75.04/96.12 Algoritmos y Programación II

Universidad de Buenos Aires – FIUBA
2º cuatrimestre 2014.

Trabajo práctico 1: Programación C++

Jeremías. Ignacio. Zec 92444.
Federico. Verstraeten 92892.

Contenido:

1. Introducción:

El desarrollo de este trabajo práctico tiene como objetivo continuar el desarrollo de la herramienta para procesar imagenes confeccionada en el primer trabajo practico entregado. De forma tal que el programa pueda recibir funciones de transformaciones arbitrarias.

2. Programa:

El programa deberá ser capaz de realizar las siguientes tareas:

- Cargar una imagen a memoria desde un archivo o desde la entrada estandar.
- Aplicar una función arbitraria, pasada por linea de comando, a la imagen.
- Guardar una imagen en memoria a un archivo o sacarlo por salida estardar.

2.1 Entrada:

Para resolver el ingreso y egreso de datos, se utilizó la clase cmdline provista por los docentes. Esta clase permite recorrer e identificar las opciones y argumentos con los que se invoca al programa mediante línea de comandos. Luego se definen una serie de métodos que permiten asignar funciones determinadas según las opciones leídas.
En particular, se definieron las funciones que deben corresponderse con los argumentos de entrada leídos y validados por cmdline() en archivos denominados *options.hpp*, *options.cpp*.
El tipo de opciones admitidas para este programa están detalladas en los diccionarios globales, que se encuentran en el archivo *dictionary.cpp*.

2.2 Carga en memoria:

Una vez procesadas las opciones se procede a la carga en memoria de la entrada, la cual puede provenir de un archivo, imagen .pgm en este caso, o del flujo estandar *cin*.
Para ello se lee el archivo y se lo procesa según las características del mismo. En nuestro caso:

Detección "Número Mágico (Magic Number)": A través de la función *'getMagicNumber(istraeam& ,string&)'* verificamos que el mismo sea P2, devolviendo falso y cortando el programa en caso contrario.

Lectura de los tamaños de la imagen: A través de la función *'readSize(istraeam&)'* verificamos que los mismo sean válidos, y se almacenan los valores de alto y ancho de la matriz de datos que representa a imagen.

Lectura máximo de intensidad de la imagen: A través de la función *'readMaxIntensity(istraeam& ,size_t&)'* verificamos que el mismo sea válido, y se almacenan los valor.

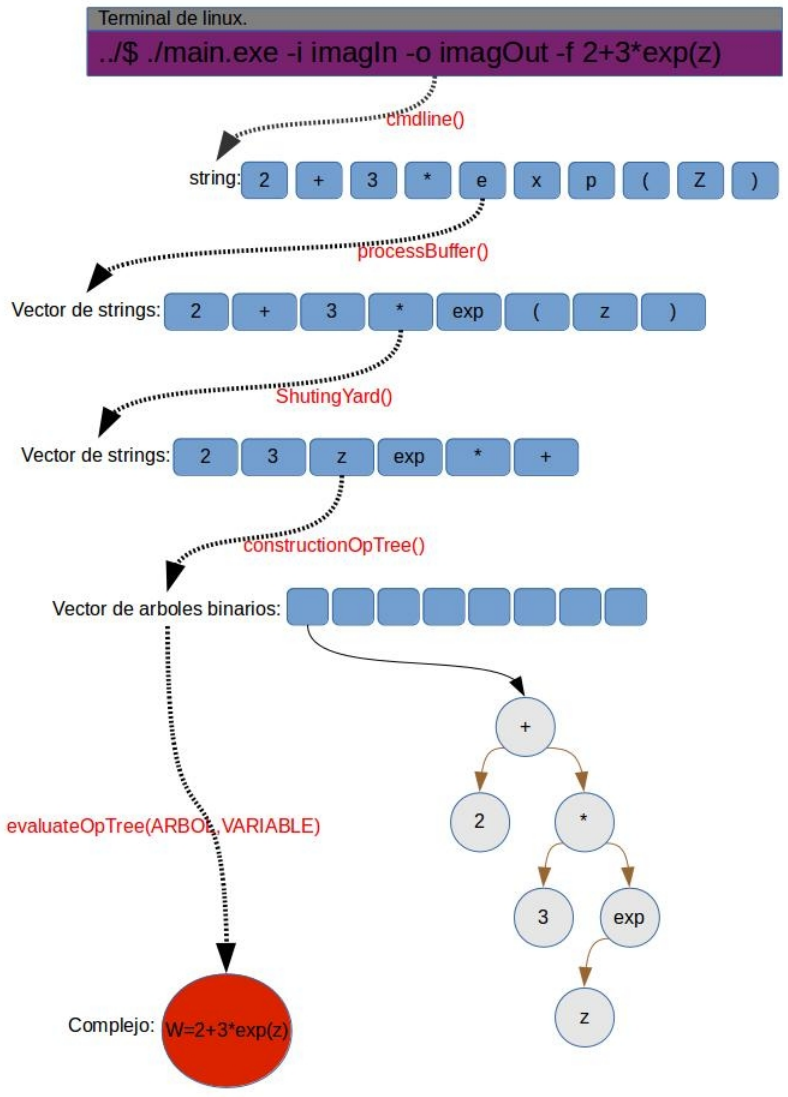
Creación de las matrices y lectura de la entrada: Otorgo memoria a través de la función *'createMatrix( )'*, según los datos obtenidos en los pasos anteriores, y leo la matriz de datos de la imagen de entrada (imagen Origen) para luego procesarla a través de la función *'readMatrixIN( )'*.
Las funciones anteriores poseen parámetros pero han sido omitidos.

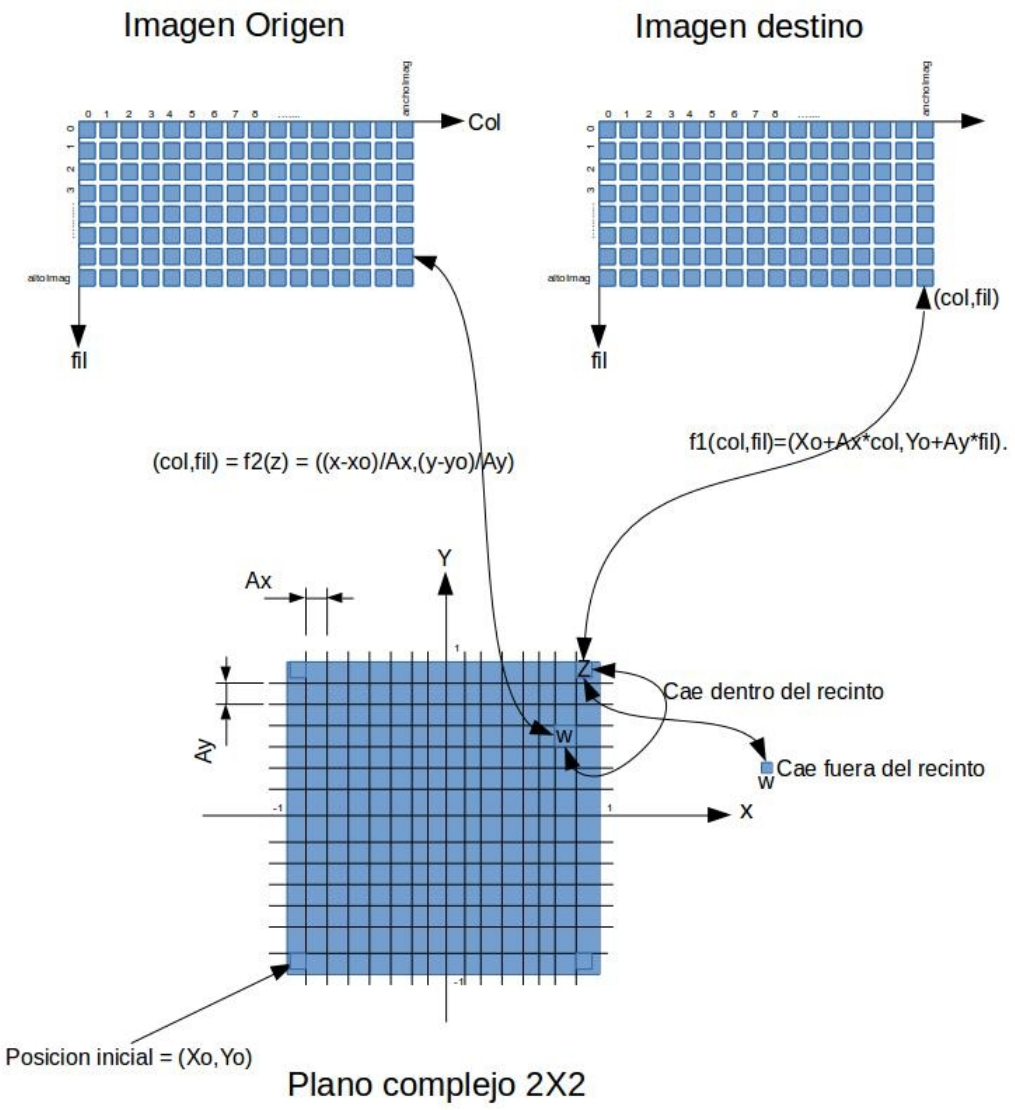
Todas las funciones anteriores devuelven ERROR en caso de haber inconvenientes en el procesamiento de la entrada y cortan la ejecución del progama, dando los mensajes de error respectivos por *cerr*. A su vez liberan toda la memoria dinámica solicitada por el programa.

2.3 Proceso:

Ilustración Gráfica del proceso llevado a cabo:

La siguiente ilustración representa el proceso que se repite para cada uno de los pixeles de la imagen destino, donde cada pixel se corresponde con un valor complejo "z", y un valor complejo "w" se corresponde con un pixel en la imagen de origen.
Como se muestra en la imagen el valor z se obtiene a través de la función f1 a partir de las coordenadas de un pixel de la imagen de destino, y w se obtiene aplicando sobre z la funcion ingresada a travez de lineas de comando.





Ax=2/anchoImag
Ay=2/altoImag

Xo=-1+(Ax/2)
Yo=-1+(Ay/2)

Algoritmo shunting yard:

Es el algoritmo para parsear las expresiones matemáticas ingresadas en notación infija por líneas de comando, producioendo una salida en notación polaca inversa (RPN).
El algoritmo *shunting yard* está utiliza cola y una pila. En la cola se va almacenando la salida con su orden correspondiente, mientras que en la pila se van apilando los operadores que con un criterio de precedencia, asociatividad y tipo.

Pseudocodigo:

- Mientras haya tokens a ser leídos:
  - Lea un token.
  - Si el token es un número, entonces agregúelo a la cola de salida.
  - Si el token es un token de función, entonces póngalo (push) sobre la pila.
  - Si el token es un separador de un argumento de función (ej., una coma):
  - Hasta que el token en el tope de la pila sea un paréntesis abierto, retire (pop) de la pila a los operadores y póngalos en la cola de salida. Si no es encontrado ningún paréntesis abierto, el separador fue colocado mal o los paréntesis fueron mal emparejados.

- Si el token es un operador, o1, entonces:
  - mientras que haya un operador, o2, en el tope de la pila (esto excluye el paréntesis abierto), y o1 es asociativo izquierdo y su precedencia es menor que (una precedencia más baja) o igual a la de o2, ó o1 es asociativo derecho y su precedencia es menor que (una precedencia más baja) que la de o2, retire (pop) de la pila el o2, y póngalo en la cola de salida; ponga (push) o1 en el tope de la pila.

