

75.04/95.12 Algoritmos y Programación II
Trabajo Práctico 1: Estructuras de datos simples.

Universidad de Buenos Aires - FIUBA
Primer Cuatrimestre de 2020

Grupo 11
Nicanor Porta Chambergó - 97388
Ulises Montenegro - 102921

Entrega n° 1 → 18/06/2020

1. Introducción

El presente trabajo es una actualización del trabajo práctico 0. En esta nueva versión, el programa realiza transformaciones del tipo " $\sin(z^3+8i)$ ", es decir podrá realizar operaciones con los operadores (suma, sustracción, multiplicación, división y potenciación) y funciones como seno, coseno, logaritmo, etc. Para el diseño e implementación se utilizaron los mismos TDA que en el trabajo anterior (TP 0) y algunos nuevos.

2. Diseño e implementación

En esta sección se detallarán los cambios en los TDA utilizados en el trabajo anterior y se especificarán los nuevos.

2.1 TDA Pila

Clase	Pila
Atributos Privados	-Lista<T> *p_entrada
Métodos Públicos	+Constructor por defecto. +Destructor. +push +pop +tope +vacía

Tabla 1: UML de la clase Pila.

A continuación los detalles de los métodos de la Tabla 1:

- Constructor por defecto:
 - Descripción: Construye un objeto de la clase pila.
 - Pre condición: No tiene.
 - Post condición: Objeto de la clase Pila listo para ser usado.
- Destructor:
 - Descripción: Destruye el objeto.
 - Pre condición: Debe existir un objeto creado.
 - Post condición: No tiene..
- push:
 - Descripción: Introduce un elemento a la pila.
 - Pre condición: Debe existir una pila. Recibe un argumento que debe ser del tipo que almacene la pila <T>.

- Post condición: Devuelve, por referencia, la pila a la que se le aplica el método con el elemento pasado por argumento en el tope de la misma.
- pop:
 - Descripción: Elimina un elemento de la pila.
 - Pre condición: Debe existir una pila que no esté vacía.
 - Post condición: Devuelve, por referencia, la pila a la cual se le aplica el método sin el último elemento agregado (en el tope).
- tope:
 - Descripción: Obtiene el elemento en el tope de la pila.
 - Pre condición: Debe existir una pila que no esté vacía.
 - Post condición: Devuelve el elemento que está en el tope de la pila, no modifica la pila.
- vacia:
 - Descripción: Verifica si es verdad que la pila está vacía.
 - Pre condición: Debe existir una pila.
 - Post condición: Devuelve un booleano. Si es true quiere decir que la pila está vacía, si es false entonces la pila tiene al menos un elemento.

2.2 TDA Queue

Clase	Queue
Atributos Privados	-first (Nodo<T> *) -last (Nodo<T> *)
Métodos Públicos	+Constructor por defecto. +Destructor. +enqueue +dequeue +empty +emitir +front

Tabla 2: UML de la clase Queue.

A continuación los detalles de los métodos de la Tabla 2:

- Constructor por defecto:
 - Descripción: Construye un objeto de la clase Queue por defecto.
 - Pre condición: No tiene.

- Post condición: Inicializa los punteros first y last a nulo. Lista para ser usada.
- Destructor:
 - Descripción: Destruye el objeto.
 - Pre condición: Debe existir el objeto.
 - Post condición: No tiene.
- enQueue:
 - Descripción: Da de alta en la cola a un elemento (que se ubicará al fondo) pasado por argumento
 - Pre condición: La cola debe existir.
 - Post condición: la cola queda modificada con la inserción del nuevo elemento
- deQueue:
 - Descripción: Elimina el elemento del frente de la cola
 - Pre condición: La cola debe existir y no debe estar vacía.
 - Post condición: La cola es modificada por la eliminación del elemento del frente.
- empty:
 - Descripción: Retorna un valor que indica si la cola está vacía.
 - Pre condición: La cola debe existir.
 - Post condición: No tiene.
- Emitir:
 - Descripción: Emite lo que contiene la cola.
 - Pre condición: La cola debe existir.
 - Post condición: No tiene.
- front:
 - Descripción: Retorna el valor del primer elemento de la cola.
 - Pre condición: La cola debe existir y no debe estar vacía.
 - Post condición: No tiene.

2.3 TDA Complejo

Este TDA sufrió algunos cambios respecto del utilizado en el TP 0.
En principio se agregó un método a la parte privada:

- pow2:

- Descripción: Eleva un Complejo al cuadrado.
- Pre condición: Debe existir un objeto de la clase Complejo.
- Post condición: Devuelve un Complejo que es el resultado de elevar al cuadrado el Complejo al cual se le aplica el método.

Este método se ubica en la parte privada ya que va a ser utilizado en el método que eleva un Complejo al número que se le pase como argumento.

También se agregaron 2 constructores:

- Constructor por un valor:
 - Descripción: Construye un objeto de la clase Complejo.
 - Pre condición: Recibe como argumento un elemento que debe ser un double.
 - Post condición: Crea un Complejo con su parte real con el valor pasado por argumento.

Este constructor puede utilizarse para castear un double a un Complejo de la siguiente manera: double a; Complejo b; b=(Complejo)a;

- Constructor por cadena:
 - Descripción: Construye un objeto de la clase Complejo.
 - Pre condición: Recibe un parámetro, por referencia, que debe ser un string del tipo "a+bi".
 - Post condición: Crea un Complejo convirtiendo a double la parte real e imaginaria. En caso de no recibir alguna de las dos, la crea con el 0.0.

Luego se agregaron los siguientes métodos:

- aString:
 - Descripción: Convierte un Complejo a un string.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve, por referencia, un string que es del tipo "a+bi" donde a es la parte real del Complejo al que se le aplica el método y b es la parte imaginaria.
- aplicarFuncion:
 - Descripción: Aplica una función pasada como argumento al Complejo al cual se le aplica el método.
 - Pre condición: Debe existir un Complejo, recibe un argumento que debe ser un string.
 - Post condición: Devuelve un Complejo que es el resultado de aplicar la función pasada por argumento al objeto al cual se le aplica el método.

Si se le pasa una función que no existe en esta clase, devuelve el objeto sin haberle aplicado ninguna función (*this).

- aplicarOperador:
 - Descripción: Aplica un operador pasado como argumento entre el objeto al cual se le aplica el método y un Complejo pasado como argumento.
 - Pre condición: Debe existir un Complejo. Recibe dos argumentos: uno, por referencia, que debe ser un Complejo y otro que debe ser un string.
 - Post condición: Devuelve un Complejo que es el resultado de aplicar el operador pasado como argumento (string) entre el objeto al cual se le aplica el método y el Complejo pasado por argumento. Si se le pasa un operador que no está en la clase, devuelve el objeto sin haberle aplicado ningún operador (*this).
- conjugado:
 - Descripción: Conjuga el Complejo.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve un Complejo que es el conjugado del objeto al que se le aplica el método.
- sinc:
 - Descripción: Aplica la función seno a un Complejo.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve un Complejo que es el seno del objeto al cual se le aplica el método.
- cosc:
 - Descripción: Aplica la función coseno a un Complejo.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve un Complejo que es el coseno del objeto al cual se le aplica el método.
- Inc:
 - Descripción: Aplica la función logaritmo natural a un Complejo.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve un Complejo que es el logaritmo natural del objeto al cual se le aplica el método.
- phase:
 - Descripción: Devuelve la fase o argumento de un Complejo.

- Pre condición: Debe existir un Complejo.
- Post condición: Devuelve un double que es el argumento del Complejo al cual se le aplica el método.

Los métodos sumar y restar se eliminaron y fueron reemplazados por la sobrecarga de los operadores + y - respectivamente. Además se sobrecargaron otros operadores:

- Sobrecargar operador +:
 - Descripción: Suma el Complejo al que se le aplica el método con el Complejo pasado por argumento.
 - Pre condición: Debe existir el Complejo. Recibe un argumento que debe ser un Complejo.
 - Post condición: Devuelve un Complejo que es la suma del objeto al cual se le aplica el método y el Complejo pasado por argumento.
- Sobrecarga operador -:
 - Descripción: Resta el Complejo al que se le aplica el método con el Complejo pasado por argumento.
 - Pre condición: Debe existir el Complejo. Recibe un argumento que debe ser un Complejo.
 - Post condición: Devuelve un Complejo que es la resta del objeto al cual se le aplica el método y el Complejo pasado por argumento.
- Sobrecarga operador -:
 - Descripción: Realiza el negado de un Complejo.
 - Pre condición: Debe existir un Complejo.
 - Post condición: Devuelve un Complejo que es el objeto al cual se le aplica el método negado.
- Sobrecarga operador *:
 - Descripción: Multiplica el Complejo al que se le aplica el método con el Complejo pasado por argumento.
 - Pre condición: Debe existir el Complejo. Recibe un argumento que debe ser un Complejo.
 - Post condición: Devuelve un Complejo que es la multiplicación del objeto al cual se le aplica el método y el Complejo pasado por argumento.
- Sobrecarga operador /:
 - Descripción: Divide el Complejo al que se le aplica el método con el Complejo pasado por argumento.

- Pre condición: Debe existir el Complejo. Recibe un argumento que debe ser un Complejo.
- Post condición: Devuelve un Complejo que es la división del objeto al cual se le aplica el método y el Complejo pasado por argumento.
- Sobrecarga operador ^:
 - Descripción: Eleva el Complejo al que se le aplica el método a la potencia que se pase como argumento.
 - Pre condición: Debe existir el Complejo. Recibe un argumento que debe ser un int.
 - Post condición: Devuelve un Complejo que es el objeto al cual se le aplica el método elevado a la potencia que se pase como argumento.
- Sobrecarga operador int:
 - Descripción: Castea un Complejo a un int. Por ejemplo: Complejo a; int b; b = (int)a;
 - Pre condición: Debe existir el Complejo.
 - Post condición: Devuelve un int que es la parte real del objeto al cual se le aplica el método casteado a int.

2.4 TDA Imagen

Clase	Imagen
Atributos	-ancho (int) -alto (int) -intensidad_maxima (int) -precision_ancho (double) -precision_alto -mapa (**Pixel)
Métodos Públicos	+operar

Tabla 3: UML de la clase Imagen.

A continuación los detalles de los métodos de la Tabla 3:

- operar
 - Descripción: A partir de la cola donde se encuentra la operación, resuelve la operación mandada
 - Pre condición: Debe existir la cola y la operación debe estar en notación polaca inversa.

- Post condición: Ninguna.

2.5 TDA Errores

Clase	Errores
Atributos	-
Métodos Públicos	+ noSePuedeOperar +operacionInvalida

Tabla 4: UML de la clase Errores

Al igual que en la versión anterior, los método públicos de la clase Errores tienen los siguientes detalles

- Descripción: Imprimen un mensaje de error y cierran el programa.
- Pre condición: No tienen.
- Post condición: Terminan la ejecución del programa.
-

2.5 TDA Cmdline

Clase	Cmdline
Atributos	-
Métodos Públicos	+ parseOperacion +preParse

Tabla 5: UML de la clase Cmdline.

- preParse:
 - Descripción: Valida la expresión de la operación
 - Pre condición: No tiene.
 - Post condición: Terminan la ejecución del programa en caso de que la operación es incorrecta.
- parseOperacion:
 - Descripción: Parsea la operación en notación polaca inversa(rpn).
 - Pre condición: La expresión de la operación tiene que ser correcta.
 - Post condición: No tiene.

3. Compilación

Para realizar la compilación de los archivos del programa se creó un archivo tipo *makefile* que se encarga de crear el archivo ejecutable *tp0* . Gracias a este archivo podemos usar la línea de comando *make* y se creará el ejecutable (ver figura 1).

```
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP0/codigo3.1$ make
g++ -Wall -g -pedantic -o main.o -c main.cpp
g++ -Wall -g -pedantic -o cmdline.o -c cmdline.cpp
g++ -Wall -g -pedantic -o Imagen.o -c Imagen.cpp
g++ -Wall -g -pedantic -o Pixel.o -c Pixel.cpp
g++ -Wall -g -pedantic -o Complejo.o -c Complejo.cpp
g++ -Wall -g -pedantic -o Errores.o -c Errores.cpp
g++ -Wall -g -pedantic -o tp0 main.o cmdline.o Imagen.o Pixel.o Complejo.o Errores.o
rm *.o
```

Figura 1: Compilación mediante el archivo *makefile*.

La Tabla 6 muestra algunas maneras de ejecutar el programa. En caso de que “*Imagen.pgm*”, “*ImagenSalida.pgm*” o “*función*” sea reemplazado por un “-”, el programa se ejecuta mediante las entradas estándar o por defecto.

Comando de línea	Interpretación
<code>\$/tp0 -i Imagen.pgm -o ImagenSalida.pgm -f función</code>	El archivo de entrada <i>Imagen.pgm</i> es transformada por el parámetro <i>función</i> , luego la imagen transformada se imprime en el archivo de <i>ImagenSalida.pgm</i>
<code>\$/tp0 -i - -o ImagenSalida.pgm -f función</code>	Se toma la entrada estándar como flujo de entrada y como flujo de salida el archivo <i>ImagenSalida.pgm</i>
<code>\$/tp0 -i - -o - -f -</code>	Se toma la entrada estándar como flujo de entrada, la salida estándar como flujo de salida y se aplica la función por defecto <i>identidad</i> .

Tabla 6: Maneras de ejecutar el ejecutable *tp0*.

4. Corridas de prueba

4.1 Aplicando la función $\text{re}(z)$

Mediante el comando `$/tp1 -i pepper.pgm -o pepper_re.pgm -f “ $\text{re}(z)$ ”` se obtiene:



Figura 2a:pepper.pgm.

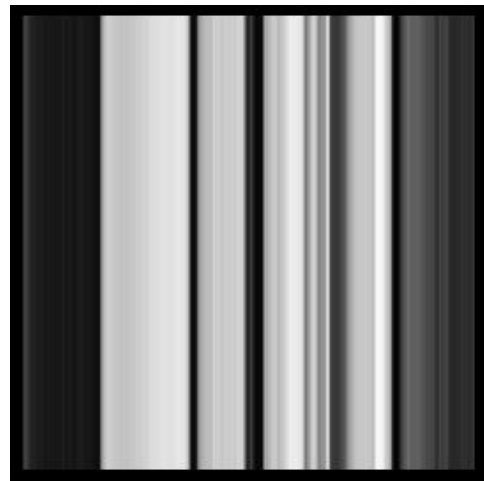


Figura 2b: pepper_re.pgm.

