



ALGORITMOS Y PROGRAMACIÓN II (95.12)

TRABAJO PRÁCTICO N° 0

Alumnos:

Urquiza, Elías	eurquiza@fi.uba.ar	#100714
Vera Guzmán, Ramiro	rverag@fi.uba.ar	#95887
Nacachian, Mauro	mnacachian@fi.uba.ar	#99619

Fecha de entrega: Jueves 28 de Mayo

Índice

1. Introducción	3
2. Formato de imagen PGM	3
3. Diseño	3
4. Implementación	4
4.1. Clase Cmdline	4
4.2. Clase Complejo	5
4.3. Clase Images	5
4.4. Clase ComplexPlane	7
4.5. Clase ComplexTransform	9
4.6. Main	9
5. Compilación	11
6. Prueba del programa	12
6.1. Sin parámetros	12
6.2. Opción Help	12
6.3. Parámetro No-Válido	12
6.4. Input inexistente	13
6.5. Input no es pgm	13
6.6. Memoria	13
6.6.1. Valgrind al ejecutar una transformación compleja	13
6.7. Valgrind al ejecutar con parámetros no-válidos	14
7. Conclusión	16
8. Anexo	16
8.1. Enunciado	16
8.2. Repositorio	21
8.3. Código fuente	21
8.3.1. main.cpp	21
8.3.2. Complejo.h	24
8.3.3. Complejo.cpp	25
8.3.4. ComplexPlane.h	27
8.3.5. ComplexPlane.cpp	28
8.3.6. ComplexTransform.h	30
8.3.7. ComplexTransform.cpp	30
8.3.8. Images.h	31
8.3.9. Images.cpp	32
8.3.10. cmdline.h	37
8.3.11. cmdline.cpp	38
8.4. Imágenes de Prueba y Transformaciones	42
8.4.1. Test 4x4	42
8.4.2. Test 16x16	42
8.4.3. FEEP	42
8.4.4. Saturn	43
8.4.5. Lena	43
8.4.6. Marcie	43
8.4.7. Fractal Tree	44

8.4.8. Venus	44
8.4.9. Casablanca	44

1. Introducción

El siguiente trabajo práctico tiene como objetivo el diseño e implementación de un programa en C++, con el cual se busca ejercitarse los conceptos básicos del lenguaje. El programa debe recibir una imagen, con extensión **PGM** (*portable graymap*), a la cual se le aplica una transformada compleja y se obtiene una imagen resultante en otro archivo de formato **PGM**.

2. Formato de imagen PGM

Se utiliza el formato de imágenes *portable graymap* o PGM, con codificación ASCII.

Este formato define un mapa de escala de grises, donde el valor de intensidad de cada píxel se define con un valor entero entre 0 (negro) y un número máximo (blanco).

- La primera línea siempre contiene P2, el identificador o magic number.
- La segunda línea tiene un número que identifica al ancho de la imagen seguido de otro número que indica la altura.
- La tercera línea tiene el número máximo de intensidad.
- Puede haber líneas con comentarios con el símbolo #.
- Las demás líneas son números que representan los colores de la imagen.

3. Diseño

Existen distintos módulos internos dentro del programa, con los cuales se busca cumplir sus objetivos. Se detallan a continuación:

- Lectura de datos de entrada por línea de comandos:
 - * Para esto, se utiliza el script dado por la cátedra (*cmdline*), modificando y adaptando los parámetros que se requieran. [Ver sección 4.1.]
- Parseo de los datos de entrada.
 - * Dado que los argumentos que necesita el código parametrizan distintos módulos, es necesario segmentar dichos datos. Se usan las funciones de *callback* sugeridas en *cmdline* que se integran con las clases diseñadas.
- Creación de una Imagen
 - * Dado que el código trabaja sobre una imagen **PGM**, parece evidente el diseño de una clase *Images*. La misma se encarga de lo inherente a las imágenes. Es decir, a la creación, tratamiento de color, píxeles, y detección del formato. [Ver Sección 4.3.]
- Mapeo de una imagen a un Plano complejo.
 - * La clase denominada *ComplexPlane* se encarga de mapear una imagen a un plano complejo y viceversa. Según las dimensiones de la imagen con la que se trabaje, convierte los subíndices que representan cada píxel de una imagen a un número complejo. Por enunciado, se considera que toda imagen se debe representar como un rectángulo de lado dos, centrado en el origen. Dicha clase opera como nexo entre una imagen y un número complejo, logrando hacer las conversiones entre dichos entornos. [Ver Sección 4.4.]
- Transformaciones Holomorfas

- * La transformación $f(z)$ se aplica sobre un número complejo y guarda el resultado. Se recorre todo el plano complejo de destino para así obtener su transformación. Dado que el conjunto de funciones holomorfas es infinito, y que uno podría querer escalar el código para trabajar con un mayor número de transformaciones, se decidió crear esta clase encargada de llevar a cabo dichas funciones. De esta manera, buscamos minimizar el área de cambios aumentando la mantenibilidad del código.

- Clase complejo

- * Se utiliza la clase complejo como soporte de las clases previamente mencionadas.

4. Implementación

La implementación es prácticamente secuencial. La imagen de entrada se valida, al igual que los argumentos, se guardan sus datos y se forma una imagen destino de igual dimensión. Cada píxel de esta imagen nueva se mapea a un plano complejo y se aplica la función $f(z)$ a cada uno de sus valores. Luego, convirtiendo el resultado a un subíndice correspondiente a un píxel de la imagen original, se obtiene el color original y es copiado en el correspondiente punto del destino.

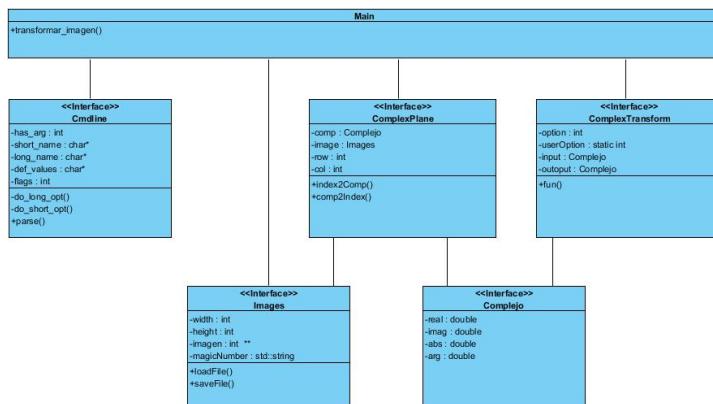


Figura 4.1: Diagrama de Clases. Obs: Se obviaron métodos constructores, destructores, *getters* y *setters*

4.1. Clase Cmdline

Para manejar argumentos y utilizar la clase cmdline requerida, se define una tabla con opciones que se esperan recibir, de tipo $option_t$. Definida dentro del main.

```

1 static option_t options[] = {
2     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
3     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
4     {1, "f", "function", NULL, opt_function, OPT_MANDATORY},
5     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
6     {0, },
7 };
8 };
  
```

Como es requerido, se definió a la cadena “-” (tercera columna de la tabla de opciones) como el valor por defecto (`std::cin` y `std::cout`) para las opciones de entrada y de salida. De esta manera, ya sea que se haya omitido esa opción o se la especifique explícitamente para que asuma su valor por defecto, en ambos casos se obtendrá el mismo resultado. Además, la clase cmdline pide especificar las funciones que se utilizarán para parsear cada opción (5ta columna de la tabla) - se las definió en el mismo main. Para las funciones que parsean las opciones de entrada, salida y ayuda se aprovecharon las que ya estaban implementadas en el archivo `main.cc` provisto con la implementación de la clase cmdline del curso.

4.2. Clase Complejo

Como se detalló en la sección 3 (Diseño), el dominio matemático sobre la cual se implementaron las clases ComplexPlane y ComplexTransform es el de los complejos. Es indispensable una función de bajo nivel que represente dichos valores. En el siguiente código se observa el esquema de la clase:

```
1 class Complejo{
2     private:
3         double real;
4         double imag;
5         double abs;
6         double arg;
7
8         void setPolar();
9
10    public:
11        //constructores:
12        Complejo();
13        Complejo(double, double);
14        Complejo(const Complejo &);
15        //destructor:
16        ~Complejo();
17
18        //obtener atributos:
19        double getReal();
20        double getImag();
21        double getAbs();
22        double getArg();
23        //setear atributos:
24        void setReal(double);
25        void setImag(double);
26
27        //conjugado:
28        Complejo conjugar();
29
30        //emitir:
31        void printRect();
32        void printPolar();
33
34        //sobrecarga de operadores
35        Complejo operator+(const Complejo &); //suma entre Complejos
36        Complejo operator+(double); //suma con un real
37        friend Complejo operator+ (double, const Complejo &);
38
39        Complejo operator-(const Complejo &); //resta entre Complejos
40        Complejo operator-(double); //resta por un real
41        friend Complejo operator- (double, const Complejo &);
42
43        Complejo operator*(const Complejo &); //producto entre Complejos
44        Complejo operator*(double); //producto por un real (escalar)
45        friend Complejo operator* (double, const Complejo &);
46
47        Complejo operator/(const Complejo &); //division entre complejos
48        Complejo operator/(double); //division por un real (escalar)
49        Complejo & operator=(const Complejo&); //asignacion entre complejos
50        bool operator==(const Complejo&); //comparacion entre complejos
51    };
```

4.3. Clase Images

Dado que el programa necesita usar imágenes PGM es esperable una clase que se encargue del tratamiento de imágenes. En el siguiente código se observa el esquema de la clase:

```
1 class Images {
2     private:
```

```

3 // Ancho de la imagen
4 int width;
5
6 // Altura de la imagen
7 int height;
8
9 // Intensidad maxima de la imagen
10 int maxInt;
11
12 // Matriz donde se almacenan los valores de intensidad
13 int ** imagen;
14
15 // El header o algo asi, donde dice P2. Si es P2, el formato es pgm
16 std::string magicNumber;
17
18 public:
19     // Constructor por defecto, inicializa todo en 0
20     Images();
21
22     // Constructor, se le pasa como argumento, ancho, alto y maxima intensidad
23     Images(int, int, int);
24
25     // Destructor, debera liberar la memoria de la matriz.
26     ~Images();
27
28     // Constructor por copia
29     Images(const Images &other);
30
31     // Sobreacarga del operador =
32     const Images& operator=(const Images &other);
33
34     // Sobrecarga del operador []
35     int & operator[](const std::pair<int,int> &);
36
37     // Sobrecarga operador ()
38     //
39     int & operator()(const int &, const int &);
40     const int & operator()(const int &, const int &) const;
41
42     // Funcion para procesar cada linea del archivo con formato .pgm (la usa el metodo de
43     // loadFile)
44     friend bool pgmParser(int &, int &, int &, std::stringstream * , Images * );
45
46     // Carga un archivo .pgm.
47     const Images & loadFile(std::istream * );
48
49     // Se guarda la instancia de Images en un archiv formato .pgm
50     const Images & saveFile(std::ostream * );
51
52     // Obtencion de los atributos
53     int getMaxInt() const;
54     int getWidth() const;
55     int getHeight() const;
56     std::string getMagicNumber() const;
57     int ** getColours() const;
58
59     // Imprime la matriz
60     void printColours();
61
62     // Devuelve true si la instancia tiene almacenado datos de una imagen pgm
63     bool isPGM();
64 };

```

La clase se encarga de procesar una imagen con formato PGM, guardando todos sus datos en los atributos privados. Además, permite generar una imagen en negro mediante unos de sus constructores con el tamaño pasado por argumento.