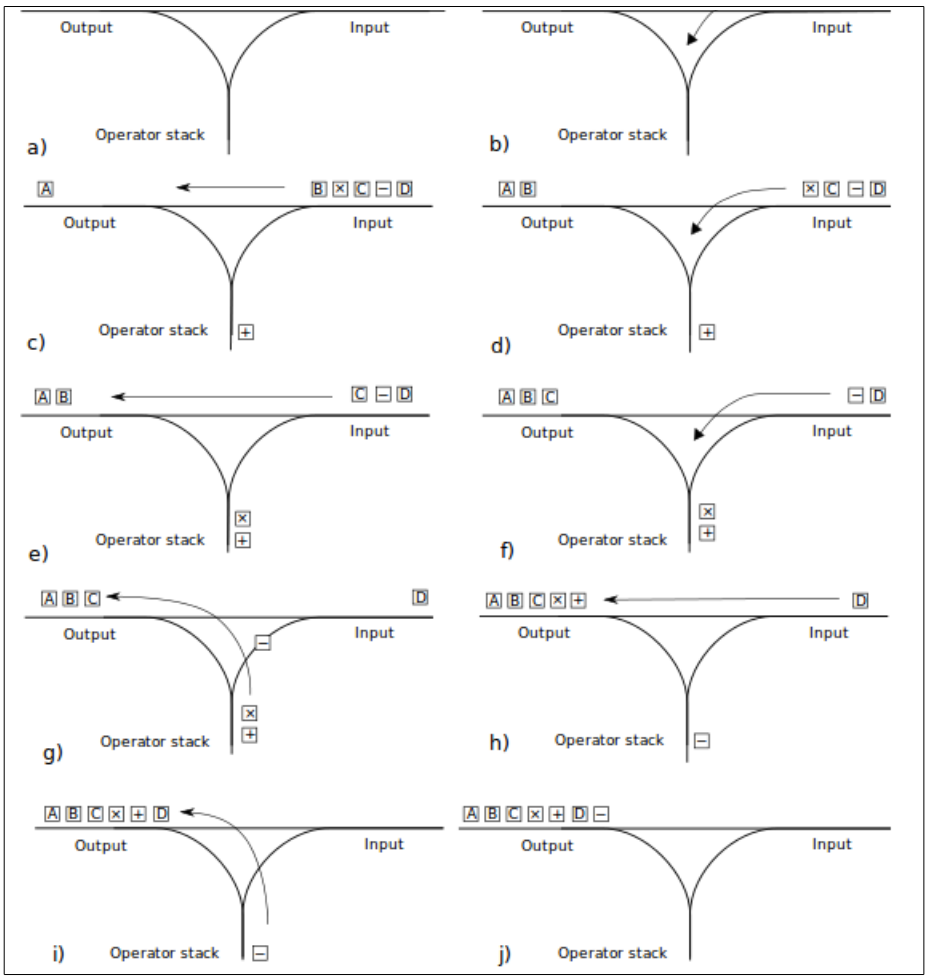
- Si el token es un paréntesis abierto, entonces póngalo en la pila:
  - Si el token es un paréntesis derecho.
  - Hasta que el token en el tope de la pila sea un paréntesis abierto, retire (pop) a los operadores de la pila y colóquelos en la cola de salida.
    Retire (pop) el paréntesis abierto de la pila, pero no lo ponga en la cola de salida.
  - Si el token en el tope de la pila es un token de función, póngalo (pop) en la cola de salida.
  - Si la pila se termina sin encontrar un paréntesis abierto, entonces hay paréntesis sin pareja.

- Cuando no hay más tokens para leer:
  - Mientras todavía haya tokens de operadores en la pila.
  - Si el token del operador en el tope de la pila es un paréntesis, entonces hay paréntesis sin la pareja correspondiente.
    retire (pop) al operador y póngalo en la cola de salida.

- Fin.

Nota: Para analizar la complejidad de tiempo de ejecución de este algoritmo, uno solo tiene que notar que cada token será leído solo una vez, cada número, función, u operador será impreso solo una vez, y cada función, operador o paréntesis será puesto (push) en la pila y retirado (pop) de la pila solo una sola vez - por lo tanto, hay a lo sumo un número constante de operaciones ejecutadas por token, y el tiempo de ejecución es así O(n) - lineal al tamaño de la entrada.

Ejemplo: ilustración gráfica del algoritmo utilizando un cruce de ferrocarril de tres vias.
Las operaciones que realiza son las mencionadas anteriormente en el pseudocódigo del algoritmo.



El algoritmo fue inventado por Edsger Dijkstra y nombró como algoritmo "shunting yard" porque su operación se asemeja al de un patio de clasificación del ferrocarril con tres vias.

Árbol de expresión binaria:

Una vez codificado el algoritmo *shunting yard* y tener la salida de la expresión matemática en notacion RPN.

Tabla de precedencias:

Operador	Precedencia	Asociatividad
^	4	de derecha a izquierda
*	3	de izquierda a derecha
/	3	de izquierda a derecha
+	2	de izquierda a derecha
-	2	de izquierda a derecha

En el código se encuentra implementada de la siguiente manera:

```
/*************************************************************
 *                    TABLA DE OPERACIONES                   *
 *************************************************************/
t_operation ops[]={

        //Operaciones
        {"^", 9, ASSOC_RIGHT,NOT_UNARY,OPERATOR, eval_pow},
        {"*", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, eval_mul},
        {"/", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, eval_div},
        {"+", 5, ASSOC_LEFT, BINARY_UNARY,OPERATOR, eval_add},
        {"-", 5, ASSOC_LEFT, BINARY_UNARY,OPERATOR, eval_sub},
        {"+_",10, ASSOC_LEFT, UNARY,OPERATOR, eval_unplus}, //suma unaria
        {"-_",10, ASSOC_LEFT, UNARY,OPERATOR, eval_unminus}, //negación unaria
        {",", 0, ASSOC_NONE, NOT_UNARY,SEPARATOR, NULL},
        {"(", 0, ASSOC_NONE, NOT_UNARY,PARENTESIS_OPEN, NULL},
        {")", 0, ASSOC_NONE, NOT_UNARY,PARENTESIS_CLOSE, NULL},
        //{"%", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, NULL},

        //variable independiente
        {"z",10, ASSOC_RIGHT,NOT_UNARY,VAR_INDEP,NULL},
        //unidad imaginaria
        {"j",10, ASSOC_NONE, NOT_UNARY,IMAGINARY_UNIT,NULL},

        //Funciones
        {"re",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_re},
        {"im",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_im},
        {"exp",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_exp},
        {"id",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_id},
        {"abs",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_abs},
        {"sinh",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_sinh},
        {"cosh",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_cosh},
        {"sin",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_sin},
        {"cos",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_cos},
        {"arg",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_arg},
        {0, },
```

2.4 Salida:

Finalmente una vez procesada la imagen de entrada, transformada y almacenada en memoria se procede a imrpimir la salida en un archivo imagen .pgm o por el flujo estándar *cout,* según lo seleccionado en las opciones por línea de comandos.
La función *'printImage( )'* se encarga de grabar los datos procesados en el archivo imagen de salida.
La función anterior posee parámetros pero han sido omitidos en la redacción del informe.

3. Testing:

3.1 Compilación:

La compilación del programa se realiza por medio de un Makefile. Como salida, se genera un archivo ejecutable de extensión '.exe'.
Para ejecutar el programa se necesita invocar con la siguiente línea de comandos:

./ main.exe – i < entrada.txt > –o < salida.txt > -f "<función>"

Las pruebas que se realizaron contemplan archivos correctos de entrada, y algunos ejemplos de diversas imágenes de distintos tamaños para las funciones codificadas.

Entradas correctas:

En las siguientes imágenes se puede apreciar el uso correcto del programa y la forma en que guarda la imagen procesada.

Función "exp(z)":

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imag.pgm
P2
# Commets test 1
# Commets test 2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0 15  0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11  0 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0  0  0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15  0  0  0  0
0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i imag.pgm -o imagout.pgm -
f "exp(z)"
Fuction enable: exp(z)
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imagout.pgm
P2
24 7
15
0 0 0 0 0 11 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 15 15 15 15 15 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 15 15 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 15 15 15 15 15 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 15 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 11 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

Función “z”:

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imag.pgm
P2
# Commets test 1
# Commets test 2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0 15  0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11  0 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0  0  0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15  0  0  0  0
0 0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i imag.pgm -o imagId.pgm -f
 "z"
Fuction enable: z
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imagId.pgm
P2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

De las misma manera se obtienen los siguientes resultados para otras imágenes de prueba.

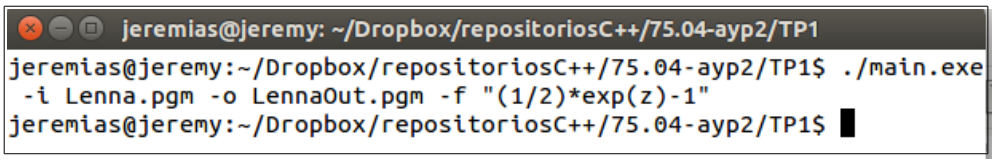
Ejemplo Lenna:

Dada la siguiente imagen original Lenna.pgm mostraremos la imagen procesada con distintas funciones arbitrarias.

Lenna.pgm

Funcion arbitraria: f(z)=-jz

Funcion arbitraria: f(z)=(1/2)*exp(z)-1



LennaOut.pgm

Función arbitraria polinómica: f(z) = -1*z+3*z^2-z^3

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i Lenna.pgm -o LennaOut.pgm -f "-1*z+3*z^2-z^3"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

LennaOut.pgm

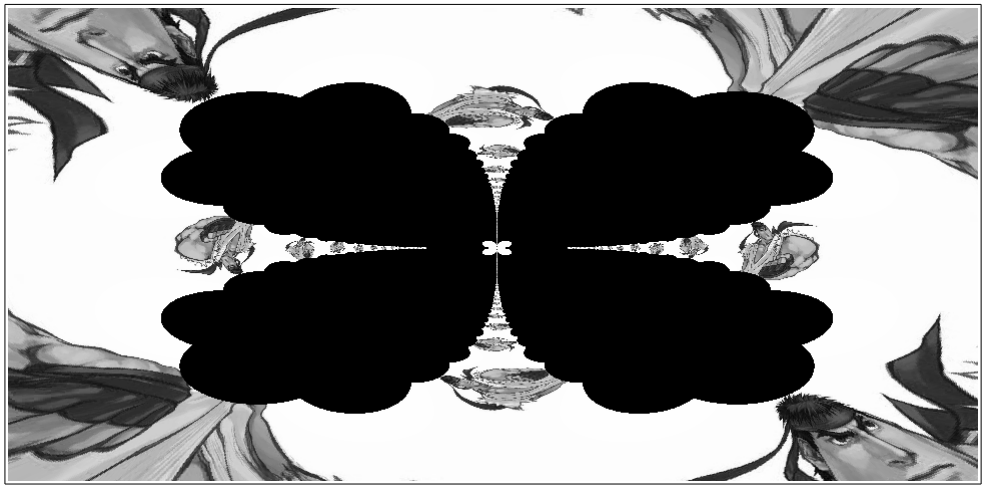
Ejemplo Ryu:

Imagen original Ryu.pgm:

Ryu.pgm

Función arbitrária: f(z)=sin(1/z^2)*cos(z^(1/2))

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
./main.exe
-i imagenesOrigen/Ryu.pgm -o imagenesDestino/RyuDestino.pgm
-f "sin(1/z^2)*cos(z^(1/2))"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

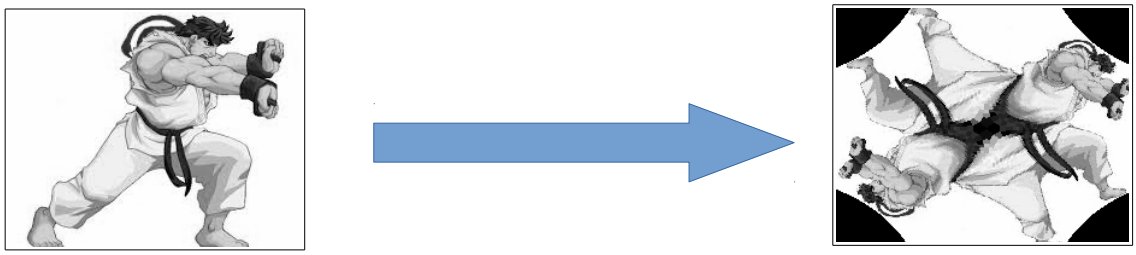


funcion arbitraria: f(z)=sinh(z).

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main
-i imagenesOrigen/Ryu.pgm -o imagenesDestino/RyuDestino.pgm
f "sinh(z)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

Con el siguiente ejemplo mostramos el correcto funcionamiento con la identidad trigonométrica de Pitágoras. Es decir, es equivalente a solo aplicar a la imagen la función f(z)=z^2.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe -i imagenes
Origen/Ryu2.pgm -o imagenesDestino/Ryu2Destino.pgm -f "(cos(z)^2+sin(z)^2)*z^2"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```



