and `--version` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).

- Il supporto per i seguenti sistemi obsoleti è stato rimosso dal codice sorgente e dalla documentazione di `gawk` versione 4.0:
  - Amiga
  - Atari
  - BeOS
  - Cray
  - MIPS RiscOS
  - MS-DOS con il compilatore Microsoft
  - MS-Windows con il compilatore Microsoft
  - NeXT
  - SunOS 3.x, Sun 386 (Road Runner)
  - Tandem (non-POSIX)
  - Compilatore pre-standard VAX C per VAX/VMS
  - GCC per VAX e Alpha non è stato verificato da parecchio tempo.
- Il supporto per i seguenti sistemi obsoleti è stato rimosso dal codice di `gawk` versione 4.1:
  - Ultrix
- Il supporto per i seguenti sistemi obsoleti è stato rimosso dal codice sorgente e dalla documentazione di `gawk` versione 4.2:
  - MirBSD
  - GNU/Linux su Alpha

## A.6 Storia delle funzionalità di `gawk`

Questa sezione descrive le funzionalità in `gawk` in aggiunta a quelle di POSIX `awk`, nell'ordine in cui sono state rese disponibili in `gawk`.

La versione 2.10 di `gawk` ha introdotto le seguenti funzionalità:

- La variabile d'ambiente `AWKPATH` per specificare un percorso di ricerca per l'opzione `-f` della riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33)
- La variabile `IGNORECASE` e i suoi effetti (si veda la [Sezione 3.8 \[Fare confronti ignorando maiuscolo/minuscolo\]](#), pagina 60).
- I file `/dev/stdin`, `/dev/stdout`, `/dev/stderr` e i nomi-file speciali `/dev/fd/N` (si veda la [Sezione 5.8 \[Nomi-file speciali in `gawk`\]](#), pagina 108)

La versione 2.13 di `gawk` ha introdotto le seguenti funzionalità:

- La variabile `FIELDWIDTHS` e i suoi effetti (si veda la [Sezione 4.6 \[Leggere campi di larghezza costante\]](#), pagina 77).

- Le funzioni predefinite `system()` e `strftime()` per ottenere e stampare data e ora (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214).
- Ulteriori opzioni dalla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):
  - L'opzione `-W lint` per fornire controlli su possibili errori e per la portabilità, sia a livello di codice sorgente che in fase di esecuzione.
  - L'opzione `-W compat` per inibire le estensioni GNU.
  - L'opzione `-W posix` per richiedere una stretta aderenza allo standard POSIX.

La versione 2.14 di `gawk` ha introdotto le seguenti funzionalità:

- L'istruzione `next file` per passare immediatamente al successivo file-dati (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160).

La versione 2.15 di `gawk` ha introdotto le seguenti funzionalità:

- Nuove variabili (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162):
  - `ARGIND`, che permette di controllare la posizione di `FILENAME` nel vettore `ARGV`.
  - `ERRNO`, che contiene il messaggio di errore del sistema quando `getline` restituisce `-1` o `close()` non termina con successo.
- I nomi-file speciali `/dev/pid`, `/dev/ppid`, `/dev/pgrp` e `/dev/user`. Questo supporto è stato rimosso in seguito.
- La possibilità di cancellare un intero vettore in una sola istruzione con `'delete vettore'` (si veda la [Sezione 8.4 \[L'istruzione delete\]](#), pagina 187).
- Modifiche nelle opzioni della riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):
  - La possibilità di usare opzioni in formato lungo (in stile GNU) che iniziano con `--`.
  - L'opzione `--source` per combinare codice sorgente immesso nella riga di comando e codice sorgente proveniente da file di libreria.

La versione 3.0 di `gawk` ha introdotto le seguenti funzionalità:

- Variabili nuove o modificate:
  - `IGNORECASE` modificato, diventa applicabile al confronto tra stringhe, come pure alle operazioni su `regex` (si veda la [Sezione 3.8 \[Fare confronti ignorando maiuscolo/minuscolo\]](#), pagina 60).
  - `RT`, che contiene il testo in input che è stato individuato da `RS` (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63).
- Supporto completo sia per le `regex` POSIX sia per quelle GNU (si veda il [Capitolo 3 \[Espressioni regolari\]](#), pagina 49).
- La funzione `gensub()` per migliorare la manipolazione di testi (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- La funzione `strftime()` prevede un formato di data e ora di default, in modo da poter essere chiamata senza alcun argomento. (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214).
- La possibilità che `FS` e il terzo argomento della funzione `split()` siano delle stringhe nulle (si veda la [Sezione 4.5.3 \[Fare di ogni carattere un campo separato\]](#), pagina 73).

- La possibilità che `RS` sia una *regex* (si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63).
- L'istruzione `next file` è diventata `nextfile` (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160).
- La funzione `fflush()` di BWK `awk` (BWK allora lavorava ai Bell Laboratories; si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210).
- Nuove opzioni della riga di comando:
  - L'opzione `--lint-old` per ottenere messaggi relativi a costrutti non disponibili nell'implementazione di `awk` per Unix Version 7 (si veda la [Sezione A.1 \[Differenze importanti tra V7 e System V Release 3.1\]](#), pagina 461).
  - L'opzione `-m` da BWK `awk`. (Brian lavorava ancora ai Bell Laboratories all'epoca.) Quest'opzione è stata in seguito rimossa, sia dal suo `awk` che da `gawk`.
  - L'opzione `--re-interval` per consentire di specificare espressioni di intervallo nelle *regex* (si veda la [Sezione 3.3 \[Operatori di espressioni regolari\]](#), pagina 52).
  - L'opzione `--traditional` aggiunta come maniera più intuitiva per richiedere l'opzione `--compat` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33).
- L'uso di GNU Autoconf per controllare il processo di configurazione (si veda la [Sezione B.2.1 \[Compilare gawk per sistemi di tipo Unix\]](#), pagina 483).
- Supporto per Amiga. Questo supporto è stato rimosso in seguito.

La versione 3.1 di `gawk` ha introdotto le seguenti funzionalità:

- Nuove variabili (si veda la [Sezione 7.5 \[Variabili predefinite\]](#), pagina 162):
  - `BINMODE`, per sistemi non aderenti allo standard POSIX, che consente I/O binario per file in input e/o output (si veda la [Sezione B.3.1.3 \[Usare gawk su sistemi operativi PC\]](#), pagina 487).
  - `LINT`, che controlla dinamicamente gli avvertimenti emessi da *lint*.
  - `PROCINFO`, un vettore che fornisce informazioni correlate con il processo in esecuzione.
  - `TEXTDOMAIN`, per impostare il dominio testuale in cui internazionalizzare un'applicazione (si veda la [Capitolo 13 \[Internazionalizzazione con gawk\]](#), pagina 349).
- La possibilità di usare costanti ottali ed esadecimali nel codice sorgente di programmi `awk`. (si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115).
- L'operatore `'|&'` per effettuare I/O bidirezionale verso un coprocesso (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339).
- I file speciali `/inet` per interagire con reti TCP/IP usando `'|&'` (si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341).
- Il secondo argomento opzionale della funzione `close()` per permettere di chiudere uno dei lati di una *pipe* bidirezionale aperta con un coprocesso (si veda la [Sezione 12.3 \[Comunicazioni bidirezionali con un altro processo\]](#), pagina 339).
- Il terzo argomento opzionale della funzione `match()` per avere a disposizione le diverse sottoespressioni individuate all'interno di una *regex* (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).

- Specificatori posizionali nelle stringhe di formato di `printf` per facilitare la traduzione di messaggi (si veda la [Sezione 13.4.2 \[Riordinare argomenti di printf\]](#), pagina 354).
- Alcune nuove funzioni predefinite:
  - Le funzioni `asort()` e `asorti()` per l'ordinamento di vettori (si veda la [Sezione 12.2 \[Controllare la visita di un vettore e il suo ordinamento\]](#), pagina 332).
  - Le funzioni `bindtextdomain()`, `dcgettext()` e `dcngettext()` per l'internazionalizzazione (si veda la [Sezione 13.3 \[Internazionalizzare programmi awk\]](#), pagina 352).
  - La funzione `extension()` e la possibilità di aggiungere nuove funzioni predefinite dinamicamente (si veda il [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395).
  - La funzione `mktime()` per generare date e ore (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214).
  - Le funzioni `and()`, `or()`, `xor()`, `compl()`, `lshift()`, `rshift()` e `strtonum()` (si veda la [Sezione 9.1.6 \[Funzioni per operazioni di manipolazione bit\]](#), pagina 219).
- Il supporto per 'next file' scritto come due parole è stato rimosso completamente (si veda la [Sezione 7.4.9 \[L'istruzione nextfile\]](#), pagina 160).
- Ulteriori opzioni sulla riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):
  - L'opzione `--dump-variables` per stampare una lista di tutte le variabili globali.
  - L'opzione `--exec`, da usare in script CGI [Common Gateway Interface].
  - L'opzione della riga di comando `--gen-po` e l'uso di un trattino basso a inizio stringa, per segnalare stringhe che dovrebbero essere tradotte (si veda la [Sezione 13.4.1 \[Estrarre stringhe marcate\]](#), pagina 354).
  - L'opzione `--non-decimal-data` per consentire di avere dati in input di tipo non decimale (si veda la [Sezione 12.1 \[Consentire dati di input non decimali\]](#), pagina 331).
  - L'opzione `--profile` e `pgawk`, la versione profilatrice di `gawk`, per produrre profili di esecuzione di programmi `awk` (si veda la [Sezione 12.5 \[Profilare i propri programmi awk\]](#), pagina 343).
  - L'opzione `--use-lc-numeric` per richiedere a `gawk` di usare il carattere di separazione decimale proprio della localizzazione nell'elaborazione dei dati in input (si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), pagina 121).
- L'uso di GNU Automake a supporto della standardizzazione del processo di configurazione (si veda la [Sezione B.2.1 \[Compilare gawk per sistemi di tipo Unix\]](#), pagina 483).
- L'uso di GNU `gettext` per i messaggi emessi da `gawk` (si veda la [Sezione 13.6 \[gawk stesso è internazionalizzato\]](#), pagina 358).
- Supporto per BeOS. Rimosso in seguito.
- Supporto per Tandem. Rimosso in seguito.
- La versione per Atari ufficialmente non è più supportata e in seguito è stata completamente rimossa.
- Modifiche al codice sorgente per usare definizioni di funzione secondo lo stile di codifica dello standard ISO C.

- Aderenza alla specifica POSIX per le funzioni `sub()` e `gsub()` (si veda la [Sezione 9.1.3.1 \[Ulteriori dettagli su ‘\’ e ‘&’ con `sub\(\)`, `gsub\(\)` e `gensub\(\)`\]](#), pagina 207).
- La funzione `length()` è stata estesa per accettare un vettore come argomento, e restituire in tal caso il numero di elementi nel vettore (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- La funzione `strftime()` accetta un terzo argomento per dare la possibilità di stampare data e ora nel formato UTC (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214).

La versione 4.0 di `gawk` ha introdotto le seguenti funzionalità:

- Aggiunta di variabili:
  - `FPAT`, che permette di specificare una *regex* che individua i campi, invece che individuare il separatore tra i campi (si veda la [Sezione 4.7 \[Definire i campi in base al contenuto\]](#), pagina 79).
  - Se esiste l'elemento di vettore `PROCINFO["sorted_in"]`, il ciclo `for(indice in pippo)` ordina gli indici, prima di iniziare il ciclo. Il valore di questo elemento permette di controllare l'ordinamento degli indici prima di iniziare il ciclo che li visita tutti (si veda la [Sezione 8.1.6 \[Visita di vettori in ordine predefinito con `gawk`\]](#), pagina 182).
  - `PROCINFO["strftime"]`, che contiene la stringa di formato di default per `strftime()` (si veda la [Sezione 9.1.5 \[Funzioni per gestire marcature temporali\]](#), pagina 214).
- I file speciali `/dev/pid`, `/dev/ppid`, `/dev/pgrp` e `/dev/user` sono stati rimossi.
- Il supporto per IPv6 è stato aggiunto attraverso il file speciale `/inet6`. Il file speciale `/inet4` consente di operare con IPv4 e `/inet` opera con il default di sistema, che probabilmente è IPv4 (si veda la [Sezione 12.4 \[Usare `gawk` per la programmazione di rete\]](#), pagina 341).
- L'uso delle sequenze di protezione `'\s'` e `'\S'` nelle espressioni regolari (si veda la [Sezione 3.7 \[Operatori \*regex\* propri di `gawk`\]](#), pagina 59).
- Le espressioni di intervallo sono consentite per default nelle espressioni regolari (si veda la [Sezione 3.3 \[Operatori di espressioni regolari\]](#), pagina 52).
- La classi di caratteri POSIX sono consentite anche se si è specificata l'opzione `--traditional` (si veda la [Sezione 3.3 \[Operatori di espressioni regolari\]](#), pagina 52).
- `break` e `continue` non sono più consentiti fuori da un ciclo, anche se si è specificata l'opzione `--traditional` (si veda la [Sezione 7.4.6 \[L'istruzione `break`\]](#), pagina 157, e anche la [Sezione 7.4.7 \[L'istruzione `continue`\]](#), pagina 158).
- `fflush()`, `nextfile` e `'delete array'` sono consentite anche se è stata specificata l'opzione `--posix` o `--traditional`, poiché questi costrutti sono ora inclusi nello standard POSIX.
- Un terzo argomento facoltativo per le funzioni `asort()` e `asorti()` permette di specificare il tipo di ordinamento desiderato (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- Il comportamento di `fflush()` è stato modificato per corrispondere a quello di `BWK awk` e per lo standard POSIX; ora sia `'fflush()'` che `'fflush("")'` forzano la scrit-

tura di tutte le ridirezioni in output aperte (si veda la [Sezione 9.1.4 \[Funzioni di Input/Output\]](#), pagina 210).

- La funzione `isarray()` determina se un elemento è un vettore oppure no per rendere possibile la visita di vettori di vettori (si veda la [Sezione 9.1.7 \[Funzioni per conoscere il tipo di una variabile\]](#), pagina 223).
- La funzione `patsplit()` che fornisce le stesse funzionalità di `FPAT`, per suddividere delle stringhe (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- Un quarto argomento opzionale per la funzione `split()`, che indica un vettore destinato a contenere i valori dei separatori (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198).
- Vettori di vettori (si veda la [Sezione 8.6 \[Vettori di vettori\]](#), pagina 190).
- I criteri di ricerca speciali `BEGINFILE` ed `ENDFILE` (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali BEGINFILE ed ENDFILE\]](#), pagina 150).
- Chiamate indirette di funzioni (si veda la [Sezione 9.3 \[Chiamate indirette di funzione\]](#), pagina 234).
- Le istruzioni `switch` / `case` sono disponibili per default (si veda la [Sezione 7.4.5 \[L'istruzione switch\]](#), pagina 156).
- Modifiche nelle opzioni della riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):
  - Le opzioni `-b` e `--characters-as-bytes`, che impediscono che `gawk` tratti l'input come composto da una stringa di caratteri multibyte.
  - Rimozione delle opzioni ridondanti (in notazione lunga) `--compat`, `--copyleft` e `--usage`.
  - L'opzione `--gen-po` è stata finalmente rinominata `--gen-pot` per correttezza.
  - L'opzione `--sandbox` che disabilita alcune funzionalità [per operare in un ambiente "protetto"].
  - Tutte le opzioni in notazione lunga hanno acquisito opzioni corrispondenti in notazione breve, per poter essere usate negli script di shell `'#!'`.
- I nomi di directory che appaiono sulla riga di comando generano adesso un messaggio di errore, ma non interrompono l'elaborazione, a meno che non siano state specificate le opzioni `--posix` o `--traditional` (si veda la [Sezione 4.12 \[Directory sulla riga di comando\]](#), pagina 92).
- Il codice interno di `gawk` è stato riscritto, aggiungendo la versione per il debug `dgawk`, con un possibile miglioramento nei tempi di esecuzione (si veda il [Capitolo 14 \[Effettuare il debug dei programmi awk\]](#), pagina 361).
- In aderenza agli standard di codifica GNU, le estensioni dinamiche devono definire un simbolo globale che indica che sono compatibili con la licenza GPL (si veda la [Sezione 16.2 \[Tipo di licenza delle estensioni\]](#), pagina 395).
- In modalità POSIX, i confronti tra stringhe usano le funzioni di libreria `strcoll()` / `wscoll()` (si veda la [Sezione 6.3.2.3 \[Confronto tra stringhe usando l'ordine di collazione locale\]](#), pagina 135).
- L'opzione per usare `socket` in maniera `raw` (nativa) è stata rimossa, perché non era mai stata implementata (si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341).

- Intervalli nella forma '[d-h]' sono elaborati come se fossero scritti nella localizzazione C, a prescindere da che tipo di *regex* è usata, anche se era stata specificata l'opzione `--posix` (si veda la [Sezione A.8 \[Intervalli \*regex\* e localizzazione: una lunga e triste storia\]](#), pagina 474).
- È stato rimosso il supporto per i seguenti sistemi:
  - Atari
  - Amiga
  - BeOS
  - Cray
  - MIPS RiscOS
  - MS-DOS con Compilatore Microsoft
  - MS-Windows con Compilatore Microsoft
  - NeXT
  - SunOS 3.x, Sun 386 (Road Runner)
  - Tandem (non-POSIX)
  - Compilatore pre-standard VAX C per VAX/VMS

La versione 4.1 di **gawk** ha introdotto le seguenti funzionalità:

- Tre nuovi vettori: `SYMTAB`, `FUNCTAB` e `PROCINFO["identifiers"]` (si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\]](#), pagina 165).
- I tre comandi eseguibili **gawk**, **pgawk** e **dgawk**, sono diventati uno solo, con il solo nome **gawk**. Di conseguenza le opzioni sulla riga di comando sono state modificate.
- Modifiche delle opzioni da riga di comando (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33):
  - L'opzione `-D` attiva il debugger.
  - Le opzioni `-i` e `--include` caricano dei file di libreria **awk**.
  - Le opzioni `-l` e `--load` caricano estensioni dinamiche compilate.
  - Le opzioni `-M` e `--bignum` abilitano la libreria MPFR per il calcolo con un numero arbitrario di cifre significative.
  - L'opzione `-o` serve solo a ottenere in output una stampa formattata elegantemente del programma da eseguire.
  - L'opzione `-p` è usata per "profilare" l'esecuzione del programma.
  - L'opzione `-R` è stata rimossa.
- Supporto per il calcolo ad alta precisione con MPFR (si veda la [Capitolo 15 \[Calcolo con precisione arbitraria con \*\*gawk\*\*\]](#), pagina 379).
- Le funzioni `and()`, `or()` e `xor()` sono state modificate per ammettere un numero qualsiasi di argomenti, con un minimo di due (si veda la [Sezione 9.1.6 \[Funzioni per operazioni di manipolazione bit\]](#), pagina 219).
- L'interfaccia che rende possibile l'estensione dinamica è stata rifatta completamente (si veda il [Capitolo 16 \[Scrivere estensioni per \*\*gawk\*\*\]](#), pagina 395).
- La funzione `getline` ridiretta è stata resa possibile all'interno di `BEGINFILE` ed `ENDFILE` (si veda la [Sezione 7.1.5 \[I criteri di ricerca speciali `BEGINFILE` ed `ENDFILE`\]](#), pagina 150).

- Il comando **where** è stato aggiunto al debugger (si veda la [Sezione 14.3.4 \[Lavorare con lo stack\]](#), pagina 371).
- Il supporto per Ultrix è stato rimosso.

La versione 4.2 ha introdotto le seguenti funzionalità:

- Differenze apportate alle variabili di ambiente (**ENVIRON**) sono riflesse in quelle rese disponibili a **gawk** e in quelle di programmi che siano da esso richiamati. Si veda la [Sezione 7.5.2 \[Variabili predefinite con cui \*\*awk\*\* fornisce informazioni\]](#), pagina 165.
- L'opzione **--pretty-print** non esegue più, dopo averlo stampato, il programma **awk**. Si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33.
- Il programma **igawk** e le relative pagine di manuale non sono più installati come parte dell'installazione di **gawk**. Si veda la [Sezione 11.3.9 \[Una maniera facile per usare funzioni di libreria\]](#), pagina 317.
- La funzione **intdiv()**. Si veda la [Sezione 9.1.2 \[Funzioni numeriche\]](#), pagina 196.
- Il massimo numero di cifre esadecimali permesse nelle sequenze di protezione **'\x'** è ora limitato a due. Si veda la [Sezione 3.2 \[Sequenze di protezione\]](#), pagina 50.
- **print** e **printf** non terminano il programma dopo alcuni errori di output. Si veda la [Sezione 5.10 \[Abilitare continuazione dopo errori in output\]](#), pagina 112.
- Per molti anni, lo standard POSIX richiedeva che la separazione dei campi di un record fosse fatta per default quando si incontrano spazi e TAB, e questo è il comportamento di **gawk** se si specifica l'opzione **--posix**. Dal 2013 il comportamento originario è stato ripristinato, e ora il default per separare i campi con l'opzione **--posix** ammette anche il ritorno a capo come separatore di campi.
- Il supporto per MirBSD è stato rimosso.
- Il supporto per GNU/Linux sull'architettura Alpha è stato rimosso.

## A.7 Sommario Estensioni Comuni

La tabella seguente dettaglia le estensioni comuni supportate da **gawk**, da Brian Kernighan **awk** e da **mawk**, le tre versioni liberamente disponibili più usate di **awk** (si veda la [Sezione B.5 \[Altre implementazioni di \*\*awk\*\* liberamente disponibili\]](#), pagina 494).

Funzionalità	BWK awk	mawk	gawk	Standard attuale
Sequenza di protezione <b>'\x'</b>	X	X	X	
Stringa nulla come FS	X	X	X	
File speciale <b>/dev/stdin</b>	X	X	X	
File speciale <b>/dev/stdout</b>	X	X	X	
File speciale <b>/dev/stderr</b>	X	X	X	
<b>delete</b> senza indici	X	X	X	X
Funzione <b>fflush()</b>	X	X	X	X
<b>length()</b> di un vettore	X	X	X	
Istruzione <b>nextfile</b>	X	X	X	X
Operatori <b>**</b> e <b>**=</b>	X		X	
Parola chiave <b>func</b>	X		X	
Variabile <b>BINMODE</b>		X	X	

RS come <i>regexp</i>	X	X
Funzioni gestione data/ora	X	X

## A.8 Intervalli *regexp* e localizzazione: una lunga e triste storia

Questa sezione descrive la storia confusionaria degli intervalli all'interno di espressioni regolari, le loro relazioni con la localizzazione, e l'effetto da ciò determinato su diverse versioni di **gawk**.

Gli strumenti originali Unix aventi a che fare con espressioni regolari stabilivano che intervalli di caratteri (come '[a-z]') individuavano un carattere qualsiasi tra il primo carattere dell'intervallo e l'ultimo carattere dello stesso, entrambi inclusi. L'ordinamento era basato sul valore numerico di ogni carattere come era rappresentato all'interno del computer, nell'insieme di caratteri proprio di ogni macchina. Quindi, su sistemi che adottano la codifica ASCII, '[a-z]' individua tutte le lettere minuscole, e solo quelle, in quanto i valori numerici che rappresentano le lettere dalla 'a' fino alla 'z' sono contigui. (In un sistema che adotta la codifica EBCDIC, l'intervallo '[a-z]' comprende anche ulteriori caratteri non alfabetici.)

Quasi tutti i testi di introduzione allo Unix spiegavano che le espressioni di intervallo funzionavano in questo modo, e in particolare insegnavano che la maniera "corretta" per individuare le lettere minuscole era con '[a-z]' e che '[A-Z]' era il modo "corretto" per individuare le lettere maiuscole. E, in effetti, era proprio così.<sup>1</sup>

Lo standard POSIX 1992 introduceva l'idea di localizzazione (si veda la [Sezione 6.6 \[Il luogo fa la differenza\]](#), pagina 141). Poiché molte localizzazioni comprendono altre lettere, oltre alle 26 lettere dell'alfabeto inglese, lo standard POSIX introduceva le classi di carattere (si veda la [Sezione 3.4 \[Usare espressioni tra parentesi quadre\]](#), pagina 55) per permettere l'individuazione di differenti insiemi di caratteri, in aggiunta a quelli tradizionali presenti nell'insieme di caratteri ASCII.

Tuttavia, lo standard *ha modificato* l'interpretazione delle espressioni di intervallo. Nelle localizzazioni "C" e "POSIX", un'espressione di intervallo come '[a-dx-z]' è ancora equivalente a '[abcdxyz]', secondo l'ordine della codifica ASCII. Ma in tutte le altre localizzazioni l'ordinamento è basato su quel che si chiama *ordine di collazione*.

Cosa vuol dire? In molte localizzazioni, le lettere 'A' e 'a' vengono entrambe prima di 'B'. In altre parole, queste localizzazioni ordinano i caratteri nel modo in cui sono ordinati in un dizionario, e '[a-dx-z]' non è detto che equivalga a '[abcdxyz]'; invece, potrebbe essere equivalente a '[ABCXYabcdxyz]', per fare un esempio.

Su questo punto è opportuno insistere: molta documentazione afferma che si dovrebbe usare '[a-z]' per identificare un carattere minuscolo. Ma su sistemi con localizzazioni non-ASCII, un tale intervallo potrebbe includere tutti i caratteri maiuscoli tranne 'A' o 'Z'! Questo ha continuato a essere una fonte di equivoci perfino nel ventunesimo secolo.

Per dare un'idea del tipo di problemi, l'esempio seguente usa la funzione `sub()`, che effettua una sostituzione di testo all'interno di una stringa (si veda la [Sezione 9.1.3 \[Funzioni di manipolazione di stringhe\]](#), pagina 198). Qui, l'idea è quella di rimuovere i caratteri maiuscoli a fine stringa:

---

<sup>1</sup> E la vita era semplice.

```
$ echo qualcosa1234abc | gawk-3.1.8 '{ sub("[A-Z]*$", ""); print }'
+ qualcosa1234a
```

Questo non è l'output che ci si aspettava, perché, il 'bc' alla fine di 'qualcosa1234abc' non dovrebbe essere individuato da '[A-Z]\*'. Un tale risultato dipende dalle impostazioni di localizzazione (e quindi potrebbe non succedere sul sistema che si sta usando).

Considerazioni simili valgono per altri intervalli. Per esempio, '['-/' è perfettamente valido in ASCII, ma non è valido in molte localizzazioni Unicode, p.es. in `en_US.UTF-8`.

Il codice delle prime versioni di **gawk** per individuare le *regex* non teneva conto della localizzazione, e quindi gli intervalli potevano essere interpretati in maniera tradizionale.

Quando **gawk** ha iniziato a usare metodi di ricerca di *regex* che tengono conto della localizzazione, sono iniziati i problemi; a maggior ragione in quanto sia GNU/Linux che i venditori di versioni commerciali di Unix avevano iniziato a implementare localizzazioni non-ASCII, *adottandole per default*. La domanda che forse si udiva più spesso era del tipo: "Perché '[A-Z]' individua lettere minuscole!?"

Questa situazione è in essere da circa 10 anni, se non di più, e il manutentore di **gawk** si è stufato di continuare a spiegare che **gawk** stava semplicemente implementando quelli che sono gli standard, e che il problema stava nella localizzazione dell'utente. Nella fase di sviluppo della versione 4.0, **gawk** è stato modificato in modo da trattare sempre gli intervalli "come si faceva prima di POSIX", a meno che non si specifichi l'opzione `--posix` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\], pagina 33](#)).<sup>2</sup>

Fortunatamente, un po' prima del rilascio definitivo della versione 4.0 di **gawk**, il manutentore ha appreso che lo standard 2008 aveva modificato la definizione di intervallo, e che, al di fuori delle localizzazioni "C" e "POSIX", il significato di espressione di intervallo era ora *indefinito*.<sup>3</sup>

Adottando questo simpatico termine tecnico, lo standard permette agli implementatori di implementare gli intervalli nella maniera che preferiscono. Il manutentore di **gawk** ha deciso di implementare la regola pre-POSIX sia per l'individuazione di default delle *regex* sia quando si specificano le opzioni `--traditional` o `--posix`. In ogni caso **gawk** aderisce allo standard POSIX.

## A.9 I principali contributori a gawk

*Riconoscere sempre il merito, se un merito va riconosciuto.*

—Anonimo

Questa sezione elenca le persone che hanno maggiormente contribuito allo sviluppo di **gawk** e/o alla stesura di questo libro, in ordine approssimativamente cronologico:

- Il Dr. Alfred V. Aho, il Dr. Peter J. Weinberger, e il Dr. Brian W. Kernighan, tutti dei Bell Laboratories, hanno progettato e implementato **awk** per Unix, da cui **gawk** trae la maggioranza delle sue funzionalità.
- Paul Rubin, autore del progetto e dell'implementazione iniziale del 1986, ha scritto la prima bozza (di circa 40 pagine) di questo libro.

<sup>2</sup> Ed è così che è nata la Campagna per l'Interpretazione Razionale degli Intervalli (in inglese, RRI [*Rational Range Interpretation*]). Un certo numero di strumenti GNU hanno già implementato questa modifica, o lo faranno presto. Grazie a Karl Berry per aver coniato la frase "Rational Range Interpretation".

<sup>3</sup> Si veda [lo standard](#) e [le motivazioni](#).

- Jay Fenlason ha completato l'implementazione iniziale.
- Diane Close ha rivisto la prima bozza di questo libro, portandolo alla lunghezza di circa 90 pagine.
- Richard Stallman ha aiutato a completare l'implementazione e la bozza iniziale di questo libro. È anche il fondatore della FSF e del progetto GNU.
- John Woods ha scritto porzioni di codice (volti principalmente alla correzione di errori) nella versione iniziale di **gawk**.
- Nel 1988, David Trueman si è fatto carico della manutenzione principale di **gawk**, rendendolo compatibile col “nuovo” **awk** e migliorandone parecchio la velocità di esecuzione.
- Conrad Kwok, Scott Garfinkle e Kent Williams hanno per primi portato il programma all'ambiente MS-DOS, usando varie versioni del compilatore MSC.
- Pat Rankin ha portato il programma all'ambiente VMS, preparando anche la relativa documentazione.
- Hal Peterson è stato di aiuto nel portare **gawk** nei sistemi Cray. (L'ambiente Cray non è più supportato.)
- Kai Uwe Rommel ha portato per primo il programma all'ambiente OS/2, preparando anche la relativa documentazione.
- Michal Jaegermann ha portato il programma all'ambiente Atari, preparando anche la relativa documentazione. (L'ambiente Atari non è più supportato.) Michal continua a effettuare controlli di portabilità, e ha molto contribuito a consentire a **gawk** di funzionare su sistemi diversi da quelli a 32 bit.
- Fred Fish ha portato il programma all'ambiente Amiga, preparando anche la relativa documentazione. (Purtroppo Fred non è più tra noi, e questo ambiente non è più supportato.)
- Scott Deifik si è occupato della manutenzione per MS-DOS usando il compilatore DJGPP.
- Eli Zaretskii si occupa della manutenzione della versione per MS-Windows, nell'ambiente MinGW.
- Juan Grigera è autore di una versione di **gawk** per sistemi Windows32. (Questa versione non è più supportata.)
- Per molti anni, il Dr. Darrel Hankerson ha fatto da coordinatore per le varie versioni che giravano su diverse piattaforme PC e ha creato distribuzioni binarie per vari sistemi operativi che girano sui PC. Il suo aiuto è stato importante per mantenere aggiornata la documentazione per le diverse piattaforme PC.
- Christos Zoulas ha scritto la funzione predefinita **extension()** per aggiungere dinamicamente nuove funzioni. (Questa funzionalità è divenuta obsoleta a partire da **gawk** 4.1.)
- Jürgen Kahrs ha scritto la prima versione del codice per interagire con la rete TCP/IP, con la relativa documentazione, e fornito le ragioni per l'aggiunta dell'operatore **|&**.
- Stephen Davies ha portato per la prima volta il programma all'ambiente Tandem, preparando anche la relativa documentazione. (Tuttavia, questa versione non è più supportata.) Stephen è anche stato determinante nel lavoro iniziale per integrare il codice interno di gestione dei byte nel complesso del codice di **gawk**.

- Matthew Woehlke ha migliorato l'aderenza allo standard POSIX nei sistemi Tandem che implementano lo standard.
- Martin Brown ha portato il programma all'ambiente BeOS, preparando anche la relativa documentazione. (L'ambiente BeOS non è più supportato.)
- Arno Peters ha fatto il lavoro iniziale necessario per consentire alla configurazione di **gawk** di usare GNU Automake e GNU **gettext**.
- Alan J. Broder ha scritto la prima versione della funzione **asort()** e anche il codice per gestire il terzo argomento opzionale della funzione **match()**.
- Andreas Buening ha aggiornato la versione di **gawk** per OS/2.
- Isamu Hasegawa, dell'IBM in Giappone, ha contribuito con il supporto per i caratteri multibyte.
- Michael Benzinger ha sviluppato il codice iniziale per l'istruzione **switch**.
- Patrick T.J. McPhee ha sviluppato il codice per il caricamento dinamico negli ambienti Windows32. (Questa funzionalità non è più supportata.)
- Anders Wallin ha aiutato a continuare il supporto della versione VMS di **gawk** per parecchi anni.
- Assaf Gordon ha scritto il codice per implementare l'opzione **--sandbox**.
- John Haque è autore dei seguenti contributi:
  - Le modifiche per convertire **gawk** in un interprete di codice a livello di byte, compreso il debugger
  - L'aggiunta di veri vettori di vettori
  - Le modifiche ulteriori per il supporto del calcolo a precisione arbitraria
  - Il testo iniziale di [Capitolo 15 \[Calcolo con precisione arbitraria con \*\*gawk\*\*\]](#), [pagina 379](#),
  - Il lavoro per unificare le tre varianti del programma **gawk**, in vista della versione 4.1
  - I miglioramenti alla gestione interna dei vettori per i vettori i cui indici sono dei numeri interi
  - A John, insieme a Pat Rankin, si devono i miglioramenti alla funzionalità di ordinamento dei vettori.
- Panos Papadopoulos ha scritto il testo originale per [Sezione 2.7 \[Come includere altri file nel proprio programma\]](#), [pagina 45](#).
- Efraim Yawitz ha scritto il testo originale per il [Capitolo 14 \[Effettuare il debug dei programmi \*\*awk\*\*\]](#), [pagina 361](#).
- Lo sviluppo dell'estensione API rilasciata per la prima volta con **gawk** 4.1 è stata principalmente guidata da Arnold Robbins e Andrew Schorr, con notevoli contributi dal resto del team di sviluppo.
- John Malmberg ha apportato miglioramenti significativi alla versione OpenVMS e alla relativa documentazione.
- Antonio Giovanni Colombo ha riscritto diversi esempi, che non erano più attuali, contenuti nei primi capitoli, e gliene sono estremamente grato.
- Arnold Robbins ha lavorato su **gawk** dal 1988, dapprima aiutando David Trueman e in seguito, dal 1994 circa, come manutentore principale.

## A.10 Sommario

- Il linguaggio **awk** si è evoluto col passare degli anni. La prima versione risale a Unix V7, circa 1978. Nel 1987, per la versione Unix System V Release 3.1, sono state fatte al linguaggio delle modifiche importanti, inclusa la possibilità di avere funzioni definite dall'utente. Ulteriori modifiche sono state fatte per la versione System V Release 4, nel 1989. Dopo di allora, sono state apportate ulteriori modifiche minori, per implementare lo standard POSIX.
- L'**awk** di Brian Kernighan prevede un piccolo numero di estensioni implementate di comune accordo con altre versioni di **awk**.
- **gawk** prevede un elevato numero di estensioni rispetto a POSIX **awk**. Queste estensioni possono essere disabilitate specificando l'opzione `--traditional` o `--posix`.
- L'interazione tra localizzazioni POSIX e individuazione di *regex* in **gawk** è stata causa di malintesi nel corso degli anni. Oggi **gawk** implementa l'Interpretazione Razionale degli Intervalli (*Rational Range Interpretation*), dove intervalli nella forma '[a-z]' individuano *solo* i caratteri numericamente compresi tra 'a' e 'z' nella rappresentazione nativa dei caratteri in quella particolare macchina. Normalmente quella in uso è quella ASCII, ma può essere EBCDIC sui sistemi IBM S/390.
- Molte persone hanno contribuito allo sviluppo di **gawk** nel corso degli anni. Spero che l'elenco fornito in questo capitolo sia esauriente e attribuisca il giusto riconoscimento quando questo è dovuto.

## Appendice B Installare gawk

Quest'appendice contiene istruzioni per installare **gawk** sulle varie piattaforme supportate dagli sviluppatori. Lo sviluppatore principale supporta GNU/Linux (e Unix), mentre le altre piattaforme sono curate da altri sviluppatori. Si veda la [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493, per gli indirizzi di posta elettronica di chi effettua la manutenzione della versione specifica di una particolare piattaforma.

### B.1 La distribuzione di gawk

Questa sezione spiega come ottenere la distribuzione di **gawk**, come scompattarla, e cosa è contenuto nei vari file e nelle sottodirectory risultanti.

#### B.1.1 Ottenere la distribuzione di gawk

Ci sono due modi per ottenere del software GNU:

- Copiarlo da qualcuno che ce l'abbia già.
- Ottenere **gawk** dal sito Internet <ftp.gnu.org>, nella directory `/gnu/gawk`. È possibile accedere al sito sia via **ftp** anonimo che via **http**. Se si dispone del programma **wget**, si può utilizzarlo digitando un comando simile a questo:

```
wget http://ftp.gnu.org/gnu/gawk/gawk-4.1.4.tar.gz
```

L'archivio che contiene il software GNU è disponibile in vari cloni (*mirror*) in tutto il mondo. La lista aggiornata dei siti clone è disponibile nel [sito web principale della FSF](#). Si tenti di usare uno dei siti-clone; dovrebbero essere meno trafficati, ed è possibile che ce ne sia uno più vicino.

Si può anche scaricare la distribuzione del sorgente di **gawk** dal deposito Git ufficiale; per maggiori informazioni, si veda [Sezione C.2.1 \[Accedere al deposito dei sorgenti Git di gawk\]](#), pagina 499.

#### B.1.2 Scompattare la distribuzione

**gawk** è distribuito sotto forma di parecchi file **tar** compressi con differenti programmi di compressione: **gzip**, **bzip2** e **xz**. Per amor di semplicità, il resto di queste istruzioni presuppone che si stia usando quella compressa col programma GNU Gzip (**gzip**).

Una volta che si ha a disposizione la distribuzione (p.es., `gawk-4.1.4.tar.gz`), va usato **gzip** per scompattare il file e quindi **tar** per estrarne i file. Si può usare la seguente *pipe* per produrre la distribuzione **gawk**:

```
gzip -d -c gawk-4.1.4.tar.gz | tar -xvpf -
```

In un sistema che abbia la versione GNU di **tar**, si può far effettuare la scompattazione direttamente a **tar**:

```
tar -xvpzf gawk-4.1.4.tar.gz
```

L'estrazione dei file dall'archivio crea una directory di nome `gawk-4.1.4` nella directory corrente.

Il nome-file della distribuzione è nella forma `gawk-V.R.P.tar.gz`. La *V* rappresenta la versione maggiore di **gawk**, la *R* rappresenta il rilascio corrente della versione *V*, e la *P* rappresenta un *patch level*, che sta a indicare che correzioni a errori minori sono state incluse

nel rilascio. Il *patch level* corrente è 4, ma quando ci si procura una distribuzione, andrà ottenuta quella con il livello più alto di versione, rilascio e *patch*. (Si noti, comunque, che livelli di *patch* maggiori o uguali a 70 denotano versioni “beta”, ossia versioni non destinate a essere usate in produzione; non si dovrebbero utilizzare tali versioni, se non si è disposti a sperimentare.) Se non si sta usando un sistema Unix o GNU/Linux, i modi per ottenere e scompattare la distribuzione di **gawk** sono differenti. Si dovrebbe sentire un esperto di quel sistema.

### B.1.3 Contenuti della distribuzione **gawk**

La distribuzione di **gawk** contiene un certo numero di file sorgente in C, di file di documentazione, di sottodirectory, e di file utilizzati durante il processo di configurazione (si veda la [Sezione B.2 \[Compilare e installare \*\*gawk\*\* su sistemi di tipo Unix\], pagina 483](#)), come pure parecchie sottodirectory relative a diversi sistemi operativi non-Unix:

Vari file `‘.c’`, `‘.y’` e `‘.h’`

Questi file contengono il codice sorgente vero e proprio di **gawk**.

`support/*`

Intestazioni C e file sorgente per routine che **gawk** usa, ma che non sono parte della sua funzionalità fondamentale. Per esempio, analisi di argomenti, controlli di corrispondenze di espressioni regolari, e routine per generare numeri casuali sono tutti mantenuti qui.

`ABOUT-NLS`

Un file contenente informazioni sul comando GNU `gettext` e sulle traduzioni.

`AUTHORS`

Un file con alcune informazioni su chi ha scritto **gawk**. Esiste solo per placare i pedanti della Free Software Foundation.

`README`

`README_d/README.*`

File descrittivi: vari `README` (“leggimi”) per **gawk** sotto Unix e per tutte le varie altre combinazioni hardware e software.

`INSTALL`

Un file che fornisce una panoramica sul processo di configurazione e installazione.

`ChangeLog`

Una lista dettagliata delle modifiche apportate al codice sorgente, ai problemi risolti e ai miglioramenti introdotti.

`ChangeLog.0`

Una lista meno recente di modifiche al codice sorgente.

`NEWS`

Una lista di modifiche a **gawk** a partire dall’ultimo rilascio o *patch*.

`NEWS.0`

Una lista meno recente di modifiche a **gawk**.

`COPYING`

La *GNU General Public License*.

`POSIX.STD`

Una descrizione di comportamenti nello standard POSIX per **awk** che sono lasciati indefiniti, o ai quali **gawk** non può conformarsi pienamente, come pure

una lista di specifiche che lo standard POSIX dovrebbe contenere, ma che non sono presenti.

`doc/awkforai.txt`

Puntatori alla bozza originale di un breve articolo che spiega perché **gawk** è un linguaggio adatto alla programmazione nel campo dell'intelligenza artificiale (AI).

`doc/bc_notes`

Una breve descrizione della struttura interna a livello di byte di **gawk** ["byte code"].

`doc/README.card`

`doc/ad.block`

`doc/awkcard.in`

`doc/cardfonts`

`doc/colors`

`doc/macros`

`doc/no.colors`

`doc/setter.outline`

Il sorgente **troff** per una scheda di riferimento a cinque colori di **awk**. Per ottenere la versione a colori è richiesta una versione recente di **troff**, come la versione GNU di **troff** (**groff**). Si veda il file **README.card** per istruzioni su come comportarsi se è disponibile solo una versione più vecchia di **troff**.

`doc/gawk.1`

Il sorgente **troff** di una pagina di manuale [*man*] che descrive **gawk**. Questa pagina è distribuita a beneficio degli utenti Unix.

`doc/gawktexi.in`

`doc/sidebar.awk`

Il file sorgente Texinfo di questo libro. Dovrebbe venire elaborato da `doc/sidebar.awk` prima di essere elaborato con **texi2dvi** o **texi2pdf** per produrre un documento stampato, o con **makeinfo** per produrre un file Info o HTML. Il **Makefile** si occupa di questa elaborazione e produce la versione stampabile tramite i comandi **texi2dvi** o **texi2pdf**.

`doc/gawk.texi`

Il file prodotto elaborando **gawktexi.in** tramite **sidebar.awk**.

`doc/gawk.info`

Il file Info generato per questo libro.

`doc/gawkinet.texi`

Il file sorgente Texinfo per *TCP/IP Internetworking with gawk*. Dovrebbe venire elaborato con **T<sub>E</sub>X** (tramite **texi2dvi** o **texi2pdf**) per produrre un documento stampato o con **makeinfo** per produrre un file Info o HTML.

`doc/gawkinet.info`

Il file Info generato per *TCP/IP Internetworking with gawk*.

doc/igawk.1

Il sorgente `troff` per una pagina di manuale relativa al programma `igawk` descritto nella [Sezione 11.3.9 \[Una maniera facile per usare funzioni di libreria\]](#), [pagina 317](#). (Poiché `gawk` prevede ora internamente l'uso della direttiva `@include`, né `igawk` né `igawk.1` sono effettivamente installati.)

doc/Makefile.in

Il file in input usato durante la procedura di configurazione per generare l'effettivo `Makefile` da usare per creare la documentazione.

Makefile.am

\*/Makefile.am

File usati dal software GNU Automake per generare il file `Makefile.in` usato da `Autoconf` e dallo script `configure`.

Makefile.in

aclocal.m4

bisonfix.awk

config.guess

config.in

configure.ac

configure

custom.h

depcomp

install-sh

missing\_d/\*

mkinstalldirs

m4/\* Questi file e sottodirectory sono usati per configurare e compilare `gawk` per vari sistemi Unix. L'uso di molti tra questi file è spiegato nella [Sezione B.2 \[Compilare e installare gawk su sistemi di tipo Unix\]](#), [pagina 483](#). I rimanenti hanno una funzione di supporto per l'infrastruttura.

po/\* La directory `po` contiene la traduzione in varie lingue dei messaggi emessi da `gawk`.

awklib/extract.awk

awklib/Makefile.am

awklib/Makefile.in

awklib/eg/\*

La directory `awklib` contiene una copia di `extract.awk` (si veda la [Sezione 11.3.7 \[Estrarre programmi da un file sorgente Texinfo\]](#), [pagina 312](#)), che può essere usato per estrarre i programmi di esempio dal file sorgente Texinfo di questo libro. Contiene anche un file `Makefile.in`, che `configure` usa per generare un `Makefile`. `Makefile.am` è usato da GNU Automake per creare `Makefile.in`. Le funzioni di libreria descritte nel [Capitolo 10 \[Una libreria di funzioni awk\]](#), [pagina 245](#), sono incluse come file pronti per l'uso nella distribuzione `gawk`. Essi sono installati come parte della procedura di installazione. I rimanenti programmi contenuti in questo libro sono disponibili nelle appropriate sottodirectory di `awklib/eg`.

<code>extension/*</code>	Il codice sorgente, le pagine di manuale, e i file di infrastruttura per gli esempi di estensione incluse con <b>gawk</b> . Si veda la <a href="#">Capitolo 16 [Scrivere estensioni per gawk]</a> , <a href="#">pagina 395</a> , per ulteriori dettagli.
<code>extras/*</code>	Ulteriori file, non-essenziali. Al momento, questa directory contiene alcuni file da eseguire al momento di iniziare una sessione, da installare nella directory <code>/etc/profile.d</code> per essere di aiuto nella gestione delle variabili di ambiente <code>AWKPATH</code> e <code>AWKLIBPATH</code> . Si veda la <a href="#">Sezione B.2.2 [File di inizializzazione della shell]</a> , <a href="#">pagina 484</a> , per ulteriori informazioni.
<code>posix/*</code>	File necessari per compilare <b>gawk</b> su sistemi conformi allo standard POSIX.
<code>pc/*</code>	File necessari per compilare <b>gawk</b> sotto MS-Windows (si veda la <a href="#">Sezione B.3.1 [Installazione su MS-Windows]</a> , <a href="#">pagina 486</a> , per i dettagli).
<code>vms/*</code>	File necessari per compilare <b>gawk</b> sotto Vax/VMS e OpenVMS (si veda la <a href="#">Sezione B.3.2 [Compilare e installare gawk su Vax/VMS e OpenVMS]</a> , <a href="#">pagina 488</a> , per i dettagli).
<code>test/*</code>	Una serie di test per <b>gawk</b> . Si può usare <code>'make check'</code> dalla directory principale di <b>gawk</b> per provare se la serie di test funziona con la versione in uso di <b>gawk</b> . Se <b>gawk</b> supera senza errori <code>'make check'</code> , si può essere sicuri che sia stato installato e configurato correttamente su un dato sistema.