Figura 2: Transformación real de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_re.pgm -f re(z)` se obtiene:



Figura 3a:coins.pgm.

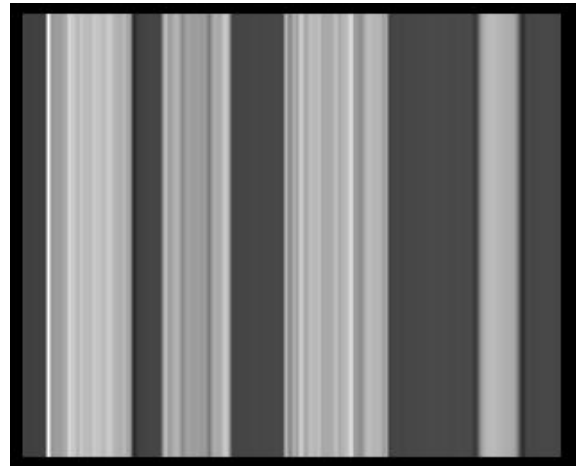


Figura 3b:coins_re.pgm.

Figura 3:Transformación real de la imagen coins.pgm.

4.2 Aplicando la función $\text{im}(z)$

Mediante el comando `./tp01-i pepper.pgm -o pepper_im.pgm -f "im(z)"` se obtiene:



Figura 4a:pepper.pgm.

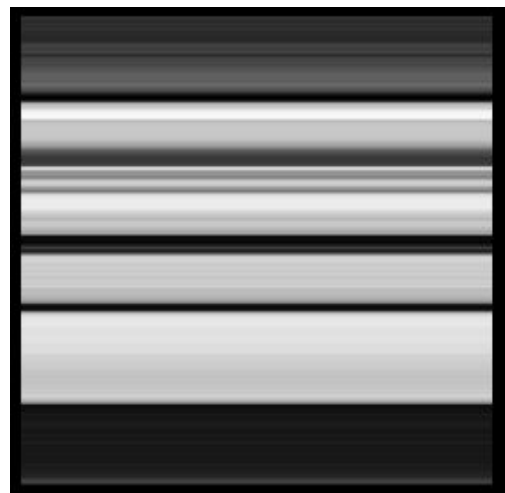


Figura 4b: pepper_im.pgm.

Figura 4:Transformación imaginaria de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_im.pgm -f "im(z)"` se obtiene:



Figura 5a:coins.pgm.

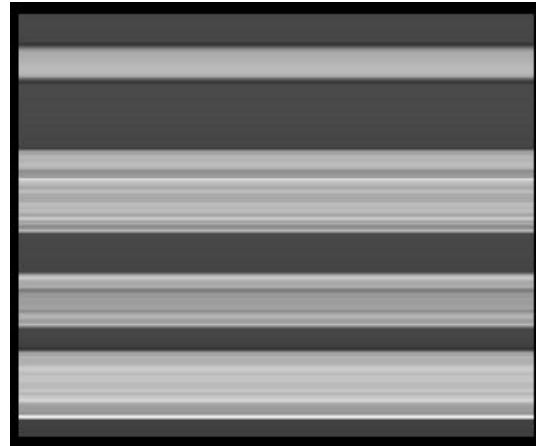


Figura 5b:coins_im.pgm.

Figura 5: Transformación imaginaria de la imagen coins.pgm.

4.3 Aplicando la función $\ln(z)$

Mediante el comando `./tp1 -i pepper.pgm -o pepper_ln.pgm -f "ln(z)"` se obtiene:



Figura 6a:pepper.pgm.



Figura 6b: pepper_ln(z).pgm.

Figura 6: Transformación logarítmica de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_ln.pgm -f "ln(z)"` se obtiene:



Figura 7a:coins.pgm.

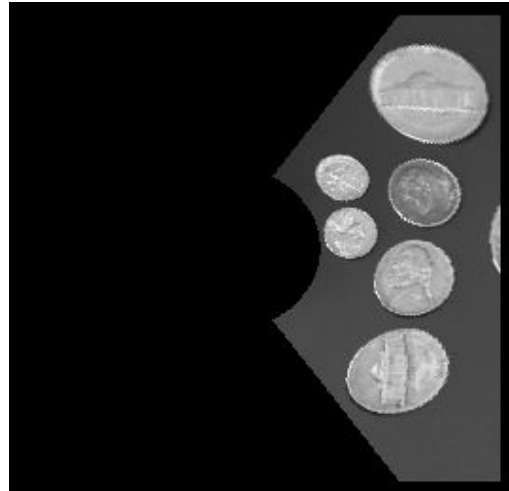


Figura 7b:coins_ln.pgm.

Figura 7: Transformación logarítmica de la imagen coins.pgm.

4.4 Aplicando la función $\text{abs}(z)$

Mediante el comando `./tp1 -i pepper.pgm -o pepper_abs.pgm -f "abs(z)"` se obtiene:



Figura 8a:pepper.pgm.

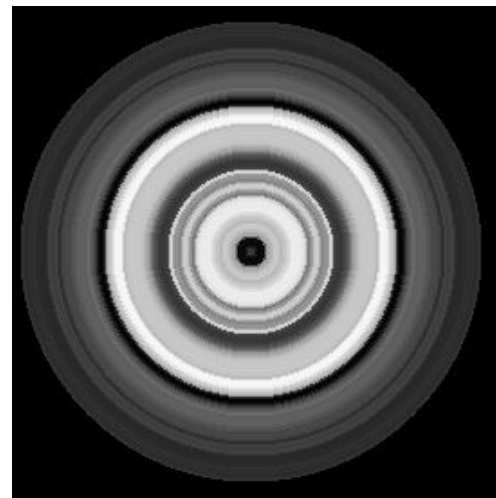


Figura 8b: pepper_abs.pgm.

Figura 8: Transformación módulo de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_abs.pgm -f abs(z)` se obtiene:



Figura 9a:coins.pgm.

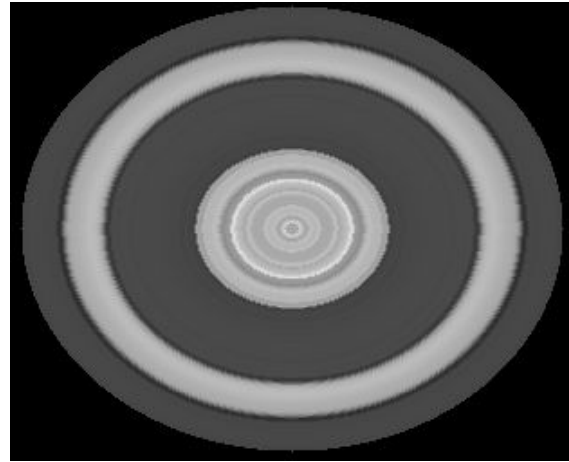


Figura 9b:coins_abs.pgm.

Figura 9:Transformación módulo de la imagen coins.pgm.

4.5 Aplicando la función $\text{phase}(z)$

Mediante el comando `./tp01-i pepper.pgm -o pepper_phase.pgm -f "phase(z)"` se obtiene:



Figura 10a:pepper.pgm.

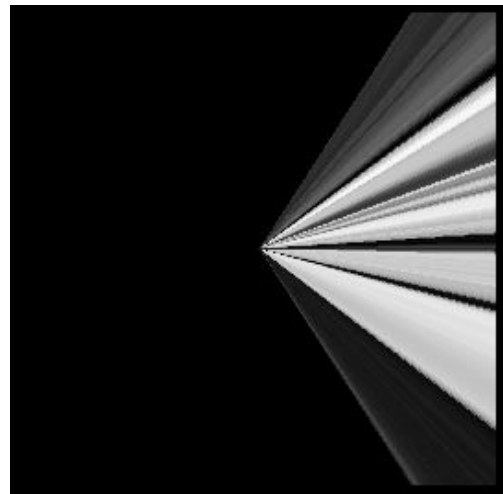


Figura 10b: pepper_phase.pgm.

Figura 10:Transformación ángulo de la imagen pepper.pgm.

Mediante el comando `.tp1 -i coins.pgm -o coins_phase.pgm -f "phase(z)"` se obtiene:



Figura 11a:coins.pgm.

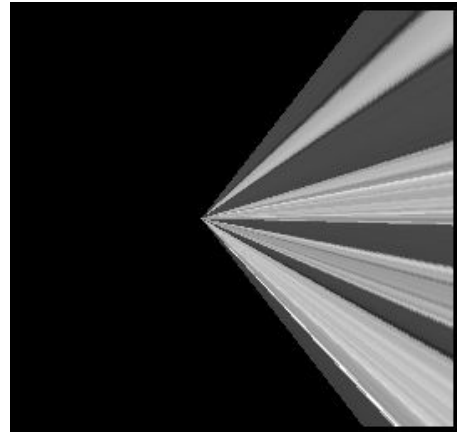


Figura 11b:coins_phase.pgm.

Figura 11: Transformación ángulo de la imagen coins.pgm.

4.6 Aplicando la función $\sin(z)$

Mediante el comando `.tp1 -i pepper.pgm -o pepper_sin.pgm -f "sin(z)"` se obtiene:



Figura 12a:pepper.pgm.



Figura 12b: pepper_sin(z).pgm.

Figura 12: Transformación seno de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_sin.pgm -f "sin(z)"` se obtiene:



Figura 13a:coins.pgm.



Figura 13b:coins_sin.pgm.

Figura 13:Transformación seno de la imagen coins.pgm.

4.7 Aplicando la función $\cos(z)$

Mediante el comando `./tp01-i pepper.pgm -o pepper_cos.pgm -f "cos(z)"` se obtiene:



Figura 14a:pepper.pgm.



Figura 14b: pepper_cos.pgm.

Figura 14:Transformación coseno de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_cos.pgm -f "cos(z)"` se obtiene:



Figura 15a:coins.pgm.

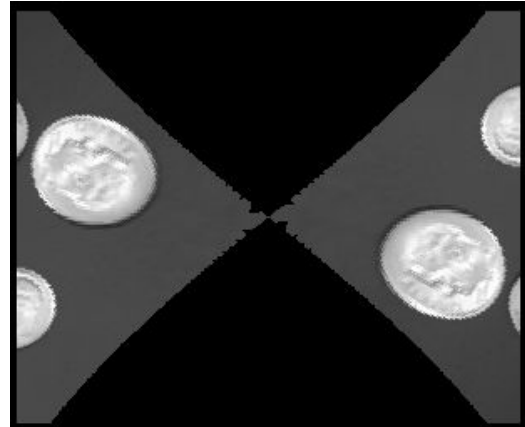


Figura 15b:coins_cos.pgm.

Figura 15: Transformación \cos de la imagen coins.pgm.

4.8 Aplicando la función -z

Mediante el comando `./tp1 -i pepper.pgm -o pepper_negado.pgm -f "-z"` se obtiene:



Figura 16a:pepper.pgm.



Figura 16b:pepper_negado.pgm.

Figura 16: Transformación \neg de la imagen pepper.pgm.

Mediante el comando `.tp1 -i coins.pgm -o coins_negado.pgm -f "-z"` se obtiene:



Figura 17a:coins.pgm.



Figura 17b:coins_negado.pgm.

Figura 17: Transformación negado de la imagen coins.pgm.

4.9 Aplicando la función z^4

Mediante el comando `.tp1 -i pepper.pgm -o pepper_pow4.pgm -f "z^4"` se obtiene:



Figura 18a:pepper.pgm.

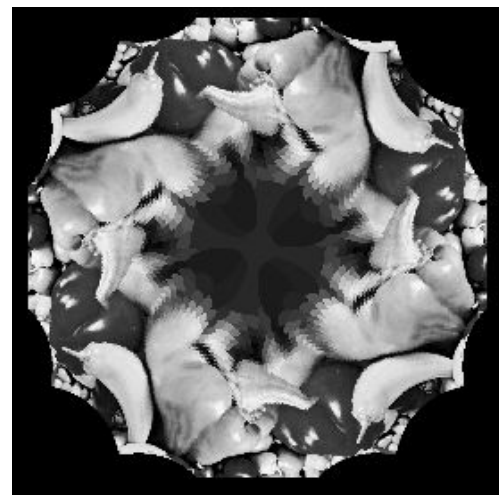


Figura 18b:pepper_pow4.pgm.

Figura 18: Transformación potencia de la imagen pepper.pgm.

Mediante el comando `.tp1 -i coins.pgm -o coins_pow4.pgm -f "z^4"` se obtiene:



Figura 19a:coins.pgm.

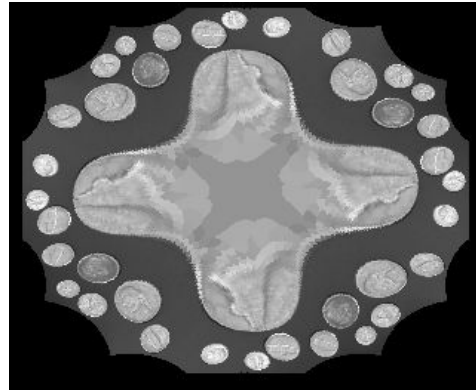


Figura 19b:coins_pow4.pgm.

Figura 19: Transformación potencia de la imagen coins.pgm.

4.10 Aplicando otras funciones

Mediante el comando `.tp1 -i pepper.pgm -o pepper_suma.pgm -f "z+(0.8+0.8i)"` se obtiene:



Figura 20a:pepper.pgm.

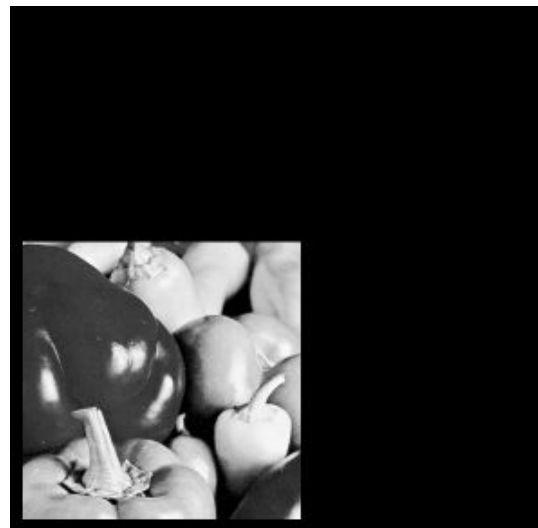


Figura 20b:pepper_suma.pgm.