Ryu.pgm RyuDestino.pgm

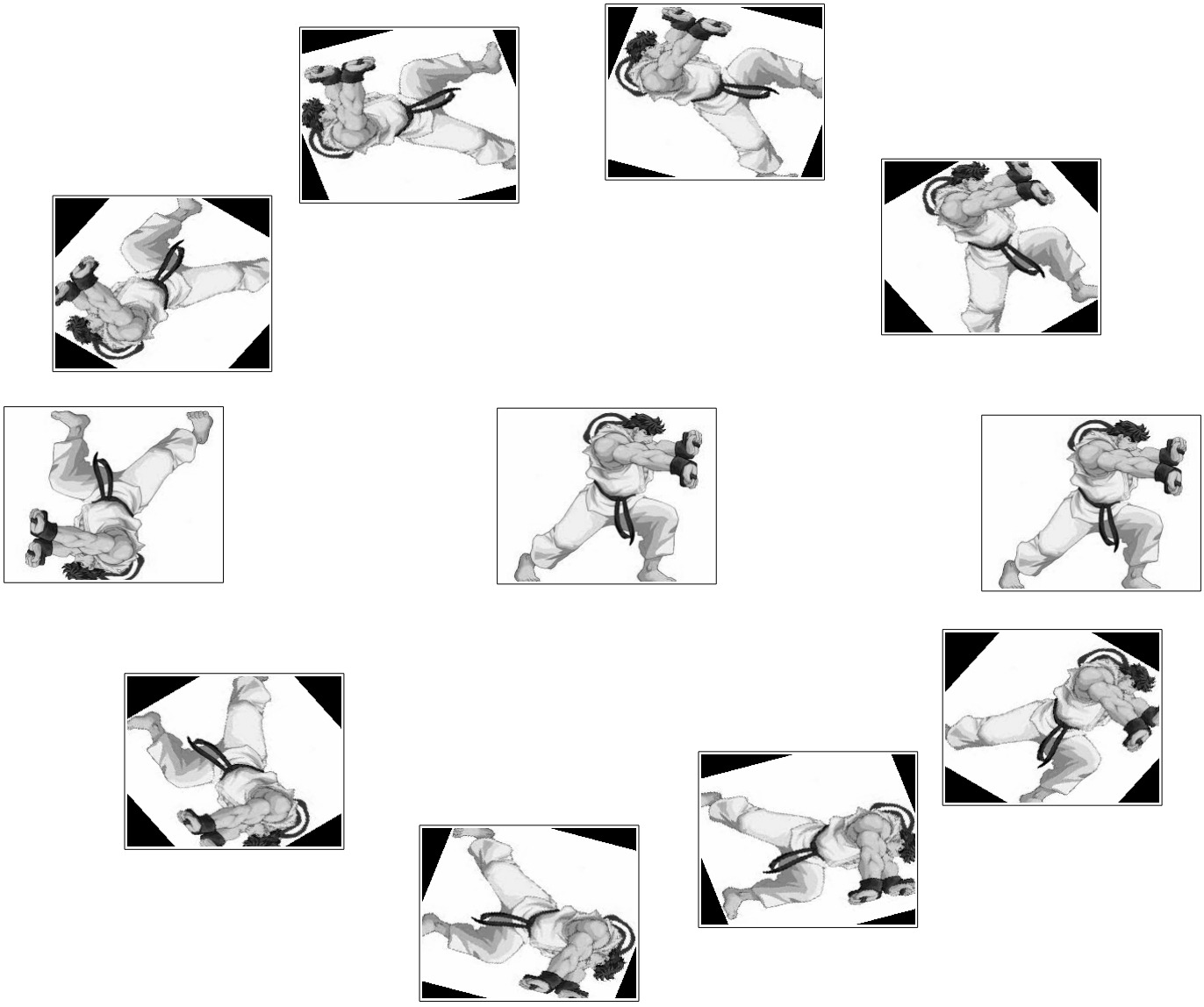
Y así se puede verificar el correcto funcionamiento de las funciones complejas a través de las distintas identidades trigonométricas.

Otro uso práctico para rotar imágenes en distintos ángulos.

Función arbitraria: f(z)= z*exp(j*(2*pi)*n/10) , n=0,1,2, … ,9.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino0.pgm
 -f "z*exp(j*(2*3.1415)*0/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino1.pgm
 -f "z*exp(j*(2*3.1415)*1/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino2.pgm
 -f "z*exp(j*(2*3.1415)*2/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino3.pgm
 -f "z*exp(j*(2*3.1415)*3/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino4.pgm
 -f "z*exp(j*(2*3.1415)*4/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino5.pgm
 -f "z*exp(j*(2*3.1415)*5/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino6.pgm
 -f "z*exp(j*(2*3.1415)*6/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino7.pgm
 -f "z*exp(j*(2*3.1415)*7/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino8.pgm
 -f "z*exp(j*(2*3.1415)*8/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
 -i imagenesOrigen/Ryu2.pgm -o imagenesDestino/Ryu2Destino9.pgm
 -f "z*exp(j*(2*3.1415)*9/10)"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

Rotación de la imágen Ryu2.pgm con 10 muestras en ángulos distintos.



Ejemplo greenranger:

Función arbitraria (escalamiento): f(z)=k*z

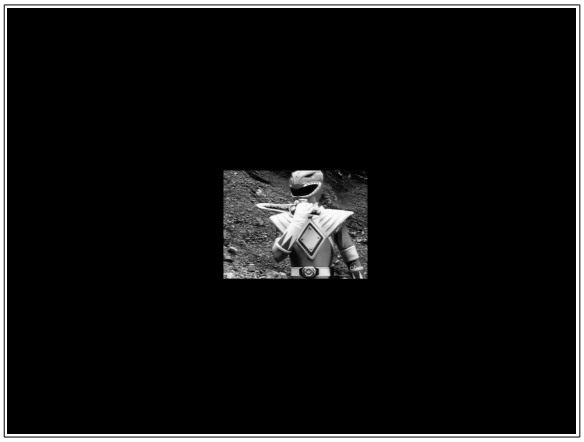


imagen con escala 1:4.

Función arbitraria (desplazamiento): f(z)=z-0.4+j*0.2 (aplicada sobre la imagen escalada).

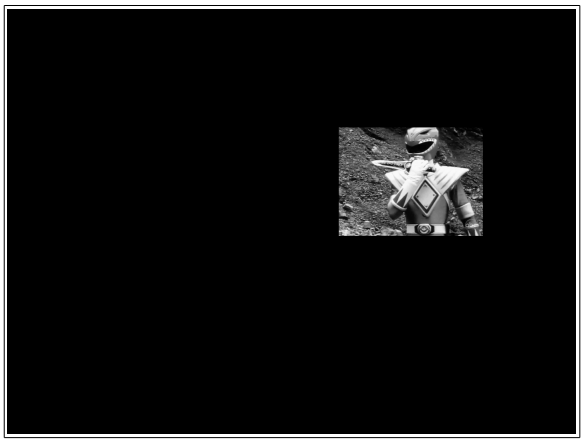


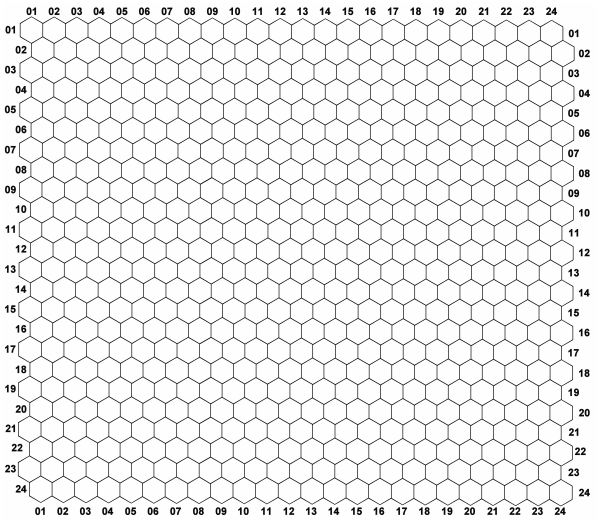
Imagen escalada y desplazada.

Ejemplo Grid:

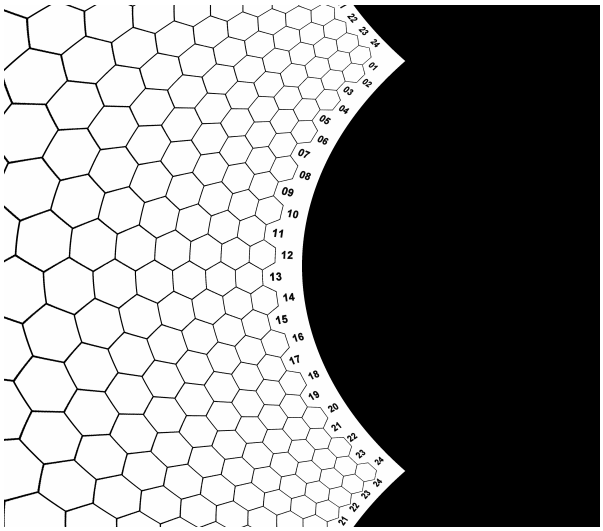
La imágen grid.pgm (47Mbytes) es de grandes dimensiónes a comparación de las imágenes anteriores (ej: Ryu.pgm es de 179Kbytes), por eso es apropiada para mostrar la velocidad de proceso del programa.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ time ./main
.exe -i imagenesOrigen/grid.pgm -o imagenesDestino/gridDestino.pgm
-f "exp(z)"

real    1m3.639s
user    1m2.863s
sys     0m0.765s
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```



grid.pgm



gridOut.pgm

Cuando la función arbitraria es una constante, evidentemente la salida resulta ser una imágen uniforme.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
-i imagenesOrigen/grid.pgm -o imagenesDestino/gridDestino.pgm
-f "200"
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

GridDestino.pgm

Entradas incorrectas.

-El nivel máximo de intensidad de un pixel excede el nivel máximo establecido.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imag.pgm
P2
# Commets test 1
# Commets test 2
24 7
15
0 0 0 79 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i imag.pgm -o imagOut.pgm -f "exp(z)"
Fuction enable: exp(z)
Invalid value 79 in position (0;3)
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

- Caracter inválido como dimensión de la imagen.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imag.pgm
P2
# Commets test 1
# Commets test 2
24 A
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i imag.pgm -o imagOut.pgm -f "exp(z)"
Fuction enable: exp(z)
Error: invalid size value
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

- Magic number P2 inválido.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ cat imag.pgm
P9
# Commets test 1
# Commets test 2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0 15  0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11  0 0 0 15 15 15 15  0
0 3 0 0 0 0 0 7 0 0 0 0 0 11  0  0  0 0 0 15  0  0  0  0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15  0  0  0  0
0 0 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0 0 0  0  0  0  0  0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i imag.pgm -o imagOut.pgm -f "exp(z)"
Fuction enable: exp(z)
Error: invalid P9 Magic Number
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

-Falta de argumento para la imagen de entrada.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i
Option requires argument: -i
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

-Intentar abrir un archivo inexistente.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP0
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$ ./main.exe -i File.pgm
cannot open File.pgm.
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP0$
```

-Errores en la sintaxis de la función matemática arbitraria.

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$ ./main.exe
-i imagenesOrigen/grid.pgm -o imagenesDestino/gridDestino.pgm
-f "sen(z)"
invalid token
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

La función “seno” está definida como sin(). Lo mismo sucede cuando se intenta utilizar una función que no esta definida, por ejemplo el logaritmo. (Si bien es fácil agregar funciones a este programa, la función logaritmo no esta definida en este trabajo).

```
jeremias@jeremy: ~/Dropbox/repositoriosC++/75.04-ayp2/TP1
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
./main.exe
-i imagenesOrigen/Ryu.pgm -o imagenesDestino/RyuDestino.pgm
-f "sin(z"
invalid token
jeremias@jeremy:~/Dropbox/repositoriosC++/75.04-ayp2/TP1$
```

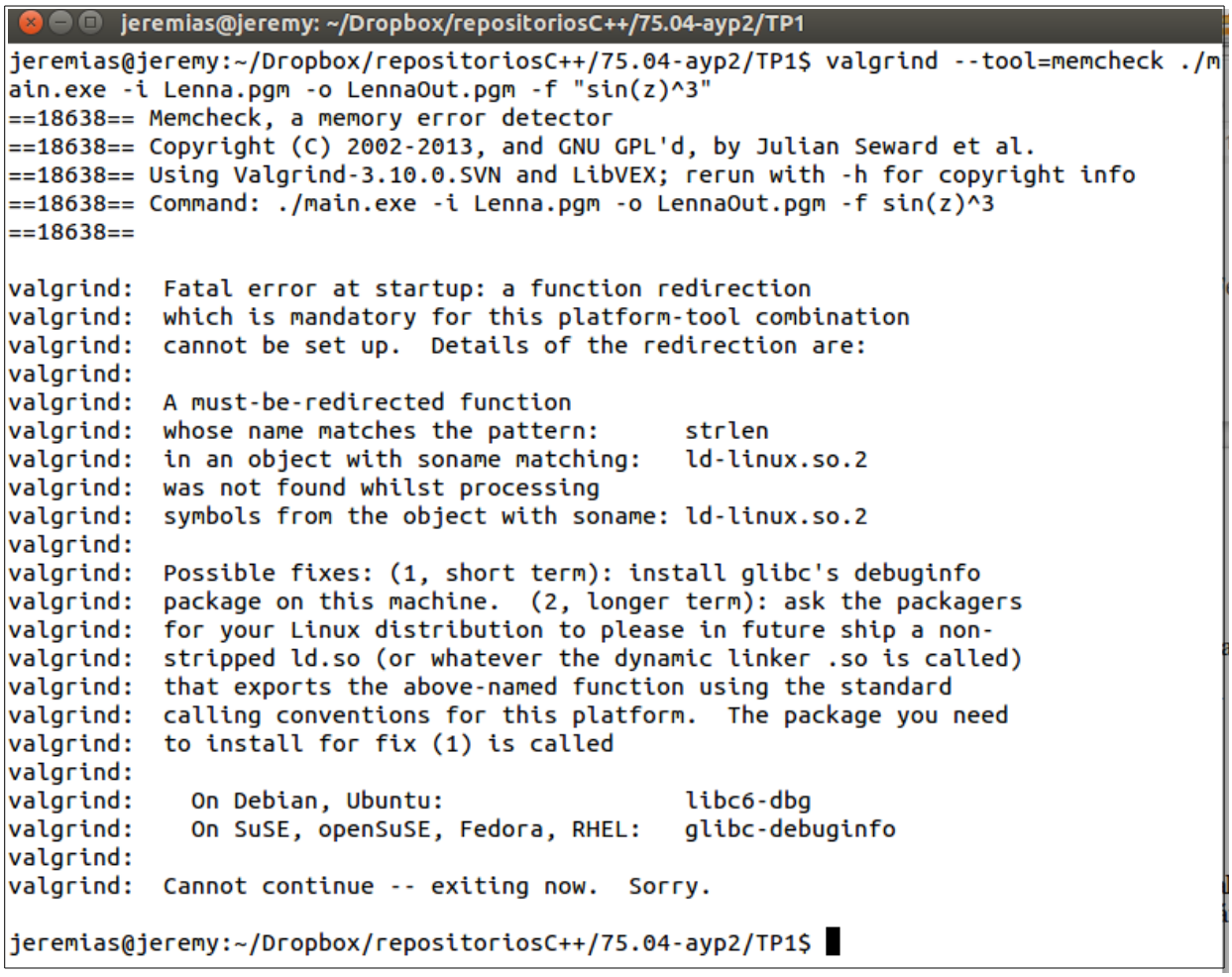
Olvidarse un parentesis es un error de sintaxis.

3.2 Comprobación de uso de memoria con Valgrind:

Utilizando la ejecución del programa Valgrind bajo el siguiente formato:

valgrind --tool = memcheck ./ main.exe – i < entrada.txt > –o < salida.txt > -f "<función>"

Obtuvimos resultaos distintos en la misma versión de software pero en distintos ordenadores:



```
federicoverstraeten@federicoverstraeten-MS-1458: ~/Escritorio/Repositorio C++/75.04-ayp2/TP1
04-ayp2/TP1$ valgrind --tool=memcheck ./main.exe -i Lenna.pgm -o out.pgm -f "sin
(z)^3"
==3882== Memcheck, a memory error detector
==3882== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==3882== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==3882== Command: ./main.exe -i Lenna.pgm -o out.pgm -f sin(z)^3
==3882==
==3882==
==3882== HEAP SUMMARY:
==3882==     in use at exit: 26 bytes in 2 blocks
==3882==   total heap usage: 10,486,971 allocs, 10,486,969 frees, 186,935,405 by
tes allocated
==3882==
==3882== LEAK SUMMARY:
==3882==    definitely lost: 12 bytes in 1 blocks
==3882==    indirectly lost: 0 bytes in 0 blocks
==3882==      possibly lost: 14 bytes in 1 blocks
==3882==    still reachable: 0 bytes in 0 blocks
==3882==         suppressed: 0 bytes in 0 blocks
==3882== Rerun with --leak-check=full to see details of leaked memory
==3882==
==3882== For counts of detected and suppressed errors, rerun with: -v
federicoverstraeten@federicoverstraeten-MS-1458:~/Escritorio/Repositorio C++/75.04-ayp2/TP1$
```

Como se puede observar, si bien el programa confeccionado funciona correctamente, el programa Valgrind no finaliza su operación exitosamente, desconocemos el origen de este error, distinto en ambos ordenadores, pero será presentado a los docentes del curso pora obtener ayuda y mas información al respecto.

```
/*****************************************************************
Jueves 23 de Octubre de 2014.
TP1 - Algoritmos II - Cátedra Calvo

Docentes TP:
      - Lucio Santi,
      - Leandro Santi.

Autores:
      - Federico Verstraeten <federico.verstraeten@gmail.com>
      - Jeremías Ignacio Zec <jeremiaszec@gmail.com>

Título: Programación C++ - Procesador de imágenes PGM

NOTA:

*****************************************************************/


/*****************************************
*       BIBLIOTECAS         *
*****************************************/

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstring>

#include "common.hpp"
#include "dictionary.hpp"
#include "cmdline.h"
#include "options.hpp"
#include "process.hpp"
#include "printers.hpp"
#include "complejo.hpp"
#include "planoC.hpp"
#include "expression.hpp"
#include "shuntingYard.hpp"
#include "operation.hpp"

/*****************************************
*     DECLARACIONES GLOBALES      *
*****************************************/

istream *iss = 0;
ostream *oss = 0;
fstream ifs, ofs;
string typeFunction;
size_t  altoImag=0, anchoImag=0;
binTree<string> opTree;

double deltaX=0;
double deltaY=0;
double initX=0;
double initY=0;

extern option_t options[];
//extern string function_dictionary[];

/*****************************************
*         TABLA DE OPERACIONES        *
*****************************************/
t_operation ops[]={

        //Operaciones
        {"^", 9, ASSOC_RIGHT,NOT_UNARY,OPERATOR, eval_pow},
        {"*", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, eval_mul},
        {"/", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, eval_div},
        {"+", 5, ASSOC_LEFT, BINARY_UNARY,OPERATOR, eval_add},
        {"-", 5, ASSOC_LEFT, BINARY_UNARY,OPERATOR, eval_sub},
        {"+_",10, ASSOC_LEFT, UNARY,OPERATOR, eval_unplus}, //suma unaria
```

```
        {"-_",10, ASSOC_LEFT, UNARY,OPERATOR, eval_unminus}, //negación unaria
        {",", 0, ASSOC_NONE, NOT_UNARY,SEPARATOR, NULL},
        {"(", 0, ASSOC_NONE, NOT_UNARY,PARENTESIS_OPEN, NULL},
        {")", 0, ASSOC_NONE, NOT_UNARY,PARENTESIS_CLOSE, NULL},
        //{"%", 8, ASSOC_LEFT, NOT_UNARY,OPERATOR, NULL},

        //variable independiente
        {"z",10, ASSOC_RIGHT,NOT_UNARY,VAR_INDEP,NULL},
        //unidad imaginaria
        {"j",10, ASSOC_NONE, NOT_UNARY,IMAGINARY_UNIT,NULL},

        //Funciones
        {"re",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_re},
        {"im",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_im},
        {"exp",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_exp},
        {"id",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_id},
        {"abs",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_abs},
        {"sinh",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_sinh},
        {"cosh",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_cosh},
        {"sin",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_sin},
        {"cos",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_cos},
        {"arg",10,ASSOC_RIGHT,NOT_UNARY,FUNCTION,eval_arg},
        {0, },

};


/*****************************************
*         MAIN          *
*****************************************/

int main(int argc,char *argv[])
{
    //VALIDACION DE OPCIONES Y ARGUMENTOS
  cmdline cmdl(options);
  cmdl.parse(argc, argv);

    //DECLARACIONES
    size_t **matrixIn = NULL;
    size_t **matrixOut = NULL;
  size_t maxInten;
  string str, sMagicNum;
  complejo z;     // z variable independiente
  complejo w;     // w=f(z)
  vector<string> parser;

  //Se parsea la funcion ingresada con -f
    processBuffer(typeFunction.c_str(),parser);

    //Se transforma de notación infija a RPN
    if(shuntingYard(parser)==ERROR_INVALID_TOKEN)
    {
        cerr<<"invalid token"<<endl;
        return EXIT_PROGRAM;
    }

    // Se construye el árbol de operaciones
    // desde el parser en RPN. Variable global.
  opTree=constructionOpTree(parser);

    // Se extrae caracteres desde iss y los coloca en str,
  // hasta encontrar '/n'.
  readLine(*iss,str);

    // Transformo el string en un stream.
    // Me permite usar '>>' en lectura.
    istringstream issMagicNum(str);

    // MagicNumber
    if(getMagicNumber(issMagicNum,sMagicNum)==ERROR)
        return EXIT_PROGRAM;
```

```
    // Lectura tamaños de matrices
    if(readSize(*iss)==ERROR) return EXIT_PROGRAM;

  deltaX = 2.0/anchoImag;
  deltaY = 2.0/altoImag;
  initX = deltaX/2.0-1;
  initY = deltaY/2.0-1;

    //Lectura máximo de intensidad
    if(readMaxIntensity(*iss,maxInten)==ERROR) return EXIT_PROGRAM;

    //Creo las matrices. Otorgo memoria.
    createMatrix(matrixIn,altoImag,anchoImag);
    createMatrix(matrixOut,altoImag,anchoImag);

    //Lectura de la matriz de entrada
    if(readMatrixIN(*iss,matrixIn,maxInten)==ERROR)
    {
        // Si no se puede procesar la matriz de entrada
        // liberamos la memoria y cortamos el programa
        deleteMatrix(matrixIn,altoImag,anchoImag);
        deleteMatrix(matrixOut,altoImag,anchoImag);
        return EXIT_PROGRAM;
    }

  // Recorro matriz destino y se copia los tonos de la matriz de origen,
  // según la función w=f(z)
  matrixTransformation(matrixIn,matrixOut);

    //Impresión de imagen por salida
    printImage(*oss,matrixOut,altoImag,anchoImag,maxInten);

    // Libero la memoria utilizada
    deleteMatrix(matrixIn,altoImag,anchoImag);
    deleteMatrix(matrixOut,altoImag,anchoImag);

    return EXIT_SUCCESS;
}
```

```
#ifndef BINTREE_HPP
#define BINTREE_HPP

#include <stdlib.h>
#include <iostream>
#include "common.hpp"


/************************************************
 *                 CLASS TEMPLATE                *
 ************************************************/

template <typename T>

class binTree
{
    private:
        T data;
        binTree *left_;
        binTree *right_;

    public:
        /*************************** CONSTRUCTORES ***************************/
        binTree();
        binTree(const T&);
        binTree(const binTree&);
        /*************************** DESTRUCTORES ****************************/
        ~binTree();
        /****************************** METODOS ******************************/
        void setData(const T&);
    T getData()const;
    binTree<T>& getLeftSon()const;
    binTree<T>& getRightSon()const;
    status_t insert(binTree <T> &);
        //bool explore(const T);
        //binTree<T> search(const T);
        void print (ostream&);
        void printTree(ostream&);
    bool hasLeftSon()const;
    bool hasRightSon()const;
        /***************************** OPERADORES ****************************/
        /************************** FUNCIONES FRIENDS ************************/
        /************************** OPERADORES FRIENDS ***********************/
};

/*****************************************************************************/
/****************************** CONSTRUCTORES ********************************/
/*****************************************************************************/
template <typename T>
binTree<T>::binTree()
{
    left_=NULL;
    right_=NULL;
}
template <typename T>
binTree<T>::binTree(const T &value)
{
    data=value;
    left_=NULL;
    right_=NULL;
}
template <typename T>
binTree<T>::binTree(const binTree &tree)
{
    left_=NULL;
    right_=NULL;

    // Copio la información de un árbol a otro
    data=tree.data;

    if(tree.left_)
    {
```

```
        // Solicito memoria y copio la estructura.
        left_=new binTree (*(tree.left_));
    }
    if(tree.right_)
    {
        right_=new binTree (*(tree.right_));
    }


}
template <typename T>
binTree <T> :: ~binTree ()
{
    delete left_;
    delete right_;
}

template <typename T>
void binTree <T> :: setData(const T& inData)
{
    data=inData;
}

template <typename T>
T binTree <T> :: getData()const
{
    return data;
}


template <typename T>
status_t binTree <T> :: insert(binTree <T> & subTree)
{
    if(left_==NULL)
    {
        left_=&subTree;
        return  OK;
    }
    else if(right_==NULL)
    {
        right_=&subTree;
        return OK;
    }
    return ERROR_INSERT_TREE_NODE;
}

template <typename T>
void binTree <T> :: print(ostream & os)
{
    os<<data<<endl;
}

template <typename T>
void binTree <T> :: printTree(ostream & os)
{
    os<<data<<endl;
    if(left_!=NULL)
        left_->printTree(os);
    if(right_!=NULL)
        right_->printTree(os);
}

template <typename T>
bool binTree <T> :: hasLeftSon()const
{
    if(left_!=NULL) return true;
  else return false;
}

template <typename T>
bool binTree <T> :: hasRightSon()const
{
```

```
    if(right_!=NULL) return true;
  else return false;
}

template <typename T>
binTree<T>& binTree <T> :: getLeftSon()const
{
    return (*left_);
}

template <typename T>
binTree<T>& binTree <T> :: getRightSon()const
{
    return (*right_);
}

#endif
```

```
#ifndef DICTIONARY_HPP_INCLUDED
#define DICTIONARY_HPP_INCLUDED

#include<string>
#include"common.hpp"

#define MAX_NUMFUCTION 4


#endif // DICTIONARY_HPP_INCLUDED
```

```
#include"dictionary.hpp"
#include"cmdline.h"
#include"options.hpp"

string function_dictionary[] = {
                              "z", "Z",
                              "exp(z)","EXP(Z)",
                            };

option_t options[] = {
  {1, "i", "input" , "-", opt_input, OPT_DEFAULT},
  {1, "o", "output", "-", opt_output, OPT_DEFAULT},
      {0, "h", "help"  , NULL, opt_help, OPT_DEFAULT},
  {1, "f", "function","z",opt_function,OPT_DEFAULT},// f(z)=z valor por defecto
  {0, },
};

 // No funciona correctamente si pongo NULL como opcion corta, lo mismo sucede con la opcion larga.
 // se invalidan las opciones cortas (o largas) siguientes en la tabla, eso sucede porque las funciones
 // do_short_opt y do_long_opt, usan como parametro de iteracion recorrer la tabla de opciones
 // hasta encontrar un valor 0 (cero) o NULL. Por esto mismo no es valido poner NULL como opcion corta
 // (o larga) en el caso de que no se considere necesario; es obligatorio poner ambar opciones, o como
 // alternativa dejar un caracter vacio " ".
```