Sus atributos son el ancho y alto de la imagen, el máximo valor de intensidad que puede tener, un string que representa el header, y una matriz *int ** imagen* que representa el color en cada píxel (dentro del programa esta matriz será mapeada al plano complejo).

El constructor por defecto es *Images()*, el cual genera una imagen en negro de tamaño nulo. Para darle un tamaño específico y un color máximo se usa el constructor *Images(int ancho, int altura, int max)*, que dará una imagen en negro. Luego, está el constructor por copia y el destructor, que se encargará de liberar la memoria pedida por el atributo *int ** imagen*.

Se sobrecargó el operador asignación ('=') y los (), para acceder al atributo de matriz, de manera de pasársele el ancho y alto y poder obtener el color en ese píxel.

Como se ha dicho, el método *const Images & loadFile(std::istream *)* recibe la imagen para procesarla. Primero verifica si el argumento pasado es válido o no (por ejemplo si la imagen de entrada existe o no), también verifica si el header es "P2", en tal caso el formato pasado es PGM, de otra manera se corta el proceso devolviendo **this*. Luego, procede a leer línea por línea, y por cada una se invoca a la función friend *bool pgmParser(int &, int &, std::stringstream *, Images *)*, la cual se encarga de obtener todos los datos de la imagen y guardarlos. Además, en caso de que una línea contenga el símbolo # se la trata como un comentario y es ignorada. Por otro lado, el método *const Images & saveFile(std::ostream*)*, recibe el stream de salida y guarda la imagen siguiendo el formato mencionado en la sección 2.

Los demás métodos son para acceder a los atributos de la clase, y un último que devuelve *true* o *false* dependiendo si el formato guardado en la instancia es PGM o no.

4.4. Clase ComplexPlane

Esta clase se encarga de construir un rectángulo complejo de lado dos, centrado en el origen, a partir de las dimensiones de una imagen. Para ello, se divide el espacio de trabajo en cuatro cuadrantes, según el signo de la parte real y de la parte compleja. Referirse a la Figura 4.2.

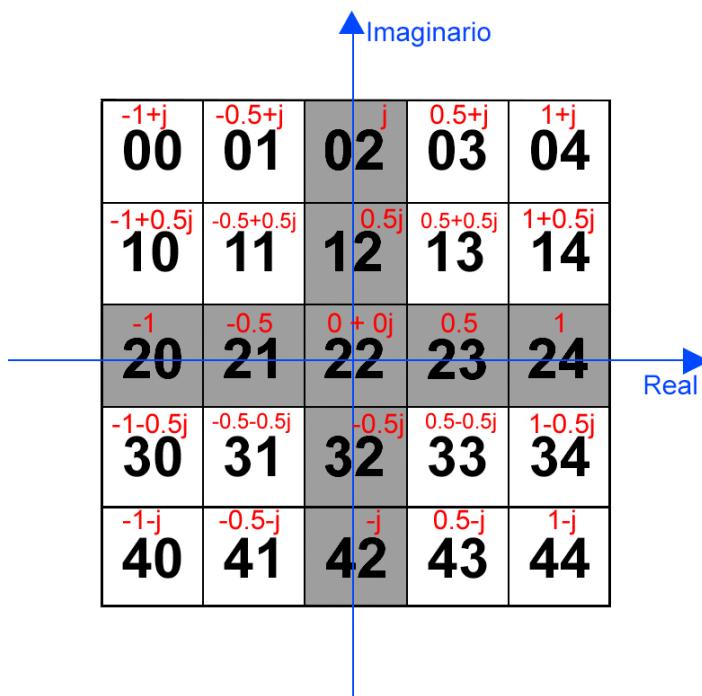


Figura 4.2: Ejemplo de ComplexPlane para una imagen de lado 5.

Para cada cuadrante se tiene que aplicar una ecuación distinta para pasar del valor de los subíndices al valor complejo. Consideramos que *columnaCentral* y *filaCentral* son números enteros correspondientes a la división del número de columnas o filas por dos. De esta manera, desarrollamos las siguientes expresiones:

$$Real = \begin{cases} \frac{columna}{columnaCentral} - 1 & , \text{ columna} < \text{columnaCentral} \\ 1 - \frac{(numColumnas - 1) - columna}{columnaCentral} & , \text{ columna} \geq \text{columnaCentral} \end{cases}$$

$$Imaginario = \begin{cases} 1 - \frac{fila}{filaCentral} & , fila < filaCentral \\ \frac{(numFilas - 1) - fila}{filaCentral} - 1 & , fila \geq filaCentral \end{cases}$$

Se puede observar que para una matriz cuadrada par no existirá una columna central ni una fila central, por lo que no habrá puntos sobre el eje real ni el imaginario. En cambio, en una matriz no-cuadrada (o cuadrada impar) sí se tendrán esos valores nulos.

Para pasar de un número complejo y obtener los subíndices asociados, se aplican las expresiones inversas.

$$Columna = \begin{cases} (Real + 1) \cdot columnaCentral & , Real < 0 \\ (numColumnas - 1) - columnaCentral \cdot (1 - Real) & , Real \geq 0 \end{cases}$$

$$Fila = \begin{cases} (1 - Imaginario) \cdot filaCentral & , Imaginario < 0 \\ (numFilas - 1) - filaCentral \cdot (Imaginario + 1) & , Imaginario \geq 0 \end{cases}$$

De esta manera, diseñamos la clase ComplexPlane de forma tal que sus atributos son un número complejo y dos enteros, correspondientes a los subíndices, y sus métodos consisten en permitirle guardar la información de un punto dado en ambas formas y en poder devolver estos valores.

```

1 class ComplexPlane {
2 private:
3     //numero complejo guardado
4     Complejo comp;
5     //imagen utilizada
6     Images image;
7     //fila y columna guardadas para el numero complejo entregado
8     int row;
9     int col;
10
11 public:
12     ComplexPlane();
13     //Constructor - guarda los datos de una imagen para tener sus dimensiones
14     ComplexPlane(const Images &);
15     //destructor
16     ~ComplexPlane();
17     // constructor copia
18     ComplexPlane(const ComplexPlane &);
19     // sobrecarga de la asignacion
20     ComplexPlane& operator=(const ComplexPlane &);
21
22
23     Complejo getComp() const;
24     Images getImage() const;
25     int getRow() const;
26     int getCol() const;
27
28     void index2Comp(int, int);
29     void comp2Index(Complejo &);
30 }
```

4.5. Clase ComplexTransform

La clase *ComplexTransform* posee las características de una transformación holomorfa. Por un lado posee una entrada, una salida y un atributo *option* que parametriza cuál es la función a escoger. Posee un atributo global y estático denominado *userOption* que es la conexión con la clase *cmdline*. El argumento se parsea y con un *setter* se precarga la opción *userOption*. Luego, cuando el objeto es creado más adelante, *userOption* inicializa al atributo *option* que es el que define el tipo de función a implementar.

```
1 class ComplexTransform{
2     private:
3         Complejo input;
4         Complejo output;
5         //1: z ; 2: exp(z) ;3 rotar; en otro caso: z
6         int option;
7         static int userOption;
8
9     public:
10        //constructor
11        ComplexTransform();
12        //con parÃ¡metros
13        ComplexTransform(int );
14        //destructor
15        ~ComplexTransform();
16
17        void fun(Complejo &);
18
19        Complejo getInput();
20        Complejo getOutput();
21
22        static void setOption(int );
23        intgetOption( void );
24
25 }
```

4.6. Main

Finalmente, en main interactúan instancias de las clases desarrolladas previamente con tal de lograr aplicar una transformación holomorfa a la imagen recibida e imprimirla en otro archivo imagen.

```
1
2 static istream* iss = 0;
3 static ostream* oss = 0;
4 static fstream ifs;
5 static fstream ofs;
6
7
8 int main(int argc, char * const argv[]) {
9
10    cmdline cmdl(options);
11    cmdl.parse(argc, argv);
12
13    Images origen;
14    origen.loadFile(iss);
15
16    Images destino(origen);
17    transformar_imagen(origen,destino);
18
19    destino.saveFile(oss);
20
21    if(iss != &cin)
22        ifs.close();
23    if(oss != &cout)
24        ofs.close();
25 }
```

```

26     exit(0);
27 }
28
29 void transformar_imagen(const Images & origen, Images & destino){
30
31     if(&origen == &destino)
32         return;
33
34     Complejo z_aux(0,0);
35     ComplexPlane plano(origen);
36     ComplexTransform transformada;
37
38     int ancho = origen.getWidth();
39     int altura = origen.getHeight();
40
41
42     for(int i = 0; i < altura; i++){
43         for(int j = 0; j < ancho; j++) {
44
45             plano.index2Comp(i, j); //guarda la coordenada en forma de num complejo
46             z_aux = plano.getComp();
47
48             transformada.fun(z_aux); //calcula la anti transformada
49             z_aux = transformada.getOutput();
50
51             plano.comp2Index(z_aux); //guarda los indices del pixel del origen
52
53             if(plano.getRow() < 0 || plano.getCol() < 0)
54                 destino(i,j)=0;
55             else
56                 destino(i,j) = origen(plano.getRow(),plano.getCol()); //guarda el pixel
57
58         }
59     }
60
61     return;
62 }
```

5. Compilación

Se optó por utilizar la herramienta make en lugar de la compilación manual, ya que esta herramienta utiliza el archivo Makefile, el cual se lo configura para automatizar el proceso, manteniendo al proyecto ordenado y facilitando su desarrollo. Se ejecuta desde la terminal de LINUX simplemente insertando el comando make, este también incluye el programa Valgrind.

makefile.