Figura 20: Transformación suma de la imagen pepper.pgm.

Mediante el comando `./tp1 -i coins.pgm -o coins_func.pgm -f "sin(exp((z/4)^3*100*pi*i))*exp(-2i*pi/4)"` se obtiene:



Figura 21a:coins.pgm.

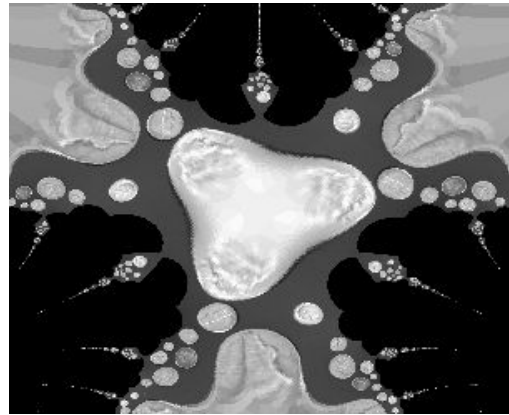


Figura 21b:coins_funcion.pgm.

Figura 21: Transformación de la imagen coins.pgm.

Mediante el comando `./tp1 -i pepper.pgm -o pepper_funcion.pgm -f "cos(z*6i*sin(2+i))"` se obtiene:



Figura 22a:pepper.pgm.



Figura 22b:pepper_funcion.pgm.

Figura 22: Transformación suma de la imagen pepper.pgm.

4.11 Construyendo videos

Con ayuda de la terminal de linux y los programas *imagemagick*, *convert* y *ffmpeg* se crearon dos videos.

El video1 se consigue a partir de los siguientes comandos:

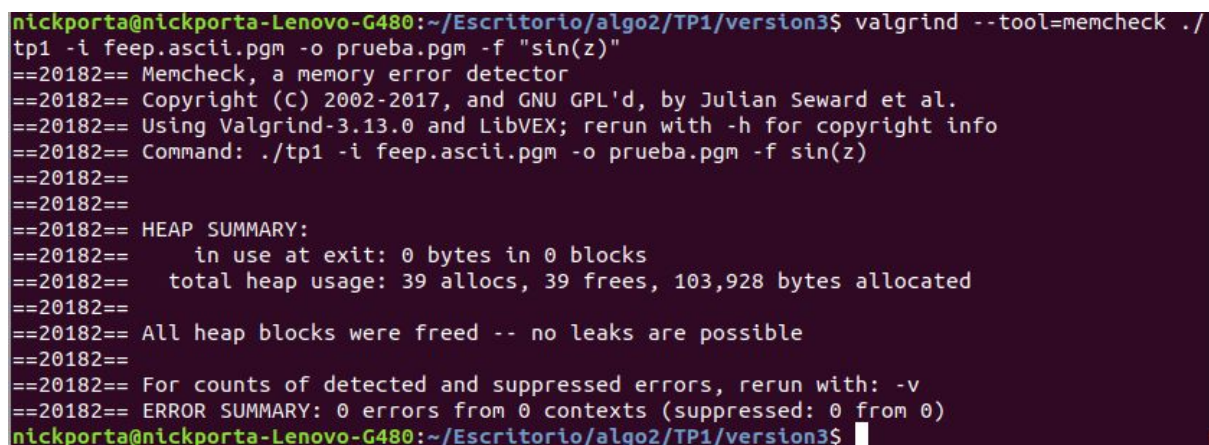
```
2030 for x in `seq -w 0 359`; do ./tp1 -i pepper.ascii.pgm -o frame-$x.pgm -f
'sin(z^4)*exp(i*$x*2*pi/360)'; done
2031 for f in *pgm; do convert $f $f.jpg; done
2032 ffmpeg -framerate 30 -pattern_type glob -i '*.jpg' -s 1920x1080 out.mp4
```

El video2 se consigue a partir de los siguientes comandos:

```
1996 for x in `seq -w 0 359`; do ./tp1 -i baboon.ascii.pgm -o frame-$x.pgm -f
'sin(exp((z/4)^3*100*pi*i))*exp(i*$x*2*pi/360)'; done
1997 for f in *pgm; do convert $f $f.jpg; done
1998 ffmpeg -framerate 30 -pattern_type glob -i '*.jpg' -s 1920x1080 out.mp4
```

4.12 Ejecución con valgrind

La figura 22 muestra una ejecución del programa usando el comando de línea valgrind, para verificar que el programa no presenta fugas de memoria ni errores. El resultado es que no presenta fugas de memoria ni errores.



```
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ valgrind --tool=memcheck ./
tp1 -i feep.ascii.pgm -o prueba.pgm -f "sin(z)"
==20182== Memcheck, a memory error detector
==20182== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20182== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20182== Command: ./tp1 -i feep.ascii.pgm -o prueba.pgm -f sin(z)
==20182==
==20182==
==20182== HEAP SUMMARY:
==20182==   in use at exit: 0 bytes in 0 blocks
==20182==   total heap usage: 39 allocs, 39 frees, 103,928 bytes allocated
==20182==
==20182== All heap blocks were freed -- no leaks are possible
==20182==
==20182== For counts of detected and suppressed errors, rerun with: -v
==20182== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$
```

Figura 22: Ejecución con valgrind.

4.13 Ejecución con operaciones erróneas

La respuesta del programa en caso de que exista algún error en la expresión de la operación es “operación inválida”, tal como se muestra en la figura 23.

```

nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "-84.56*(z+0.0001)"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "-84.56*z+0.0001)"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "-84.56*(z^^+34i)"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "-84.56&(z^^+34i)"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "cos(sin6-34i)"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "cos(sin(abs()))"
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "cos(sin(abs("
Operación inválida
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ █

```

Figura 23: Ejecución con operaciones erróneas.

4.13 Ejecución con expresiones de operaciones correctas

La figura 24 muestra la ejecución de algunas operaciones correctamente escritas, en principio se nota que el programa termina su ejecución sin retornar algún error.

```

nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "abs(6)"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "sin(abs(6))"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "sin(cos(z^6))"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "z^8+(45.6+i-23i)"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "(-z)^8+(45.6+i-23i)"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ ./tp1 -i feep.a
scii.pgm -o prueba.pgm -f "sin(z+(45.6+i-23i))"
nickporta@nickporta-Lenovo-G480:~/Escritorio/algo2/TP1/version3$ █

```

Figura 24: Ejecución con operaciones correctas.

5. Conclusión

Teniendo en cuenta las pruebas realizadas observamos y concluimos que el programa funciona de la manera esperada. Comparando con el trabajo anterior realizamos la misma observación de la pérdida de resolución debida a la incertidumbre existente al asignar cada pixel a un número complejo.

6. Código

Template_clase_nodo.h

```
1  #ifndef CLASE_NODO_H_INCLUDED
2  #define CLASE_NODO_H_INCLUDED
3  // #ifndef NULL
4  // #define NULL 0
5
6  #include <iostream>
7
8  template<typename T>
9  class Nodo{
10
11 private:
12     T info;
13     Nodo<T> *sig;
14
15 public:
16     Nodo(T, Nodo<T> *); //Constructor
17     Nodo(const T& _v):info(_v),sig(0){} //Constructor
18
19     T getInfo();
20     Nodo<T> * getSig();
21     void setInfo(T);
22     void setSig(Nodo<T> *);
23 };
24
25
26 /*template<typename T>
27 Nodo<T>::Nodo(T x, Nodo<T> *p=0):info(x),sig(p){ //Asigna x al atributo info. Asigna p al atributo sig. Si
no pasan el segundo argumento lo toma como un 0.
28 }*/
29
30 template<typename T>
31 Nodo<T>::Nodo(T x, Nodo<T> *p){
32     info = x;
33     sig = p;
34 }
35
36 template<typename T>
37 T Nodo<T>::getInfo(){
38     return info;
39 }
40
41 template<typename T>
42 Nodo<T> *Nodo<T>::getSig(){
43     return sig;
44 }
45
46 template<typename T>
47 void Nodo<T>::setInfo(T x){
48     info = x;
49 }
50
51 template<typename T>
52 void Nodo<T>::setSig(Nodo<T> *p){
53     sig = p;
54 }
55
56
57 #endif // CLASE_NODO_H_INCLUDED
```


Template_clase_lista.h

```
1  #ifndef CLASE_LISTA_H_INCLUDED
2  #define CLASE_LISTA_H_INCLUDED
3
4  #include "Template_clase_nodo.h"
5
6
7  template<typename T>
8  class Lista{
9  private:
10     Nodo<T> *primero;
11
12 public:
13     Lista(); //Constructor
14     ~Lista(); //Destructor
15
16     void altafin(T);
17     void altaprin(T);
18     void altapos(T, int);
19     void baja(T);
20     void bajaprin();
21     bool busca(T);
22     T primerElemento();
23     void emite() const;
24     int conteoNodos() const; //Cuenta los nodos que tiene la lista.
25     int valoresPositivos() const; //Devuelve 1 si todos los valores de la lista son positivos, de lo
contrario devuelve 0.
26     Lista<T> *inversion(); //Devuelve una lista con los nodos en el orden inverso. REVISAR!
27     void pertenece(Lista<T>); //HACER!!!
28     Lista<T> interseccionListas(Lista<T>); //VERIFICAR CUANDO SE REPITEN LAS INTERSECCIONES, VER POR QUE NO
ANDA!
29     Lista<T> unionListas(Lista<T>); //HACER!!!
30 };
31
32
33 template<typename T>
34 Lista<T>::Lista(){
35     primero = NULL;
36 }
37
38 template<typename T>
39 Lista<T>::~~Lista(){
40     if(primero){ //Si primero no es NULL
41         Nodo<T> *aux = primero;
42         while(aux){ //Mientras aux no sea NULL
43             primero = primero->getSig();
44             delete aux;
45             aux = primero;
46         }
47     }
48 }
49
50 template<typename T>
51 void Lista<T>::altafin(T x){
52     Nodo<T> *aux = primero;
53
54     while((aux) && (aux->getSig())) //Mientras aux y aux->sig sean distintos de NULL
55         aux = aux->getSig();
56     if(aux){ //Si aux es distinto de NULL (aux->sig es NULL)
57         aux->setSig(new Nodo<T>(x, aux->getSig()));
58     }
59     else
60         primero = new Nodo<T>(x, primero);
61 }
62
63 template<typename T>
64 void Lista<T>::altaprin(T x){
```

```

65     primero = new Nodo<T>(x, primero); //Llama al constructor apuntando a primero y asigna x a info.
66 }
67
68 template<typename T>
69 void Lista<T>::altapos(T x, int pos){
70     if(pos == 0)
71         altaprin(x);
72     else{
73         Nodo<T> *aux1 = primero;
74         Nodo<T> *aux2;
75         int contador = 0;
76         while(((aux1) && (aux1->getSig())) && (contador < pos-1)){
77             aux1 = aux1->getSig();
78             contador++;
79         }
80         if(aux1){
81             aux2 = new Nodo<T>(x);
82             aux2->setSig(aux1->getSig());
83             aux1->setSig(aux2);
84         }
85         else
86             altafin(x);
87     }
88 }
89
90 template<typename T>
91 void Lista<T>::baja(T x){ //Da de baja al primer nodo que contenga x.
92     if(primeros){
93         if((primero->getInfo() != x) && (primero->getSig())){
94             Nodo<T> *aux1 = primero, *aux2 = primero->getSig();
95             while((aux2->getInfo() != x) && (aux2->getSig())){
96                 aux1 = aux2;
97                 aux2 = aux2->getSig();
98             }
99             if(aux2->getInfo() == x){
100                 aux1->setSig(aux2->getSig());
101                 delete aux2;
102             }
103         }
104         else if(primero->getInfo() == x){
105             Nodo<T> *aux = primero;
106             primero = primero->getSig();
107             delete aux;
108         }
109     }
110 }
111
112 template <typename T>
113 void Lista<T>::bajaprin(){
114     if(primeros){
115         Nodo<T> *aux = primero;
116         primero = primero->getSig();
117         delete aux;
118     }
119 }
120
121 template<typename T>
122 bool Lista<T>::busca(T x){
123     Nodo<T> *aux = primero;
124
125     while(aux){
126         if(aux->getInfo() == x)
127             return true;
128         aux = aux->getSig();
129     }
130     return false;

```

```

131 }
132
133 template <typename T>
134 T Lista<T>::primerElemento(){
135     return primero->getInfo();
136 }
137
138
139 template<typename T>
140 void Lista<T>::emite() const{
141     Nodo<T> *aux = primero;
142
143     while(aux){
144         std::cout<<aux->getInfo();
145         aux = aux->getSig();
146     }
147 }
148
149 template<typename T>
150 int Lista<T>::conteoNodos()const{
151     Nodo<T> *aux = primero;
152     int cont = 0;
153
154     while(aux){
155         cont++;
156         aux = aux->getSig();
157     }
158     return cont;
159 }
160
161 template<typename T>
162 int Lista<T>::valoresPositivos() const{
163     Nodo<T> *aux = primero;
164
165     if(!aux)
166         return 0;
167
168     while((aux) && (aux->getInfo() > 0))
169         aux = aux->getSig();
170
171     if(aux)
172         return 0;
173
174     return 1;
175 }
176
177 template<typename T>
178 Lista<T> *Lista<T>::inversion(){
179     Nodo<T> *nodo_aux = primero;
180
181     if(!nodo_aux || (!nodo_aux->getSig()))
182         return this;
183
184     Lista<T> *lista_aux;
185
186     while(nodo_aux){
187         lista_aux->altaprin(nodo_aux->getInfo());
188         nodo_aux = nodo_aux->getSig();
189     }
190
191     return lista_aux;
192 }
193
194 template<typename T>
195 Lista<T> Lista<T>::interseccionListas(Lista<T> l){
196     Nodo<T> *nodo_aux1 = primero, *nodo_aux2 = l.primero;

```

```
197
198     if(!nodo_aux2)
199         return 1;
200     if(!nodo_aux1)
201         return *this;
202
203     Lista<T> lista_aux;
204
205     while(nodo_aux2){
206         nodo_aux1 = primero;
207         while(((nodo_aux1->getInfo()) != (nodo_aux2->getInfo())) && nodo_aux1)
208             nodo_aux1 = nodo_aux1->getSig();
209
210         if(nodo_aux1)
211             lista_aux.atafin(nodo_aux1->getInfo());
212
213         nodo_aux2 = nodo_aux2->getSig();
214     }
215
216     return lista_aux;
217 }
218
219 #endif // CLASE_LISTA_H_INCLUDED
```

