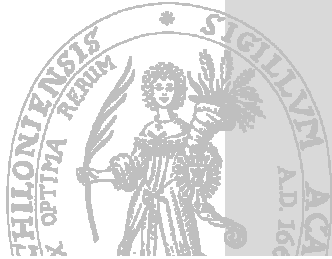


## Programmieren in C

Simone Knief – Rechenzentrum der CAU



## Programmieren in C

### Literatur

- Online verfügbares Buch: C von A bis Z
  - [www.galileocomputing.de/openbook/c\\_von\\_a\\_bis\\_z/](http://www.galileocomputing.de/openbook/c_von_a_bis_z/)
- Die Programmiersprache C:
  - Heft aus der RRZN-Reihe (in der UB erhältlich)
- Kursunterlagen:
  - [www.rz.uni-kiel.de/kurse/doku/c/](http://www.rz.uni-kiel.de/kurse/doku/c/)

## Programmieren in C

### Gliederung

- Einleitung
- Komponenten eines C-Programms
- Daten speichern: Variablen und Konstanten
- Anweisungen und Ausdrücke
- Operatoren
- Kontrollstrukturen
- Arrays und Strings
- Funktionen, globale und lokale Variablen
- Zeiger
- Arbeiten mit Dateien
- Strukturen

## Programmieren in C

### Geschichte von C

- 1972 entwickelt von Brian W. Kernighan und Dennis Ritchie
- 1983: American National Standard Institute (ANSI) bildet eine Komitee, um eine einheitliche Definition von C zu erreichen
- 1989: ANSI Standard-C (C89-Standard)
- 1990: Standard wird mit kleinen Änderungen von der International Standard Organization (ISO) übernommen.
- 1999: Überarbeitung und Ergänzung zum C99-Standard

- C ist eine leistungsfähige und flexible Sprache
- breite Palette von Compilern und Tools verfügbar
- C ist portabel
- C kommt mit wenigen Schlüsselwörtern aus
- C ist modular
- C ermöglicht hardwarenahe Programmierung
- C-Programme sind ressourcensparend
- C ist eine Teilmenge von C++

- Problembeschreibung
- Modellbildung: Entwurf von Datenstruktur und Algorithmen
- Programmcode erstellen
- Compilieren in eine Objektdatei
- Linken der Objektdateien zu einem ausführbaren Programm
- Testen des Programms
- Produktionsbetrieb

- Übersetzt Programmcode in die Maschinensprache
- besteht eigentlich aus 3 Programmen
  - Compiler
  - Präprozessor
  - Linker
- Präprozessor
  - fügt vor der Programmübersetzung zusätzlichen Text in den Code ein
  - entfernt Codezeilen aus dem Code (z.B. Kommentare)
- Linker
  - auch als Binder bezeichnet
  - verbindet die einzelnen Programmmodule zu einer ausführbaren Datei

- Problem
  - Programm soll auf dem Bildschirm ausgeben:  
"Viel Spass mit C !"
- Programmcode mit einem Editor eingeben:
 

```
#include <stdio.h>
int main ()
{
    printf ("Viel Spass mit C !\n");
    return 0;
}
```
- Code abspeichern unter dem Namen hello.c
- Programmcode compilieren:
  - gcc -o hello.exe hello.c
  - Kein Fehler → Datei hello.exe wird erzeugt
- Programm starten mit ./hello.exe
- ohne Option -o: executable hat den Namen a.out

- Linux:
  - Gnu-Compiler: [gcc.gnu.org](http://gcc.gnu.org)
- Windows:
  - Dev-Compiler: [www.bloodshed.net](http://www.bloodshed.net)
  - Pelles-C-Compiler: [www.christian-heffner.de](http://www.christian-heffner.de)

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf(" %d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf(" %d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

- /\* Kommentartext \*/
  - z.B.: /\* Programm zur Bildung einer Summe zwei Zahlen \*/
  - enthalten Erklärungen und Kommentare zum Code
  - kann über mehrere Zeilen gehen
  - darf nicht verschachtelt werden
  - Compiler ignoriert alle Zeichen innerhalb des Kommentarblocks
  - beeinflussen die Performance eines Programms nicht
  - sollen helfen das Programm auch ohne fremde Hilfe zu verstehen
  - keine unnötigen Kommentare für offensichtliche Anweisungen:
    - z.B.: /\* die folgende Programmzeile gibt die Anweisung: hello world auf dem Bildschirm aus \*/
- ```
printf("hello world\n");
```

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf(" %d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

13

- #include <stdio.h>
- include-Dateien oder Header-Dateien
- gehören zu den Präprozessoranweisungen
- sind Anweisungen an den Präprozessor
- stehen immer am Anfang eines Programms
- beginnen mit einem #
- Inhalt wird beim Compilieren in den Programmcode eingefügt
- Funktion <stdio.h> enthält Informationen über Ein- und Ausgabefunktionen (z.B. printf und scanf)
- am Zeilenende darf kein Semikolon stehen (<-> Anweisungen)

14

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf(" %d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

15

- ist in jedem C-Programm vorhanden
- int main ()
  - {
    - {*Programmanweisungen*}
    - return 0;
- erste Funktion, die in einem C-Programm aufgerufen wird
- geschweifte Klammern schließen Programmteile ein, die eine C-Funktion bilden
- return 0;
  - Hauptprogramm beenden und Fehlercode 0 (d.h. kein Fehler) zurückgeben

16

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf("%d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

17

- `int x,y,summe;`
- Variablen sind Speicherstellen für verschiedene Arten von Daten
- Variablen müssen vor der ersten Verwendung zunächst definiert werden
- informiert den Compiler über Variablennamen und Datentyp
- reserviert einen festen Speicherbereich

18

```
/* Programm zur Bildung einer Summe zwei Zahlen */
#include <stdio.h>
int main()
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf("%d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

19

- Anweisungen realisieren alle Operationen, die ein Programm ausführen muss
- typische C-Anweisungen:
  - Informationen auf Bildschirm ausgeben :  
`printf("Bitte geben Sie zwei ganze Zahlen ein! \n");`
  - Daten von der Tastatur einlesen:  
`scanf("%d %d", &x, &y);`
  - mathematische Operationen ausführen:  
`summe=x+y;`
  - Funktionen aufrufen
- Quellcode enthält gewöhnlich eine Anweisung pro Codezeile
- Anweisungen immer mit einem Semikolon (;) abschließen

20

```
#include <stdio.h>
int main()
{
    printf ("Hallo \n");
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    printf ("Hallo \n");
    return 0;
}
```