```
#ifndef OPTIONS_HPP_INCLUDED
#define OPTIONS_HPP_INCLUDED
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <sstream>
#include "dictionary.hpp"

using namespace std;


void opt_input(string const &);
void opt_output(string const &);
void opt_help(string const &);
void opt_function(string const &);

#endif // OPTIONS_HPP_INCLUDED
```

```
#include "options.hpp"

using namespace std;

extern istream *iss;
extern ostream *oss;
extern fstream ifs;
extern fstream ofs;
extern string typeFunction;

void opt_input(string const &arg)
{

  if (arg == "-") {
    iss = &cin;
  } else {
    ifs.open(arg.c_str(), ios::in);
    iss = &ifs;
  }

  // Verificamos que el stream este OK.
  //
  if (!iss->good()) {
    cerr << "cannot open "
         << arg
         << "."
         << endl;
    exit(EXIT_FAILURE);
  }
}

void opt_output(string const &arg)
{
  if (arg == "-") {
    oss = &cout;
  } else {
    ofs.open(arg.c_str(), ios::out);
    oss = &ofs;
  }

  if (!oss->good()) {
    cerr << "cannot open "
         << arg
         << "."
         << endl;
    exit(EXIT_FAILURE);
  }
}

void opt_help(string const &arg)
{
  cout << "cmdline [-i stream_in] [-o stream_out] [-f function]"
         << endl;
  exit(EXIT_SUCCESS);
}

void opt_function(string const &arg)
{
    // Leer linea de comandos y guardar globalmente
    // la función se valida en el algoritmo shunting yard
  typeFunction=arg;
}
```

```
#ifndef PROCESS_HPP_INCLUDED
#define PROCESS_HPP_INCLUDED

#include <iostream>
#include <string>
#include <sstream>
#include "common.hpp"
#include "dictionary.hpp"
#include "planoC.hpp"
#include "binTree.hpp"
#include "operation.hpp"

bool FunctionType(string);
status_t getMagicNumber(istream& ,string& );
bool createMatrix(size_t** &matrix,size_t ,size_t );
void deleteMatrix(size_t** &matrix,size_t ,size_t );
void readLine(istream& , string& );
status_t readSize(istream&);
status_t readMaxIntensity(istream&,size_t&);
status_t readMatrixIN(istream& ,size_t** &matrix,const size_t& );
void matrixTransformation(size_t** &matrixIn,size_t** &matrixOut);

#endif // PROCESS_HPP_INCLUDED
```

```
#include "process.hpp"

/** Variables y estructuras externas **/
extern size_t  altoImag, anchoImag;
extern binTree<string> opTree;

/*********************************************************************/

status_t getMagicNumber(istream &is,string &str)
{
    is >> str;
    if(str==MAGICNUM) return OK;
    else
    {
        cerr << "Error: invalid "
             << str
             << " Magic Number" << endl;

        return ERROR;
    }
}

bool createMatrix(size_t** &matrix,size_t row,size_t col)
{
    if(row>0 && col>0)
    {
        matrix=new size_t* [row];
        for(size_t i=0 ; i<row ; ++i)
        {
            matrix[i]=new size_t [col];
        }
        return true;
    }
    return false;
}

void deleteMatrix(size_t** &matrix,size_t row,size_t col)
{
    for(size_t i = 0 ; i < row ; i++)
        delete[] matrix[i];

    delete[] matrix;
}

void readLine(istream &is, string &str)
{
  getline(is,str);
  while(str.find('#')==0) getline(is,str);
}

status_t readSize(istream &is)
{
    string str;
    bool good;

    readLine(is,str);
    istringstream issSize(str);

    if(issSize>>anchoImag && issSize>>altoImag
          && anchoImag!=0 && altoImag!=0) good=true;

    else good=false;

    if(good==false)
    {
        issSize.clear(ios::badbit);
        cerr<<"Error: invalid size value "<<endl;
        return ERROR;
    }

    return OK;
}
```

```
status_t readMaxIntensity(istream &is,size_t &maxInten)
{
    string str;
    bool good;

    readLine(is,str);
    istringstream issMax(str);

    if(issMax>>maxInten && maxInten<=255)good=true;
    else good=false;

    if(good==false)
    {
        issMax.clear(ios::badbit);
        cerr<<"Error: invalid max intensity value "<<endl;
        return ERROR;
    }

    return OK;
}

status_t readMatrixIN(istream& is,size_t** &matrix,const size_t &maxInten)
{
    size_t i=0; //iterador de "Alto de imagen"
    size_t j=0; //iterador de "Ancho de imagen"
    size_t aux;

    while(is>>aux && aux<=maxInten)
    {
         matrix[i][j]=aux;
         j++;

        if(j>=anchoImag){ j=0; i++;}
    }
    if((is.fail() && !is.eof()) || aux>maxInten)
    {
        cerr<<"Invalid value "
            <<"in position "
            <<"("<<i<<";"<<j<<")"
            <<endl;
            return ERROR;
     }

    return OK;
}

void matrixTransformation(size_t** &matrixIn,size_t** &matrixOut)
{
    complejo z;
  complejo w; // w=f(z)

    for(size_t fil=0 ; fil < altoImag ; fil++)
  {
    for(size_t col=0 ; col < anchoImag ; col++)
    {
      // Recibo las coordenadas de la matriz destino,
      // z es el correspondiente valor del plano complejo 2x2.
      z = matrizAplanoC(col,fil);

            // Procesar árbol de operaciones para cada pto.
            // de la matriz y se obtiene w=f(z)
      w=evaluateOpTree(opTree,z);

      if(w.re()<-1 || w.re()>1 || w.im()<-1 || w.im()>1)
        matrixOut[fil][col]=0;
      else
      {
        w = planoCaMatriz(w);
        matrixOut[fil][col]=matrixIn[(int)w.im()][(int)w.re()];
      }
    }
```

```
  }
}
```

```
#ifndef _CMDLINE_H_INCLUDED_
#define _CMDLINE_H_INCLUDED_

#include <string>
#include <iostream>

#define OPT_DEFAULT   0
#define OPT_SEEN      1
#define OPT_MANDATORY 2

struct option_t {
  int has_arg;
  const char *short_name;
  const char *long_name;
  const char *def_value;
  void (*parse)(std::string const &);
  int flags;
};

class cmdline {
  // Este atributo apunta a la tabla que describe todas
  // las opciones a procesar. Por el momento, sólo puede
  // ser modificado mediante contructor, y debe finalizar
  // con un elemento nulo.
  //
  option_t *option_table;

  // El constructor por defecto cmdline::cmdline(), es
  // privado, para evitar construir parsers sin opciones.
  //
  cmdline();
  int do_long_opt(const char *, const char *);
  int do_short_opt(const char *, const char *);

    public:
  cmdline(option_t *);
  void parse(int, char * const []);
};

#endif
```

```
// cmdline - procesamiento de opciones en la línea de comando.
//
// $Date: 2012/09/14 13:08:33 $
//
#include <string>
#include <cstdlib>
#include <iostream>
#include "cmdline.h"

using namespace std;

cmdline::cmdline()
{
}

cmdline::cmdline(option_t *table) : option_table(table)
{
}

void
cmdline::parse(int argc, char * const argv[])
{
#define END_OF_OPTIONS(p)       \
  ((p)->short_name == 0   \
   && (p)->long_name == 0 \
   && (p)->parse == 0)

  // Primer pasada por la secuencia de opciones: marcamos
  // todas las opciones, como no procesadas. Ver código de
  // abajo.
  //
  for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
    op->flags &= ~OPT_SEEN;

  // Recorremos el arreglo argv. En cada paso, vemos
  // si se trata de una opción corta, o larga. Luego,
  // llamamos a la función de parseo correspondiente.
  //
  for (int i = 1; i < argc; ++i) {
    // Todos los parámetros de este programa deben
    // pasarse en forma de opciones. Encontrar un
    // parámetro no-opción es un error.
    //
    if (argv[i][0] != '-') {
      cerr << "Invalid non-option argument: "
           << argv[i]
           << endl;
      exit(1);
    }

    // Usamos "--" para marcar el fin de las
    // opciones; todo los argumentos que puedan
    // estar a continuación no son interpretados
    // como opciones.
    //
    if (argv[i][1] == '-'
        && argv[i][2] == 0)
      break;

    // Finalmente, vemos si se trata o no de una
    // opción larga; y llamamos al método que se
    // encarga de cada caso.
    //
    if (argv[i][1] == '-')
      i += do_long_opt(&argv[i][2], argv[i + 1]);
    else
      i += do_short_opt(&argv[i][1], argv[i + 1]);
  }

  // Segunda pasada: procesamos aquellas opciones que,
  // (1) no hayan figurado explícitamente en la línea
  // de comandos, y (2) tengan valor por defecto.
```

```
  //
  for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
#define OPTION_NAME(op) \
  (op->short_name ? op->short_name : op->long_name)
    if (op->flags & OPT_SEEN)
      continue;
    if (op->flags & OPT_MANDATORY) {
      cerr << "Option "
           << "-"
           << OPTION_NAME(op)
           << " is mandatory."
           << "\n";
      exit(1);
    }
    if (op->def_value == 0)
      continue;
    op->parse(string(op->def_value));
  }
}

int
cmdline::do_long_opt(const char *opt, const char *arg)
{
  // Recorremos la tabla de opciones, y buscamos la
  // entrada larga que se corresponda con la opción de
  // línea de comandos. De no encontrarse, indicamos el
  // error.
  //

  for (option_t *op = option_table; op->long_name != 0; ++op) {
    if (string(opt) == string(op->long_name)) {
      // Marcamos esta opción como usada en
      // forma explícita, para evitar tener
      // que inicializarla con el valor por
      // defecto.
      //
      op->flags |= OPT_SEEN;

      if (op->has_arg) {
        // Como se trada de una opción
        // con argumento, verificamos que
        // el mismo haya sido provisto.
        //
        if (arg == 0) {
          cerr << "Option requires argument: "
               << "--"
               << opt
               << "\n";
          exit(1);
        }
        op->parse(string(arg));
        return 1;
      } else {
        // Opción sin argumento.
        //
        op->parse(string(""));
        return 0;
      }
    }
  }

  // Error: opción no reconocida. Imprimimos un mensaje
  // de error, y finalizamos la ejecución del programa.
  //
  cerr << "Unknown option: "
       << "--"
       << opt
       << "."
       << endl;
  exit(1);

  // Algunos compiladores se quejan con funciones que
```

```
  // lógicamente no pueden terminar, y que no devuelven
  // un valor en esta última parte.
  //
  return -1;
}

int
cmdline::do_short_opt(const char *opt, const char *arg)
{
  option_t *op;

  // Recorremos la tabla de opciones, y buscamos la
  // entrada corta que se corresponda con la opción de
  // línea de comandos. De no encontrarse, indicamos el
  // error.
  //
    for (op = option_table; op->short_name != 0; ++op)
  {
    if (string(opt) == string(op->short_name)) {
      // Marcamos esta opción como usada en
      // forma explícita, para evitar tener
      // que inicializarla con el valor por
      // defecto.
      //
      op->flags |= OPT_SEEN;

      if (op->has_arg) {
        // Como se trata de una opción
        // con argumento, verificamos que
        // el mismo haya sido provisto.
        //
        if (arg == 0) {
          cerr << "Option requires argument: "
               << "-"
               << opt
               << "\n";
          exit(1);
        }
        op->parse(string(arg));
        return 1;
      } else {
        // Opción sin argumento.
        //
        op->parse(string(""));
        return 0;
      }
    }
  }

  // Error: opción no reconocida. Imprimimos un mensaje
  // de error, y finalizamos la ejecución del programa.
  //
  cerr << "Unknown option: "
       << "-"
       << opt
       << "."
       << endl;
  exit(1);

  // Algunos compiladores se quejan con funciones que
  // lógicamente no pueden terminar, y que no devuelven
  // un valor en esta última parte.
  //
  return -1;
}
```

```
#ifndef EXPRESSION_HPP_INCLUDED
#define EXPRESSION_HPP_INCLUDED

#include <iostream>
#include <string.h>
#include <vector>
#include <ctype.h>

#define LBUFF 50           // longitud del buffer de salida

using namespace std;

enum caracter {
               NONE, LETTER, NUMBER,MATH,UNKNOWN,SPACE
              };

// función auxiliar limpiar el buffer de salida
void limpiar(char buff[],const int longBuff,size_t &pbuff,caracter &chant);

// función auxiliar mostrar el buffer de salida
void mostrar(const vector<string> &);

// guardar en un string la salida luego del Parser
void guardar(vector<string> &,char buff[],const size_t &);

// Función que procesa el buffer de entrada
void processBuffer(const char InBuff[],vector<string> &);

#endif
```

```
#include "expression.hpp"

// Global enum
//caracter chact;     // Caracter actual
//caracter chant;     // Caracter anaterior


void limpiar(char buff[],const int longBuff,size_t &pbuff,caracter &chant)
{
  for (int i=0; i<longBuff; ++i) buff[i] = '\0';
  pbuff =0;
  chant = NONE;
}

void mostrar(const vector<string> &parser)
{
  if (!parser.empty())
  {
      for(size_t i=0; i<parser.size(); i++)
        cout<<parser[i]<<endl;
  }
}

void guardar(vector<string> &parser,char buff[],const size_t &pbuff)
{
    char c;
    size_t i=0;

    if(pbuff!=0)
    {
       //transformo mayúsculas a minúsculas
        while (buff[i])
        {
            c=buff[i];
            buff[i]=tolower(c);
            i++;
        }