Template_Pila.h

```
1  #ifndef CLASE_PILA_H_INCLUDED
2  #define CLASE_PILA_H_INCLUDED
3
4  #include "Template_clase_lista.h"
5
6  template <typename T>
7  class Pila{
8  private:
9      Lista<T> *p_entrada;
10
11 public:
12     Pila(); //Constructor por defecto.
13     ~Pila(); //Destructor por defecto.
14
15     Pila<T>& push(T); //Hace un alta al principio de la lista.
16     Pila<T>& pop(); //Hace una baja al principio de la lista.
17     T tope(); //Retorna el valor del tope de la pila (primer elemento de la lista).
18     bool vacia(); //Verifica si la pila esta vacia.
19 };
20
21
22 template <typename T>
23 Pila<T>::Pila(){
24     p_entrada = new Lista<T>;
25 }
26
27 template <typename T>
28 Pila<T>::~~Pila(){
29     if(p_entrada){
30         p_entrada->~Lista();
31         delete p_entrada;
32         p_entrada = NULL;
33     }
34 }
35
36 template <typename T>
37 Pila<T>& Pila<T>::push(T dato){
38     if(p_entrada)
39         p_entrada->altaprin(dato);
40     return *this;
41 }
42
43 template <typename T>
44 Pila<T>& Pila<T>::pop(){
45     if(p_entrada)
46         p_entrada->bajaprin();
47     return *this;
48 }
49
50 template <typename T>
51 T Pila<T>::tope(){
52     return p_entrada->primerElemento();
53 }
54
55 template <typename T>
56 bool Pila<T>::vacia(){
57     if((!p_entrada) || p_entrada->conteoNodos() == 0)
58         return true;
59     return false;
60 }
61
62
63 #endif // CLASE_PILA_H_INCLUDED
```

Queue.h

```
1  #ifndef QUEUE_H_INCLUDED
2  #define QUEUE_H_INCLUDED
3
4  //Clase Queue implementado con una lista circular
5  //ejemplo sencillo de listas circulares
6  //este programa solo intenta mostrar el comportamiento de los punteros en las listas circulares
7
8  #include <iostream>
9
10 #include "Template_clase_nodo.h"
11
12 using namespace std;
13
14 template <class T> class Queue ;
15
16
17 template <typename T>
18 class Queue
19 {
20
21     private:
22         Nodo <T> * first;
23         Nodo <T> * last;
24     public:
25         Queue( ); //constructor
26                 // Escribir el constructor copia como ej
27         ~Queue( ); //destructor
28
29         void enqueue(const T &); // Da alta en la cola a un elemento.
30         void dequeue( ); // Elimina el elemento del frente de la misma
31         bool empty( ) const; //retorna true si lista vacia.
32         void Emitir ( ) const; //emite la lista.
33         T front( ) const; // Retorna el valor del primer elemento de la cola.
34 };
35
36 template <typename T>
37 Queue <T>::Queue(){
38     first = last = 0;
39 }
40
41 template <typename T>
42 Queue<T>::~~Queue()
43 {
44     if(first)
45     {
46         if (first->getSig() == first)delete first;
47         else
48         {
49             Nodo <T> *_aux1, *_aux2;
50             _aux1=first->getSig();
51             first->setSig(0);
52             while(_aux1!=0)
53             {
54                 _aux2=_aux1;
55                 _aux1=_aux1->getSig();
56                 delete _aux2;
57             }
58         }
59     }
60 }
61
62 template <typename T>
63 void Queue <T> :: enqueue (const T & _v)
64 {
65     Nodo<T> *_nuevo = new Nodo <T>(_v);
66     if(empty())
```

```

67         first = _nuevo;
68         last = _nuevo;
69         _nuevo->setSig(_nuevo);
70     }
71     else
72     {
73         last->setSig(_nuevo);
74         _nuevo->setSig(first);
75         last = _nuevo;
76     }
77 }
78
79 template <typename T>
80 void Queue <T> :: deQueue()
81 {
82     Nodo<T> *_aux1;
83     if (!empty())
84     {
85         if (first->getSig() == first)
86         {
87             delete first;
88             first = 0;
89             last = 0;
90         }
91         else
92         {
93             _aux1=first;
94             first = first->getSig();
95             //last->getSig() = first;
96             last->setSig(first);
97             delete _aux1;
98         }
99     }
100 }
101
102 template <typename T>
103 void Queue <T> :: Emitir () const
104 {
105     Nodo<T> *_aux=first;
106     if(!empty())
107     {   do
108         {   cout<<_aux->getInfo()<<"          ";
109             _aux=_aux->getSig();
110         }
111         while(_aux!=first);
112     }
113 }
114
115 template <typename T>
116 bool Queue <T> :: empty () const
117 {
118     return (first==0);
119 }
120
121 template <typename T>
122 T Queue <T> :: front() const
123 {
124     return first->getInfo(); //ojo lo cambie de front a first
125 }
126
127 #endif

```

Complejo.h

```
1  #ifndef COMPLEJO_H_INCLUDED
2  #define COMPLEJO_H_INCLUDED
3
4  #include "constantes.h"
5  #include "Errores.h"
6  #include <string>
7
8  using namespace std;
9
10 class Complejo{
11 private:
12     double real;
13     double imaginario;
14     Complejo pow2();
15 public:
16     Complejo(); //Constructor por defecto.
17     Complejo(const Complejo &); //Constructor por copia.
18     Complejo(double, double); //Constructor por valores.
19     Complejo(double); //Constructor por un valor (real). Se usa para castear.
20     Complejo(string); //Constructor por cadena.
21     ~Complejo(); //Destructor
22
23     void setReal(double);
24     void setImag(double);
25     double getReal();
26     double getImag();
27
28     string aString(); //Convierte un complejo a un string.
29     Complejo aplicarFuncion(string);
30     Complejo aplicarOperador(Complejo &, string);
31
32     Complejo conjugado();
33     Complejo identidad(); // Aplica la identidad y retorna el resultado.
34     Complejo expc(); // Aplica la exponencial y retorna el resultado.
35     Complejo sinc(); //Aplica el seno y retorna el resultado.
36     Complejo cosc(); //Aplica el coseno y retorna el resultado.
37     Complejo lnc(); //Aplica el logaritmo natural y retorna el resultado.
38     double modulo();
39     double phase();
40
41     Complejo operator+(Complejo);
42     Complejo operator-(Complejo);
43     Complejo operator-();
44     Complejo operator*(Complejo);
45     Complejo operator/(Complejo);
46     Complejo operator^(int);
47     Complejo &operator=(const Complejo &);
48     bool operator==(Complejo &);
49     operator int(); //Castea a int un objeto de la clase Complejo.
50 };
51
52 #endif // COMPLEJO_H_INCLUDED
```


Complejo.cpp

```
1  #include "Complejo.h"
2  #include <cmath>
3
4  Complejo::Complejo(){
5      real = 0.0;
6      imaginario = 0.0;
7  }
8
9  Complejo::Complejo(const Complejo &c){
10     real = c.real;
11     imaginario = c.imaginario;
12 }
13
14 Complejo::Complejo(double re, double im){
15     real = re;
16     imaginario = im;
17 }
18
19 Complejo::Complejo(double re){
20     real = re;
21     imaginario = 0.0;
22 }
23
24 Complejo::Complejo(string str){
25     int i = 0;
26     string aux="";
27     string aux_imaginario;
28     string aux_real;
29     double num;
30
31     if(str == VARIABLE_PI || str == VARIABLE_PI_NEGADO || str == "+pi")
32     {
33         if (str == VARIABLE_PI_NEGADO)
34             real = -M_PI;
35         else{
36             real = M_PI;
37         }
38         imaginario = 0.0;
39
40     }else if (str.back() == NUMERO_IMAGINARIO[0] && str[str.size()-2] != VARIABLE_PI[0]){
41         if(str[0] == OPERADOR_SUSTRACCION[0]) // Entra sólo si str = -i
42         {
43             real = 0.0;
44             imaginario = 0.1;
45
46         }else{
47             num = stod(str); // Convertimos el número a un double.
48             // Generamos el número complejo
49             real = 0.0;
50             imaginario = num;
51         }
52     }else{
53
54         while ( str[i] != '\0')
55         {
56             aux = aux+str[i];
57             if (str[i+1] == ' ' || str[i+1] == '\0')
58             {
59                 aux_real = aux;
60                 aux.clear(); // Limpio la variable aux
61                 i++;
62                 while ( str[i] != '\0')
63                 {
64                     if(str[i] == ' ')
65                         i++;
66                     aux = aux+str[i];
```

```

67         i++;
68     }
69     aux_imaginario = aux;
70     i--;
71 }
72 i++;
73 }
74
75 // una vez obtengo aux_real y aux_imaginario
76
77 if(aux_real == VARIABLE_PI || aux_real == VARIABLE_PI_NEGADO || aux_imaginario == VARIABLE_PI ||
aux_imaginario == VARIABLE_PI_NEGADO)
78 {
79     if (aux_real == VARIABLE_PI_NEGADO)
80         real = -M_PI;
81     else if (aux_real == VARIABLE_PI)
82         real = M_PI;
83     if (aux_imaginario == VARIABLE_PI_NEGADO)
84         imaginario = -M_PI;
85     else if (aux_imaginario == VARIABLE_PI)
86         imaginario = M_PI;
87 }
88 else{
89     num = stod(aux_real);          // Convertimos el número a un double.
90     real = num;
91     if(aux_imaginario[0] == '\0')
92     {
93         imaginario = 0.0;
94     }else{
95         num = stod(aux_imaginario);
96         imaginario = num;
97     }
98 }
99 }
100 }
101
102
103
104 Complejo::~Complejo(){
105 }
106
107 void Complejo::setReal(double r){
108     real = r;
109 }
110
111 void Complejo::setImag(double i){
112     imaginario = i;
113 }
114
115 double Complejo::getReal(){
116     return real;
117 }
118
119 double Complejo::getImag(){
120     return imaginario;
121 }
122
123
124
125 string Complejo::asString()
126 {
127     string aux;
128     string aux_real = "", aux_imaginario="";
129
130     if( (fabs(real) == M_PI || fabs(imaginario) == M_PI))
131     {

```

```

132     if (real == M_PI)
133         aux_real = VARIABLE_PI;
134     else if (real == -M_PI)
135         aux_real = VARIABLE_PI_NEGADO;
136     else
137         aux_real = to_string(real);
138     if (imaginario == M_PI)
139         aux_imaginario = VARIABLE_PI;
140     else if(imaginario == -M_PI)
141         aux_imaginario = VARIABLE_PI_NEGADO;
142     else
143         aux_imaginario = to_string(imaginario);
144     aux = aux_real+' '+aux_imaginario;
145 }else
146     aux= to_string(real)+' '+to_string(imaginario);
147 return aux;
148 }
149
150 Complejo Complejo::aplicarFuncion(string fun){ //Aplica la funcion a this y devuelve otro Complejo (modulo
y fase devuelven otro Complejo!=this). Si no es una funcion, devuelve this.
151     Complejo aux;
152     if(fun == FUNCION_EXPONENCIAL){
153         aux = (this->exp());
154         return aux;
155     }
156     else if(fun == FUNCION_SENO){
157         aux = (this->sinc());
158         return aux;
159     }
160     else if(fun == FUNCION_COSENO){
161         aux = (this->cosc());
162         return aux;
163     }
164     else if(fun == FUNCION_LOGARITMO){
165         aux = (this->lnc());
166         return aux;
167     }else if(fun == FUNCION_REAL){
168         aux = (Complejo)(this->getReal());
169         return aux;
170     }else if(fun == FUNCION_IMAGINARIA){
171         aux = (Complejo)(this->getImag());
172         return aux;
173     }
174     else if(fun == FUNCION_ABS){
175         aux = (Complejo)(this->modulo());
176         return aux;
177     }
178     else if(fun == FUNCION_PHASE){
179         aux = (Complejo)(this->phase());
180         return aux;
181     }
182     else
183         return (*this);
184 }
185
186 Complejo Complejo::aplicarOperador(Complejo &c, string op){ //Aplica el operador entre this y c, devuelve
this modificado. Si no es ningun operador, devuelve this.
187     Complejo aux;
188     if(op == OPERADOR_SUMA){
189         aux = ((*this)+c);
190         return aux;
191     }
192     else if(op == OPERADOR_SUSTRACCION){
193         aux = ((*this)-c);
194         return aux;
195     }

```

```

196     else if(op == OPERADOR_MULTIPLICACION){
197         aux = ((*this)*c);
198         return aux;
199     }
200     else if(op == OPERADOR_DIVISION){
201         aux = ((*this)/c);
202         return aux;
203     }
204     else if(op == OPERADOR_POTENCIA){
205         aux = (*this)^(int)c;
206         return aux;
207     }
208     else
209         return (*this);
210 }
211
212
213
214 Complejo Complejo::conjugado(){
215     Complejo aux;
216     aux.real = this->real;
217     aux.imaginario = (this->imaginario * (-1));
218     return aux;
219 }
220
221 Complejo Complejo::exp(){
222 {
223     //cout<<"exp"<<endl;
224     Complejo aux(0.0, 0.0);
225     aux.real = exp(real)*cos(imaginario);
226     aux.imaginario = exp(real)*sin(imaginario);
227     //cout<<"esp: sin (imaginario) "<<sin(imaginario)<<endl;
228     //cout<<"esp: cos (imaginario) "<<cos(imaginario)<<endl;
229     return aux;
230 }
231
232 Complejo Complejo::identidad(){
233     Complejo aux(0.0, 0.0);
234     aux.real = real;
235     aux.imaginario = imaginario;
236     return aux;
237 }
238
239 Complejo Complejo::sinc(){
240     Complejo aux;
241     aux.real = (sin(this->real)*cosh(this->imaginario));
242     aux.imaginario = (cos(this->real)*sinh(this->imaginario));
243     return aux;
244 }
245
246 Complejo Complejo::csc(){
247     Complejo aux;
248     aux.real = (cos(this->real)*cosh(this->imaginario));
249     aux.imaginario = -(sin(this->real)*sinh(this->imaginario));
250     return aux;
251 }
252
253 Complejo Complejo::lnc(){
254     Complejo aux;
255     aux.real = log(this->modulo());
256     aux.imaginario = this->phase();
257     return aux;
258 }
259
260
261 double Complejo::modulo(){