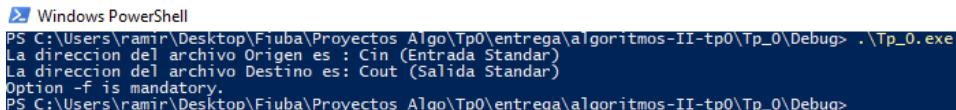
```
1 #####  
2 # Automatically-generated file. Do not edit!  
3 #####  
4  
5 -include ../makefile.init  
6  
7 RM := rm -rf  
8  
9 # All of the sources participating in the build are defined here  
10 -include sources.mk  
11 -include src/subdir.mk  
12 -include subdir.mk  
13 -include objects.mk  
14  
15 ifneq ($(MAKECMDGOALS),clean)  
16 ifneq ($(strip $(CC_DEPS)),)  
17 -include $(CC_DEPS)  
18 endif  
19 ifneq ($(strip $(C++_DEPS)),)  
20 -include $(C++_DEPS)  
21 endif  
22 ifneq ($(strip $(C_UPPER_DEPS)),)  
23 -include $(C_UPPER_DEPS)  
24 endif  
25 ifneq ($(strip $(CXX_DEPS)),)  
26 -include $(CXX_DEPS)  
27 endif  
28 ifneq ($(strip $(CPP_DEPS)),)  
29 -include $(CPP_DEPS)  
30 endif  
31 ifneq ($(strip $(C_DEPS)),)  
32 -include $(C_DEPS)  
33 endif  
34 endif  
35  
36 -include ../makefile.defs  
37  
38 # Add inputs and outputs from these tool invocations to the build variables  
39  
40 # All Target  
41 all: Tp_0.exe  
42  
43 # Tool invocations  
44 Tp_0.exe: $(OBJS) $(USER_OBJS)  
45     @echo 'Building target: $@'  
46     @echo 'Invoking: Cygwin C++ Linker'  
47     g++ -o "Tp_0.exe" $(OBJS) $(USER_OBJS) $(LIBS)  
48     @echo 'Finished building target: $@'  
49     @echo ''  
50  
51 # Other Targets  
52 clean:  
53     -$(RM) $(CC_DEPS) $(C++_DEPS) $(EXECUTABLES) $(OBJS) $(C_UPPER_DEPS) $(CXX_DEPS) $(CPP_DEPS) $(  
54         C_DEPS) Tp_0.exe  
55     -@echo ''  
56  
57 .PHONY: all clean dependents  
58 -include ../makefile.targets
```

6. Prueba del programa

A continuación se presentan las distintas formas en que se comporta el programa para distintos parámetros. Se incluyen pruebas en Linux, seguidas de las mismas pruebas en Windows. Para ver los resultados obtenidos con imágenes de prueba, referirse al Anexo, sección 8.3.

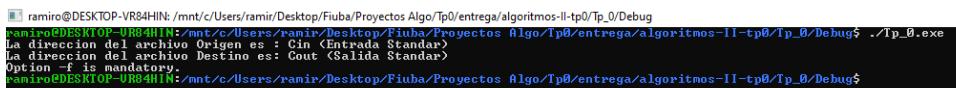
6.1. Sin parámetros

```
1 ./Tp_0.exe
2 La direccion del archivo Origen es : Cin (Entrada Standar)
3 La direccion del archivo Destino es: Cout (Salida Standar)
4 Option -f is mandatory.
```



```
PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug> ./Tp_0.exe
La direccion del archivo Origen es : Cin (Entrada Standar)
La direccion del archivo Destino es: Cout (Salida Standar)
Option -f is mandatory.
```

Figura 6.1: Prueba sin parámetros en Windows

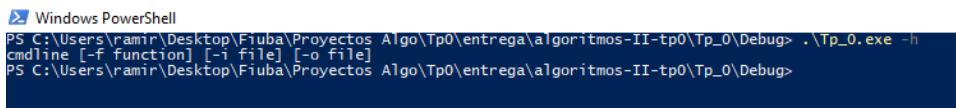


```
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug$ ./Tp_0.exe
La direccion del archivo Origen es : Cin (Entrada Standar)
La direccion del archivo Destino es: Cout (Salida Standar)
Option -f is mandatory.
```

Figura 6.2: Prueba sin parámetros en Linux

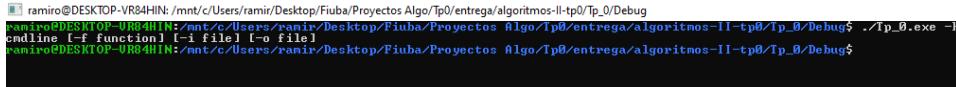
6.2. Opción Help

```
1 ./Tp_0.exe -h
2 cmdline [-f function] [-i file] [-o file]
```



```
PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug> ./Tp_0.exe -h
cmdline [-f function] [-i file] [-o file]
```

Figura 6.3: Prueba opción Help en Windows



```
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug$ ./Tp_0.exe -h
cmdline [-f function] [-i file] [-o file]
```

Figura 6.4: Prueba opción Help en Linux

6.3. Parámetro No-Válido

```
1 ./Tp_0.exe -f hola
2 La transformacion elegida es f(z)= hola
3 hola No es un parametro Valido
```

```

PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug> ./Tp_0.exe -f hola
La transformacion elegida es f(z)= hola
hola No es un parametro valido
PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug>

```

Figura 6.5: Prueba parámetro Invalido en Windows

```

ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug
La transformacion elegida es f(z)= hola
hola No es un parametro valido
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug$ 

```

Figura 6.6: Prueba parámetro Invalido en Linux

6.4. Input inexistente

```

1 ./Tp_0.exe -i inexistente.pgm -f z
2 La direccion del archivo Origen es :inexistente.pgm
3 cannot open inexistente.pgm.

```

```

PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug> ./Tp_0.exe -i inexistente.pgm -f z
La direccion del archivo Origen es :inexistente.pgm
cannot open inexistente.pgm.
PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug>

```

Figura 6.7: Prueba Entrada inexistente en Windows

```

ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug
La direccion del archivo Origen es :inexistente.pgm
cannot open inexistente.pgm.
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug$ 

```

Figura 6.8: Prueba Entrada inexistente en Linux

6.5. Input no es pgm

```

1 ./Tp_0.exe -i utils/false-saturn.pgm -o salida.pgm -f z
2 La direccion del archivo Origen es :utils/false-saturn.pgm
3 La direccion del archivo Destino es: salida.pgm
4 La transformacion elegida es f(z)= z
5 Formato no .pgm

```

```

PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug> ./Tp_0.exe -i utils/false-saturn.pgm -o salida.pgm -f z
La direccion del archivo Origen es :utils/false-saturn.pgm
La direccion del archivo Destino es: salida.pgm
La transformacion elegida es f(z)= z
Formato no .pgm
PS C:\Users\ramir\Desktop\Fiuba\Proyectos Algo\Tp0\entrega\algoritmos-II-tp0\Tp_0\Debug>

```

Figura 6.9: Prueba Entrada no PGM en Windows

```

ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug
La direccion del archivo Origen es :utils/false-saturn.pgm
La direccion del archivo Destino es: salida.pgm
La transformacion elegida es f(z)= z
Formato no .pgm
ramiro@DESKTOP-VR84HIN: /mnt/c/Users/ramir/Desktop/Fiuba/Proyectos Algo/Tp0/entrega/algoritmos-II-tp0/Tp_0/Debug$ 

```

Figura 6.10: Prueba Entrada no PGM en Linux

6.6. Memoria

6.6.1. Valgrind al ejecutar una transformación compleja

```

1 valgrind ./Tp_0.exe -i utils/fractal_tree.ascii.pgm -o utils/out-tree -f "exp(z)"
2 ==12041== Memcheck, a memory error detector
3 ==12041== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
4 ==12041== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
5 ==12041== Command: ./Tp_0.exe -i utils/fractal_tree.ascii.pgm -o utils/out-tree -f exp(z)
6 ==12041==
7 La direccion del archivo Origen es :utils/fractal_tree.ascii.pgm
8 La direccion del archivo Destino es: utils/out-tree
9 La transformacion elegida es f(z)= exp(z)
10 ==12041==
11 ==12041== HEAP SUMMARY:
12 ==12041==     in use at exit: 0 bytes in 0 blocks
13 ==12041==   total heap usage: 368,460 allocs, 368,460 frees, 74,946,479 bytes allocated
14 ==12041==
15 ==12041== All heap blocks were freed -- no leaks are possible
16 ==12041==
17 ==12041== For counts of detected and suppressed errors, rerun with: -v
18 ==12041== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

6.7. Valgrind al ejecutar con parámetros no-válidos

```

mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe
==1866== Memcheck, a memory error detector
==1866== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1866== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1866== Command: ./Tp_0.exe
==1866==
La direccion del archivo Origen es : Cin (Entrada Standar)
La direccion del archivo Destino es: Cout (Salida Standar)
Option -f is mandatory.
==1866==
==1866== HEAP SUMMARY:
==1866==     in use at exit: 0 bytes in 0 blocks
==1866==   total heap usage: 2 allocs, 2 frees, 73,728 bytes allocated
==1866==
==1866== All heap blocks were freed -- no leaks are possible
==1866==
==1866== For counts of detected and suppressed errors, rerun with: -v
==1866== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.11: Prueba con Valgrind 01

```

mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe
==1861== Memcheck, a memory error detector
==1861== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1861== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1861== Command: ./Tp_0.exe
==1861==
La direccion del archivo Origen es : Cin (Entrada Standar)
La direccion del archivo Destino es: Cout (Salida Standar)
Option -f is mandatory.
==1861==
==1861== HEAP SUMMARY:
==1861==     in use at exit: 0 bytes in 0 blocks
==1861==   total heap usage: 2 allocs, 2 frees, 73,728 bytes allocated
==1861==
==1861== All heap blocks were freed -- no leaks are possible
==1861==
==1861== For counts of detected and suppressed errors, rerun with: -v
==1861== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.12: Prueba con Valgrind 02