       //guardo el token en el vector
       parser.push_back(string(buff));
    }
}

void processBuffer(const char InBuff[],vector<string> &parser)
{
    size_t pos=0,pbuff=0;   // posición inicial
    char c;
    char OutBuff[LBUFF];
    caracter chact;     // Caracter actual
    caracter chant;     // Caracter anaterior

    limpiar(OutBuff,LBUFF,pbuff,chant);
    do
    {
        // Análisis del string de entrada.
        // Se forman tokens según sean números, letras u operaciones
        // Se utilizan las funciones isdigit y isalpha de la biblioteca
        // ctype.h que corroboran si son números o letras la entrada.
        c = InBuff[pos];

        // Si el caracter es numérico. Se considera el punto decimal para los float
        if (isdigit(c) || c=='.')   chact = NUMBER;

        // Si es una palabra
        else if (isalpha(c)) chact = LETTER;

        // Si es una operación matemática
        else if (c=='+' || c=='-' || c=='*' || c=='/' || c=='^') chact = MATH;

        // Si es un espacio en blanco, lo eliminamos
        else if (c==' ') chact = SPACE;
```

```
        // Cualquier otra condición de análisis,
        // por ejemplo los paréntisis u otro símbolo.
        else chact = UNKNOWN;

        //-----------------------------------------------------------
        // CAMBIO DE TIPO: guardo el token hasta donde se ha leído
        // Si es un espacio (SPACE) salteo el cambio de tipo
        if (chact != chant  || chact==UNKNOWN || chact==MATH )
        {
          guardar(parser,OutBuff,pbuff);
          limpiar(OutBuff,LBUFF,pbuff,chant);// limpia el buffer, inicializa pbuff
          chant = chact;
        }

        // actualizar buffer de salida
        if(chact != SPACE)
        {
            OutBuff[pbuff] = c;
            ++pbuff;
        }

        // incrementar iterador a InBuffer
         ++pos;

    } while (pos < strlen(InBuff)); // fin del análisis

    // Guardo el último resultado
    guardar(parser,OutBuff,pbuff);
}
```

```
#ifndef SHUNTINGYARD_HPP_INCLUDED
#define SHUNTINGYARD_HPP_INCLUDED

#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <stack>
#include <cctype>
#include <cstdlib>
#include "common.hpp"
#include "operation.hpp"


/*** Funciones ***/

status_t shuntingYard(vector<string> &);
t_typeop isOperation(const string &str);

#endif
```

```
#include "shuntingYard.hpp"

//extern vector<string> parserRPN;
extern t_operation ops[];

/*** Implementación de funciones ***/

/*****************************************************
isOperation: recibe un token a procesar, si es una
función,operación o paréntesis lo busca en la tabla de
operaciones y devuelve el tipo de operador que es.
*****************************************************/

t_typeop isOperation(const string &str)
{
    for(t_operation *op=ops ; !END_TABLEOP(op) ; ++op)
    {
        if(op->op == str)
        {
            if(op->func==FUNCTION) return FUNCTION;
            if(op->func==SEPARATOR) return SEPARATOR;
            if(op->func==PARENTESIS_OPEN) return PARENTESIS_OPEN;
            if(op->func==PARENTESIS_CLOSE) return PARENTESIS_CLOSE;

        }
    }
    return NONEOP;
}

/*****************************************************
shuntingYard: Función que aplica el algoritmo del mismo
nombre. Recibe un vector a analizar (parsing) con una
expresión matemática separada en tokens en notación
infija y la transforma en Notación Polaca Inversa (RPN).

*****************************************************/
status_t shuntingYard (vector<string> &parser)
{
    float num;
    stack<string> opStack;
    t_operation *op=NULL,*op_aux=NULL;
    vector<string> parserRPN; //Vector auxiliar

    // Flag que indica si el token anterior a uno
    // es un operador válido,y ver si este un
    // operador es unario.
    bool f_opAnt=true;

    // Se lee el vector hasta que haya tokens
    for(size_t i=0; i<parser.size() ; i++)
    {
        istringstream issparser(parser[i]);
        ostringstream ossparser;

        //Si el token es un número
        if(issparser>>num)
        {
            ossparser<<num;
            parserRPN.push_back(ossparser.str());
            f_opAnt=false;
        }

        else
        {
            //Valida el token con la tabla de operaciones
            op=getOp(parser[i]);
            if(op==NULL) return ERROR_INVALID_TOKEN;

            //Si es función, variable independ. o unidad imag. lo coloco en la pila
            if(op->func==FUNCTION || op->func==VAR_INDEP || op->func==IMAGINARY_UNIT)
            {
                opStack.push(parser[i]);
```

```
            f_opAnt=false;
        }
        // Si es un separador desapilar hasta encontrar "("
        else if(op->func==SEPARATOR)
        {
            f_opAnt=false;
            while(!opStack.empty() && opStack.top()!="(")//<--mejorar
            {
                parserRPN.push_back(opStack.top());
                opStack.pop();
            }
            if(opStack.empty())
                return ERROR_INVALID_TOKEN;
        }

        //Si el token es un operador matemático
        else if(op->func==OPERATOR)
        {
            //Si el operador es unario lo concatenamos
            //al siguiente token y luego lo procesamos
            if(f_opAnt && op->unary==BINARY_UNARY)
            {
               t_operation *unaryOp=getOp(parser[i]+"_");
               parser[i]=unaryOp->op;
            }

            if(!opStack.empty())
            {
                op_aux=getOp(opStack.top());

                //Evaluamos la precedencia y la asociatividad del operador
                //a insertar a la pila con los ya existentes, y desapilamos
                //según lo evaluado.
                while( opStack.top()!="(" && !opStack.empty() &&
                      ((op->assoc == ASSOC_LEFT && op->prec <= op_aux->prec) ||
                      (op->assoc == ASSOC_RIGHT && op->prec < op_aux->prec))
                      )
                  {
                    parserRPN.push_back(opStack.top());
                    opStack.pop();
                    if(opStack.empty()) break;
                    op_aux=getOp(opStack.top());
                  }

                //Insertamos el nuevo operador a la pila
                opStack.push(parser[i]);
                f_opAnt=false;
            }
            else
            {
                opStack.push(parser[i]);
                f_opAnt=false;
            }
        }

        //Si el token es un parentesis abierto "("
        else if(op->func==PARENTESIS_OPEN)
        {
            f_opAnt=true;
            opStack.push(parser[i]);
        }

        //Si el token es un parentesis cerrado ")"
        else if(op->func==PARENTESIS_CLOSE)
        {
            f_opAnt=false;
            //Se vacía el stack hasta encontrar la pareja
            while(!opStack.empty() && opStack.top()!="(")
            {
                parserRPN.push_back(opStack.top());
                opStack.pop();
            }
```

```
                //Si no se encuntra
                if(opStack.empty())
                    return ERROR_INVALID_TOKEN;

                //Si lo encuentro lo desapilo
                else if(opStack.top()=="(") opStack.pop();

                //Si en el tope hay una función lo pongo en la cola
                if(!opStack.empty())
                {
                    op_aux=getOp(opStack.top());

                    if(op_aux->func==FUNCTION)
                    {
                        parserRPN.push_back(opStack.top());
                        opStack.pop();
                    }
                }
            }

            //Si el token no es ninguno de los permitidos
            else if(op->func==NONEOP)
                return ERROR_INVALID_TOKEN;
        }
    }

    //Si no hay tokens que leer pero hay operadores
    //en la pila, entonces se desapila y se pone en la cola.
    while(!opStack.empty() && opStack.top()!="(")
    {
        parserRPN.push_back(opStack.top());
        opStack.pop();
    }
    if(!opStack.empty() && opStack.top()=="(")
        return ERROR_INVALID_TOKEN;

    // Se copia el vector en RPN al de entrada
    parser=parserRPN;

  return OK;
}
```

```
#ifndef PLANOC_HPP_INCLUDED
#define PLANOC_HPP_INCLUDED

#include "complejo.hpp"

complejo matrizAplanoC(int col, int fil);
complejo planoCaMatriz(const complejo &z);

#endif
```

```
#include "planoC.hpp"
extern double deltaX;
extern double deltaY;
extern double initX;
extern double initY;

/************************ matrizAplanoC ************************
 * matrizAplanoC: recive como argumento una coordenada de una matriz
 * retorna un complejo comprendido en el plano complejo 2x2
 * correspondiente a dichas coordenadas.
 ***************************************************************/
complejo matrizAplanoC(int col, int fil)
{
  return complejo(initX+deltaX*col,initY+deltaY*fil);
}

/************************** planoCaMatriz ************************
 * planoCaMatriz: modifica el complejo que recive por referencia
 * para que queden las coordenadas de la matriz correspondiente
 * En la parte real e imaginaria quedan asignadas las columnas y
 * las filas respectivamente.
 ***************************************************************/
complejo planoCaMatriz(const complejo &z)
{
  complejo a((z.re()-initX)/deltaX,(z.im()-initY)/deltaY);
  a.redondeo();
  return a;
}
```

```
#ifndef COMMON_HPP
#define COMMON_HPP

#define EXIT_PROGRAM   1
#define EXIT_SUCCESS   0
#define ERROR_PROCESS   -1
#define MAGICNUM      "P2"
#define ANCHO_IMAGEN  20
#define ALTO_IMAGEN   40


using namespace std;

typedef enum {
                OK,
                ERROR,
                ERROR_NULL_POINTER,
                ERROR_MEMORY_NO_AVAILABLE,
                ERROR_INVALID_TOKEN,
                ERROR_INSERT_TREE_NODE
                }status_t;

#endif
```

```
#ifndef _COMPLEJO_H_INCLUDED_
#define _COMPLEJO_H_INCLUDED_

#include <iostream>
#include <cmath>

using namespace std;

class complejo
{
private:
    double re_, im_;
public:
  /************************ CONSTRUCTORES ************************/
  complejo();
  complejo(double);
  complejo(double, double);
  complejo(const complejo &);
  /************************  DESTRUCTORES ************************/
  ~complejo();
  /************************ OPERADORES ****************************/
  complejo const &operator=(complejo const &);
  complejo const &operator*=(complejo const &);
  complejo const &operator+=(complejo const &);
  complejo const &operator-=(complejo const &);
  /************************** METODOS *****************************/
  double re() const;
  double im() const;
  double abs() const;
  double abs2() const;
  complejo const &conjugar();
  complejo const conjugado() const;
  bool zero() const;
  void redondeo(void);//en el caso que la parte decimal sea 0.5 se redondea hacia abajo.
  /********************** OPERADORES FRIEND ************************/
  friend complejo const operator+(complejo const &, complejo const &);
  friend complejo const operator-(complejo const &, complejo const &);
  friend complejo const operator*(complejo const &, complejo const &);
  friend complejo const operator/(complejo const &, complejo const &);
  friend complejo const operator/(complejo const &, double);
  friend bool operator==(complejo const &, double);
  friend bool operator==(complejo const &, complejo const &);
  friend std::ostream &operator<<(std::ostream &, const complejo &);
  friend std::istream &operator>>(std::istream &, complejo &);
  friend complejo const operator^(complejo const &, int const &);
  /********************** FUNCIONES FRIEND ************************/
  friend double const re(const complejo &);
  friend double const im(const complejo &);
  friend complejo const exp(const complejo &);
  friend complejo const id(const complejo &);
  friend double const abs(const complejo &);
  friend const complejo sinh(const complejo &);
  friend const complejo cosh(const complejo &);
  friend const complejo sin(const complejo &);
  friend const complejo cos(const complejo &);
  friend const double arg(const complejo &);
  /****************************************************************/
};

#endif
```

```
#include "complejo.hpp"
#include <iostream>
#include <cmath>

using namespace std;
/*****************************************************************/
/*************************** CONSTRUCTORES ***********************/
/*****************************************************************/
complejo::complejo()
{
  re_ = 0;
  im_ = 0;
}
complejo::complejo(double r)
{
  re_ = r;
  im_ = 0;
}
complejo::complejo(double r, double i)
{
  re_ = r;
  im_ = i;
}
complejo::complejo(complejo const &c)
{
  re_ = c.re_;
  im_ = c.im_;
}
/*****************************************************************/
/*************************** DESTRUCTORES ************************/
/*****************************************************************/
complejo::~complejo()
{
}
/*****************************************************************/
/*************************** OPERADORES **************************/
/*****************************************************************/
complejo const & complejo::operator=(complejo const &c)
{
  re_ = c.re_;
  im_ = c.im_;
  return *this;
}
complejo const & complejo::operator*=(complejo const &c)
{
  double re = re_ * c.re_ - im_ * c.im_;
  double im = re_ * c.im_ + im_ * c.re_;
  re_ = re, im_ = im;
  return *this;
}

complejo const & complejo::operator+=(complejo const &c)
{
  double re = re_ + c.re_;
  double im = im_ + c.im_;
  re_ = re, im_ = im;
  return *this;
}

complejo const & complejo::operator-=(complejo const &c)
{
  double re = re_ - c.re_;
  double im = im_ - c.im_;
  re_ = re, im_ = im;
  return *this;
}
/*****************************************************************/
/*************************** METODOS *****************************/
/*****************************************************************/
double complejo::re() const
{
  return re_;
```