```
#include<stdio.h>
int main(){printf("Hallo \n"); return
0;}
```

- Anordnung des Programmcodes
- nur eine Anweisung pro Zeile
- Anweisungsblöcke einrücken
- Programme großzügig kommentieren

- Entwicklungszyklus eines Programms vorgestellt
- Hauptkomponenten eines C-Programms eingeführt
- bisher behandelte C-Komponenten:
  - Programmkommentare  
/\* Kommentartext \*/
  - Include-Direktive für Ein- und Ausgabefunktionen  
#include <stdio.h>
  - in jedem C-Programm gibt es eine Funktion main
 

```
int main()
{
    Programmanweisungen
    return 0;
}
```

## Daten speichern: Variablen und Konstanten

Rechenzentrum

- Was ist eine Variable ?
- Wie definiere und verwende ich Variablen ?
- Welche Datentypen gibt es in C ?
- Wie definiere und verwende ich Konstanten ?

25

## Was ist eine Variable ?

Rechenzentrum

- benannte Speicherstelle für Daten (Zahlen oder Zeichen) im Hauptspeicher
- Wert kann sich während der Programmausführung verändern
- zu einer Variablen gehören folgende Informationsbestandteile:
  - Variablenname:
    - stellt die Speicherstelle des Wertes symbolisch dar
  - Variablenwert:
    - Zahlenwert, der an dieser Speicherstelle gespeichert wird
  - Adresse:
    - genaue Position im Speicher des Computers
  - Datentyp:
    - legt das Format für den zu speichernden Wert fest
  - Beispiel: `int xwert = 5;`

26

## Variablenname

Rechenzentrum

- kann Zeichen, Ziffern und den Unterstrich enthalten.
- erste Zeichen muss immer ein Buchstabe (oder Unterstrich) sein
- C unterscheidet zwischen Groß- und Kleinschreibung
  - *d.h. Summe und summe sind unterschiedliche Variablen*
- C-Schlüsselwörter (z.B. `float`, `double`) sind als Variablenname nicht zulässig
- Namen sollten immer aussagekräftig sein
- sollten in Kleinbuchstaben geschrieben werden (<-> Konstanten)
- Variablennamen müssen eindeutig in einer Funktion sein
- Beispiele für gültige Variablennamen:
  - `Prozent`, `summe`, `gewinn_pro_jahr`, `umsatz`, `y2x5`
- Beispiele für nicht zulässige Variablennamen:
  - `sparkassen#konto`, `double`, `9winter`, `Zähler`

27

## Elementare Datentypen

Rechenzentrum

- sind die Baupläne für die Darstellung einer Variablen
- verschiedene numerische Werte haben unterschiedlichen Speicherbedarf und ausführbare mathematische Operationen sind nicht für alle Datentypen gleich
- geben Auskunft über das Format eines Wertes an einer bestimmten Speicherstelle
- legen Regeln fest, wie ein Wert interpretiert und verwendet werden kann
- 3 Kategorien von Datentypen
  - Textzeichen: Darstellung eines einzelnen Zeichens
  - integer-Zahlen: Darstellung ganzer Zahlen
    - vorzeichenbehaftet oder vorzeichenlos
  - Fließkommazahlen: Zahlen mit Nachkommastellen
    - `float`: Dezimalzahlen mit einfacher Genauigkeit
    - `double`: Dezimalzahlen mit doppelter Genauigkeit
    - `long double`: Dezimalzahlen mit zusätzlicher Genauigkeit

28

| Variablentyp                         | Schlüsselwort  | Bytes | Wertebereich                                                                   |
|--------------------------------------|----------------|-------|--------------------------------------------------------------------------------|
| Zeichen                              | char           | 1     | -128 bis 127                                                                   |
| Ganzzahl                             | int            | 4     | -32768 bis 32767                                                               |
| Kurze Ganzzahl                       | short          | 2     | -32768 bis 32767                                                               |
| Lange Ganzzahl                       | long           | 4     | -2147483646 bis 2147483647                                                     |
| Zeichen ohne VZ                      | unsigned char  | 1     | 0 bis 255                                                                      |
| Ganzzahl ohne VZ                     | unsigned int   | 2     | 0 bis 65535                                                                    |
| Kurze Ganzzahl ohne VZ               | unsigned short | 2     | 0 bis 65535                                                                    |
| Lange Ganzzahl ohne VZ               | unsigned long  | 4     | 0 bis 4294967295                                                               |
| Gleitkommazahl einfache Genauigkeit  | float          | 4     | Ca. 3.4E38 bis -1.2E-38 und 1.2E-38 bis 3.4E38<br>Genauigkeit 7 Dezimalziffern |
| Gleitkommazahl doppelter Genauigkeit | double         | 8     | Genauigkeit von 19 Dezimalziffern                                              |

29

- Programm übersetzen:
  - gcc -o size.exe sizeof.c
- Programm ausführen:
  - ./size.exe
- Programmausgabe:
  - Ein char belegt 1 Bytes.
  - Ein int belegt 4 Bytes.
  - Ein short belegt 2 Bytes.
  - Ein float belegt 4 Bytes.
  - Ein double belegt 8 Bytes.
  - Ein long double belegt 16 Bytes.

31

- sizeof-Operator ermittelt die Länge eines Datentyps auf einem System
- Beispielprogramm:
 

```
/* sizeof-Funktion */
#include <stdio.h>
int main()
{
    printf("Ein char belegt %d Bytes.\n",sizeof(char));
    printf("Ein int belegt %d Bytes.\n",sizeof(int));
    printf("Ein short belegt %d Bytes.\n",sizeof(short));
    printf("Ein float belegt %d Bytes.\n",sizeof(float));
    printf("Ein double belegt %d Bytes.\n",sizeof(double));
    printf("Ein long double belegt %d Bytes.\n",sizeof(long double));
    return 0;
}
```

30

- 3 Schritte für Variablenverwendung erforderlich:
  - Variablendeklaration
  - Variableninitialisierung
  - Verwendung

32



## Variablendeklaration

Rechenzentrum

- jede Variable muss vor der Verwendung erst deklariert werden
- teilt dem Compiler Namen und Typ der Variablen mit
- mit der Deklaration wird auch zusätzlich Speicherplatz reserviert
- allg. Form: Datentyp Variablenname;
- auf einer Zeile können mehrere Variablen desselben Typs deklariert werden
- Variablenname kann frei gewählt werden
- Variablen werden immer am Anfang eines Blocks deklariert
- Variablen sind im allgemeinen nur innerhalb ihres Blockes definiert
- Beispiele:
  - int zaehler, start;
  - float prozent, total;
  - char zeichen;

33

## Variablen initialisieren

Rechenzentrum

- nach der Deklaration haben Variablen einen unbestimmten Wert
- Variablen sollten vor der ersten Verwendung zusätzlich immer initialisiert werden
- Initialisierung legt einen Anfangswert für die Variable fest
- Initialisierung kann bei der Deklaration oder in einer extra Zuweisung erfolgen
- Variablen dürfen nicht mit Werten außerhalb der Gültigkeit initialisiert werden
- Beispiele:
  - int zaehler=0;
  - int zaehler;  
  zaehler=0;
  - int breite=5;
  - float prozent=0.25;
  - char zeichen='X';

34

## Variablen verwenden (Beispiel)

Rechenzentrum