```

mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe -o utils/test.pgm -i NOEXISTE -f z
==1964== Memcheck, a memory error detector
==1964== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1964== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1964== Command: ./Tp_0.exe -o utils/test.pgm -i NOEXISTE -f z
==1964==
La direccion del archivo Destino es: utils/test.pgm
La direccion del archivo Origen es :NOEXISTE
cannot open NOEXISTE.
==1964==
==1964== HEAP SUMMARY:
==1964==     in use at exit: 0 bytes in 0 blocks
==1964==   total heap usage: 5 allocs, 5 frees, 83,024 bytes allocated
==1964==
==1964== All heap blocks were freed -- no leaks are possible
==1964==
==1964== For counts of detected and suppressed errors, rerun with: -v
==1964== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.13: Prueba con Valgrind 03

```

mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe -o utils/test.pgm
==1965== Memcheck, a memory error detector
==1965== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1965== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1965== Command: ./Tp_0.exe -o utils/test.pgm
==1965==
La direccion del archivo Destino es: utils/test.pgm
La direccion del archivo Origen es : Cin (Entrada Standar)
Option -f is mandatory.
==1965==
==1965== HEAP SUMMARY:
==1965==     in use at exit: 0 bytes in 0 blocks
==1965==   total heap usage: 4 allocs, 4 frees, 82,472 bytes allocated
==1965==
==1965== All heap blocks were freed -- no leaks are possible
==1965==
==1965== For counts of detected and suppressed errors, rerun with: -v
==1965== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.14: Prueba con Valgrind 04

```

==1965== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe -o utils/test.pgm -i utils/dragon.ascii.pgm
==1967== Memcheck, a memory error detector
==1967== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1967== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1967== Command: ./Tp_0.exe -o utils/test.pgm -i utils/dragon.ascii.pgm
==1967==
La direccion del archivo Destino es: utils/test.pgm
La direccion del archivo Origen es :utils/dragon.ascii.pgm
Option -f is mandatory.
==1967==
==1967== HEAP SUMMARY:
==1967==     in use at exit: 0 bytes in 0 blocks
==1967==   total heap usage: 7 allocs, 7 frees, 91,239 bytes allocated
==1967==
==1967== All heap blocks were freed -- no leaks are possible
==1967==
==1967== For counts of detected and suppressed errors, rerun with: -v
==1967== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.15: Prueba con Valgrind 05

```

mauro@mauro:~/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=yes ./Tp_0.exe -o utils/test.pgm -i utils/dragon.ascii.pgm -f exp\(\z\
==1970== Memcheck, a memory error detector
==1970== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1970== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1970== Command: ./Tp_0.exe -o utils/test.pgm -i utils/dragon.ascii.pgm -f exp\(\z\
==1970==
La direccion del archivo Destino es: utils/test.pgm
La direccion del archivo Origen es :utils/dragon.ascii.pgm
La transformacion elegida es f(z)=exp(z)
==1970==
==1970== HEAP SUMMARY:
==1970==     in use at exit: 0 bytes in 0 blocks
==1970==   total heap usage: 1,393 allocs, 1,393 frees, 2,724,396 bytes allocated
==1970==
==1970== All heap blocks were freed -- no leaks are possible
==1970==
==1970== For counts of detected and suppressed errors, rerun with: -v
==1970== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.16: Prueba con Valgrind 06

```

elias@Urquiza-Debian:~/Documentos/AlgoII/algoritmos-II-tp0/Tp_0/Debug$ valgrind --leak-check=full --show-leak-kinds=all ./Tp_0
./Tp_0 -i utils/saturn.ascii -o utils/out-saturn.pgm -f z
==2110== Memcheck, a memory error detector
==2110== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2110== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==2110== Command: ./Tp_0.exe -i utils/saturn.ascii -o utils/out-saturn.pgm -f z
==2110==
La dirección del archivo Origen es :utils/saturn.ascii
cannot open utils/saturn.ascii.
==2110==
==2110== HEAP SUMMARY:
==2110==     in use at exit: 19 bytes in 1 blocks
==2110==   total heap usage: 4 allocs, 3 frees, 74,299 bytes allocated
==2110==
==2110== 19 bytes in 1 blocks are still reachable in loss record 1 of 1
==2110==    at 0x4835DEF: operator new(unsigned long) (vg_replace_malloc.c:334)
==2110==      by 0x499657E: void std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::M_construct<char const*, char const*, std::forward_iterator_tag> (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25)
==2110==      by 0x10D580: cmdline::do_short_opt(char const*, char const*) (cmdline.cpp:195)
==2110==      by 0x10CF41: cmdline::parse(int, char* const*) (cmdline.cpp:81)
==2110==      by 0x10D87F: main (main.cpp:60)
==2110==
==2110== LEAK SUMMARY:
==2110==   definitely lost: 0 bytes in 0 blocks
==2110==   indirectly lost: 0 bytes in 0 blocks
==2110==   possibly lost: 0 bytes in 0 blocks
==2110==   still reachable: 19 bytes in 1 blocks
==2110==   suppressed: 0 bytes in 0 blocks
==2110==
==2110== For counts of detected and suppressed errors, rerun with: -v
==2110== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 6.17: Prueba con Valgrind 07 - Repetición de la Prueba 03 en otro sistema

Notamos que tenemos una memory leak de un bloque todavía alcanzable. Sospechamos que la fuente del mismo proviene del uso de `exit(1)`; puesto que ese comando, dependiendo del sistema, no llama a los destructores de toda instancia de clase construida. Al hacer las corridas notamos que en un sistema operativo distinto no se libera memoria (Prueba 7), en cambio en la (Prueba 3) no hay fuga de memoria.

7. Conclusión

A lo largo de este trabajo se pudieron implementar los conocimientos de clases adquiridos durante la primera parte de la cursada. Tuvimos que diseñar y trabajar con nuevas clases y esto nos permitió aprender a trabajar con la metodología de la programación orientada a objetos, con la cual trabajaremos durante el resto de la materia.

El uso de las clases permite abstraerse de la implementación y hacer el programa mas sencillo de leer. Por ejemplo, al poder sobrecargar o definir operadores, se hizo más sencillo poder trabajar con una imagen. También se simplifica el código al poder sobrecargar un operador para sumar objeto con objeto, objeto con número, etc. Lo cual mejora la legibilidad del mismo.

Finalmente, consideramos que el programa diseñado cumple satisfactoriamente con lo propuesto por el enunciado. Dejando lugar para que se pueda trabajar con más funciones, si así se quisiera, en el futuro. Por otro lado, también podríamos mejorar la portabilidad del programa.

8. Anexo

8.1. Enunciado

75.04/95.12 Algoritmos y Programación II

Trabajo práctico 0: Programación C++

Universidad de Buenos Aires - FIUBA
Primer cuatrimestre de 2020
\$Date: 2020/05/10 20:08:56 \$

1. Objetivos

Ejercitarse conceptos básicos de programación C++. 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 implementaremos una herramienta para procesar imágenes. Para ello, se deberá programar una clase que permita realizar operaciones básicas sobre las mismas:

- Cargar una imagen a memoria desde un archivo o desde la entrada estándar.
- Aplicar una función predefinida 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. Despues 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 ultimo 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 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 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 15 0 0 0 0
0 3 0 0 0 0 0 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 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 siempre un rectángulo de lado 2 centrado en el origen del plano complejo, con lo cual 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].

Funciones. La funciones a implementar en este TP son $f(z) = z$ y $f(z) = e^z$.

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 $f(z) = z$. $f(z) = e^z$ se corresponde con el valor de argumento `exp(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.

```
$ ./tp0 -i grid.pgm -o grid-id.pgm -f z
```

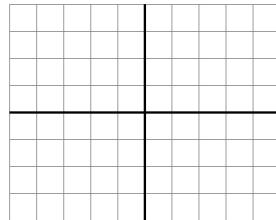


Figura 1: grid.pgm

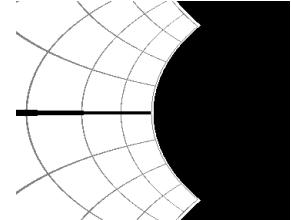


Figura 2: grid-exp.pgm

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

```
$ ./tp0 -i evolution.pgm -o evolution-exp.pgm -f exp(z)
```

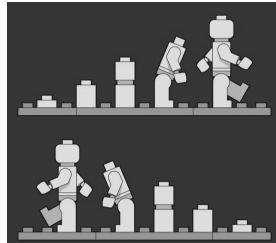


Figura 3: evolution.pgm

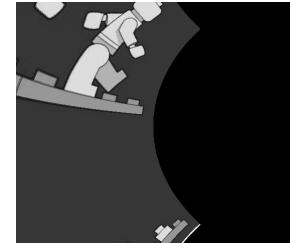


Figura 4: evolution-exp.pgm

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 tanto en Windows así como 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. Entrega de TPs

La entrega de este trabajo deberá realizarse usando el campus virtual de la materia [3]. Asimismo, en todos los casos, estas presentaciones deberán ser realizadas durante los días jueves. El *feedback* estará disponible de un jueves hacia el otro, como ocurre durante la modalidad presencial de cursada.

Por otro lado, la última fecha de entrega y presentación para esta trabajo será el jueves 28/5.