## B.2 Compilare e installare gawk su sistemi di tipo Unix

Normalmente, si può compilare e installare **gawk** immettendo solo un paio di comandi. Comunque, se si ci si trova in un sistema insolito, può essere necessario dover configurare **gawk** per quel dato sistema.

### B.2.1 Compilare gawk per sistemi di tipo Unix

Questi normali passi di installazione dovrebbero essere sufficienti in tutti i moderni sistemi in commercio derivati da Unix, ossia GNU/Linux, sistemi basati su BSD, e l'ambiente Cygwin sotto MS-Windows.

Dopo aver estratto la distribuzione di **gawk**, posizionarsi con `cd` nella directory **gawk-4.1.4**. Come per la maggior parte dei programmi GNU, occorre configurare **gawk** per il sistema in uso, eseguendo il programma **configure**. Questo programma è uno script della shell Bourne, che è stato generato automaticamente usando il comando GNU Autoconf. (Il software Autoconf è descritto in dettaglio in *Autoconf—Generating Automatic Configuration Scripts*, che può essere trovato in rete sul sito [della Free Software Foundation](#).)

Per configurare **gawk** basta eseguire **configure**:

```
sh ./configure
```

Questo produce i file **Makefile** e **config.h** adatti al sistema in uso. Il file **config.h** descrive varie situazioni relative al sistema in uso. È possibile modificare il **Makefile** per cambiare la variabile **CFLAGS**, che controlla le opzioni di riga di comando da passare al compilatore C (come i livelli di ottimizzazione o la richiesta di generare informazioni per il *debug*).

In alternativa, si possono specificare dei valori a piacere per molte delle variabili di `make` sulla riga di comando, come `CC` e `CFLAGS`, quando si chiama il programma `configure`:

```
CC=cc CFLAGS=-g sh ./configure
```

Si veda il file `INSTALL` nella distribuzione di `gawk` per tutti i dettagli.

Dopo aver eseguito `configure` ed eventualmente modificato `Makefile`, va dato il comando:

```
make
```

Poco dopo, si dovrebbe avere a disposizione una versione eseguibile di `gawk`. Questo è tutto! Per verificare se `gawk` funziona correttamente, va dato il comando `'make check'`. Tutti i test dovrebbero terminare con successo. Se questi passi non producono il risultato desiderato, o se qualche test fallisce, controllare i file nella directory `README_d` per determinare se quello che è capitato è un problema noto. Se il problema capitato non è descritto lì, inviare una segnalazione di *bug* (si veda la [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493).

Naturalmente, dopo aver compilato `gawk`, verosimilmente andrà installato. Per fare ciò, occorre eseguire il comando `'make install'`, disponendo delle autorizzazioni necessarie. Come acquisirle varia da sistema a sistema, ma su molti sistemi si può usare il comando `sudo` per ottenerle. Il comando da immettere diventa in questo caso `'sudo make install'`. È probabile che sia necessario fornire una password, ed essere stati messi nella lista degli utenti che possono utilizzare il comando `sudo`.

## B.2.2 File di inizializzazione della shell

La distribuzione contiene i file da usare a inizio sessione `gawk.sh` e `gawk.csh`, che contengono funzioni che possono essere di aiuto nel gestire le variabili di ambiente `AWKPATH` e `AWKLIBPATH`. Su un sistema Fedora GNU/Linux, questi file dovrebbero essere installati nella directory `/etc/profile.d`; su altre piattaforme, la posizione corretta può essere differente.

`gawkpath_default`

Ripristina la variabile d'ambiente `AWKPATH` al suo valore di default.

`gawkpath_prepend`

Aggiunge l'argomento all'inizio della variabile d'ambiente `AWKPATH`.

`gawkpath_append`

Aggiunge l'argomento alla fine della variabile d'ambiente `AWKPATH`.

`gawklibpath_default`

Reimposta la variabile d'ambiente `AWKLIBPATH` al suo valore di default.

`gawklibpath_prepend`

Aggiunge l'argomento all'inizio della variabile d'ambiente `AWKLIBPATH`.

`gawklibpath_append`

Aggiunge l'argomento alla fine della variabile d'ambiente `AWKLIBPATH`.

## B.2.3 Ulteriori opzioni di configurazione

Ci sono parecchie altre opzioni che si possono utilizzare sulla riga di comando di `configure` quando si compila `gawk` a partire dai sorgenti, tra cui:

#### `--disable-extensions`

Richiede di non configurare e generare le estensioni di esempio nella directory `extension`. Questo è utile quando si genera `gawk` per essere eseguito su un'altra piattaforma. L'azione di default è di controllare dinamicamente se le estensioni possono essere configurate e compilate.

#### `--disable-lint`

Disabilita i controlli *lint* all'interno di `gawk`. Le opzioni `--lint` e `--lint-old` (si veda la [Sezione 2.2 \[Opzioni sulla riga di comando\]](#), pagina 33) sono accettate, ma non fanno nulla, e non emettono alcun messaggio di avvertimento. Analogamente, se si imposta la variabile `LINT` (si veda la [Sezione 7.5.1 \[Variabili predefinite modificabili per controllare awk\]](#), pagina 162) questa non ha alcun effetto sul programma `awk` in esecuzione.

Se si specifica l'opzione del compilatore GNU Compiler Collection (GCC) che elimina il codice non eseguito, quest'opzione riduce di quasi 23K byte la dimensione del programma eseguibile `gawk` su sistemi GNU/Linux x86\_64. I risultati su altri sistemi e con altri compilatori sono probabilmente diversi. L'uso di questa opzione può apportare qualche piccolo miglioramento nei tempi di esecuzione di un programma.

**ATTENZIONE:** Se si usa quest'opzione alcuni dei test di funzionalità non avranno successo. Quest'opzione potrà essere rimossa in futuro.

#### `--disable-nls`

Non attiva la traduzione automatica dei messaggi. Ciò normalmente non è consigliabile, ma può apportare qualche lieve miglioramento nei tempi di esecuzione di un programma.

#### `--with-whiny-user-strftime`

Forza l'uso della versione della funzione C `strftime()` inclusa nella distribuzione di `gawk`, per i sistemi in cui la funzione stessa non sia disponibile.

Si usi il comando `./configure --help` per ottenere la lista completa delle opzioni disponibili in `configure`.

## B.2.4 Il processo di configurazione

Questa sezione interessa solo a chi abbia un minimo di familiarità con il linguaggio C e con i sistemi operativi di tipo Unix.

Il codice sorgente di `gawk`, in generale, cerca di aderire, nei limiti del possibile, a degli standard formali. Ciò significa che `gawk` usa routine di libreria che sono specificate nello standard ISO C e nello standard POSIX per le interfacce dei sistemi operativi. Il codice sorgente di `gawk` richiede l'uso di un compilatore ISO C (standard 1990).

Molti sistemi Unix non aderiscono completamente né allo standard ISO né a quello POSIX. La sottodirectory `missing_d` nella distribuzione di `gawk` contiene delle versioni sostitutive per quelle funzioni che più frequentemente risultano essere non disponibili.

Il file `config.h` creato da `configure` contiene definizioni che elencano funzionalità del particolare sistema operativo nel quale si tenta di compilare `gawk`. Le tre cose descritte

da questo file sono: quali file di intestazione sono disponibili, in modo da poterli includere correttamente, quali funzioni (presumibilmente) standard sono realmente disponibili nelle librerie C, e varie informazioni assortite riguardo al sistema operativo corrente. Per esempio, può non esserci l'elemento `st_blksize` nella struttura `stat`. In questo caso, `'HAVE_STRUCT_STAT_ST_BLKSIZE'` è indefinito.

È possibile che il compilatore C del sistema in uso "tragga in inganno" `configure`. Può succedere nel caso in cui non viene restituito un errore se una funzione di libreria non è disponibile. Per superare questo problema, si può modificare il file `custom.h`. Basta usare una direttiva `#ifdef` appropriata per il sistema corrente, e definire, tramite `#define`, tutte le costanti che `configure` avrebbe dovuto definire, ma non è riuscito a farlo, oppure, usando `#undef` annullare le costanti che `configure` ha definito, ma non avrebbe dovuto farlo. Il file `custom.h` è automaticamente incluso dal file `config.h`.

È anche possibile che il programma `configure` generato da Autoconf non funzioni in un dato sistema per una ragione differente. Se c'è un problema, si tenga presente che il file `configure.ac` è quello preso in input da Autoconf. È possibile modificare questo file e generare una nuova versione di `configure` che funzioni sul sistema corrente (si veda la [Sezione B.4 \[Segnalazione di problemi e bug\], pagina 493](#), per informazioni su come segnalare problemi nella configurazione di `gawk`). Lo stesso meccanismo si può usare per inviare aggiornamenti al file `configure.ac` e/o a `custom.h`.

## B.3 Installazione su altri Sistemi Operativi

Questa sezione descrive come installare `gawk` su vari sistemi non-Unix.

### B.3.1 Installazione su MS-Windows

Questa sezione tratta dell'installazione e uso di `gawk` su macchine con architettura Intel che eseguono qualsiasi versione di MS-Windows. In questa sezione, il termine "Windows32" si riferisce a una qualsiasi versione di Microsoft Windows 95/98/ME/NT/2000/XP/Vista/7/8/10.

Si veda anche il file `README_d/README.pc` nella distribuzione.

#### B.3.1.1 Installare una distribuzione predisposta per sistemi MS-Windows

La sola distribuzione binaria predisposta supportata per i sistem MS-Windows è quella messa a disposizione da Eli Zaretskii [progetto "ezwinports"](#). Si parta da lì per installare il comando `gawk` precompilato.

#### B.3.1.2 Compilare `gawk` per sistemi operativi di PC

`gawk` può essere compilato per Windows32, usando MinGW (per Windows32). Il file `README_d/README.pc` nella distribuzione `gawk` contiene ulteriori annotazioni, e il file `pc/Makefile` contiene informazioni importanti sulle opzioni di compilazione.

Per compilare `gawk` per Windows32, occorre copiare i file dalla directory `pc` (*tranne* il file `ChangeLog`) alla directory che contiene il resto dei sorgenti di `gawk`, e quindi chiamare `make`, specificando il nome appropriato di obiettivo come argomento, per generare `gawk`. Il `Makefile` copiato dalla directory `pc` contiene una sezione di configurazione con commenti, e può essere necessario modificarlo perché funzioni con il programma di utilità `make` corrente.

Il `Makefile` contiene un certo numero di alternative, che permettono di generare `gawk` per diverse versioni MS-DOS e Windows32. Se il comando `make` è richiamato senza specificare alcun argomento viene stampata una lista delle alternative disponibili. Per esempio, per generare un codice binario di `gawk` nativo per MS-Windows usando gli strumenti MinGW, scrivere `'make mingw32'`.

### B.3.1.3 Usare gawk su sistemi operativi PC

Sotto MS-Windows, gli ambienti Cygwin e MinGW consentono di usare sia l'operatore `'|&'` che le operazioni su rete TCP/IP (si veda la [Sezione 12.4 \[Usare gawk per la programmazione di rete\]](#), pagina 341).

Le versioni MS-Windows di `gawk` ricercano i file di programma come descritto in [Sezione 2.5.1 \[Ricerca di programmi awk in una lista di directory.\]](#), pagina 42. Comunque, gli elementi della variabile `AWKPATH` sono separati tra di loro da un punto e virgola (anziché da due punti `(:)`). Se `AWKPATH` è non impostata o ha per valore la stringa nulla, il percorso di ricerca di default è `'.;c:/lib/awk;c:/gnu/lib/awk'`.

Sotto MS-Windows, `gawk` (come molti altri programmi di trattamento testi) converte automaticamente la stringa di fine riga `'\r\n'` in `'\n'` leggendo dall'input e `'\n'` in `'\r\n'` scrivendo sull'output. La variabile speciale `BINMODE` (e.c.) permette di controllare come avvengono queste conversioni, ed è interpretata come segue:

- Se `BINMODE` è `"r"` o uno, la modalità binaria è impostata in lettura (cioè, nessuna conversione in lettura).
- Se `BINMODE` è `"w"` o due, la modalità binaria è impostata in scrittura (cioè, nessuna conversione in scrittura).
- Se `BINMODE` è `"rw"` o `"wr"` o tre, la modalità binaria è impostata sia in lettura che in scrittura.
- `BINMODE=stringa-non-nulla` equivale a specificare `'BINMODE=3'` (cioè, nessuna conversione in lettura e scrittura). Tuttavia, `gawk` emette un messaggio di avviso se la stringa non è `"rw"` o `"wr"`.

La modalità di trattamento dello standard input e standard output sono impostate una volta sola (dopo aver letto la riga di comando, ma prima di iniziare a elaborare qualsiasi programma `awk`). L'impostazione di `BINMODE` per standard input o standard output va fatta usando un'appropriata opzione `'-v BINMODE=N'` sulla riga di comando. `BINMODE` è impostato nel momento in cui un file o *pipe* è aperto e non può essere cambiato in corso di elaborazione.

Il nome `BINMODE` è stato scelto in analogia con `mawk` (si veda la [Sezione B.5 \[Altre implementazioni di awk liberamente disponibili\]](#), pagina 494). `mawk` e `gawk` gestiscono `BINMODE` in maniera simile; tuttavia, `mawk` prevede un'opzione `'-W BINMODE=N'` e una variabile d'ambiente che può impostare `BINMODE`, `RS`, e `ORS`. I file `binmode[1-3].awk` (nella directory `gnu/lib/awk` in alcune delle distribuzioni binarie già predisposte) sono stati inclusi per rendere disponibile l'opzione di `mawk` `'-W BINMODE=N'`. Questi possono essere modificati o ignorati; in particolare, quale sia l'impostazione di `RS` che dà meno "sorprese" rimane una questione aperta. `mawk` usa `'RS = "\r\n"'` se si imposta la modalità binaria in lettura, il che è appropriato per file che abbiano i caratteri di fine riga in stile MS-DOS.

Per chiarire, gli esempi seguenti impostano la modalità binaria in scrittura per lo standard output e altri file, e impostano `ORS` in modo da ottenere la fine riga "normale" in stile MS-DOS:

```
gawk -v BINMODE=2 -v ORS="\r\n" ...
```

o:

```
gawk -v BINMODE=w -f binmode2.awk ...
```

Questi comandi danno lo stesso risultato dell'opzione '-W BINMODE=2' in `mawk`. Quanto segue modifica il separatore di record a "\r\n" e imposta la modalità binaria in lettura, senza modificare le letture da standard input:

```
gawk -v RS="\r\n" -e "BEGIN { BINMODE = 1 }" ...
```

o:

```
gawk -f binmode1.awk ...
```

Usando i caratteri di protezione appropriati, nel primo esempio l'impostazione di `RS` può essere spostata in una regola `BEGIN`.

### B.3.1.4 Usare gawk in ambiente Cygwin

`gawk` può essere compilato e usato “così com'è” sotto MS-Windows se si opera all'interno dell'ambiente **Cygwin**. Questo ambiente consente un'eccellente simulazione di GNU/Linux, con l'uso di Bash, GCC, GNU Make, e altri programmi GNU. La compilazione e l'installazione per Cygwin è la stessa usata nei sistemi di tipo Unix:

```
tar -xvpzf gawk-4.1.4.tar.gz
cd gawk-4.1.4
./configure
make && make check
```

In confronto a un sistema GNU/Linux sulla stessa macchina, l'esecuzione del passo di 'configure' sotto Cygwin richiede molto più tempo. Tuttavia si conclude regolarmente, e poi 'make' funziona come ci si aspetta.

### B.3.1.5 Usare gawk in ambiente MSYS

Nell'ambiente MSYS sotto MS-Windows, `gawk` automaticamente usa la modalità binaria per leggere e scrivere file. Non è quindi necessario usare la variabile `BINMODE`.

Questo può causare problemi con altri componenti di tipo Unix che sono stati resi disponibili in MS-Windows, che si aspettano che `gawk` faccia automaticamente la conversione di "\r\n", mentre così non è.

## B.3.2 Compilare e installare gawk su Vax/VMS e OpenVMS

Questa sottosezione descrive come compilare e installare `gawk` sotto VMS. Il termine classico “VMS” è usato qui anche per designare OpenVMS.

### B.3.2.1 Compilare gawk su VMS

Per compilare `gawk` sotto VMS, esiste una procedura di comandi DCL che esegue tutti i comandi CC e LINK necessari. C'è anche un `Makefile` da usare con i programmi di utilità MMS e MMK. A partire della directory che contiene i file sorgente, si usi:

```
$ @[.vms]vmsbuild.com
```

o:

```
$ MMS/DESCRIPTION=[.vms]descrip.mms gawk
```

o:

```
$ MMK/DESCRIPTION=[.vms]descrip.mms gawk
```

Il comando MMK è un quasi-clone, a codice aperto e gratuito, di MMS, che gestisce in maniera migliore i volumi ODS-5 con nomi-file a caratteri maiuscoli e minuscoli. MMK è disponibile da <https://github.com/endlesssoftware/mmk>.

Avendo a che fare con volumi ODS-5 e con l'analisi sintattica estesa abilitata, il nome del parametro che specifica l'obiettivo può dover essere scritto digitando esattamente le lettere maiuscole e minuscole.

**gawk** è stato testato sotto VAX/VMS 7.3 e Alpha/VMS 7.3-1 usando il compilatore Compaq C V6.4, e sotto Alpha/VMS 7.3, Alpha/VMS 7.3-2, e IA64/VMS 8.3. Le compilazioni più recenti hanno usato il compilatore HP C V7.3 su Alpha VMS 8.3 e su VMS 8.4, sia Alpha che IA64, hanno usato il compilatore HP C 7.3.<sup>1</sup>

Si veda la [Sezione B.3.2.5 \[Il progetto VMS GNV\]](#), pagina 492, per informazioni su come compilare **gawk** come un kit PCSI compatibile con il prodotto GNV.

### B.3.2.2 Compilare estensioni dinamiche di gawk in VMS

Le estensioni che sono state rese disponibile su VMS possono essere costruite dando uno dei comandi seguenti:

```
$ MMS/DESCRIPTION=[.vms]descrip.mms extensions
```

o:

```
$ MMK/DESCRIPTION=[.vms]descrip.mms extensions
```

**gawk** usa AWKLIBPATH come una variabile d'ambiente oppure come un nome logico per trovare le estensioni dinamiche.

Le estensioni dinamiche devono essere compilate con le stesse opzioni del compilatore usate per compilare **gawk** riguardanti numeri in virgola mobile, dimensione dei puntatori e trattamento dei nomi simbolici. I computer con architettura Alpha e Itanium dovrebbero usare i numeri in virgola mobile col formato IEEE. La dimensione dei puntatori è 32 bit, e il trattamento dei nomi simbolici dovrebbe richiedere il rispetto esatto di maiuscole/minuscole, con le abbreviazioni CRC per simboli più lunghi di 32 bit.

Per Alpha e Itanium:

```
/name=(as_is,short)
/float=ieee/ieee_mode=denorm_results
```

Per VAX:

```
/name=(as_is,short)
```

Le macro da usare al momento della compilazione devono essere definite prima di includere il primo file di intestazione proveniente da VMS, come segue:

```
#if (__CRTL_VER >= 70200000) && !defined (__VAX)
#define _LARGEFILE 1
#endif

#ifdef __VAX
```

---

<sup>1</sup> L'architettura IA64 è anche nota come "Itanium".

```

#ifdef __CRTL_VER
#if __CRTL_VER >= 80200000
#define _USE_STD_STAT 1
#endif
#endif
#endif

```

Se si scrivono delle estensioni utente da eseguire su VMS, vanno fornite anche queste definizioni. Il file `config.h` creato quando si compila `gawk` su VMS lo fa già; se invece si usa qualche altro file simile, occorre ricordarsi di includerlo prima di qualsiasi file di intestazione proveniente da VMS.

### B.3.2.3 Installare gawk su VMS

Per usare `gawk`, tutto ciò che serve è un comando “esterno”, che è un simbolo DCL il cui valore inizia col segno del dollaro. Per esempio:

```
$ GAWK := $disk1:[gnubin]gawk
```

Si sostituisca la posizione corrente di `gawk.exe` a `'$disk1:[gnubin]'`. Il simbolo dovrebbe essere posto nel file `login.com` di ogni utente che desideri eseguire `gawk`, in modo che sia definito ogni volta che l'utente inizia una sessione. Alternativamente, il simbolo può essere messo nella procedura di sistema `sylogin.com`, in modo da permettere a tutti gli utenti di eseguire `gawk`.

Se `gawk` è stato installato da un kit PCSI nell'albero di directory `GNV$GNU:`, il programma avrà come nome `GNV$GNU:[bin]gnv$gawk.exe`, e il file di aiuto sarà chiamato `GNV$GNU:[vms_help]gawk.hlp`.

Il kit PCSI installa anche un file `GNV$GNU:[vms_bin]gawk_verb.cld` che può essere usato per aggiungere `gawk` e `awk` alla lista dei comandi DCL.

Per farlo solo nella sessione corrente si può usare:

```
$ set command gnv$gnu:[vms_bin]gawk_verb.cld
```

Oppure il sistemista VMS può usare `GNV$GNU:[vms_bin]gawk_verb.cld` per aggiungere `gawk` e `awk` alla tabella 'DCLTABLES' valida per tutto il sistema.

La sintassi DCL è documentata nel file `gawk.hlp`.

In alternativa, l'elemento `gawk.hlp` può essere caricato in una libreria di aiuto VMS:

```
$ LIBRARY/HELP sys$help:helplib [.vms]gawk.hlp
```

(Una libreria specifica dell'installazione potrebbe venir usata invece della libreria standard VMS library 'HELPLIB'.) Dopo aver installato il testo di aiuto, il comando:

```
$ HELP GAWK
```

fornisce informazioni sia sull'implementazione di `gawk` sia sul linguaggio di programmazione `awk`.

Il nome logico 'AWK\_LIBRARY' può designare una posizione di default per i file di programma `awk`. Riguardo all'opzione `-f`, se il nome-file specificato non contiene un dispositivo o un percorso di directory, `gawk` cerca dapprima nella directory corrente, poi nella directory specificata dalla traduzione di 'AWK\_LIBRARY' se il file non è stato trovato. Se, dopo aver cercato in entrambe queste directory, il file non è ancora stato trovato, `gawk` appone il suffisso '.awk' al nome-file e ritenta la ricerca del file. Se 'AWK\_LIBRARY' non è definita, si usa per essa il valore di default 'SYS\$LIBRARY:'.

### B.3.2.4 Eseguire gawk su VMS

L'elaborazione della riga di comando e le convenzioni per proteggere i caratteri sono significativamente differenti in VMS, e quindi gli esempi presenti in questo libro o provenienti da altre fonti necessitano piccole modifiche. Le modifiche, tuttavia, *sono* veramente piccole, e tutti i programmi **awk** dovrebbero funzionare correttamente.

Ecco un paio di semplici test:

```
$ gawk -- "BEGIN {print \"Hello, World!\"}"
$ gawk -"W" version
! ma anche -"W version" o "-W version"
```

Si noti che il testo con caratteri maiuscoli e misti maiuscoli/minuscoli dev'essere incluso tra doppi apici.

La versione VMS di **gawk** comprende un'interfaccia in stile DCL, oltre a quella originale, di tipo shell (si veda il file di aiuto per ulteriori dettagli). Un effetto indesiderato della duplice analisi della riga di comando è che se c'è solo un unico parametro (come nel programma con le righe contenenti doppi apici), il comando diviene ambiguo. Per evitare questo inconveniente, il flag, normalmente non necessario, `--` è necessario per forzare un esame dei parametri in stile Unix, piuttosto che nella modalità DCL. Se qualsiasi altra opzione preceduta dal segno `-` (o più parametri, per esempio, più file-dati in input) è presente, non c'è ambiguità, e l'opzione `--` può essere omessa.

Il valore di `exit` è un valore in stile Unix e viene trasformato in una condizione VMS all'uscita del programma.

I bit di severità di VMS saranno impostati a partire dal valore dell'istruzione `exit`. Un errore grave è indicato da 1, e VMS imposta la condizione `ERROR`. Un errore fatale è indicato da 2, e VMS imposta la condizione `FATAL`. Ogni altro valore imposta la condizione `SUCCESS`. Il valore d'uscita è codificato per aderire agli standard di codifica VMS e avrà un `C_FACILITY_NO` di `0x350000` con il codice costante `0xA000` aggiunto al numero spostato a sinistra di 3 bit per far posto al codice di severità.

Per estrarre il codice reale di ritorno dell'istruzione `exit` di **gawk** dalla condizione impostata da VMS, si usi:

```
unix_status = (vms_status .and. %x7f8) / 8
```

Un programma C che usa `exec()` per chiamare **gawk** riceverà il valore originale della exit in stile Unix.

Precedenti versioni di **gawk** per VMS consideravano un codice di ritorno a Unix di 0 come 1, un errore come 2, un errore fatale come 4, e tutti gli altri valori erano restituiti immutati. Questa era una violazione rispetto alle specifiche di codifica delle condizioni di uscita di VMS.

L'aritmetica in virgola mobile VAX/VMS usa un arrotondamento statistico. Si veda la [Sezione 10.2.3 \[Arrotondamento di numeri\], pagina 250](#).

VMS restituisce data e ora in formato GMT, a meno che non siano stati impostati i nomi logici `SYS$TIMEZONE_RULE` o `TZ`. Precedenti versioni di VMS, come VAX/VMS 7.3, non impostano questi nomi logici.

Il percorso di ricerca di default, nella ricerca dei file di programma per **awk** specificati dall'opzione `-f`, è `"SYS$DISK: [], AWK_LIBRARY: "`. Il nome logico `AWKPATH` può essere usato per sostituire questo di default. Il formato di `AWKPATH` è una lista di directory, separate da

virgola. Nel definirla, il valore dovrebbe essere incluso tra doppi apici, in modo che consenta una sola traduzione, e non una lista di ricerca multistraduzione RMS.

Questa restrizione vale anche se si esegue **gawk** sotto GNV, in quanto la ridirezione è sempre verso un comando DCL.

Se si ridirigono dati verso un comando o un programma di utilità VMS, l'implementazione corrente richiede la creazione di un comando VMS esterno che esegua un file di comandi, prima di invocare **gawk**. (Questa restrizione potrebbe essere rimossa in una futura versione di **gawk** per VMS.)

Senza un tale file di comandi, i dati in input saranno presenti anche in output, prima dei dati di output veri e propri.

Ciò consente la simulazione di comandi POSIX non disponibili in VMS o l'uso di programmi di utilità GNV.

L'esempio seguente mostra come ridirigere dati da **gawk** verso il comando VMS **sort**.

```
$ sort = "@device:[dir]vms_gawk_sort.com"
```

Il file di comandi deve avere il formato dell'esempio seguente.

La prima riga serve a evitare che i dati in input siano presenti anche nell'output. Dev'essere nel formato mostrato nell'esempio.

La riga seguente crea un comando esterno che prevale sul comando esterno superiore, che serve a prevenire una ricorsione infinita di file di comandi.

Il penultimo comando ridirige **sys\$input** su **sys\$command**, per poter ottenere i dati che sono ridiretti verso il comando.

L'ultima riga esegue il comando vero e proprio. Dev'essere l'ultimo comando, perché i dati ridiretti da **gawk** saranno letti quando il file di comandi finisce di essere eseguito.

```
$!'f$verify(0,0)'
$ sort := sort
$ define/user sys$input sys$command:
$ sort sys$input: sys$output:
```

### B.3.2.5 Il progetto VMS GNV

Il pacchetto VMS GNV fornisce un ambiente di sviluppo simile a POSIX tramite una collezione di strumenti *open source*. Il **gawk** presente nel pacchetto base GNV è una vecchia versione. Attualmente, il progetto GNV è in fase di riorganizzazione, con l'obiettivo di offrire pacchetti PCSI separati per ogni componente. Si veda <https://sourceforge.net/p/gnv/wiki/InstallingGNVPackages/>.

La procedura normale per compilare **gawk** produce un programma adatto a essere usato con GNV.

Il file **vms/gawk\_build\_steps.txt** nella distribuzione documenta la procedura per compilare un pacchetto PCSI compatibile con GNV.

### B.3.2.6 Vecchia versione di **gawk** su sistemi VMS

Alcune versioni di VMS includono una vecchia versione di **gawk**. Per utilizzarla, occorre definire un simbolo, come segue:

```
$ gawk := $sys$common:[syshlp.examples.tcpip.snmp]gawk.exe
```

La versione appare essere la versione 2.15.6, che è molto vecchia. Si raccomanda di compilare e usare la versione corrente.

## B.4 Segnalazione di problemi e bug

*Non c'è niente di più pericoloso di un archeologo annoiato.*

—Douglas Adams, *Guida galattica per autostoppisti*

Se si incontrano problemi con **gawk** o se si ritiene di aver trovato un bug, si raccomanda di segnalarlo agli sviluppatori; non c'è un impegno preciso a intervenire, ma c'è una buona possibilità che ci si sforzi di risolverlo.

### B.4.1 Segnalare Bug

Prima di segnalare un bug, occorre assicurarsi che sia davvero un bug. La documentazione va letta attentamente, per controllare se dice che è possibile fare quel che si sta tentando di fare. Se non è chiaro se sia possibile fare quella particolare cosa o no, occorre segnalarlo; in questo caso si tratta di un bug nella documentazione!

Prima di segnalare un bug o di tentare di risolverlo personalmente, si tenti di isolarlo preparando un programma **awk** il più piccolo possibile, con un file-dati in input che possa riprodurre il problema. Dopo averlo fatto, si spedisca il programma e il file-dati, insieme a informazioni sul tipo di sistema Unix in uso, il compilatore usato per compilare **gawk**, e i risultati esatti che **gawk** ha prodotto. Inoltre andrebbe specificato cosa ci si aspettava che il programma facesse; questo è di aiuto per decidere se il problema è un problema di documentazione.

È importante includere il numero di versione di **gawk** in uso. Questa informazione si può ottenere con il comando `'gawk --version'`.

Una volta pronta la descrizione precisa del problema, si spedisca un messaggio di posta elettronica a [bug-gawk@gnu.org](mailto:bug-gawk@gnu.org).

I manutentori di **gawk** sono i destinatari, e riceveranno la segnalazione di errore. Sebbene sia possibile spedire messaggi direttamente ai manutentori, è preferibile usare l'indirizzo sopra fornito perché quella mailing list rimane in archivio presso il Progetto GNU. *Tutti i messaggi devono essere in inglese. È questo il solo linguaggio che tutti i manutentori conoscono.* Inoltre, occorre accertarsi di spedire tutti i messaggi in formato *testo*, e non (o non soltanto) in formato HTML.

**NOTA:** Molte distribuzioni di GNU/Linux e i vari sistemi operativi basati su BSD hanno un loro proprio canale per segnalare i bug. Se si segnala un bug usando il canale della distribuzione, una copia del messaggio andrebbe inviata anche a [bug-gawk@gnu.org](mailto:bug-gawk@gnu.org).

Questo per due ragioni. La prima è che, sebbene alcune distribuzioni inoltrino i messaggi sui problemi “verso l'alto” alla mailing list GNU, molte non lo fanno, e quindi c'è una buona probabilità che i manutentori di **gawk** non vedano affatto il messaggio relativo al bug! La seconda ragione è che la posta diretta alla mailing list GNU è archiviata, e il poter disporre di ogni cosa all'interno del progetto GNU consente di avere a disposizione tutte le informazioni rilevanti senza dover dipendere da altre organizzazioni.

Suggerimenti non correlati a bug sono pure sempre benvenuti. Se si hanno domande riguardo a qualcosa di non chiaro nella documentazione o a proposito di funzionalità oscure, si scriva alla mailing list dei bug; si proverà a essere di aiuto nei limiti del possibile.

### B.4.2 Non segnalare bug a USENET!

Ho iniziato a ignorare Usenet un paio di anni fa, e non me ne sono mai pentito. È come quando si parla di sport alla radio—ci si sente più intelligenti per aver lasciato perdere.

—Chet Ramey

Siete pregati di *non* provare a notificare bug di **gawk** scrivendo al gruppo di discussione Usenet/Internet `comp.lang.awk`. Sebbene alcuni degli sviluppatori di **gawk** leggano talora i messaggi di questo gruppo di discussione, il manutentore principale di **gawk** non lo fa più. Quindi è praticamente certo che un messaggio inviato là *non* sia da lui letto. La procedura qui descritta è la sola maniera ufficialmente riconosciuta per notificare problemi. Davvero!

### B.4.3 Notificare problemi per versioni non-Unix

Se si riscontrano bug in una delle versioni non-Unix di **gawk**, una copia del messaggio inviato alla mailing list dei bug andrebbe spedita alla persona che si occupa di quella versione. I manutentori sono elencati nella lista seguente, come pure nel file **README** nella distribuzione **gawk**. Le informazioni nel file **README** dovrebbero essere considerate come le più aggiornate, se risultano in conflitto con questo libro.

Le persone che si occupano delle varie versioni di **gawk** sono:

Unix e sistemi POSIX	Arnold Robbins, <a href="mailto:arnold@skeeve.com">arnold@skeeve.com</a>
MS-Windows con MinGW	Eli Zaretskii, <a href="mailto:eliz@gnu.org">eliz@gnu.org</a>
OS/2	Andreas Buening, <a href="mailto:andreas.buening@nexgo.de">andreas.buening@nexgo.de</a>
VMS	John Malmberg, <a href="mailto:wb8tyw@qsl.net">wb8tyw@qsl.net</a>
z/OS (OS/390)	Daniel Richard G. <a href="mailto:skunk@iSKUNK.ORG">skunk@iSKUNK.ORG</a> Dave Pitts (Maintainer Emeritus), <a href="mailto:dpitts@cozix.com">dpitts@cozix.com</a>

Se il problema riscontrato è riproducibile anche sotto Unix, si dovrebbe spedire una copia del messaggio anche alla mailing list [bug-gawk@gnu.org](mailto:bug-gawk@gnu.org).

La versione generata usando gli strumenti DJGPP non è più supportata; il codice relativo resterà nella distribuzione ancora per qualche tempo, nel caso che qualche volontario desideri prenderla in carico. Se questo non dovesse succedere, la parte di codice relativa questa versione sarà rimossa dalla distribuzione.

## B.5 Altre implementazioni di awk liberamente disponibili

*È piuttosto divertente mettere commenti simili nel vostro codice awk:*

```
// Funzionano i commenti in stile C++? Risposta: sì! certo
```

—Michael Brennan

Ci sono alcune altre implementazioni di **awk** disponibili gratuitamente. Questa sezione descrive in breve dove è possibile trovarle:

**Unix awk** Brian Kernighan, uno degli sviluppatori originali di Unix **awk**, ha reso disponibile liberamente la sua implementazione di **awk**. Si può scaricare questa versione dalla [sua pagina principale](#). È disponibile in parecchi formati compressi:

Archivio Shell

<http://www.cs.princeton.edu/~bwk/btl.mirror/awk.shar>

File **tar** compresso

<http://www.cs.princeton.edu/~bwk/btl.mirror/awk.tar.gz>

File Zip

<http://www.cs.princeton.edu/~bwk/btl.mirror/awk.zip>

È anche disponibile in GitHub:

```
git clone git://github.com/onetrueawk/awk bawk
```

Questo comando crea una copia del deposito **Git** in una directory chiamata **bawk**. Se si omette questo argomento della riga di comando **git**, la copia del deposito è creata nella directory di nome **awk**.

Questa versione richiede un compilatore ISO C (standard 1990); il compilatore C contenuto in GCC (la collezione di compilatori GNU) è più che sufficiente.

Si veda la [Sezione A.7 \[Sommario Estensioni Comuni\]](#), pagina 473, per una lista di estensioni in questo **awk** che non sono in POSIX **awk**.

Incidentalmente, Dan Bornstein ha creato un deposito Git che contiene tutte le versioni di **BWK awk** che è riuscito a trovare. È disponibile in [git://github.com/danfuzz/one-true-awk](https://github.com/danfuzz/one-true-awk).

**mawk** Michael Brennan ha scritto un'implementazione indipendente di **awk**, di nome **mawk**. È disponibile sotto la licenza GPL (si veda la [Licenza Pubblica Generale GNU \(GPL\)](#)), proprio come **gawk**.

Il sito di distribuzione originale di **mawk** non contiene più il codice sorgente. Una copia è disponibile in <http://www.skeeve.com/gawk/mawk1.3.3.tar.gz>.

Dal 2009 è Thomas Dickey a occuparsi della manutenzione di **mawk**. Le informazioni di base sono disponibili nella [pagine web del progetto](#). Il puntatore URL da cui scaricare è <http://invisible-island.net/datafiles/release/mawk.tar.gz>.

Una volta scaricato, per scompattare questo file può essere usato **gunzip**. L'installazione è simile a quella di **gawk** (si veda la [Sezione B.2 \[Compilare e installare gawk su sistemi di tipo Unix\]](#), pagina 483).

Si veda la [Sezione A.7 \[Sommario Estensioni Comuni\]](#), pagina 473, per una lista di estensioni in **mawk** che non sono in POSIX **awk**.

**awka** Scritto da Andrew Sumner, **awka** traduce i programmi **awk** in C, li compila, e prepara il codice eseguibile usando una libreria di funzioni che implementano le funzionalità di base di **awk**. Comprende anche un certo numero di estensioni.

Il traduttore di **awk** è rilasciato sotto la licenza GPL, e la relativa libreria sotto la licenza LGPL.

Per ottenere **awka**, si visiti il sito <http://sourceforge.net/projects/awka>.

Il progetto sembra essere stato congelato; non ci sono state modifiche nel codice sorgente dal 2001 circa.

**pawk** Nelson H.F. Beebe all'Università dello Utah ha modificato BWK **awk** per fornire informazioni di temporizzazione e profilatura. Questo è differente dall'usare **gawk** con l'opzione `--profile` (si veda la [Sezione 12.5 \[Profilare i propri programmi awk\]](#), [pagina 343](#)) nel senso che fornisce un profilo basato sul consumo di CPU, non sul numero di esecuzioni di una data riga di codice. Sia può trovare sia in <ftp://ftp.math.utah.edu/pub/pawk/pawk-20030606.tar.gz> che in <http://www.math.utah.edu/pub/pawk/pawk-20030606.tar.gz>.

#### BusyBox **awk**

BusyBox è un programma distribuito con licenza GPL che fornisce versioni ridotte di parecchie piccole applicazioni, all'interno di un singolo modulo eseguibile. È stato ideato per sistemi integrati. Include un'implementazione completa di POSIX **awk**. Quando lo si compila occorre prestare attenzione a non eseguire `'make install'`, perché in questo modo si andrebbero a sostituire copie di altre applicazioni nella directory `/usr/local/bin` del sistema corrente. Per ulteriori informazioni, si veda [la pagina principale del progetto](#).

#### POSIX **awk** per OpenSolaris

Le versioni di **awk** in `/usr/xpg4/bin` e `/usr/xpg6/bin` su Solaris sono *grosso modo* conformi allo standard POSIX. Sono basate sul comando **awk** preparato per i PC dalla ditta Mortice Kern. È stato possibile compilare e far funzionare questo codice sotto GNU/Linux dopo 1–2 ore di lavoro. Rendere questo codice più generalmente portabile (usando gli strumenti GNU Autoconf e/o Automake) richiederebbe ulteriore lavoro, che non è stato fin qui compiuto, almeno per quel che risulta a chi scrive.

Il codice sorgente era un tempo disponibile dal sito web OpenSolaris. Tuttavia, il progetto è terminato, e il sito web chiuso. Fortunatamente, il progetto **Illumos** mette a disposizione questa implementazione. Si possono vedere i singoli file in [https://github.com/joyent/illumos-joyent/blob/master/usr/src/cmd/awk\\_xpg4](https://github.com/joyent/illumos-joyent/blob/master/usr/src/cmd/awk_xpg4).

**jawk** Questo è un interprete per **awk** scritto in Java. Dichiara di essere un interprete completo, anche se, poiché usa funzionalità di Java per l'I/O e per la ricerca di *regex*, il linguaggio che supporta è differente da **awk** POSIX. Ulteriori informazioni sono disponibili sulla [pagina principale del progetto](#).

**Libmawk** Questo è un interprete **awk** incorporabile, derivato da **mawk**. Per ulteriori informazioni, si veda <http://repo.hu/projects/libmawk/>.

**pawk** Questo è un modulo Python che intende introdurre funzionalità di tipo **awk** in Python. Si veda <https://github.com/alecthomas/pawk> per ulteriori informazioni. (Questo programma non è correlato con la versione modificata da Nelson Beebe di BWK **awk**, descritta prima.)

**QSE awk** Questo è un interprete di **awk** incorporabile. Per ulteriori informazioni, si veda <http://code.google.com/p/qse/> e <http://awk.info/?tools/qse>.

**QTawk** Questa è un'implementazione indipendente di **awk** distribuita con la licenza GPL. Ha un gran numero di estensioni rispetto ad **awk** standard, e può non essere sintatticamente compatibile al 100% con esso. Si veda <http://www.quiktrim.org/QTawk.html> per ulteriori informazioni, compreso il manuale. Il puntatore per scaricare QuikTrim non punta all'ultima versione: si veda <http://www.quiktrim.org/#AdditionalResources> per un puntatore alla versione corrente.

Il progetto sembra essere fermo; non ci sono nuove versioni del codice a partire dal 2014 circa.

Altre versioni

Si veda anche [in inglese] la sezione “Versions and implementations” della voce di [Wikipedia](#) su **awk** per informazioni su ulteriori versioni.

## B.6 Sommario

- La distribuzione di **gawk** è disponibile dal sito principale di distribuzione del Progetto GNU <ftp.gnu.org>. La maniera canonica per scaricarlo e installarlo è:  

```
wget http://ftp.gnu.org/gnu/gawk/gawk-4.1.4.tar.gz
tar -xvpzf gawk-4.1.4.tar.gz
cd gawk-4.1.4
./configure && make && make check
```
- **gawk** può essere installato anche su sistemi non-POSIX. I sistemi correntemente supportati sono MS-Windows, usando MSYS, MinGW, e Cygwin, e sia Vax/VMS che OpenVMS. Le istruzioni per ognuno di questi sistemi sono incluse in questa appendice.
- Le segnalazioni di errori (bug) dovrebbero essere spedite tramite email a [bug-gawk@gnu.org](mailto:bug-gawk@gnu.org). Le segnalazioni di errore dovrebbero essere scritte in inglese e dovrebbero specificare la versione di **gawk** in uso, come è stata compilata, un breve programma e un file-dati che permettono di riprodurre il problema.
- Ci sono alcune altre implementazioni di **awk** disponibili gratuitamente. Molte rispettano lo standard POSIX; altre un po' meno.



## Appendice C Note di implementazione

Quest'appendice contiene informazioni che interessano soprattutto le persone che aggiornano e mantengono **gawk**. L'intero contenuto si applica specificatamente a **gawk** e non ad altre implementazioni.

### C.1 Compatibilità all'indietro e debug

Si veda la [Sezione A.5 \[Estensioni di \*\*gawk\*\* non in POSIX \*\*awk\*\*\]](#), pagina 464, per un compendio delle estensioni GNU per il linguaggio e il programma **awk**. Tutte queste funzionalità possono essere inibite invocando **gawk** con l'opzione `--traditional` o con l'opzione `--posix`.

Se **gawk** è stato compilato per il debug con `'-DDEBUG'`, è possibile specificare un'ulteriore opzione sulla riga di comando:

```
-Y
--parsedebug
    Stampa l'informazione contenuta nella pila di analisi, durante la fase di analisi
    iniziale del programma.
```

Quest'opzione è utile solo a chi sviluppa **gawk** e non all'utente occasionale. È probabile che non sia neppure disponibile nella versione di **gawk** che si sta usando, perché rallenta l'esecuzione del programma.

### C.2 Fare aggiunte a **gawk**

Se si desidera migliorare **gawk** in maniera significativa, c'è la massima libertà di farlo. È questo lo scopo del software libero; il codice sorgente è disponibile, ed è possibile modificarlo a piacere (si veda la [Licenza Pubblica Generale GNU \(GPL\)](#)], pagina 529).

Questa sezione tratta di come è possibile modificare **gawk**, ed espone alcune considerazioni che si dovrebbero tenere presenti.

#### C.2.1 Accedere al deposito dei sorgenti Git di **gawk**

Poiché **gawk** è Software Libero, il codice sorgente è sempre disponibile. La [Sezione B.1 \[La distribuzione di \*\*gawk\*\*\]](#), pagina 479, descrive come scaricare e installare le versioni ufficiali rilasciate di **gawk**.

Peraltro, se si intende modificare **gawk** e mettere a disposizione le modifiche, è preferibile lavorare sulla versione in via di sviluppo. Per far ciò è necessario accedere al deposito del codice sorgente di **gawk**. Il codice è mantenuto usando il [sistema distribuito di controllo delle versioni Git](#). Sarà necessario installarlo se non è già presente nel sistema. Quando **git** è disponibile, va usato il comando:

```
git clone git://git.savannah.gnu.org/gawk.git
```

Questo comando scarica in locale una copia esatta del deposito dei sorgenti di **gawk**. Se si sta usando un *firewall* che non consente l'uso del protocollo nativo di Git, è possibile accedere ugualmente al deposito usando il comando:

```
git clone http://git.savannah.gnu.org/r/gawk.git
```

Una volta modificato il sorgente, è possibile usare `'git diff'` per produrre una *patch*, e spedirla al manutentore di **gawk**; si veda [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493, per come farlo.

In passato era disponibile un'interfaccia Git-CVS utilizzabile da persone che non avevano a disposizione Git. Purtroppo, quest'interfaccia non funziona più, ma si potrebbe avere maggior fortuna usando un sistema di controllo versioni più moderno, come Bazaar, che è dotato di un'estensione Git per lavorare con depositi di sorgenti Git.