```
/* Flaeche und Umfang eines Rechtecks berechnen */
#include<stdio.h>
int main()
{
    int umfang=0, flaeche=0;
    int breite=5;
    int hoehe=3;
    umfang=2*(breite+hoehe);
    flaeche=(breite*hoehe);
    printf("Der Umfang betraegt %d. \n",umfang);
    printf("Die Flaeche betraegt %d. \n",flaeche);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

35

## Variablen verwenden (Beispiel)

Rechenzentrum

```
/* Flaeche und Umfang eines Rechtecks berechnen */
#include<stdio.h>
int main()
{
    int umfang, flaeche;
    int breite=5;
    int hoehe=3;
    umfang=2*(breite+hoehe);
    flaeche=(breite*hoehe);
    printf("Der Umfang betraegt %d. \n",umfang);
    printf("Die Flaeche betraegt %d. \n",flaeche);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

- **Programm übersetzen:**
  - gcc -o rechteck.exe rechteck.c
- **Programm ausführen:**
  - ./rechteck.exe
- **Programmausgabe:**

**Der Umfang betraegt 16.**  
**Die Flaeche betraegt 15.**  
**Programm erfolgreich beendet.**

36

## Konstanten

Rechenzentrum

- Speicherbereich für Daten mit dem ein Programm arbeitet
- Wert kann während der Programmausführung nicht verändert werden
- Konstanten sind Variablen mit einem festen Wert
- wird mit dem Schlüsselwort const definiert:
 

```
const datentyp NAME = wert;
```
- werden in einem C-Programm immer durch das Schlüsselwort const gekennzeichnet
- Konstanten werden gleichzeitig deklariert und initialisiert
- Konstantennamen bestehen vorzugsweise aus Großbuchstaben
- Für Konstantennamen gelten die gleiche Regeln wie bei Variablennamen
- Beispiele:
  - `const int RADIUS=4;`
  - `const float BREITE=10.25;`
  - `const int SCHULDEN=10000, float ZINSSATZ=4.5;`

37

## Konstanten (Beispiel)

Rechenzentrum

```
/* Kreisberechnung: Umfang und Flaeche */
#include <stdio.h>
int main()
{
    int radius;
    const float PI=3.14159;
    float umfang=0, flaeche=0;
    printf("Bitte den Radius eingeben.\n");
    scanf("%d",&radius);
    flaeche=2*PI*radius*radius;
    umfang=2*PI*radius;
    printf("Der Kreisumfang betraegt %f \n",umfang);
    printf("Die Kreisflaeche betraegt %f \n",flaeche);
    return 0;
}
```

38

## Zusammenfassung: Variablen und Konstanten

Rechenzentrum

- Speicherstellen für Daten (Zahlen und Zeichen)
- Datentypen in C:
  - char: Zeichen
  - short, long und int: Ganzzahlen
  - float und double: Gleitkommazahlen
- Variablen müssen vor der ersten Verwendung deklariert und initialisiert werden
  - `datentyp name = wert;`
- Konstanten werden mit dem Schlüsselwort const deklariert:
  - `const dentyp NAME = wert;`

39

## Anweisungen und Ausdrücke

Rechenzentrum

- Was ist eine Anweisung ?
- Was ist ein Ausdruck ?

40

- Anweisungen
  - vollständige Vorschrift an den Compiler eine bestimmte Aufgabe auszuführen
  - C-Anweisungen sind immer mit einem Semikolon abzuschließen
    - Ausnahme: Präprozessoranweisungen z.B. `#include <stdio.h>`
  - Verbundanweisung (Block):
    - Gruppe von C-Anweisungen, die in geschweiften Klammern stehen
  - Anweisungen werden in C von rechts nach links ausgewertet
  - Beispiel:
    - `summe = x + y;`
    - `printf("Hallo C-Programmierer.\n");`

41

- besteht aus einem oder mehreren Operanden, die miteinander durch Operatoren verknüpft sind
- Ausdrücke haben als Ergebnis immer einen numerischen Wert
- sind eine Verarbeitungsvorschrift
- liefern ein Ergebnis in Abhängigkeit der Verarbeitung und Operanden
- bestehen aus Operanden und Operatoren, die nach festen Regeln zusammen gesetzt sind
- Beispiel:  $(x+y)*z$

42

- Grundlagen der Ein- und Ausgabeanweisungen in C
- Informationen auf dem Bildschirm ausgeben mit der Funktion `printf`
- Informationen von der Tastatur einlesen mit der Funktion `scanf`

43

- Wird realisiert durch die Bibliotheksfunktion `printf`
- Allgemeine Syntax:
 

```
#include <stdio.h>
printf(formatstring);
```
- Ausgabe einer Textmeldung:
 

```
printf("In Ihrem Programm ist ein Fehler aufgetreten ! \n");
```

Ausgabe: *In Ihrem Programm ist ein Fehler aufgetreten !*
- Ausgabe von Text und Programmvariablen
 

```
printf("Der Wert von x ist %d \n",xwert);
```

Ausgabe: *Der Wert von x ist 12*

44

## Formatstrings der Funktion printf

Rechenzentrum

- gibt an, wie die Ausgabe formatiert werden soll
- allgemein:
 

```
#include <stdio.h>
printf(Formatstring);
```
- Bsp: `printf("der Wert von x ist %d \n",xwert);`
- enthält folgende Informationen:
  - Literalen Text
  - Escape-Sequenzen: z.B: `\n` für neue Zeile
  - Formatbezeichner: gibt Format der auszugebenden Variablen an
    - `%d` : Integer-Zahl (int)
    - `%f`: Gleitkommazahl (float)
    - `%lf`: Gleitkommazahl (double)
    - `%s`: Zeichenstring (char-Array)
    - `%c`: Einfaches Zeichen (char)
  - Argument: Variablenname
- für jede Variable muss ein Formatbezeichner vorhanden sein :
 

```
printf("Die Summe von %d und %d ist %d \n",x,y,z);
```

Programmausgabe: *Die Summe von 5 und 4 ist 9.*

45

## printf-Funktion (Beispiel)

Rechenzentrum

```
/* Beispiel fuer printf-Funktion */
#include <stdio.h>
int main()
{
    int x=10;
    float f=12.25;
    char ergebnis='j';
    printf("Nur Text wird ausgegeben. \n");
    printf("Eine Integer-Variable hat den Wert %d. \n",x);
    printf("Eine float-Variable hat den Wert %f. \n",f);
    printf("Textausgabe mit einem Zeichen: %c. \n",ergebnis);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