```
}
double complejo::im() const
{
  return im_;
}

double complejo::abs() const
{
  return std::sqrt(re_ * re_ + im_ * im_);
}

double complejo::abs2() const
{
  return re_ * re_ + im_ * im_;
}

complejo const & complejo::conjugar()
{
  im_*= -1;
  return *this;
}

complejo const complejo::conjugado() const
{
  return complejo(re_, -im_);
}

bool complejo::zero() const
{
 #define ZERO(x) ((x) == +0.0 && (x) == -0.0)

  if(ZERO(re_) && ZERO(im_)) return true;
  else return false;
}
void complejo::redondeo(void)
{
  if( (((int)re_)+0.5) < re_)
  { ++re_;
    re_=(int)re_;
  }
  else re_=(int)re_;

  if( (((int)im_)+0.5) < im_)
  { ++im_;
    im_=(int)im_;
  }
  else im_=(int)im_;
}
/*****************************************************************************/
/************************* OPERADORES FRIEND *********************************/
/*****************************************************************************/
complejo const operator+(complejo const &x, complejo const &y)
{
  complejo z(x.re_ + y.re_, x.im_ + y.im_);
  return z;
}

complejo const operator-(complejo const &x, complejo const &y)
{
  complejo r(x.re_ - y.re_, x.im_ - y.im_);
  return r;
}

complejo const operator*(complejo const &x, complejo const &y)
{
  complejo r(x.re_ * y.re_ - x.im_ * y.im_,
             x.re_ * y.im_ + x.im_ * y.re_);
  return r;
}

complejo const operator/(complejo const &x, complejo const &y)
{
```

```
  return x * y.conjugado() / y.abs2();
}

complejo const operator/(complejo const &c, double f)
{
  return complejo(c.re_ / f, c.im_ / f);
}

bool operator==(complejo const &c, double f)
{
  bool b = (c.im_ != 0 || c.re_ != f) ? false : true;
  return b;
}

bool operator==(complejo const &x, complejo const &y)
{
  bool b = (x.re_ != y.re_ || x.im_ != y.im_) ? false : true;
  return b;
}

ostream &operator<<(ostream &os, const complejo &c)
{
  return os << "(" << c.re_<< "," << c.im_ << ")";
}

istream & operator>>(istream &is, complejo &c)
{
  int good = false;
  int bad  = false;
  double re = 0;
  double im = 0;
  char ch = 0;

  if (is >> ch && ch == '(')
  {
    if (is >> re && is >> ch && ch == ',' && is >> im && is >> ch && ch == ')')
      good = true;
    else
      bad = true;
  }
  else if (is.good())
  {
    is.putback(ch);
    if (is >> re)
        good = true;
    else
        bad = true;
  }

  if (good)
    c.re_ = re, c.im_ = im;
  if (bad)
    is.clear(ios::badbit);

  return is;
}
complejo const operator^(complejo const &z, int const &a)
{
  return(pow(abs(z),a)*exp(complejo(0,1)*a*arg(z)));
}
/*****************************************************************/
/************************** FUNCIONES FRIEND **********************/
/*****************************************************************/
double const re(const complejo & z)
{
  return z.re();
}
double const im(const complejo & z)
{
  return z.im();
}
const complejo exp(const complejo & z)
```

```
{
  return complejo(exp(z.re())*cos(z.im()),exp(z.re())*sin(z.im()));
}
const complejo id(const complejo & z)
{
  return complejo(z);
}
const double abs(const complejo & z)
{
  return sqrt(z.re() * z.re() + z.im() * z.im());
}
const complejo sinh(const complejo & z)
{
  return((exp(z)-exp((-1)*z))/2);
}
complejo const cosh(const complejo & z)
{
  return((exp(z)+exp((-1)*z))/2);
}
complejo const sin(const complejo & z)
{
  return sinh(complejo(0,1)*z)*complejo(0,-1);
}
const complejo cos(const complejo & z)
{
  return cosh(complejo(0,1)*z);
}
const double arg(const complejo & z)
{
  return atan(im(z)/re(z));
}
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
```

```
#ifndef PRINTERS_HPP
#define PRINTERS_HPP

#include"common.hpp"
#include<iostream>
#include <string>
#include <sstream>
#include<fstream>

void printMatrix(size_t**,ostream &,const size_t &,const size_t &);
void printImage(ostream &,size_t**,const size_t &,const size_t &,const size_t &);
void printString(string, ostream&);


#endif // PRINTERS_HPP_INCLUDED
```

```cpp
#include "printers.hpp"
#include "common.hpp"

void printMatrix(size_t** matrix, ostream &os,const size_t &row,const size_t &col)
{
   // os << number_of_elements[0] << " Hubs"<<"\n";
   for(size_t i=0 ; i<row ; ++i)
   {
       for(size_t j=0 ; j<col ; ++j)
       {
           os<<matrix[i][j]<<" ";
       }
       os<<"\n";
   }

}

void printImage(ostream &os,size_t** matrix,const size_t &row,const size_t &col,const size_t &maxInten)
{
    os<<MAGICNUM<<endl;
    os<<col<<" "<<row<<endl;
    os<<maxInten<<endl;

    //Impresión de matriz
    printMatrix(matrix,os,row,col);
}

void printString(string s, ostream& os){    os << s << "\n"; }
```

```
#ifndef OPERATION_HPP_INCLUDED
#define OPERATION_HPP_INCLUDED

#include <stack>
#include <iostream>
#include <stdlib.h>
#include <string>
#include <sstream>
#include <vector>

#include "common.hpp"
#include "binTree.hpp"
#include "complejo.hpp"

/*** Definiciones ***/

#define END_TABLEOP(x) ((x)->op==0 && (x)->prec==0 &&  \
                        (x)->assoc==0 && (x)->unary==0)


/*** Tabla de asociatividad de operadores ***/

typedef enum {
                 ASSOC_NONE=0,
                 ASSOC_LEFT,     //Izquierda a derecha
                 ASSOC_RIGHT     //Derecha a izquierda
               }t_assoc;

/*** Tabla de clasificador tipo de operadores ***/

typedef enum {
                 FUNCTION,          // Funciones
                 OPERATOR,          // Operadores matemáticos
                 SEPARATOR,         // Separadores: coma, punto y coma, etc.
                 PARENTESIS_OPEN,   // Abierto: Paréntesis, corchetes, llaves,etc.
                 PARENTESIS_CLOSE,  // Cerrado:  Paréntesis, corchetes, llaves,etc.
                 VAR_INDEP,         // Variable independiente
                 IMAGINARY_UNIT,    // Unidad imaginaria "j" o "i"
                 NONEOP           // Sin clasificación. Indicador de error.
               }t_typeop;

/*** Tabla de clasificador de operadores unarios***/
typedef enum {
                UNARY,         // Operador unario
                NOT_UNARY,     // Operador no unario
                BINARY_UNARY  // Operador binario y unario. Ej: menos (-X)
               }t_unaryop;

/*** Estructura para manejar los operadores ***/

typedef struct operation {
                           const char *op;  //Nombre o símbolo
                           int prec;        //Valor de precedencia
                           t_assoc assoc;    //Asociatividad
                           int unary;       //Es unario
                           t_typeop func;    //tipo de operador
                           complejo (*eval)(complejo &a1, complejo &a2); //Función
                         } t_operation;


/*** Evaluadores ***/
// Operadores
complejo eval_unminus(complejo &a1, complejo &a2);
complejo eval_unplus(complejo &a1, complejo &a2);
complejo eval_add(complejo &a1, complejo &a2);
complejo eval_sub(complejo &a1, complejo &a2);
complejo eval_mul(complejo &a1, complejo &a2);
complejo eval_div(complejo &a1, complejo &a2);
complejo eval_pow(complejo &a1, complejo &a2);

//Funciones
complejo eval_re(complejo &a1,complejo &a2);
```

```
complejo eval_im(complejo &a1,complejo &a2);
complejo eval_exp(complejo &a1,complejo &a2);
complejo eval_id(complejo &a1,complejo &a2);
complejo eval_abs(complejo &a1,complejo &a2);
complejo eval_sinh(complejo &a1,complejo &a2);
complejo eval_cosh(complejo &a1,complejo &a2);
complejo eval_sin(complejo &a1,complejo &a2);
complejo eval_cos(complejo &a1,complejo &a2);
complejo eval_arg(complejo &a1,complejo &a2);

/*** Evaluador árbol de operaciones ***/
t_operation* getOp(const string &);
complejo evaluateOpTree(binTree<string>&,const complejo&);
binTree<string>& constructionOpTree(const vector<string>&);


#endif
```

```
#include "operation.hpp"

extern t_operation ops[];
/*****************************************************
getOp: recibe un token a procesar, si es una
operación matemática lo busca en la tabla de operaciones
y devuelve un puntero a la posición en la misma, si el
token no está en la tabla lo descarta y devuelve NULL.
*****************************************************/

t_operation* getOp(const string &str)
{
    t_operation *op=ops;

  for( ; !END_TABLEOP(op) ; ++op)
  {
        // Devuelve la posición en el vector de op
    if(string(op->op)==str) return op;
  }

  return NULL;
}

/*****************************************************
                    EVALUADORES
*****************************************************/

/** Operadores **/

complejo eval_unminus(complejo &a1, complejo &a2)
{
  return a2-a1;
}

complejo eval_unplus(complejo &a1, complejo &a2)
{
  return a2+a1;
}

complejo eval_add(complejo &a1, complejo &a2)
{
  return a1+a2;
}

complejo eval_sub(complejo &a1, complejo &a2)
{
  return a1-a2;
}

complejo eval_mul(complejo &a1, complejo &a2)
{
    return a1*a2;
}

complejo eval_div(complejo &a1, complejo &a2)
{
    return a1/a2;
}

complejo eval_pow(complejo &a1, complejo &a2)
{
    return a1^a2.re();
}

/** Funciones **/
complejo eval_re(complejo &a1,complejo &a2)
{
    return re(a1);
}

complejo eval_im(complejo &a1,complejo &a2)
{
```

```
    return im(a1);
}
complejo eval_exp(complejo &a1,complejo &a2)
{
    return exp(a1);
}
complejo eval_id(complejo &a1,complejo &a2)
{
    return id(a1);
}
complejo eval_abs(complejo &a1,complejo &a2)
{
    return abs(a1);
}
complejo eval_sinh(complejo &a1,complejo &a2)
{
    return sinh(a1);
}
complejo eval_cosh(complejo &a1,complejo &a2)
{
    return cosh(a1);
}
complejo eval_sin(complejo &a1,complejo &a2)
{
    return sin(a1);
}
complejo eval_cos(complejo &a1,complejo &a2)
{
    return cos(a1);
}
complejo eval_arg(complejo &a1,complejo &a2)
{
    return arg(a1);
}


/*****************************************************
evaluateOpTree: recibe el árbol de operaciones y el
valor de la variable 'var' que será el que corresponde
a la función "z" (VARDEP: variable independiente) dentro
del árbol de operaciones. En el caso que en el árbol se
encuentre "z", al procesar será intercambiado e inter-
pretado por el valor numérico que contiene 'var'.
*****************************************************/

complejo evaluateOpTree(binTree<string>& tree,const complejo &var)
{
    //float result=0,num=0,lRes=0,rRes=0;
    double num;
    complejo result, lRes, rRes,zero(0,0);

    //Si el nodo es un número
    istringstream issNum(tree.getData());
    if(issNum>>num)
    {
        complejo numberC(num,0);
        return numberC;
    }

    //Si es una operación o función
    else
    {
        t_operation *op=getOp(tree.getData());
        if(!op) exit(1);

    // Si el token es la variable independiente "z"
        if(op->func==VAR_INDEP) return var;

        else if(op->func==IMAGINARY_UNIT)
        {
            complejo imag(0,1);
            return imag;
```

```
        }

    if (tree.hasLeftSon())
      lRes=evaluateOpTree(tree.getLeftSon(),var);

        if (tree.hasRightSon())
      rRes=evaluateOpTree(tree.getRightSon(),var);

    if(op->func==FUNCTION || op->unary==UNARY)
      result=op->eval(lRes,zero); // Tienen un solo hijo
    else
      result=op->eval(lRes,rRes);
    }

    return result;
}

/*****************************************************
constructionOpTree: recibe un vector de string con los
tokens en Notación Polaca Inversa (RPN), lo procesa y
construye el árbol binario de operaciones matemáticas
(binary expressions tree), y devuelve una referencia al
nodo raíz del mismo.
*****************************************************/

binTree<string>& constructionOpTree(const vector<string>& parserRPN)
{
   t_operation *op=NULL;
   float num;
   binTree<string> *node=NULL,*t1=NULL,*t2=NULL;
   stack<binTree<string>*> opTreeStk;

  // Mientras hay tokens por leer
  for(size_t i=0 ; i<parserRPN.size() ; i++)
  {
    // Si el token es un número
    istringstream issparser(parserRPN[i]);

    if(issparser>>num)
        {
            node=new binTree<string>(parserRPN[i]);
            opTreeStk.push(node);