```

```

262     double aux = real*real + imaginario*imaginario;
263     return sqrt(aux);
264 }
265
266 double Complejo::phase(){
267     if(real == 0 && imaginario == 0)
268         return 0;
269     if(real < 0){
270         if(imaginario > 0)
271             return (M_PI + (atan(imaginario/real)));
272         if(imaginario < 0)
273             return (-M_PI + (atan(imaginario/real)));
274         return M_PI;
275     }
276     return atan(imaginario/real);
277 }
278
279
280 Complejo Complejo::operator+(Complejo c){
281     Complejo aux;
282     aux.real = (this->real + c.real);
283     aux.imaginario = (this->imaginario + c.imaginario);
284     return aux;
285 }
286
287 Complejo Complejo::operator-(Complejo c){
288     Complejo aux;
289     aux.real = (this->real - c.real);
290     aux.imaginario = (this->imaginario - c.imaginario);
291     return aux;
292 }
293
294 Complejo Complejo::operator-(){
295     Complejo aux;
296     aux.real = -(this->real);
297     aux.imaginario = -(this->imaginario);
298     return aux;
299 }
300
301 Complejo Complejo::operator*(Complejo c){
302     Complejo aux;
303     aux.real = ((this->real * c.real) - (this->imaginario * c.imaginario));
304     aux.imaginario = ((this->real * c.imaginario) + (this->imaginario * c.real));
305     return aux;
306 }
307
308 Complejo Complejo::operator/(Complejo c){
309     Complejo aux, aux2;
310     aux2 = (*this * (c.conjugado()));
311     aux = (c * (c.conjugado()));
312     aux.imaginario = (aux2.imaginario / aux.real);
313     aux.real = (aux2.real / aux.real);
314     return aux;
315 }
316
317 Complejo Complejo::operator^(int i){
318     if(i<0)
319         return ((Complejo)1/((this)^(-i)));
320     if(i == 0)
321         return Complejo(1, 0);
322     Complejo aux;
323     if(i/2 < 1){
324         aux = *this;
325         return aux;
326     }
327     int aux2 = i/2;

```

```

328     aux = (this->pow2());
329     for(int j=1; j<aux2; j++)
330         aux = aux * (this->pow2());
331     if(i%2 == 1)
332         return aux*(*this);
333     return aux;
334 }
335
336 Complejo &Complejo::operator=(const Complejo &c){
337     real = c.real;
338     imaginario = c.imaginario;
339     return *this;
340 }
341
342 bool Complejo::operator==(Complejo &c){
343     if(real == c.getReal()){
344         if(imaginario == c.getImag())
345             return true;
346     }
347     return false;
348 }
349
350 Complejo::operator int(){
351     return (int)real;
352 }
353
354
355
356 Complejo Complejo::pow2(){
357     Complejo aux(0.0, 0.0);
358     aux.real = real*real-imaginario*imaginario;
359     aux.imaginario = 2*real*imaginario;
360     return aux;
361 }

```

Pixel.h

```
1  #ifndef PIXEL_H_INCLUDED
2  #define PIXEL_H_INCLUDED
3
4  #include "Complejo.h"
5
6  #define REAL_DEFECTO 0.0
7  #define IMAGINARIO_DEFECTO 0.0
8  #define COLOR_DEFECTO 0
9
10 class Pixel{
11 private:
12     Complejo posicion; //Posicion del pixel en el mapa.
13     int color; //Color del pixel (Escala de grises).
14
15 public:
16     Pixel(); //Constructor por defecto.
17     ~Pixel(); //Destructor.
18
19     void setPosition(const Complejo &); //Setea la posicion del pixel en el mapa.
20     void setColor(const int &); //Setea el color del pixel.
21     Complejo &getPosition(); //Devuelve la posicion del Pixel.
22     int &getColor(); //Devuelve el color del pixel (Escala de grises).
23
24     Pixel &operator=(Pixel &); // operador =.
25 };
26
27
28
29
30 #endif // PIXEL_H_INCLUDED
```

Pixel.cpp

```
1  #include "Pixel.h"
2
3  //Constructor por defecto. Crea un pixel negro en el origen.
4  Pixel::Pixel(){
5      posicion.setReal(REAL_DEFECTO);
6      posicion.setImag(IMAGINARIO_DEFECTO);
7      color = COLOR_DEFECTO;
8  }
9
10 Pixel::~Pixel(){
11 }
12
13 void Pixel::setPosition(const Complejo &c){
14     posicion = c;
15 }
16
17 void Pixel::setColor(const int &colour){
18     color = colour;
19 }
20
21 Complejo &Pixel::getPosition(){
22     return posicion;
23 }
24
25 int &Pixel::getColor(){
26     return color;
27 }
28
29 Pixel &::Pixel::operator=(Pixel &p){
30     posicion = p.getPosition();
31     color = p.getColor();
32     return *this;
33 }
```


Imagen.h

```
1  #ifndef IMAGEN_H_INCLUDED
2  #define IMAGEN_H_INCLUDED
3
4  #include <iostream>
5  #include <fstream>
6  #include <cmath>
7  #include <string.h>
8  #include "Pixel.h"
9  #include "Errores.h"
10 #include "Complejo.h"
11 #include "constantes.h"
12
13 #include "Queue.h"
14 #include "Template_Pila.h"
15
16 #define ANCHO_POR_DEFECTO 3
17 #define ALTO_POR_DEFECTO 3
18 #define INTENSIDAD_MAXIMA_POR_DEFECTO 0
19
20 using namespace std;
21
22
23 /* Se define la clase Imagen que se encarga de guardar los datos de una imagen
24  *
25  */
26 class Imagen{
27 private:
28     int ancho; //Ancho de la imagen (Cantidad de pixeles).
29     int alto; //Alto de la imagen (Cantidad de pixeles).
30     int intensidad_max; //Intensidad maxima de la escala de grises.
31     double precision_ancho; // Precisión para obtener la imagen trasformada.
32     double precision_alto;
33
34     Pixel **mapa; //Puntero al primer elemento de la matriz de pixels.
35
36 public:
37     Imagen(); //Constructor por defecto.
38     Imagen(int, int, int); //Constructor por parametros.
39     Imagen(istream&); //Constructor por archivo.
40     ~Imagen(); //Destructor.
41
42     void setAncho(const int&); //Carga en el atributo ancho.
43     void setAlto(const int&); //Carga en el atributo alto.
44     void setIntensidadMax(const int&); //Carga en el atributo intensidad_max.
45     void setPrecisionAncho(const double&); //Carga en el atributo precision_ancho.
46     void setPrecisionAlto(const double&); //Carga en el atributo precision_alto.
47     int &getAncho(); //Retorna el atributo ancho.
48     int &getAlto(); //Retorna el atributo alto.
49     int &getIntensidadMax(); //Retorna el atributo intensidad_max.
50     double &getPrecisionAncho(); //Retorna el atributo precision_ancho.
51     double &getPrecisionAlto(); //Retorna el atributo precision_alto.
52     Pixel &getPixel(int, int); //Recibe como argumento la posicion (i,j) de la matriz y retorna el objeto
pixel que se encuentra ahí.
53     int getPixelColor(Complejo &); //Recibe un complejo y retorna la intensidad máxima.
54     int getPixelColor(double&);
55
56     void operar(Imagen&, Queue<string>*);
57
58     void print(ostream&); // Método que imprime la imagen por el flujo de salida.
59
60     Imagen &operator=(Imagen &); //operador =.
61 };
62
63 #endif // IMAGEN_H_INCLUDED
```

Imagen.cpp

```
1  #include "Imagen.h"
2  #include <iostream>
3  #include <string>
4  #include <sstream>
5  #include <cmath>
6
7  using namespace std;
8
9  Imagen::Imagen(){ //Constructor. Crea una imagen de 3x3 en negro.
10
11     ancho = ANCHO_POR_DEFECTO;
12     alto = ALTO_POR_DEFECTO;
13     intensidad_max = INTENSIDAD_MAXIMA_POR_DEFECTO;
14
15     //Pide memoria para la matriz de pixels.
16     mapa = new Pixel *[alto];
17     for(int i=0; i<alto; i++)
18         mapa[i] = new Pixel[ancho];
19
20     //Coloca el (0,0) en el medio
21     for(int i=0, j=(alto/2); i<alto; i++, j--){
22         for(int k=0, l=(-ancho)/2; k<ancho; k++, l++){
23             Complejo c((double)l/(ancho/2), (double)j/(alto/2));
24             mapa[i][k].setPosition(c);
25         }
26     }
27 }
28
29 Imagen::Imagen(int an, int al, int in){
30
31     int alto_aux;
32     int ancho_aux;
33
34     ancho = an;
35     alto = al;
36     intensidad_max = in;
37
38     //Calculamos y asignamos la precision con la que se formara la imagen de salida
39     precision_ancho = 2/((double)ancho);
40     precision_alto = 2/((double)alto);
41
42     //Pide memoria para la matriz de Pixels validando que alto y ancho sean impares.
43     if((alto%2) == 0)
44         alto_aux = alto + 1;
45     else
46         alto_aux = alto;
47     if((ancho%2) == 0)
48         ancho_aux = ancho + 1;
49     else
50         ancho_aux = ancho;
51     mapa = new Pixel *[alto_aux];
52     for(int i=0; i<alto_aux; i++)
53         mapa[i] = new Pixel[ancho_aux];
54
55     //Coloca el (0,0) en el medio
56     for(int i=0, j=(alto_aux/2); i<alto_aux; i++, j--){
57         for(int k=0, l=(-ancho_aux)/2; k<ancho_aux; k++, l++){
58             Complejo c((double)l/(ancho_aux/2), (double)j/(alto_aux/2));
59             mapa[i][k].setPosition(c);
60         }
61     }
62
63     //Actualizamos los valores de ancho y alto
64     ancho = ancho_aux;
65     alto = alto_aux;
66 }
```

```

67
68 Imagen::Imagen(istream& archivoEntrada)
69 {
70
71     int pixel;
72     int alto_aux;
73     int ancho_aux;
74
75     string lineaEntrada = "";
76     stringstream ss; // Variable Stream para almacenar en un bufer.
77
78     if (!getline(archivoEntrada, lineaEntrada)) // Leo la primera línea
79         Errores::noSePuedeLeerArchivo(); // En caso de que no se pudo leer
correctamente.
80
81     if(lineaEntrada.compare(FORMATO_IMAGEN) != 0){ //En caso de que el archivo de entradas sea distinto a
.pgm
82         Errores::archivoIncorrecto(); //retornamos un error y terminamos la ejecución del
programa.
83     }
84
85     if(!getline(archivoEntrada, lineaEntrada)) // Lee la segunda línea
86         Errores::noSePuedeLeerArchivo();
87
88     if(lineaEntrada[0] != CARACTER_COMENTARIO){
89         size_t pos_espacio = lineaEntrada.find(DELIMITADOR); //Guardo la posicion donde esta el
DELIMITADOR.
90         string aux;
91         aux = lineaEntrada.substr(0, pos_espacio); //En el string aux guardo lo que hay hasta el
DELIMITADOR.
92
93         if(!(stringstream(aux) >> ancho)) //En la variable width guardo el string aux convertido en int.
94             Errores::noSePuedeObtenerAncho();
95         aux = lineaEntrada.substr(pos_espacio+1); //En el string aux guardo lo que hay desde el caracter
que
96
97         //le sigue al espacio hasta el final del string s.
98         if(!(stringstream(aux) >> alto)) //En el atributo alto guardo el string aux convertido en int.
99             Errores::noSePuedeObtenerLargo();
100
101         if(!(ss << archivoEntrada.rdbuf())) //Almacenamos en un bufer los datos que quedan en el archivo.
102             Errores::noSePuedeLeerArchivo();
103     }
104     else{
105         if(!(ss << archivoEntrada.rdbuf())) // Almacenamos en un bufer los datos que quedan en el
archivo.
106             Errores::noSePuedeLeerArchivo();
107         //Inserto los datos de ancho y largo de la imagen
108         if(!(ss >> ancho))
109             Errores::noSePuedeObtenerAncho();
110         if(!(ss >> alto))
111             Errores::noSePuedeObtenerLargo();
112     }
113
114     //Inserto la intensidad máxima de color
115
116     if(!(ss >> intensidad_max))
117     {
118         Errores::noSePuedeObtenerIntensidadMaxima();
119     }
120
121     //Calculamos y asignamos la precision con la que se formara la imagen de salida
122
123     precision_ancho = 2/((double)ancho);
124     precision_alto = 2/((double)alto);
125

```

```

126 //Pide memoria para la matriz de Pixels validando que alto y ancho sean impares.
127 if((alto%2) == 0)
128     alto_aux = alto + 1;
129 else
130     alto_aux = alto;
131 if((ancho%2) == 0)
132     ancho_aux = ancho + 1;
133 else
134     ancho_aux = ancho;
135 mapa = new Pixel *[alto_aux];
136 for(int i=0; i<alto_aux; i++)
137     mapa[i] = new Pixel[ancho_aux];
138
139 //Coloca el (0,0) en el medio
140
141 for(int i=0, j=(alto_aux/2); i<alto_aux; i++, j--){
142     for(int k=0, l=(-ancho_aux)/2; k<ancho_aux; k++, l++){
143         Complejo c((double)l/(ancho_aux/2), (double)j/(alto_aux/2));
144         mapa[i][k].setPosition(c);
145     }
146 }
147
148 for(int i=0; i<alto; i++){
149     for(int j=0; j<ancho; j++){
150         if(!(ss>>pixel))
151             Errores::noSePuedeObtenerIntensidad();
152         mapa[i][j].setColor(pixel);
153     }
154 }
155
156 // Actualizamos los valores de ancho y alto
157 ancho = ancho_aux;
158 alto = alto_aux;
159 }
160
161
162 Imagen::~Imagen(){
163     if(mapa){
164         for(int i=0; i<alto; i++)
165             delete[] mapa[i];
166         delete[] mapa;
167         mapa = NULL;
168     }
169 }
170
171 void Imagen::setAncho(const int &an){
172     ancho = an;
173 }
174
175 void Imagen::setAlto(const int &al){
176     alto = al;
177 }
178
179 void Imagen::setIntensidadMax(const int &in){
180     intensidad_max = in;
181 }
182
183 void Imagen::setPrecisionAncho(const double &prec){
184     precision_ancho= prec;
185 }
186
187 void Imagen::setPrecisionAlto(const double &prec){
188     precision_alto= prec;
189 }
190
191 int &Imagen::getAncho(){