Referencias

- [1] Netpbm format (Wikipedia).
http://en.wikipedia.org/wiki/Netpbm_format
- [2] Holomorphic function (Wikipedia).
http://en.wikipedia.org/wiki/Holomorphic_function
- [3] Curso: 75.04/95.12 Algoritmos y Programación II - Curso P. Calvo.
<https://campus.fi.uba.ar/course/view.php?id=999>

8.2. Repositorio

Para una mejor organización de trabajo usamos un repositorio en github, a continuación dejamos el link para que se pueda acceder:

Repositorio

8.3. Código fuente

8.3.1. main.cpp

main.cpp

```
1 //=====
2 // Name      : main.cpp
3 // Author    : Nacachian, Urquiza, Vera
4 // Version   : 1.1.1
5 // Description : Trabajo Práctico Nro. 0
6 //=====
7
8
9 #include <iostream>
10 #include "cmdline.h"
11 #include "Images.h"
12 #include "Complejo.h"
13 #include "ComplexPlane.h"
14 #include "ComplexTransform.h"
15
16 using namespace std;
17
18
19 /*=====
20 //                      PROTOTIPOS
21 /*=====*/
22
23
24 static void opt_input(string const &);
25 static void opt_output(string const &);
26 static void opt_function(string const &);
27 static void opt_help(string const &);
28 void transformar_imagen(const Images & origen, Images & destino);
29
30
31 /*=====
32 //                      ELEMENTOS GLOBALES
33 /*=====*/
34
35 static option_t options[] = {
36     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
37     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
38     {1, "f", "function", NULL, opt_function, OPT_MANDATORY},
39     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
40     {0, },
41 };
42 //typedef enum token_functions {EXPZ, Z}; //, COSZ, SINZ };
43 const string functions[] = {"exp(z)", "z"}; //, "cos(z)", "sin(z)" };
44
45 static istream* iss = 0;
46 static ostream* oss = 0;
47 static fstream ifs;
48 static fstream ofs;
49
50
51 /*=====
52 //                      MAIN
53 /*=====*/
54
55 int main(int argc, char * const argv[]){
```

```

58 //-----Valido Argumentos -----//
59 cmdline cmdl(options);
60 cmdl.parse(argc, argv);
61
62 //-----Creo Imagenes de origen y destino -----//
63 Images origen;
64 origen.loadFile(iiss);
65 Images destino(origen);
66
67 //-----Trasnformo y guardo -----//
68 transformar_imagen(origen,destino);
69 destino.saveFile(oss);
70
71 if(iiss != &cin)
72     ifs.close();
73 if(oss != &cout)
74     ofs.close();
75
76 return 0;
77 }
78
79
80
81
82 /*=====
83 //          FUNCIONES INVOCADAS EN EL MAIN
84 =====*/
85
86
87 void transformar_imagen(const Images & origen, Images & destino){
88
89     if(&origen == &destino)
90         return;
91
92     Complejo z_aux(0,0);
93     ComplexPlane plano(origen);
94     ComplexTransform transformada;
95
96     int ancho = origen.getWidth();
97     int altura = origen.getHeight();
98
99     for(int i = 0; i < altura; i++){
100         for(int j = 0; j < ancho; j++){
101             //guarda la coordenada en forma de num complejo
102             plano.index2Comp(i, j);
103             z_aux = plano.getComp();
104
105             //calcula la anti transformada
106             transformada.fun(z_aux);
107             z_aux = transformada.getOutput();
108
109             //guarda los indices del pixel del origen
110             plano.comp2Index(z_aux);
111
112             //guarda vacío si algun subindice es negativo; caso contrario, guarda el pixel
113             if(plano.getRow() < 0 || plano.getCol() < 0)
114                 destino(i,j)=0;
115             else
116                 destino(i,j) = origen(plano.getRow(),plano.getCol());
117
118         }
119     }
120
121     return;
122 }
123
124 //----- Callbacks de CMDLINE -----//
125
126 static void
127 opt_input(string const &arg)

```

```

128 {
129     // Si el nombre del archivos es "-", usaremos la entrada
130     // est?dar. De lo contrario, abrimos un archivo en modo
131     // de lectura.
132     //
133     if (arg == "-") {
134         iss = &cin;      // Establezco la entrada estandar cin como flujo de entrada
135         cout<<"La direccion del archivo Origen es : Cin (Entrada Standar)" <<endl;
136     }
137     else {
138         ifs.open(arg.c_str(), ios::in); // c_str(): Returns a pointer to an array that
139                                         // contains a null-terminated
140                                         // sequence of characters (i.e., a C-string)
141         representing
142                                         // the current value of the string object.
143         iss = &ifs;
144         cout<<"La direccion del archivo Origen es :"<< arg.c_str() <<endl;
145     }
146
147     // Verificamos que el stream este OK.
148     //
149     if (!iss->good()) {
150         cerr << "cannot open "
151             << arg
152             << "."
153             << endl;
154         exit(1);
155     }
156 }
157
158 static void
159 opt_output(string const &arg)
160 {
161
162     // Si el nombre del archivos es "-", usaremos la salida
163     // est?dar. De lo contrario, abrimos un archivo en modo
164     // de escritura.
165     //
166     if (arg == "-") {
167         oss = &cout;      // Establezco la salida estandar cout como flujo de salida
168         cout<<"La direccion del archivo Destino es: Cout (Salida Standar)" << endl;
169     } else {
170         ofs.open(arg.c_str(), ios::out);
171         oss = &ofs;
172         cout<< "La direccion del archivo Destino es: "<< arg.c_str() <<endl;
173     }
174
175     // Verificamos que el stream este OK.
176     //
177     if (!oss->good()) {
178         cerr << "cannot open "
179             << arg
180             << "."
181             << endl;
182         exit(1);          // EXIT: Terminaci? del programa en su totalidad
183     }
184 }
185
186 static void
187 opt_function(string const &arg)
188 {
189     istringstream iss(arg);
190     cout<< "La transformacion elegida es f(z)= " <<arg.c_str() <<endl;
191
192     // Intentamos extraer el factor de la l?ea de comandos.
193     // Para detectar argumentos que ?nicamente consistan de
194     // n?meros enteros, vamos a verificar que EOF llegue justo
195     // desp? de la lectura exitosa del escalar.

```

```

196 // 
197     string option;
198     iss >> option;
199
200     if (iss.bad()) {
201         cerr << "cannot read integer factor."
202             << endl;
203         exit(1);
204     }
205
206     if( option.compare("z") == 0){
207         ComplexTransform::setOption(1);
208         //cout<< "Testpoint 1" <<endl;
209     }else if( option.compare("exp(z)") ==0 ) {
210         ComplexTransform::setOption(2);
211         //cout<< "Testpoint 2" <<endl;
212     }else{
213         cerr << option <<" No es un parametro Valido "<< endl;
214         exit(1);
215     }
216 }
217
218 static void
219 opt_help(string const &arg)
220 {
221     cout << "cmdline [-f function] [-i file] [-o file]"
222         << endl;
223     exit(0);
224 }
```

8.3.2. Complejo.h

Complejo.h

```

1 #ifndef COMPLEJOS_INCLUDED
2 #define COMPLEJOS_INCLUDED
3
4
5 #include <iostream> //lo dejamos?
6 #include <cmath>
7 #include <iomanip> // std::setprecision (para qué?)
8
9 class Complejo{
10 private:
11     double real;
12     double imag;
13     double abs; //valor absoluto
14     double arg; //argumento en radianes. Devuelve un valor entre -PI y PI
15
16     void setPolar();
17
18 public:
19     //constructores:
20     Complejo();
21     Complejo(double, double);
22     Complejo(const Complejo &);
23     //destructor:
24     ~Complejo();
25     //obtener atributos:
26     double getReal();
27     double getImag();
28     double getAbs();
29     double getArg();
30     //setear atributos:
31     void setReal(double);
32     void setImag(double);
33
34     //conjugado:
35     Complejo conjugado();
```

```

36
37     //emitter:
38     void printRect();
39     void printPolar();
40
41     //sobrecarga de operadores
42     Complejo operator+(const Complejo &); //suma entre Complejos
43     Complejo operator+(double); //suma con un real
44     friend Complejo operator+ (double, const Complejo &);
45
46     Complejo operator-(const Complejo &); //resta entre Complejos
47     Complejo operator-(double); //resta por un real
48     friend Complejo operator- (double, const Complejo &);
49
50     Complejo operator*(const Complejo &); //producto entre Complejos
51     Complejo operator*(double); //producto por un real (escalar)
52     friend Complejo operator* (double, const Complejo &);
53
54     Complejo operator/(const Complejo&); //división entre Complejos
55     Complejo operator/(double); //división por un real (escalar)
56
57
58     Complejo& operator=(const Complejo&); //asignación entre Complejos
59     bool operator==(const Complejo&); //comparación entre Complejos
60 };
61
62
63 #endif//COMPLEJOS_INCLUDED

```

8.3.3. Complejo.cpp

Complejo.cpp

```

1 #include "Complejo.h"
2
3 //constructores:
4 Complejo::Complejo() {
5     real = 0;
6     imag = 0;
7     abs = 0;
8     arg = 0;
9 }
10
11 Complejo::Complejo(double x, double y) {
12     real = x;
13     imag = y;
14     this->setPolar();
15 }
16
17 Complejo::Complejo(const Complejo & init){
18     real = init.real;
19     imag = init.imag;
20     abs = init.abs;
21     arg = init.arg;
22 }
23
24 //destructor:
25 Complejo::~Complejo() {
26     real = 0;
27     imag = 0;
28     abs = 0;
29     arg = 0;
30 }
31
32 //obtener atributos:
33 double Complejo::getReal(){
34     return real;
35 }

```

```

37 double Complejo::getImag() {
38     return imag;
39 }
40
41 double Complejo::getAbs() {
42     return abs;
43 }
44
45 double Complejo::getArg() {
46     return arg;
47 }
48
49 //setear atributos:
50 void Complejo::setReal(double val) {
51     real=val;
52     setPolar();
53 }
54
55 void Complejo::setImag(double val) {
56     imag=val;
57     setPolar();
58 }
59
60 void Complejo::setPolar(){
61     abs=sqrt(pow(real,2) + pow(imag,2));
62     arg=atan2(imag, real); //calcula atan2(imag/real) y devuelve un valor entre -PI y PI
63 }
64
65
66 //conjugar
67 Complejo Complejo::conjugar(){
68     Complejo aux(real, -imag);
69     return aux;
70 }
71
72
73 //emitir
74 void Complejo::printRect(){
75     std::cout << real << "+j(" << imag << ')' << std::endl;
76 }
77
78 void Complejo::printPolar(){
79     std::cout << abs << "*exp(j" << arg << ')' << std::endl;
80 }
81
82
83 //sobrecarga de operadores:
84 //suma
85 Complejo Complejo::operator+(const Complejo & right){
86     Complejo aux(real + right.real, imag + right.imag);
87     return aux;
88 }
89
90 Complejo Complejo::operator+(double right){
91     Complejo aux(real + right, imag);
92     return aux;
93 }
94
95 Complejo operator+ (double left, const Complejo & right){
96     return Complejo(right.real + left, right.imag);
97 }
98
99 //resta
100 Complejo Complejo::operator-(const Complejo & right){
101     Complejo aux(real - right.real, imag - right.imag);
102     return aux;
103 }
104
105 Complejo Complejo::operator-(double right){
106     Complejo aux(real - right, imag);