## C.2.2 Aggiungere nuove funzionalità

Ognuno è libero di aggiungere tutte le nuove funzionalità che vuole a **gawk**. Comunque, se si desidera che tali modifiche siano incorporate nella distribuzione di **gawk**, ci sono parecchi passi da fare per rendere possibile la loro inclusione:

1. Prima di inserire la nuova funzionalità all'interno di **gawk**, prendere in considerazione la possibilità di scriverla sotto forma di estensione (si veda la [Capitolo 16 \[Scrivere estensioni per gawk\]](#), pagina 395). Se ciò non è possibile, continuare con i passi rimanenti descritti in questa lista.
2. Essere disposti a firmare un documento liberatorio appropriato. Se l'FSF deve poter distribuire le modifiche, queste vanno dichiarate di pubblico dominio, tramite la firma di un documento apposito, oppure attribuendo il copyright delle modifiche all'FSF. Entrambe queste azioni sono semplici da fare, e *molte* persone già l'hanno fatto. Se ci sono domande da fare, mettersi in contatto con me (si veda la [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493), oppure [assign@gnu.org](mailto:assign@gnu.org).
3. Utilizzare l'ultima versione. È molto più semplice per me integrare modifiche se sono basate sull'ultima versione disponibile di **gawk** o, meglio ancora, sull'ultimo codice sorgente presente nel deposito Git. Se la versione di **gawk** è molto vecchia, potrei non essere affatto in grado di integrare le modifiche. (Si veda la [Sezione B.1.1 \[Ottenere la distribuzione di gawk\]](#), pagina 479, per informazioni su come ottenere l'ultima versione di **gawk**.)
4. Seguire gli *Standard di codifica GNU*. Questo documento descrive come dovrebbe essere scritto il software GNU. Se non lo si è letto, è bene farlo, preferibilmente *prima* di iniziare a modificare **gawk**. (Gli *Standard di codifica GNU* sono disponibili nel sito web del [Progetto GNU](#). Sono disponibili anche versioni in formato Texinfo, Info, e DVI.)
5. Usare lo stile di codifica **gawk**. Il codice sorgente in C di **gawk** segue le istruzioni dello *Standard di codifica GNU*, con qualche piccola eccezione. Il codice è formattato usando lo stile tradizionale “K&R”, in particolare per ciò che riguarda il posizionamento delle parentesi graffe e l'uso del carattere TAB. In breve, le regole di codifica per **gawk** sono le seguenti:
  - Usare intestazioni di funzione (prototipi) in stile ANSI/ISO quando si definiscono delle funzioni.
  - Mettere il nome della funzione all'inizio della riga in cui la si sta definendo.
  - Usare `#elif` invece di nidificare istruzioni `#if` all'interno di un'istruzione `#else`.
  - Mettere il tipo di codice di ritorno della funzione, anche se è `int`, sulla riga immediatamente sopra quella che contiene il nome e gli argomenti della funzione.
  - Mettere degli spazi attorno alle parentesi usate nelle strutture di controllo (`if`, `while`, `for`, `do`, `switch` e `return`).
  - Non mettere spazi davanti alle parentesi usate nelle chiamate di funzione.

- Mettere spazi attorno a tutti gli operatori C e dopo le virgole, nelle chiamate di funzione.
- Non usare l'operatore *virgola* per produrre degli effetti collaterali multipli, tranne che nelle parti di inizializzazione e incremento dei cicli `for`, e nel corpo delle macro.
- Usare dei caratteri TAB per l'indentazione, e non dei semplici spazi.
- Usare lo stile “K&R” per formattare le parti di programma incluse fra parentesi graffe.
- Usare confronti con `NULL` e `'\0'` per le condizioni contenute nelle istruzioni `if`, `while` e `for`, e anche nelle varie clausole `case` delle istruzioni `switch`, invece del semplice puntatore o il semplice valore del carattere.
- Usare i valori `true` e `false` per le variabili booleane, la costante simbolica `NULL` per i valori dei puntatori, e la costante carattere `'\0'` quando è il caso, invece dei valori 1 e 0.
- Fornire un commento descrittivo di una riga per ogni funzione.
- Non usare la funzione `alloca()` per allocare memoria dalla *stack*. Il farlo genera dei problemi di portabilità che non giustificano il vantaggio secondario di non doversi preoccupare di liberare la memoria. Usare invece `malloc()` e `free()`.
- Non usare confronti nella forma `'! strcmp(a, b)'` o simili. Come disse una volta Henry Spencer, “`strcmp()` non è una funzione booleana!” Usare invece `'strcmp(a, b) == 0'`.
- Per aggiungere nuovi valori a *flag* binari, usare costanti esadecimali esplicite (`0x001`, `0x002`, `0x004`, etc.) invece che spostare di un bit a sinistra in incrementi successivi (`'(1<<0)'`, `'(1<<1)'`, etc.).

**NOTA:** Qualora fossi costretto a riformattare completamente il codice per farlo aderire allo stile di codifica usato in `gawk`, potrei anche decidere di ignorare del tutto le modifiche proposte.

6. Aggiornare la documentazione. Insieme col nuovo codice, fornire nuove sezioni e/o capitoli per questo libro. Per quanto possibile, usare il formato Texinfo, invece di fornire soltanto del testo ASCII non formattato (sebbene un semplice testo sia già meglio che nessuna documentazione). Le convenzioni da seguire in *GAWK: Programmare efficacemente in AWK* sono elencate dopo la parola chiave `@bye` alla fine del file sorgente Texinfo. Se possibile, aggiornare anche la pagina di manuale in formato `man`.

Si dovrà anche firmare un documento liberatorio relativo alle modifiche apportate alla documentazione.

7. Inviare le modifiche come file di differenze nel formato contestuale unificato. Usare `'diff -u -r -N'` per confrontare i sorgenti originali dell'albero di sorgenti `gawk` con la versione proposta. Si raccomanda di usare la versione GNU di `diff` o, ancora meglio, `'git diff'` o `'git format-patch'`. Per inviare le modifiche proposte spedire l'output prodotto da `diff`. (Si veda la [Sezione B.4 \[Segnalazione di problemi e bug\]](#), pagina 493, per l'indirizzo di posta elettronica.)

L'uso di questo formato rende semplice per me l'applicazione delle modifiche alla versione principale del sorgente di `gawk` (usando il programma di utilità `patch`). Se invece tocca a me applicare le modifiche a mano, con un editor di testo, potrei decidere di non farlo, specialmente se le modifiche sono molte.

8. Includere una descrizione da aggiungere al file **ChangeLog** riguardo alla modifica da voi proposta. Questo serve a minimizzare l'attività a me richiesta, rendendo più facile per me l'accettazione delle modifiche, che diventa più semplice se si include anche questa parte nel file di differenze (nel formato *diff*).

Sebbene questa possa sembrare una richiesta molto esigente, si tenga presente che, anche se il nuovo codice non è opera mia, tocca poi a me mantenerlo e correggere eventuali errori. Se non è possibile per me farlo senza perderci troppo tempo, potrei anche lasciar perdere la modifica.

### C.2.3 Portare gawk su un nuovo Sistema Operativo

Se si vuol portare **gawk** su di un nuovo sistema operativo, sono necessari parecchi passi:

1. Seguire le linee-guida contenute nella precedente sezione relative allo stile di codifica, all'invio delle differenze proposte, etc.
2. Essere disposti a firmare un documento liberatorio appropriato. Se l'FSF deve poter distribuire le modifiche, queste vanno dichiarate di pubblico dominio, tramite la firma di un documento apposito, oppure attribuendo il copyright delle modifiche all'FSF. Entrambe queste azioni sono semplici da fare, e *molte* persone già l'hanno fatto. Se ci sono domande da fare, scrivere a me oppure all'indirizzo [gnu@gnu.org](mailto:gnu@gnu.org).
3. Nel realizzare un *port*, tener presente che il codice deve coesistere pacificamente con il resto di **gawk** e con le versioni per altri sistemi operativi. Evitare modifiche non necessarie alla parte di codice che è indipendente dal sistema operativo. Se possibile, evitare di disseminare `#ifdef`, utili solo per il proprio *port*, all'interno del codice sorgente.

Se le modifiche necessarie per un particolare sistema coinvolgono una parte troppo rilevante del codice, è probabile che io non le accetti. In questo caso si possono, naturalmente, distribuire le modifiche per proprio conto, basta che si rispettino i vincoli della GPL (si veda la [\[Licenza Pubblica Generale GNU \(GPL\)\]](#), pagina 529).

4. Un certo numero di file che fanno parte della distribuzione di **gawk** sono mantenuti da terze persone e non dagli sviluppatori di **gawk**. Quindi, non si dovrebbero cambiare, se non per ragioni molto valide; vale a dire, modifiche a questi file non sono impossibili, ma le modifiche a questi file saranno controllate con estrema attenzione. I file sono `dfa.c`, `dfa.h`, `getopt.c`, `getopt.h`, `getopt1.c`, `getopt_int.h`, `gettext.h`, `regcomp.c`, `regex.c`, `regex.h`, `regex_internal.c`, `regex_internal.h` e `regexexec.c`.
5. Un certo numero di altri file sono prodotti dagli Autotool [comandi di configurazione] di GNU (Autoconf, Automake, e GNU `gettext`). Neppure questi file dovrebbero essere modificati, se non per ragioni molto valide. I file sono `ABOUT-NLS`, `config.guess`, `config.rpath`, `config.sub`, `depcomp`, `INSTALL`, `install-sh`, `missing`, `mkinstalldirs`, `xalloc.h` e `ylwrap`.
6. Essere disponibili a continuare a mantenere il *port*. I sistemi operativi non-Unix sono supportati da volontari che tengono aggiornato il codice necessario per compilare ed eseguire **gawk** nei loro sistemi. Se nessuno è disponibile a tener aggiornato un *port*, questo diventa non più supportato, e può essere necessario rimuoverlo dalla distribuzione.
7. Fornire un appropriato file `gawkmisc.???.` Ogni *port* ha il proprio `gawkmisc.???.` che implementa alcune funzioni specifiche per quel sistema operativo. Questa è una soluzione più pulita, rispetto a una quantità di `#ifdef` sparsi nel codice. Il file `gawkmisc.c`

nella directory principale dei sorgenti include gli appropriati file `gawkmisc.???` da ogni sottodirectory. Anche quest'ultimo file va aggiornato.

Ogni file `gawkmisc.???` del *port* ha un suffisso esplicativo del tipo di macchina o del sistema operativo in questione—per esempio, `pc/gawkmisc.pc` e `vms/gawkmisc.vms`. L'uso di suffissi distinti invece di un semplice `gawkmisc.c`, rende possibile spostare file da una sottodirectory propria del *port* nella sottodirectory principale, senza cancellare incidentalmente il file `gawkmisc.c` vero e proprio. (Al momento, questo rappresenta un problema solo per i *port* ai sistemi operativi dei PC.)

8. Fornire un `Makefile` e ogni altro file sorgente o di intestazione in C che sia necessario per il proprio sistema operativo. Tutto il codice dovrebbe stare in una sottodirectory a parte, il cui nome sia lo stesso, o sia indicativo, del proprio sistema operativo o del tipo di computer. Se possibile, tentare di strutturare il codice in modo che non sia necessario spostare file dalla propria sottodirectory nella directory principale del codice sorgente. Se ciò non è possibile, evitare nel modo più assoluto di usare nomi per i file che siano duplicati di nomi di file presenti nella directory principale del codice sorgente.
9. Aggiornare la documentazione. Scrivere una sezione (o più sezioni) per questo libro che descriva i passi di installazione e compilazione necessari per compilare e/o installare `gawk` per il sistema desiderato.

Seguire queste indicazioni facilita molto l'integrazione delle modifiche in `gawk` e la loro felice coesistenza con il codice di altri sistemi operativi già presenti.

Nel codice che viene fornito e tenuto aggiornato, si possono tranquillamente usare uno stile di codifica e una disposizione delle parentesi graffe di proprio gradimento.

### C.2.4 Perché i file generati sono tenuti in Git

Se si esaminano i sorgenti di `gawk` nel deposito Git si noterà che sono inclusi file generati automaticamente dagli strumenti dell'infrastruttura GNU, come `Makefile.in` generato da Automake e anche `configure` proveniente da Autoconf.

Questo comportamento è differente da quello di molti progetti di Libero Software che non memorizzano i file derivati, per mantenere più sgombro il deposito Git, rendendo così più facile comprendere quali sono le modifiche sostanziali confrontando differenti versioni, nel tentativo di capire cosa è cambiato tra una modifica e la precedente.

Tuttavia, ci sono parecchie ragioni per le quali il manutentore di `gawk` preferisce mantenere ogni cosa nel deposito Git.

Innanzitutto, perché in questo modo è facile generare completamente ogni data versione, senza doversi preoccupare di avere a disposizione altri strumenti (più vecchi, probabilmente obsoleti, e in qualche caso perfino impossibili da trovare).

Come esempio estremo, se solo si pensa di tentare di compilare, diciamo, la versione Unix V7 di `awk`, ci si accorge che non solo è necessario scaricare e ricompilare la versione V7 del comando `yacc` per farlo, ma anche che serve la versione V7 del comando `lex`. E quest'ultima è praticamente impossibile farla funzionare in un sistema GNU/Linux dei giorni nostri.<sup>1</sup>

(Oppure, diciamo che la versione 1.2 di `gawk` richiede il comando `bison` come funzionava nel 1989, e non è presente il file `awkgram.c` [generato tramite `bison`] nel deposito Git.

<sup>1</sup> Ci abbiamo provato. È stata un'esperienza dolorosa.

Che cosa ci garantisce di riuscire a trovare quella versione di **bison**? O che *quella* riesca a generarlo?)

Se il deposito Git comprende tutti i file derivati, è facile, dopo averli scaricati, ricostruire il programma. (Oppure è *più facile*, a seconda di quanto si vuole risalire indietro nel tempo).

E qui arriviamo alla seconda (e più valida) ragione per cui tutti i file devono essere proprio nel deposito Git. Domandiamoci a chi ci si rivolge: agli sviluppatori di **gawk**, oppure a un utilizzatore che intende solo scaricare una data versione e provarla?

Il manutentore di **gawk** desidera che per tutti gli utenti **awk** interessati sia possibile limitarsi a clonare il deposito sul proprio computer, selezionare la variante che lo interessa e costruirla. Senza doversi preoccupare di avere a disposizione le versioni corrette degli Autotool GNU.<sup>2</sup> A questo serve il file **bootstrap.sh**. Va a "toccare" [tramite il comando **touch**] vari altri file nell'ordine richiesto in modo che

```
# La formula canonica per compilare il software GNU:
./bootstrap.sh && ./configure && make
```

tutto *funzioni senza problemi*.

Questo è estremamente importante per i rami **master** e **gawk-X.Y-stable**.

Inoltre, il manutentore di **gawk** potrebbe sostenere che ciò è importante anche per gli sviluppatori di **gawk**. Tentando di scaricare il ramo **xgawk**<sup>3</sup> per compilarlo, non ci riuscì. (Mancava il file **ltmain.sh**, ed egli non aveva idea di come crearlo, e c'erano anche ulteriori problemi.)

La cosa lo lasciò in uno stato di frustrazione *estrema*. Riguardo a quel ramo, il manutentore è in una posizione non differente da quella di un utente generico che voglia tentare di compilare **gawk-4.1-stable** o **master** dal deposito Git.

Quindi, il manutentore ritiene che sia non solo importante, ma cruciale, che per ogni dato ramo la formula canonica evocata prima *funzioni senza problemi*.

Una terza ragione per avere tutti i file è che senza di essi, usare '**git bisect**' per tentare di trovare quale modifica ha introdotto un errore diventa estremamente difficile. Il manutentore ha tentato di farlo su un altro progetto che richiede di eseguire *script* di inizializzazione allo scopo di creare lo *script configure* e così via; è stata un'esperienza veramente dolorosa. Se invece il deposito Git contiene tutto il necessario, usare **git bisect** al suo interno è molto facile.

Quali sono, quindi, alcune delle conseguenze e/o delle cose da fare?

1. Non importa se ci sono file differenti nei diversi rami prodotti da versioni differenti degli Autotool.

A. Spetta al manutentore integrarli tra loro, e se ne occuperà lui.

---

<sup>2</sup> Ecco un programma GNU che (secondo noi) è estremamente difficile da estrarre dal deposito Git e compilare. Per esempio, in un vecchio (ma ancora funzionante) PowerPC Macintosh, con il sistema operativo Mac Os X 10.5, è stato necessario scaricare e compilare una tonnellata di software, incominciando dallo stesso programma Git, per riuscire a lavorare con l'ultima versione del codice. Non è un'esperienza piacevole e, specie sui vecchi sistemi, è una notevole perdita di tempo.

Neppure partire dall'ultimo archivio **tar** compresso è stata una passeggiata: i manutentori avevano eliminato i file compressi in formato **.gz** e **.bz2** sostituendoli con file di tipo **.tar.xz**. Bisognava quindi per prima cosa scaricare e compilare **xz**

<sup>3</sup> Un ramo (non più presente) creato da uno degli altri sviluppatori, e che non includeva i file generati.

- B. È facile per lui eseguire `'git diff x y > /tmp/diff1 ; gvim /tmp/diff1'` per rimuovere le differenze che non sono rilevanti ai fini della revisione del codice sorgente.
2. Sarebbe sicuramente d'aiuto se tutti usassero le stesse versioni degli Autotool GNU che lui usa, che in generale sono le ultime versioni rilasciate di Automake, Autoconf, **bison** e GNU **gettext**.

Installare a partire dal sorgente è abbastanza facile. È il modo con cui il manutentore ha lavorato per anni (e ancora lavora). Egli aveva `/usr/local/bin` all'inizio del suo `PATH` e dava i seguenti comandi:

```
wget http://ftp.gnu.org/gnu/package/package-x.y.z.tar.gz
tar -xpvf package-x.y.z.tar.gz
cd package-x.y.z
./configure && make && make check
make install    # come utente root
```

La maggior parte del testo precedente fa parte di messaggi scritti originalmente dal manutentore agli altri sviluppatori **gawk**. Da uno degli sviluppatori è stata avanzata l'obiezione "... che chi scarica il sorgente da Git non è un utente finale".

Tuttavia, questo non è esatto. Ci sono "utenti avanzati di **awk**" che possono installare **gawk** (usando la formula canonica vista sopra) ma che non conoscono il linguaggio C. Quindi, i rami più rilevanti dovrebbero essere sempre compilabili.

È stato proposto poi di lanciare ogni notte uno *script* tramite il programma di utilità **cron** per creare archivi in formato **tar** contenenti tutto "il codice sorgente." Il problema in questo caso è che ci sono differenti alberi di sorgenti, che corrispondono ai vari rami! Quindi gli archivi notturni in questione non sono una risposta valida, anche per il fatto che il deposito Git può rimanere senza alcuna modifica significativa per settimane intere.

Fortunatamente, il server Git può rispondere a questa esigenza. Per ogni dato ramo chiamato *nome-ramo*, basta usare:

```
wget http://git.savannah.gnu.org/cgit/gawk.git/snapshot/gawk-nome-ramo.tar.gz
```

per ottenere una copia utilizzabile del ramo in questione.

### C.3 Probabili estensioni future

*AWK è un linguaggio simile a PERL, solo che è notevolmente più elegante.*

—Arnold Robbins

*Hey!*

—Larry Wall

Il file `TODO` nel ramo `master` del deposito Git di **gawk** contiene un elenco di possibili futuri miglioramenti. Alcuni di questi riguardano il codice sorgente, e altri possibili nuove funzionalità. Consultare quel file per esaminare la lista. Si veda la [Sezione C.2 \[Fare aggiunte a \*\*gawk\*\*\]](#), [pagina 499](#), se si è interessati a intraprendere qualcuno dei progetti colà elencati.

### C.4 Alcune limitazioni dell'implementazione

La tabella seguente specifica i limiti di **gawk** in un sistema di tipo Unix (sebbene anche tra questi ci potrebbero essere variazioni). Altri sistemi operativi possono avere limiti differenti.

<b>Caratteristica</b>	<b>Limiti</b>
Caratteri in una classe di caratteri	$2^{\text{(numero di bit per byte)}}$
Lunghezza di un record in input	MAX_INT
Lunghezza di un record in output	Illimitata
Lunghezza di una riga di codice sorgente	Illimitata
Numero di campi in un record	MAX_LONG
Numero di ridirezioni di file	Illimitata
Numero di record in input in un singolo file	MAX_LONG
Numero totale di record in input	MAX_LONG
Numero di ridirezioni via <i>pipe</i>	min(numero processi per utente, numero di file aperti)
Valori numerici	Numeri a virgola mobile in doppia precisione (se non si usa la funzionalità MPFR)
Dimensione di un campo	MAX_INT
Dimensione di una stringa letterale	MAX_INT
Dimensione di una stringa di <i>printf</i>	MAX_INT

## C.5 Note di progetto dell'estensione API

Questa sezione documenta l'architettura dell'estensione API, inclusa una trattazione sommaria della progettazione e dei problemi che andavano risolti.

La prima versione delle estensioni per **gawk** è stata sviluppata a metà degli anni '90, e distribuita con la versione 3.1 di **gawk**, verso la fine degli anni '90. Il meccanismo e l'architettura sono rimasti gli stessi per quasi 15 anni, fino al 2012.

Il vecchio meccanismo delle estensioni usava tipi di dati e funzioni dello stesso **gawk**, con un “abile trucco” per installare le funzioni di estensione.

La distribuzione **gawk** conteneva alcuni esempi di estensioni, solo poche delle quali erano realmente utili. Tuttavia era chiaro fin da principio che il meccanismo di estensione era un'aggiunta improvvisata, e non era realmente ben concepito.

### C.5.1 Problemi con le vecchie estensioni

Il vecchio meccanismo delle estensioni presentava parecchi problemi:

- Dipendeva eccessivamente dalla struttura interna di **gawk**. Ogni volta che la struttura **NODE**<sup>4</sup> veniva modificata, ogni estensione doveva essere ricompilata. Inoltre, la scrittura di estensioni richiedeva una certa familiarità con le funzioni interne di **gawk**. Esisteva un po' di documentazione in questo libro, ma era ridotta al minimo.
- Per poter utilizzare servizi di **gawk** da un'estensione era necessario disporre di funzionalità del *linker* normalmente disponibili in ambiente di tipo Unix, ma non implementate nei sistemi MS-Windows; chi voleva utilizzare estensioni in MS-Windows doveva aggungerle al modulo eseguibile di **gawk**, anche se MS-Windows supporta il caricamento dinamico di oggetti condivisi.

---

<sup>4</sup> Una struttura di dati fondamentale all'interno di **gawk**.

- L'API di tanto in tanto veniva modificata, in parallelo ai cambiamenti di **gawk**; nessuna compatibilità tra le versioni è stata mai prevista o resa disponibile.

Nonostante questi inconvenienti, gli sviluppatori del progetto **xgawk** si basarono su **gawk** per sviluppare parecchie estensioni significative. Inoltre, migliorarono le capacità, in **gawk**, di includere file e di accedere a oggetti condivisi.

Una nuova API è rimasta un desiderio per lungo tempo, ma solo nel 2012 il manutentore di **gawk** e gli sviluppatori di **xgawk** iniziarono finalmente a lavorare insieme. Ulteriori informazioni riguardanti il progetto **xgawk** sono forniti nella [Sezione 16.8 \[Il progetto gawkextlib\]](#), pagina 455.

### C.5.2 Obiettivi per un nuovo meccanismo

Alcuni obiettivi per la nuova API sono:

- L'API dovrebbe essere indipendente dalla struttura interna di **gawk**. Le modifiche alla struttura interna di **gawk** dovrebbero essere irrilevanti per chi scrive una funzione di estensione.
- L'API dovrebbe consentire una compatibilità *binaria* [ossia a livello di codice eseguibile, e non solo a livello di codice sorgente] tra diverse versioni di **gawk**, se la stessa API rimane invariata.
- L'API dovrebbe consentire che le estensioni scritte in C o C++ abbiano all'incirca la stessa "struttura", per il codice awk, di quella che hanno le funzioni di **awk**. Questo vuol dire che le estensioni dovrebbero avere:
  - La capacità di accedere ai parametri delle funzioni.
  - La capacità di trasformare un parametro indefinito in un vettore (chiamata per riferimento [*by reference*]).
  - La capacità di creare, leggere e aggiornare variabili globali.
  - Un accesso facile a tutti gli elementi di un vettore simultaneamente ("appiattimento del vettore") in modo da poter visitare tutti gli elementi del vettore in una maniera semplice per un programma scritto in C.
  - La capacità di creare vettori (compresi i veri "vettori di vettori" di **gawk**).

Alcuni ulteriori obiettivi rilevanti sono:

- L'API dovrebbe usare solo funzionalità disponibili nella specifica ISO C 90, in modo da consentire estensioni scritte con una vasta gamma di compilatori C e C++. L'intestazione dovrebbe includere le appropriate direttive `#ifdef __cplusplus` e `extern "C"`, in modo da poter utilizzare un compilatore C++. (Se si usa C++, il sistema operativo in uso dev'essere in grado di invocare dei costruttori e dei distruttori, poiché **gawk** è un programma scritto in C. Al momento in cui queste note sono scritte, la cosa non è stata verificata).
- Il meccanismo API non dovrebbe aver bisogno di accedere ai simboli di **gawk**<sup>5</sup> da parte del *linker* statico, usato in fase di compilazione, o di quello dinamico, in modo da rendere possibile la creazione di estensioni che funzionino anche in ambiente MS-Windows.

---

<sup>5</sup> I *simboli* sono le variabili e le funzioni definite all'interno di **gawk**. Accedere a questi simboli da parte di codice esterno a **gawk** caricato dinamicamente al momento dell'esecuzione è problematico in ambiente MS-Windows.

In fase di sviluppo, è apparso evidente che dovevano essere disponibili alle estensioni anche altre funzionalità, che sono state successivamente implementate:

- Le estensioni dovrebbero essere in grado di agganciarsi al meccanismo di ridirezione dell'I/O di **gawk**. In particolare, gli sviluppatori di **xgawk** hanno programmato un cosiddetto “gancio aperto” (*open hook*) per gestire autonomamente la lettura dei record. In fase di sviluppo, questo meccanismo è stato generalizzato, per consentire alle estensioni di agganciarsi sia all'elaborazione dell'input, che a quella dell'output, nonché all'I/O bidirezionale.
- Un'estensione dovrebbe poter rendere disponibile una funzione di “call back” (richiamo) per effettuare operazioni di pulizia all'uscita di **gawk**.
- Un'estensione dovrebbe poter rendere disponibile una stringa di versione così che l'opzione `--version` di **gawk** possa informare anche sulle versioni delle estensioni.

Il vincolo di evitare di accedere ai simboli di **gawk** può parere a prima vista piuttosto difficile da rispettare.

Un tipo di architettura, apparentemente usato da Perl e Ruby e forse da altri programmi, potrebbe consistere nel mettere il codice principale di **gawk** in una libreria, limitando il programma di utilità **gawk** a una piccola funzione `main()` in C che richiamerebbe dinamicamente la libreria.

Questo inverte i ruoli del programma principale e della sua estensione, e complica sia la compilazione che l'installazione, trasformando la semplice copia del programma eseguibile **gawk** da un sistema all'altro (o da una posizione all'altra all'interno dello stesso sistema) in un'operazione ad alto rischio.

Pat Rankin ha suggerito la soluzione che è stata adottata. Si veda la [Sezione 16.3 \[Una panoramica sul funzionamento ad alto livello\]](#), [pagina 396](#), per maggiori dettagli.

### C.5.3 Altre scelte progettuali

Per una scelta progettuale arbitraria, le estensioni possono accedere ai valori delle variabili e dei vettori predefiniti (come `ARGV` e `FS`), ma non possono modificarli, con la sola eccezione di `PROCINFO`.

Il motivo di questa scelta è di impedire a una funzione di estensione di alterare il flusso di un programma **awk** togliendogli il controllo. Mentre una vera funzione di **awk** può fare quel che vuole, a discrezione del programmatore, una funzione di estensione dovrebbe fornire un servizio, o rendere disponibile un'API C da utilizzare all'interno di **awk**, senza interferire con le variabili `FS` o `ARGC` e `ARGV`.

Inoltre, diverrebbe facile avviarsi su un sentiero scivoloso. Quante funzionalità di **gawk** dovrebbero essere disponibili alle estensioni? Devono poter usare `getline`? Oppure richiamare `gsub()` o compilare espressioni regolari? Oppure richiamare funzioni interne di **awk**? (*Questo potrebbe creare molta confusione.*)

Per evitare questi problemi, gli sviluppatori di **gawk** hanno scelto di iniziare con le funzionalità più semplici e di base, che sono comunque veramente utili.

Sebbene **gawk** consenta cose interessanti come l'MPFR, e vettori indicizzati internamente con numeri interi, un'altra decisione è stata quella di non rendere disponibili all'API queste funzionalità, per amor di semplicità e per restare fedeli alla tradizionale semantica di **awk**.

(In effetti, i vettori indicizzati internamente con numeri interi sono talmente trasparenti che non sono neppure documentati!)

In più, tutte le funzioni nell'API controllano che i puntatori ai parametri passati loro in input non siano `NULL`. Se lo sono, viene emesso un messaggio di errore. (È bene che il codice di estensione verifichi che i puntatori ricevuti da `gawk` non siano `NULL`. Ciò non dovrebbe succedere, ma gli sviluppatori di `gawk` sono solo degli esseri umani, e capita anche a loro di commettere degli errori, di tanto in tanto.)

Col tempo, l'API si svilupperà certamente; gli sviluppatori di `gawk` si aspettano che questo avvenga in base alle necessità degli utenti. Per ora, l'API disponibile sembra fornire un insieme di funzionalità minimo, eppure potente, per creare estensioni.

### C.5.4 Possibilità di sviluppo futuro

L'API può ancora essere ampliata, in due modi:

- `gawk` passa un “identificativo di estensione” all'estensione in fase di caricamento dell'estensione. L'estensione a sua volta restituisce questo identificativo a `gawk` a ogni chiamata di funzione. Questo meccanismo consente a `gawk` di identificare l'estensione che lo chiama, se la cosa dovesse risultare necessaria.
- Analogamente, l'estensione passa uno “spazio dei nomi” a `gawk` in fase di registrazione di ogni funzione estesa. Questo è fatto in vista di un possibile futuro meccanismo per raggruppare funzioni di estensione, e per evitare in questo modo possibili conflitti nei nomi di funzione.

Naturalmente, al momento in cui queste righe sono state scritte, nessuna decisione è stata presa riguardo ai punti sopra descritti.

## C.6 Compatibilità per le vecchie estensioni

Il [Capitolo 16 \[Scrivere estensioni per `gawk`\]](#), [pagina 395](#), descrive le API supportate e i meccanismi per scrivere estensioni per `gawk`. Quest'API è stata introdotta nella versione 4.1. Peraltro, già da molti anni `gawk` metteva a disposizione un meccanismo di estensione che richiedeva una familiarità con la struttura interna di `gawk` e che non era stato progettato altrettanto bene.

Per garantire un periodo di transizione, `gawk` versione 4.1 continua a supportare il meccanismo di estensione originale. Questo rimarrà disponibile per la durata di una sola versione principale. Il supporto cesserà, e sarà rimosso dal codice sorgente, al rilascio della prossima versione principale.

In breve, le estensioni in stile originale dovrebbero essere compilate includendo il file di intestazione `awk.h` nel codice sorgente dell'estensione. Inoltre, va definito l'identificativo ‘`GAWK`’ durante la preparazione (si usi ‘`-DGAWK`’ con compilatori in stile Unix). Se non lo si fa, le definizioni in `gawkapi.h` risulteranno in conflitto con quelle in `awk.h` e l'estensione non sarà compilabile.

Come nelle versioni precedenti, un'estensione vecchio stile sarà caricata usando la funzione predefinita `extension()` (che non viene ulteriormente documentata). Questa funzione, a sua volta, trova e carica il file oggetto condiviso che contiene l'estensione e chiama la sua routine `C dl_load()`.

Poiché le estensioni in stile originale e quelle nello stile nuovo usano differenti routine di inizializzazione(`dl_load()` e `dlload()`, rispettivamente), esse possono tranquillamente essere installate nella stessa directory (il cui nome deve essere contenuto nella variabile `AWKLIBPATH`) senza problemi di conflitti.

Il *team* di sviluppo di **gawk** raccomanda caldamente di convertire ogni estensione del vecchio tipo ancora in uso, in modo da utilizzare la nuova API descritta nel [Capitolo 16](#) [Scrivere estensioni per **gawk**], pagina 395.

## C.7 Sommario

- Le estensioni di **gawk** possono essere disabilitata sia con l'opzione `--traditional` che con l'opzione `--posix`. L'opzione `--parsedebug` è disponibile se **gawk** è stato compilato con `'-DDEBUG'`.
- Il codice sorgente di **gawk** è conservato in un deposito Git pubblicamente accessibile. Tutti possono scaricarlo e visualizzare il codice sorgente.
- I contributi a **gawk** sono benvenuti. Seguire le istruzioni delineate in questo capitolo renderà più agevole integrare i contributi degli utenti nel codice principale. Questo vale sia per il contributo di nuove funzionalità che per il *porting* a ulteriori sistemi operativi.
- **gawk** ha alcuni limiti: generalmente quelli imposti dall'architettura hardware della macchina.
- La progettazione dell'estensione API è volta a risolvere alcuni problemi riscontrati nel precedente meccanismo di estensione, ad abilitare funzionalità richieste dal progetto **xgawk**, e a fornire una compatibilità binaria in futuro.
- Il precedente meccanismo di estensione è ancora supportato nella versione 4.1 di **gawk**, ma sarà *rimosso* nella prossima versione principale.

## Appendice D Concetti fondamentali di programmazione

Quest'appendice si propone di definire alcuni dei concetti e termini fondamentali usati nel resto di questo libro. Poiché questo libro è dedicato ad `awk`, e non riguarda la programmazione al computer in generale, la trattazione è necessariamente piuttosto generica e semplificata. (Se serve qualcosa di più approfondito, ci sono molti altri testi introduttivi che si possono consultare.)

### D.1 Quel che fa un programma

Al livello più fondamentale, il compito di un programma è di elaborare alcuni dati in input e produrre dei risultati. Si veda la [Figura D.1](#).

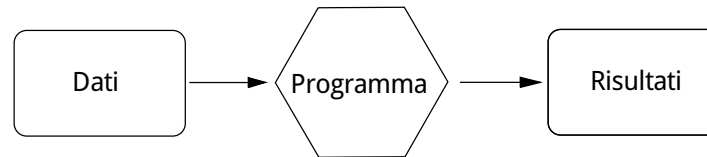


Figura D.1: Flusso generico di un programma

Il “programma” nella figura può essere sia un programma compilato<sup>1</sup> (come `ls`), sia un programma *interpretato*. In quest’ultimo caso, un programma direttamente eseguibile dal computer, come `awk`, legge il programma, e quindi usa le istruzioni in esso contenute per elaborare i dati.

Quando si scrive un programma, esso è composto normalmente dai seguenti insiemi di istruzioni di base, come si vede nella [Figura D.2](#):

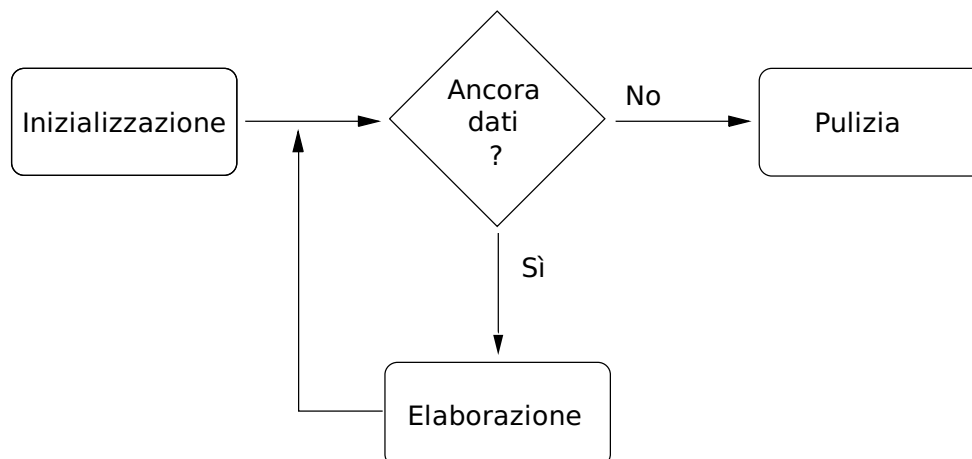


Figura D.2: Fasi di un programma generico

<sup>1</sup> I programmi compilati sono normalmente scritti in linguaggi di programmazione di livello più basso, come C, C++, o Ada, e quindi tradotti, o *compilati*, in una forma che il computer può eseguire direttamente.

## Inizializzazione

Queste sono le cose da fare prima di iniziare la reale elaborazione dei dati, per esempio controllare gli argomenti con cui è stato invocato il programma, inizializzare dei dati di cui si potrà aver bisogno per la successiva elaborazione, e così via. Questa fase corrisponde alla regola **BEGIN** di **awk** (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148).

Nella preparazione di una torta, questa fase corrisponde a quella di tirar fuori tutti i contenitori in cui mischiare gli ingredienti, e la teglia in cui cuocerla, e nell'assicurarsi di avere a disposizione tutti gli ingredienti necessari.

## Elaborazione

Questa fase è quella in cui si svolge il lavoro vero e proprio. Il programma legge i dati, raggruppati in “pezzi logici”, e li elabora secondo necessità.

In quasi tutti i linguaggi di programmazione, la lettura dei dati va gestita manualmente, controllando dopo ogni lettura se è rimasto ancora qualcosa d'altro da leggere. Il paradigma *criterio di ricerca-azione* di **awk** (si veda il [Capitolo 1 \[Per iniziare con awk\]](#), pagina 17) gestisce automaticamente la parte di lettura dati.

Nella preparazione di una torta, l'elaborazione corrisponde all'attività vera e propria: rompere le uova, mescolare la farina, l'acqua e gli altri ingredienti, e quindi mettere la torta a cuocere nel forno.

## Pulizia

Una volta elaborati tutti i dati, ci sono attività da svolgere prima di aver finito. Questa fase corrisponde alla regola **END** di **awk**. (si veda la [Sezione 7.1.4 \[I criteri di ricerca speciali BEGIN ed END\]](#), pagina 148).

Dopo che la torta è stata tirata fuori dal forno, va fatta raffreddare e avvolta in una pellicola trasparente per evitare che qualcuno la assaggi, e inoltre vanno lavati i contenitori e le posate.

Un *algoritmo* è la descrizione dettagliata della procedura necessaria per svolgere un compito o per elaborare dati. Lo si può paragonare alla ricetta per preparare una torta. I programmi sono il modo con cui un algoritmo viene eseguito da un computer. Spesso è compito del programmatore sia sviluppare un algoritmo, sia programmarlo.

I “pezzi logici” nominati precedentemente sono chiamati *record* (registrazioni), in analogia con le registrazioni del personale di una ditta, degli studenti di una scuola, o dei pazienti di un dottore. Ogni record è composto di molte parti, per esempio nome, cognome, data di nascita, indirizzo, e così via. Le parti di cui è composto un record sono chiamate *campi* del record.

L'atto di leggere i dati è noto come *input*, e quello di generare risultati è, come facilmente prevedibile, chiamato *output*. Spesso i due sono riuniti sotto il nome di “input/output” e, ancor più spesso, con l'abbreviazione “I/O”. (In inglese “input” e “output” sono spesso usati come verbi, nel gergo informatico, al posto di leggere e scrivere.)

**awk** gestisce la lettura dei dati, come anche la divisione in record e campi. Lo scopo del programma dell'utente è di dire ad **awk** cosa fare con i dati. Questo vien fatto descrivendo *modelli* da ricercare nei dati e *azioni* da eseguire qualora si siano trovati questi modelli. Questa caratteristica dei programmi **awk**, di essere *guidati-dai-dati*, di solito li rende più facili sia da scrivere che da leggere.

## D.2 Valore dei dati in un computer

In un programma si tiene traccia di informazioni e valori in contenitori chiamati *variabili*. Una variabile è solo un nome per designare un certo valore, come **nome**, **cognome**, **indirizzo**, e così via. **awk** ha molte variabili predefinite, e ha dei nomi speciali per designare il record in input corrente e i campi che compongono il record stesso. Si possono inoltre raggruppare molti valori associati tra di loro sotto un unico nome, utilizzando un vettore.

I dati, in particolare in **awk**, possono avere valori numerici, come 42 o 3.1415927, o avere come valore delle stringhe. Un valore di tipo stringa è essenzialmente qualsiasi cosa che non sia un numero, per esempio un nome. Le stringhe sono talora chiamate *dati di tipo carattere*, poiché memorizzano i singoli caratteri che le formano. Le singole variabili, come pure le variabili numeriche e di tipo stringa, sono definite come valori *scalari*. Raggruppamenti di valori, come i vettori, non sono scalari.

La [Sezione 15.1 \[Una descrizione generale dell'aritmetica del computer\]](#), pagina 379, ha fornito un'introduzione di base ai tipi numerici (interi e a virgola mobile) e a come questi sono usati in un computer. Si consiglia di rileggere quelle informazioni, comprese le numerose avvertenze là esposte.

Mentre è probabile che ci si sia abituati all'idea di un numero senza un valore (cioè, allo zero), richiede un po' più di riflessione abituarsi all'idea di dati di tipo carattere a lunghezza zero. Nonostante ciò, questo tipo di dato esiste. È chiamato *stringa nulla*. La stringa nulla è un dato di tipo carattere che non ha un valore. In altre parole, è vuoto. Si scrive così nei programmi **awk**: `""`.

Gli esseri umani sono abituati a usare il sistema decimale, cioè a base 10. In base 10, i numeri vanno da 0 a 9, e poi “vengono riportati” nella colonna successiva. (Chi si ricorda la scuola elementare?  $42 = 4 \times 10 + 2$ .)

Ma esistono anche altre basi per i numeri. I computer normalmente usano la base 2 o *binaria*, la base 8 o *ottale*, e la base 16 o *esadecimale*. Nella numerazione binaria, ogni colonna rappresenta il doppio del valore della colonna alla sua destra. Ogni colonna può contenere solo uno 0 o un 1. Quindi, il numero binario 1010 rappresenta  $(1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)$ , ossia il numero decimale 10. Le numerazioni ottale ed esadecimale sono trattate più ampiamente nella [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115.

Al livello più basso possibile, i computer memorizzano i valori come gruppi di cifre binarie, o *bit*. I computer moderni raggruppano i bit in gruppi di otto, detti *byte*. Applicazioni avanzate talora hanno necessità di manipolare i bit direttamente, e **gawk** è dotato di apposite funzioni.

I programmi sono scritti nei linguaggi di programmazione. Esistono centinaia, se non migliaia, di linguaggi di programmazione. Uno dei più diffusi è il linguaggio di programmazione C. Il linguaggio C ha esercitato un'influenza molto forte nella progettazione del linguaggio **awk**.

Ci sono state parecchie versioni di C. La prima è spesso designata come “K&R” C, dalle iniziali di Brian Kernighan e Dennis Ritchie, gli autori del primo libro sul C. (Dennis Ritchie ha creato il linguaggio, e Brian Kernighan è stato uno dei creatori di **awk**.)

A metà degli anni '80 è iniziato uno sforzo rivolto a produrre uno standard internazionale per il C. Questo lavoro ha raggiunto un punto di arrivo nel 1989 con la produzione dello standard ANSI per il C. Questo standard è diventato uno standard ISO nel 1990. Nel 1999,

uno standard ISO C revisionato è stato approvato e pubblicato. Dove è opportuno, POSIX `awk` è compatibile con lo standard ISO C del 1999.

# Glossario

## Abbraccio mortale

La situazione in cui due processi che comunicano tra loro sono entrambi bloccati, in attesa che l'altro processo faccia qualcosa.

**Ada** Un linguaggio di programmazione originalmente definito dal Department of Defense U.S.A. per la programmazione integrata. È stato progettato per favorire dei buoni metodi da seguire nell'ingegneria del software.

**Ambiente** Si veda “Variabili d’ambiente”.

**Àncora** I metacaratteri *regex* ‘^’ e ‘\$’, che richiedono che la corrispondenza che si sta cercando si trovi all’inizio o alla fine di una stringa, rispettivamente.

## Angolo buio

Un’area del linguaggio le cui specifiche spesso non erano (o ancora non sono) chiare, col risultato di ottenere un comportamento inatteso o non desiderabile. Tali aree sono segnalate in questo libro con il disegno di una torcia a margine e sono riportate nell’indice analitico sotto la voce “angolo buio”.

**ANSI** L’American National Standards Institute. Questo ente produce parecchi standard, e tra questi gli standard per i linguaggi di programmazione C e C++. Questi standard spesso diventano anche internazionali. Si veda anche “ISO”.

## Argomento

Un argomento può essere due cose differenti. Può essere un’opzione o un nome-file passato a un comando mentre lo si invoca dalla riga dei comandi, oppure può essere qualcosa passato a una *funzione* all’interno di un programma, per esempio all’interno di **awk**.

In quest’ultimo caso, un argomento può essere passato a una funzione in due modi. Nel primo modo è passato come valore alla funzione chiamata, ossia una copia del valore della variabile è reso disponibile alla funzione chiamata, ma la variabile originale non può essere modificata dalla funzione stessa. Nel secondo modo l’argomento è passato per riferimento, ossia un puntatore alla variabile in questione è passato alla funzione, che può quindi modificarla direttamente. In **awk** le variabili scalari sono passate per valore, e i vettori sono passati per riferimento. Si veda “Passaggio per valore/riferimento”.

## Arrotondamento

Arrotondare il risultato di un’operazione aritmetica può essere difficile. C’è più di un modo di arrotondare, e in **gawk** è possibile scegliere quale metodo dovrebbe essere usato all’interno di un programma. Si veda la [Sezione 15.4.5 \[Impostare la modalità di arrotondamento\]](#), pagina 387.

## Assegnamento

Un’espressione **awk** che cambia il valore di qualche variabile o dato oggetto di **awk**. Un oggetto a cui si può assegnare un valore è detto un *lvalue*. I valori assegnati sono chiamati *rvalue*. Si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126.

Assembler incredibilmente scritto in **awk**

Henry Spencer dell'Università di Toronto ha scritto un assembler adatto a molti diversi hardware, usando solo *script sed* e **awk**. È lungo migliaia di righe, e include la descrizione dell'hardware di numerosi micro-computer a 8 bit. È un buon esempio di programma per cui sarebbe stato meglio utilizzare un altro linguaggio. Si può scaricare da <http://awk.info/?awk100/aaa>.

**Asserzione** Un'istruzione in un programma che afferma che una condizione è verificata in un dato punto di un programma. Utile per ragionare su come si suppone funzioni un programma.

**Azione** Una serie di istruzioni **awk** associate a una regola. Se l'espressione di ricerca della regola individua un record in input, **awk** esegue su quel record l'azione relativa. Le azioni sono sempre racchiuse tra parentesi graffe. (Si veda la [Sezione 7.3 \[Azioni\]](#), [pagina 152](#)).

**Bash** La versione GNU della shell standard (il **B**ourne-**A**gain **S**hell). Si veda anche “Bourne Shell”.

**Binario** Notazione a base due, che usa le cifre 0–1. Poiché i circuiti elettronici funzionano “naturalmente” in base 2 (basta pensare a Off/On), ogni cosa all'interno di un computer è calcolata usando la base 2. Ciascuna cifra rappresenta la presenza (o l'assenza) di una potenza di 2 ed è chiamata un *bit*. Così, per esempio, il numero in base due 10101 rappresenta il numero in base decimale 21,  $((1 \times 16) + (1 \times 4) + (1 \times 1))$ .