```

```

192     return ancho;
193 }
194
195 int &Imagen::getAlto(){
196     return alto;
197 }
198
199 int &Imagen::getIntensidadMax(){
200     return intensidad_max;
201 }
202
203 double &Imagen::getPrecisionAncho(){
204     return precision_ancho;
205 }
206
207 double &Imagen::getPrecisionAlto(){
208     return precision_alto;
209 }
210
211 Pixel &Imagen::getPixel(int i, int j){
212     if(i<0 || i>alto || j<0 || j>ancho){
213         return mapa[0][0];
214     }
215     return mapa[i][j];
216 }
217
218 int Imagen::getPixelColor(Complejo &c){
219
220     if(fabs(c.getReal()) > 1 || fabs(c.getImag()) > 1) // Si el complejo no pertenece al cuadro de 2*2 se
retorna
221     {
222         // el color negro.
223
224         return 0;
225     }
226
227     for(int i=0; i<alto; i++){
228         for(int j=0; j<ancho; j++){
229
230             Complejo aux = c-(mapa[i][j].getPosition());
231
232             if(fabs(aux.getReal()) < precision_ancho){// || fabs(aux.getReal()) == precision_ancho){
233                 for(int k=i; k<alto; k++){
234                     Complejo aux = c-(mapa[k][j].getPosition());
235                     if(fabs(aux.getImag())< precision_alto){// || fabs(aux.getImag()) == precision_alto){
236                         return mapa[k][j].getColor();
237                     }
238                 }
239             }
240         }
241     }
242     return 0;
243 }
244 //Función que imprime la imagen
245 void Imagen::print(ostream& oss)
246 {
247     if(!(oss<<FORMATO_IMAGEN)) //Imprime el formato de la imagen .pgm
248         Errores::noSePuedeImprimirFormato();
249     oss<<endl;
250
251     if(!(oss<<ancho))
252         Errores::noSePuedeImprimirAncho();
253     if(!(oss<<DELIMITADOR)) // Imprime el delimitador
254         Errores::noSePuedeImprimirLargo();
255     if(!(oss<<alto))
256         Errores::noSePuedeImprimirLargo();

```

```

257     oss<<endl;
258
259     if(!(oss<<intensidad_max)) //Imprime intensidad máxima
260         Errores::noSePuedeImprimirIntensidadMaxima();
261     oss<<endl;
262     //Lo siguiente imprime las intensidades de color
263     for(int i=0; i<alto; i++){
264
265         for(int j=0; j<ancho; j++){
266             if(!(oss<<mapa[i][j].getColor()))
267                 Errores::noSePuedeImprimirIntensidad();
268             if(!(oss<<DELIMITADOR))
269                 Errores::noSePuedeImprimirDelimitador();
270             if(!(oss<<DELIMITADOR))
271                 Errores::noSePuedeImprimirDelimitador();
272         }
273         oss<<endl;
274     }
275 }
276
277 //Asigna pixel a pixel una imagen a otra siendo ambas del mismo alto, alto e intensidad maxima.
278 Imagen &Imagen::operator=(Imagen &im){
279
280     if(alto != im.getAlto() || ancho != im.getAncho() || intensidad_max != im.getIntensidadMax()){
281         Errores::imagenesDeDistintoTamano();
282     }
283
284     //Copia pixel a pixel.
285     for(int i=0; i<im.getAlto(); i++){
286         for(int j=0; j<im.getAncho(); j++){
287             mapa[i][j] = im.getPixel(i, j);
288         }
289     }
290     return *this;
291 }
292
293 /*Método operar*/
294 void Imagen::operar(Imagen& img, Queue<string> *cola)
295 {
296     img = *this;
297
298     Pila<string> pila;
299     string simbolo;
300     string aux1, aux2;
301
302     while(!(*cola).empty())
303     {
304
305         simbolo = (*cola).front();
306         (*cola).deQueue();
307         if(isdigit(simbolo[0]))
308         {
309             pila.push(simbolo);
310
311         }else if (isalpha(simbolo[0]))
312         {
313             if(simbolo == VARIABLE_PI)
314             {
315                 pila.push(simbolo);
316             }
317             else if(simbolo == VARIABLE_INDEPENDIENTE )
318             {
319                 if((*cola).empty())
320                     break;
321                 else
322                     pila.push(simbolo);

```

```

323         }else if(simbolo == FUNCION_EXPONENCIAL || simbolo == FUNCION_LOGARITMO || simbolo == FUNCION_REAL
|| simbolo == FUNCION_IMAGINARIA || simbolo == FUNCION_ABS
324             || simbolo == FUNCION_PHASE || simbolo == FUNCION_SENO || simbolo == FUNCION_COSENO){
325             aux1 = pila.tope();
326             pila.pop();
327             if(aux1 == VARIABLE_INDEPENDIENTE)
328             {
329                 for (int fila = 0; fila < alto; fila++){
330                     for (int col = 0; col < ancho; col++){
331                         img.getPixel(fila,col).getPosition() = img.getPixel(fila,col).getPosition().
aplicarFuncion(simbolo);
332                     }
333                 }
334
335                 pila.push(VARIABLE_INDEPENDIENTE);
336
337             }else{
338                 Complejo comp1(aux1); // Generamos un complejo
339                 comp1 = comp1.aplicarFuncion(simbolo);
340                 aux1 = comp1.aString(); // Convertimos el el complejo en string
341                 pila.push(aux1); // Calculamos y luego cargamos a la pila,FALTA IMPLEMENTAR
EN LA CLASE COMPLEJO
342             }
343         }
344         }else if ( (simbolo == OPERADOR_SUMA || simbolo == OPERADOR_SUSTRACCION || simbolo ==
OPERADOR_MULTIPLICACION || simbolo == OPERADOR_DIVISION || simbolo == OPERADOR_POTENCIA) && simbolo.size() == 1 ) {
345
346             aux1 = pila.tope();
347             pila.pop();
348
349             aux2 = pila.tope();
350             pila.pop();
351
352             if(aux1 == VARIABLE_INDEPENDIENTE) // Si encuentro una z, aplico una
transformacion a la imagen.
353             {
354                 Complejo comp2(aux2);
355
356                 for (int fila = 0; fila < alto; fila++){
357                     for (int col = 0; col < ancho; col++){
358
359                         img.getPixel(fila,col).getPosition() = img.getPixel(fila,col).getPosition().
aplicarOperador(comp2,simbolo);
360                     }
361                 }
362
363                 pila.push(VARIABLE_INDEPENDIENTE);
364
365             }else if(aux2 == VARIABLE_INDEPENDIENTE){
366
367                 Complejo comp1(aux1);
368
369                 for (int fila = 0; fila < alto; fila++){
370                     for (int col = 0; col < ancho; col++){
371
372                         img.getPixel(fila,col).getPosition() = img.getPixel(fila,col).getPosition().
aplicarOperador(comp1,simbolo);
373                     }
374                 }
375
376                 pila.push(VARIABLE_INDEPENDIENTE);
377             }else{
378                 Complejo comp1(aux1);
379                 Complejo comp2(aux2);
380
381                 comp2 = comp2.aplicarOperador(comp1,simbolo);

```

```

382         pila.push(comp2.aString());
383     }
384
385     }else if ( (simbolo[0] ==OPERADOR_SUSTRACCION[0] && simbolo.size() != 1)){
386
387         if(simbolo.back() == VARIABLE_INDEPENDIENTE[0]){
388             for (int fila = 0; fila < alto; fila++){
389                 for (int col = 0; col < ancho; col++){
390
391                     img.getPixel(fila,col).getPosition() = -(img.getPixel(fila,col).getPosition());
392                 }
393             }
394
395             pila.push(VARIABLE_INDEPENDIENTE);
396         }else{
397             pila.push(simbolo);           // Cargamos el numero complejo a la pila.
398         }
399     }
400 }//cierro while
401 for (int fila = 0; fila < alto; fila++){
402     for (int col = 0; col < ancho; col++){
403
404         img.getPixel(fila,col).setColor(this->getPixelColor(img.getPixel(fila,col).getPosition()));
405     }
406 }
407 }

```


Errores.h

```
1  #ifndef _ERRORS_H_INCLUDED_
2  #define _ERRORS_H_INCLUDED_
3
4  #include <iostream>
5  #include <string>
6  #include <stdlib.h>
7
8  using namespace std;
9
10 /* Se definen las macros de los mensajes de error
11  */
12
13 #define MSJ_ERROR_ARCHIVO_INCORRECTO "Ingresar una imagen con identificador P2"
14 #define MSJ_ERROR_IMAGENES_DE_DISTINTO_TAMANIO "Imágenes de distinto tamaño"
15 #define MSJ_ERROR_NO_SE_PUEDE_LEER_ARCHIVO "No se puede leer el archivo"
16 #define MSJ_ERROR_NO_SE_PUEDE_OBTENER_ANCHO "No se puede obtener el ancho de la imagen"
17 #define MSJ_ERROR_NO_SE_PUEDE_OBTENER_LARGO "No se puede obtener el largo de la imagen"
18 #define MSJ_ERROR_NO_SE_PUEDE_OBTENER_INTENSIDAD_MAXIMA "No se puede obtener la intensidad máxima de
color"
19 #define MSJ_ERROR_NO_SE_PUEDE_OBTENER_INTENSIDAD "No se puede obtener la intensidad de color"
20
21 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_FORMATO "No se puede imprimir formato (P2)"
22 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_ANCHO "No se puede imprimir el ancho de la imagen"
23 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_LARGO "No se puede imprimir el largo de la imagen"
24 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_INTENSIDAD_MAXIMA "No se puede imprimir la intensidad máxima
de color"
25 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_INTENSIDAD "No se puede imprimir la intensidad de
color"
26 #define MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_DELIMITADOR "No se puede imprimir el delimitador"
27 #define MSJ_ERROR_NO_SE_PUEDE_OPERAR "No se puede operar"
28 #define MSJ_ERROR_OPERACION_INVALIDA "Operación inválida"
29
30
31 /* Se define una clase Errors que tienen métodos que se encargan de
32  * imprimir por std::cerr los mensajes de error. Los métodos se definen
33  * con static para que cuando sean convocadas de la siguiente manera:
34  * Errors::método();
35  */
36
37 class Errores
38 {
39     public:
40         Errores();
41         ~Errores();
42         static void archivoIncorrecto();
43         static void imagenesDeDistintoTamanio();
44         static void noSePuedeLeerArchivo();
45         static void noSePuedeObtenerAncho();
46         static void noSePuedeObtenerLargo();
47         static void noSePuedeObtenerIntensidadMaxima();
48         static void noSePuedeObtenerIntensidad();
49
50         static void noSePuedeImprimirFormato();
51         static void noSePuedeImprimirAncho();
52         static void noSePuedeImprimirLargo();
53         static void noSePuedeImprimirIntensidadMaxima();
54         static void noSePuedeImprimirIntensidad();
55         static void noSePuedeImprimirDelimitador();
56
57         static void noSePuedeOperar();
58         static void operacionInvalida();
59 };
60 #endif
```

Errores.cpp

```
1  #include "Errores.h"
2
3  Errores::Errores(){}
4
5  Errores::~Errores(){}
6
7  void Errores::archivoIncorrecto(void)
8  {
9      cerr<<MSJ_ERROR_ARCHIVO_INCORRECTO<<endl;
10     exit(1);
11 }
12
13 void Errores::imagenesDeDistintoTamano(void)
14 {
15     cerr<<MSJ_ERROR_IMAGENES_DE_DISTINTO_TAMANIO<<endl;
16     exit(1);
17 }
18
19 void Errores::noSePuedeLeerArchivo(void)
20 {
21     cerr<<MSJ_ERROR_IMAGENES_DE_DISTINTO_TAMANIO<<endl;
22     exit(1);
23 }
24
25 void Errores::noSePuedeObtenerAncho(void)
26 {
27     cerr<<MSJ_ERROR_NO_SE_PUEDE_OBTENER_ANCHO<<endl;
28     exit(1);
29 }
30
31 void Errores::noSePuedeObtenerLargo(void)
32 {
33     cerr<<MSJ_ERROR_NO_SE_PUEDE_OBTENER_LARGO<<endl;
34     exit(1);
35 }
36
37 void Errores::noSePuedeObtenerIntensidadMaxima(void)
38 {
39     cerr<<MSJ_ERROR_NO_SE_PUEDE_OBTENER_INTENSIDAD_MAXIMA<<endl;
40     exit(1);
41 }
42
43 void Errores::noSePuedeObtenerIntensidad(void)
44 {
45     cerr<<MSJ_ERROR_NO_SE_PUEDE_OBTENER_INTENSIDAD<<endl;
46     exit(1);
47 }
48
49
50 void Errores::noSePuedeImprimirFormato(void)
51 {
52     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_FORMATO<<endl;
53     exit(1);
54 }
55
56
57 void Errores::noSePuedeImprimirAncho(void)
58 {
59     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_ANCHO<<endl;
60     exit(1);
61 }
62
63 void Errores::noSePuedeImprimirLargo(void)
64 {
65     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_LARGO<<endl;
66     exit(1);
```

```
67 }
68
69 void Errores::noSePuedeImprimirIntensidadMaxima(void)
70 {
71     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_INTENSIDAD_MAXIMA<<endl;
72     exit(1);
73 }
74
75 void Errores::noSePuedeImprimirIntensidad(void)
76 {
77     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_INTENSIDAD<<endl;
78     exit(1);
79 }
80
81 void Errores::noSePuedeImprimirDelimitador(void)
82 {
83     cerr<<MSJ_ERROR_NO_SE_PUEDE_IMPRIMIR_DELIMITADOR<<endl;
84     exit(1);
85 }
86
87 void Errores::noSePuedeOperar(void)
88 {
89     cerr<<MSJ_ERROR_NO_SE_PUEDE_OPERAR<<endl;
90     exit(1);
91 }
92
93 void Errores::operacionInvalida(void)
94 {
95     cerr<<MSJ_ERROR_OPERACION_INVALIDA<<endl;
96     exit(1);
97 }
```

```

1  #ifndef _CMDLINE_H_INCLUDED_
2  #define _CMDLINE_H_INCLUDED_
3
4  #include <string>
5  #include <iostream>
6  #include <string.h>
7  #include <cctype>
8
9  #include "Errores.h"
10 #include "Queue.h"
11 #include "Template_Pila.h"
12 #include "constantes.h"
13
14 #define OPT_DEFAULT    0
15 #define OPT_SEEN       1
16 #define OPT_MANDATORY 2
17
18 using namespace std;
19
20 struct option_t {
21     int has_arg;
22     const char *short_name;
23     const char *long_name;
24     const char *def_value;
25     void (*parse)(std::string const &);
26     int flags;
27 };
28
29 class cmdline {
30     // Este atributo apunta a la tabla que describe todas
31     // las opciones a procesar. Por el momento, sólo puede
32     // ser modificado mediante constructor, y debe finalizar
33     // con un elemento nulo.
34     //
35     option_t *option_table;
36
37     // El constructor por defecto cmdline::cmdline(), es
38     // privado, para evitar construir parsers sin opciones.
39     //
40     cmdline();
41     int do_long_opt(const char *, const char *);
42     int do_short_opt(const char *, const char *);
43 public:
44     cmdline(option_t *);
45     void parse(int, char * const []);
46
47     void parseOperacion(string, Queue<string> *);
48
49     int precedencia(string, string);
50
51     bool preParse(string);
52 };
53
54
55
56 #endif