```

```

107     return aux;
108 }
109
110 Complejo operator- (double left, const Complejo & right){
111     return Complejo(left - right.real, - right.imag );
112 }
113
114 //producto
115 Complejo Complejo::operator*(const Complejo & right){
116     Complejo aux(real*right.real - imag*right.imag, real*right.imag + imag*right.real);
117     return aux;
118 }
119
120 Complejo Complejo::operator*(double right){
121     Complejo aux(right*real, right*imag);
122     return aux;
123 }
124
125 Complejo operator* (double left, const Complejo & right){
126     return Complejo(left * right.real, left * right.imag);
127 }
128
129
130 //división
131 Complejo Complejo::operator/ (const Complejo & right){
132     double a = (real*right.real - imag*right.imag)/pow(right.abs,2);
133     double b = (real*right.imag + imag*right.real)/pow(right.abs,2);
134     Complejo aux(a, b);
135     return aux;
136 }
137
138 Complejo Complejo::operator/ (double right){
139     Complejo aux(real/right, imag/right);
140     return aux;
141 }
142
143 //asignación
144 Complejo& Complejo::operator=(const Complejo & right){
145     real = right.real;
146     imag = right.imag;
147     abs = right.abs;
148     return *this;
149 }
150
151 //comparación
152 bool Complejo::operator==(const Complejo & right){
153     if(real == right.real && imag == right.imag)
154         return true;
155
156     return false;
157 }
```

8.3.4. ComplexPlane.h

ComplexPlane.h

```

1 #ifndef COMPLEXPLANE_H_
2 #define COMPLEXPLANE_H_
3
4 #include "Complejo.h"
5 #include "Images.h"
6
7 class ComplexPlane {
8 private:
9     Complejo comp; //número complejo guardado
10    Images image; //imagen utilizada
11    int row; //fila y
12    int col; //columna guardadas para el número complejo entregado
13 }
```

```

14 public:
15     ComplexPlane();
16     ComplexPlane(const Images &); //guarda los datos de una imagen para tener sus dimensiones
17     ~ComplexPlane(); //destructor
18     ComplexPlane(const ComplexPlane &); // constructor copia
19     ComplexPlane& operator=(const ComplexPlane &); //sobrecarga de la asignación
20
21
22     Complejo getComp() const;
23     Images getImage() const;
24     int getRow() const;
25     int getCol() const;
26
27     void index2Comp(int, int);
28     void comp2Index(Complejo &);
29
30 };
31
32 #endif /* COMPLEXPLANE_H_ */

```

8.3.5. ComplexPlane.cpp

ComplexPlane.cpp

```

1 #include "ComplexPlane.h"
2 using namespace std;
3
4 ComplexPlane::ComplexPlane() {
5     row = col = 0;
6     this->comp.setReal(0);
7     this->comp.setImag(0);
8 }
9
10 ComplexPlane::ComplexPlane(const Images & original) {
11     row = col = 0;
12     this->image = original;
13     this->comp.setReal(0);
14     this->comp.setImag(0);
15 }
16
17 ComplexPlane::~ComplexPlane() {
18 }
19
20
21 ComplexPlane::ComplexPlane(const ComplexPlane & init) {
22     this->comp = init.getComp();
23     this->image = init.getImage();
24     this->row = init.getRow();
25     this->col = init.getCol();
26 }
27
28 ComplexPlane& ComplexPlane::operator=(const ComplexPlane & right) {
29     comp = right.getComp();
30     image = right.getImage();
31     row = right.getRow();
32     col = right.getCol();
33     return *this;
34 }
35
36 Complejo ComplexPlane::getComp() const {
37     return this->comp;
38 }
39
40 Images ComplexPlane::getImage() const {
41     return this->image;
42 }
43
44 int ComplexPlane::getRow() const {
45     return this->row;

```

```

46 }
47
48 int ComplexPlane::getCol() const {
49     return this->col;
50 }
51
52 //guarda en this->comp un número complejo correspondiente a la posición solicitada de la
53 //imagen.
54 void ComplexPlane::index2Comp(int row, int col) {
55     Complejo z(0,0);
56
57     if(0 > row || row >= image.getHeight() || 0 > col || col >= image.getWidth()) //valor
58         inválido
59         this->comp = z;
60
61     else
62     {
63         /* Se divide la matriz en cuatro regiones:
64          se calcula el valor de la fila y columna "central" de la matriz.
65          se reduce a un número entero puesto que, si se tiene una matriz cuadrada,
66          no existirá un centro alcanzable. Pero igualmente se puede operar corriendo
67          dicho centro un paso hacia atrás.
68         */
69         int rowCentral = this->image.getHeight()/2;
70         int colCentral = this->image.getWidth()/2;
71         this->row = row;
72         this->col = col;
73
74         if(row < rowCentral)
75             z.setImag( (1-(double)row/rowCentral) );
76
77         if(row >= rowCentral)
78             z.setImag( (-1 + (double)((this->image.getHeight() - 1) - row) / (double)rowCentral) );
79     };
80
81         if(col < colCentral)
82             z.setReal( (-1+(double)col/colCentral) );
83
84         if(col >= colCentral)
85             z.setReal( (1 - (double)((this->image.getWidth()-1) - col) / (double)colCentral) );
86
87         this->comp = z;
88     }
89 }
90
91 //guarda en row y col los índices asociados al número complejo entregado
92 void ComplexPlane::comp2Index(Complejo & z) {
93     double real = z.getReal();
94     double imaginario = z.getImag();
95
96     //si sale por fuera del plano, se guardan valores negativos de col y row
97     if(abs(real) > 1 || abs(imaginario) > 1)
98     {
99         this->row = -1;
100        this->col = -1;
101    }
102
103    else
104    {
105        int rowCentral = this->image.getHeight()/2;
106        int colCentral = this->image.getWidth()/2;
107        this->comp = z;
108
109        //se aplican las funciones inversas a las utilizadas en index2comp();
110        if(real <= 0)
111            this->col = std::round(colCentral*(real+1));
112    }

```

```

113
114     if(real > 0)
115         this->col = std::round((real-1)*colCentral + (this->image.getWidth()-1));
116
117     if(imaginario >= 0)
118         this->row = std::round(rowCentral*(1-imaginario));
119
120     if(imaginario < 0)
121         this->row = std::round(this->image.getHeight()-1) - (imaginario+1)*rowCentral;
122 }
123 }
```

8.3.6. ComplexTransform.h

ComplexTransform.h

```

1 #ifndef COMPLEXTRANSFORM_H_
2 #define COMPLEXTRANSFORM_H_
3
4 #include "Complejo.h"
5
6 class ComplexTransform{
7 private:
8     Complejo input;
9     Complejo output;
10    int option; //1: z ; 2: exp(z) ;3 rotar, en otro caso: z
11    static int userOption;
12
13 public:
14     ComplexTransform(); //constructor
15     ComplexTransform(int ); //con parámetros
16     ~ComplexTransform(); //destructor
17
18     void fun(Complejo &);
19
20     Complejo getInput();
21     Complejo getOutput();
22
23     static void setOption(int );
24     intgetOption( void );
25
26 };
27
28#endif
```

8.3.7. ComplexTransform.cpp

ComplexTransform.cpp

```

1 #include "ComplexTransform.h"
2
3
4 int ComplexTransform::userOption = 0;
5
6 ComplexTransform::ComplexTransform() {
7     this->option = userOption;
8 }
9
10 ComplexTransform::ComplexTransform(int i) {
11     this->option=i;
12 }
13
14 ComplexTransform::~ComplexTransform() {
15 /*No solicita memoria dinámica*/
16 }
```

```

18 void ComplexTransform::fun(Complejo & input) {
19     switch(option) {
20         case 1:
21             this->input=input;
22             this->output=input;
23             break;
24         case 2:
25             this->input=input;
26             this->output.setReal(exp(input.getReal())*cos(input.getImag()));
27             this->output.setImag(exp(input.getReal())*sin(input.getImag()));
28             break;
29         case 3:
30             this->input=input;
31             this->output.setReal(-1*input.getReal());
32             this->output.setImag(-1*input.getImag());
33             break;
34     default:
35         this->input=input;
36         this->output=input;
37     }
38 }
39
40
41 Complejo ComplexTransform::getInput() {
42     return this->input;
43 }
44
45 Complejo ComplexTransform::getOutput() {
46     return this->output;
47 }
48
49 void ComplexTransform::setOption(int option)
50 {
51     ComplexTransform::userOption = option;
52 }
53
54 int ComplexTransform::getOption(void)
55 {
56     return this->option;
57 }

```

8.3.8. Images.h

Images.h

```

1 /*
2  Clase Images.
3  En ella se guardan los datos de una imagen con formato .pgm
4 */
5
6 #ifndef IMAGES_H_
7 #define IMAGES_H_
8
9 // Librerias para manejo de archivos
10 #include <sstream>
11 #include <iostream>
12 #include <fstream>
13 #include <iomanip>
14
15 #include <string.h>
16 #include <cstdlib>
17
18
19 class Images {
20 private:
21     // Ancho de la imagen
22     int width;
23
24     // Altura de la imagen