Poiché i numeri in base due diventano rapidamente molto lunghi sia da leggere che da scrivere, normalmente li si unisce a gruppi di tre (ossia, sono visti come numeri ottali) o a gruppi di quattro (ossia, sono visti come numeri esadecimali). Non c'è un modo diretto per inserire numeri a base due in un programma C. Se necessario, tali numeri vengono solitamente inseriti come numeri ottali o esadecimali. Il numero di cifre in base due contenuto nei registri usati per rappresentare i numeri interi all'interno dei computer è un'indicazione approssimativa della potenza di calcolo del computer stesso. La maggior parte dei computer oggi usa 64 bit per rappresentare i numeri interi nei registri di calcolo, ma registri a 32 bit, 16 bit e 8 bit sono stati largamente in uso in passato. Si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), [pagina 115](#).

**Biscotto della fortuna**

Una particolare perla di saggezza, segno, detto o ricordo prodotto da (o presentato a) un programma. (Con vivi ringraziamenti al Prof. Doug McIlroy).

**Bit** Abbreviazione di “Binary Digit” [cifra binaria]. Tutti i valori nella memoria di un computer sono rappresentati nella forma di cifre binarie: valori che sono zero o uno. Gruppi di bit possono essere interpretati diversamente—come numeri interi, numeri a virgola mobile, dati di tipo carattere, indirizzi di altri oggetti contenuti in memoria, o altri dati ancora. **awk** permette di lavorare con numeri a virgola mobile e stringhe. **gawk** permette di manipolare bit con le funzioni predefinite descritte nella [Sezione 9.1.6 \[Funzioni per operazioni di manipolazione bit\]](#), [pagina 219](#).

I computer sono spesso definiti dal numero di bit che usano per rappresentare valori interi. Molti sistemi sono a 32-bit, ma i sistemi a 64-bit sono sempre più numerosi, mentre i sistemi a 16-bit [e quelli a 8-bit] sono praticamente scomparsi.

#### Bourne Shell

La shell standard (`/bin/sh`) in Unix e nei sistemi derivati da Unix, Originariamente scritto da Steven R. Bourne dei Bell Laboratories. Molte shell (`Bash`, `ksh`, `pdksh`, `zsh`) sono generalmente compatibili con la Bourne shell, anche quando offrono ulteriori funzionalità.

#### C

Il linguaggio di programmazione di sistema con cui è scritta la maggior parte del software GNU. Il linguaggio di programmazione `awk` ha una sintassi simile a quella del C, e questo libro puntualizza, quando serve, le somiglianze esistenti fra `awk` e C.

In generale, `gawk` tenta di essere ragionevolmente simile alla versione 1990 del C ISO.

#### C Shell

La C Shell (`cs`h o la sua versione migliorata `tc`sh) è una shell Unix creata da Bill Joy verso la fine degli anni '70. La C shell si differenzia dalle altre shell per le sue funzionalità interattive, e per lo stile complessivo, che è abbastanza simile a quello del linguaggio C. La C shell non è compatibile all'indietro con la Bourne Shell, e per questo motivo un'attenzione speciale è necessaria se si convertono alla C shell degli script scritti per altre shell Unix, in particolare per ciò che concerne la gestione delle variabili di shell. Si veda anche "Bourne Shell".

#### C++

Un linguaggio di programmazione molto diffuso, orientato agli oggetti, derivato dal C.

#### Campo

Quando `awk` legge un record in input, suddivide il record in parti separate da spazi vuoti (o da una *regex* che individua il separatore, modificabile reimpostando la variabile predefinita `FS`). Tali parti sono dette campi. Se le parti sono di lunghezza fissa, si può usare la variabile predefinita `FIELDWIDTHS` per descriverne le lunghezze. Se si desidera specificare i contenuti dei campi, piuttosto che il separatore fra i campi, si può usare la variabile predefinita `FPAT` per farlo. (Si veda la [Sezione 4.5 \[Specificare come vengono separati i campi\]](#), [pagina 71](#), la [Sezione 4.6 \[Leggere campi di larghezza costante\]](#), [pagina 77](#), e la [Sezione 4.7 \[Definire i campi in base al contenuto\]](#), [pagina 79](#)).

#### Caratteri

L'insieme di codici numerici usati da un computer per rappresentare i caratteri (lettere, numeri, segni d'interpunzione, etc.) di un particolare paese o località. L'insieme di caratteri più comunemente in uso oggi è l'ASCII (American Standard Code for Information Interchange). Molti paesi europei usano un'estensione dell'ASCII nota come ISO-8859-1 (ISO Latin-1). L'insieme di caratteri **Unicode** sta guadagnando popolarità e affermandosi come standard, e il suo uso è particolarmente esteso nei sistemi GNU/Linux.

#### CHEM

Un preprocessore per `pic` che legge descrizioni di molecole e produce l'input a `pic` che serve a disegnarle. È stato scritto in `awk` da Brian Kernighan e Jon Bentley, ed è disponibile in <http://netlib.org/typesetting/chem>.

**Classe di caratteri**

Si veda “Espressione tra parentesi quadre”.

**Compilatore**

Un programma che traduce codici sorgente scritti in qualche linguaggio in codici eseguibili su un particolare computer. Il codice oggetto risultante può quindi essere eseguito direttamente dal computer. Si veda anche “Interprete”.

**Concatenazione**

Concatenare due stringhe significa unirle, producendo una nuova stringa. Per esempio, la stringa ‘**pippo**’ concatenata con la stringa ‘**pluto**’ produce la stringa ‘**pippopluto**’. (Si veda la [Sezione 6.2.2 \[Concatenazione di stringhe\]](#), [pagina 124](#)).

**Contatore di riferimenti**

Un meccanismo interno di **gawk** per minimizzare la quantità di memoria necessaria per contenere il valore delle variabili di tipo stringa. Se il valore assunto da una variabile è usato in più di un posto nel programma, solo una copia del valore stesso è tenuta in memoria, e il contatore di riferimenti ad esso associato è aumentato di uno quando lo stesso valore è usato da un’ulteriore variabile, e diminuito di uno quando la variabile relativa non è più utilizzata. Quando il contatore di riferimenti va a zero, la parte di memoria utilizzata per contenere il valore della variabile è liberato.

**Coprocesso**

Un programma subordinato con il quale è possibile una comunicazione bidirezionale dal programma principale.

**Dati oggetto**

Sono costituiti da numeri e stringhe di caratteri. I numeri sono convertiti in stringhe e viceversa, a seconda delle necessità. (Si veda la [Sezione 6.1.4 \[Conversione di stringhe e numeri\]](#), [pagina 121](#)).

**Debugger** Un programma che serve agli sviluppatori per rimuovere “bug” (de-bug) dai loro programmi.

**Dominio di testo**

Un nome unico che identifica un’applicazione. Usato per raggruppare messaggi che sono tradotti in fase di esecuzione nel linguaggio locale.

**Doppia precisione**

Una rappresentazione di numeri all’interno del computer che ha una parte espressa sotto forma di frazione. I numeri a doppia precisione hanno più cifre decimali che quelli a singola precisione, ma le operazioni che la usano consumano più risorse di quelle eseguite in singola precisione. La doppia precisione è il formato con cui **awk** memorizza i valori numerici. Nel linguaggio C è il tipo di dati detto **double**.

**Editore di flusso**

Un programma che legge record da un flusso in input e li elabora uno o più alla volta. Questo è diverso da quel che farebbe un programma batch il quale potrebbe leggere completamente i file in input, prima di iniziare a fare alcunché,

ed è diverso anche da un programma interattivo, che richiede input dall'utente [tipicamente, una riga alla volta].

#### Effetto collaterale

Un effetto collaterale ha luogo quando un'espressione ha un effetto ulteriore, invece di produrre solo un valore. Espressioni di assegnamento, incremento e decremento, e invocazioni di funzioni hanno effetti collaterali. (Si veda la [Sezione 6.2.3 \[Espressioni di assegnamento\]](#), pagina 126).

#### Epoca [Inizio del tempo in Unix]

la data usata come "inizio del tempo" per i campi che contengono date. I valori del tempo nella maggior parte dei sistemi sono rappresentati in numero di secondi trascorsi dall'Epoca, con funzioni di libreria che consentono di convertire tali valori nei formati normali di data e ora.

L'Epoca nei sistemi Unix e POSIX parte dal primo gennaio 1970 alle ore 00:00:00 UTC. Si veda anche "GMT" e "UTC".

#### Esadecimale

Notazione per l'aritmetica in base 16, che usa le cifre 0–9 e le lettere A–F, con 'A' che rappresenta 10, 'B' che rappresenta 11, e così via, fino a 'F' per 15. I numeri esadecimali sono scritti in C prefissandoli con '0x', per indicarne la base. Quindi, 0x12 è 18 ((1 x 16) + 2). Si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115.

#### Espressione booleana

Così detta dal nome del matematico inglese George Boole. Si veda anche "Espressione logica".

#### Espressione condizionale

Un'espressione che usa l'operatore ternario '?:', come p.es. '*expr1* ? *expr2* : *expr3*'. Dell'espressione *expr1* viene calcolato il valore; se risulta verificata, il valore dell'intera espressione diviene quello di *expr2*; altrimenti il valore è quello di *expr3*. In ogni caso, solo una delle due espressioni *expr2* e *expr3* viene calcolata. (Si veda la [Sezione 6.3.4 \[Espressioni condizionali\]](#), pagina 137).

#### Espressione di confronto

Una relazione che è vera o falsa, del tipo di '*a* < *b*'. Espressioni di confronto sono usate nelle istruzioni *if*, *while*, *do*, *for* e nelle espressioni di ricerca per scegliere quale record in input elaborare. (Si veda la [Sezione 6.3.2 \[Tipi di variabile ed espressioni di confronto\]](#), pagina 130).

#### Espressione di intervallo

Una parte di un'espressione regolare che permette di specificare corrispondenze multiple di qualche parte della *regex*. Le espressioni di intervallo non erano originariamente ammesse nei programmi *awk*.

#### Espressione di ricerca [*pattern*]

(detta anche "criterio di ricerca" o "modello di ricerca")

Le espressioni di ricerca individuano per *awk* a quali record in input sono applicabili determinate regole.

Un'espressione di ricerca [*pattern*] è un'espressione condizionale specifica che viene confrontata con ogni record in input. Se la corrispondenza esiste, si dice

che il modello *individua* il record in input. Una tipica espressione di ricerca potrebbe confrontare il record in input con un'espressione regolare. (Si veda la [Sezione 7.1 \[Elementi di un criterio di ricerca\]](#), pagina 145).

#### Espressione logica

Un'espressione che usa gli operatori logici AND, OR e NOT, scritti come '&&', '||', e '!' in **awk**. Spesso chiamate espressioni booleane, dal nome del matematico che per primo ha sistematizzato questo tipo di logica matematica.

#### Espressione regolare

un'espressione regolare (abbreviabile come "*regex*") è un modello che descrive un insieme di stringhe, potenzialmente illimitato. Per esempio l'espressione regolare '**R.\*xp**' corrisponde a qualsiasi stringa che inizia con la lettera '**R**' e termina con le lettere '**xp**'. In **awk**, le espressioni regolari sono usate nei modelli [pattern] e nelle espressioni condizionali. Le espressioni regolari possono contenere sequenze di protezione. (si veda il [Capitolo 3 \[Espressioni regolari\]](#), pagina 49).

#### Espressione regolare calcolata

Si veda "Espressioni regolari dinamiche".

#### Espressione regolare costante

Un'espressione regolare costante è un'espressione regolare scritta tra barre, come **/pippo/**. A una tale espressione viene assegnato un valore quando si scrive un programma **awk** e non può essere modificata in fase di esecuzione del programma. (Si veda la [Sezione 3.1 \[Uso di espressioni regolari\]](#), pagina 49.)

#### Espressione regolare dinamica

Un'espressione regolare dinamica è un'espressione regolare scritta come un'espressione normale. Potrebbe essere una costante stringa, come **"pippo"**, ma potrebbe anche essere un'espressione il cui valore è variabile (Si veda la [Sezione 3.6 \[Usare regex dinamiche\]](#), pagina 57).

#### Espressione tra parentesi quadre

All'interno di una *espressione regolare*, un'espressione racchiusa fra parentesi quadre sta a indicare che un singolo carattere appartiene a una specifica classe di caratteri. Un'espressione tra parentesi quadre può contenere una lista di uno o più caratteri, come **'[abc]'**, un intervallo di caratteri, come **'[A-Z]'**, o un nome, delimitato da **':'**, che designa un insieme di caratteri conosciuto, come **'[:digit:]'**. La forma di espressione tra parentesi quadre racchiusa tra **':'** è indipendente dalla rappresentazione binaria dei caratteri stessi, che potrebbe utilizzare le codifiche ASCII, EBCDIC, o Unicode, a seconda dell'architettura del computer, e della localizzazione. Si veda anche "Espressioni regolari".

#### Espressione tra parentesi quadre complementata

La negazione di una *espressione tra parentesi quadre*. Tutto ciò che *non* è descritto da una data espressione tra parentesi quadre. Il simbolo **'^'** precede l'espressione tra parentesi quadre che viene negata. Per esempio: **'[^:digit:]'** designa qualsiasi carattere che non sia una cifra. **'[^bad]'** designa qualsiasi carattere che non sia una delle lettere **'b'**, **'a'**, o **'d'**. Si veda "Espressione tra parentesi quadre".

- Estensione** Una funzionalità aggiunta o una modifica a un linguaggio di programmazione o a un programma di utilità, non definita dallo standard di quel linguaggio o di quel programma di utilità. **gawk** ha molte estensioni rispetto al POSIX **awk** (fin troppe).
- FDL** Free Documentation License. Si veda “Licenza Documentazione Libera”.
- File speciale** Un nome-file interpretato internamente da **gawk**, invece che gestito direttamente dal sistema operativo in cui viene eseguito **gawk**—per esempio, `/dev/stderr`. (Si veda la [Sezione 5.8 \[Nomi-file speciali in gawk\]](#), pagina 108).
- Flag [Indicatore]** Una variabile [di tipo booleano] che, se verificata, indica la presenza o l’assenza di qualche condizione.
- Formato** Le stringhe di formato controllano il modo in cui le funzioni `strftime()`, `sprintf()` e l’istruzione `printf` visualizzano l’output che producono. Inoltre, le conversioni da numeri a stringhe sono controllate dalle stringhe di formato contenute nelle variabili predefinite `CONVFMT` e `OFMT`. (Si veda la [Sezione 5.5.2 \[Lettere di controllo del formato\]](#), pagina 99).
- Formattatore incredibilmente duttile (**awf**)** Henry Spencer all’Università di Toronto ha scritto un formattatore che accetta un ampio sottoinsieme dei comandi di formattazione `‘nroff -ms’` e `‘nroff -man’` usando **awk** e **sh**. Si può scaricare da <http://awk.info/?tools/awf>.
- Fortran** Abbreviazione di FORMula TRANslator (traduttore di formule), è uno dei primi linguaggi di programmazione, pensato per il calcolo scientifico. È stato ideato da John Backus ed è disponibile a partire dal 1957. È ancora in uso ai giorni nostri.
- Free Software Foundation** Un’organizzazione senza fini di lucro dedicata alla produzione e distribuzione di software liberamente distribuibile. È stata fondata da Richard M. Stallman, l’autore dell’originale editor Emacs. GNU Emacs è la versione di Emacs maggiormente usata oggi.
- FSF** Si veda “Free Software Foundation”.
- Funzione** Una parte di un programma **awk** che si può chiamare da qualsiasi punto del programma, per eseguire un compito. **awk** ha parecchie funzioni predefinite. Gli utenti possono definire essi stessi delle funzioni in qualsiasi parte del programma. Le funzioni possono essere ricorsive, ossia possono chiamare se stesse. si veda il [Capitolo 9 \[Funzioni\]](#), pagina 195. In **gawk** è anche possibile avere funzioni condivise tra diversi programmi, incluse secondo necessità usando la direttiva `@include` (si veda la [Sezione 2.7 \[Come includere altri file nel proprio programma\]](#), pagina 45). In **gawk** il nome della funzione da chiamare può essere generato in fase di esecuzione, ossia in maniera dinamica. L’estensione API di **gawk** fornisce funzioni di costruzione (si veda la [Sezione 16.4.4 \[Funzioni per creare valori\]](#), pagina 404).

**Funzioni predefinite**

Il linguaggio **awk** fornisce funzioni predefinite, che compiono calcoli vari, di tipo numerico, di input/output e di tipo carattere. Esempi sono **sqrt()** ([square root], la radice quadrata di un numero) e **substr()** (che estrae una sottostringa da una stringa). **gawk** fornisce funzioni per la gestione di data e ora, le operazioni a livello di bit, l'ordinamento di vettori, il controllo di tipo [di variabile] e la traduzione di stringhe in fase di esecuzione di programma. (Si veda la [Sezione 9.1 \[Funzioni predefinite\]](#), pagina 195).

**gawk** L'implementazione GNU di **awk**.

**General Public License**

Un documento che descrive le condizioni alle quali **gawk** e i suoi file sorgenti possono essere distribuiti. (Si veda la [\[Licenza Pubblica Generale GNU \(GPL\)\]](#), pagina 529).

**GMT** “Greenwich Mean Time”. Il termine tradizionalmente usato per UTC. È la datazione usata internamente dai sistemi Unix e POSIX. Si veda anche “Epoca” e “UTC”.

**GNU** “GNU’s not Unix” (GNU non è Unix). Un progetto della Free Software Foundation, ancora in corso, che mira a creare un ambiente di calcolo completo, liberamente distribuibile, aderente allo standard POSIX.

**GNU/Linux**

Una variante del sistema GNU che usa il kernel Linux, invece del kernel proprio della Free Software Foundation, noto come Hurd. Il kernel Linux è un clone di Unix stabile, efficiente, completo di tutte le funzionalità, ed è stato portato su varie architetture hardware. È molto diffuso su sistemi del tipo dei Personal Computer, ma funziona bene anche in parecchi altri computer. Il codice sorgente del kernel Linux è disponibile nei termini della GNU General Public License, la qual cosa è forse il suo aspetto più rilevante.

**GPL** Si veda “General Public License”.

**Graffe** I caratteri ‘{’ e ‘}’. Le parentesi graffe sono usate in **awk** per delimitare azioni, istruzioni composte, e il codice che costituisce le funzioni.

**Guidato dai dati**

Una descrizione dei programmi **awk**, nei quali si specifica quali sono i dati che si vogliono elaborare, e cosa fare quando si trovano tali dati.

**I/O** Abbreviazione per “Input/Output,” ovvero il trasferimento di dati da e verso un programma in esecuzione.

**Individuazione**

L'azione che consiste nel confrontare una stringa con un'espressione regolare. Se la *regex* descrive qualcosa che è contenuto nella stringa, si dice che la *individua*.

**Internazionalizzazione**

La procedura con cui si scrive o si modifica un programma in modo che possa inviare messaggi in lingue differenti, senza richiedere ulteriori modifiche al codice sorgente.

Intero	Un numero intero, cioè un numero che non ha una parte frazionaria.
Interprete	Un programma che accetta come input del codice sorgente, e usa le istruzioni contenute nello stesso per elaborare dati e fornire risultati. <b>awk</b> è tipicamente (ma non sempre) implementato come un interprete. Si veda anche “Compilatore”.
Intervallo (nelle righe di input)	Una sequenza di righe consecutive nel/nei file in input. Un’espressione di ricerca può specificare intervalli di righe di input da far elaborare ad <b>awk</b> oppure può specificare singole righe. (Si veda la <a href="#">Sezione 7.1 [Elementi di un criterio di ricerca]</a> , pagina 145).
ISO	Acronimo di International Organization for Standardization. Questo ente elabora degli standard internazionali in vari settori, inclusi i linguaggi di programmazione, come il C e il C++. In ambito informatico, standard importanti come quelli per il C, C++, e POSIX sono allo stesso tempo standard nazionali americani e standard internazionali ISO. In questo libro lo Standard C è chiamato “ISO C”. Si veda <a href="#">il sito web ISO</a> per ulteriori informazioni sul nome dell’ente e sul suo acronimo di tre lettere, che rimane lo stesso in tutte le lingue.
Istruzione	Un’espressione all’interno di un programma <b>awk</b> nella parte “azione” di una regola <i>criterio di ricerca-azione</i> , o all’interno di una funzione <b>awk</b> . Un’espressione può essere un assegnamento di variabile, un’operazione su un vettore, un ciclo, etc.
Istruzione composta	Una serie di istruzioni <b>awk</b> , racchiuse tra parentesi graffe. Le istruzioni composte possono essere nidificate [possono esserci più livelli di parentesi graffe]. (Si veda la <a href="#">Sezione 7.4 [Istruzioni di controllo nelle azioni]</a> , pagina 153).
Istruzione di controllo	Un’istruzione di controllo è un’istruzione per eseguire una data operazione o un insieme di operazioni all’interno di un programma <b>awk</b> , se una determinata condizione è verificata. Istruzioni di controllo sono: <b>if</b> , <b>for</b> , <b>while</b> , e <b>do</b> (si veda la <a href="#">Sezione 7.4 [Istruzioni di controllo nelle azioni]</a> , pagina 153).
Java	Un moderno linguaggio di programmazione originalmente sviluppato da Sun Microsystems (ora Oracle) che prevede la programmazione orientata agli oggetti. Sebbene normalmente sia implementato compilando le istruzioni per una macchina virtuale standard (la JVM—Java Virtual Machine) il linguaggio può essere compilato per essere eseguito in maniera nativa.
Korn Shell	La Korn Shell ( <b>ksh</b> ) è una shell Unix sviluppata da David Korn, presso i Bell Laboratories, nei primi anni ’80. La Korn shell è compatibile all’indietro con la Bourne shell e comprende molte funzionalità presenti nella C Shell. Si veda anche “Bourne Shell”.
LDL	Si veda “Licenza Documentazione Libera”.

**Lesser General Public License**

Questo documento descrive i termini nei quali possono essere distribuiti degli archivi contenenti librerie in formato eseguibile o oggetti condivisi, e il relativo codice sorgente.

**LGPL** Si veda “Lesser General Public License”.

**Licenza Documentazione Libera**

Questo documento descrive i termini in base ai quali questo libro è pubblicato e può essere copiato. (Si veda la [\[Licenza per Documentazione Libera GNU \(FDL\)\]](#), pagina 543).

**Linguaggio `awk`**

Il linguaggio in cui i programmi `awk` sono scritti.

**Linux** Si veda “GNU/Linux”.

**Lista di caratteri**

Si veda “Espressione tra parentesi quadre”.

**Localizzazioni**

La funzionalità che fornisce i dati necessari perché un programma internazionalizzato interagisca con l’utente in un particolare linguaggio.

***Lvalue*** [left-value, ossia valore a sinistra] Un’espressione che può stare alla sinistra di un operatore di assegnamento. Nella maggior parte dei linguaggi, gli *lvalue* possono essere variabili o elementi di un vettore. In `awk`, un designatore di campo può anche essere usato come un *lvalue*.

**Marcatura temporale**

Un valore nel formato “secondi a partire dall’epoch” usato dai sistemi Unix e POSIX. Usato per le funzioni `gawk mktime()`, `strftime()`, e `systemtime()`. Si veda anche “Epoca,” “GMT,” e “UTC”.

**Metacaratteri**

Caratteri usati all’interno di una *regexp* e che non rappresentano se stessi. Servono invece per rappresentare operazioni con espressioni regolari, come per esempio delle ripetizioni, dei raggruppamenti, o delle alternanze.

**Nidificazione**

Una nidificazione si riscontra dove l’informazione è organizzata a strati, o dove degli oggetti contengono altri oggetti simili. In `gawk` la direttiva `@include` può essere nidificata. La nidificazione “naturale” delle operazioni aritmetiche e logiche può essere modificato attraverso l’uso di parentesi. (si veda la [Sezione 6.5 \[Precedenza degli operatori \(Come si nidificano gli operatori\)\]](#), pagina 140).

**No-op** Un’operazione che non fa nulla.

**Numero** Un dato oggetto il cui valore è numerico. Le implementazioni di `awk` usano numeri a virgola mobile in doppia precisione per rappresentare i numeri. Le primissime implementazioni di `awk` usavano numeri a virgola mobile in singola precisione.

## Numero a virgola mobile

Spesso descritto, in termini matematici, come un numero “razionale” o reale, è soltanto un numero che può avere una parte frazionaria. Si veda anche “Doppia precisione” e “Singola precisione”.

## Operatori di espressioni regolari

Si veda “Metacaratteri”.

## Ottale

Notazione avente come base 8, nella quale le cifre sono 0–7. I numeri ottali in C sono scritti premettendo uno ‘0’, per indicare la base. Quindi, 013 è 11 ( $(1 \times 8) + 3$ ). Si veda la [Sezione 6.1.1.2 \[Numeri ottali ed esadecimali\]](#), pagina 115.

## Parentesi Graffe

Si veda “Graffe”.

## Parola chiave

nel linguaggio `awk`, una parola chiave (keyword) è una parola che ha un significato speciale. Queste parole sono riservate e non possono essere usate come nomi di variabili.

Le parole chiave di `gawk` sono: `BEGIN`, `BEGINFILE`, `END`, `ENDFILE`, `break`, `case`, `continue`, `default`, `delete`, `do...while`, `else`, `exit`, `for...in`, `for`, `function`, `func`, `if`, `next`, `nextfile`, `switch`, e `while`.

## PEBKAC

Un acronimo inglese che descrive qual è probabilmente la causa più frequente di problemi nell’uso di un computer. (*Problem Exists Between Keyboard and Chair* [il problema si trova tra la tastiera e la sedia].)

## Percorso di ricerca

In `gawk`, una lista di directory in cui cercare file contenenti del codice sorgente per `awk`. Nella shell, una lista di directory in cui ricercare un programma eseguibile.

## Plug-in

Si veda “Estensione”.

## POSIX

Il nome di una serie di standard che specificano l’interfaccia di un Sistema Operativo Portabile (Portable Operating System). La “IX” specifica che questi standard sono stati originati dallo Unix. Lo standard più rilevante per gli utenti `awk` è lo *IEEE Standard for Information Technology, Standard 1003.1-2008*. Lo standard POSIX 2008 può essere trovato in rete all’indirizzo: <http://www.opengroup.org/onlinepubs/9699919799/>.

## Precedenza

L’ordine in cui le operazioni sono eseguite quando si usano degli operatori se non si stabiliscono precedenze per mezzo di parentesi.

## Private

Variabili e/o funzioni che sono riservate all’uso esclusivo di funzioni di libreria, e non per il programma principale `awk`. Un’attenzione particolare va prestata quando si designano tali variabili e funzioni. (Si veda la [Sezione 10.1 \[Dare un nome a variabili globali in funzioni di libreria\]](#), pagina 246).

Programma `awk`

Un programma `awk` consiste in una serie di *espressioni di ricerca* e *azioni*, che formano delle *regole*. Per ogni record in input a un programma, le regole del

programma sono elaborate nell'ordine in cui sono scritte. I programmi **awk** possono anche contenere definizioni di funzioni.

**Record** Si veda “Record in input” e “Record in output”.

**Record in input**

Una singola parte di dati letta da **awk**. Solitamente, un record in input di **awk** consiste in una linea di testo. (Si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63).

**Record in output**

Un singolo pezzo di dati scritto da **awk**. Solitamente, un record in output di **awk** consiste di una o più righe di testo. Si veda la [Sezione 4.1 \[Controllare come i dati sono suddivisi in record\]](#), pagina 63.

**Ricorsione** Quando una funzione chiama se stessa, direttamente o indirettamente. Se questo è chiaro, si può passare a leggere la definizione successiva. Altrimenti, si veda la voce “Ricorsione”.

**regex** Si veda “Espressione regolare”.

**Regola** Un segmento di un programma **awk** che specifica come trattare singoli record in input. Una regola consiste in una *espressione di ricerca* e in una *azione*. **awk** legge un record in input; poi, per ogni regola, se il record in input soddisfa l'espressione di ricerca della regola, **awk** esegue l'azione specificata dalla regola. Altrimenti, la regola non ha alcun effetto su quel record in input.

**Ridirezione**

Ridirezione significa ricevere input da qualcosa che non sia il flusso dello standard input, o dirigere output a qualcosa di diverso dal flusso dello standard output.

Si può ridirigere input all'istruzione **getline** usando gli operatori '<', '|', e '|&'. Si può ridirigere l'output delle istruzioni **print** e **printf** verso un file o un comando di sistema, usando gli operatori '>', '>>', '|', e '|&'. (Si veda la [Sezione 4.9 \[Richiedere input usando getline\]](#), pagina 83, e [Sezione 5.6 \[Ridirigere l'output di print e printf\]](#), pagina 104).

**Rvalue** [right-value, ossia valore a destra] Un valore che può apparire alla destra di un operatore di assegnazione. In **awk**, essenzialmente ogni espressione ha un valore. Ognuno di questi valori è un *rvalue*.

**Scalare** Un valore singolo, sia numerico che di tipo stringa. Le variabili normali sono scalari; i vettori e le funzioni non lo sono.

**Scorciatoia**

La natura degli operatori logici **awk** '&&' e '||'. Se il valore dell'intera espressione in cui sono contenuti è determinabile valutando solo una parte iniziale dell'espressione, la parte seguente non è presa in considerazione. (Si veda la [Sezione 6.3.3 \[Espressioni booleane\]](#), pagina 136).

**Script awk** Un altro nome per designare un programma **awk**.

**sed** Si veda “Editore di flusso”.

Seme	Il valore iniziale, o il punto di partenza, di una sequenza di numeri casuali.
Sequenze di protezione	Una speciale sequenza di caratteri usata per descrivere caratteri non stampabili, come ‘\n’ (ritorno a capo) o ‘\033’ per il carattere ASCII ESC (Escape). (Si veda la <a href="#">Sezione 3.2 [Sequenze di protezione]</a> , <a href="#">pagina 50</a> ).
Shell	Il programma che interpreta i comandi nei sistemi Unix e in quelli che rispettano lo standard POSIX. La shell funziona sia interattivamente che come un linguaggio di programmazione, che elabora file sequenziali, detti <i>script</i> di shell.
Singola precisione	Una rappresentazione di numeri all’interno del computer che ha una parte espressa sotto forma di frazione. I numeri a singola precisione hanno meno cifre significative di quelli a doppia precisione, ma le operazioni relative richiedono talora meno risorse elaborative da parte del computer. Questo tipo di numero è quello usato da alcune tra le prime versioni di <b>awk</b> per memorizzare valori numerici. Nel linguaggio C, sono numeri di tipo <b>float</b> .
Spazio	Il carattere generato premendo la barra spaziatrice sulla tastiera.
Spazio vuoto	Una sequenza di spazi, TAB, o caratteri di ritorno a capo presenti in un record in input o in una stringa.
Stringa	Un dato che consiste in una sequenza di caratteri, come ‘Io sono una <b>stringa</b> ’. Le costanti stringa sono scritte tra doppi apici nel linguaggio <b>awk</b> e possono contenere sequenze di protezione (Si veda la <a href="#">Sezione 3.2 [Sequenze di protezione]</a> , <a href="#">pagina 50</a> ).
Stringa nulla	Una stringa che non contiene alcun carattere. È rappresentabile esplicitamente nei programmi <b>awk</b> mettendo due caratteri di doppio apice uno dietro all’altro (“”). La si può inserire nei dati in input mettendo due separatori di campo uno dietro all’altro.
Stringa vuota	Si veda “Stringa nulla”.
Tab	Il carattere generato premendo il tasto <b>TAB</b> sulla tastiera. Normalmente può generare sino a otto spazi in output.
Unix	Un sistema operativo per computer originalmente sviluppato nei primi anni ’70 presso gli AT&T Bell Laboratories. Inizialmente si diffuse nelle università di tutto il mondo e in seguito si estese agli ambienti del mondo del lavoro come un sistema per lo sviluppo del software e come server di rete. Ci sono parecchie versioni di Unix a pagamento, come pure parecchi sistemi operativi modellati su Unix e il cui codice sorgente è liberamente disponibile. (come GNU/Linux, <b>NetBSD</b> , <b>FreeBSD</b> , e <b>OpenBSD</b> ).
UTC	L’abbreviazione comune per “Universal Coordinated Time” (tempo coordinato universale). Questa è l’ora standard di Greenwich, (UK), usata come tempo di riferimento per i calcoli relativi a marcature temporali. Si veda anche “Epoca” e “GMT”.

**Variabile** Un nome per designare un valore. In **awk**, le variabili possono essere degli scalari o dei vettori.

**Variabili d'ambiente**

Una collezione di stringhe, in formato '*nome=valore*', che ogni programma ha a disposizione. Gli utenti in generale assegnano valori alle variabili d'ambiente per fornire informazioni a vari programmi. Esempi tipici sono le variabili d'ambiente HOME e PATH.

**Variabili predefinite**

ARGC, ARGV, CONVFMT, ENVIRON, FILENAME, FNR, FS, NF, NR, OFMT, OFS, ORS, RLENGTH, RSTART, RS, e SUBSEP sono le variabili con un significato speciale in **awk**. In più, ARGIND, BINMODE, ERRNO, FIELDWIDTHS, FPAT, IGNORECASE, LINT, PROCINFO, RT, e TEXTDOMAIN sono le variabili con un significato speciale in **gawk**. Se i loro valori sono modificati, il contesto di esecuzione di **awk** cambia. (Si veda la [Sezione 7.5 \[Variabili predefinite\], pagina 162](#)).

**Vettore** Un raggruppamento di molti valori con uno stesso nome. La maggior parte dei linguaggi fornisce solo vettori sequenziali. **awk** fornisce vettori associativi.

**Vettore associativo**

Un vettore i cui indici possono essere numeri o stringhe, e non solamente interi sequenziali compresi in un intervallo prestabilito.

# Licenza Pubblica Generale GNU (GPL)

Versione 3, 29 Giugno 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

This is an unofficial translation of the GNU General Public License into Italian. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU GPL—only the original English text of the GNU GPL does that. However, we hope that this translation will help Italian speakers understand the GNU GPL better.

Questa è una traduzione non ufficiale in italiano della GNU General Public License. Questa traduzione non è stata pubblicata dalla Free Software Foundation, e non stabilisce i termini legali di distribuzione del software che usa la GNU GPL. Soltanto la versione originale in inglese della GNU GPL fa ciò. Ciononostante, speriamo che questa traduzione possa aiutare gli utenti di lingua italiana a comprendere un po' meglio la GNU GPL.

A chiunque è permesso copiare e ridistribuire copie esatte di questo documento di licenza, ma non è in alcun modo consentito apportarvi modifiche.

## Preambolo

La GNU General Public License è una licenza libera e basata su copyleft per software e altri tipi di opere.

Le licenze della maggior parte del software e di altre opere materiali sono pensate per togliere la libertà di condividere e modificare tali opere. Al contrario, la GNU General Public License ha l'obiettivo di garantire la libertà di condividere e modificare tutte le versioni di un programma e di fare in modo che esso rimanga software libero per tutti gli utenti. Noi, Free Software Foundation, usiamo la GNU General Public License per la maggior parte del nostro software; essa viene applicata anche a qualunque altro software rilasciato dall'autore sotto questa licenza. Chiunque può utilizzare questa licenza per i suoi programmi.

Quando parliamo di software libero (free software), ci riferiamo al concetto di libertà, non al prezzo. Le nostre General Public License sono progettate per garantire che chiunque abbia la libertà di distribuire copie di software libero (anche dietro pagamento di un prezzo, se lo desidera), che chiunque riceva o possa ricevere il codice sorgente se lo vuole, che chiunque possa apportare modifiche al software o utilizzarne delle porzioni in altri software liberi, e che chiunque sappia che ha il diritto di fare tutte queste cose col software libero.

Per proteggere i vostri diritti, abbiamo la necessità di impedire che altri vi neghino questi diritti o vi obblighino a rinunciarvi. Pertanto, chiunque distribuisce o modifica software rilasciato con questa licenza assume dei precisi doveri: il dovere di rispettare la libertà degli altri.

Per esempio, chi distribuisce copie di un programma rilasciato sotto questa licenza, sia gratis che dietro pagamento di un prezzo, è obbligato a riconoscere a chi riceve il software esattamente gli stessi diritti che ha ricevuto. Deve garantire che chi riceva il software abbia o possa avere accesso al codice sorgente. E deve chiaramente far conoscere ai destinatari del software queste condizioni, così che essi conoscano quali sono i loro diritti.

Gli sviluppatori che usano la GNU GPL proteggono i vostri diritti in due modi: (1) Rivendicando il copyright sul software, e (2) offrendovi questa licenza che vi garantisce il diritto legale di copiarlo e/o di modificarlo.

Al fine di proteggere gli sviluppatori e gli autori, la GPL spiega chiaramente che non c'è nessuna garanzia per questo software libero. Nell'interesse degli utenti e degli autori, la GPL impone che le versioni modificate del software vengano esplicitamente marcate come "modificate", in maniera tale che eventuali problemi non vengano erroneamente attribuiti agli autori delle versioni precedenti.

Alcuni dispositivi sono progettati per negare agli utenti l'installazione o l'esecuzione di versioni modificate del software che gira sugli stessi, anche se il costruttore si riserva la possibilità di farlo. Ciò è fondamentalmente incompatibile con l'obiettivo di garantire la libertà degli utenti di modificare il software. Una ripetizione sistematica di tali abusi avviene nel campo dei dispositivi per usi individuali, e ciò rende questi abusi ancora più inaccettabili. Pertanto, abbiamo realizzato questa versione della GPL al fine di proibire queste pratiche. Se problemi simili dovessero sorgere in altri ambiti, saremo pronti ad estendere queste misure a questi nuovi ambiti in versioni future della GPL, nella maniera che si renderà necessaria per difendere la libertà degli utenti.

In conclusione, tutti i programmi sono costantemente minacciati dai brevetti sul software. Gli Stati non dovrebbero permettere ai brevetti sul software di limitare lo sviluppo e l'utilizzo di software per computer, ma nei Paesi in cui ciò avviene noi vogliamo evitare in particolare il pericolo che i brevetti sul software applicati ad un programma libero possano renderlo, a tutti gli effetti, proprietario. Per impedire ciò, la GPL assicura che non è possibile utilizzare i brevetti sul software per rendere un programma non libero.

I termini e le condizioni esatte per la copia, la distribuzione e la modifica del software sono riportate di seguito.

## TERMINI E CONDIZIONI

### 0. Definizioni

"Questa Licenza" si riferisce alla versione 3 della GNU General Public License.

"Copyright" indica anche leggi simili al copyright che riguardano altri tipi di opere, come le maschere per la produzione di semiconduttori.

"Il Programma" indica qualunque opera che sia soggetta a copyright e che sia rilasciata sotto questa Licenza. I detentori della licenza sono indicati come "tu" o "voi". Licenziatari e destinatari possono essere individui o organizzazioni.

"Modificare" un'opera significa copiare o adattare tutta o parte dell'opera in una maniera che richieda un permesso di copyright, e non indica la semplice azione di fare una esatta copia dell'opera. L'opera risultante viene chiamata "versione modificata" dell'opera precedente, oppure viene detta opera "basata sulla" opera precedente.

Una "opera coperta da questa licenza" indica il Programma originale non modificato oppure un'opera basata sul Programma.

"Propagare" un'opera significa fare qualunque cosa con essa che, in mancanza di un esplicito permesso, ti renda direttamente o indirettamente perseguibile per violazione secondo le vigenti normative sul copyright, ad eccezione della semplice esecuzione del Programma su un computer o della modifica di una copia privata. La Propagazione

include la copia, la distribuzione (con o senza modifiche), la messa a disposizione al pubblico e, in alcuni stati, altre attività simili e connesse.

“Distribuire” un’opera indica qualunque forma di propagazione che permetta a terze parti di effettuare o ricevere delle copie. La mera interazione con un utente attraverso una rete di computer, senza che ci sia alcun trasferimento di una copia, non è considerata “Distribuzione”.

Una interfaccia utente interattiva fornisce delle “Adeguate Informazioni Legali” soltanto nel caso in cui include una apposita funzionalità, resa adeguatamente visibile, che (1) visualizzi un’adeguata informazione di copyright, e (2) informi l’utente che non c’è alcuna garanzia sull’opera (eccetto nel caso in cui delle garanzie sono espressamente fornite), dica che il licenziatario può distribuire l’opera utilizzando questa Licenza, indichi come è possibile prendere visione di una copia di questa Licenza. Se l’interfaccia presenta una lista di comandi o di opzioni, come per esempio un menù, una delle opzioni fornite nella lista deve rispettare questa condizione.

## 1. Codice Sorgente

Il “codice sorgente” di un’opera indica la forma più indicata dell’opera per effettuare modifiche su di essa. Il “codice oggetto” indica qualunque forma dell’opera che non sia codice sorgente.

Una “Interfaccia Standard” è una interfaccia che risponde ad uno standard ufficiale definito da un ente di standardizzazione riconosciuto o, nel caso di interfacce specifiche per un particolare linguaggio di programmazione, una interfaccia che è largamente utilizzata dagli sviluppatori per sviluppare in tale linguaggio.

Le “Librerie di Sistema” di un eseguibile includono qualsiasi cosa, eccetto l’opera nel suo insieme, che (a) sia inclusa nella normale forma di pacchettizzazione di un “Componente Principale”, ma che non è parte di quel Componente Principale, e (b) che serva solo a consentire l’uso dell’opera con quel Componente Principale, o per implementare una Interfaccia Standard per la quale esista una implementazione disponibile al pubblico in forma sorgente. Un “Componente Principale”, in questo contesto, è un componente essenziale (kernel, gestore di finestre eccetera) dello specifico sistema operativo (ammesso che ce ne sia uno) sul quale l’eseguibile esegue, o un compilatore utilizzato per produrre il programma, o un interprete di codice oggetto utilizzato per eseguire il programma.

Il “Sorgente Corrispondente” per un’opera in forma di codice oggetto è il codice sorgente necessario per generare, installare e (per un programma eseguibile) eseguire il codice oggetto e per modificare l’opera, inclusi gli script per controllare le suddette attività di generazione, installazione ed esecuzione. Non sono incluse le Librerie di Sistema usate dal programma, o gli strumenti di utilità generica o i programmi liberamente accessibili che sono utilizzati, senza modifiche, per portare a termine le suddette attività ma che non fanno parte dell’opera. Per esempio, il sorgente corrispondente include i file con le definizioni delle interfacce associati ai file sorgente dell’opera, e il codice sorgente delle librerie condivise e sottoprogrammi collegati dinamicamente specificatamente necessari per il programma, ad esempio a causa di stretta comunicazione dati o di controllo di flusso tra questi sottoprogrammi e altre parti del programma.

Il Sorgente Corrispondente non include qualunque cosa che l’utente possa rigenerare automaticamente da altre parti del Sorgente Corrispondente stesso.

Il Sorgente Corrispondente di un'opera in forma di codice sorgente è l'opera stessa.

## 2. Principali Diritti

Tutti i diritti garantiti da questa Licenza sono garantiti per la durata del copyright sul Programma, e sono irrevocabili ammesso che le suddette condizioni siano rispettate. Questa Licenza afferma esplicitamente il tuo permesso illimitato di eseguire il Programma non modificato. Il risultato dell'esecuzione di un programma coperto da questa Licenza è a sua volta coperto da questa Licenza solo se il risultato stesso, a causa del suo contenuto, è un'opera coperta da questa Licenza. Questa Licenza riconosce il tuo diritto all'uso legittimo o altri diritti equivalenti, come stabilito dalla legislazione sul copyright.

Puoi creare, eseguire e propagare programmi coperti da questa Licenza che tu non distribuisi, senza alcuna condizione fino a quando la tua Licenza rimane valida. Puoi distribuire opere coperte da questa Licenza ad altri al solo scopo di ottenere che essi facciano delle modifiche al programma esclusivamente per te, o che ti forniscano dei servizi per l'esecuzione di queste opere, ammesso che tu rispetti i termini di questa Licenza nel distribuire tutto il materiale per il quale non detieni il copyright. Coloro i quali creano o eseguono per conto tuo un programma coperto da questa Licenza lo fanno esclusivamente in tua vece, sotto la tua direzione e il tuo controllo, in maniera tale che sia proibito a costoro effettuare copie di materiale di cui detieni il copyright al di fuori della relazione che intrattengono nei tuoi confronti.

Distribuire opere coperte da licenza in qualunque altra circostanza è consentito soltanto alle condizioni espresse in seguito. Non è consentito sottolicensing le opere: la sezione 10 lo rende non necessario.

## 3. Protezione dei diritti legali degli utenti dalle leggi anti-elusione

Nessun programma protetto da questa Licenza può essere considerato parte di una misura tecnologica di restrizione che sottosta ad alcuna delle leggi che soddisfano l'articolo 11 del "WIPO copyright treaty" adottato il 20 Dicembre 1996, o a simili leggi che proibiscono o limitano l'elusione di tali misure tecnologiche di restrizione.

Quando distribuisi un programma coperto da questa Licenza, rifiuti tutti i poteri legali atti a proibire l'elusione di misure tecnologiche di restrizione ammesso che tale elusione sia effettuata nell'esercizio dei diritti garantiti da questa Licenza riguardo al programma coperto da questa Licenza, e rinunci all'intenzione di limitare l'operatività o la modifica del programma per far valere, contro i diritti degli utenti del programma, diritti legali tuoi o di terze parti che impediscano l'elusione di misure tecnologiche di restrizione.

## 4. Distribuzione di Copie Esatte

Ti è permesso distribuire copie esatte del codice sorgente del Programma come lo hai ricevuto, con qualunque mezzo, ammesso che tu aggiunga in maniera appropriata su ciascuna copia una appropriata nota di copyright; che tu lasci intatti tutti gli avvisi che affermano che questa Licenza e tutte le clausole non-permissive aggiunte in accordo con la sezione 7 sono valide per il codice che distribuisi; che tu lasci intatti tutti gli avvisi circa l'assenza di garanzia; che tu fornisca a tutti i destinatari una copia di questa Licenza assieme al Programma.

Puoi richiedere il pagamento di un prezzo o di nessun prezzo per ciascuna copia che distribuisi, e puoi offrire supporto o garanzia a pagamento.

## 5. Distribuzione di Versioni modificate del sorgente

Puoi distribuire un'opera basata sul Programma, o le modifiche per produrla a partire dal Programma, nella forma di codice sorgente secondo i termini della sezione 4, ammesso che tu rispetti anche tutte le seguenti condizioni:

- a. L'opera deve recare con sè delle informazioni adeguate che affermino che tu l'hai modificata, indicando la data di modifica.
- b. L'opera deve recare informazioni adeguate che affermino che essa è rilasciata sotto questa Licenza e sotto le condizioni aggiuntive secondo quanto indicato dalla Sezione 7. Questa condizione modifica la condizione espressa alla sezione 4 di "lasciare intatti tutti gli avvisi".
- c. Devi rilasciare l'intera opera, nel suo complesso, sotto questa Licenza a chiunque venga in possesso di una copia di essa. Questa Licenza sarà pertanto applicata, assieme ad eventuali clausole aggiunte in osservanza della Sezione 7, all'opera nel suo complesso, a tutte le sue parti, indipendentemente da come esse siano pacchettizzate. Questa Licenza nega il permesso di licenziare l'opera in qualunque altro modo, ma non rende nullo un tale permesso ammesso che tu lo abbia ricevuto separatamente.
- d. Se l'opera ha delle interfacce utente interattive, ciascuna deve mostrare delle Adeguate Informazioni Legali; altrimenti, se il Programma ha delle interfacce interattive che non visualizzano delle Adeguate Informazioni Legali, il tuo programma non è obbligato a visualizzarle.