- häufige Fehlerquelle:
  - Datentypen vom Formatbezeichner und Variablen stimmen nicht überein

46

## printf-Funktion (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int x=10;
    float f=12.25;
    char ergebnis='j';
    printf("Nur Text wird ausgegeben \n");
    printf("Eine Integer-Variable hat den Wert %d \n",x);
    printf("Eine float-Variable hat den Wert %f \n",f);
    printf("Textausgabe mit einem Zeichen: %c\n",ergebnis);
    printf("Programm erfolgreich beendet\n");
    return 0;
}
```

Programmübersetzung:  
`gcc -o p2.exe printf.c`  
 Programm starten:  
`./p2.exe`

Programmausgabe:  
 Ein Text wird ausgegeben.  
 Eine Integer-Variable hat den Wert 10.  
 Eine float-Variable hat den Wert 12.25  
 Textausgabe mit einem Zeichen j.  
 Programm beendet

47

## Escape-Sequenzen

Rechenzentrum

- nicht druckbare Steuerzeichen
- Zeichenkombinationen zur Darstellung nicht direkt angegebbarer Zeichen
- stellen immer ein einzelnes Zeichen dar
- beginnen mit einem Backslash (`\`)
- Beispiele:
  - `\n` bewegt den Cursor auf die Anfangsposition der nächsten Zeile
  - `\"` gibt das Zeichen " aus
  - `\'` gibt das Zeichen ' aus
  - `\?` gibt das ? aus
  - `\\` gibt den Backslash (`\`) aus
  - `\t` setzt den Cursor auf die nächste horizontale Tabulatorposition
  - `\v` setzt den Cursor auf die nächste vertikale Tabulatorposition

48

## Daten einlesen

Rechenzentrum

- erfolgt mit der Bibliotheksfunktion `scanf`
- allgemeine Syntax:  

```
#include <stdio.h>
scanf(Formatstring);
```
- Beispiele:  

```
scanf("%d", &wert); ---> Integer-Zahl einlesen
scanf("%f", &wert); ---> Gleitkommazahl einlesen
scanf("%d %f", &wert, &wert); ---> 2 Variablen einlesen
```
- Adressoperator `&`: gibt die genaue Adresse im Speicher an

49

## scanf-Funktion (Beispiel)

Rechenzentrum

```
/* Daten einlesen mit scanf */
#include <stdio.h>
int main()
{
    int xwert;
    float ywert;
    printf("Bitte geben Sie einen Integer und eine Gleitkommazahl ein.\n");
    scanf("%d %f", &xwert,&ywert);
    printf("Ihre Eingabe lautet: %d und %f. \n",xwert,ywert);
    printf("Die Speicheradressen lauten: %d und %d.\n",&xwert,&ywert);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

50

## scanf-Funktion (Beispiel)

Rechenzentrum

```
/* Daten einlesen mit scanf */
#include <stdio.h>
int main()
{
    int xwert;
    float ywert;
    printf("Bitte geben Sie einen Integer und eine
    Gleitkommazahl ein.\n");
    scanf("%d %f", &xwert,&ywert);
    printf("Ihre Eingabe lautet: %d und %f.
    \n",xwert,ywert);
    printf("Die Speicheradressen lauten: %d und
    %d.\n",&xwert,&ywert);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

- Programm übersetzen:  

```
- gcc -o scanf.exe scanf.c
```
- Programm starten:  

```
- ./scanf.exe
```
- Programmausgabe  
 Bitte geben Sie einen Integer und eine  
 Gleitkommazahl ein.  
 2 10.25  
 Ihre Eingabe lautet: 2 und 10.250000.  
 Die Speicheradressen lauten: 12223 und  
 1223455  
 Programm erfolgreich beendet.

51

## Zusammenfassung: Ein- und Ausgabe

Rechenzentrum

- Bildschirmausgabe mit der `printf`-Funktion:
  - Textausgabe
    - `printf("Dies ist eine reine Textausgabe.\n");`
  - Text und Variablenwerte ausgeben:
    - `printf("Der Wert von x ist %d \n",xwert);`
- Einlesen von der Tastatur
  - z.B. `scanf("%d",&xwert);`
- Formatbezeichner:
  - `%d`: Integer-Zahl
  - `%f`: Gleitkommazahl
  - `%s`: Zeichenstring
  - `%c`: Zeichen
  - `%%`: Prozentzeichen (%) ausgeben
- Bibliotheksfunktion `stdio.h` einfügen
  - `#include <stdio.h>`

52

# Übungsblatt 1

## Aufgaben 1 und 2

53

- der Zuweisungsoperator ist das Gleichheitszeichen (=)
- allgemeine Form der Wertzuweisung:  
var = ausdruck;  
z.B. summe = x + y;
- Nach der Auswertung des Ausdrucks auf der rechten Seite wird sein Wert der Variablen auf der linken Seite zugewiesen
- rechte Seite kann beliebiger Ausdruck sein
- linke Seite muss ein Variablenname sein
- Zuweisungsoperator darf nicht mit dem Gleichheitszeichen verwechselt werden

55

- sind Symbole zum Verknüpfen von Zahlen, Variablen und Ausdrücken
  - z.B.: +, -, \*, /
- können einen, zwei oder drei Operanden miteinander verbinden
- Ein Operator ist ein Symbol, mit dem man in C eine Operation oder Aktion auf einen oder mehreren Operanden vorschreibt.
- Ein Operand ist ein Element, das der Operator verarbeitet:
  - Beispiel: (3+2)\*8
- C-Operatoren lassen sich in mehrere Kategorien einteilen:
  - Zuweisungsoperator
  - arithmetische Operatoren
  - relationale Operatoren
  - logische Operatoren

54

- 7 mathematische Operatoren
  - fünf binäre Operatoren
  - zwei unäre Operatoren

56

## Binäre arithmetische Operatoren

Rechenzentrum

- wirken immer auf zwei Operanden
- 5 binäre Operatoren:
  - Addition: +
  - Subtraktion: -
  - Multiplikation: \*
  - Division: /
  - Modulo-Operator: %

57

## Modulo-Operator

Rechenzentrum

- Modulo-Operator %
- liefert den Rest einer ganzzahligen Division
- Beispiele:
  - 100 % 9 ---> 1
  - 10 % 5 ---> 0
  - 40 % 6 ---> 4
- nur auf Integer-Zahlen anwendbar und nicht auf Gleitkommazahlen

58

## Operatoren (Beispiel)

Rechenzentrum

```
/* Beispiel für Modulo-Operator */
#include <stdio.h>
int main()
{
    int x=5, y=2;
    int d1, d2;
    d1=x/y;
    d2=x%y;
    printf("Division: %d / %d = %d \n",x,y,d1);
    printf("Modulo: %d %% %d = %d \n",x,y,d2);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

59

## Operatoren (Beispiel)

Rechenzentrum

```
/* Beispiel für Modulo-Operator */
#include <stdio.h>
int main()
{
    int x=5, y=2;
    int d1, d2;
    d1=x/y;
    d2=x%y;
    printf("Division: %d / %d = %d \n",x,y,d1);
    printf("Modulo: %d %% %d = %d \n",x,y,d2);
    printf("Programm erfolgreich beendet. \n");
    return 0;
}
```

- Programm übersetzen:
  - gcc -o mod.exe mod1.c
- Programm ausführen:
  - ./mod.exe
- Programmausgabe
  - Division 5 / 2 ist 2
  - Divisionsrest von 5 / 2 ist 1
  - Programm erfolgreich beendet.