```

```

25     int height;
26
27     // Intensidad maxima de la imagen
28     int maxInt;
29
30     // Matriz donde se almacenan los valores de intensidad
31     int ** imagen;
32
33     // El header. Si es P2, el formato es pgm
34     std::string magicNumber;
35
36 public:
37     // Constructor por defecto, inicializa en 0
38     Images();
39
40     // Constructor, se le pasa como argumento, ancho, alto y maxima intensidad
41     Images(int, int, int);
42
43     // Destructor, debera liberar la memoria de la matriz.
44     ~Images();
45
46     // Constructor por copia
47     Images(const Images &other);
48
49     // Sobreacarga del operador =
50     const Images& operator=(const Images &other);
51
52     // Sobreacarga del operador []
53     int & operator[](const std::pair<int,int> &);
54
55     // Sobrecarga operador ()
56     //
57     int & operator()(const int &, const int &);
58     const int & operator()(const int &, const int &) const;
59
60     // Funcion para procesar cada linea del archivo con formato .pgm (la usa el metodo de
61     // loadFile)
62     friend bool pgmParser(int &, int &, int &, std::stringstream * , Images * );
63
64     // Carga un archivo .pgm.
65     const Images & loadFile(std::istream * );
66
67     // Se guarda la instancia de Images en un archiv formato .pgm
68     const Images & saveFile(std::ostream * );
69
70     // Obtencion de los atributos
71     int getMaxInt() const;
72     int getWidth() const;
73     int getHeight() const;
74     std::string getMagicNumber() const;
75     int ** getColours() const;
76
77     // Imprime la matriz
78     void printColours();
79
80     // Devuelve true si la instanciacione almacenado datos de una imagen pgm
81     bool isPGM();
82 };
83 #endif /* IMAGES_H_ */

```

8.3.9. Images.cpp

Images.cpp

```

1
2 #include "Images.h"
3 #include <cstdlib>
4

```

```

5  using namespace std;
6
7 Images::Images() {
8     // Valores nulos, puesto que no se tiene imagen alguna
9     //
10    this->width = 0;
11    this->height = 0;
12    this->maxInt = 0;
13
14    // El puntero apunta a null.
15    //
16    this->imagen = NULL;
17
18    // El header esta vacio
19    this->magicNumber = "";
20 }
21
22 Images::Images(int width, int height, int max) {
23
24     if(width <= 0 || height <= 0){
25         // Valores por defecto: (Los elegi sin criterio)
26         //
27         this->width = 10;
28         this->height = 10;
29         this->maxInt = 255;
30
31     } else
32     {
33         this->width = width;
34         this->height = height;
35         this->maxInt = max;
36     }
37
38     // Pido memoria para la matriz.
39     //
40     this->imagen = new int * [this->height];
41
42     for(int filas = 0; filas < height; filas++)
43         this->imagen[filas] = new int[width];
44
45     // Inicializo la matriz con 0's
46     //
47     for(int filas = 0; filas < height; filas++)
48         for(int cols = 0; cols < width; cols++)
49             this->imagen[filas][cols] = 0;
50
51     // El nombre del header queda en blanco
52     this->magicNumber = "";
53 }
54
55 Images::~Images() {
56     // Destructor, debe eliminar la memoria pedida
57     //
58     for(int i = 0; i < height; i++)
59         delete [] this->imagen[i];
60     delete [] this->imagen;
61
62     // Pongo todos los valores en 0
63     this->width = 0;
64     this->height = 0;
65     this->maxInt = 0;
66     this->magicNumber = "";
67 }
68
69 Images::Images(const Images &other) {
70     // Constructor copia. Le asigna todos los atributos iguales a other, y
71     // Pide memoria para una nueva matriz
72     //
73
74 }
```

```

75     this->width = other.width;
76     this->height = other.height;
77     this->maxInt = other.maxInt;
78     this->magicNumber = other.magicNumber;
79
80     // Pido memoria p/ la matriz
81     //
82     this->imagen = new int * [this->height];
83     for(int filas = 0; filas < height; filas++)
84         this->imagen[filas] = new int[width];
85
86     // Inicializo la matriz con los valores de other
87     //
88     for(int filas = 0; filas < height; filas++)
89         for(int cols = 0; cols < width; cols++)
90             this->imagen[filas][cols] = other.imagen[filas][cols];
91 }
92
93 const Images& Images::operator=(const Images &other) {
94     // Si son iguales, no hace nada.
95     //
96     if(this == &other)
97         return *this;
98
99     for(int i = 0; i < this->height; i++)
100        delete [] this->imagen[i];
101    delete [] this->imagen;
102
103    this->width = other.width;
104    this->height = other.height;
105    this->maxInt = other.maxInt;
106    this->magicNumber = other.magicNumber;
107
108    // Pido memoria p/ la matriz
109    //
110    this->imagen = new int * [this->height];
111    for(int filas = 0; filas < height; filas++)
112        this->imagen[filas] = new int[width];
113
114    // Inicializo la matriz
115    //
116    for(int filas = 0; filas < height; filas++)
117        for(int cols = 0; cols < width; cols++)
118            this->imagen[filas][cols] = other.imagen[filas][cols];
119
120    return *this;
121 }
122
123
124 int & Images::operator[](const std::pair<int,int> & index){
125
126     // Le asigno a row el primer valor de <int,int> (fila)
127     // Y a col el segundo valor (columna)
128     //
129     int row = index.first;
130     int col = index.second;
131
132     // Si los indices estan afuera devuelve el valor de [0][0]
133     if( (row < 0 || row >= this->height) && (col < 0 || col >= this->width) )
134         return this->imagen[0][0];
135
136     return this->imagen[row][col];
137 }
138
139 int & Images::operator()(const int & row, const int & col){
140
141     if(row < 0 || col < 0 || row >= this->height || col >= this->width)
142         return imagen[0][0];
143
144     return (this->imagen[row][col]);

```

```

145 }
146
147 const int & Images::operator()(const int & row, const int & col) const{
148
149     if(row < 0 || col < 0 || row >= this->height || col >= this->width)
150         return imagen[0][0];
151
152     return(this->imagen[row][col]);
153 }
154
155
156
157 int Images::getWidth() const{
158     return this->width;
159 }
160
161 int Images::getMaxInt() const{
162     return this->maxInt;
163 }
164
165 int Images::getHeight() const{
166     return this->height;
167 }
168
169 int ** Images::getColours() const{
170     return this->imagen;
171 }
172
173 std::string Images::getMagicNumber() const{
174     return this->magicNumber;
175 }
176
177 bool Images::isPGM(){
178     if(magicNumber == "P2")
179         return true;
180     return false;
181 }
182
183 void Images::printColours(){
184     for(int i = 0; i < this->height; i++){
185         for(int j = 0; j < this->width; j++){
186             std::cout << std::left << std::setw(2) << this->imagen[i][j];
187         }
188         cout << endl;
189     }
190 }
191
192 /*-----*/
193 /*----- Manejo de archivos -----*/
194
195 bool pgmParser(int & nline, int & nfils, int & ncols, std::stringstream * ss , Images * image
196 ) {
197     if(ss->peek() == EOF)
198         return true;
199
200     // Chequea si en la linea a leer existe el caracter #. En caso de que exista corta el
201     // proceso.
202     size_t npos = string::npos;
203     if (ss->str().find('#') != npos){
204         nline--;
205         return true;
206     }
207
208     // Primera linea, el header
209     //
210     if(nline == 1){
211         image->magicNumber = ss->str();
212
213         // Elimina el \n o \r que en algunos .pgm aparece

```

```

213     // 
214     for (size_t i = 0; i < image->magicNumber.length(); i++) {
215         if (
216             image->magicNumber.c_str()[i] == '\r'
217             || image->magicNumber.c_str()[i] == '\n'
218             || image->magicNumber.c_str()[i] == '\r\n') {
219
220             image->magicNumber = image->magicNumber.substr(0, i);
221         }
222
223         // Si no es P2, la funcion devuelve falso. Y en ese caso debera cortar la carga del
224         // archivo ya que no estamos
225         // interesados en otro tipo de formato.
226         //
227         if( image->magicNumber.compare("P2") != 0 ){
228             cerr << "Formato no .pgm" << endl;
229             return false;
230         }
231         return true;
232     }
233
234     // Segunda linea. Primero elimina la matriz cargada por el constructor. Luego lee los
235     // nuevos tama;os
236     // Por ultimo pide memoria para la matriz con los nuevos tamalios
237     //
238     if(nline == 2){
239         for(int i = 0; i < image->height; i++)
240             delete [] image->imagen[i];
241         delete [] image->imagen;
242
243         *ss >> image->width >> image->height;
244
245         image->imagen = new int * [image->height];
246
247         for(int filas = 0; filas < image->height; filas++)
248             image->imagen[filas] = new int[image->width];
249
250         // Inicializo la matriz con 0's
251         //
252         for(int filas = 0; filas < image->height; filas++)
253             for(int cols = 0; cols < image->width; cols++)
254                 image->imagen[filas][cols] = 0;
255
256         return true;
257     }
258
259     // Tercera linea, lee el maximo brillo.
260     //
261     if(nline == 3){
262         *ss >> image->maxInt;
263         return true;
264     }
265
266     // Comienza a leer los colores
267     //
268     while( nfils < image->height && !ss->eof() ) {
269         for( (ncols < image->width) && !ss->eof(); ncols++) {
270             if( !( *ss >> image->imagen[nfils][ncols]) )
271                 ncols--;
272         }
273         // Esta verificacion es por si el for corto por el lado de que se llego a eof
274         if(ncols == image->width){
275             ncols = 0;
276             nfils++;
277         }
278     }
279
280     return true;
281 }
282
283 const Images & Images::loadFile(std::istream * image) {

```

```

281     if( !(image->good()) ){
282         cerr << "Fallo al abrir el archivo" << endl;
283         //return *this;
284         exit(1);
285     }
286
287     // line es el string donde se almacenara la linea leida en cuestion.
288     // nline es el numero de linea. La primer linea es donde esta el magic number, la segunda
289     // linea
290     // es donde esta la altura y ancho y la tercera linea el maximo brillo.
291     // Todas las lineas siguientes corresponden a los valores de intensidad de la imagen. Si
292     // hay un comentario
293     // se ignora esa linea en cuestion.
294     //
295     string line;
296     int nline = 1;
297     int nfils = 0;
298     int ncols = 0;
299
300     // Lee linea por linea y la almacena en string, hasta encontrar eof.
301     // Luego se llama a la funcion pgmParse, la cual se encargara de procesar que linea es en
302     // cuestion
303     // y guardar los valores en la clase imagen. Notar que en esa funcion si se encuentra un
304     // comentario
305     // nline no aumenta.
306     //
307     while( getline(*image, line) ){
308         stringstream ss(line);
309         if(!pgmParser(nline, nfils, ncols, &ss, this))
310             return *this;
311         nline++;
312     }
313 }
314
315 const Images & Images::saveFile(ostream * image){
316
317     if(!image->good()){
318         cerr << "Fallo al abrir el archivo" << endl;
319         return *this;
320     }
321
322     *image << this->magicNumber << endl;
323     *image << "# Imagen .pgm generada desde tp0.exe" << endl;
324     *image << this->width << " " << this->height << endl;
325     *image << this->maxInt << endl;
326
327     for(int fils = 0; fils < this->height; fils++){
328         for(int cols = 0; cols < this->width; cols++){
329             *image << this->imagen[fils][cols] << endl;
330         }
331     }
332
333     return *this;
334 }
```