La giustapposizione di un'opera coperta da questa Licenza assieme ad altre opere separate e indipendenti, che non sono per loro natura estensioni del Programma, e che non sono combinate con esso a formare un altro programma più grande, dentro o in uno stesso supporto di memorizzazione a lungo termine o di distribuzione, è semplicemente detto "aggregato" se la raccolta e il suo copyright non sono utilizzati per limitare l'accesso o i diritti legali degli utenti della raccolta stessa oltre ciò che ciascun singolo programma consente. L'inclusione di un programma coperto da questa Licenza in un aggregato non comporta l'applicazione di questa Licenza alle altre parti dell'aggregato.

## 6. Distribuzione in formato non-sorgente

Puoi distribuire un programma coperto da questa Licenza in formato di codice oggetto secondo i termini delle sezioni 4 e 5, ammesso che tu fornisca anche il Sorgente Corrispondente in formato comprensibile da un computer sotto i termini di questa stessa Licenza, in uno dei seguenti modi:

- a. Distribuendo il codice oggetto in, o contenuto in, un prodotto fisico (inclusi i mezzi fisici di distribuzione), accompagnato dal Sorgente Corrispondente su un supporto fisico duraturo comunemente utilizzato per lo scambio di software.
- b. Distribuendo il codice oggetto in, o contenuto in, un prodotto fisico (inclusi i mezzi fisici di distribuzione), accompagnato da un'offerta scritta, valida per almeno tre anni e valida per tutto il tempo durante il quale tu offri ricambi o supporto per quel modello di prodotto, di fornire a chiunque possieda il codice oggetto (1) una copia del Sorgente Corrispondente di tutto il software contenuto nel prodotto che è coperto da questa Licenza, su un supporto fisico duraturo comunemente utilizzato per lo scambio di software, ad un prezzo non superiore al costo ragionevole per effettuare fisicamente tale distribuzione del sorgente, oppure (2) accesso alla

copia del Sorgente Corrispondente attraverso un server di rete senza alcun costo aggiuntivo.

- c. Distribuendo copie singole del codice oggetto assieme ad una copia dell'offerta scritta di fornire il Sorgente Corrispondente. Questa possibilità è permessa soltanto occasionalmente e per fini non commerciali, e solo se tu hai ricevuto il codice oggetto assieme ad una tale offerta, in accordo alla sezione 6b.
- d. Distribuendo il codice oggetto mediante accesso da un luogo designato (gratis o dietro pagamento di un prezzo), e offrendo un accesso equivalente al Sorgente Corrispondente alla stessa maniera a partire dallo stesso luogo senza costi aggiuntivi. Non devi obbligare i destinatari a copiare il Sorgente Corrispondente assieme al codice oggetto. Se il luogo dal quale copiare il codice oggetto è un server di rete, il Sorgente Corrispondente può trovarsi su un server differente (gestito da te o da terze parti) che fornisca funzionalità equivalenti per la copia, a patto che tu fornisca delle indicazioni chiare accanto al codice oggetto che indichino dove trovare il Sorgente Corrispondente. Indipendentemente da quale server ospiti il Sorgente Corrispondente, tu rimani obbligato ad assicurare che esso rimanga disponibile per tutto il tempo necessario a soddisfare queste condizioni.
- e. Distribuendo il codice oggetto mediante trasmissione peer-to-peer, a patto che tu informi gli altri peer circa il luogo in cui il codice oggetto e il Sorgente Corrispondente sono gratuitamente offerti al pubblico secondo i termini della sezione 6d.

Una porzione separabile del codice oggetto, il cui sorgente è escluso dal Sorgente Corrispondente e trattato come Libreria di Sistema, non deve essere obbligatoriamente inclusa nella distribuzione del codice oggetto del programma.

Un "Prodotto Utente" è un (1) "prodotto consumer", cioè qualunque proprietà personale tangibile che è normalmente utilizzata per scopi personali, familiari o domestici, oppure (2) qualunque cosa progettata o venduta per essere utilizzata in ambiente domestico. Nella classificazione di un prodotto come "prodotto consumer", i casi dubbi andranno risolti in favore dell'ambito di applicazione. Per un dato prodotto ricevuto da un dato utente, "normalmente utilizzato" si riferisce ad un uso tipico o comune di quella classe di prodotti, indipendentemente dallo stato dell'utente specifico o dal modo in cui l'utente specifico utilizza, o si aspetta o ci si aspetta che utilizzi, il prodotto. Un prodotto è un "prodotto consumer" indipendentemente dal fatto che abbia usi commerciali, industriali o diversi da quelli "consumer", a meno che questi usi non rappresentino il solo modo utile di utilizzare il prodotto in questione.

Le "Informazioni di Installazione" per un Prodotto Utente sono i metodi, le procedure, le chiavi di autorizzazioni o altre informazioni necessarie per installare ed eseguire versioni modificate di un programma coperto da questa Licenza all'interno di un Prodotto Utente, a partire da versioni modificate dei suoi Sorgenti Corrispondenti. Tali informazioni devono essere sufficienti ad assicurare che il funzionamento del codice oggetto modificato non sia in nessun caso proibito o ostacolato per il solo fatto che sono state apportate delle modifiche.

Se distribuisce un codice oggetto secondo le condizioni di questa sezione in, o assieme, o specificatamente per l'uso in o con un Prodotto Utente, e la distribuzione avviene come parte di una transazione nella quale il diritto di possesso e di uso del Prodotto

Utente viene trasferito al destinatario per sempre o per un periodo prefissato (indipendentemente da come la transazione sia caratterizzata), il Sorgente Corrispondente distribuito secondo le condizioni di questa sezione deve essere accompagnato dalle Informazioni di Installazione. Questa condizione non è richiesta se né tu né una terza parte ha la possibilità di installare versioni modificate del codice oggetto sul Prodotto Utente (per esempio, se il programma è installato su una ROM)

La condizione che richiede di fornire delle Informazioni di Installazione non implica che venga fornito supporto, garanzia o aggiornamenti per un programma che è stato modificato o installato dal destinatario, o per il Prodotto Utente in cui esso è stato modificato o installato. L'accesso ad una rete può essere negato se le modifiche apportate impattano materialmente sull'operatività della rete o se violano le regole e i protocolli di comunicazione attraverso la rete.

Il Sorgente Corrispondente distribuito, e le Informazioni di Installazione fornite, in accordo con questa sezione, devono essere in un formato che sia pubblicamente documentato (e con una implementazione pubblicamente disponibile in formato di codice sorgente), e non devono richiedere speciali password o chiavi per essere spaccettate, lette o copiate.

## 7. Condizioni Aggiuntive

Le “Condizioni Aggiuntive” sono condizioni che completano le condizioni di questa Licenza permettendo delle eccezioni a una o più delle condizioni sopra elencate. Le condizioni aggiuntive che sono applicabili all'intero Programma devono essere considerate come se fossero incluse in questa Licenza, a patto che esse siano valide secondo le normative vigenti. Se alcune condizioni aggiuntive fanno riferimento soltanto ad alcune parti del Programma, quelle parti possono essere utilizzate separatamente sotto le stesse condizioni, ma l'intero Programma rimane sottoposto a questa Licenza senza riferimento ad alcuna condizione aggiuntiva.

Quando distribuisce una copia di un programma coperto da questa Licenza, puoi, a tua discrezione, eliminare qualunque condizione aggiuntiva dalla copia, o da parte di essa. (Le Condizioni Aggiuntive possono essere scritte in maniera tale da richiedere la loro rimozione in certi casi di modifica del Programma). Puoi aggiungere Condizioni Aggiuntive su materiale, aggiunto da te ad un'opera coperta da questa Licenza, per il quale hai o puoi garantire un'adeguata licenza di copyright.

Indipendentemente da qualunque altra condizione di questa Licenza, e per il materiale che aggiungi ad un'opera coperta da questa Licenza, puoi (se autorizzato dai legittimi detentori del copyright per il suddetto materiale) aggiungere alle condizioni di questa Licenza delle condizioni che:

- a. Negano la garanzia o limitano la responsabilità del Programma in maniera differente da quanto riportato nelle sezioni 15 e 16 di questa Licenza; oppure
- b. Richiedono il mantenimento di specifiche e circostanziate informative legali o di note di attribuzione ad autori nel materiale o assieme alle Adeguate Informazioni Legali mostrate dal Programma che lo contiene; oppure
- c. Proibiscono di fornire informazioni errate o ingannevoli sull'origine e la provenienza del materiale in oggetto, o richiedono che versioni modificate di tale materiale siano appositamente marcate in maniera differente rispetto alla versione originale; oppure

- d. Limitano l'utilizzo per scopi pubblicitari del nome dei detentori del copyright o degli autori del materiale; oppure
- e. Rifiutano di garantire diritti secondo le leggi sulla proprietà intellettuale circa l'uso di nomi, marchi di fabbrica o simili; oppure
- f. Richiedono l'indennizzo dei detentori del copyright o degli autori del materiale in oggetto da parte di chi distribuisce il materiale (o versioni modificate dello stesso) con impegni contrattuali circa la responsabilità nei confronti del destinatario, per qualunque responsabilità che questi impegni contrattuali dovessero imporre direttamente ai suddetti detentori del copyright e autori.

Tutte le altre condizioni addizionali non-permissive sono considerate “ulteriori restrizioni”, secondo il significato specificato alla sezione 10. Se il Programma o parti di esso contengono, all'atto della ricezione dello stesso, informative che specificano che esso è soggetto a questa Licenza assieme ad una condizione che è una “ulteriore restrizione”, puoi rimuovere quest'ultima condizione. Se un documento di licenza contiene ulteriori restrizioni ma permette di rilicenziare o distribuire il Programma con questa Licenza, puoi aggiungere al Programma del materiale coperto dalle condizioni di quel documento di licenza, a patto che le ulteriori restrizioni non compaiano nelle versioni rilicenziate o ridistribuite.

Se aggiungi ad un Programma coperto da questa Licenza delle condizioni aggiuntive in accordo con questa sezione, devi aggiungere anche, nei file sorgenti corrispondenti, un avviso che riassume le condizioni aggiuntive applicate a quei file, ovvero un avviso che specifichi dove è possibile trovare copia delle condizioni aggiunte.

Tutte le Condizioni aggiuntive, permissive o non-permissive, devono essere espresse nella forma di una licenza scritta e separata, o espresse esplicitamente come eccezioni; in entrambi i casi valgono le condizioni succitate.

## 8. Cessazione di Licenza

Non puoi propagare o modificare un programma coperto da questa Licenza in maniera diversa da quanto espressamente consentito da questa Licenza. Qualunque tentativo di propagare o modificare altrimenti il Programma è nullo, e provoca l'immediata cessazione dei diritti garantiti da questa Licenza (compresi tutte le eventuali licenze di brevetto garantite ai sensi del terzo paragrafo della sezione 11).

In ogni caso, se cessano tutte le violazioni di questa Licenza, allora la tua licenza da parte di un dato detentore del copyright viene ripristinata (a) in via cautelativa, a meno che e fino a quando il detentore del copyright non cessa esplicitamente e definitivamente la tua licenza, e (b) in via permanente se il detentore del copyright non ti notifica in alcun modo la violazione entro 60 giorni dalla cessazione della licenza.

Inoltre, la tua licenza da parte di un dato detentore del copyright viene ripristinata in maniera permanente se il detentore del copyright ti notifica la violazione in maniera adeguata, se questa è la prima volta che ricevi una notifica di violazione di questa Licenza (per qualunque Programma) dallo stesso detentore di copyright, e se rimedi alla violazione entro 30 giorni dalla data di ricezione della notifica di violazione.

La cessazione dei tuoi diritti come specificato in questa sezione non provoca la cessazione delle licenze di terze parti che abbiano ricevuto copie o diritti da te secondo questa Licenza. Se i tuoi diritti cessano e non sono ristabiliti in via permanente, non hai

diritto di ricevere nuove licenze per lo stesso materiale, secondo quanto stabilito nella sezione 10.

9. L'ottenimento di copie non richiede l'accettazione della Licenza

Non sei obbligato ad accettare i termini di questa Licenza al solo fine di ottenere o eseguire una copia del Programma. Similmente, propagazioni collaterali di un Programma coperto da questa Licenza che occorrono come semplice conseguenza dell'utilizzo di trasmissioni peer-to-peer per la ricezione di una copia non richiedono l'accettazione della Licenza. In ogni caso, solo e soltanto questa Licenza ti garantiscono il permesso di propagare e modificare qualunque programma coperto da questa Licenza. Queste azioni violano le leggi sul copyright nel caso in cui tu non accetti questa Licenza. Pertanto, modificando o propagando un programma coperto da questa Licenza, indichi implicitamente la tua accettazione della Licenza.

10. Licenza Automatica per i successivi destinatari

Ogni qual volta distribuisce un programma coperto da questa Licenza, il destinatario riceve automaticamente una licenza, dal detentore originario del copyright, di eseguire, modificare e propagare il programma, nel rispetto di questa Licenza. Non sei ritenuto responsabile del rispetto di questa Licenza da parte di terze parti.

Una "transazione d'entità" è una transazione che trasferisce il controllo di una organizzazione, o sostanzialmente di tutti i suoi beni, che suddivide una organizzazione o che fonde più organizzazioni. Se la propagazione di un programma coperto da questa Licenza è conseguente ad una transazione di entità, ciascuna parte che ha ruolo nella transazione e che riceve una copia del programma riceve allo stesso tempo qualsiasi licenza sul programma che i predecessori della parte possedevano o potevano rilasciare nel rispetto del paragrafo precedente, e in più il diritto di possesso del Sorgente Corrispondente del programma dal predecessore in interesse, se il predecessore lo possiede o se può ottenerlo senza troppe difficoltà.

Non puoi imporre nessuna ulteriore restrizione sull'esercizio dei diritti garantiti o affermati da questa Licenza. Per esempio, non puoi imporre un prezzo di licenza, una royalty, o altri costi per l'esercizio dei diritti garantiti da questa Licenza, a non puoi dar corso ad una controversia (ivi incluse le controversie incrociate o la difesa in cause legali) affermando che siano stati violati dei brevetti a causa della produzione, dell'uso, della vendita, della messa in vendita o dell'importazione del Programma o di sue parti.

11. Brevetti

Un "contribuente" è un detentore di copyright che autorizza l'uso secondo questa Licenza di un Programma o di un'opera basata sul Programma. L'opera così licenziata viene chiamata "versione del contribuente".

I "diritti essenziali di brevetto" da parte di un contribuente sono tutti i diritti di brevetto che appartengono o che sono controllati dal contribuente, che siano già acquisiti o che saranno acquisiti in futuro, che possano essere violati in qualche maniera, consentita da questa Licenza, generando, modificando o vendendo la versione del contribuente, ma non includono i diritti che possano essere violati soltanto come conseguenza di ulteriori modifiche alla versione del contribuente. In relazione a questa definizione, il termine "controllo" include il diritto di garantire sottolicenze di brevetto in maniera consistente con le condizioni di questa Licenza.

Ciascun contribuente ti garantisce la licenza di brevetto sui diritti essenziali di brevetto del contribuente stesso non-esclusiva, valida in tutto il mondo, esente da royalty, di creare, usare, vendere, offrire in vendita, importare e altrimenti eseguire, modificare e propagare i contenuti della versione del contribuente.

Nei tre paragrafi successivi, con “licenza di brevetto” si intende qualunque accordo o contratto, comunque denominato, di non rivendicazione di un brevetto (come per esempio un permesso esplicito di utilizzare un brevetto o un accordo di rinuncia alla persecuzione per violazione di brevetto). “Garantire” una tale licenza di brevetto ad una parte significa portare a termine un tale accordo o contratto di non rivendicazione di brevetto contro la parte.

Se distribuisi un programma coperto da questa Licenza, confidando consapevolmente su una licenza di brevetto, e il Sorgente Corrispondente per il programma non è reso disponibile per la copia, senza alcun onere aggiuntivo e comunque nel rispetto delle condizioni di questa Licenza, attraverso un server di rete pubblicamente accessibile o tramite altri mezzi facilmente accessibili, allora devi (1) fare in modo che il Sorgente Corrispondente sia reso disponibile come sopra, oppure (2) fare in modo di rinunciare ai benefici della licenza di brevetto per quel particolare programma, oppure (3) adoperarti, in maniera consistente con le condizioni di questa Licenza, per estendere la licenza di brevetto a tutti i destinatari successivi. “Confidare consapevolmente” significa che tu sei attualmente cosciente che, eccettuata la licenza di brevetto, la distribuzione da parte tua di un programma protetto da questa Licenza in un paese, o l'utilizzo in un paese del programma coperto da questa Licenza da parte di un destinatario, può violare uno o più brevetti in quel paese che tu hai ragione di ritenere validi.

Se, come conseguenza o in connessione con una singola transazione o con un dato accordo, distribuisi, o fai in modo di distribuire, un programma coperto da questa Licenza, e garantischi una licenza di brevetto per alcune delle parti che ricevono il Programma autorizzandole ad utilizzare, propagare, modificare o distribuire una specifica copia del Programma, allora la licenza di brevetto che fornisci è automaticamente estesa a tutti i destinatari del Programma coperto da questa Licenza e delle opere basate sul Programma.

Una licenza di brevetto è “discriminatoria” se non include nell'ambito della sua copertura, proibisce l'esercizio, o è vincolata al non-esercizio di uno o più dei diritti che sono specificatamente garantiti da questa Licenza. Non puoi distribuire un Programma coperto da questa Licenza se sei parte di un accordo con una terza parte la cui attività comprende la distribuzione di software, secondo il quale tu sei costretto ad un pagamento alla parte terza in funzione della tua attività di distribuzione del Programma, e in conseguenza del quale la parte terza garantisce, a qualunque delle parti che riceveranno il Programma da te, una licenza di brevetto discriminatoria (a) assieme a copie del Programma coperto da questa Licenza distribuite da te (o ad altre copie fatte da codeste copie), oppure (b) principalmente per e in connessione con specifici prodotti o raccolte di prodotti che contengono il Programma, a meno che l'accordo non sia stato stipulato, o le licenze di brevetto non siano state rilasciate, prima del 28 Marzo 2007.

Nessuna parte di questa Licenza può essere interpretata come atta ad escludere o limitare gli effetti di qualunque altra licenza o altri meccanismi di difesa dalla violazione che possano altrimenti essere resi disponibili dalla normativa vigente in materia di brevetti.

## 12. Nessuna resa di libertà altrui

Se ti vengono imposte delle condizioni (da un ordine giudiziario, da un accordo o da qualunque altra eventualità) che contraddicono le condizioni di questa Licenza, non sei in nessun modo esonerato dal rispetto delle condizioni di questa Licenza. Se non puoi distribuire un Programma coperto da questa Licenza per sottostare simultaneamente agli obblighi derivanti da questa Licenza e ad altri obblighi pertinenti, allora non puoi distribuire il Programma per nessun motivo. Per esempio, se accetti delle condizioni che ti obbligano a richiedere il pagamento di una royalty per le distribuzioni successivamente effettuate da coloro ai quali hai distribuito il Programma, l'unico modo per soddisfare sia queste condizioni che questa Licenza è evitare del tutto la distribuzione del Programma.

## 13. Utilizzo con la GNU Affero General Public License

Indipendentemente da qualunque altra condizione espressa da questa Licenza, hai il permesso di collegare o combinare qualunque Programma coperto da questa Licenza con un'opera rilasciata sotto la versione 3 della licenza GNU Affero General Public License, ottenendo un singolo Programma derivato, e di distribuire il Programma risultante. Le condizioni di questa Licenza continuano a valere per le parti riguardanti il Programma che sono coperte da questa Licenza, mentre le condizioni speciali della GNU Affero General Public License, sezione 13, riguardanti l'interazione mediante rete, saranno applicate al Programma così risultante.

## 14. Versioni rivedute di questa Licenza

La Free Software Foundation può pubblicare delle versioni rivedute e/o delle nuove versioni della GNU General Public License di tanto in tanto. Tali versioni saranno simili, nello spirito, alla presente versione, ma potranno differire nei dettagli al fine di affrontare nuovi problemi e nuove situazioni.

A ciascuna versione viene assegnato un numero identificativo di versione. Se il Programma specifica che si applica a sè stesso una certa versione della GNU General Public License, "o qualunque altra versione successiva", hai la possibilità di sottostare alle condizioni di quella specifica versione o di qualunque altra versione successiva pubblicata dalla Free Software Foundation. Se il Programma non specifica un numero di versione della GNU General Public License, puoi scegliere qualunque versione della GNU General Public License pubblicata dalla Free Software Foundation.

Se il Programma specifica che un sostituto o un procuratore può decidere quali versioni future della GNU General Public License posso essere utilizzate, allora tale scelta di accettazione di una data versione ti autorizza, in maniera permanente, ad utilizzare quella versione della Licenza per il Programma.

Versioni successive della Licenza possono garantire diritti aggiuntivi o leggermente differenti. Ad ogni modo, nessun obbligo aggiuntivo viene imposto agli autori o ai detentori di copyright come conseguenza della tua scelta di adottare una versione successiva della Licenza.

## 15. Rinuncia alla Garanzia

NON C'È NESSUNA GARANZIA PER IL PROGRAMMA, PER QUANTO CONSENTITO DALLE VIGENTI NORMATIVE. ECCETTO QUANDO ALTRIMENTI STABILITO PER ISCRITTO, I DETENTORI DEL COPYRIGHT E/O LE ALTRE PARTI FORNISCONO IL PROGRAMMA "COSÌ COME È" SENZA GARANZIA DI

ALCUN TIPO, NÉ ESPRESSA NÉ IMPLICITA, INCLUSE, MA NON LIMITATE A, LE GARANZIE DI COMMERCIALIZZABILITÀ O DI UTILIZZABILITÀ PER UN PARTICOLARE SCOPO. L'INTERO RISCHIO CONCERNENTE LA QUALITÀ E LE PRESTAZIONI DEL PROGRAMMA È DEL LICENZIATARIO. SE IL PROGRAMMA DOVESSE RISULTARE DIFETTOSO, IL LICENZIATARIO SI ASSUME I COSTI DI MANUTENZIONE, RIPARAZIONE O CORREZIONE.

16. Limitazione di Responsabilità

IN NESSUN CASO, A MENO CHE NON SIA RICHIESTO DALLA NORMATIVA VIGENTE O CONCORDATO PER ISCRITTO, I DETENTORI DEL COPYRIGHT, O QUALUNQUE ALTRA PARTE CHE MODIFICA E/O DISTRIBUISCE IL PROGRAMMA SECONDO LE CONDIZIONI PRECEDENTI, POSSONO ESSERE RITENUTI RESPONSABILI NEI CONFRONTI DEL LICENZIATARIO PER DANNI, INCLUSO QUALUNQUE DANNEGGIAMENTO GENERICO, SPECIALE, INCIDENTALE O CONSEGUENZIALE DOVUTO ALL'USO O ALL'IMPOSSIBILITÀ D'USO DEL PROGRAMMA (INCLUSI, MA NON LIMITATI A, LE PERDITE DI DATI, LA CORRUZIONE DI DATI, LE PERDITE SOSTENUTE DAL LICENZIATARIO O DA TERZE PARTI O L'IMPOSSIBILITÀ DEL PROGRAMMA A FUNZIONARE ASSIEME AD ALTRI PROGRAMMI), ANCHE NEL CASO IN CUI IL DETENTORE O LE ALTRE PARTI SIANO STATI AVVISATI CIRCA LA POSSIBILITÀ DI TALI DANNEGGIAMENTI.

17. Interpretazione delle Sezioni 15 e 16

Se la dichiarazione di garanzia e la limitazione di responsabilità fornite precedentemente non hanno effetto legale in un paese a causa delle loro condizioni, le corti di giustizia devono applicare la norma locale che più si avvicini al rifiuto assoluto di qualsivoglia responsabilità civile relativa al Programma, a meno che una garanzia o una assunzione di responsabilità scritta non accompagni una copia del programma ottenuta dietro pagamento.

## FINE DEI TERMINI E DELLE CONDIZIONI

### Come applicare queste condizioni di Licenza ai vostri programmi

Se sviluppi un nuovo programma, e vuoi che esso sia della massima utilità, il modo migliore è renderlo software libero in modo che chiunque possa ridistribuirlo e modificarlo secondo i termini di questa Licenza.

Per fare ciò, allega le seguenti note informative al programma. Il modo migliore è inserirle all'inizio di ciascun file sorgente, al fine di rimarcare adeguatamente la mancanza di garanzia; ciascun file dovrebbe inoltre contenere la dichiarazione di copyright e un riferimento al posto in cui è possibile ottenere la versione completa delle note informative.

*<una riga con nome del programma e breve descrizione di ciò che fa.>*  
Copyright (C) <anno> <nome dell'autore>

Questo software è libero; lo puoi distribuire e/o modificare alle condizioni stabilite nella 'GNU General Public License' pubblicata dalla Free Software Foundation; fai riferimento alla versione 3 della Licenza, o (a tua scelta) a una qualsiasi versione successiva.

Questo programma è distribuito con la speranza che sia utile, ma SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di COMMERCIALIZZABILITÀ o IDONEITÀ AD UN PARTICOLARE SCOPO. Si veda la 'GNU General Public License' per ulteriori dettagli.

Dovresti aver ricevuto una copia della GNU General Public License assieme a questo programma; se non è così, si veda  
<http://www.gnu.org/licenses/>.

Inoltre, aggiungi le informazioni necessarie a contattarti via posta ordinaria o via posta elettronica.

Se il programma interagisce mediante terminale, fai in modo che visualizzi, quando viene avviato in modalità interattiva, un breve messaggio come quello che segue:

```
<programma> Copyright (C) <anno> <nome dell'autore>
Questo programma non ha ALCUNA GARANZIA; per dettagli usare il comando
'show w'.
Questo è software libero, e ognuno è libero di ridistribuirlo
sotto certe condizioni; usare il comando 'show c' per i dettagli.
```

Gli ipotetici comandi 'show w' e 'show c' devono visualizzare le parti corrispondenti della GNU General Public License. Naturalmente i comandi del tuo programma potrebbero essere differenti; per una interfaccia di tipo GUI, dovresti usare un bottone "About" o "Info".

Devi inoltre fare in modo che il tuo datore di lavoro (se lavori come programmatore presso terzi) o la tua scuola, eventualmente, firmino una "rinuncia al copyright" sul programma, se necessario. Per maggiori informazioni su questo punto, e su come applicare e rispettare la GNU GPL, consultare la pagina <http://www.gnu.org/licenses/>.

La GNU General Public License non consente di incorporare il programma all'interno di software proprietario. Se il tuo programma è una libreria di funzioni, potresti ritenere più opportuno consentire il collegamento tra software proprietario e la tua libreria. Se è questo ciò che vuoi, allora utilizza la GNU Lesser General Public License anziché questa Licenza, ma prima leggi <http://www.gnu.org/philosophy/why-not-lgpl.html>.



# Licenza per Documentazione Libera GNU (FDL)

Versione 1.3, 3 Novembre 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org>

This is an unofficial translation of the GNU Free Documentation License into Italian. It was not published by the Free Software Foundation, and does not legally state the distribution terms for software that uses the GNU FDL—only the original English text of the GNU FDL does that. However, we hope that this translation will help Italian speakers understand the GNU FDL better.

Questa è una traduzione non ufficiale della GNU Free Documentation License in italiano. Non è una pubblicazione della Free Software Foundation, e non ha validità legale per i termini di distribuzione della documentazione che usa la GNU FDL; solo il testo originale inglese della GNU FDL ha tale validità. Comunque, speriamo che questa traduzione aiuti chi parla italiano a comprendere meglio la GNU FDL.

A chiunque è permesso copiare e ridistribuire copie esatte di questo documento di licenza, ma non è in alcun modo consentito apportarvi modifiche.

## 0. PREAMBOLO

Lo scopo di questa licenza è di rendere *liberi* un manuale, un testo o altri documenti funzionali e utili, nel senso di assicurare a tutti la libertà effettiva di copiarli e ridistribuirli, con o senza modifiche, con o senza fini di lucro. In secondo luogo questa licenza prevede per autori ed editori il modo per ottenere il giusto riconoscimento del proprio lavoro, preservandoli dall'essere considerati responsabili per modifiche apportate da altri.

Questa licenza garantisce il “copyleft”: questo significa che i lavori che derivano dal documento originale devono essere ugualmente liberi. È il complemento alla Licenza Pubblica Generale GNU, che è una licenza di tipo “copyleft” pensata per il software libero.

Questa licenza è stata progettata appositamente per l'uso con manuali di software libero, perché il software libero ha bisogno di documentazione libera: un programma libero dovrebbe accompagnarsi a manuali che forniscano le stesse libertà del software. Questa licenza non è limitata alla manualistica del software; può essere utilizzata per ogni testo che tratti un qualsiasi argomento e al di là dell'avvenuta pubblicazione cartacea. Si raccomanda l'uso di questa licenza principalmente per opere che abbiano fini didattici o per manuali.

## 1. APPLICABILITÀ E DEFINIZIONI

Questa licenza si applica a qualsiasi manuale o altra opera, su ogni tipo di supporto, che contenga la nota, posta dal detentore del copyright, che attesti la possibilità di distribuzione secondo i termini di questa licenza. Tale nota permette universalmente, senza pagamento di diritti e senza limiti di durata di utilizzare il lavoro secondo le condizioni qui specificate. Con “documento”, nel seguito ci si riferisce a qualsiasi manuale

o opera. Ogni fruitore è un destinatario della licenza ed è ad esso che si fa riferimento. Si conviene che la licenza viene accettata se si copia, modifica o distribuisce il lavoro in una maniera tale da richiedere il permesso secondo le leggi sul copyright.

Una “versione modificata” del documento è ogni opera contenente il documento stesso o parte di esso, sia riprodotto alla lettera che con modifiche, oppure traduzioni in un’altra lingua.

Una “sezione secondaria” è un’appendice cui si fa riferimento o una premessa del documento e riguarda esclusivamente il rapporto dell’editore o dell’autore del documento con l’argomento generale del documento stesso (o argomenti affini) e non contiene nulla che possa essere compreso nell’argomento principale. (Perciò, se il documento è in parte un manuale di matematica, una sezione secondaria non può contenere spiegazioni di matematica). Il rapporto con l’argomento può essere un tema collegato storicamente con il soggetto principale o con soggetti affini, o essere costituito da argomentazioni legali, commerciali, filosofiche, etiche o politiche pertinenti.

Le “sezioni non modificabili” sono alcune sezioni secondarie i cui titoli sono esplicitamente elencati come titoli delle sezioni non modificabili nella nota che indica che il documento è realizzato sotto questa licenza. Se una sezione non rientra nella precedente definizione di sezione secondaria, allora non è permesso che venga definita come non modificabile. Il documento può anche non contenere sezioni non modificabili. Se nel documento non vengono indicate sezioni non modificabili, allora significa che non ve ne sono.

I “testi di copertina” sono dei brevi brani di testo che sono elencati, nella prima o quarta pagina di copertina, nella nota che indica che il documento è rilasciato sotto questa licenza. Il testo sulla prima di copertina può essere composto al massimo di 5 parole mentre quello sulla quarta di copertina può essere al massimo di 25 parole.

Una copia “trasparente” indica una copia leggibile da un calcolatore, codificata in un formato le cui specifiche sono disponibili pubblicamente, tale che il suo contenuto possa essere modificato in modo semplice con generici editor di testi o (per immagini composte da pixel) con generici editor di immagini o (per i disegni) con qualche editor di disegni ampiamente diffuso; la copia deve essere adatta al trattamento per la formattazione o per la conversione in una varietà di formati atti alla successiva formattazione. Una copia fatta in un formato di file, per il resto trasparente, i cui marcatori o assenza di tali sono stati progettati per intralciare o scoraggiare modifiche future da parte dei lettori non è trasparente. Un formato immagine non è trasparente se viene usato per rappresentare una notevole quantità di testo. Una copia non “trasparente” viene detta “opaca”.

Esempi di formati adatti per copie trasparenti sono l’ASCII puro senza marcatori, il formato di ingresso per Texinfo, il formato di ingresso per LaTeX, SGML o XML accoppiati ad una DTD pubblica e disponibile, e i formati conformi agli standard HTML semplice, Postscript e PDF progettati per essere modificati manualmente. Esempio di formati immagine trasparenti includono il PNG, XCF e JPG. I formati opachi includono i formati proprietari che possono essere letti e modificati solo con word processor proprietari, SGML o XML per cui non è in genere disponibile la DTD o gli strumenti per il trattamento, e i formati HTML, Postscript e PDF generati automaticamente da qualche word processor esclusivamente come output.

La “pagina del titolo” di un libro stampato indica la pagina del titolo stessa, più qualche pagina seguente per quanto necessario a contenere in modo leggibile, il materiale che la licenza prevede che compaia nella pagina del titolo. Per opere in formati in cui non sia contemplata esplicitamente la pagina del titolo, con “pagina del titolo” si intende il testo prossimo al titolo dell’opera, precedente l’inizio del corpo del testo.

Il termine “editore” indica qualunque persona o entità che distribuisce al pubblico copie del documento.

Una sezione “Intitolata XYZ” significa una sottosezione con nome del documento il cui titolo sia precisamente XYZ o che contenga XYZ in parentesi dopo il testo che traduce XYZ in un’altra lingua (in questo caso XYZ sta per uno specifico nome di sezione menzionato sotto, come per i “Riconoscimenti”, “Dediche”, “Approvazioni”, o “Storia”). Secondo questa definizione, “preservare il titolo” di tale sezione quando si modifica il documento, significa che essa rimane una sezione “Intitolata XYZ”.

Il Documento può includere dei limiti alla garanzia accanto alla nota affermando l’applicazione di questa licenza al documento. Questi limiti alla garanzia sono da considerare da includere come riferimento a questa licenza, ma solo per quanto riguarda le limitazioni alla garanzia: ogni altra implicazione che questi limiti alla garanzia possono avere è da considerarsi nulla e non ha effetto sul significato di questa licenza.

## 2. COPIE LETTERALI

Si può copiare e distribuire il documento con qualsiasi mezzo, con o senza fini di lucro, purché tutte le copie contengano questa licenza, le note di copyright e l’avviso che questa licenza si applica al documento, e che non si aggiungano altre condizioni al di fuori di quelle della licenza stessa. Non si possono usare misure tecniche per impedire o controllare la lettura o la produzione di copie successive alle copie che si producono o distribuiscono. Si possono comunque accettare compensi per la copiatura. Se si distribuiscono un numero sufficiente di copie si devono seguire anche le condizioni della sezione 3.

Alle stesse condizioni sopra menzionate si possono prestare copie e mostrarle pubblicamente.

## 3. COPIARE IN NOTEVOLI QUANTITÀ

Se si pubblicano a mezzo stampa (o in formati che tipicamente posseggono copertine) più di 100 copie del documento, e la nota della licenza richiede uno o più testi di copertina, si devono includere nelle copie, in modo chiaro e leggibile, tutti i testi di copertina indicati: il testo della prima di copertina in prima di copertina e il testo di quarta di copertina in quarta di copertina. Ambedue devono identificare l’editore che pubblica il documento. La prima di copertina deve presentare il titolo completo con tutte le parole che lo compongono egualmente visibili ed evidenti. Si può aggiungere altro materiale alle copertine. Il copiare con modifiche limitate alle sole copertine, purché si preservino il titolo e le altre condizioni viste in precedenza, è considerato alla stregua di copiare alla lettera.

Se il testo richiesto per le copertine è troppo voluminoso per essere riprodotto in modo leggibile, se ne può mettere una prima parte (per quanto ragionevolmente può stare) in copertina, e continuare il resto nelle pagine immediatamente seguenti.

Se si pubblicano o distribuiscono copie opache del documento in numero superiore a 100, si deve anche includere una copia trasparente leggibile da un calcolatore in ogni copia oppure menzionare in ogni copia opaca un indirizzo di rete di calcolatori pubblicamente accessibile che utilizzi un protocollo di rete standard pubblico, da cui si possa scaricare liberamente una copia trasparente completa del documento, senza materiale aggiuntivo. Se si adotta quest'ultima opzione, si deve prestare la giusta attenzione, nel momento in cui si inizia la distribuzione in quantità elevata di copie opache, ad assicurarsi che la copia trasparente rimanga accessibile all'indirizzo stabilito fino ad almeno un anno dopo l'ultima distribuzione (direttamente o attraverso distributori o rivenditori) di quell'edizione al pubblico.

È caldamente consigliato, benché non obbligatorio, contattare l'autore del documento prima di distribuirne un numero considerevole di copie, per metterlo in grado di fornire una versione aggiornata del documento.

#### 4. MODIFICHE

Si possono copiare e distribuire versioni modificate del documento rispettando le condizioni delle precedenti sezioni 2 e 3, purché la versione modificata sia realizzata seguendo questa stessa licenza, con la versione modificata che svolga il ruolo del “documento”, così da estendere la licenza sulla distribuzione e la modifica a chiunque ne possieda una copia. Inoltre nelle versioni modificate si deve:

- A. Usare nella pagina del titolo (e nelle copertine se ce ne sono) un titolo diverso da quello del documento, e da quelli di versioni precedenti (che devono essere elencati nella sezione storia del documento ove presenti). Si può usare lo stesso titolo di una versione precedente se l'editore di quella versione originale ne ha dato il permesso.
- B. Elencare nella pagina del titolo, come autori, una o più persone o gruppi responsabili in qualità di autori delle modifiche nella versione modificata, insieme ad almeno cinque tra i principali autori del documento (tutti gli autori principali se sono meno di cinque), a meno che questi non abbiano acconsentito a liberarvi da quest'obbligo.
- C. Dichiarare nella pagina del titolo il nome dell'editore della versione modificata in qualità di editore.
- D. Conservare tutte le note di copyright del documento originale.
- E. Aggiungere un'appropriata nota di copyright per le modifiche di seguito alle altre note di copyright.
- F. Includere, immediatamente dopo la nota di copyright, una nota di licenza che dia pubblicamente il permesso di usare la versione modificata nei termini di questa licenza, nella forma mostrata nell'Addendum alla fine di questo testo.
- G. Preservare in tale nota di licenza l'elenco completo di sezioni non modificabili e testi di copertina richiesti come previsto dalla licenza del documento.
- H. Includere una copia non modificata di questa licenza.
- I. Conservare la sezione intitolata “Storia”, e il suo titolo, e aggiungere a questa un elemento che riporti almeno il titolo, l'anno, i nuovi autori, e gli editori della versione modificata come figurano nella pagina del titolo. Se non ci sono sezioni intitolate “Storia” nel documento, crearne una che riporti il titolo, gli autori, gli editori del documento come figurano nella pagina del titolo, quindi aggiungere un elemento che descriva la versione modificata come detto in precedenza.

- J. Conservare l'indirizzo in rete riportato nel documento, se c'è, al fine del pubblico accesso ad una copia trasparente, e possibilmente l'indirizzo in rete per le precedenti versioni su cui ci si è basati. Questi possono essere collocati nella sezione "Storia". Si può omettere un indirizzo di rete per un'opera pubblicata almeno quattro anni prima del documento stesso, o se l'originario editore della versione cui ci si riferisce ne dà il permesso.
- K. In ogni sezione di "Ringraziamenti" o "Dediche", si conservi il titolo della sezione, e all'interno della sezione tutta la sostanza e il tono di ognuno dei ringraziamenti ai contributori e/o le dediche ivi contenute.
- L. Si conservino inalterate le sezioni non modificabili del documento, nei propri testi e nei propri titoli. I numeri della sezione o equivalenti non sono considerati parte del titolo della sezione.
- M. Si cancelli ogni sezione intitolata "Approvazioni". Tale sezione non può essere inclusa nella versione modificata.
- N. Non si cambi il titolo di sezioni esistenti in "Approvazioni" o in modo tale che si possa creare confusione con i titoli di sezioni non modificabili.
- O. Si conservino tutti i limiti alla garanzia.

Se la versione modificata comprende nuove sezioni di primaria importanza o appendici che ricadono in "sezioni secondarie", e non contengono materiale copiato dal documento, si ha facoltà di rendere non modificabili quante sezioni si voglia. Per fare ciò si aggiunga il loro titolo alla lista delle sezioni non modificabili nella nota di licenza della versione modificata. Questi titoli devono essere distinti dai titoli di ogni altra sezione.

Si può aggiungere una sezione intitolata "Approvazioni", a patto che non contenga altro che le approvazioni alla versione modificata prodotte da vari soggetti—per esempio, affermazioni di revisione o che il testo è stato approvato da una organizzazione come la definizione normativa di uno standard.

Si può aggiungere un brano fino a cinque parole come testo di prima di copertina e un brano fino a 25 parole come testo di quarta di copertina, alla fine dell'elenco dei testi di copertina nella versione modificata. Solamente un brano del testo di prima di copertina e uno del testo di quarta di copertina possono essere aggiunti (anche con adattamenti) da ciascuna persona o organizzazione. Se il documento include già un testo di copertina per la stessa copertina, precedentemente aggiunto o adattato da qualunque fruitore o dalla stessa organizzazione nel nome della quale si agisce, non se ne può aggiungere un altro, ma si può rimpiazzare il vecchio ottenendo l'esplicita autorizzazione dall'editore precedente che aveva aggiunto il testo di copertina.

L'autore/i e l'editore/i del documento non danno, tramite questa licenza, il permesso di usare i loro nomi per pubblicizzare o asserire, anche implicitamente, la loro approvazione di ogni versione modificata.

## 5. COMBINAZIONE DI DOCUMENTI

Si può combinare il documento con altri pubblicati con questa licenza, seguendo i termini definiti nella precedente sezione 4 per le versioni modificate, a patto che si includa l'insieme di tutte le sezioni non modificabili di tutti i documenti originali, senza modifiche, e si elenchino tutte come sezioni non modificabili della combinazione di documenti nella licenza della stessa, mantenendo tutti i limiti alla garanzia.

Nella combinazione è necessaria una sola copia di questa licenza, e più sezioni non modificabili possono essere rimpiazzate da una singola copia se identiche. Se ci sono più sezioni non modificabili con lo stesso nome ma contenuti differenti, si renda unico il titolo di ciascuna sezione aggiungendovi, alla fine e tra parentesi, il nome dell'autore o editore della sezione, se noti, o altrimenti un numero distintivo. Si facciano gli stessi aggiustamenti ai titoli delle sezioni nell'elenco delle sezioni non modificabili nella nota di copyright della combinazione.

Nella combinazione si devono unire le varie sezioni intitolate "Storia" nei vari documenti originali di partenza per formare una unica sezione intitolata "Storia"; allo stesso modo si unisca ogni sezione intitolata "Ringraziamenti", e ogni sezione intitolata "Dediche". Si devono eliminare tutte le sezioni intitolate "Approvazioni".

## 6. RACCOLTE DI DOCUMENTI

Si può produrre una raccolta che consista del documento e di altri documenti rilasciati sotto questa licenza, e rimpiazzare le singole copie di questa licenza nei vari documenti con una sola inclusa nella raccolta, solamente se si seguono le regole fissate da questa licenza per le copie alla lettera come se si applicassero a ciascun documento.

Si può estrarre un singolo documento da tale raccolta e distribuirlo separatamente sotto questa licenza, solo se si inserisce una copia di questa licenza nel documento estratto e se si seguono tutte le altre regole fissate da questa licenza per le copie alla lettera del documento.

## 7. AGGREGAZIONE A LAVORI INDIPENDENTI

Un'unione del documento o sue derivazioni con altri documenti o lavori separati o indipendenti, all'interno di, o a formare un, archivio o un supporto, per la memorizzazione o la distribuzione, viene chiamato un "aggregato" se il copyright risultante dall'unione non viene usato per limitare i diritti legali degli utilizzatori oltre a ciò che viene permesso dai singoli lavori. Quando il documento viene incluso in un aggregato, questa licenza non si applica ad altri lavori nell'aggregato che non siano essi stessi dei lavori derivati dal documento.

Se le esigenze del testo di copertina della sezione 3 sono applicabili a queste copie del documento allora, se il documento è inferiore alla metà dell'intero aggregato i testi di copertina del documento possono essere piazzati in copertine che delimitano il documento all'interno dell'aggregato, o dell'equivalente elettronico delle copertine se il documento è in un formato elettronico. Altrimenti devono apparire nella copertina dell'intero aggregato.

## 8. TRADUZIONE

La traduzione è considerata un tipo di modifica, di conseguenza si possono distribuire traduzioni del documento nei termini della sezione 4. Rimpiazzare sezioni non modificabili con traduzioni richiede un particolare permesso da parte dei detentori del copyright, ma è possibile includere la traduzione di parti o di tutte le sezioni non modificabili in aggiunta alle versioni originali di queste sezioni. È possibile includere una traduzione di questa licenza, di tutte le avvertenze del documento e di tutti i limiti di garanzia, a condizione che si includa anche la versione originale in inglese della licenza completa, comprese le avvertenze e limitazioni di garanzia. In caso di discordanza tra la traduzione e la versione originale inglese di questa licenza o avvertenza o limitazione di garanzia, prevale sempre la versione originale inglese.

Se una sezione del documento viene titolata “Riconoscimenti”, “Dediche”, o “Storia”, il requisito (sezione 4) di preservare il titolo (sezione 1) richiederà tipicamente il cambiamento del titolo.

## 9. CESSAZIONE DELLA LICENZA

Non si può sublicenziare il documento, copiarlo, modificarlo, o distribuirlo al di fuori dei termini espressamente previsti da questa licenza. Ogni altro tentativo di applicare una licenza al documento, copiarlo, modificarlo, o distribuirlo è nullo e pone fine automaticamente ai diritti previsti da questa licenza.

In ogni caso, se cessano tutte le violazioni di questa Licenza, allora la specifica licenza di un particolare detentore del copyright viene ripristinata (a) in via provvisoria, a meno che e fino a quando il detentore del copyright non faccia estinguere esplicitamente e definitivamente la licenza, e (b) in via permanente se il detentore del copyright non notifica in alcun modo la violazione entro 60 giorni dalla cessazione della licenza.

Inoltre, la licenza di un dato detentore del copyright viene ripristinata in maniera permanente se quest'ultimo notifica la violazione in maniera adeguata, se si tratta della prima volta che si riceve una notifica di violazione della licenza (per qualsiasi opera) dallo stesso detentore di copyright, e se la violazione viene corretta entro 30 giorni dalla data di ricezione della notifica di violazione.

La cessazione dei diritti come specificato in questa sezione non provoca la cessazione delle licenze di terze parti che abbiano ricevuto copie o diritti secondo questa licenza. Se i diritti sono cessati e non sono stati ristabiliti in via permanente, la ricezione di una copia dello stesso materiale, in tutto o in parte, non dà alcun diritto ad utilizzarlo.

## 10. REVISIONI FUTURE DI QUESTA LICENZA

La Free Software Foundation può occasionalmente pubblicare versioni nuove o rivedute della Licenza per Documentazione Libera GNU. Le nuove versioni saranno simili nello spirito alla versione attuale ma potrebbero differirne in qualche dettaglio per affrontare nuovi problemi e concetti. Si veda <http://www.gnu.org/copyleft/>.