```

cmdline.cpp

```
1 // cmdline - procesamiento de opciones en la línea de comando.
2
3 #include <string>
4 #include <cstdlib>
5 #include <iostream>
6 #include "cmdline.h"
7
8 using namespace std;
9
10 cmdline::cmdline()
11 {
12 }
13
14 cmdline::cmdline(option_t *table) : option_table(table)
15 {
16 }
17
18 void cmdline::parse(int argc, char * const argv[])
19 {
20
21 #define END_OF_OPTIONS(p)          \
22     ((p)->short_name == 0 \
23      && (p)->long_name == 0 \
24      && (p)->parse == 0)
25
26 // Primer pasada por la secuencia de opciones: marcamos
27 // todas las opciones, como no procesadas. Ver código de
28 // abajo.
29 //
30 for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
31     op->flags &= ~OPT_SEEN;
32
33 // Recorremos el arreglo argv. En cada paso, vemos
34 // si se trata de una opción corta, o larga. Luego,
35 // llamamos a la función de parseo correspondiente.
36 //
37 for (int i = 1; i < argc; ++i) {
38     // Todos los parámetros de este programa deben
39     // pasarse en forma de opciones. Encontrar un
40     // parámetro no-opción es un error.
41     //
42     if (argv[i][0] != '-') {
43         cerr << "Invalid non-option argument: "
44              << argv[i]
45              << endl;
46         exit(1);
47     }
48
49     // Usamos "--" para marcar el fin de las
50     // opciones; todo los argumentos que puedan
51     // estar a continuación no son interpretados
52     // como opciones.
53     //
54     if (argv[i][1] == '-'
55         && argv[i][2] == 0)
56         break;
57
58     // Finalmente, vemos si se trata o no de una
59     // opción larga; y llamamos al método que se
60     // encarga de cada caso.
61     //
62     if (argv[i][1] == '-')
63         i += do_long_opt(&argv[i][2], argv[i + 1]);
64     else
65         i += do_short_opt(&argv[i][1], argv[i + 1]);
66 }
```

```

67
68 // Segunda pasada: procesamos aquellas opciones que,
69 // (1) no hayan figurado explícitamente en la línea
70 // de comandos, y (2) tengan valor por defecto.
71 //
72 for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
73 #define OPTION_NAME(op) \
74 (op->short_name ? op->short_name : op->long_name)
75 if (op->flags & OPT_SEEN)
76     continue;
77 if (op->flags & OPT_MANDATORY) {
78     cerr << "Option "
79         << "-"
80         << OPTION_NAME(op)
81         << " is mandatory."
82         << "\n";
83     exit(1);
84 }
85 if (op->def_value == 0)
86     continue;
87 op->parse(string(op->def_value));
88 }
89 }
90
91 int
92 cmdline::do_long_opt(const char *opt, const char *arg)
93 {
94     // Recorremos la tabla de opciones, y buscamos la
95     // entrada larga que se corresponda con la opción de
96     // línea de comandos. De no encontrarse, indicamos el
97     // error.
98     //
99     for (option_t *op = option_table; op->long_name != 0; ++op) {
100         if (string(opt) == string(op->long_name)) {
101             // Marcamos esta opción como usada en
102             // forma explícita, para evitar tener
103             // que inicializarla con el valor por
104             // defecto.
105             //
106             op->flags |= OPT_SEEN;
107
108             if (op->has_arg) {
109                 // Como se trata de una opción
110                 // con argumento, verificamos que
111                 // el mismo haya sido provisto.
112                 //
113                 if (arg == 0) {
114                     cerr << "Option requires argument: "
115                         << "--"
116                         << opt
117                         << "\n";
118                     exit(1);
119                 }
120                 op->parse(string(arg));
121                 return 1;
122             } else {
123                 // Opción sin argumento.
124                 //
125                 op->parse(string(""));
126                 return 0;
127             }
128         }
129     }
130
131     // Error: opción no reconocida. Imprimimos un mensaje
132     // de error, y finalizamos la ejecución del programa.

```

```

133     //
134     cerr << "Unknown option: "
135         << "--"
136         << opt
137         << "."
138         << endl;
139     exit(1);
140
141     // Algunos compiladores se quejan con funciones que
142     // lógicamente no pueden terminar, y que no devuelven
143     // un valor en esta última parte.
144     //
145     return -1;
146 }
147
148 int
149 cmdline::do_short_opt(const char *opt, const char *arg)
150 {
151     option_t *op;
152
153     // Recorremos la tabla de opciones, y buscamos la
154     // entrada corta que se corresponda con la opción de
155     // línea de comandos. De no encontrarse, indicamos el
156     // error.
157     //
158     for (op = option_table; op->short_name != 0; ++op) {
159         if (string(opt) == string(op->short_name)) {
160             // Marcamos esta opción como usada en
161             // forma explícita, para evitar tener
162             // que inicializarla con el valor por
163             // defecto.
164             //
165             op->flags |= OPT_SEEN;
166
167             if (op->has_arg) {
168                 // Como se trata de una opción
169                 // con argumento, verificamos que
170                 // el mismo haya sido provisto.
171                 //
172                 if (arg == 0) {
173                     cerr << "Option requires argument: "
174                         << "--"
175                         << opt
176                         << "\n";
177                     exit(1);
178                 }
179                 op->parse(string(arg));
180                 return 1;
181             } else {
182                 // Opción sin argumento.
183                 //
184                 op->parse(string(""));
185                 return 0;
186             }
187         }
188     }
189
190     // Error: opción no reconocida. Imprimimos un mensaje
191     // de error, y finalizamos la ejecución del programa.
192     //
193     cerr << "Unknown option: "
194         << "--"
195         << opt
196         << "."
197         << endl;
198     exit(1);

```

```

199
200 // Algunos compiladores se quejan con funciones que
201 // lógicamente no pueden terminar, y que no devuelven
202 // un valor en esta última parte.
203 //
204 return -1;
205 }
206
207
208 //Esta función está implementada ...
209 void
210 cmdline::parseOperacion(string str, Queue<string> *cola)
211 {
212     Pila<string> pila;
213     string aux, simbolo;
214     string aux2="";
215
216     if(!preParse(str))
217         Errores::operacionInvalida();
218
219     unsigned int leng = str.size();
220
221     unsigned int i=0;
222     while (i < leng)
223     {
224         simbolo = str[i];
225
226         if(isdigit(simbolo[0]) ) //Si símbolo es un número ingreso al if
227         {
228             int j=0;
229             while( isdigit(simbolo[0]) || simbolo == "." || simbolo == NUMERO_IMAGINARIO ){
230                 aux2.insert(j,simbolo);
231                 j++;
232                 i++;
233                 simbolo = str[i];
234             }
235             if(simbolo == NUMERO_IMAGINARIO)
236             {
237                 aux2.insert(j,simbolo);
238             }else{
239                 i--;
240             }
241             (*cola).enqueue(aux2); // Agrego a la cola.
242             aux2.clear();
243
244             }else if(isalpha(simbolo[0])){
245
246                 if(simbolo == "p" && str[i+1] == NUMERO_IMAGINARIO[0])
247                 {
248                     simbolo.insert(1,"i"); //cargamos el valor pi
249                     (*cola).enqueue(simbolo);
250                     i++;
251                 }else if(simbolo == NUMERO_IMAGINARIO && !isalpha(str[i+1])){
252
253                     simbolo.insert(0,"1"); // Para cargar el valor "1i"
254                     (*cola).enqueue(simbolo); // Agregamos a la cola
255                 }else if(simbolo == VARIABLE_INDEPENDIENTE){ // Si simbolo es "z"
256                     (*cola).enqueue(simbolo); // Agregamos a la cola
257                 }else{
258                     int j = 0;
259                     while(isalpha(simbolo[0]))
260                     {
261                         aux2.insert(j,simbolo);
262                         j++;
263                         i++;
264                         simbolo = str[i];

```



```

265     }
266     i--;
267     pila.push(aux2);           // Se agrega a la pila.
268     aux2.clear();             // Se limpia la variable aux2;
269 }
270 }else if(simbolo == PARENTESIS_IZQUIERDO){
271     pila.push(simbolo);       // Se agrega a la pila
272 }
273 else if(simbolo == PARENTESIS_DERECHO){
274     while(pila.tope() != PARENTESIS_IZQUIERDO){
275         aux = pila.tope();
276         pila.pop();           // Se quita a la pila
277         (*cola).enqueue(aux); // Se agrega a la cola
278     }
279     if(pila.tope() == PARENTESIS_IZQUIERDO)
280     {
281         pila.pop();           // eliminar el token y el "("
282     }
283 }
284 else if(simbolo == OPERADOR_SUMA || simbolo == OPERADOR_SUSTRACCION || simbolo == OPERADOR_MULTIPLICACION
|| simbolo == OPERADOR_DIVISION || simbolo == OPERADOR_POTENCIA){
285     if( (simbolo == OPERADOR_SUMA || simbolo == OPERADOR_SUSTRACCION) && i == 0)
286     {
287
288         int j=0;
289         aux2.insert(j,simbolo);
290         j++;
291         i++;
292         simbolo = str[i];
293
294         while( isdigit(simbolo[0]) || simbolo == "." || simbolo == NUMERO_IMAGINARIO ){
295             aux2.insert(j,simbolo);
296             j++;
297             i++;
298             simbolo = str[i];
299         }
300         if(simbolo == NUMERO_IMAGINARIO)
301         {
302             aux2.insert(j,"1");
303             j++;
304             aux2.insert(j,simbolo);
305             i++;
306             simbolo = str[i];
307
308             if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == ","){
309                 (*cola).enqueue(aux2); // Agrego a la cola.
310                 aux2.clear();
311                 continue;
312             }
313         }
314         if(simbolo == VARIABLE_INDEPENDIENTE)
315         {
316             aux2.insert(j,simbolo);
317             i++;
318             simbolo = str[i];
319
320             if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == ","){
321                 (*cola).enqueue(aux2); // Agrego a la cola.
322                 aux2.clear();
323                 continue;
324             }
325         }
326         if(simbolo == "p" && str[i+1] == NUMERO_IMAGINARIO[0])
327         {
328             aux2.insert(j,VARIABLE_PI);
329             i=i+2;

```

```

330         simbolo = str[i];
331         if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == "," ){
332             (*cola).enqueue(aux2);           // Agrego a la cola.
333             aux2.clear();
334             continue;
335         }
336     }
337     (*cola).enqueue(aux2);           // Agrego a la cola.
338     aux2.clear();
339     }else if( (simbolo == OPERADOR_SUMA || simbolo == OPERADOR_SUSTRACCION) && ( !isdigit(str[i-1])
&& str[i-1] != NUMERO_IMAGINARIO[0]
340         && str[i-1] != VARIABLE_INDEPENDIENTE[0] && str[i-1] != PARENTESIS_DERECHO[0]) ){
341         int j=0;
342         aux2.insert(j,simbolo);
343         j++;
344         i++;
345         simbolo = str[i];
346
347         while( isdigit(simbolo[0]) || simbolo == "." || simbolo == NUMERO_IMAGINARIO ){
348             aux2.insert(j,simbolo);
349             j++;
350             i++;
351             simbolo = str[i];
352         }
353         if(simbolo == NUMERO_IMAGINARIO)
354         {
355             aux2.insert(j,"1");
356             j++;
357             aux2.insert(j,simbolo);
358             i++;
359             simbolo = str[i];
360             if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == "," ){
361                 (*cola).enqueue(aux2);           // Agrego a la cola.
362                 aux2.clear();
363                 continue;
364             }
365         }
366         if(simbolo == VARIABLE_INDEPENDIENTE)
367         {
368             aux2.insert(j,simbolo);
369             i++;
370             simbolo = str[i];
371             if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == "," ){
372                 (*cola).enqueue(aux2);           // Agrego a la cola.
373                 aux2.clear();
374                 continue;
375             }
376         }
377         if(simbolo == "p" && str[i+1] == NUMERO_IMAGINARIO[0])
378         {
379             aux2.insert(j,VARIABLE_PI);
380             i=i+2;
381             simbolo = str[i];
382             if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == "," ){
383                 (*cola).enqueue(aux2);           // Agrego a la cola.
384                 aux2.clear();
385                 continue;
386             }
387         }
388         (*cola).enqueue(aux2);           // Agrego a la cola.
389         aux2.clear();
390     }
391     if(simbolo == PARENTESIS_IZQUIERDO || simbolo == PARENTESIS_DERECHO || simbolo == "," ){
392         continue;
393     }
394     while ((!pila.vacia()) && (precedencia(pila.tope(),simbolo) <= 0) && (pila.tope() !=