        }

    // Si el token es una operación o función
    else if((op=getOp(parserRPN[i]))!=NULL)
    {
       node=new binTree<string>(string(op->op));

            // Si el token es una variable independiente o la
            // unidad imaginaria, apilarlo.
       if(op->func==VAR_INDEP || op->func==IMAGINARY_UNIT)
                opTreeStk.push(node);

            // Si el token es una función o un operador
      // unario, desapilo un nodo y se inserta el
      // nodo función u operador al stack luego.
       else if (  op->func==FUNCTION ||
                       (op->func==OPERATOR && op->unary==UNARY)
                   )
       {
                if(!opTreeStk.empty())
                {
                    t1=opTreeStk.top();
                    opTreeStk.pop();

                    // Insertar el nodo hijo al padre
                    node->insert(*t1);

                    // Colocar el nodo padre en la pila
                    opTreeStk.push(node);
```

```
                }
          else
                {   //si hay errores en el parserRPN
                    cerr<<"ERROR: Function entered "
                        <<"without independent variable"
                        << "or numeric value ."<<endl;
                    exit(EXIT_FAILURE);
                }
      }

      // Si el token es una operación, desapilar dos
      // nodos y los insertar al nodo padre operación
      else if(op->func==OPERATOR)
      {
       if(!opTreeStk.empty())
       {
           t2=opTreeStk.top();
                   opTreeStk.pop();
       }
       if(!opTreeStk.empty())
       {
           t1=opTreeStk.top();
                   opTreeStk.pop();
       }
                else
                {   //si hay errores en el parserRPN
                    cerr<<"ERROR: Operation entered incomplete,"
                        <<" missing terms."<<endl;
                    exit(EXIT_FAILURE);
                }

       // Insertar los nodos hijos al padre
       node->insert(*t1);
       node->insert(*t2);

       // Colocar el nodo padre en la pila
       opTreeStk.push(node);
      }
    }
  }

  // Si no hay más tokens por leer
  node=opTreeStk.top();
  opTreeStk.pop();

  if(!opTreeStk.empty())
  {
      cerr<<"ERROR: Expression entered incomplete."<<endl;
        exit(EXIT_FAILURE);
  }

  return (*node);

}
```

```
#compilacion y ejecucion de los archivos
CC       =  g++
CFLAGS   = -g -Wall  -pedantic
OBJS   = main.o   dictionary.o  cmdline.o options.o process.o printers.o complejo.o planoC.o
expression.o shuntingYard.o operation.o


muestras.exe:$(OBJS)
  $(CC)  $(OBJS) -o main.exe

main.o:main.cpp dictionary.hpp common.hpp cmdline.h options.hpp process.hpp printers.hpp complejo.hpp
planoC.hpp
  $(CC) main.cpp -c -o main.o $(CFLAGS)

dictionary.o:dictionary.cpp dictionary.hpp cmdline.h  options.hpp
  $(CC) dictionary.cpp -c -o dictionary.o $(CFLAGS)

cmdline.o: cmdline.cc cmdline.h
  $(CC)  cmdline.cc -c -o cmdline.o $(CFLAGS)

options.o: options.cpp options.hpp dictionary.hpp
  $(CC)  options.cpp -c -o options.o $(CFLAGS)

process.o:process.cpp process.hpp dictionary.hpp common.hpp planoC.hpp
  $(CC) process.cpp -c -o process.o $(CFLAGS)

printers.o:printers.cpp printers.hpp common.hpp
  $(CC) printers.cpp -c -o printers.o $(CFLAGS)

complejo.o:complejo.cpp complejo.hpp
  $(CC) complejo.cpp -c -o complejo.o $(CFLAGS)

planoC.o:planoC.cpp planoC.hpp
  $(CC) planoC.cpp -c -o planoC.o $(CFLAGS)

expression.o:expression.cpp expression.hpp common.hpp
  $(CC) expression.cpp -c -o expression.o $(CFLAGS)

shuntingYard.o:shuntingYard.cpp shuntingYard.hpp operation.hpp common.hpp
  $(CC) shuntingYard.cpp -c -o shuntingYard.o $(CFLAGS)

operation.o:operation.cpp operation.hpp common.hpp binTree.hpp complejo.hpp
  $(CC) operation.cpp -c -o operation.o $(CFLAGS)

clean:
  rm *.o
```

75.04/95.12 Algoritmos y Programación II
Trabajo práctico 1: Programación C++

Universidad de Buenos Aires - FIUBA
Segundo cuatrimestre de 2014
`$Date: 2014/10/05 22:49:17 $`

1. Objetivos

Ejercitar conceptos básicos de programación C++ e implementación de TDAs. Escribir un programa en este lenguaje (y su correspondiente documentación) que resuelva el problema que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá será entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Introducción

En este trabajo continuaremos desarrollando nuestra herramienta para procesar imágenes, de forma tal que el programa pueda recibir funciones de transformación arbitrarias.
Con esto en mente, nuestro programa deberá poder:

- Cargar una imagen a memoria desde un archivo o desde la entrada estándar.
- Aplicar una función arbitraria, pasada por línea de comando, a la imagen.
- Guardar una imagen en memoria a un archivo o sacarlo por la salida estándar.

4.1. Interfaz

Tanto en este TP, como en el siguiente, la interacción con el programa se dará a través de la línea de comando.

Formato. Por simplicidad se usará un formato de imágenes basado en archvios de texto: *portable graymap* o PGM, con codificación ASCII[1].
Este formato define un mapa de grises: cada pixel va a tener un valor que define su intensidad entre `0` (negro) y cierto número `max` (blanco).

1. La primer línea siempre contiene `P2`, el identificador del tipo de archivo o *magic number*.

2. Luego puede haber comentarios identificados con `#` al inicio de la línea. Estos comentarios deben ser ignorados por el programa.

3. Después se presenta el tamaño de la imagen. En el ejemplo de más abajo, `24` pixels de ancho y `7` de alto.

4. Una vez definido el tamaño encontramos el máximo valor de intensidad de la imagen. En el ejemplo, `15`.

5. Por último está la imágen en sí: cada número define la intensidad de un pixel, comenzando en el margen superior izquierdo de la imagen y barriéndola por líneas hacia abajo.

Ejemplo. En el siguiente ejemplo se puede ver una imagen en formato pgm (ampliada):

Y a continuación el contenido del archivo correspondiente:

```
P2
# Shows the word "FEEP" (example from Netpbm main page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Transformación. Para modificar la imagen usaremos una función compleja $f : \mathbb{C} \to \mathbb{C}$. Para esto se asocia cada pixel de la imagen a un número complejo $z = a + b \cdot i$. A lo largo de este TP, vamos a suponer que la imagen cubre una región rectangular del plano complejo, cuyas coordenadas son pasadas por línea de comando. Así, los pixels de la imagen conformarán una grilla de puntos contenida dentro de esta región.

Los pixels de la imagen destino se colorean aplicando la función: para cada punto z de la imagen destino se asocia con un punto $f(z)$ en la imagen origen. Es decir, esta transformación solamente deforma la imagen original sin alterar el color del pixel.

Teniendo en cuenta las dimensiones acotadas de nuestras imágenes, se van a dar los siguientes casos:

- z pertenece a la imagen destino y $f(z)$ cae dentro de la imagen origen: este es el caso esperable.
- z pertenece a la imagen destino y $f(z)$ cae fuera de la imagen origen: asumir que z es coloreado de negro.

Este tipo de transformación permite hacer un remapeo de las imágenes. Si la función involucrada es holomorfa, se trata de una transformación conforme: la imagen transformada conserva los ángulos de la imagen original [2].

Algoritmo. En este TP, el algoritmo de evaluación de funciones a implementar es el descripto en [3]. El mismo puede ser usado para reescribir expresiones *infix* en formato RPN para luego ser evaluadas durante el proceso de transformación.

Funciones. La funciones a implementar en este TP son expresiones arbitrarias conformadas por números complejos de la forma `a + b*j`, los operadores aritméticos usuales `+`, `-`, `*`, `/`, `^`, las funciones `exp(z)`, `ln(z)`, `re(z)`, `im(z)`, `abs(z)`, `phase(z)`, y paréntesis para agrupar subexpresiones cuando sea conveniente.

Se propone a los alumnos pensar e implementar distintas funciones $f(z)$ para usar por fuera del contexto de este. Luego procesar imágenes y mandar a la lista de mails de la materia la imagen original, la procesada y la función involucrada.

4.2. Línea de comando

Las opciones `-i` y `-o` permitirán seleccionar los streams de entrada y salida de datos respectivamente. Por defecto, éstos serán `cin` y `cout`. Lo mismo ocurrirá al recibir "`-`" como argumento de cada una.

La opción `-f` permite seleccionar qué función se quiere usar para procesar la imágen. Por defecto se usará la función identidad $f(z) = z$.

Al finalizar, todos nuestros programas retornarán un valor nulo en caso de no detectar ningún problema; y, en caso contrario, devolveremos un valor no nulo (por ejemplo `1`).

Como comentario adicional, el orden de las opciones es irrelevante. Por este motivo, no debe asumirse un orden particular de las mismas a la hora de desarrollar la toma de argumentos.

4.3. Ejemplos

Primero, transformamos la imagen `grid.pgm` con la función identidad: $f(z) = z$ y guardamos la salida en `grid-id.pgm`. Ver figura 1.

```
$ ./tp1 -i grid.pgm -o grid-id.pgm -f z
```

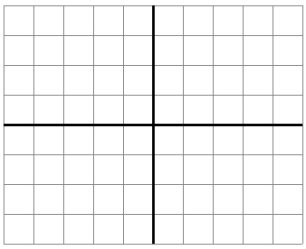


Figura 1: grid.pgm y grid-id.pgm

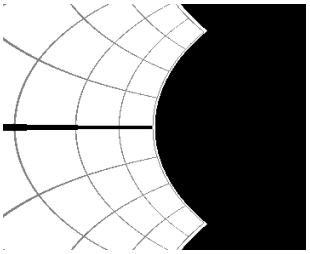


Figura 2: grid-exp.pgm

Ahora, transformamos con $f(z) = e^z$ y guardamos la salida en `evolution-exp.pgm`. Ver figuras 3 y 4.

```
$ ./tp1 -i evolution.pgm -o evolution-exp.pgm -f exp(z)
```

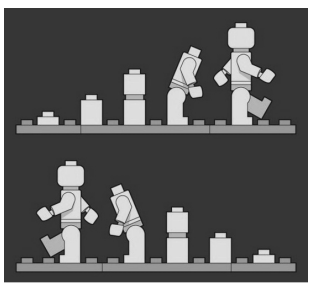


Figura 3: evolution.pgm

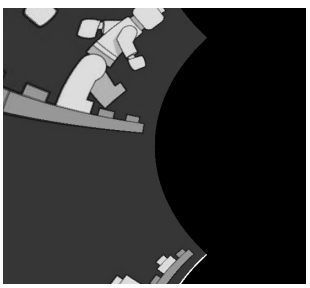


Figura 4: evolution-exp.pgm

Por último, transformamos la misma imagen con las funciones $f(z) = z^2$ y $f(z) = z^3$, dejando los resultados en `evolution-sqr.pgm` y `evolution-cube.pgm`. Ver figuras 5 y 6).

```
$ ./tp1 -i evolution.pgm -o evolution-sqr.pgm -f z^2
$ ./tp1 -i evolution.pgm -o evolution-cube.pgm -f z^3
```

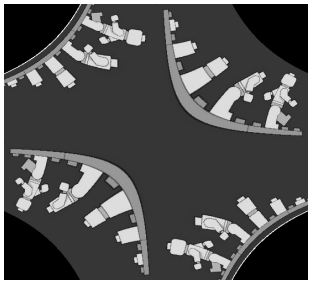


Figura 5: evolution-sqr.pgm

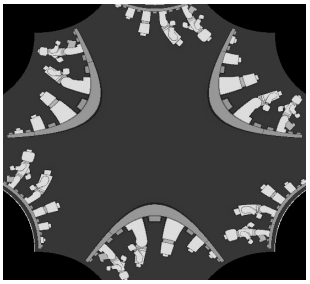


Figura 6: evolution-cube.pgm

Como siempre, estos ejemplos deben ser incluidos como punto de partida de los casos de prueba del trabajo prático.

4.4. Portabilidad

Es deseable que la implementación desarrollada provea un grado mínimo de portabilidad. Sugerimos verificar nuestros programas en alguna versión reciente de UNIX: BSD o Linux.

5. Informe

El contenido mínimo del informe deberá incluir:

- Una carátula que incluya los nombres de los integrantes y el listado de todas las entregas realizadas hasta ese momento, con sus respectivas fechas.
- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C++ (en dos formatos, digital e impreso).
- Este enunciado.

6. Fechas

La última fecha de entrega y presentación será el jueves 23/10.

Referencias

[1] Netpbm format (Wikipedia).
`http://en.wikipedia.org/wiki/Netpbm_format`

[2] Holomorphic function (Wikipedia).
`http://en.wikipedia.org/wiki/Holomorphic_function`

[3] Shunting yard algorithm (Wikipedia). `http://en.wikipedia.org/wiki/Shunting_yard_algorithm`.