Ad ogni versione della licenza viene dato un numero che la distingue. Se il documento specifica che si riferisce ad una versione particolare della licenza “o ogni versione successiva”, si ha la possibilità di seguire termini e condizioni sia della versione specificata che di ogni versione successiva pubblicata (non come bozza) dalla Free Software Foundation. Se il documento non specifica un numero di versione particolare di questa licenza, si può scegliere ogni versione pubblicata (non come bozza) dalla Free Software Foundation. Se il documento specifica che un delegato può decidere quale futura versione di questa licenza può essere utilizzata, allora la dichiarazione pubblica di accettazione della versione, da parte del delegato, autorizza in maniera permanente a scegliere tale versione per il documento.

## 11. CAMBIO DI LICENZA

Il termine “sito per la collaborazione massiva multiautore” (o “sito MMC”) indica qualsiasi server web che pubblica opere sottoponibili a copyright e fornisce a chiunque appositi strumenti per modificare tali opere. Un wiki pubblico modificabile da chiunque è un esempio di server in questione. Una “collaborazione massiva multiautore” (o “MMC”) contenuta nel sito indica un qualunque insieme di opere sottoponibili a copyright pubblicate sul sito MMC.

Il termine “CC-BY-SA” indica la licenza Creative Commons Attribution-Share Alike 3.0 pubblicata dalla Creative Commons Corporation, un’organizzazione senza fini di lucro con sede principale a San Francisco, California, come anche le future versioni di tale licenza pubblicate dalla stessa organizzazione.

“Incorporare” significa pubblicare o ripubblicare un documento in tutto o in parte, come parte di un altro documento.

Una MMC è “qualificata a cambiare questa licenza” se ha adottato questa licenza e se tutte le opere precedentemente pubblicate con questa licenza altrove rispetto alla MMC e successivamente incorporate del tutto o in parte nella MMC (1) non hanno testo di copertina o sezioni invariati e (2) sono state incorporate prima del 1 ° Novembre 2008.

L’operatore di un sito MMC può ripubblicare un MMC contenuto nel sito con una CC-BY-SA nello stesso sito in qualsiasi momento prima del 1 ° Agosto 2009, da parte di una MMC qualificata a cambiare questa licenza.

## **ADDENDUM: Come usare questa licenza per i vostri documenti**

Per applicare questa licenza ad un documento che si è scritto, si includa una copia della licenza nel documento e si inserisca la seguente nota di copyright appena dopo la pagina del titolo:

```
Copyright (C) <anno> <il vostro nome>
È permesso copiare, distribuire e/o modificare questo documento
seguendo i termini della ‘Licenza per documentazione libera GNU’, versione 1.3
o ogni versione successiva pubblicata dalla Free Software Foundation;
senza sezioni non modificabili, senza testi di prima di copertina e di quarta di copertina.
Una copia della licenza è inclusa nella sezione intitolata
‘Licenza per la documentazione libera GNU’.
```

Se ci sono sezioni non modificabili, testi di prima di copertina e di quarta di copertina, scrivere nella parte “with... di copertina” il testo seguente:

```
con le seguenti sezioni non modificabili lista dei loro titoli,
con i seguenti testi di prima di copertina elenco,
e con i seguenti testi di quarta di copertina elenco,
```

Se esistono delle sezioni non modificabili ma non i testi di copertina, o qualche altra combinazione dei tre elementi sopra riportati, fondere assieme le due alternative in modo da conformarsi alla situazione descritta.

Se il vostro documento contiene esempi non banali di programma in codice sorgente si raccomanda di rilasciare gli esempi contemporaneamente applicandovi anche una licenza di software libero di vostra scelta, come per esempio la Licenza Pubblica Generica GNU, al fine di permetterne l’uso come software libero.

# Indice analitico

## !

- ! (punto esclamativo), operatore ! .. 137, 140, 148, 289
- ! (punto esclamativo), operatore != ..... 133, 141
- ! (punto esclamativo), operatore !~ .... 49, 57, 61, 117, 133, 135, 141, 146

## "

- " (doppio apice), in costanti *regexp* ..... 58
- " (doppio apice), nei comandi shell ..... 22

## #

- # (cancellotto), #! (*script* eseguibili) ..... 19
- # (cancellotto), commentare ..... 20

## \$

- \$ (dollaro), incrementare campi e vettori ..... 129
- \$ (dollaro), operatore di campo \$ ..... 67, 140
- \$ (dollaro), operatore *regexp* ..... 53

## %

- % (percento), operatore % ..... 140
- % (percento), operatore %= ..... 128, 141

## &

- & (e commerciale), funzioni  
  `gsub()/gensub()/sub()` e ..... 207
- & (e commerciale), operatore && ..... 137, 141

## ,

- ' (apice singolo) ..... 17
- ' (apice singolo), con doppio apice ..... 22
- ' (apice singolo), nei comandi di shell ..... 22
- ' (apice singolo), nella riga di comando di *gawk* ..... 19
- ' (apice singolo), vs. apostrofo ..... 20

## (

- () (parentesi), in un profilo ..... 345
- () (parentesi), operatore *regexp* ..... 53

## \*

- \* (asterisco), operatore \*\* ..... 124, 140
- \* (asterisco), operatore \*\*= ..... 128, 141
- \* (asterisco), operatore \*, come operatore di moltiplicazione ..... 140
- \* (asterisco), operatore \*, come operatore *regexp* ..... 54
- \* (asterisco), operatore \*, individuare la stringa nulla ..... 207
- \* (asterisco), operatore \*= ..... 128, 141

## +

- + (più), operatore + ..... 140
- + (più), operatore ++ ..... 128, 129, 140
- + (più), operatore += ..... 127, 141
- + (più), operatore *regexp* ..... 54

## ,

- , (virgola), negli intervalli di ricerca ..... 147

## —

- (meno), in espressioni tra parentesi quadre ... 55
- (meno), nomi di file che iniziano con ..... 34
- (meno), operatore - ..... 140
- (meno), operatore -- ..... 129, 140
- (meno), operatore -= ..... 128, 141
- assign, opzione ..... 34
- bignum, opzione ..... 37
- characters-as-bytes, opzione ..... 34
- copyright, opzione ..... 35
- debug, opzione ..... 35
- disable-extensions, opzione di configurazione ..... 485
- disable-lint, opzione di configurazione .... 485
- disable-nls, opzione di configurazione .... 485
- dump-variables, opzione ..... 35
- dump-variables, opzione, uso per funzioni di libreria ..... 246
- exec, opzione ..... 35
- field-separator, opzione ..... 33
- file, opzione ..... 33
- gen-pot, opzione ..... 36, 354
- help, opzione ..... 36
- include, opzione ..... 36
- lint, opzione ..... 33, 37
- lint-old, opzione ..... 39
- load, opzione ..... 36
- no-optimize, opzione ..... 39
- non-decimal-data, opzione ..... 37
- non-decimal-data, opzione, funzione `strtonum()` e ..... 332

--optimize, opzione .....	38	.	
--posix, opzione .....	38	. (punto), operatore <i>regexp</i> .....	53
--posix, opzione, e opzione --traditional ....	38	.gmo, file .....	350
--pretty-print, opzione .....	37	.gmo, file, specificare la directory di .....	350, 352
--profile, opzione .....	38, 343	.mo, file, conversione da .po .....	358
--re-interval, opzione .....	38	.po, file .....	350, 354
--sandbox, opzione .....	39	.po, file, conversione in .mo .....	358
--sandbox, opzione, disabilitare la funzione <i>system()</i> .....	213	.pot, file .....	350
--sandbox, opzione, ridirezione dell'input con <i>getline</i> .....	83	/	
--sandbox, opzione, ridirezione dell'output con <i>print</i> , <i>printf</i> .....	104	/ (barra), criteri di ricerca e .....	146
--source, opzione .....	35	/ (barra), operatore / .....	140
--traditional, opzione .....	35	/ (barra), operatore /= .....	128, 141
--traditional, opzione, e opzione --posix ....	38	/ (barra), operatore /=, vs. costante <i>regexp</i> /=.../ .....	128
--use-lc-numeric, opzione .....	37	/ (barra), per delimitare le espressioni regolari .....	49
--version, opzione .....	39	/=, operatore, vs. costante <i>regexp</i> /=.../ .....	128
--with-whiny-user-strftime, opzione di configurazione .....	485	/dev/..., file speciali .....	108
-b, opzione .....	34	/dev/fd/ <i>N</i> , file speciali (in <i>gawk</i> ) .....	108
-c, opzione .....	35	/inet/..., file speciali (in <i>gawk</i> ) .....	341
-C, opzione .....	35	/inet4/..., file speciali (in <i>gawk</i> ) .....	341
-d, opzione .....	35	/inet6/..., file speciali (in <i>gawk</i> ) .....	341
-D, opzione .....	35		
-e, opzione .....	35, 40	;	
-E, opzione .....	35	; (punto e virgola), <i>AWKPATH</i> variabile e .....	487
-f, opzione .....	19, 33	; (punto e virgola), separare istruzioni nelle azioni .....	29, 152, 153
-f, opzione, usi multipli .....	39		
-F, opzione .....	33	<	
-F, opzione sulla riga di comando .....	74	< (parentesi acuta sinistra), operatore < ..	133, 141
-F, opzione, opzione -Ft imposta FS a TAB ....	39	< (parentesi acuta sinistra), operatore < (I/O) ..	85
-g, opzione .....	36	< (parentesi acuta sinistra), operatore <= ..	133, 141
-h, opzione .....	36		
-i, opzione .....	36	=	
-l, opzione .....	36, 37	= (uguale), operatore = .....	126
-L, opzione .....	39	= (uguale), operatore == .....	133, 141
-M, opzione .....	37		
-n, opzione .....	37	>	
-N, opzione .....	37	> (parentesi acuta destra), operatore > ...	133, 141
-o, opzione .....	37	> (parentesi acuta destra), operatore > (I/O) ..	105
-O, opzione .....	38	> (parentesi acuta destra), operatore >= ..	133, 141
-p, opzione .....	38	> (parentesi acuta destra), operatore >> (I/O) .....	105, 141
-P, opzione .....	38		
-r, opzione .....	38	?	
-s, opzione .....	39	? (punto interrogativo), operatore ?: .....	141
-S, opzione .....	39	? (punto interrogativo), operatore <i>regexp</i> ...	54, 60
-v, opzione .....	34, 120		
-V, opzione .....	39		
-W, opzione .....	34		

**@**

- @, notazione per la chiamata indiretta di funzioni ..... 235
- @include, direttiva ..... 45
- @load, direttiva ..... 47

**[**

- [ (parentesi quadre), operatore *regexp* ..... 53

**^**

- ^ (circonflesso), in espressioni tra parentesi quadre ..... 55
- ^ (circonflesso), in FS ..... 73
- ^ (circonflesso), operatore ^ ..... 140
- ^ (circonflesso), operatore ^= ..... 128, 141
- ^ (circonflesso), operatore *regexp* ..... 53, 60

**-**

- (trattino basso), macro C ..... 350
- (trattino basso), nei nomi di variabili private ..... 246
- (trattino basso), stringa traducibile ..... 353
- \_gr\_init(), funzione definita dall'utente ..... 274
- \_ord\_init(), funzione definita dall'utente ..... 252
- \_pw\_init(), funzione definita dall'utente ..... 270

**\**

- \ (barra inversa) ..... 21
- \ (barra inversa), \", sequenza di protezione .... 51
- \ (barra inversa), \', operatore (*gawk*) ..... 60
- \ (barra inversa), \/, sequenza di protezione .... 51
- \ (barra inversa), \<, operatore (*gawk*) ..... 59
- \ (barra inversa), \>, operatore (*gawk*) ..... 59
- \ (barra inversa), \', operatore (*gawk*) ..... 60
- \ (barra inversa), \a, sequenza di protezione .... 50
- \ (barra inversa), \b, sequenza di protezione .... 50
- \ (barra inversa), \B, operatore (*gawk*) ..... 59
- \ (barra inversa), \f, sequenza di protezione .... 50
- \ (barra inversa), \n, sequenza di protezione .... 50
- \ (barra inversa), \nnn, sequenza di protezione ..... 51
- \ (barra inversa), \r, sequenza di protezione .... 50
- \ (barra inversa), \s, operatore (*gawk*) ..... 59
- \ (barra inversa), \S, operatore (*gawk*) ..... 59
- \ (barra inversa), \t, sequenza di protezione .... 50
- \ (barra inversa), \v, sequenza di protezione .... 50
- \ (barra inversa), \w, operatore (*gawk*) ..... 59
- \ (barra inversa), \W, operatore (*gawk*) ..... 59
- \ (barra inversa), \x, sequenza di protezione .... 51
- \ (barra inversa), \y, operatore (*gawk*) ..... 59
- \ (barra inversa), come separatore di campo .... 74
- \ (barra inversa), continuazione di riga e ..... 28

- \ (barra inversa), continuazione di riga e, in *csh* ..... 29
- \ (barra inversa), continuazione di riga, commenti e ..... 29
- \ (barra inversa), gsub()/gensub()/sub() funzioni e ..... 207
- \ (barra inversa), in costanti *regexp* ..... 58
- \ (barra inversa), in espressioni tra parentesi quadre ..... 55
- \ (barra inversa), in sequenze di protezione ..... 50, 51
- \ (barra inversa), in sequenze di protezione, POSIX e ..... 52
- \ (barra inversa), nei comandi di shell ..... 22
- \ (barra inversa), operatore *regexp* ..... 53

**{**

- { (parentesi graffe) ..... 345
- { (parentesi graffe), azioni e ..... 152
- { (parentesi graffe), istruzioni, raggruppare ... 153

**|**

- | (barra verticale) ..... 53
- | (barra verticale), operatore | (I/O) .. 86, 105, 141
- | (barra verticale), operatore |& (I/O) .... 88, 106, 141, 339
- | (barra verticale), operatore |& (I/O), *pipe*, chiusura ..... 111
- | (barra verticale), operatore || ..... 137, 141

**~**

- ~ (tilde), operatore ~ ..... 49, 57, 61, 117, 133, 135, 141, 146

## A

- a capo, come separatore di record ..... 63
- a capo, separatore di
  - istruzioni nelle azioni ..... 152, 153
- a.b., si veda angolo buio ..... 10
- a\_fine\_file()**, funzione definita dall'utente .. 259
- a\_inizio\_file()**, funzione
  - definita dall'utente ..... 259
- abbracci mortali ..... 340
- abilitare un punto d'interruzione ..... 368
- accedere alle variabili globali dalle estensioni .. 417
- accesso ai campi ..... 67
- account, informazioni sugli ..... 268, 272
- Ada, linguaggio di programmazione ..... 515
- aggiungere funzionalità a **gawk** ..... 443, 500
- aggiungere, campi ..... 70
- Aho, Alfred ..... 6, 475
- alarm.awk**, programma ..... 303
- Algoritmi ..... 512
- allocare memoria per estensioni ..... 403
- ambiguità sintattica: operatore **/=** vs.
  - costante **regex** **/=...** ..... 128
- anagram.awk**, programma ..... 324
- anagrammi, trovare ..... 324
- analizzatore di input personalizzato ..... 408
- and()**, funzione (**gawk**) ..... 220
- and**, operatore logico-booleano ..... 136
- AND**, operazione sui bit ..... 219, 220
- andare a capo ..... 28
- angolo buio ..... 10, 515
- angolo buio, "0" è effettivamente vero ..... 130
- angolo buio, **^**, in **FS** ..... 73
- angolo buio, argomenti da riga di comando .... 121
- angolo buio, carattere di separazione dei
  - decimali nelle localizzazioni ..... 122
- angolo buio, caratteri di
  - controllo del formato ..... 99, 100
- angolo buio, costanti **regex** ..... 117
- angolo buio, costanti **regex**, come argomenti a
  - funzioni definite dall'utente ..... 118
- angolo buio, costanti **regex**, operatore **/=** e ... 128
- angolo buio, file in input ..... 65
- angolo buio, **FS** come stringa nulla ..... 74
- angolo buio, funzione **close()** ..... 112
- angolo buio, funzione **length()** ..... 201
- angolo buio, funzione **split()** ..... 204
- angolo buio, indici di vettori ..... 186
- angolo buio, invocare **awk** ..... 33
- angolo buio, istruzione **break** ..... 158
- angolo buio, istruzione **continue** ..... 159
- angolo buio, istruzione **exit** ..... 161
- angolo buio, operatore **/=** vs.
  - costante **regex** **/=...** ..... 128
- angolo buio, record multiriga ..... 81
- angolo buio, **regex** come secondo
  - argomento di **index()** ..... 200
- angolo buio, separatori di campo ..... 76
- angolo buio, sequenze di protezione ..... 41
- angolo buio, sequenze di protezione,
  - per metacaratteri ..... 52
- angolo buio, stringhe, memorizzazione ..... 67
- angolo buio, valore di **ARGV[0]** ..... 165
- angolo buio, variabile **CONVFM** ..... 121
- angolo buio, variabile **FILENAME** ..... 88, 167
- angolo buio, variabile **NF**, decremento ..... 70
- angolo buio, variabile **OFMT** ..... 98
- angolo buio, variabili **FNR/NR** ..... 172
- ANSI** ..... 515
- API**, delle estensioni ..... 398
- API**, variabili informative dell'estensione ..... 432
- API**, versione ..... 431
- apice singolo (**'**) ..... 17
- apice singolo (**'**), con doppio apice ..... 22
- apice singolo (**'**), nei comandi di shell ..... 22
- apice singolo (**'**), nella riga di
  - comando di **gawk** ..... 19
- apice singolo (**'**), vs. apostrofo ..... 20
- archeologi ..... 493
- arcotangente ..... 196
- ARGC/ARGV**, variabili ..... 165
- ARGC/ARGV**, variabili, argomenti da
  - riga di comando ..... 40
- ARGC/ARGV**, variabili, come usarle ..... 172
- ARGIND**, variabile ..... 166
- ARGIND**, variabile, argomenti da
  - riga di comando ..... 40
- argomenti, elaborazione di ..... 263
- argomenti, nelle chiamate di funzione ..... 138
- argomenti, riga di comando ..... 40, 165, 172
- argomenti, riga di comando, eseguire **awk** ..... 33
- ARGV**, vettore, indicizzare all'interno di ..... 40
- aritmetici, operatori ..... 123
- arrotondamento all'intero più vicino ..... 196
- arrotondare numeri ..... 250
- ASCII** ..... 252, 517
- asort()**, funzione (**gawk**) ..... 198, 336
- asort()**, funzione (**gawk**),
  - ordinamento di vettori ..... 336
- asorti()**, funzione (**gawk**) ..... 198, 336
- asorti()**, funzione (**gawk**),
  - ordinamento di vettori ..... 336
- assegnamenti di variabile e file in input ..... 40
- assegnamenti di variabile, visti
  - come nomi di file ..... 262
- assegnamento, operatori di ..... 126
- assegnamento, operatori di, **lvalue/rvalue** ..... 126
- assegnamento, operatori di, ordine
  - di valutazione ..... 127
- assegnare valori a variabili, nel debugger ..... 371
- assert()**, funzione (libreria C) ..... 249
- assert()**, funzione definita dall'utente ..... 249
- asserzioni ..... 249
- associativi, vettori ..... 178
- asterisco (**\***), operatore **\*\*** ..... 124, 140
- asterisco (**\***), operatore **\*\*=** ..... 128, 141

asterisco (\*), operatore \*, come operatore  
     di moltiplicazione..... 140  
 asterisco (\*), operatore \*, come  
     operatore *regexp*..... 54  
 asterisco (\*), operatore \*,  
     individuare la stringa nulla ..... 207  
 asterisco (\*), operatore \*=..... 128, 141  
 atan2(), funzione..... 196  
 avanzate, funzionalità, di **gawk** ..... 331  
 avanzate, funzionalità,  
     programmazione di rete..... 341  
 avvertimenti, emissione di..... 37  
 avviare il debugger..... 363  
 awk, asserzioni in programmi lunghi ..... 249  
 awk, costanti *regexp* e..... 135  
 awk, debug, abilitare ..... 35  
 awk, descrizione dei termini..... 7  
 awk, eseguire ..... 33  
 awk, funzione di ..... 17  
 awk, **gawk** e..... 5, 7  
 awk, implementazioni di..... 494  
 awk, implementazioni, limiti ..... 88  
 awk, linguaggio, versione POSIX ..... 128  
 awk, nuovo e vecchio ..... 6  
 awk, nuovo vs. vecchio, variabile OFMT ..... 122  
 awk, POSIX e..... 5  
 awk, POSIX e, si veda anche POSIX **awk** ..... 5  
 awk, problemi di implementazione, *pipe* ..... 106  
 awk, profilatura, abilitare la ..... 38  
 awk, programmi ..... 17  
 awk, programmi, collocazione dei ..... 33, 35, 36  
 awk, programmi, eseguire..... 159  
 awk, programmi, esempi di ..... 281  
 awk, programmi, profilare ..... 343  
 awk, si veda anche **gawk** ..... 5  
 awk, storia di ..... 6  
 awk, uso di ..... 5, 17, 30  
 awk, versioni di ..... 461  
 awk, versioni di, differenze tra  
     SVR3.1 e SVR4 ..... 462  
 awk, versioni di, differenze tra  
     SVR4 e POSIX **awk** ..... 463  
 awk, versioni di, differenze tra V7 e SVR3.1 ... 461  
 awk, versioni di, si veda anche Brian  
     Kernighan, **awk** di..... 463, 495  
 awka, compilatore per **awk**..... 495  
 AWKLIBPATH, variabile d'ambiente..... 43  
 AWKPATH, variabile d'ambiente ..... 42, 487  
 awkprof.out, file ..... 343  
 awksed.awk, programma ..... 316  
 awkvars.out, file..... 35  
 azioni..... 152  
 azioni, default ..... 25  
 azioni, istruzioni di controllo in ..... 153  
 azioni, omesse ..... 25

## B

b, comando del debugger (alias per **break**) .... 367  
 backtrace, comando del debugger..... 372  
 barra (/), criteri di ricerca e ..... 146  
 barra (/), operatore / ..... 140  
 barra (/), operatore /=..... 128, 141  
 barra (/), operatore /=, vs.  
     costante *regexp* /=.../ ..... 128  
 barra (/), per delimitare le  
     espressioni regolari..... 49  
 barra inversa (\) ..... 21  
 barra inversa (\), \", sequenza di protezione.... 51  
 barra inversa (\), \', operatore (**gawk**) ..... 60  
 barra inversa (\), \/, sequenza di protezione.... 51  
 barra inversa (\), \<, operatore (**gawk**) ..... 59  
 barra inversa (\), \>, operatore (**gawk**) ..... 59  
 barra inversa (\), \', operatore (**gawk**) ..... 60  
 barra inversa (\), \a, sequenza di protezione.... 50  
 barra inversa (\), \b, sequenza di protezione.... 50  
 barra inversa (\), \B, operatore (**gawk**) ..... 59  
 barra inversa (\), \f, sequenza di protezione.... 50  
 barra inversa (\), \n, sequenza di protezione.... 50  
 barra inversa (\), \nnn,  
     sequenza di protezione ..... 51  
 barra inversa (\), \r, sequenza di protezione.... 50  
 barra inversa (\), \s, operatore (**gawk**) ..... 59  
 barra inversa (\), \S, operatore (**gawk**) ..... 59  
 barra inversa (\), \t, sequenza di protezione.... 50  
 barra inversa (\), \v, sequenza di protezione.... 50  
 barra inversa (\), \w, operatore (**gawk**) ..... 59  
 barra inversa (\), \W, operatore (**gawk**) ..... 59  
 barra inversa (\), \x, sequenza di protezione.... 51  
 barra inversa (\), \y, operatore (**gawk**) ..... 59  
 barra inversa (\), come separatore di campo.... 74  
 barra inversa (\), continuazione di riga e ..... 28  
 barra inversa (\), continuazione  
     di riga e, in **csh**..... 29  
 barra inversa (\), continuazione di  
     riga, commenti e..... 29  
 barra inversa (\),  
     **gsub()**/**gensub()**/**sub()** funzioni e..... 207  
 barra inversa (\), in costanti *regexp* ..... 58  
 barra inversa (\), in espressioni tra  
     parentesi quadre..... 55  
 barra inversa (\), in sequenze  
     di protezione ..... 50, 51  
 barra inversa (\), in sequenze di  
     protezione, POSIX e..... 52  
 barra inversa (\), nei comandi di shell ..... 22  
 barra inversa (\), operatore *regexp* ..... 53  
 barra verticale (|) ..... 53  
 barra verticale (|), operatore | (I/O) ..... 86, 141  
 barra verticale (|), operatore |& (I/O) .. 88, 141, 339  
 barra verticale (|), operatore || ..... 137, 141  
 Beebe, Nelson H.F..... 12, 496  
 BEGIN, criterio di ricerca..... 72, 148

BEGIN, criterio di ricerca, criteri di	
ricerca booleani e .....	147
BEGIN, criterio di ricerca, e profilatura .....	344
BEGIN, criterio di ricerca, eseguire	
programmi <b>awk</b> e .....	282
BEGIN, criterio di ricerca, funzione definita	
dall'utente <b>assert()</b> e .....	250
BEGIN, criterio di ricerca, <b>getline</b> e .....	88
BEGIN, criterio di ricerca,	
intestazioni, aggiungere .....	96
BEGIN, criterio di ricerca, istruzione <b>exit</b> e ....	161
BEGIN, criterio di ricerca, istruzione <b>print</b> e ...	149
BEGIN, criterio di ricerca, istruzioni	
<b>next/nextfile</b> e .....	150, 160
BEGIN, criterio di ricerca, operatori e .....	148
BEGIN, criterio di ricerca, programma <b>pwcat</b> ...	271
BEGIN, criterio di ricerca,	
variabile <b>TEXTDOMAIN</b> e .....	353
BEGIN, criterio di ricerca, variabili <b>OFS/ORS</b> ,	
assegnare valori a .....	97
BEGINFILE, criterio di ricerca .....	150
BEGINFILE, criterio di ricerca, criteri di	
ricerca booleani e .....	147
Bentley, Jon .....	517
Benzinger, Michael .....	477
Berry, Karl .....	12, 13, 475
bidirezionale, processore personalizzato .....	414
binario, input/output .....	162
<b>bindtextdomain()</b> , funzione ( <b>gawk</b> ) .....	224, 352
<b>bindtextdomain()</b> , funzione	
( <b>gawk</b> ), portabilità e .....	356
<b>bindtextdomain()</b> , funzione (libreria C) .....	350
<b>BINMODE</b> , variabile .....	162, 487
biscotto della fortuna .....	516
bit di parità (in ASCII) .....	252
bit, complemento a livello di .....	219
bit, funzioni per la manipolazione di .....	219
bit, operazioni sui .....	219
bit, spostamento di .....	219
<b>bits2str()</b> , funzione definita dall'utente .....	220
booleane, espressioni .....	136
booleani, operatori, si veda	
espressioni booleane .....	136
Bourne shell, uso di apici, regole per la .....	21
<b>break</b> , comando del debugger .....	367
<b>break</b> , istruzione .....	157
breakpoint .....	362
breakpoint, come cancellare .....	367
breakpoint, impostare .....	367
Brennan, Michael .....	2, 3, 13, 187, 316, 494, 495
Brian Kernighan, <b>awk</b> di .....	5, 30, 52, 60, 73, 87, 125, 149, 158, 159, 161, 187, 206, 207, 211
Brian Kernighan, <b>awk</b> di, codice sorgente .....	495
Brian Kernighan, <b>awk</b> di, estensioni .....	463
Brini, Davide .....	326
Brink, Jeroen .....	23
Broder, Alan J. ....	477
Brown, Martin .....	477

<b>bt</b> , comando del debugger (alias	
per <b>backtrace</b> ) .....	372
Buening, Andreas .....	12, 477, 494
buffer, operatori per .....	60
buffer, scrivere su disco un .....	210, 214
bufferizzazione, dell'input/output .....	214, 340
bufferizzazione, dell'output .....	214
bufferizzazione, interattiva vs.	
non interattiva .....	212
bug, segnalare, indirizzo email,	
<b>bug-gawk@gnu.org</b> .....	493
<b>bug-gawk@gnu.org</b> indirizzo per la	
segnalazione dei bug .....	493
BusyBox Awk .....	496

## C

campi .....	63, 67, 512
campi di larghezza costante .....	77
campi di un solo carattere .....	73
campi, aggiungere .....	70
campi, cambiare il contenuto dei .....	69
campi, esame dei .....	67
campi, numero dei .....	67, 68
campi, ritagliare .....	282
campi, separare .....	71
campi, stampare .....	96
campo, separatori di .....	71, 163, 164
campo, separatori di, si veda anche <b>OFS</b> .....	70
campo, separatori di, variabile	
<b>FIELDWIDTHS</b> e .....	163
campo, separatori di, variabile <b>FPAT</b> e .....	163
cancellare punto d'interruzione da una	
determinata posizione .....	367
cancellare punto d'interruzione per numero ....	368
cancellare punto d'osservazione .....	371
cancelletto ( <b>#</b> ), <b>#!</b> ( <i>script</i> eseguibili) .....	19
cancelletto ( <b>#</b> ), commentare .....	20
carattere, valore come numero .....	251
caratteri (codifiche macchina di caratteri) ....	517
caratteri, contare i .....	300
caratteri, elenchi di, in un'espressione regolare ..	55
caratteri, rimpiazzare .....	305
caricare estensioni .....	36
caricare estensioni, direttiva <b>@load</b> .....	47
<b>case</b> , parola chiave .....	156
casuali, numeri, generatore Cliff di .....	251
categoria di localizzazione <b>LC_ALL</b> .....	351
categoria di localizzazione <b>LC_COLLATE</b> .....	351
categoria di localizzazione <b>LC_CTYPE</b> .....	351
categoria di localizzazione <b>LC_MESSAGES</b> .....	351
categoria di localizzazione <b>LC_MONETARY</b> .....	351
categoria di localizzazione <b>LC_NUMERIC</b> .....	351
categoria di localizzazione <b>LC_TIME</b> .....	351
categorie di localizzazione .....	350
Cavaliere Jedi .....	47
cercare e rimpiazzare in stringhe .....	199
CGI, <b>awk script</b> per .....	35

Chassell, Robert J.....	12	colonne, allineamento .....	96
<code>chdir()</code> , estensione .....	445	colonne, ritagliare .....	282
<code>chem</code> , programma di utilità .....	517	comandi da eseguire al punto d'interruzione ...	368
chiamare comandi di shell .....	212	comando <code>cs</code> h .....	29
chiamare funzioni definite dall'utente .....	228	comando <code>cs</code> h, operatore <code> &amp;</code> , confronto con ....	339
chiamare per riferimento .....	231	comando del debugger, <code>b</code> (alias per <code>break</code> ) ....	367
chiamare per valore .....	231	comando del debugger, <code>backtrace</code> .....	372
chiamata di funzione .....	138	comando del debugger, <code>break</code> .....	367
chiamata indiretta di funzione .....	234	comando del debugger, <code>bt</code> (alias	
chiamata indiretta di funzioni, notazione <code>@</code> ....	235	per <code>backtrace</code> ) .....	372
chiamate, <i>stack</i> (pila) delle,		comando del debugger, <code>c</code> (alias	
mostrare nel debugger .....	372	per <code>continue</code> ) .....	369
chiudere un file o un coprocesso .....	210	comando del debugger, <code>clear</code> .....	367
<code>chr()</code> , funzione definita dall'utente .....	252	comando del debugger, <code>commands</code> .....	368
<code>Chr</code> , estensione .....	451	comando del debugger, <code>condition</code> .....	368
cicli .....	154	comando del debugger, <code>continue</code> .....	369
cicli, conteggi per l'intestazione, in un profilo ..	345	comando del debugger, <code>d</code> (alias per <code>delete</code> ) ...	368
cicli, <code>do-while</code> .....	154	comando del debugger, <code>delete</code> .....	368
cicli, <code>for</code> , iterativi .....	155	comando del debugger, <code>disable</code> .....	368
cicli, <code>for</code> , visita di un vettore .....	181	comando del debugger, <code>display</code> .....	370
cicli, istruzione <code>break</code> e .....	157	comando del debugger, <code>down</code> .....	372
cicli, istruzione <code>continue</code> e .....	156	comando del debugger, <code>dump</code> .....	374
cicli, si veda anche <code>while</code> , istruzione .....	154	comando del debugger, <code>e</code> (alias per <code>enable</code> ) ...	368
cicli, uscita .....	157	comando del debugger, <code>enable</code> .....	368
cicli, <code>while</code> .....	154	comando del debugger, <code>end</code> .....	368
circonflesso (^), in espressioni tra		comando del debugger, <code>eval</code> .....	370
parentesi quadre .....	55	comando del debugger, <code>exit</code> .....	375
circonflesso (^), operatore <code>^</code> .....	140	comando del debugger, <code>f</code> (alias per <code>frame</code> ) ....	372
circonflesso (^), operatore <code>^=</code> .....	128, 141	comando del debugger, <code>finish</code> .....	369
circonflesso (^), operatore <code>regex</code> p .....	53, 60	comando del debugger, <code>frame</code> .....	372
classi di caratteri, si veda espressioni tra		comando del debugger, <code>h</code> (alias per <code>help</code> ) .....	375
parentesi quadre .....	53	comando del debugger, <code>help</code> .....	375
<code>clear</code> , comando del debugger .....	367	comando del debugger, <code>i</code> (alias per <code>info</code> ) .....	372
Cliff, generatore di numeri casuali .....	251	comando del debugger, <code>ignore</code> .....	368
<code>cliff_rand()</code> , funzione definita dall'utente ...	251	comando del debugger, <code>info</code> .....	372
<code>close()</code> , funzione .....	109, 210	comando del debugger, <code>l</code> (alias per <code>list</code> ) .....	375
<code>close()</code> , funzione, <i>pipe</i> bidirezionali e .....	340	comando del debugger, <code>list</code> .....	375
<code>close()</code> , funzione, portabilità .....	110	comando del debugger, <code>n</code> (alias per <code>next</code> ) .....	369
<code>close()</code> , funzione, valore di ritorno .....	112	comando del debugger, <code>next</code> .....	369
Close, Diane .....	11, 476	comando del debugger, <code>nexti</code> .....	369
codice di ritorno, di <code>gawk</code> .....	45	comando del debugger, <code>ni</code> (alias per <code>nexti</code> ) ...	369
codice sorgente di <code>awka</code> .....	495	comando del debugger, <code>o</code> (alias per <code>option</code> ) ...	373
codice sorgente di <code>gawk</code> .....	479	comando del debugger, <code>option</code> .....	373
codice sorgente di <code>gawk</code> , ottenere il .....	479	comando del debugger, <code>p</code> (alias per <code>print</code> ) ....	370
codice sorgente, Brian Kernighan <code>awk</code> .....	495	comando del debugger, <code>print</code> .....	370
codice sorgente, BusyBox <code>Awk</code> .....	496	comando del debugger, <code>printf</code> .....	371
codice sorgente, combinare .....	35	comando del debugger, <code>q</code> (alias per <code>quit</code> ) .....	375
codice sorgente, Illumos <code>awk</code> .....	496	comando del debugger, <code>quit</code> .....	375
codice sorgente, <code>jawk</code> .....	496	comando del debugger, <code>r</code> (alias per <code>run</code> ) .....	369
codice sorgente, <code>libmawk</code> .....	496	comando del debugger, <code>return</code> .....	369
codice sorgente, <code>mawk</code> .....	495	comando del debugger, <code>run</code> .....	369
codice sorgente, <code>pawk</code> .....	496	comando del debugger, <code>s</code> (alias per <code>step</code> ) .....	369
codice sorgente, <code>pawk</code> (versione Python) .....	496	comando del debugger, <code>set</code> .....	371
codice sorgente, QSE <code>awk</code> .....	496	comando del debugger, <code>si</code> (alias per <code>stepi</code> ) ...	370
codice sorgente, QuikTrim <code>Awk</code> .....	497	comando del debugger, <code>silent</code> .....	368
codice sorgente, Solaris <code>awk</code> .....	496	comando del debugger, <code>step</code> .....	369
Collado, Manuel .....	12	comando del debugger, <code>stepi</code> .....	370
Colombo, Antonio .....	12, 477	comando del debugger, <code>t</code> (alias per <code>tbreak</code> ) ...	368

comando del debugger, <b>tbreak</b> .....	368	controllare l'ordine di visita dei vettori .....	183
comando del debugger, <b>trace</b> .....	375	controlli <i>lint</i> .....	164
comando del debugger, <b>u</b> (alias per <b>until</b> ) ....	370	controlli <i>lint</i> per funzione indefinita .....	232
comando del debugger, <b>undisplay</b> .....	371	controlli <i>lint</i> , elementi di vettori .....	187
comando del debugger, <b>until</b> .....	370	controlli <i>lint</i> , indici di vettori .....	186
comando del debugger, <b>unwatch</b> .....	371	controllo, tramite istruzioni, in azioni .....	153
comando del debugger, <b>up</b> .....	372	convenzioni di programmazione, istruzione <b>exit</b> .....	161
comando del debugger, <b>w</b> (alias per <b>watch</b> ) ....	371	convenzioni di programmazione, nella scrittura di funzioni .....	225
comando del debugger, <b>watch</b> .....	371	convenzioni di programmazione, nelle chiamate di funzione .....	195
comando del debugger, <b>where</b> (alias per <b>backtrace</b> ) .....	372	convenzioni di programmazione, nomi di variabili private .....	246
comando <b>getline</b> , funzione definita dall'utente, <b>_gr_init()</b> .....	274	convenzioni di programmazione, parametri di funzione .....	233
comando <b>getline</b> , funzione definita dall'utente, <b>_pw_init()</b> .....	271	convenzioni di programmazione, variabili <b>ARGC/ARGV</b> .....	165
comando <b>getline</b> , risoluzione di problemi .....	261	conversione da numeri a stringhe .....	121, 221
comando <b>kill</b> , profilazione dinamica e .....	346	conversione da stringhe a numeri .....	121, 221
comando <b>ls</b> .....	27	conversione di date in marcature temporali ....	216
<b>commands</b> , comando del debugger .....	368	conversione di tipo variabile .....	121
commentare .....	20	conversione di una stringa in un numero .....	204
commenti, continuazione di riga con barra inversa e <b>i</b> .....	29	convertire numeri interi che sono indici di vettore .....	185
<b>comp.lang.awk</b> gruppo di discussione .....	494	convertire stringa in maiuscolo .....	207
compatibile, modalità ( <b>gawk</b> ), numeri esadecimali .....	116	convertire stringa in minuscolo .....	207
compatibile, modalità ( <b>gawk</b> ), numeri ottali .....	116	<b>CONVFMT</b> , variabile .....	121, 163
compilare <b>gawk</b> per Cygwin .....	488	<b>CONVFMT</b> , variabile, e indici di vettore .....	185
compilare <b>gawk</b> per MS-Windows .....	486	coprocessi .....	106, 339
compilare <b>gawk</b> per VMS .....	488	coprocessi, chiusura .....	109
compilatore per <b>awk</b> , <b>awka</b> .....	495	coprocessi, <b>getline</b> da .....	88
<b>compl()</b> , funzione ( <b>gawk</b> ) .....	220	corpo, nei cicli .....	154
complemento a livello di bit .....	219, 220	corpo, nelle azioni .....	153
completamento dei comandi nel debugger .....	376	cortocircuito, operatori .....	137
composte, istruzioni, istruzioni di controllo e ..	153	<b>cos()</b> , funzione .....	196
comuni, estensioni, campi di un solo carattere ..	73	coseno .....	196
comuni, estensioni, <b>delete</b> per eliminare interi vettori .....	187	costanti numeriche .....	115
comuni, estensioni, <b>\x</b> , sequenza di protezione ..	51	costanti <i>regexp</i> .....	50, 117, 135
comuni, estensioni, <b>length()</b> applicato a un vettore .....	201	costanti <i>regexp</i> , barre vs. doppi apici .....	58
comuni, estensioni, <b>RS</b> come espressione regolare .....	65	costanti <i>regexp</i> , in <b>gawk</b> .....	117
concatenare .....	124	costanti <i>regexp</i> , vs. costanti stringa .....	58
conchiglie, mare .....	47	costanti stringa .....	115
<b>condition</b> , comando del debugger .....	368	costanti stringa, vs. costanti <i>regexp</i> .....	58
condizionali, espressioni .....	137	costanti, non decimali .....	331
condizione dei punti d'interruzione .....	368	costanti, tipi di .....	115
configurazione di <b>gawk</b> .....	485	creare un vettore da una stringa .....	203
configurazione di <b>gawk</b> , opzioni di .....	484	criteri di ricerca .....	145
confronto, espressioni di .....	130	criteri di ricerca vuoti .....	151
contare .....	300	criteri di ricerca, conteggi, in un profilo .....	345
conteggio riferimenti, ordinamento vettori ....	337	criteri di ricerca, costanti <i>regexp</i> come .....	146
continuazione di riga .....	137	criteri di ricerca, default .....	25
continuazione di riga, <b>gawk</b> .....	138	criteri di ricerca, espressioni come .....	145
continuazione di riga, in istruzione <b>print</b> .....	97	criteri di ricerca, espressioni di confronto come .....	146
continuazione di riga, nella C shell .....	27	criteri di ricerca, intervalli nei .....	147
<b>continue</b> , comando del debugger .....	369	criteri di ricerca, tipi di .....	145
<b>continue</b> , istruzione .....	158	criterio di ricerca <b>BEGIN</b> .....	148

criterio di ricerca <b>BEGIN</b> , e profilatura .....	344
criterio di ricerca <b>BEGIN</b> , eseguire programmi <b>awk</b> e .....	282
criterio di ricerca <b>BEGIN</b> , funzione definita dall'utente <b>assert()</b> e .....	250
criterio di ricerca <b>BEGIN</b> , istruzione <b>exit</b> e .....	161
criterio di ricerca <b>BEGIN</b> , programma <b>pwcat</b> .....	271
criterio di ricerca <b>BEGIN</b> , variabile <b>TEXTDOMAIN</b> e .....	353
criterio di ricerca <b>END</b> .....	148
criterio di ricerca <b>END</b> , e profilatura .....	344
criterio di ricerca <b>END</b> , funzione definita dall'utente <b>assert()</b> e .....	250
criterio di ricerca <b>END</b> , istruzione <b>exit</b> e .....	161
criterio di ricerca <b>END</b> , operatori e .....	148
<b>cs</b> , comando, operatore <b> &amp;</b> , confronto con ....	339
<b>cs</b> , comando, variabile d'ambiente <b>POSIXLY_CORRECT</b> .....	40
<b>ctime()</b> , funzione definita dall'utente .....	227
<b>custom.h</b> , file .....	486
<b>cut</b> , programma di utilità .....	282
<b>cut.awk</b> , programma .....	282
Cygwin, compilare <b>gawk</b> per .....	488

## D

<b>d</b> , comando del debugger (alias per <b>delete</b> ) ...	368
<b>D-o</b> .....	13
data e ora corrente del sistema .....	216
data e ora, formato stringa .....	215
data e ora, formattate .....	254
data e ora, si veda marcature temporali ..	214, 216
date, conversione in marcature temporali .....	216
date, informazioni relative alla localizzazione ..	351
<b>date</b> , programma di utilità GNU .....	214
<b>date</b> , programma di utilità POSIX .....	218
Davies, Stephen .....	12, 476
Day, Robert P.J. ....	13
<b>dcgettext()</b> , funzione ( <b>gawk</b> ) .....	224, 352
<b>dcgettext()</b> , funzione ( <b>gawk</b> ), portabilità e ...	356
<b>dcngettext()</b> , funzione ( <b>gawk</b> ) .....	224, 352
<b>dcngettext()</b> , funzione ( <b>gawk</b> ), portabilità e ..	356
<b>deadlocks</b> , vedi stalli .....	340
debug dei programmi <b>awk</b> .....	361
debug, doppio apice con nomi di file .....	108
debug, errori fatali, <b>printf</b> , stringhe di formato .....	103
debug, esempio di sessione .....	363
debug, <b>gawk</b> , segnalare bug .....	493
debug, istruzione <b>print</b> , omissione virgole .....	96
debug, stampare .....	106
debugger, comandi del, si veda comando del debugger .....	363
debugger, come avviarlo .....	363
debugger, completamento dei comandi nel .....	376
debugger, descrizione degli <i>stack</i> <i>frame</i> delle chiamate .....	372

debugger, dimensione della cronologia .....	373
debugger, elencare definizioni delle funzioni .....	372
debugger, elencare tutte le variabili locali .....	373
debugger, file della cronologia .....	373
debugger, leggere comandi da un file .....	374
debugger, mostrare argomenti delle funzioni ...	372
debugger, mostrare i punti d'osservazione .....	373
debugger, mostrare il nome del file sorgente corrente .....	372
debugger, mostrare punti d'interruzione .....	372
debugger, mostrare tutti i file sorgenti .....	373
debugger, mostrare variabili locali .....	372
debugger, numero di righe nella lista di default .....	373
debugger, opzioni del .....	373
debugger, prompt .....	373
debugger, ridirezionare l'output di <b>gawk</b> .....	373
debugger, salvataggio opzioni .....	373
debugger, tener traccia delle istruzioni .....	373
debugger, uscire dal .....	375
debugger, visualizzazioni automatiche .....	372
decremento, operatori di .....	129
<b>default</b> , parola chiave .....	156
definite dall'utente, funzioni, conteggi, in un profilo .....	345
definite dall'utente, variabili .....	119
definizione di funzioni .....	224
Deifik, Scott .....	12, 476, 494
<b>delete</b> , comando del debugger .....	368
<b>delete</b> , istruzione .....	187
<b>delete</b> , <i>vettore</i> .....	187
Demaille, Akim .....	12
descrittori di file .....	107
descrizione degli <i>stack frame</i> delle chiamate, nel debugger .....	372
differenze tra <b>awk</b> e <b>gawk</b> , argomenti di funzione ( <b>gawk</b> ) .....	195
differenze tra <b>awk</b> e <b>gawk</b> , campi di un solo carattere .....	73
differenze tra <b>awk</b> e <b>gawk</b> , chiamata indiretta di funzione .....	234
differenze tra <b>awk</b> e <b>gawk</b> , comando <b>getline</b> ....	83
differenze tra <b>awk</b> e <b>gawk</b> , continuazione di riga .....	138
differenze tra <b>awk</b> e <b>gawk</b> , costanti <i>regexp</i> .....	118
differenze tra <b>awk</b> e <b>gawk</b> , criteri di ricerca <b>BEGIN/END</b> .....	149
differenze tra <b>awk</b> e <b>gawk</b> , criteri di ricerca <b>BEGINFILE/ENDFILE</b> .....	150
differenze tra <b>awk</b> e <b>gawk</b> , directory sulla riga di comando .....	92
differenze tra <b>awk</b> e <b>gawk</b> , elementi dei vettori, eliminazione .....	187
differenze tra <b>awk</b> e <b>gawk</b> , espressioni regolari ...	61
differenze tra <b>awk</b> e <b>gawk</b> , funzione <b>close()</b> .....	110, 112
differenze tra <b>awk</b> e <b>gawk</b> , funzione <b>match()</b> ....	202

differenze tra <b>awk</b> e <b>gawk</b> , funzione <b>split()</b> ....	204
differenze tra <b>awk</b> e <b>gawk</b> , limitazioni di implementazione .....	88, 106
differenze tra <b>awk</b> e <b>gawk</b> , messaggi di errore...	107
differenze tra <b>awk</b> e <b>gawk</b> , operatori di input/output .....	88, 106
differenze tra <b>awk</b> e <b>gawk</b> , operazione di modulo-troncamento .....	124
differenze tra <b>awk</b> e <b>gawk</b> , proseguire dopo errore in input .....	91
differenze tra <b>awk</b> e <b>gawk</b> , separatori di record ..	65
differenze tra <b>awk</b> e <b>gawk</b> , stringhe .....	115
differenze tra <b>awk</b> e <b>gawk</b> , stringhe, memorizzazione .....	67
differenze tra <b>awk</b> e <b>gawk</b> , tempo limite per lettura .....	90
differenze tra <b>awk</b> e <b>gawk</b> , tra istruzioni <b>print</b> e <b>printf</b> .....	101
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>ARGIND</b> ....	166
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>BINMODE</b> .....	162, 487
differenze tra <b>awk</b> e <b>gawk</b> , variabile d'ambiente <b>AWKLIBPATH</b> .....	43
differenze tra <b>awk</b> e <b>gawk</b> , variabile d'ambiente <b>AWKPATH</b> .....	42
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>ERRNO</b> .....	166
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>FIELDWIDTHS</b> .....	163
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>FPAT</b> .....	163
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>FUNCTAB</b> ....	167
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>IGNORECASE</b> .....	163
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>LINT</b> .....	164
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>RT</b> .....	170
differenze tra <b>awk</b> e <b>gawk</b> , variabile <b>TEXTDOMAIN</b> .....	165
differenze tra <b>awk</b> e <b>gawk</b> , variabili <b>ARGC/ARGV</b> .....	173
differenze tra <b>awk</b> e <b>gawk</b> , variabili <b>RS/RT</b> .....	66
differenze tra <b>awk</b> e <b>gawk</b> , vettore <b>PROCINFO</b> ....	167
differenze tra <b>awk</b> e <b>gawk</b> , vettore <b>SYMTAB</b> .....	170
differenze tra <b>gawk</b> e <b>awk</b> .....	201
dinamiche, estensioni .....	395
directory, ricerca .....	328
directory, ricerca di estensioni caricabili .....	43
directory, ricerca di file sorgente .....	42
directory, riga di comando .....	92
direttiva <b>@include</b> .....	45
direttiva <b>@load</b> .....	47
disabilitare punto d'interruzione .....	368
<b>disable</b> , comando del debugger .....	368
<b>display</b> , comando del debugger .....	370
distinzione maiuscolo/minuscolo, <b>gawk</b> .....	61
distinzione maiuscolo/minuscolo, programmi di esempio .....	245
distribuzione di <b>gawk</b> .....	480
dividere in un vettore una stringa .....	203

divisione .....	124
<b>do-while</b> .....	154
<b>do-while</b> , istruzione, uso di espressioni regolari in .....	49
documentazione, di programmi <b>awk</b> .....	246
documentazione, online .....	10
documenti, ricerca in .....	302
dollaro (\$), incrementare campi e vettori .....	129
dollaro (\$), operatore di campo \$ .....	67, 140
dollaro (\$), operatore <b>regexp</b> .....	53
doppio apice ("), in costanti <b>regexp</b> .....	58
doppio apice ("), nei comandi shell .....	22
<b>down</b> , comando del debugger .....	372
Drepper, Ulrich .....	12
Duman, Patrice .....	13
<b>dump</b> , comando del debugger .....	374
<b>dupword.awk</b> , programma .....	302