60

- Laufzeitbibliothek math.h enthält alle wichtigen mathematischen Funktionen
  - Beispiele:
    - Quadratwurzel sqrt
    - Potenzfunktion pow
    - Sinus, Kosinus und Tangens
- #include <math.h> in Code einfügen
- Variablen sind stets vom Datentyp double
- für Programmübersetzung mit gcc ist -lm Compilerflag erforderlich
  - gcc -o prog.exe prog.c -lm

61

```

/* Quadratwurzel bilden */
#include <stdio.h>
#include <math.h>
int main()
{
    double var2, var1;
    var1=169;
    var2=sqrt(var1);
    printf("Die Quadratwurzel von %lf ist %lf. \n", var1,var2);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
    
```

62

```

/* Quadratwurzel bilden */
#include <stdio.h>
#include <math.h>
int main()
{
    double var2, var1;
    var1=169;
    var2=sqrt(var1);
    printf("Die Quadratwurzel von %f ist %f. \n",
        var1,var2);
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
    
```

- Programm übersetzen
  - gcc -o sqrt.exe sqrt.c -lm
- Programm ausführen:
  - ./sqrt.exe
- Programmausgabe:
  - Die Quadratwurzel von 169 ist 13.0000
  - Programm erfolgreich beendet.

63

- Zuweisungsoperator:
  - var = ausdrück1;
- binäre mathematische Operatoren
  - Addition: +
  - Subtraktion: -
  - Multiplikation \*
  - Division: /
  - Modulo-Operator: % → Divisionsrest
- mathematische Bibliotheksfunktionen
  - Headerdatei einfügen : #include <math.h>
  - Variablen vom Datentyp double
  - Programmübersetzung mit -lm Compilerflag

64



- C ist eine sehr typenstrenge Programmiersprache
- Datentypen müssen konform sein

- Beispiel:

```
#include <stdio.h>
int main()
{
    int wert_int=0;
    float wert_float=10.1234;
    wert_int=wert_float;
    printf("wert_int ist %d ist \n",wert_int);
    return 0;
}
```

65

- C ist eine sehr typenstrenge Programmiersprache
- Datentypen müssen konform sein

- Beispiel:

```
#include <stdio.h>
int main()
{
    int wert_int=0;
    float wert_float=10.1234;
    wert_int=wert_float;
    printf("wert_int ist %d ist \n",wert_int);
    return 0;
}
```

→ Ausgabe: wert\_int ist 10

66

- implizite Datentypumwandlung
  - automatische Konvertierung durch den Compiler
- explizite Datentypumwandlung
  - Programmierer führt Typumwandlung manuell aus

67

```
#include <stdio.h>
int main()
{
    int int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=int1/int2;
    printf("Das Ergebnis ist %f \n",ergebnis);
    return 0;
}
```

→ Ausgabe: Das Ergebnis ist 0.000 → Fehler

68

## explizite Datentypumwandlung

Rechenzentrum

- erfolgt mit dem cast-Operator
- (datentyp) ausdrück;
- Beispiel:
 

```
int x,y;
ergebnis=(float)x/y;
```
- nicht auf Strings (Zeichenketten) anwendbar

69

## explizite Datentypumwandlung

Rechenzentrum

```
#include <stdio.h>
int main()
{
    int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=int1/int2;
    printf("Das Ergebnis ist %f\n",ergebnis);
    return 0;
}
```

→ Ausgabe: Das Ergebnis ist 0.000  
→ Fehler

```
#include <stdio.h>
int main()
{
    int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=(float) int1/int2;
    printf("Das Ergebnis ist %f\n",ergebnis);
    return 0;
}
```

→ Ausgabe: Das Ergebnis ist 0.75000

70

## Arithmetische Operatoren

Rechenzentrum

- 7 mathematische Operatoren
  - zwei unäre Operatoren
  - fünf binäre Operatoren

71

## Unäre mathematische Operatoren

Rechenzentrum

- unäre Operatoren wirken nur auf einen Operanden
- sind nur auf Variablen und nicht auf Konstanten anwendbar
- in C gibt es zwei unäre Operatoren:
  - Inkrementoperator: ++
    - erhöht einen Operanden um 1
    - variable = variable + 1;
  - Dekrementoperator: ---
    - erniedrigt den Operanden um 1
    - variable = variable – 1;
- x++ hat eine andere Bedeutung als ++x
- unäre Operatoren können sowohl vor dem Operanden (Präfix-Modus) oder nach dem Operanden (Postfix-Modus) stehen.
- können nur auf einfache Variablen angewendet werden, nicht dagegen auf Ausdrücke wie etwa (i+j)++;
- werden vorwiegend bei Schleifen verwendet

72

- Präfixmodus
  - Inkrement-/Dekrementoperator wirkt auf den Operanden, bevor er verwendet wird
  - Variable wird erst dekrementiert/inkrementiert und anschließend einer Variablen zugewiesen
  - ++x ist äquivalent zu x=x+1;
  - --y ist äquivalent zu y=y-1;
- Postfix-Modus
  - Inkrement-/Dekrementoperator wirkt auf den Operanden, nachdem er verwendet wurde
  - z.B.: y=x++
  - Wert der Variablen rechts vom Zuweisungsoperator wird der Variablen links zugewiesen. Anschließend wird die Variable dekrementiert / inkrementiert

73

- Inkrementierung
  - Präfix: x=10; und y= ++x; ---> x=11; und y=11;
  - Postfix: x=10 und y=x++; ----> x=11; und y=10;
- Dekrementierung
  - Präfix: x=10; und y=--x ; ---> x=9; und y=9;
  - Postfix: x=10 und y=x--; --> x=9; und y=10;

74

```
/* Beispiel für In-/Dekrementoperationen */
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

75

```
/* Beispiel für In-/Dekrementoperationen */
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

• Programmausgabe:

Wert von x: 0

76

/\* Beispiel für In-/Dekrementoperationen • Programmausgabe:

```
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

Wert von x: 0  
Wert von x: 1

77

/\* Beispiel für In-/Dekrementoperationen • Programmausgabe:

```
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

Wert von x: 0  
Wert von x: 1  
Wert von x: 1

78

/\* Beispiel für In-/Dekrementoperationen • Programmausgabe:

```
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

Wert von x: 0  
Wert von x: 1  
Wert von x: 1  
Wert von x: 2

79

/\* Beispiel für In-/Dekrementoperationen • Programmausgabe:

```
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

Wert von x: 0  
Wert von x: 1  
Wert von x: 1  
Wert von x: 2  
Wert von x: 2

80

## Operatoren (Beispiel)

Rechenzentrum

/\* Beispiel für In-/Dekrementoperationen • Programmausgabe:

```
#include <stdio.h>
int main()
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

```
Wert von x: 0
Wert von x: 1
Wert von x: 1
Wert von x: 2
Wert von x: 2
Wert von x: 0
```

81

## Rangfolge der Operatoren

Rechenzentrum

- In C hat jeder Operator eine bestimmte Priorität
- Wert von x in diesem Beispiel ist ?
  - $x=4+5*3$ ;
- Priorität nimmt in der folgenden Reihenfolge ab:
  - Inkrement-/Dekrementoperationen
  - Multiplikation, Division und Modulo-Operator
  - Addition und Subtraktion
- Operatoren mit höherer Priorität werden zuerst ausgewertet
- bei gleicher Priorität erfolgt die Auswertung von links nach rechts
- Verwenden Sie Klammern, um die Auswertungsreihenfolge eindeutig festzulegen

82

## Relationale Operatoren

Rechenzentrum

- Vergleichsoperatoren (dienen zum Vergleich von Ausdrücken)
- Ergebnis ist entweder wahr oder falsch
- Ergebnis falsch entspricht Zahlenwert von Null
- Ergebnis wahr entspricht Zahlenwert von ungleich Null meist 1
- werden häufig für Bedingungen und Schleifen verwendet
- 6 Vergleichsoperatoren
  - größer als (>): Bsp.:  $9>5 \rightarrow$  wahr
  - größer gleich (>=) : Bsp.:  $9 \geq 5 \rightarrow$  wahr
  - kleiner (<): Bsp.:  $9 < 5 \rightarrow$  falsch
  - kleiner gleich (<=): Bsp.:  $9 \leq 5 \rightarrow$  falsch
  - gleich (==): Bsp.:  $9==5 \rightarrow$  falsch
  - ungleich (!=): Bsp.:  $0!=5 \rightarrow$  wahr
- Operatoren == und != haben geringere Priorität als die anderen Operatoren
  - $x == y > z$  entspricht  $x == (y > z)$

83

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;
    printf("Ergebnis ist %d\n",a<b);
    printf("Ergebnis ist %d\n", b<c);
    printf("Ergebnis ist %d\n",a+c<=b);
    printf("Ergebnis ist %d\n",b-a>=c);
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

84

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c);
    printf("Ergebnis ist %d\n",a+c<=b);
    printf("Ergebnis ist %d\n",b-a>=c);
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

85

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b);
    printf("Ergebnis ist %d\n",b-a>=c);
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

86

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c);
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

87

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

88

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

89

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

90

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a!=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

91

## Vergleichsoperatoren (Beispiel)

Rechenzentrum

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main()
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a!=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a<b<c); ---> Ergebnis ist 1
    return 0;
}
```

92

- verwendet, wenn mehrere Vergleiche auf einmal ausgeführt werden sollen
- mehrere relationale Ausdrücke werden zu einem Ausdruck zusammen gefasst
- 3 logische Operatoren:
  - AND-Operator (&&)
    - `ausdruck1 && ausdruck2`
    - nur wahr, wenn `ausdruck1` und `ausdruck2` wahr sind
    - Bsp.: `(5==5) && (6!=2)`
  - OR-Operator (||)
    - `ausdruck1 || ausdruck2`
    - wahr sobald mind. eine Bedingung wahr ist
    - Bsp.: `((5>1) || (6<1))`
  - NOT-Operator (!)
    - `!(ausdruck)`
    - falsch, wenn `ausdruck` wahr ist
    - Bsp.: `!(5==4)`

93

- Möglichkeit eine binäre mathematische Operation mit einer Zuweisung zu kombinieren
- z.B.: `x=x+5`; identisch mit `x += 5`;
- allg. Form: `ausd1 op= ausd2`
- Operator `+=` heißt verkürzter Zuweisungsoperator
- anwendbar auf: `+`, `-`, `*`, `/`, `%`
- Beispiele:
  - `x += y`;  $\rightarrow x = x + y$ ;
  - `y -= z+1`  $\rightarrow y = y - z + 1$ ;
  - `a /= b`;  $\rightarrow a = a / b$ ;
  - `x += y/8`;  $\rightarrow x = x + y/8$ ;
  - `y %= 3`;  $\rightarrow y = y \% 3$ ;
- reduzieren den Schreibaufwand

94

- Operatoren mit höherer Priorität haben Vorrang
- Priorität nimmt in folgender Reihenfolge ab:
  1. unäre Inkrement- und Dekrementoperationen
  2. Multiplikation, Division und Modulo-Operator
  3. Addition und Subtraktion
  4. Vergleichsoperatoren: `<`, `>`, `<=` und `>=`
  5. Vergleichsoperatoren: `=` und `!=`
  6. Logische Operatoren: `&&`, `||` und `!`

95

- Zuweisungsoperator
  - `=`
- unäre mathematische Operatoren
  - `x++`, `++x`, `x--` und `--x`
- binäre mathematische Operatoren:
  - `+`, `-`, `*`, `/` und `%`
- Vergleichsoperatoren:
  - `<`, `>`, `<=`, `>=`, `=` und `!=`
- logische Operatoren
  - `&&`, `||` und `!`
- Rangfolge der Operatoren beachten

96



- Wie lauten die Ergebnisse der folgenden Ausdrücke:
  - $(1 + 2 * 3) = ?$
  - $10 \% 3 * 3 - (1 + 2) = ?$
  - $((1 + 2) * 3) = ?$
  - $(5 == 5) = ?$
- Stellen Sie fest, ob die folgenden Ausdrücke wahr oder falsch sind
  - $5 == 1$
  - $5 > 1$
  - $5 != 1$
  - $(5 + 10) == (3 * 5)$

97

- Wie lauten die Ergebnisse der folgenden Ausdrücke:
  - $(1 + 2 * 3) = 7$
  - $10 \% 3 * 3 - (1 + 2) = 0$
  - $((1 + 2) * 3) = 9$
  - $(5 == 5) = 1$
- Stellen Sie fest, ob die folgenden Ausdrücke wahr oder falsch sind
  - $5 == 1 \rightarrow 0$  (falsch)
  - $5 > 1 \rightarrow 1$  (wahr)
  - $5 != 1 \rightarrow 1$  (wahr)
  - $(5 + 10) == (3 * 5) \rightarrow 1$  (wahr)