```

```

PARENTESIS_IZQUIERDO))
395         {
396             aux = pila.tope();          // Se lee el tope
397             pila.pop();                // Se quita a la pila
398             (*cola).enqueue(aux);      // Se agrega a la cola
399         }
400         pila.push(simbolo);            // Se agrega a la pila
401     }
402     i++;
403 }
404 if (i == leng)
405 {
406     while(!pila.vacia())
407     {
408
409         aux = pila.tope();
410         if(aux == PARENTESIS_IZQUIERDO || aux == PARENTESIS_DERECHO)
411         {
412             Errores::operacionInvalida();
413         }
414         pila.pop();                    // Se quita a la pila
415         (*cola).enqueue(aux);          // Se agrega a la cola
416     }
417 }
418 }
419
420 /* Función que:
421  * retorna -1 si el operador1 tiene mayor precedencia que el operador2
422  * retorna 0 si el operador1 tiene la misma precedencia que el operador2
423  * retorna 1 si el operador1 tiene menor precedencia que el operador 2
424  */
425
426 int cmdline::precedencia(string operador1, string operador2)
427 {
428     if( (operador1==OPERADOR_SUMA||operador1==OPERADOR_SUSTRACCION) && (operador2==OPERADOR_MULTIPLICACION
||operador2==OPERADOR_DIVISION||operador2==OPERADOR_POTENCIA) )
429     {
430         return 1;
431     }else if( (operador1==OPERADOR_MULTIPLICACION||operador1==OPERADOR_DIVISION||operador1==
OPERADOR_POTENCIA) && (operador2==OPERADOR_SUMA||operador2==OPERADOR_SUSTRACCION) ){
432
433         return -1;
434     }else if( (operador1==OPERADOR_MULTIPLICACION||operador1==OPERADOR_DIVISION) && (operador2==
OPERADOR_POTENCIA) ){
435         return 1;
436     }else if( (operador1==OPERADOR_POTENCIA) && (operador2==OPERADOR_MULTIPLICACION||operador2==
OPERADOR_DIVISION) ){
437         return -1;
438     }else return 0;                    // Encaso de que los operadores tengan igual precedencia.
439 }
440
441
442
443 bool cmdline::preParse(string st){
444
445     if(st.length() == 0) //Si la cadena esta vacia
446         return false;
447
448     int i=0, open=0;
449     string aux="";
450
451     while(st[i] != '\0'){
452         if(isdigit(st[i]) == 0){ //Si no es un numero.
453             if((isalpha(st[i]) != 0) && (st[i] != VARIABLE_INDEPENDIENTE[0])){ //Si es una letra distinta
de z
454                 if(st[i] == NUMERO_IMAGINARIO[0])

```

```

455         i++;
456     else{ //Si no es una i
457         while(isalpha(st[i]) != 0){ //Mientras sea una letra guarda en aux lo que hay en st
caracter a caracter
458             aux.push_back(st[i]);
459             i++;
460         }
461         if(aux != FUNCION_SENO && aux != FUNCION_COSENO && aux != FUNCION_EXPONENCIAL && aux !=
FUNCION_LOGARITMO &&
462             aux != FUNCION_REAL && aux != FUNCION_IMAGINARIA && aux != FUNCION_ABS && aux !=
FUNCION_PHASE &&
463             aux != FUNCION_MAX && aux != VARIABLE_PI && st[i] != PARENTESIS_IZQUIERDO[0])
464             return false;
465
466         if(aux != VARIABLE_PI){
467
468             i++;
469
470             aux.clear();
471
472             while(st[i] != PARENTESIS_DERECHO[0] || open != 0){ //Mientras no sea ')' ni fin de
la cadena --> guarda el caracter actual en aux y avanza.
473                 if(st[i] == '\0')
474                     return false;
475                 aux.push_back(st[i]);
476                 if(st[i] == PARENTESIS_IZQUIERDO[0])
477                     open++;
478                 else if(st[i] == PARENTESIS_DERECHO[0])
479                     open--;
480                 i++;
481             }
482             if(!preParse(aux))
483                 return false;
484             i++;
485         }
486     }
487 }else{
488
489     if(st[i] != OPERADOR_SUMA[0] && st[i] != OPERADOR_SUSTRACCION[0] && st[i] !=
OPERADOR_MULTIPLICACION[0] && st[i] != OPERADOR_DIVISION[0]
490         && st[i] != OPERADOR_POTENCIA[0] && st[i] != VARIABLE_INDEPENDIENTE[0] && st[i] !=
PARENTESIS_IZQUIERDO[0] && st[i] != '.')
491         return false;
492
493     aux.clear();
494
495     if(st[i] == PARENTESIS_IZQUIERDO[0]){ //Si es un '(' va guardando hasta que encuentre ')' o
llegue al final de la cadena
496         i++;
497         while(st[i] != PARENTESIS_DERECHO[0] && st[i] != '\0'){
498             aux.push_back(st[i]);
499             i++;
500         }
501         if(st[i] != PARENTESIS_DERECHO[0]) //Si llego al final sin encontrar el ')' -->
Devuelve falso
502             return false;
503         else{ //Si encontro el ')' --> Llama recursivamente a la funcion y le pasa lo guardado
en aux.
504             if(!preParse(aux))
505                 return false;
506             i++;
507         }
508     }
509
510     else if(st[i] == '.'){
511         if(isdigit(st[i+1]) == 0)

```

```
512         return false;
513         i++;
514     }
515
516     else if(st[i] == OPERADOR_SUSTRACCION[0]){
517         i++;
518     }
519     else{ //Si es un operador válido distinto del - y no es una z o un '(' o un '.'
520         if(i == 0 && st[i] != VARIABLE_INDEPENDIENTE[0])
521             return false;
522
523         if(st[i] != VARIABLE_INDEPENDIENTE[0] && isdigit(st[i+1]) == 0 && st[i+1] !=
VARIABLE_INDEPENDIENTE[0] && st[i+1] != '.' && st[i] != '\0'){ //Si es distinto de z y el que le sigue no es un
numero ni la variable z ni el numero i, ni \0
524             if(isalpha(st[i+1]) == 0 && st[i+1] != '(')
525                 return false;
526             }
527             i++;
528         }
529     }
530     }else if(st[i] != '\0') //Si es un numero --> avanza
531         i++;
532     }
533     return true;
534 }
```

constantes.h

```
1  #ifndef  CONSTANTES_H_INCLUDED
2  #define  CONSTANTES_H_INCLUDED
3
4  #define  FORMATO_IMAGEN  "P2"
5  #define  CARACTER_COMENTARIO  '#'
6  #define  DELIMITADOR  " "
7
8  #define  FUNCION_EXPONENCIAL  "exp"
9  #define  FUNCION_REAL  "re"
10 #define  FUNCION_IMAGINARIA  "im"
11 #define  FUNCION_LOGARITMO  "ln"
12 #define  FUNCION_ABS  "abs"
13 #define  FUNCION_PHASE  "phase"
14 #define  FUNCION_SENO  "sin"
15 #define  FUNCION_COSENO  "cos"
16 #define  FUNCION_MAX  "max"
17
18 #define  NUMERO_IMAGINARIO  "i"
19 #define  VARIABLE_INDEPENDIENTE  "z"
20 #define  PARENTESIS_IZQUIERDO  "("
21 #define  PARENTESIS_DERECHO  ")"
22 #define  VARIABLE_PI  "pi"
23 #define  VARIABLE_PI_NEGADO  "-pi"
24
25 #define  OPERADOR_POTENCIA  "^"
26 #define  OPERADOR_SUMA  "+"
27 #define  OPERADOR_SUSTRACCION  "-"
28 #define  OPERADOR_DIVISION  "/"
29 #define  OPERADOR_MULTIPLICACION  "*"
30
31 #endif
```

main.h

```
1  /*
2      Archivo: main.h
3  */
4
5  #ifndef _MAIN_HPP_INCLUDED_
6  #define _MAIN_HPP_INCLUDED_
7
8  #include <fstream>
9  #include <iomanip>
10 #include <iostream>
11 #include <sstream>
12 #include <cstdlib>
13 #include <string>
14
15 #include "cmdline.h"
16 #include "Imagen.h"
17
18 #include "Complejo.h"
19
20 #include "Queue.h"
21 #include "Template_Pila.h"
22
23
24 using namespace std;
25
26 static void opt_input(string const &);
27 static void opt_output(string const &);
28 static void opt_function(string const &);
29 static void opt_help(string const &);
30
31
32
33 #endif
```

main.cpp

```
1  #include "main.h"
2
3  using namespace std;
4
5
6  static option_t options[] = {
7      {1, "i", "input", "-", opt_input, OPT_DEFAULT},
8      {1, "o", "output", "-", opt_output, OPT_DEFAULT},
9      {1, "f", "function", "-", opt_function, OPT_DEFAULT},
10     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
11     {0, },
12 };
13
14
15 static istream *iss = 0;    // Input stream objects (clase para manejo de los flujos de entrada)
16 static ostream *oss = 0;    // Output Stream objects (clase para manejo de los flujos de salida)
17 static fstream ifs;         // File Stream (para el manejo de archivos)
18 static fstream ofs;         // File Stream (para el manejo de archivos)
19 static string fun;          //String para manejo de la función de entrada
20
21 static void
22 opt_input(string const &arg)
23 {
24     // Si el nombre del archivos es "-", usaremos la entrada
25     // estándar. De lo contrario, abrimos un archivo en modo
26     // de lectura.
27     //
28     if (arg == "-") {
29         iss = &cin;         // Asignamos la entrada estandar cin como flujo de entrada.
30     }
31     else {
32         ifs.open(arg.c_str(), ios::in); // Abrimos el archivo de entrada.
33         iss = &ifs;          // Asignamos el archivo leído al flujo de entrada.
34     }
35
36     // Verificamos que el stream este OK.
37     //
38     if (!iss->good()) {
39         cerr << "cannot open "
40             << arg
41             << ". "
42             << endl;
43         exit(1);
44     }
45 }
46
47 static void
48 opt_output(string const &arg)
49 {
50     // Si el nombre del archivos es "-", usaremos la salida
51     // estándar. De lo contrario, abrimos un archivo en modo
52     // de escritura.
53     //
54     if (arg == "-") {
55         oss = &cout;         // Asignamos la salida estandar cout como flujo de salida.
56     } else {
57         ofs.open(arg.c_str(), ios::out); //Abrimos y/o creamos el archivo de salida
58         oss = &ofs;          // Asignamos el archivo como flujo de salida.
59     }
60
61     // Verificamos que el stream este OK.
62     //
63     if (!oss->good()) {
64         cerr << "cannot open "
65             << arg
66             << ". "
```



```

67         << endl;
68     exit(1);
69 }
70 }
71
72 static void
73 opt_function(string const &arg)
74 {
75     // Si el nombre del archivos es "-", usaremos la función por defecto
76     //de lo contrario, guardamos en un string la función que se va a evaluar.
77     //
78
79     if (arg == "-") {
80         fun = "z"; // Asignam la función por default "En este caso f(z)=z"
81     }
82     else {
83         fun = arg; // Asignamos la función pasada por línea de comandos.
84     }
85 }
86
87 static void
88 opt_help(string const &arg)
89 {
90     cout << "cmdline [-i file] [-o file] [-f fuction]"
91         << endl;
92     exit(0);
93 }
94
95 int main(int argc, char * const argv[])
96 {
97     cmdline cmdl(options); // Inicializamos el objeto con parametro tipo option_t.
98     cmdl.parse(argc, argv); // Parseamos argv.
99     Imagen imgEntrada(*iss); // Inicializamos los datos de la imagen recibida por el flujo de entrada.
100
101     // Construimos la imagen de salida con las mismas dimensiones que la imagen de entrada.
102     Imagen imgSalida(imgEntrada.getAncho(),imgEntrada.getAlto(),imgEntrada.getIntensidadMax());
103
104     Queue<string> cola;
105
106     cmdl.parseOperacion(fun, &cola); // Parseamos la operación pasada por línea de comandos.
107
108     imgEntrada.operar(imgSalida,&cola); // Realizamos la operación.
109
110     imgSalida.print(*oss); // Imprimimos la imagen por el flujo de salida.
111
112     return 0;
113 }

```

7. Enunciado

75.04/95.12 Algoritmos y Programación II Trabajo práctico 1: Estructuras de datos simples

Universidad de Buenos Aires - FIUBA
Primer cuatrimestre de 2020
\$Date: 2020/06/04 23:15:59 \$

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 ser 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 archivos 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 imagen 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 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 $f : \mathbb{C} \rightarrow \mathbb{C}$. 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 descrito en [3]. El mismo puede ser usado para reescribir expresiones *infix* en formato RPN para luego ser evaluadas durante el proceso de transformación.

Funciones. Las 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 $\exp(z)$, $\ln(z)$, $\text{re}(z)$, $\text{im}(z)$, $\text{abs}(z)$, $\text{phase}(z)$, 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 imagen. 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
```

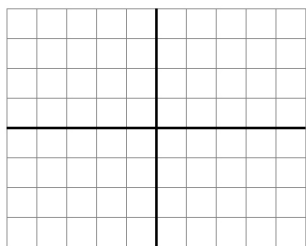


Figura 1: grid.pgm y grid-id.pgm

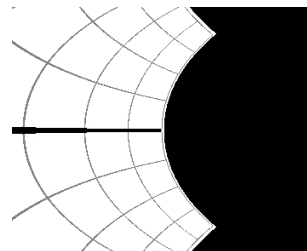


Figura 2: grid-exp.pgm

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)
```

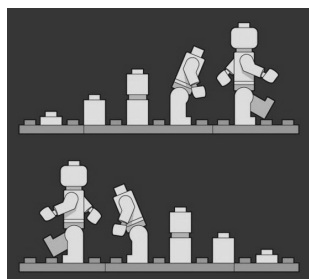


Figura 3: evolution.pgm

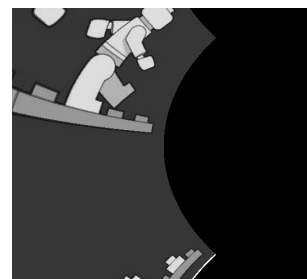


Figura 4: evolution-exp.pgm

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
```

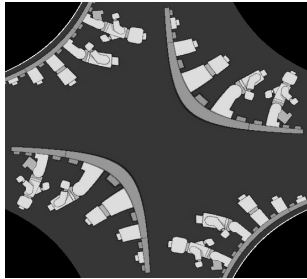


Figura 5: evolution-sqr.pgm

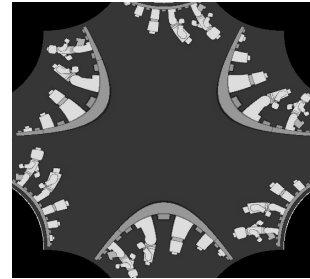


Figura 6: evolution-cube.pgm

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

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. Implementación

En función de los objetivos descriptos en la sección 1, cada grupo deberá proveer la implementación de las estructuras de datos como listas, pilas, colas, arreglos, etc. Es decir, en esta oportunidad no se aceptarán trabajos que usen la STL para sustituir estas funciones.

7. Fechas

La última fecha de entrega y presentación será el jueves 18/6.

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.