## E

e commerciale (&), funzioni <b>gsub()</b> / <b>gensub()</b> / <b>sub()</b> e .....	207
e commerciale (&), operatore <b>&amp;&amp;</b> .....	137, 141
<b>e</b> , comando del debugger (alias per <b>enable</b> ) ..	368
e.c., si veda estensioni comuni .....	10
<b>EBCDIC</b> .....	252
editori di flusso .....	76, 316
effetti collaterali .....	125, 128, 130
effetti collaterali delle istruzioni .....	152
effetti collaterali, chiamate di funzione .....	139
effetti collaterali, espressioni condizionali .....	138
effetti collaterali, espressioni di assegnamento .....	126
effetti collaterali, funzione <b>asort()</b> .....	337
effetti collaterali, indicizzazione di vettori .....	179
effetti collaterali, operatori booleani .....	136
effetti collaterali, operatori di decremento/incremento .....	128
effetti collaterali, variabile <b>FILENAME</b> .....	88
effettivo, <i>ID di gruppo</i> dell'utente di <b>gawk</b> .....	167
<b>egrep</b> , programma di utilità .....	55, 286
<b>egrep.awk</b> , programma .....	287
elaborazione dati .....	511
elementi di collazione .....	56, 57
elementi di un vettore .....	179
elementi di vettore non assegnati .....	179
elementi di vettore vuoti .....	179
elementi di vettori, assegnare valori .....	180
elementi di vettori, controlli <i>lint</i> per .....	187
elementi di vettori, eliminazione di .....	187
elementi di vettori, ordine di accesso da parte dell'operatore <b>in</b> .....	182
elementi di vettori, visitare .....	181
elementi inesistenti di un vettore .....	179
elenare definizioni delle funzioni, nel debugger .....	372
elenare tutte le variabili locali, nel debugger ..	373
elenchi di caratteri in un'espressione regolare ..	55

- eliminare elementi di vettori ..... 187
- eliminare interi vettori ..... 187
- email, indirizzo per segnalare bug,  
  **bug-gawk@gnu.org** ..... 493
- EMRED ..... 341
- enable**, comando del debugger ..... 368
- end**, comando del debugger ..... 368
- END, criterio di ricerca ..... 148
- END, criterio di ricerca, criteri di  
  ricerca booleani e ..... 147
- END, criterio di ricerca, e profilatura ..... 344
- END, criterio di ricerca, funzione definita  
  dall'utente **assert()** e ..... 250
- END, criterio di ricerca, istruzione **exit** e ..... 161
- END, criterio di ricerca, istruzione **print** e ..... 149
- END, criterio di ricerca, istruzioni  
  **next/nextfile** e ..... 150, 160
- END, criterio di ricerca, operatori e ..... 148
- ENDFILE, criterio di ricerca ..... 150
- ENDFILE, criterio di ricerca, criteri di  
  ricerca booleani e ..... 147
- endgrent()**, funzione (libreria C) ..... 276
- endgrent()**, funzione definita dall'utente ..... 276
- endpwent()**, funzione (libreria C) ..... 272
- endpwent()**, funzione definita dall'utente ..... 272
- English, Steve ..... 331
- ENVIRON, vettore ..... 166
- epoch, definizione di ..... 519
- ERE (espressioni regolari estese) ..... 55
- ERRNO, variabile ..... 166, 342
- ERRNO, variabile, con comando **getline** ..... 83
- ERRNO, variabile, con criterio di  
  ricerca **BEGINFILE** ..... 150
- ERRNO, variabile, con funzione **close()** ..... 112
- errore in input, possibilità di proseguire ..... 91
- errore, output ..... 107
- errori, gestione degli ..... 107
- esadecimali, numeri ..... 115
- esadecimali, valori, abilitare  
  l'interpretazione di ..... 37
- esaminare i campi ..... 67
- esecuzione di un solo passo, nel debugger ..... 369
- esempi di programmi **awk** ..... 281
- esempio di definizione di funzione ..... 226
- esempio di estensione ..... 434
- esempio di sessione di debug ..... 363
- espansione della cronologia, nel debugger ..... 376
- esponenziale ..... 196
- espressione di ricerca ..... 145
- espressioni ..... 115
- espressioni booleane ..... 136
- espressioni booleane, come criteri di ricerca ..... 146
- espressioni condizionali ..... 137
- espressioni di assegnamento ..... 126
- espressioni di confronto ..... 130
- espressioni di confronto, come  
  criteri di ricerca ..... 146
- espressioni di confronto, stringa vs. *regex* ..... 135
- espressioni di intervallo, (*regex*) ..... 54, 55
- espressioni regolari ..... 49
- espressioni regolari calcolate ..... 57
- espressioni regolari come separatori di campo ..... 72
- espressioni regolari dinamiche ..... 57
- espressioni regolari dinamiche, contenenti  
  dei ritorni a capo ..... 59
- espressioni regolari estese (ERE) ..... 55
- espressioni regolari, ancora nelle ..... 53
- espressioni regolari, come  
  criteri di ricerca ..... 49, 145, 146
- espressioni regolari, come separatori di campo .. 72
- espressioni regolari, come separatori di record .. 65
- espressioni regolari,  
  corrispondenza più a sinistra ..... 57
- espressioni regolari, costanti, si veda  
  costanti *regex* ..... 50
- espressioni regolari, espressioni di intervallo e ... 38
- espressioni regolari, **gawk**, opzioni sulla  
  riga di comando ..... 60
- espressioni regolari, maiuscolo/minuscolo .. 60, 163
- espressioni regolari, operatori ..... 49, 52
- espressioni regolari, operatori, **gawk** ..... 59
- espressioni regolari, operatori, per buffer ..... 60
- espressioni regolari, operatori, per parole ..... 59
- espressioni regolari, operatori, precedenza di ... 55
- espressioni regolari, ricerca di ..... 286
- espressioni tra parentesi ..... 53
- espressioni tra parentesi quadre ..... 55
- espressioni tra parentesi quadre,  
  classi di caratteri ..... 55
- espressioni tra parentesi quadre,  
  classi di equivalenza ..... 57
- espressioni tra parentesi quadre,  
  complementate ..... 53
- espressioni tra parentesi quadre,  
  elementi di collazione ..... 56, 57
- espressioni tra parentesi quadre,  
  espressioni di intervallo ..... 55
- espressioni tra parentesi quadre, non-ASCII .... 56
- espressioni, ricerca di corrispondenze, si veda  
  espressioni di confronto ..... 130
- espressioni, selezionare ..... 137
- estensione API, numero di versione ..... 169
- estensione API, variabili informative ..... 432
- estensione **chdir()** ..... 445
- estensione **Chr** ..... 451
- estensione **fnmatch()** ..... 448
- estensione **fork()** ..... 449
- estensione **fts()** ..... 446
- estensione **gawk**, versione API ..... 431
- estensione **gettimeofday()** ..... 454
- estensione **inplace** ..... 449
- estensione **Ord** ..... 451
- estensione **reada()** ..... 453
- estensione **readdir** ..... 451
- estensione **readfile()** ..... 454
- estensione **revoutput** ..... 452

estensione <b>revtwoway</b> .....	452
estensione <b>sleep()</b> .....	454
estensione <b>stat()</b> .....	445
estensione <b>testtext</b> .....	455
estensione <b>wait()</b> .....	449
estensione <b>waitpid()</b> .....	449
estensione <b>writea()</b> .....	453
estensione, esempio .....	434
estensione, registrazione di .....	405
estensioni caricate dinamicamente .....	395
estensioni comuni, \x, sequenza di protezione... 51	
estensioni comuni, campi di un solo carattere... 73	
estensioni comuni, file speciale /dev/stderr... 108	
estensioni comuni, file speciale /dev/stdin... 108	
estensioni comuni, file speciale /dev/stdout... 108	
estensioni comuni, funzione <b>fflush()</b> .....	211
estensioni comuni, <b>length()</b> applicato a un vettore .....	201
estensioni comuni, operatore <b>**</b> .....	124
estensioni comuni, operatore <b>**=</b> .....	128
estensioni comuni, parola chiave <b>func</b> .....	226
estensioni comuni, RS come espressione regolare .....	65
estensioni comuni, variabile <b>BINMODE</b> .....	487
estensioni comuni, <b>delete</b> per eliminare interi vettori .....	187
estensioni comuni, operatore <b>**=</b> .....	128
estensioni distribuite con <b>gawk</b> .....	445
estensioni <b>gawk</b> , convenzioni di programmazione .....	437
estensioni nella modalità compatibile di ( <b>gawk</b> ) .....	464
estensioni, accesso alle variabili globali .....	417
estensioni, allocare memoria per .....	403
estensioni, API delle .....	398
estensioni, Brian Kernighan <b>awk</b> .....	463, 473
estensioni, caricamento, direttiva <b>@load</b> .....	47
estensioni, come trovarle .....	434
estensioni, dove trovarle .....	455
estensioni, <b>gawkextlib</b> .....	455
estensioni, in <b>gawk</b> , non in POSIX <b>awk</b> .....	464
estensioni, manipolazione di vettori .....	421
estensioni, <b>mawk</b> .....	473
estensioni, percorso di ricerca per .....	434
estensioni, stampare messaggi dalle .....	415
estese, opzioni .....	33
estrarre programma da file sorgente Texinfo ... 312	
estrazione di stringhe marcate (internazionalizzazione) .....	354
etichette per lettera, stampare .....	308
<b>eval</b> , comando del debugger .....	370
<b>exit</b> , codice di ritorno, in VMS .....	491
<b>exit</b> , comando del debugger .....	375
<b>exit</b> , istruzione .....	161
<b>exp()</b> , funzione .....	196
<b>expand</b> , programma .....	25
Expat, libreria per analizzare XML .....	455
<b>extract.awk</b> , programma .....	313

## F

f, comando del debugger (alias per <b>frame</b> )	372
falso, valore logico (zero o stringa nulla)	130
FDL (Free Documentation License)	543
Fenlason, Jay	6, 476
<b>fflush()</b> , funzione	210
<b>FIELDWIDTHS</b> , variabile	77, 163
file <b>.gmo</b>	350
file <b>.gmo</b> , specificare la directory di	350, 352
file <b>.mo</b> , conversione da <b>.po</b>	358
file <b>.po</b>	350, 354
file <b>.po</b> , conversione in <b>.mo</b>	358
file <b>.pot</b>	350
file <b>awkprof.out</b>	343
file <b>awkvars.out</b>	35
file dei gruppi	272
file delle password	268
file di <b>mail-list</b>	23
file di registro ( <i>log</i> ), marcature temporali nei	214
file in input	63
file in input, assegnamenti di variabile e	40
file in input, contare elementi nel	300
file in input, eseguire <b>awk</b> senza usarli	18
file in input, leggere	63
file <b>inventory-shipped</b>	24
file oggetto portabili, generare	36
file sorgente, percorso di ricerca per	328
file speciali <b>/inet/...</b> (in <b>gawk</b> )	341
file speciali <b>/inet4/...</b> (in <b>gawk</b> )	341
file speciali <b>/inet6/...</b> (in <b>gawk</b> )	341
file Texinfo, estrarre programma da	312
file, chiusura	210
file, descrittori, si veda descrittori di file	107
file, elaborazione, variabile <b>ARGIND</b> e	166
file, file speciali <b>/dev/...</b>	108
file, gestione di	257
file, gestione di, limiti dei file-dati	258
file, inizializzazione e pulizia	258
file, leggere un	259
file, lettura dei record multiriga	80
file, <i>message object</i> (.mo)	350
file, <i>message object</i> (.mo), conversione da <i>portable object</i> file	358
file, <i>message object</i> (.mo), specificare la directory di	350, 352
file, multipli, duplicare l'output su	294
file, nomi di, assegnamenti di variabile visti come	262
file, non elaborare	261
file, passaggi multipli su	41
file, <i>portable object</i> (.po)	350, 354
file, <i>portable object</i> (.po), conversione in <i>message object</i> file	358
file, <i>portable object template</i> (.pot)	350
file, programmi <b>awk</b> in	19
file, ricercare espressioni regolari nei	286

- file, rileggere un ..... 259
- file, splitting ..... 292
- FILENAME, variabile ..... 63, 167
- FILENAME, variabile, impostare con `getline` ..... 88
- `finish`, comando del debugger ..... 369
- Fish, Fred ..... 476
- `flag` [indicatore], variabili ..... 137
- flag, variabili di tipo ..... 294
- flusso, editori di ..... 316
- `fnmatch()`, estensione ..... 448
- FNR, variabile ..... 63, 167
- FNR, variabile, modifica di ..... 172
- `for`, istruzione ..... 155
- `for`, istruzione, esecuzione di
  - cicli su un vettore ..... 181
- `fork()`, estensione ..... 449
- formati numerici di output ..... 98
- formato stringa data e ora ..... 215
- formato stringa marcature temporali ..... 215
- formato, specificatori di, frammisti a
  - specificatori posizionali non standard ..... 355
- formato, specificatori di, funzione
  - `strftime()` di (`gawk`) ..... 216
- formattare l'output ..... 98
- formattare stringhe ..... 204
- formattatore incredibilmente duttile (`awf`) ..... 521
- fornire una lista di tutte le variabili
  - del programma ..... 35
- FPAT, variabile ..... 79, 163
- `frame`, comando del debugger ..... 372
- Free Documentation License (FDL) ..... 543
- Free Software Foundation (FSF) .. 10, 479, 521, 522
- FreeBSD ..... 527
- FS, contenente ^ ..... 73
- FS, nei record multiriga ..... 81
- FS, variabile ..... 71, 163
- FS, variabile, cambiare il valore di una ..... 72
- FS, variabile, come carattere TAB ..... 38
- FS, variabile, come stringa nulla ..... 74
- FS, variabile, eseguire programmi `awk` e ..... 282
- FS, variabile, impostare da riga di comando ..... 74
- FS, variabile, l'opzione `--field-separator` e ... 33
- FSF (Free Software Foundation) .. 10, 479, 521, 522
- `fts()`, estensione ..... 446
- FUNCTAB, vettore ..... 167
- funzionalità avanzate, campi di
  - larghezza costante ..... 77
- funzionalità avanzate, processi,
  - comunicare con ..... 339
- funzionalità avanzate,
  - programmazione di rete ..... 341
- funzionalità avanzate, specificare il
  - contenuto dei campi ..... 79
- funzionalità deprecate ..... 47
- funzionalità non documentate ..... 47
- funzionalità, aggiungere a `gawk` ..... 443, 500
- funzione `asort()` (`gawk`),
  - ordinamento di vettori ..... 336
- funzione `asorti()` (`gawk`),
  - ordinamento di vettori ..... 336
- funzione `assert()` (libreria C) ..... 249
- funzione `bindtextdomain()`
  - (`gawk`), portabilità e ..... 356
- funzione `bindtextdomain()` (libreria C) ..... 350
- funzione `close()`, *pipe* bidirezionali e ..... 340
- funzione `close()`, portabilità ..... 110
- funzione `close()`, valore di ritorno ..... 112
- funzione `dcgettext()` (`gawk`), portabilità e ... 356
- funzione `dcngettext()` (`gawk`), portabilità e ... 356
- funzione definita dall'utente, `_gr_init()` ..... 274
- funzione definita dall'utente, `_ord_init()` ..... 252
- funzione definita dall'utente, `_pw_init()` ..... 270
- funzione definita dall'utente, `a_fine_file()` .. 259
- funzione definita dall'utente,
  - `a_inizio_file()` ..... 259
- funzione definita dall'utente, `assert()` ..... 249
- funzione definita dall'utente, `bits2str()` ..... 220
- funzione definita dall'utente, `chr()` ..... 252
- funzione definita dall'utente, `cliff_rand()` .... 251
- funzione definita dall'utente, `ctime()` ..... 227
- funzione definita dall'utente, `endgrent()` ..... 276
- funzione definita dall'utente, `endpwent()` ..... 272
- funzione definita dall'utente, `getgrent()` .. 272, 276
- funzione definita dall'utente, `getgrgid()` ..... 276
- funzione definita dall'utente, `getgrnam()` ..... 276
- funzione definita dall'utente, `getgruser()` ..... 276
- funzione definita dall'utente,
  - `getlocaltime()` ..... 254
- funzione definita dall'utente, `getopt()` ..... 265
- funzione definita dall'utente, `getpwent()` .. 268, 272
- funzione definita dall'utente, `getpwnam()` ..... 271
- funzione definita dall'utente, `getpwuid()` ..... 271
- funzione definita dall'utente, `join()` ..... 253
- funzione definita dall'utente, `ord()` ..... 252
- funzione definita dall'utente, `readfile()` ..... 256
- funzione definita dall'utente, `rev()` ..... 227
- funzione definita dall'utente, `rewind()` ..... 259
- funzione definita dall'utente, `round()` ..... 250
- funzione definita dall'utente, `walk_array()` .... 277
- funzione della shell `gawklibpath_append` ..... 484
- funzione della shell `gawklibpath_default` ..... 484
- funzione della shell `gawklibpath_prepend` ..... 484
- funzione della shell `gawkpath_append` ..... 484
- funzione della shell `gawkpath_default` ..... 484
- funzione della shell `gawkpath_prepend` ..... 484
- funzione `endgrent()` (libreria C) ..... 276
- funzione `endpwent()` (libreria C) ..... 272
- funzione `gensub()` (`gawk`),
  - protezione caratteri ..... 207
- funzione `getaddrinfo()` (libreria C) ..... 342
- funzione `getgrent()` (libreria C) ..... 272, 276
- funzione `getgrgid()` (libreria C) ..... 276
- funzione `getgrnam()` (libreria C) ..... 276
- funzione `getgruser()` (libreria C) ..... 276
- funzione `getopt()` (libreria C) ..... 263

funzione <code>getpwent()</code> (libreria C) .....	268, 271
funzione <code>getpwnam()</code> (libreria C) .....	271
funzione <code>getpwuid()</code> (libreria C) .....	271
funzione <code>gettext()</code> (libreria C) .....	350
funzione <code>gsub()</code> , argomenti di .....	206
funzione <code>gsub()</code> , protezione caratteri .....	207
funzione indefinita, controlli <i>lint</i> per .....	232
funzione <code>intdiv</code> .....	196
funzione <code>match()</code> , variabili <code>RSTART</code> / <code>RLENGTH</code> ..	202
funzione <code>sort()</code> , ordinamento di vettori .....	336
funzione <code>split()</code> , eliminazione di elementi di vettori .....	187
funzione <code>sprintf()</code> , istruzioni <code>print/printf</code> e .....	250
funzione <code>sprintf()</code> , variabile <code>OFMT</code> e .....	164
funzione <code>sub()</code> , argomenti di .....	206
funzione <code>sub()</code> , protezione caratteri .....	207
funzione <code>textdomain()</code> (libreria C) .....	350
funzione, esempio di definizione di .....	226
funzione, puntatori a .....	234
funzioni definite dall'utente .....	224
funzioni definite dall'utente, chiamare .....	228
funzioni definite dall'utente, conteggi, in un profilo .....	345
funzioni definite dall'utente, istruzioni <code>next/nextfile</code> e .....	160, 161
funzioni definite dall'utente, libreria di .....	245
funzioni di input/output .....	210
funzioni di libreria .....	245
funzioni di manipolazione di stringhe .....	198
funzioni di tempo .....	214
funzioni di traduzione di stringhe .....	224
funzioni indefinite .....	232
funzioni numeriche .....	196
funzioni per la manipolazione di bit .....	219
funzioni predefinite .....	138, 195
funzioni predefinite, ordine di valutazione ..	195
funzioni ricorsive .....	226
funzioni, convenzioni di programmazione, nella scrittura di .....	225
funzioni, definizione di .....	224
funzioni, libreria di, arrotondamento di numeri .....	250
funzioni, libreria di, asserzioni .....	249
funzioni, libreria di, gestione delle ore del giorno .....	254
funzioni, libreria di, gestire file di dati .....	257
funzioni, libreria di, leggera la lista degli utenti .....	268
funzioni, libreria di, leggere la lista dei gruppi .....	272
funzioni, libreria di, libreria C .....	263
funzioni, libreria di, numeri casuali Cliff .....	251
funzioni, libreria di, opzioni sulla riga di comando .....	263
funzioni, libreria di, trasformare vettori in stringhe .....	253

funzioni, libreria di, valori di carattere come numeri .....	251
funzioni, libreria di, vettori associativi e .....	247
funzioni, librerie di, programma di esempio per usare .....	317
funzioni, nomi di .....	225
funzioni, vettori come parametri di .....	231

## G

G., Daniel Richard .....	12, 494
Garfinkle, Scott .....	476
gawk e l'intelligenza artificiale .....	481
gawk, aggiungere funzionalità a .....	443, 500
gawk, andare a capo .....	28
gawk, argomenti di funzione e .....	195
gawk, awk e .....	5, 7
gawk, caratteri di controllo del formato ....	99, 100
gawk, classi di caratteri e .....	57
gawk, codice sorgente, ottenere il .....	479
gawk, configurazione di .....	485
gawk, configurazione, opzioni di .....	484
gawk, continuazione di riga in .....	138
gawk, costanti <i>regexp</i> e .....	117
gawk, data e ora (marcature temporali) .....	214
gawk, distribuzione di .....	480
gawk, espressioni di intervallo e .....	54
gawk, espressioni regolari, differenza maiuscolo/minuscolo .....	61
gawk, espressioni regolari, operatori .....	59
gawk, espressioni regolari, precedenza .....	55
gawk, estensioni, disabilitare .....	38
gawk, funzionalità avanzate .....	331
gawk, funzioni di traduzione di stringhe .....	224
gawk, installare .....	479
gawk, internazionalizzazione e, si veda internazionalizzazione .....	349
gawk, istruzione <code>break</code> in .....	158
gawk, istruzione <code>continue</code> in .....	159
gawk, lista di contributori a .....	475
gawk, nomi di file in .....	108
gawk, numeri esadecimali e .....	116
gawk, numeri ottali e .....	116
gawk, operatore limite-di-parola .....	60
gawk, operazioni a livello di bit in .....	220
gawk, opzioni sulla riga di comando, ed espressioni regolari .....	60
gawk, problemi di implementazione .....	499
gawk, problemi di implementazione, compatibilità all'indietro .....	499
gawk, problemi di implementazione, debug .....	499
gawk, problemi di implementazione, <i>pipe</i> .....	106
gawk, problemi di implementazioni, limiti .....	88
gawk, programma, profilazione dinamica .....	346
gawk, RT variabile in .....	83
gawk, separatori di campo e .....	163
gawk, separazione in campi e .....	78
gawk, sequenze di protezione .....	52

- `gawk`, si veda anche `awk` ..... 5
  - `gawk`, stile di codifica in ..... 500
  - `gawk`, uso di ..... 5
  - `gawk`, variabile `ARGIND` in ..... 40
  - `gawk`, variabile `ERRNO` in ..... 83, 112, 150, 166, 342
  - `gawk`, variabile `FIELDWIDTHS` in ..... 77, 163
  - `gawk`, variabile `FPAT` in ..... 79, 163
  - `gawk`, variabile `IGNORECASE` in ... 61, 163, 179, 199, 338
  - `gawk`, variabile `LINT` in ..... 164
  - `gawk`, variabile `RT` in ..... 65, 170
  - `gawk`, variabile `TEXTDOMAIN` in ..... 165
  - `gawk`, variabili predefinite e ..... 162
  - `gawk`, versione di ..... 169
  - `gawk`, versione MS-Windows di ..... 487
  - `gawk`, versione VMS di ..... 488
  - `gawk`, versioni di, informazioni su, stampa ..... 39
  - `gawk`, vettore `FUNCTAB` in ..... 167
  - `gawk`, vettore `PROCINFO` in ..... 167, 215, 341
  - `gawk`, vettore `SYMTAB` in ..... 170
  - `gawkextlib`, estensioni ..... 455
  - `gawkextlib`, progetto ..... 455
  - `gawklibpath_append`, funzione della shell ..... 484
  - `gawklibpath_default`, funzione della shell .... 484
  - `gawklibpath_prepend`, funzione della shell .... 484
  - `gawkpath_append`, funzione della shell ..... 484
  - `gawkpath_default`, funzione della shell ..... 484
  - `gawkpath_prepend`, funzione della shell ..... 484
  - General Public License (GPL) ..... 522
  - General Public License, si veda GPL ..... 10
  - generare data e ora ..... 215
  - `gensub()`, funzione (`gawk`) ..... 118, 199
  - `gensub()`, funzione (`gawk`), protezione caratteri ..... 207
  - gestione di file ..... 257
  - gestione errori ..... 107
  - gestione errori, variabile `ERRNO` e ..... 166
  - `getaddrinfo()`, funzione (libreria C) ..... 342
  - `getgrent()`, funzione (libreria C) ..... 272, 276
  - `getgrent()`, funzione definita dall'utente .. 272, 276
  - `getgrgid()`, funzione (libreria C) ..... 276
  - `getgrgid()`, funzione definita dall'utente ..... 276
  - `getgrnam()`, funzione (libreria C) ..... 276
  - `getgrnam()`, funzione definita dall'utente ..... 276
  - `getgruser()`, funzione (libreria C) ..... 276
  - `getgruser()`, funzione definita dall'utente .... 276
  - `getline` da un file ..... 85
  - `getline` in una variabile ..... 85
  - `getline`, comando ..... 63
  - `getline`, comando, coprocessi, usare dal ... 88, 109
  - `getline`, comando, criteri di ricerca `BEGINFILE/ENDFILE` e ..... 151
  - `getline`, comando, funzione definita dall'utente, `_gr_init()` ..... 274
  - `getline`, comando, funzione definita dall'utente, `_pw_init()` ..... 271
  - `getline`, comando, input esplicito con ..... 83
  - `getline`, comando, risoluzione di problemi .... 261
  - `getline`, comando, stalli e ..... 340
  - `getline`, comando, valori di ritorno ..... 83
  - `getline`, comando, variabile `FILENAME` e ..... 88
  - `getline`, comando, varianti ..... 89
  - `getlocaltime()`, funzione definita dall'utente ..... 254
  - `getopt()`, funzione (libreria C) ..... 263
  - `getopt()`, funzione definita dall'utente ..... 265
  - `getpwent()`, funzione (libreria C) ..... 268, 271
  - `getpwent()`, funzione definita dall'utente .. 268, 272
  - `getpwnam()`, funzione (libreria C) ..... 271
  - `getpwnam()`, funzione definita dall'utente ..... 271
  - `getpwuid()`, funzione (libreria C) ..... 271
  - `getpwuid()`, funzione definita dall'utente ..... 271
  - `gettext()`, funzione (libreria C) ..... 350
  - `gettext`, libreria ..... 349
  - `gettext`, libreria, categorie di localizzazione ... 350
  - `gettimeofday()`, estensione ..... 454
  - `git`, programma di utilità ..... 455, 495, 499, 501
  - Git, uso per il codice sorgente di `gawk` ..... 503
  - globali, variabili, per funzioni di libreria ..... 246
  - GNITS mailing list ..... 12
  - GNU `awk`, si veda `gawk` ..... 5
  - GNU Free Documentation License ..... 543
  - GNU General Public License ..... 522
  - GNU Lesser General Public License ..... 523
  - GNU, opzioni estese ..... 33
  - GNU, opzioni estese, stampare una lista di .... 36
  - GNU, Progetto ..... 10
  - GNU/Linux ..... 11, 357, 527
  - Gordon, Assaf ..... 477
  - GPL (General Public License) ..... 10, 522
  - GPL (General Public License), stampare ..... 35
  - `grcat`, programma C ..... 273
  - Grigera, Juan ..... 476
  - gruppi supplementari Unix con `gawk` ..... 169
  - gruppi, file dei ..... 272
  - gruppi, lista dei, leggere la ..... 272
  - gruppi, informazioni su ..... 272
  - gruppo di discussione `comp.lang.awk` ..... 494
  - `gsub()`, funzione ..... 118, 200
  - `gsub()`, funzione, argomenti di ..... 206
  - `gsub()`, funzione, protezione caratteri ..... 207
  - guidato-dai-dati, linguaggio di programmazione ..... 512
- ## H
- `h`, comando del debugger (alias per `help`) ..... 375
  - Hankerson, Darrel ..... 12, 476
  - Haque, John ..... 477
  - Hartholz, Elaine ..... 12
  - Hartholz, Marshall ..... 12
  - Hasegawa, Isamu ..... 477
  - `help`, comando del debugger ..... 375
  - `histsort.awk`, programma ..... 312
  - Hughes, Phil ..... 12
  - HUP, segnale, per profilazione dinamica ..... 346

## I

- i, comando del debugger (alias per **info**)..... 372
- id, programma di utilità..... 290
- id.awk, programma..... 291
- ID di gruppo dell'utente **gawk**..... 168
- ID effettivo dell'utente di **gawk**..... 168
- identificativi in un programma..... 168
- if, istruzione..... 153
- if, istruzione, azioni, modificabili..... 147
- if, istruzione, uso di espressioni regolari in..... 49
- igawk.sh, programma..... 319
- ignorare un punto d'interruzione..... 368
- ignore, comando del debugger..... 368
- IGNORECASE, variabile..... 163
- IGNORECASE, variabile, con operatori ~ e !~..... 61
- IGNORECASE, variabile, e funzioni di
  - ordinamento dei vettori..... 338
- IGNORECASE, variabile, e indici dei vettori..... 179
- IGNORECASE, variabile, nei
  - programmi di esempio..... 245
- Illumos..... 496
- Illumos, awk conforme a POSIX e..... 496
- implementazione Java di awk..... 496
- implementazione, problemi, gawk, limiti..... 88
- implementazioni di awk..... 494
- impostare directory con catalogo
  - messaggi tradotti..... 224
- impostare un punto d'interruzione..... 367
- impostare un punto d'osservazione..... 371
- in, operatore..... 133, 141, 156
- in, operatore, ordine di accesso dei vettori.... 182
- in, operatore, uso in cicli..... 181
- in, operatore, verificare se un elemento
  - esiste in un vettore..... 179
- in, operatore, verificare se un elemento esiste in
  - un vettore multidimensionale..... 188
- includere file, direttiva @include..... 45
- incredibile assembler (aaa) scritto in awk..... 515
- incremento, operatori di..... 128
- indefinite, funzioni..... 232
- index(), funzione..... 200
- indici di vettore, variabili non
  - inizializzate come..... 186
- indici di vettori multidimensionali..... 188
- indici di vettori
  - multidimensionali, visitare gli..... 189
- indici di vettori, numeri come..... 185
- indici di vettori, ordinamento per..... 198
- indici, separatori di..... 165
- indicizzare i vettori..... 178
- indiretta, chiamata di funzione..... 234
- indirizzo email per segnalare bug,
  - bug-gawk@gnu.org..... 493
- individuare la stringa nulla..... 207
- individuare la stringa più lunga da sinistra..... 81
- individuazione, espressioni di, si veda
  - espressioni di confronto..... 130
- info, comando del debugger..... 372
- informazioni di tipo monetario,
  - localizzazione..... 351
- inizializzazione automatica..... 27
- inizializzazione generazione di numeri casuali.. 197
- inplace, estensione..... 449
- input esplicito..... 83
- input file, esempi..... 23
- input, analizzatore di, personalizzato..... 408
- input, dati non decimali..... 331
- input, file in, chiusura..... 109
- input, file in, si veda file in input..... 18, 80
- input, pipeline..... 86
- input, record multiriga..... 80
- input, ridirezione dell'..... 85
- input, standard..... 18, 107
- input, suddividere in record..... 63
- input/output bidirezionale..... 339
- input/output binario..... 162
- input/output, bufferizzazione..... 340
- input/output, dalle regole BEGIN ed END..... 149
- input/output, funzioni di..... 210
- insiemi di caratteri (codifiche
  - macchina di caratteri)..... 517
- insiemi di caratteri, si veda anche espressioni
  - tra parentesi quadre..... 53
- insonnia, cura per..... 303
- installare gawk..... 479
- installare gawk su sistemi operativi per PC.... 486
- installare gawk su VMS..... 488
- int(), funzione..... 196
- INT, segnale (MS-Windows)..... 346
- intdiv(), funzione..... 196
- intelligenza artificiale, gawk e..... 481
- interagire con altri programmi..... 212
- interi a precisione arbitraria..... 389
- interi senza segno..... 379
- internazionalizzare un programma..... 349
- internazionalizzazione..... 224
- internazionalizzazione di programmi awk.. 349, 352
- internazionalizzazione, localizzazione..... 165, 349
- internazionalizzazione, localizzazione,
  - categorie di localizzazione..... 350
- internazionalizzazione, localizzazione,
  - classi di caratteri..... 57
- internazionalizzazione, localizzazione, gawk e.. 349
- internazionalizzazione,
  - localizzazione, portabilità e..... 356
- internazionalizzazione, localizzazione,
  - stringhe marcate..... 352
- interruzione visualizzazioni
  - automatiche, nel debugger..... 371
- interruzioni di riga..... 28
- intervalli di ricerca..... 147
- intervalli di ricerca, continuazione di riga e.... 148
- isarray(), funzione (gawk)..... 223
- ISO..... 523
- ISO 8859-1..... 517

ISO Latin-1.....	517
istruzione <b>break</b> .....	157
istruzione <b>continue</b> .....	158
istruzione <b>delete</b> .....	187
istruzione <b>exit</b> .....	161
istruzione <b>for</b> .....	155
istruzione <b>if</b> .....	153
istruzione <b>next</b> .....	159
istruzione <b>nextfile</b> .....	160
istruzione <b>print</b> .....	95
istruzione <b>print</b> , continuazione di riga e.....	97
istruzione <b>print</b> , funzione <b>sprintf()</b> e.....	250
istruzione <b>print</b> , operatori I/O nell'.....	141
istruzione <b>print</b> , si veda anche	
ridirezione dell'output.....	104
istruzione <b>print</b> , variabile <b>OFMT</b> e.....	164
istruzione <b>print</b> , virgole, omettere.....	96
istruzione <b>printf</b> .....	95, 98
istruzione <b>printf</b> , colonne, allineamento.....	96
istruzione <b>printf</b> , funzione <b>sprintf()</b> e.....	250
istruzione <b>printf</b> , lettere di	
controllo del formato.....	99
istruzione <b>printf</b> , modificatori.....	101
istruzione <b>printf</b> , operatori I/O nell'.....	141
istruzione <b>printf</b> , si veda anche	
ridirezione dell'output.....	104
istruzione <b>printf</b> , sintassi dell'.....	98
istruzione <b>printf</b> ,	
specificatori posizionali.....	101, 354
istruzione <b>printf</b> , specificatori posizionali,	
frammisti a formati standard.....	355
istruzione <b>return</b> , in funzioni	
definite dall'utente.....	232
istruzione <b>while</b> .....	154
istruzioni composte, istruzioni di controllo e...	153
istruzioni di controllo.....	153
istruzioni multiple.....	29
istruzioni, effetti collaterali delle.....	152
istruzioni, tener traccia delle, nel debugger....	373

## J

Jacobs, Andrew.....	270
Jaegermann, Michal.....	12, 476
Java, implementazione di <b>awk</b> .....	496
Java, linguaggio di programmazione.....	523
<b>jawk</b> .....	496
Jedi, Cavalieri.....	47
Johansen, Chris.....	326
<b>join()</b> , funzione definita dall'utente.....	253

## K

Kahrs, Jürgen.....	12, 476
Kasal, Stepan.....	12
Kenobi, Obi-Wan.....	47
Kernighan, Brian.....	6, 10, 13, 86, 124, 245, 463, 475, 495, 513, 517
<b>kill</b> , comando, profilazione dinamica e.....	346
Kwok, Conrad.....	476

## L

1, comando del debugger (alias per <b>list</b> ).....	375
<b>labels.awk</b> , programma.....	308
Langston, Peter.....	331
<b>LANGUAGE</b> , variabile d'ambiente.....	351
larghezza costante, campi di.....	77
<b>LC_ALL</b> , categoria di localizzazione.....	351
<b>LC_COLLATE</b> , categoria di localizzazione.....	351
<b>LC_CTYPE</b> , categoria di localizzazione.....	351
<b>LC_MESSAGES</b> , categoria di localizzazione.....	351
<b>LC_MESSAGES</b> , categoria di localizzazione,	
funzione <b>bindtextdomain()</b> di ( <b>gawk</b> ).....	353
<b>LC_MONETARY</b> , categoria di localizzazione.....	351
<b>LC_NUMERIC</b> , categoria di localizzazione.....	351
<b>LC_TIME</b> , categoria di localizzazione.....	351
leggere file in input.....	63
leggibilità, file-dati, controllare la.....	261
<b>length()</b> , funzione.....	201
Lesser General Public License (LGPL).....	523
LGPL (Lesser General Public License).....	523
<b>libmawk</b> .....	496
libreria di funzioni <b>awk</b> .....	245
libreria di funzioni <b>awk</b> ,	
arrotondamento di numeri.....	250
libreria di funzioni <b>awk</b> , asserzioni.....	249
libreria di funzioni <b>awk</b> , gestire file di dati....	257
libreria di funzioni <b>awk</b> , gestire ora del giorno	
(marcature temporali).....	254
libreria di funzioni <b>awk</b> , leggere la	
lista degli utenti.....	268
libreria di funzioni <b>awk</b> , leggere la	
lista dei gruppi.....	272
libreria di funzioni <b>awk</b> , opzioni sulla	
riga di comando.....	263
libreria di funzioni <b>awk</b> , programma di	
esempio per usare.....	317
libreria di funzioni <b>awk</b> , trasformare	
vettori in stringhe.....	253
libreria di funzioni <b>awk</b> , valori di	
carattere come numeri.....	251
libreria di funzioni <b>awk</b> , vettori associativi e...	247
libreria <b>gettext</b> .....	349
libreria <b>gettext</b> , categorie di localizzazione....	350
limitazioni nei nomi di funzione.....	225
limite-di-parola, individuare il.....	59
limite-di-parola, operatore ( <b>gawk</b> ).....	60

linguaggi di programmazione, guidati-dai-dati/procedurali .....	17
linguaggio di programmazione, Ada .....	515
linguaggio di programmazione, guidato dai dati .....	512
linguaggio di programmazione, Java .....	523
linguaggio di programmazione, ricetta per un ...	6
<i>lint</i> , controlli .....	164
<i>lint</i> , controlli con programma vuoto .....	33
<i>lint</i> , controlli, elementi di vettori .....	187
<i>lint</i> , controlli, emissione di avvertimenti .....	37
<i>lint</i> , controlli, funzione indefinita .....	232
<i>lint</i> , controlli, indici di vettori .....	186
<i>lint</i> , controlli, variabile d'ambiente POSIXLY_CORRECT .....	40
LINT, variabile .....	164
Linux .....	11, 357, 527
<i>list</i> , comando del debugger .....	375
lista degli utenti, leggere la .....	268
lista dei gruppi, leggere la .....	272
liste di caratteri, si veda espressioni tra parentesi quadre .....	53
locali, variabili, per una funzione .....	228
localizzazione .....	349
localizzazione, definizione di .....	141
localizzazione, separatore decimale della .....	38
localizzazione, si veda internazionalizzazione, localizzazione .....	349
<i>log</i> (registro), file di, marcature temporali nei .....	214
<i>log</i> (), funzione .....	196
logaritmo .....	196
logici, operatori, si veda espressioni booleane ..	136
logico, valore, vero/falso .....	130
login, informazioni .....	268
<i>lshift</i> (), funzione ( <i>gawk</i> ) .....	220
lunghezza di un record in input .....	201
lunghezza di una stringa .....	201
<i>lvalue/rvalue</i> .....	126

## M

mailing list, GNITS .....	12
maiuscolo/minuscolo e confronti tra stringhe ..	163
maiuscolo/minuscolo e <i>regex</i> .....	60, 163
maiuscolo/minuscolo, conversione da/a .....	207
maiuscolo/minuscolo, distinzione, indici dei vettori e .....	179
Malmberg, John E. ....	12, 477, 494
manipolazione di bit, funzioni per la .....	219
marcate, estrazione di stringhe (internazionalizzazione) .....	354
marcature temporali .....	214, 216
marcature temporali, conversione date nelle ...	216
marcature temporali, formato stringa .....	215
marcature temporali, formattate .....	254
<i>mark</i> , bit di parità (in ASCII) .....	252
Marx, Groucho .....	130

<i>match</i> (), funzione .....	201
<i>match</i> (), funzione, variabili <i>RSTART</i> / <i>RLENGTH</i> ...	202
<i>mawk</i> , programma di utilità ...	52, 87, 125, 161, 495
McIlroy, Doug .....	516
McPhee, Patrick .....	477
memoria tampone, scrivere su disco .....	210
memoria, allocare per estensioni .....	403
meno (-), in espressioni tra parentesi quadre ...	55
meno (-), nomi di file che iniziano con .....	34
meno (-), operatore - .....	140
meno (-), operatore -- .....	129, 140
meno (-), operatore -= .....	128, 141
<i>message object</i> file (.mo) .....	350
<i>message object</i> file (.mo), conversione da <i>portable object</i> file .....	358
<i>message object</i> file (.mo), specificare la directory di .....	350, 352
messaggi tradotti, impostare directory con catalogo .....	224
messaggi, stampare dalle estensioni .....	415
metacaratteri in espressioni regolari .....	52
metacaratteri, sequenze di protezione per .....	52
<i>mktime</i> (), funzione ( <i>gawk</i> ) .....	215
modalità compatibile di ( <i>gawk</i> ), estensioni nella .....	464
modalità compatibile di ( <i>gawk</i> ), nomi di file ...	109
modalità compatibile di ( <i>gawk</i> ), numeri esadecimali .....	116
modalità compatibile di ( <i>gawk</i> ), numeri ottali .....	116
modalità compatibile di ( <i>gawk</i> ), specificare ....	35
modificabili dall'utente, variabili .....	162
modificatori, in specificatori di formato .....	101
modulo-troncamento, operazione di .....	124
monete, rappresentazioni di, nella localizzazione .....	351
monete, simboli di, nella localizzazione .....	351
Moore, Duncan .....	89
mostrare argomenti delle funzioni, nel debugger .....	372
mostrare i punti d'osservazione, nel debugger ..	373
mostrare il nome del file sorgente corrente, nel debugger .....	372
mostrare punti d'interruzione, nel debugger ...	372
mostrare tutti i file sorgente, nel debugger ....	373
mostrare variabili locali, nel debugger .....	372
<i>msgfmt</i> , programma di utilità .....	358
multidimensionali, vettori .....	188