98

## Übungsblatt 1

### Aufgaben 3 bis 7

99

- beeinflussen den Programmablauf
- in Abhängigkeit vom Wert einer oder mehrerer Variablen wird der Programmablauf gesteuert
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung
  - Schleifen (Iterationsanweisungen):
    - kopfgesteuerte Schleifen: while-Schleife
    - fußgesteuerte Schleifen: do-while-Schleife
    - Zählschleife: for-Schleifen

100

## if-Anweisung

Rechenzentrum

- Code wird nur unter bestimmten Bedingungen ausgeführt
- allgemeine Form
 

```
if (Bedingung)
{
    Anweisung1;
    Anweisung2;
}
Anweisung3;
```
- Anweisungen 1 und 2 werden nur ausgeführt, wenn die Bedingung wahr ist
- wenn Bedingung nicht wahr ist, wird gleich Anweisung 3 ausgeführt

101

## if-Anweisung (Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein.\n");
    scanf("%d %d",&x, &y);
    if (x == y)
        printf("x ist gleich y \n");
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

102

## if-Anweisung (fehlerhaftes Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein.\n");
    scanf("%d %d",&x, &y);
    if (x == y) ;
        printf("x ist gleich y \n");
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

103

## if-Anweisung (fehlerhaftes Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
    scanf("%d %d",&x, &y);
    if (x == y) ;
        printf("x ist gleich y \n"); --> Zeile wird immer ausgegeben
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

104

## if-Anweisung (fehlerhaftes Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
    scanf("%d %d",&x, &y);
    if (x == y) ;
        printf("x ist gleich y \n"); --> Zeile wird immer ausgegeben
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

--> if-Anweisung nie mit einem Semikolon abschließen !!

105

## if-Anweisung (Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
    scanf("%d %d",&x, &y);
    if (x == y)
        printf("x ist gleich y \n");
    if(x>y)
        printf("x ist groesser als y \n ");
    if (x<y)
        printf("x ist kleiner als y \n");
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

106

## if-else-Anweisung

Rechenzentrum

- wird verwendet, wenn mehrere Auswahlmöglichkeiten vorliegen
- allgemeine Struktur
 

```
if (Bedingung1)
    Anweisung1;
else if (Bedingung2)
    Anweisung2;
else
    Anweisung3;
naechste-Anweisung;
```

107

## if-else-Anweisung (Beispiel)

Rechenzentrum

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main()
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
    scanf("%d %d",&x, &y);
    if (x == y)
        printf("x ist gleich y \n");
    else if (x>y)
        printf("x ist groesser als y \n");
    else
        printf("x ist kleiner als y \n");
    printf("Programm erfolgreich beendet.\n");
    return 0;
}
```

108

## switch-Anweisung

Rechenzentrum

- dient wie if-Bedingungen zur Formulierung von Mehrwegeentscheidungen
- Wert eines Ausdrucks wird mit einer Liste von Konstanten vom Datentyp char oder int verglichen
- jeder Vergleich wird mit case (Fall) eingeleitet
- es ist nur ein Vergleich auf Gleichheit möglich

109

## switch-Anweisung

Rechenzentrum

- allgemeine Form:  

```
switch (Ausdruck)
{
    case konstante1:
        anweisung1;
        break;
    case konstante2:
        anweisung2;
        break;
    ....
    case konstanten:
        anweisung;
        break;
    default:
        anweisung;
}
naechste-Anweisung;
```

110

## switch-Anweisung (Beispiel)

Rechenzentrum

```
#include <stdio.h>
int main()
{
    char eingabe;
    printf("Bitte geben Sie Ihr Geschlecht ein. \n");
    printf("w für weiblich und m für männlich\n");
    scanf("%c", &eingabe);
    switch (eingabe)
    {
        case 'w' :
            printf("Weiblich\n");
            break;
        case 'm':
            printf("Männlich \n");
            break;
        default:
            printf("falsche Eingabe \n") ;
    }
}
```

111

## Kontrollstrukturen II

Rechenzentrum

- Iterationsanweisungen (Schleifen)
- dienen zur wiederholten Ausführung von Anweisungen
- in C unterscheidet man drei unterschiedliche Schleifenarten
  - for-Schleife
  - while-Anweisung
  - do-while-Anweisung

112

## for-Schleife

Rechenzentrum

- allgemeine Struktur:  
 for (Initialisierung; Bedingung; Inkrement)  
 {  
     Anweisungen  
 }  
 naechste-Anweisung;
- Initialisierung: legt Anfangswert für Zählvariable fest
- Bedingung: Schleife bricht ab, wenn Bedingung nicht mehr erfüllt ist
- Inkrement: legt fest, wie die Zählvariable nach jedem Durchlauf verändert wird
- wird verwendet, wenn die Anzahl der Schleifendurchläufe vorher bekannt ist
- Schleifen dürfen verschachtelt werden.
- for-Schleife ist eine Zählschleife

113

## for-Schleife (Beispiel)

Rechenzentrum

```
/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main()
{
    int x=0;
    for (x=0; x<5;x++)
    {
        printf("Der Wert von x ist: %d \n",x);
    }
    printf("Programm beendet \n");
    return 0;
}
```

114

## for-Schleife (Beispiel)

Rechenzentrum

```
/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main()
{
    int x=0;
    for (x=0; x<5;x++)
    {
        printf("Der Wert von x ist: %d. \n",x);
    }
    printf("Programm beendet \n");
    return 0;
}
```

- Programm übersetzen:  
 – gcc -o for.exe for.c
- Programm ausführen:  
 – ./for.exe
- Programmausgabe  
 Der Wert von x ist 0.  
 Der Wert von x ist 1.  
 Der Wert von x ist 2.  
 Der Wert von x ist 3.  
 Der Wert von x ist 4.

115

## while-Schleife

Rechenzentrum

- allgemeine Form  
 while (Bedingung)  
     Anweisung;  
 naechste-Anweisung;
- Anweisung wird solange ausgeführt solange die Bedingung wahr ist
- enthält keinen Initialisierungsschritt
- Initialisierung muss vor der Schleife erfolgen
- Inkrementierungsschritt muss innerhalb der Schleife erfolgen
- Endlosschleife entsteht, wenn Schleifenvariable nicht richtig geändert wird
- wird auch als abweisende Schleife bezeichnet
- wird verwendet, wenn Schleife bei einem bestimmten Ereignis abgebrochen werden soll

116

## while-Schleife (Beispiel)

Rechenzentrum

```

/* Beispiel für eine while-Schleife */
#include <stdio.h>
int main()
{
    int eingabe;
    int zaehler=1;
    printf("Bitte Anzahl der Schleifendurchläufe eingeben. \n");
    scanf ("%d", &eingabe);
    while (zaehler<=eingabe)
    {
        printf("Schleifendurchlauf  %d \n",zaehler);
        zaehler=zaehler+1;
    }
    printf("Programm beendet \n");
    return 0;
}
    