## N

**n**, comando del debugger (alias per **next**)..... 369  
 nascondere valori di variabile..... 226  
 NetBSD ..... 527  
 newsgroup **comp.lang.awk**..... 494  
**next file** statement..... 469  
**next**, comando del debugger ..... 369  
**next**, istruzione ..... 137, 159  
**next**, istruzione, criteri di  
     ricerca **BEGIN/END** e ..... 150  
**next**, istruzione, criteri di ricerca  
     **BEGINFILE/ENDFILE** e ..... 151  
**next**, istruzione, in funzioni  
     definite dall'utente ..... 160, 232  
**nextfile**, in funzioni definite dall'utente ..... 161  
**nextfile**, istruzione ..... 160  
**nextfile**, istruzione, criteri di  
     ricerca **BEGIN/END** e ..... 150  
**nextfile**, istruzione, criteri di ricerca  
     **BEGINFILE/ENDFILE** e ..... 150  
**nexti**, comando del debugger ..... 369  
**NF**, variabile..... 67, 167  
**NF**, variabile, decremento ..... 70  
**ni**, comando del debugger (alias for **nexti**).... 369  
**noassign.awk**, programma..... 262  
 nomi di file, assegnamenti di  
     variabile visti come ..... 262  
 nomi di file, distinguere..... 166  
 nomi di file, flussi standard in **gawk**..... 108  
 nomi di file, nella modalità  
     compatibile di **gawk**..... 109  
 nomi di funzione ..... 225, 246  
 nomi di funzione, limitazioni nei ..... 225  
 nomi di vettore/variabile ..... 246  
 nomi permessi, questioni sui ..... 246  
 non documentate, funzionalità ..... 47  
 non inizializzate, variabili, come  
     indici di vettore ..... 186  
**not**, operatore logico-booleoano ..... 136  
**NR**, variabile..... 63, 167  
**NR**, variabile, modifica di ..... 172  
 nulle, stringhe ..... 130  
 nulle, stringhe, come indici di vettore..... 186  
 nulle, stringhe, ed eliminazione di  
     elementi di un vettore ..... 187  
 numeri casuali, funzioni **rand()/srand()** ..... 196  
 numeri casuali, generatore Cliff ..... 251  
 numeri casuali, inizializzazione  
     generazione di ..... 197  
 numeri casuali, seme di ..... 197  
 numeri di campo ..... 68  
 numeri esadecimali ..... 115  
 numeri esadecimali nella modalità  
     compatibile di (**gawk**)..... 116  
 numeri in virgola mobile, VAX/VMS ..... 491  
 numeri interi a precisione arbitraria ..... 389  
 numeri interi, indici di vettore..... 185

numeri ottali ..... 115  
 numeri ottali nella modalità  
     compatibile di (**gawk**)..... 116  
 numeri, arrotondamento di ..... 250  
 numeri, come indici di vettore..... 185  
 numeri, come valori di carattere..... 251  
 numeri, conversione in stringhe.. 121, 163, 164, 221  
 numeriche, costanti ..... 115  
 numeriche, funzioni ..... 196  
 numeriche, stringhe ..... 132  
 numerico, formato di output ..... 98  
 numero di elementi di un vettore ..... 201  
 numero visto come stringa di bit ..... 221

## O

**o**, comando del debugger (alias per **option**)... 373  
 obsolete, funzionalità..... 47  
**OFMT**, variabile..... 98, 122, 164  
**OFS**, variabile..... 70, 164  
 OpenBSD ..... 527  
 OpenSolaris ..... 496  
 operatore di campo **\$** ..... 67  
 operatore di campo, dollaro come ..... 67  
 operatore **in**..... 156  
 operatore **in**, ordine di accesso dei vettori .... 182  
 operatore **in**, uso in cicli ..... 181  
 operatori aritmetici ..... 123  
 operatori booleani, si veda  
     espressioni booleane ..... 136  
 operatori di assegnamento ..... 126  
 operatori di assegnamento, ordine  
     di valutazione ..... 127  
 operatori di cortocircuito..... 137  
 operatori di decremento/incremento ..... 128  
 operatori di input/output ..... 85  
 operatori di stringa ..... 124  
 operatori logici, si veda espressioni booleane... 136  
 operatori relazionali, si veda  
     espressioni di confronto ..... 130  
 operatori, input/output ..... 86, 88, 105, 106, 141  
 operatori, limite-di-parola (**gawk**)..... 60  
 operatori, precedenza ..... 130, 140  
 operatori, ricerca in stringhe ..... 49  
 operatori, ricerca in stringhe, per buffer ..... 60  
 operatori, specifici per GNU..... 59  
 operazioni sui bit ..... 219  
**option**, comando del debugger ..... 373  
 opzione **--dump-variables**, uso per  
     funzioni di libreria..... 246  
 opzione **--gen-pot**..... 354  
 opzione **--no-optimize** ..... 39  
 opzione **--non-decimal-data** ..... 331  
 opzione **--profile**..... 343  
 opzione **--sandbox**, disabilitare la  
     funzione **system()** ..... 213  
 opzione **-s** ..... 39

opzione di configurazione	
--disable-extensions .....	485
opzione di configurazione --disable-lint .....	485
opzione di configurazione --disable-nls .....	485
opzione di configurazione	
--with-whiny-user-strftime .....	485
opzioni deprecate .....	47
opzioni estese .....	33
opzioni sulla riga di comando .....	33
opzioni sulla riga di comando,	
elaborazione di .....	263
opzioni sulla riga di comando, eseguire <b>awk</b> .....	33
opzioni sulla riga di comando,	
estrazione stringhe .....	354
opzioni sulla riga di comando, fine delle .....	34
opzioni, stampare una lista di .....	36
<b>or()</b> , funzione ( <b>gawk</b> ) .....	220
<b>or</b> , operatore logico-booleano .....	136
<b>OR</b> , operazione sui bit .....	219, 220
ora del giorno, gestire .....	254
<b>ord()</b> , estensione .....	451
<b>ord()</b> , funzione definita dall'utente .....	252
ordinamento di vettori .....	198
ordinamento vettori per indici .....	198
ordinare caratteri in lingue differenti .....	351
ordine di valutazione, concatenazione .....	125
ordine di valutazione, funzioni .....	195
<b>ORS</b> , variabile .....	97, 164
ottali, numeri .....	115
ottali, valori, abilitare l'interpretazione di .....	37
output, a <i>pipe</i> .....	105
output, bufferizzazione .....	210, 214
output, duplicarlo su più file .....	294
output, file in, chiusura .....	109
output, file in, si veda file in output .....	109
output, formattato .....	98
output, nella memoria tampone (buffer) .....	210
output, processore di .....	412
output, processore di, personalizzato .....	412
output, record .....	97
output, ridirezione .....	104
output, separatore di campo, si	
veda <b>OFS</b> , variabile .....	70
output, specificatore di formato, <b>OFMT</b> .....	98
output, stampare, si veda stampare .....	95
output, standard .....	107

## P

<b>p</b> , comando del debugger (alias per <b>print</b> ) ....	370
Papadopoulos, Panos .....	477
<i>parent process ID</i> del programma <b>gawk</b> .....	168
parentesi ( <b>()</b> ), in un profilo .....	345
parentesi ( <b>()</b> ), operatore <i>regexp</i> .....	53
parentesi acuta destra ( <b>&gt;</b> ), operatore <b>&gt;</b> .....	133, 141
parentesi acuta destra ( <b>&gt;</b> ), operatore <b>&gt;</b> (I/O) ..	105
parentesi acuta destra ( <b>&gt;</b> ), operatore <b>&gt;=</b> ..	133, 141

parentesi acuta destra ( <b>&gt;</b> ),	
operatore <b>&gt;&gt;</b> (I/O) .....	105, 141
parentesi acuta sinistra ( <b>&lt;</b> ), operatore <b>&lt;</b> ..	133, 141
parentesi acuta sinistra ( <b>&lt;</b> ), operatore <b>&lt;</b> (I/O) ..	85
parentesi acuta sinistra ( <b>&lt;</b> ), operatore <b>&lt;=</b> ..	133, 141
parentesi graffe ( <b>{}</b> ) .....	345
parentesi graffe ( <b>{}</b> ), azioni e .....	152
parentesi graffe ( <b>{}</b> ), istruzioni, raggruppare ..	153
parentesi quadre ( <b>[]</b> ), operatore <i>regexp</i> .....	53
parola chiave <b>case</b> .....	156
parola chiave <b>default</b> .....	156
parola, definizione in <i>regexp</i> .....	59
parole duplicate, ricerca di .....	302
parole, contare le .....	300
parole, statistica utilizzo delle .....	309
password, file delle .....	268
<b>patsplit()</b> , funzione ( <b>gawk</b> ) .....	203
<i>pattern</i> , si veda criteri di ricerca .....	145
<b>pawk</b> (versione con profilatura di Brian	
Kernighan <b>awk</b> ) .....	496
<b>pawk</b> , implementazione simile ad	
<b>awk</b> per Python .....	496
PC, <b>gawk</b> su sistemi operativi .....	486, 487
per cento ( <b>%</b> ), operatore <b>%</b> .....	140
per cento ( <b>%</b> ), operatore <b>%=</b> .....	128, 141
percorso di ricerca .....	487, 491
percorso di ricerca per estensioni .....	43, 434
percorso di ricerca per file sorgente ...	42, 328, 487,
	491
Perl .....	505
personalizzato, analizzatore di input .....	408
personalizzato, processore bidirezionale .....	414
personalizzato, processore di output .....	412
Peters, Arno .....	477
Peterson, Hal .....	476
pila ( <i>stack</i> ) delle chiamate,	
mostrare nel debugger .....	372
<i>pipe</i> , chiusura .....	109
<i>pipe</i> , input .....	86
<i>pipe</i> , output .....	105
Pitts, Dave .....	12, 494
più ( <b>+</b> ), operatore <b>+</b> .....	140
più ( <b>+</b> ), operatore <b>++</b> .....	128, 129, 140
più ( <b>+</b> ), operatore <b>+=</b> .....	127, 141
più ( <b>+</b> ), operatore <i>regexp</i> .....	54
Plauger, P.J. ....	245
plug-in .....	395
portabilità .....	51
portabilità, <b>#!</b> ( <i>script</i> eseguibili) .....	20
portabilità, barra inversa in	
sequenze di protezione .....	52
portabilità, continuazione di riga	
con barra inversa e .....	28
portabilità, eliminazione di	
elementi di un vettore .....	187
portabilità, file di dati come un unico record ..	67
portabilità, funzione <b>close()</b> .....	110
portabilità, funzione <b>length()</b> .....	201

- portabilità, funzione **substr()** ..... 207
- portabilità, **gawk** ..... 502
- portabilità, generare file oggetto ..... 36
- portabilità, internazionalizzazione e ..... 356
- portabilità, istruzione **next** in funzioni
  - definite dall'utente ..... 232
- portabilità, libreria **gettext** e ..... 349
- portabilità, nella definizione di funzioni ..... 226
- portabilità, nuovo **awk** vs. vecchio **awk** ..... 122
- portabilità, operatore **\*\*** ..... 124
- portabilità, operatore **\*\*=** ..... 128
- portabilità, operatori ..... 130
- portabilità, operatori, non in POSIX **awk** ..... 141
- portabilità, programmi di esempio ..... 245
- portabilità, variabile **ARGV** ..... 20
- portabilità, variabile d'ambiente
  - POSIXLY\_CORRECT** ..... 40
- portabilità, variabile **NF**, decremento ..... 71
- portable object file* (.po) ..... 350, 354
- portable object file* (.po), conversione in
  - message object file* ..... 358
- portable object template* (.pot), file ..... 350
- portare **gawk** ..... 502
- POSIX **awk** ..... 7, 128
- POSIX **awk**, **\*\*** e ..... 141
- POSIX **awk**, barre inverse in costanti stringa ... 52
- POSIX **awk**, **break** e ..... 158
- POSIX **awk**, criteri di ricerca **BEGIN/END** ..... 149
- POSIX **awk**, differenze tra versioni **awk** ..... 463
- POSIX **awk**, espressioni di intervallo in ..... 54
- POSIX **awk**, espressioni regolari e ..... 55
- POSIX **awk**, espressioni tra parentesi quadre e .. 55
- POSIX **awk**, espressioni tra parentesi quadre
  - e, classi di caratteri ..... 55, 57
- POSIX **awk**, funzione **length()** e ..... 201
- POSIX **awk**, funzioni **gsub()/sub()** e ..... 209
- POSIX **awk**, istruzione **continue** e ..... 159
- POSIX **awk**, istruzioni **next/nextfile** e ..... 160
- POSIX **awk**, marcature temporali e ..... 214
- POSIX **awk**, operatore **\*\*=** e ..... 128
- POSIX **awk**, operatore **<** e ..... 85
- POSIX **awk**, operatore **I/O |** e ..... 87
- POSIX **awk**, operatori aritmetici e ..... 124
- POSIX **awk**, opzioni estese GNU e ..... 33
- POSIX **awk**, parola chiave **function** in ..... 226
- POSIX **awk**, programma di utilità **date** e ..... 218
- POSIX **awk**, separatori di campo e ..... 76
- POSIX **awk**, stringhe di formato **printf** e ..... 103
- POSIX **awk**, stringhe numeriche e ..... 132
- POSIX **awk**, uso del punto (.) ..... 53
- POSIX **awk**, variabile **CONVFMT** e ..... 163
- POSIX **awk**, variabile **FS** e ..... 163
- POSIX **awk**, variabile **OFMT** e ..... 98, 122
- POSIX, **awk** e ..... 5
- POSIX, estensioni **gawk** non incluse in ..... 464
- POSIX, modalità ..... 38, 40
- POSIX, programmi, implementazione in **awk** .. 281
- POSIXLY\_CORRECT**, variabile d'ambiente ..... 40
- posizionali, specificatori, istruzione **printf** .... 354
- PREC**, variabile ..... 164
- precedenza ..... 130, 140
- precedenza, operatore **regex** ..... 55
- precisione arbitraria ..... 379
- precisione arbitraria, interi a ..... 389
- precisione infinita ..... 379
- precisione massima consentita dalla
  - libreria **MPFR** ..... 169
- precisione minima richiesta dalla
  - libreria **MPFR** ..... 169
- precisione multipla ..... 379
- predefinite, funzioni ..... 195
- predefinite, funzioni, ordine di valutazione .... 195
- predefinite, variabili ..... 162
- predefinite, variabili, che
  - forniscono informazioni ..... 165
- predefinite, variabili, opzione **-v**,
  - impostare con ..... 34
- print**, comando del debugger ..... 370
- print**, istruzione, continuazione di riga e ..... 97
- print**, istruzione, criteri di
  - ricerca **BEGIN/END** e ..... 149
- print**, istruzione, funzione **sprintf()** e ..... 250
- print**, istruzione, variabile **OFMT** e ..... 164
- printf**, comando del debugger ..... 371
- printf**, istruzione ..... 98
- printf**, istruzione, funzione **sprintf()** e ..... 250
- printf**, istruzione, lettere di
  - controllo del formato ..... 99
- printf**, istruzione, modificatori ..... 101
- printf**, istruzione, specificatori di posizione ... 354
- printf**, istruzione, specificatori posizionali .... 101
- printf**, istruzione, specificatori posizionali,
  - frammisti a formati standard ..... 355
- printf**, sintassi dell'istruzione ..... 98
- private, variabili ..... 246
- problemi di implementazione, **gawk** ..... 499
- problemi di implementazione,
  - gawk**, limitazioni ..... 106
- problemi di implementazione, **gawk**, debug .... 499
- problemi, risoluzione di, **awk** usa
  - FS** anziché **IFS** ..... 71
- problemi, risoluzione di, barra inversa prima di
  - caratteri non speciali ..... 52
- problemi, risoluzione di,
  - concatenazione di stringhe ..... 125
- problemi, risoluzione di, costanti **regex**
  - vs. costanti stringa ..... 58
- problemi, risoluzione di, divisione ..... 124
- problemi, risoluzione di, errori fatali, specificare
  - larghezza dei campi ..... 77
- problemi, risoluzione di, funzione **fflush()** .... 211
- problemi, risoluzione di, funzione **match()** .... 203
- problemi, risoluzione di, funzione **substr()** .... 206
- problemi, risoluzione di, funzione **system()** .... 213

problemi, risoluzione di, funzioni		
gsub()/sub() .....	206	
problemi, risoluzione di, gawk .....	499	
problemi, risoluzione di, gawk, errori fatali,		
argomenti di funzione e .....	195	
problemi, risoluzione di, operatore == .....	134	
problemi, risoluzione di, refusi,		
variabili globali .....	35	
problemi, risoluzione di, sintassi della		
chiamata di funzione .....	138	
process group ID del programma gawk .....	168	
process ID del programma gawk .....	168	
processi, comunicazioni bidirezionali con .....	339	
processore bidirezionale personalizzato .....	414	
processore di output .....	412	
processore di output personalizzato .....	412	
PROCINFO, valori di sorted_in .....	183	
PROCINFO, vettore .....	167, 215, 268	
PROCINFO, vettore, e appartenenza a gruppi .....	272	
PROCINFO, vettore, e comunicazioni		
attraverso le pty .....	341	
PROCINFO, vettore, e process ID di		
utente e di gruppo .....	290	
PROCINFO, vettore, verificare la		
divisione in campi .....	271	
profilare programmi awk .....	343	
profilare programmi awk, dinamicamente .....	346	
profilazione dinamica .....	346	
progetto gawkextlib .....	455	
Progetto GNU .....	10, 522	
programma alarm.awk .....	303	
programma anagram.awk .....	324	
programma awf (formattatore		
incredibilmente duttile) .....	521	
programma awk sed .....	316	
programma C, grcat .....	273	
programma cut.awk .....	282	
programma di utilità chem .....	517	
programma di utilità cut .....	282	
programma di utilità date GNU .....	214	
programma di utilità egrep .....	55, 286	
programma di utilità git .....	455, 495, 499, 501	
programma di utilità id .....	290	
programma di utilità mawk .....	52, 87, 125, 161, 495	
programma di utilità msgfmt .....	358	
programma di utilità POSIX date .....	218	
programma di utilità sed .....	76, 316, 515	
programma di utilità sleep .....	305	
programma di utilità sort .....	310	
programma di utilità sort, coprocessi e .....	340	
programma di utilità split .....	292	
programma di utilità split.awk .....	293	
programma di utilità tee .....	294	
programma di utilità tee.awk .....	294	
programma di utilità tr .....	305	
programma di utilità uniq .....	296	
programma di utilità uniq.awk .....	296	
programma di utilità w .....	77	
programma di utilità wc .....	300	
programma di utilità wc.awk .....	300	
programma di utilità xgettext .....	354	
programma dupword.awk .....	302	
programma egrep.awk .....	287	
programma expand .....	25	
programma extract.awk .....	313	
programma histsort.awk .....	312	
programma id.awk .....	291	
programma igawk.sh .....	319	
programma labels.awk .....	308	
programma noassign.awk .....	262	
programma signature .....	326	
programma testbits.awk .....	220	
programma translate.awk .....	306	
programma wordfreq.awk .....	310	
programma, definizione di .....	17	
programma, identificativi in un .....	168	
programmazione, concetti di .....	511	
programmazione, convenzioni di,		
estensioni gawk .....	437	
programmazione, convenzioni di, nomi		
di variabili private .....	246	
programmazione, convenzioni di, opzione		
--non-decimal-data .....	332	
programmazione, convenzioni di,		
parametri di funzione .....	233	
programmazione, passi fondamentali .....	511	
programmazione, ricetta per un linguaggio di .....	6	
programmi awk .....	19, 26	
programmi awk, complessi .....	30	
programmi awk, documentazione .....	20, 246	
programmi awk, eseguire .....	17, 19	
programmi awk, eseguire, da script di shell .....	18	
programmi awk, eseguire, senza file in input .....	18	
programmi awk, esempi di .....	281	
programmi awk, esempi molto corti .....	25	
programmi awk, internazionalizzare .....	224, 352	
programmi awk, lunghi .....	19	
programmi awk, variabili di shell in .....	151	
programmi compilati .....	511, 518	
programmi interpretati .....	511, 523	
programmi POSIX, implementazione in awk .....	281	
proseguire dopo errore in input .....	91	
protezione caratteri nelle funzioni		
gsub()/gensub()/sub() .....	207	
protezione, nella riga di comando di gawk .....	19	
protezione, nella riga di comando di		
gawk, trucchi per .....	22	
protezione, per piccoli programmi awk .....	20	
prova, modalità di .....	39	
puntatori a funzioni .....	234	
punto (.), operatore regexp .....	53	
punto d'interruzione .....	362	
punto d'interruzione (breakpoint), impostare ..	367	
punto d'interruzione in una determinata		
posizione, come cancellare .....	367	
punto d'interruzione temporaneo .....	368	

punto d'interruzione, cancellare per numero ... 368  
 punto d'interruzione, comandi ... 368  
 punto d'interruzione, come  
   disabilitare o abilitare ... 368  
 punto d'osservazione ... 362  
 punto e virgola (;), **AWKPATH** variabile e ... 487  
 punto e virgola (;), separare  
   istruzioni nelle azioni ... 29, 152, 153  
 punto esclamativo (!), operatore ! .. 137, 140, 148, 289  
 punto esclamativo (!), operatore != ... 133, 141  
 punto esclamativo (!), operatore !~ ... 49, 57, 61, 117, 133, 135, 141, 146  
 punto interrogativo (?), operatore ?: ... 141  
 punto interrogativo (?), operatore *regex* ... 54, 60  
**pwc**, programma ... 268

## Q

**q**, comando del debugger (alias per **quit**) ... 375  
**QSE awk** ... 496  
 Quastrom, Erik ... 303  
 questioni sui nomi permessi ... 246  
**QuikTrim Awk** ... 497  
**quit**, comando del debugger ... 375  
**QUIT**, segnale (MS-Windows) ... 346

## R

**r**, comando del debugger (alias per **run**) ... 369  
 radice quadrata ... 197  
 Rakitzis, Byron ... 312  
 Ramey, Chet ... 12, 400  
**rand()**, funzione ... 196  
 Rankin, Pat ... 12, 127, 476  
 rappresentazioni di monete, nella  
   localizzazione ... 351  
**reada()**, estensione ... 453  
**readable.awk**, programma ... 261  
**readdir**, estensione ... 451  
**readfile()**, estensione ... 454  
**readfile()**, funzione definita dall'utente ... 256  
 record ... 63, 512  
 record in input, lunghezza di un ... 201  
 record multiriga ... 80  
 record, fine dei ... 65  
 record, separatori di ... 63, 65, 164  
 record, separatori di, espressioni regolari come .. 65  
 record, stampare ... 95  
 record, suddividere l'input in ... 63  
 record, trattare un file come un solo ... 67  
*regex* ... 49  
*regex*, costanti ... 50, 117, 135  
*regex*, costanti, /=.../, operatore /= e ... 128  
*regex*, costanti, come criteri di ricerca ... 146  
*regex*, maiuscolo/minuscolo ... 60  
 registrazione di estensione ... 405

registro (*log*), file di, marcature  
   temporali nel ... 214  
 regola, definizione di ... 17  
 relazionali, operatori, si veda  
   espressioni di confronto ... 130  
 reti, funzionalità per ... 109  
 reti, programmazione di ... 341  
**return**, comando del debugger ... 369  
**return**, istruzione, in funzioni  
   definite dall'utente ... 232  
**rev()**, funzione definita dall'utente ... 227  
**revoutput**, estensione ... 452  
**revtoway**, estensione ... 452  
**rewind()**, funzione definita dall'utente ... 259  
 ricerca di parole ... 302  
 ricerca in stringhe ... 200  
 ricerca in stringhe, operatori ... 49  
 ricerca *regex* in stringhe ... 201  
 ricerca, percorso di, per file sorgente ... 42, 328  
 ricercare, in file, espressioni regolari ... 286  
 ricetta per un linguaggio di programmazione ... 6  
 ricorsive, funzioni ... 226  
 ridirezionare l'output di **gawk**, nel debugger ... 373  
 ridirezione dell'input ... 85  
 ridirezione dell'output ... 104  
 ridirezione in VMS ... 492  
 riga di comando, argomenti ... 40, 165, 172  
 riga di comando, directory su ... 92  
 riga di comando, eseguire **awk** da ... 33  
 riga di comando, formati ... 17  
 riga di comando, impostare FS sulla ... 74  
 riga di comando, opzione **-f** ... 19  
 riga di comando, opzioni ... 33  
 riga di comando, opzioni, elaborazione di ... 263  
 riga di comando, opzioni, estrazione stringhe .. 354  
 riga di comando, opzioni, fine delle ... 34  
 riga di comando, variabili, assegnare da ... 120  
 righe multiple, record su ... 80  
 righe, contare le ... 300  
 righe, duplicate, rimuovere ... 311  
 righe, individuare intervalli di ... 147  
 righe, saltare tra delimitatori ... 148  
 righe, vuote, stampare ... 95  
 rimpiazzare caratteri ... 305  
 rimpiazzare in una stringa ... 205  
 rimuovere righe duplicate ... 311  
 risoluzione di problemi, **awk** usa  
   **FS** anziché **IFS** ... 71  
 risoluzione di problemi, barra inversa prima di  
   caratteri non speciali ... 52  
 risoluzione di problemi,  
   concatenazione di stringhe ... 125  
 risoluzione di problemi, costanti *regex*  
   vs. costanti stringa ... 58  
 risoluzione di problemi, divisione ... 124  
 risoluzione di problemi, errori fatali, specificare  
   larghezza dei campi ... 77  
 risoluzione di problemi, funzione **fflush()** ... 211

risoluzione di problemi, funzione <code>getline</code> .....	261
risoluzione di problemi, funzione <code>match()</code> .....	203
risoluzione di problemi, funzione <code>substr()</code> ....	206
risoluzione di problemi, funzione <code>system()</code> ....	213
risoluzione di problemi, funzioni	
<code>gsub()/sub()</code> .....	206
risoluzione di problemi, <code>gawk</code> .....	499
risoluzione di problemi, <code>gawk</code> , errori fatali,	
argomenti di funzione e.....	195
risoluzione di problemi, leggibilità file-dati.....	261
risoluzione di problemi, operatore <code>==</code> .....	134
risoluzione di problemi, opzione	
<code>--non-decimal-data</code> .....	37
risoluzione di problemi, refusi,	
variabili globali .....	35
risoluzione di problemi, sintassi della	
chiamata di funzione.....	138
risoluzione problemi <code>gawk</code> , segnalare bug .....	493
Ritchie, Dennis .....	513
ritorno a capo.....	38, 137
ritorno a capo, come separatore di record.....	63
ritorno a capo, in costanti <code>regexp</code> .....	59
ritorno a capo, in <code>regexp</code> dinamiche .....	59
ritorno a capo, stampare un .....	95
<code>RLENGTH</code> , variabile.....	170
<code>RLENGTH</code> , variabile, funzione <code>match()</code> e .....	202
Robbins, Arnold ...	75, 87, 270, 303, 400, 477, 494, 505
Robbins, Bill .....	87
Robbins, Harry .....	13
Robbins, Jean .....	13
Robbins, Miriam .....	13, 87, 270
Rommel, Kai Uwe .....	476
<code>round()</code> , funzione definita dall'utente .....	250
<code>ROUNDMODE</code> , variabile .....	164
<code>RS</code> , variabile.....	63, 164
<code>RS</code> , variabile, record multiriga e .....	81
<code>rshift()</code> , funzione ( <code>gawk</code> ) .....	220
<code>RSTART</code> , variabile.....	170
<code>RSTART</code> , variabile, funzione <code>match()</code> e .....	202
<code>RT</code> , variabile .....	65, 83, 170
Rubin, Paul .....	6, 475
<code>run</code> , comando del debugger .....	369
<code>rvalue/lvalue</code> .....	126
<b>S</b>	
<code>s</code> , comando del debugger (alias per <code>step</code> ).....	369
salvataggio opzioni debugger .....	373
sandbox, modalità .....	39
scalare o vettore .....	223
scalari, valori .....	513
Schorr, Andrew.....	12, 171, 477
Schreiber, Bert.....	12
Schreiber, Rita.....	12
scrivere su disco i buffer di output	
contenuti in memoria.....	210
<code>sed</code> , programma di utilità .....	76, 316, 515
segnalare bug, indirizzo email,	
<code>bug-gawk@gnu.org</code> .....	493
segnali <code>HUP/SIGHUP</code> , per profilazione .....	346
segnali <code>INT/SIGINT</code> (MS-Windows).....	346
segnali <code>QUIT/SIGQUIT</code> (MS-Windows).....	346
segnali <code>USR1/SIGUSR1</code> , per profilazione.....	346
segno, interi senza .....	379
seno .....	197
senza segno, interi .....	379
separatore decimale, carattere, specifico	
della localizzazione .....	38
separatore di campo, nei record multiriga.....	81
separatore di campo, POSIX e il .....	76
separatore di campo, specificare sulla	
riga di comando .....	74
separatore di record in output, si	
veda <code>ORS</code> , variabile.....	97
separatori di campo.....	163, 164
separatori di campo, espressioni regolari come ..	72
separatori di campo, scelta dei .....	72
separatori di campo, si veda anche <code>OFS</code> .....	70
separatori di campo, spazi come.....	283
separatori di campo, variabile <code>FIELDWIDTHS</code> e ..	163
separatori di campo, variabile <code>FPAT</code> e.....	163
separatori di indici .....	165
separatori di record .....	63, 164
separatori di record, cambiare i.....	65
separatori di record, espressioni regolari come ..	65
separatori di record, per record multiriga .....	80
separatori, per istruzioni in azioni .....	152
sequenze di protezione, in stringhe .....	50
serie di caratteri (codifiche dei caratteri da	
parte della macchina).....	252
<code>set</code> , comando del debugger .....	371
shell, inviare comandi tramite <code>pipe</code> alla.....	107
shell, <code>script</code> .....	18
shell, uso di apici, regole per .....	21
shell, uso di doppio apice .....	151
shell, variabili di .....	151
<code>si</code> , comando del debugger (alias per <code>stepi</code> ) ...	370
sidebar, Allora perché <code>gawk</code> ha	
<code>BEGINFILE</code> e <code>ENDFILE</code> ? .....	259
sidebar, Ambiguità sintattiche tra <code>'/=</code> ' e le	
espressioni regolari .....	128
sidebar, Attenzione. Non è tutto oro	
quel che luccica! .....	222
sidebar, <code>awk</code> prima di POSIX usava <code>OFMT</code> per la	
conversione di stringhe.....	122
sidebar, Barra inversa prima di un	
carattere normale.....	51
sidebar, Bufferizzazione interattiva e	
non interattiva .....	211
sidebar, Cambiare <code>FS</code> non incide sui campi .....	75
sidebar, Comprendere <code>'#!'</code> .....	20
sidebar, Comprendere <code>\$0</code> .....	71
sidebar, Controllare la bufferizzazione	
dell'output con <code>system()</code> .....	214
sidebar, <code>FS</code> e <code>IGNORECASE</code> .....	77

- sidebar, Individuare la stringa nulla ..... 207
- sidebar, Inviare *pipe* alla **sh** ..... 107
- sidebar, La base di una costante non  
influisce sul suo valore ..... 116
- sidebar, Modificare **NR** e **FNR** ..... 172
- sidebar, Ordine di valutazione  
degli operatori ..... 129
- sidebar, Ricetta per un linguaggio di  
programmazione ..... 6
- sidebar, **RS = "\0"** non è portabile ..... 66
- sidebar, Sequenze di protezione  
per metacaratteri ..... 52
- sidebar, Usare **\n** in espressioni tra parentesi  
quadre in *regex* dinamiche ..... 59
- sidebar, Usare il valore di ritorno di **close()** .. 111
- SIGHUP**, segnale, per profilazione dinamica ..... 346
- SIGINT**, segnale (MS-Windows) ..... 346
- signature**, programma ..... 326
- SIGQUIT**, segnale (MS-Windows) ..... 346
- SIGUSR1**, segnale, per profilazione dinamica ..... 346
- silent**, comando del debugger ..... 368
- simboli di monete, nella localizzazione ..... 351
- sin()**, funzione ..... 197
- singolo carattere, campi ..... 73
- sintattica, ambiguità: operatore **/=** vs.  
costante *regex* **/=.../** ..... 128
- sistemi operativi basati su BSD ..... 11, 527
- sistemi operativi PC, **gawk** su ..... 487
- sistemi operativi per PC, **gawk** su ..... 486
- sistemi operativi, portare **gawk** su altri ..... 502
- sistemi operativi, si veda anche GNU/Linux,  
sistemi operativi per PC, Unix ..... 479
- Skywalker, Luke ..... 47
- sleep()**, estensione ..... 454
- sleep**, programma di utilità ..... 305
- Solaris, versione POSIX **awk** ..... 496
- sorgente, codice, **awka** ..... 495
- sorgente, codice, Brian Kernighan **awk** ..... 495
- sorgente, codice, BusyBox **Awk** ..... 496
- sorgente, codice, **gawk** ..... 479
- sorgente, codice, Illumos **awk** ..... 496
- sorgente, codice, **jawk** ..... 496
- sorgente, codice, **libmawk** ..... 496
- sorgente, codice, **pawk** ..... 496
- sorgente, codice, **pawk** (versione Python) ..... 496
- sorgente, codice, QSE **awk** ..... 496
- sorgente, codice, QuikTrim **Awk** ..... 497
- sorgente, codice, Solaris **awk** ..... 496
- sort()**, funzione, ordinamento di vettori ..... 336
- sort**, programma di utilità ..... 310
- sort**, programma di utilità, coprocessi e ..... 340
- sostituzione in stringa ..... 199
- sottostringa ..... 206
- sparsi, vettori ..... 178
- spazi come separatori di campo ..... 283
- spazi vuoti, come separatori di campo ..... 72
- spazi vuoti, ritorno a capo invece che ..... 38
- spazio bianco, nelle chiamate di funzione ..... 195
- specificatori di formato ..... 98
- specificatori di formato, frammisti a specificatori  
posizionali non standard ..... 355
- specificatori di formato, funzione  
**strftime()** di (**gawk**) ..... 216
- specificatori di formato, istruzione **printf** ..... 99
- specificatori posizionali,  
istruzione **printf** ..... 101, 354
- specificatori posizionali, istruzione **printf**,  
frammisti a formati standard ..... 355
- Spencer, Henry ..... 515
- split()**, funzione ..... 203
- split()**, funzione, eliminazione di  
elementi di vettori ..... 187
- split**, programma di utilità ..... 292
- split.awk**, programma di utilità ..... 293
- spostamento a destra ..... 220
- spostamento a destra, bit a bit ..... 219
- spostamento a sinistra ..... 220
- spostamento a sinistra, bit a bit ..... 219
- spostamento, bit a bit ..... 219
- sprintf()**, funzione ..... 98, 204
- sprintf()**, funzione, istruzioni  
**print/printf** e ..... 250
- sprintf()**, funzione, variabile **OFMT** e ..... 164
- sqrt()**, funzione ..... 197
- srand()**, funzione ..... 197
- stack* (pila) delle chiamate,  
mostrare nel debugger ..... 372
- stack frame* ..... 362
- stalli ..... 340
- Stallman, Richard ..... 10, 12, 476, 521
- stampa, lista di opzioni ..... 36
- stampare ..... 95
- stampare etichette per lettera ..... 308
- stampare informazioni utente ..... 290
- stampare messaggi dalle estensioni ..... 415
- stampare righe di testo non duplicate ..... 296
- stampare variabili, nel debugger ..... 370
- standard error ..... 107
- standard input ..... 18, 107
- standard output ..... 107
- stat()**, estensione ..... 445
- statistica utilizzo delle parole ..... 309
- stato d'uscita, di **gawk** ..... 45
- step**, comando del debugger ..... 369
- stepi**, comando del debugger ..... 370
- strftime()**, funzione (**gawk**) ..... 215
- stringa nulla come argomento a **gawk**,  
protezione della ..... 22
- stringa nulla, individuare la ..... 207
- stringa più lunga da sinistra, individuare la ..... 81
- stringa, lunghezza di una ..... 201
- stringa, operatori di ..... 124
- stringa, ricercare espressioni regolari in una ... 201
- stringa, rimpiazzare in una ..... 205
- stringhe marcate, estrazione di ..... 354

stringhe marcate, estrazione di (internazionalizzazione) .....	354
stringhe nulle .....	65, 73, 513
stringhe nulle, come indici di vettore .....	186
stringhe nulle, conversione da tipo numerico a tipo stringa .....	121
stringhe nulle, ed eliminazione di elementi di un vettore .....	187
stringhe vuote .....	65
stringhe vuote, si veda stringhe nulle .....	73
stringhe, conversione .....	121, 221
stringhe, conversione in numeri .....	163, 164, 204
stringhe, convertire maiuscolo/minuscolo .....	207
stringhe, divisione, esempio .....	204
stringhe, estrazione di .....	354
stringhe, formattazione .....	204
stringhe, limitazioni della lunghezza .....	115
stringhe, marcare per localizzazione .....	352
stringhe, numeriche .....	132
stringhe, trasformare vettori in .....	253
strtonum(), funzione (gawk) .....	204
strtonum(), funzione (gawk), opzione --non-decimal-data e .....	332
sub(), funzione .....	118, 205
sub(), funzione, argomenti di .....	206
sub(), funzione, protezione caratteri .....	207
SUBSEP, variabile .....	165
SUBSEP, variabile, e vettori multidimensionali ..	188
substr(), funzione .....	206
Sumner, Andrew .....	495
sveglia, programma di esempio .....	303
switch, istruzione .....	156
SYMTAB, vettore .....	170
system(), funzione .....	212
systemtime(), funzione (gawk) .....	216

## T

t, comando del debugger (alias per <b>tbreak</b> ) ...	368
<b>tbreak</b> , comando del debugger .....	368
Tcl .....	247
TCP/IP .....	341
TCP/IP, funzionalità per .....	109
<b>tee</b> , programma di utilità .....	294
<b>tee.awk</b> , programma di utilità .....	294
tempo limite, leggere input .....	90
tempo, localizzazione e .....	351
tempo, ottenerlo .....	214
tempo, sveglia, programma di esempio .....	303
temporaneo, punto d'interruzione .....	368
<b>testbits.awk</b> , programma .....	220
<b>testext</b> , estensione .....	455
testo, stampare .....	95
testo, stampare, righe non duplicate di .....	296
Texinfo .....	9, 245, 302, 312, 481, 501
Texinfo, estrarre programma da file sorgente ..	312
Texinfo, inizi di capitolo nei file .....	53
<b>textdomain()</b> , funzione (libreria C) .....	350

<b>TEXTDOMAIN</b> , variabile .....	165, 352
<b>TEXTDOMAIN</b> , variabile, criterio di ricerca <b>BEGIN</b> e .....	353
<b>TEXTDOMAIN</b> , variabile, portabilità e .....	356
tilde (~), operatore ~ .....	49, 57, 61, 117, 133, 135, 141, 146
<i>timeout</i> , si veda tempo limite .....	90
tipo di una variabile .....	223
<b>tolower()</b> , funzione .....	207
<b>toupper()</b> , funzione .....	207
<b>tr</b> , programma di utilità .....	305
tracciatura a ritroso, mostrare nel debugger ...	372
<b>trace</b> , comando del debugger .....	375
traduzione di stringhe .....	224
traduzione di stringhe, funzioni di .....	224
<b>translate.awk</b> , programma .....	306
trattare un file, come un solo record .....	67
trattino basso (_), macro C .....	350
trattino basso (_), nei nomi di variabili private .....	246
trattino basso (_), stringa traducibile .....	353
trovare le estensioni .....	434
trovare sottostringhe in una stringa .....	200
Trueman, David .....	6, 12, 476
<b>typeof()</b> , funzione (gawk) .....	223

## U

u, comando del debugger (alias per <b>until</b> ) ....	370
uguale (=), operatore = .....	126
uguale (=), operatore == .....	133, 141
<b>undisplay</b> , comando del debugger .....	371
Unicode .....	252, 475, 517
<b>uniq</b> , programma di utilità .....	296
<b>uniq.awk</b> , programma di utilità .....	296
Unix .....	527
Unix <b>awk</b> , barre inverse in sequenze di protezione .....	52
Unix <b>awk</b> , file di password, separatori di campo e .....	75
Unix <b>awk</b> , funzione <b>close()</b> e .....	112
Unix, <i>script awk</i> e .....	19
<b>until</b> , comando del debugger .....	370
<b>unwatch</b> , comando del debugger .....	371
<b>up</b> , comando del debugger .....	372
uscire dal debugger .....	375
<b>USR1</b> , segnale, per profilazione dinamica .....	346
utente, funzioni definite dall' .....	224
utenti, informazioni riguardo agli, ottenere ....	268
utenti, informazioni riguardo agli, stampare ...	290
utenti, leggere la lista degli .....	268

## V

- valore di ritorno, funzione `close()` ..... 112
- valori di verità ..... 130
- valori numerici ..... 513
- valori scalari ..... 513
- valori tipo stringa ..... 513
- valutare espressioni, nel debugger ..... 370
- valutazione, ordine di ..... 130
- valutazione, ordine di, concatenazione ..... 125
- variabile `ARGIND` ..... 166
- variabile `BINMODE` ..... 162, 487
- variabile `CONVMT` ..... 163
- variabile d'ambiente `AWKPATH` ..... 487
- variabile d'ambiente `LANGUAGE` ..... 351
- variabile `ERRNO` ..... 166, 342
- variabile `ERRNO`, con funzione `close()` ..... 112
- variabile `FIELDWIDTHS` ..... 163
- variabile `FILENAME` ..... 63, 167
- variabile `FNR` ..... 167
- variabile `FNR`, modifica di ..... 172
- variabile `FPAT` ..... 163
- variabile `FS` ..... 163
- variabile `FS`, eseguire programmi `awk` e ..... 282
- variabile `IGNORECASE` ..... 163
- variabile `IGNORECASE`, e funzioni di
  - ordinamento dei vettori ..... 338
- variabile `IGNORECASE`, e indici dei vettori ..... 179
- variabile `IGNORECASE`, nei
  - programmi di esempio ..... 245
- variabile `LINT` ..... 164
- variabile `NF` ..... 167
- variabile `NR` ..... 167
- variabile `NR`, modifica di ..... 172
- variabile `OFMT` ..... 164
- variabile `OFMT`, POSIX `awk` e ..... 98
- variabile `OFS` ..... 97, 164
- variabile `ORS` ..... 164
- variabile `PREC` ..... 164
- variabile `RLENGTH` ..... 170
- variabile `RLENGTH`, funzione `match()` e ..... 202
- variabile `ROUNDMODE` ..... 164
- variabile `RS` ..... 164
- variabile `RSTART` ..... 170
- variabile `RSTART`, funzione `match()` e ..... 202
- variabile `RT` ..... 170
- variabile `SUBSEP` ..... 165
- variabile `SUBSEP`, e vettori multidimensionali .. 188
- variabile `TEXTDOMAIN` ..... 165, 352
- variabile `TEXTDOMAIN`, criterio di
  - ricerca `BEGIN` e ..... 353
- variabile `TEXTDOMAIN`, portabilità e ..... 356
- variabile, conversione di tipo ..... 121
- variabile, tipi di ..... 130
- variabile, tipo di una ..... 223
- variabili ..... 30, 513
- variabili `ARGC/ARGV`, come usarle ..... 172
- variabili `ARGC/ARGV`, portabilità e ..... 20
- variabili d'ambiente usate da `gawk` ..... 41
- variabili d'ambiente, nel vettore `ENVIRON` ..... 166
- variabili definite dall'utente ..... 119
- variabili di tipo indicatore [*flag*] ..... 137, 294
- variabili globali, accesso dalle estensioni ..... 417
- variabili globali, per funzioni di libreria ..... 246
- variabili globali, stampare una lista delle ..... 35
- variabili informative dell'API ..... 432
- variabili locali, in una funzione ..... 228
- variabili nascoste ..... 226
- variabili non inizializzate, come
  - indici di vettore ..... 186
- variabili predefinite ..... 162
- variabili predefinite, che
  - forniscono informazioni ..... 165
- variabili predefinite, impostare con opzione `-v` .. 34
- variabili predefinite, modificabili dall'utente .. 162
- variabili private ..... 246
- variabili, assegnare da riga di comando ..... 120
- variabili, impostazione ..... 34
- variabili, inizializzazione ..... 120
- variabili, predefinite ..... 120
- variabili, tipi di ..... 126
- variabili, tipi di, espressioni di confronto e .... 130
- variabili, usare in comando `getline` .. 85, 86, 87, 88
- VAX/VMS, numeri in virgola mobile, ..... 491
- vero, valore logico (diverso da zero e
  - da stringa nulla) ..... 130
- versione dell'estensione API `gawk` ..... 169, 431
- versione della libreria GNU MP ..... 169
- versione della libreria GNU MPFR ..... 169
- versione di `gawk` ..... 169
- vettore `ENVIRON` ..... 166
- vettore `FUNCTAB` ..... 167
- vettore `PROCINFO` ..... 167, 215, 268
- vettore `PROCINFO`, verificare la
  - divisione in campi ..... 271
- vettore `SYMTAB` ..... 170
- vettore, determinare il numero degli elementi .. 201
- vettore, elementi di un ..... 179
- vettori ..... 177
- vettori associativi ..... 178
- vettori associativi, funzioni di libreria e ..... 247
- vettori di vettori ..... 190
- vettori multidimensionali ..... 188
- vettori sparsi ..... 178
- vettori, come parametri di funzione ..... 231
- vettori, determinare il numero degli elementi .. 198
- vettori, elementi che non esistono ..... 179
- vettori, elementi di, ordine di accesso da
  - parte dell'operatore `in` ..... 182
- vettori, elementi non assegnati ..... 179
- vettori, elementi, assegnare valori ..... 180
- vettori, elementi, eliminazione di ..... 187
- vettori, eliminare l'intero contenuto ..... 187
- vettori, esaminare elementi ..... 179
- vettori, indici numerici di ..... 185
- vettori, indici, variabili non
  - inizializzate come ..... 186

vettori, indicizzazione .....	178
vettori, istruzione <b>for</b> e .....	181
vettori, manipolazione nelle estensioni .....	421
vettori, multidimensionali, visitare .....	189
vettori, ordinamento dei .....	198, 336
vettori, ordinamento per indici .....	198
vettori, ordinamento, variabile <b>IGNORECASE</b> e .....	179, 338
vettori, ordine di visita, controllo dell' .....	183
vettori, trasformare in stringhe .....	253
vettori, un esempio sull'uso .....	180
vettori, visitare .....	181
Vinschen, Corinna .....	12
virgola (,), negli intervalli di ricerca .....	147
virgola mobile, numeri in, precisione arbitraria .....	379
visitare vettori .....	181
visitare vettori multidimensionali .....	189
visualizzare le opzioni del debugger .....	373
visualizzazioni automatiche, nel debugger .....	372
VMS, compilare <b>gawk</b> per .....	488
VMS, installare <b>gawk</b> su .....	488
vuoto, criterio di ricerca .....	151

## W

<b>w</b> , comando del debugger (alias per <b>watch</b> ) ....	371
<b>w</b> , programma di utilità .....	77
<b>wait()</b> , estensione .....	449
<b>waitpid()</b> , estensione .....	449
<b>walk_array()</b> , funzione definita dall'utente ....	277
Wall, Larry .....	177, 505
Wallin, Anders .....	477
<b>watch</b> , comando del debugger .....	371

<b>watchpoint</b> .....	362
<b>wc</b> , programma di utilità .....	300
<b>wc.awk</b> , programma di utilità .....	300
Weinberger, Peter .....	6, 475
<b>where</b> , comando del debugger .....	372
<b>where</b> , comando del debugger (alias per <b>backtrace</b> ) .....	372
<b>while</b> , istruzione .....	154
<b>while</b> , istruzione, uso di espressioni regolari in .....	49
Williams, Kent .....	476
Woehlke, Matthew .....	477
Woods, John .....	476
<b>wordfreq.awk</b> , programma .....	310
<b>writea()</b> , estensione .....	453

## X

<b>xgettext</b> , programma di utilità .....	354
XML, Expat, libreria per analizzare .....	455
<b>xor()</b> , funzione ( <b>gawk</b> ) .....	220
XOR, operazione sui bit .....	219, 220

## Y

Yawitz, Efraim .....	477
----------------------	-----

## Z

Zaretskii, Eli .....	12, 476, 494
<b>zerofile.awk</b> , programma .....	261
Zoulas, Christos .....	476