```

117

## while-Schleife (Beispiel)

Rechenzentrum

```

/* Beispiel für eine while-Schleife */
#include <stdio.h>
int main()
{
    int eingabe;
    int zaehler=1;
    printf("Bitte Anzahl der Schleifendurchläufe eingeben. \n");
    scanf ("%d", &eingabe);
    while (zaehler<=eingabe)
    {
        printf("Schleifendurchlauf  %d \n",zaehler);
        zaehler=zaehler+1;
    }
    printf("Programm beendet \n");
    return 0;
}
    
```

- Programm übersetzen  
– gcc -o while.exe while.c
- Programm ausführen:  
– ./while.exe
- Programmausgabe:  
Bitte Anzahl der Schleifendurchläufe eingeben.  
4  
Schleifendurchlauf 1  
Schleifendurchlauf 2  
Schleifendurchlauf 3  
Schleifendurchlauf 4

118

## Vergleich: for- und while Schleife

Rechenzentrum

```

/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main()
{
    int x;
    for (x=0; x<5;x++)
    {
        printf("Der Wert von x ist: %d. \n",x);
    }
    printf("Programm beendet \n");
    return 0;
}

/* Beispiel fuer eine while-Schleife */
#include <stdio.h>
int main()
{
    int x=0;
    while(x<5)
    {
        printf("Der Wert von x ist %d. \n",x);
        x=x+1;
    }
    printf("Programm beendet \n");
    return 0;
}
    
```

119

## do-while-Schleife

Rechenzentrum

- allgemeine Form  
do  
    Anweisung;  
while (Bedingung);  
    naechste-Anweisung;
- fußgesteuerte Schleife, da Bedingung erst am Ende abgefragt wird
- Schleife wird mindestens einmal durchlaufen
- wird auch als nichtabweisende Schleife bezeichnet
- wird relativ selten verwendet
- bis die Bedingung erfüllt ist, führe die Anweisung aus
- do-while-Schleife muss mit einem Semikolon abgeschlossen werden

120

do-while-Schleife (Beispiel)

Rechenzentrum

```
/* 1. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main()
{
    int count, ergebnis;
    count=1;
    ergebnis=0;
    do
    {
        ergebnis=ergebnis+count;
        printf("Das ergebnis ist: %d \n",ergebnis);
        count++;
    }
    while (count <=10);
    printf("Programm beendet \n");
    return 0;
}
```

121

do-while-Schleife (Beispiel)

Rechenzentrum

```
/* 1. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main()
{
    int count, ergebnis;
    count=1;
    ergebnis=0;
    do
    {
        ergebnis=ergebnis+count;
        printf("Das Ergebnis ist: %d \n",ergebnis);
        count++;
    }
    while (count <=10);
    printf("Programm beendet \n");
    return 0;
}
```

- Programmausgabe  
Das Ergebnis ist: 1  
Das Ergebnis ist: 3  
Das Ergebnis ist: 6  
Das Ergebnis ist: 10  
Das Ergebnis ist: 15  
Das Ergebnis ist: 21  
Das Ergebnis ist: 28  
Das Ergebnis ist: 36  
Das Ergebnis ist: 45  
Das Ergebnis ist: 55

122

do-while-Schleife (Beispiel)

Rechenzentrum

```
/* 2. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main()
{
    int summe=0;
    int eingabe;
    do
    {
        printf("Bitte Zahl zum Addieren eingeben (0 fuer Ende).\n");
        scanf("%d",&eingabe);
        summe=summe+eingabe;
    }
    while (eingabe !=0);
    printf("Die Summe ist %d.\n",summe);
    printf("Programm beendet \n");
    return 0;
}
```

123

do-while Schleife (Beispiel 2)

Rechenzentrum

- Programm übersetzen:  
– gcc -o while2.exe dowhile2.c
- Programm ausführen  
– ./while2.exe
- Programmausgabe:  
Bitte Zahl zum Addieren eingeben (0 fuer Ende).  
2  
Bitte Zahl zum Addieren eingeben (0 fuer Ende).  
4  
Bitte Zahl zum Addieren eingeben (0 fuer Ende).  
5  
Bitte Zahl zum Addieren eingeben (0 fuer Ende).  
0  
Die Summe ist 11.  
Programm beendet.

124

## Schleifen abbrechen

Rechenzentrum

- 2 Möglichkeiten Schleifen vorzeitig abzubrechen
  - **continue-Anweisung**
    - beendet nur den aktuellen Schleifendurchlauf
    - setzt Schleife fort, wenn Bedingung noch weiter erfüllt ist
  - **break-Anweisung**
    - beendet die gesamte Schleife
    - Programm wird nach der Schleife fortgesetzt

125

## continue-Anweisung (Beispiel)

Rechenzentrum

```
/* Schleifenabbruch mit continue */
#include <stdio.h>
int main()
{
    int i;
    for(i=-3; i<=3; i++)
    {
        if(i==0)
            continue;
        printf("Der Wert ist  %d\n",i);
    }
    printf("Program beendet\n");
    return 0;
}
```

126

## continue-Anweisung (Beispiel)

Rechenzentrum

```
/* Schleifenabbruch mit continue */
#include <stdio.h>
int main()
{
    int i;
    for(i=-3, i<=3, i++)
    {
        if(i==0)
            continue;
        printf("Der Wert ist:  %d\n",i);
    }
    printf("Program beendet\n");
    return 0;
}
```

- Programmausgabe:  
 Der Wert ist: -3  
 Der Wert ist: -2  
 Der Wert ist: -1  
 Der Wert ist: 1  
 Der Wert ist: 2  
 Der Wert ist: 3

127

## break-Anweisung (Beispiel)

Rechenzentrum

```
/* Beispiel fuer eine break-Anweisung */
#include<stdio.h>
int main()
{
    int eingabe;
    int i=1;
    for(i=1;i<10;i++)
    {
        printf("Bitte geben Sie eine Integer-Zahl ein. (Abbruch mit 999)\n");
        scanf("%d",&eingabe);
        if(eingabe == 999) break;
        printf("Sie haben die Zahl %d eingegeben.\n",eingabe);
    }
    printf("Programm beendet.\n");
    return 0;
}
```

128



- beeinflussen den Programmablauf
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung
  - Schleifen (Iterationsanweisungen):
    - kopfgesteuerte Schleifen: while-Schleife
    - fußgesteuerte Schleifen: do-while-Schleife
    - Zählschleife: for-Schleifen
- Schleifenabbruch:
  - continue-Anweisung: aktueller Schleifendurchlauf wird abgebrochen
  - break-Anweisung: gesamte Schleife wird beendet

## Übungsblatt 1

### Aufgaben 8 bis 16