8.3.10. cmdline.h

cmdline.h

```

1 #ifndef _CMDLINE_H_INCLUDED_
2 #define _CMDLINE_H_INCLUDED_
3
4 #include <string>
5 #include <iostream>
6
```

```

7 #define OPT_DEFAULT    0
8 #define OPT_SEEN       1
9 #define OPT_MANDATORY 2
10
11 struct option_t {
12     int has_arg;
13     const char *short_name;
14     const char *long_name;
15     const char *def_value;
16     void (*parse)(std::string const &); // Puntero a funcin de opciones
17     int flags;
18 };
19
20 class cmdline {
21     // Este atributo apunta a la tabla que describe todas
22     // las opciones a procesar. Por el momento, slo puede
23     // ser modificado mediante contructor, y debe finalizar
24     // con un elemento nulo.
25     //
26     option_t *option_table;
27
28     // El constructor por defecto cmdline::cmdline(), es
29     // privado, para evitar construir "parsers" (analizador
30     // sintctico, recibe una palabra y lo interpreta en
31     // una accin dependiendo su significado para el programa)
32     // sin opciones. Es decir, objetos de esta clase sin opciones.
33     //
34
35     cmdline();
36     int do_long_opt(const char *, const char *);
37     int do_short_opt(const char *, const char *);
38 public:
39     cmdline(option_t *);
40     void parse(int, char * const []);
41 };
42
43 #endif

```

8.3.11. cmdline.cpp

cmdline.cpp

```

1 // cmdline - procesamiento de opciones en la lnea de comando.
2 //
3 // $Date: 2012/09/14 13:08:33 $
4 //
5 #include <string>
6 #include <cstdlib>
7 #include <iostream>
8 #include "cmdline.h"
9
10 using namespace std;
11
12 cmdline::cmdline()
13 {
14     option_table = NULL;
15 }
16
17 cmdline::cmdline(option_t *table) : option_table(table)
18 {
19     /*
20     - Lo mismo que hacer:
21
22     option_table = table;
23
24     Siendo "option_table" un atributo de la clase cmdline
25     y table un puntero a objeto o struct de "option_t".
26
27     Se estaría contruyendo una instancia de la clase cmdline

```

```

28     cargandole los datos que se hayan en table (la table con
29     las opciones, ver el código en main.cc)
30
31     */
32 }
33
34 void
35 cmdline::parse(int argc, char * const argv[])
36 {
37 #define END_OF_OPTIONS(p)          \
38     ((p)->short_name == 0          \
39     && (p)->long_name == 0          \
40     && (p)->parse == 0)
41
42 // Primer pasada por la secuencia de opciones: marcamos
43 // todas las opciones, como no procesadas. Ver código de
44 // abajo.
45 //
46 for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
47     op->flags &= ~OPT_SEEN;
48
49 // Recorremos el arreglo argv. En cada paso, vemos
50 // si se trata de una opción corta, o larga. Luego,
51 // llamamos a la función de parseo correspondiente.
52 //
53 for (int i = 1; i < argc; ++i) {
54     // Todos los parámetros de este programa deben
55     // pasarse en forma de opciones. Encontrar un
56     // parámetro no-opción es un error.
57     //
58     if (argv[i][0] != '-')
59         cerr << "Invalid non-option argument: "
60         << argv[i]
61         << endl;
62     exit(1);
63 }
64
65 // Usamos "--" para marcar el fin de las
66 // opciones; todos los argumentos que puedan
67 // estar a continuación no son interpretados
68 // como opciones.
69 //
70 if (argv[i][1] == '-')
71     && argv[i][2] == 0)
72     break;
73
74 // Finalmente, vemos si se trata o no de una
75 // opción larga; y llamamos al método que se
76 // encarga de cada caso.
77 //
78 if (argv[i][1] == '-')
79     i += do_long_opt(&argv[i][2], argv[i + 1]);
80 else
81     i += do_short_opt(&argv[i][1], argv[i + 1]);
82 }
83
84 // Segunda pasada: procesamos aquellas opciones que,
85 // (1) no hayan figurado explícitamente en la línea
86 // de comandos, y (2) tengan valor por defecto.
87 //
88 for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
89 #define OPTION_NAME(op) \
90     (op->short_name ? op->short_name : op->long_name)
91     if (op->flags & OPT_SEEN)
92         continue;
93     if (op->flags & OPT_MANDATORY) {
94         cerr << "Option "
95         << "_"
96         << OPTION_NAME(op)
97         << " is mandatory."

```

```

98         << "\n";
99     exit(1);
100 }
101 if (op->def_value == 0)
102     continue;
103 op->parse(string(op->def_value));
104 }
105 }
106
107 int
108 cmdline::do_long_opt(const char *opt, const char *arg)
109 {
110     // Recorremos la tabla de opciones, y buscamos la
111     // entrada larga que se corresponda con la opción de
112     // linea de comandos. De no encontrarse, indicamos el
113     // error.
114     //
115     for (option_t *op = option_table; op->long_name != 0; ++op) {
116         if (string(opt) == string(op->long_name)) {
117             // Marcamos esta opción como usada en
118             // forma explícita, para evitar tener
119             // que inicializarla con el valor por
120             // defecto.
121             //
122             op->flags |= OPT_SEEN;
123
124             if (op->has_arg) {
125                 // Como se trata de una opción
126                 // con argumento, verificamos que
127                 // el mismo haya sido provisto.
128                 //
129                 if (arg == 0) {
130                     cerr << "Option requires argument: "
131                         << "--"
132                         << opt
133                         << "\n";
134                     exit(1);
135                 }
136                 op->parse(string(arg));
137                 return 1;
138             } else {
139                 // Opción sin argumento.
140                 //
141                 op->parse(string(""));
142                 return 0;
143             }
144         }
145     }
146
147     // Error: opción no reconocida. Imprimimos un mensaje
148     // de error, y finalizamos la ejecución del programa.
149     //
150     cerr << "Unknown option: "
151         << "--"
152         << opt
153         << "."
154         << endl;
155     exit(1);
156
157     // Algunos compiladores se quejan con funciones que
158     // lógicamente no pueden terminar, y que no devuelven
159     // un valor en esta última parte.
160     //
161     return -1;
162 }
163
164 int
165 cmdline::do_short_opt(const char *opt, const char *arg)
166 {
167     option_t *op;

```

```

168
169 // Recorremos la tabla de opciones, y buscamos la
170 // entrada corta que se corresponda con la opción de
171 // línea de comandos. De no encontrarse, indicamos el
172 // error.
173 //
174 for (op = option_table; op->short_name != 0; ++op) {
175     if (string(opt) == string(op->short_name)) {
176         // Marcamos esta opción como usada en
177         // forma explícita, para evitar tener
178         // que inicializarla con el valor por
179         // defecto.
180         //
181         op->flags |= OPT_SEEN;
182
183     if (op->has_arg) {
184         // Como se trata de una opción
185         // con argumento, verificamos que
186         // el mismo haya sido provisto.
187         //
188         if (arg == 0) {
189             cerr << "Option requires argument: "
190             << "_"
191             << opt
192             << "\n";
193             exit(1);
194         }
195         op->parse(string(arg));
196         return 1;
197     } else {
198         // Opción sin argumento.
199         //
200         op->parse(string(""));
201         return 0;
202     }
203 }
204
205
206 // Error: opción no reconocida. Imprimimos un mensaje
207 // de error, y finalizamos la ejecución del programa.
208 //
209 cerr << "Unknown option: "
210     << "_"
211     << opt
212     << "."
213     << endl;
214 exit(1);
215
216 // Algunos compiladores se quejan con funciones que
217 // lógicamente no pueden terminar, y que no devuelven
218 // un valor en esta última parte.
219 //
220 return -1;
221 }

```

8.4. Imágenes de Prueba y Transformaciones

8.4.1. Test 4x4

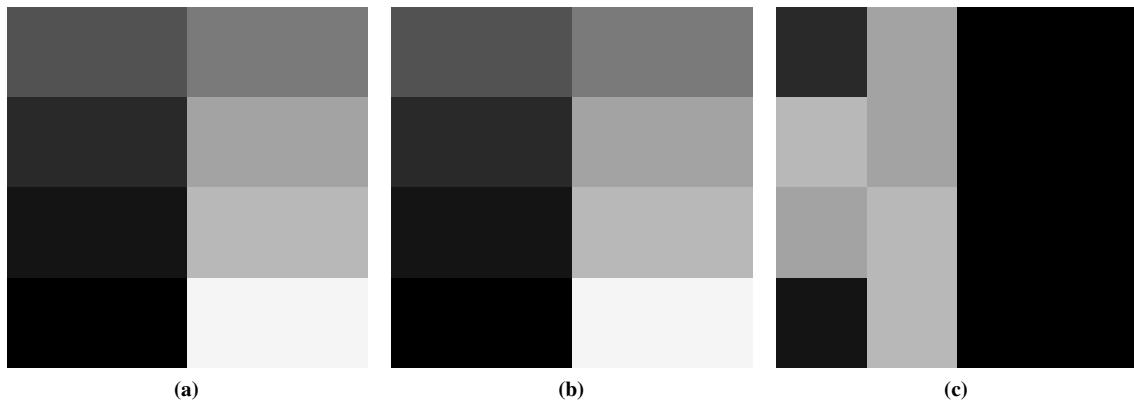


Figura 8.1: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.2. Test 16x16

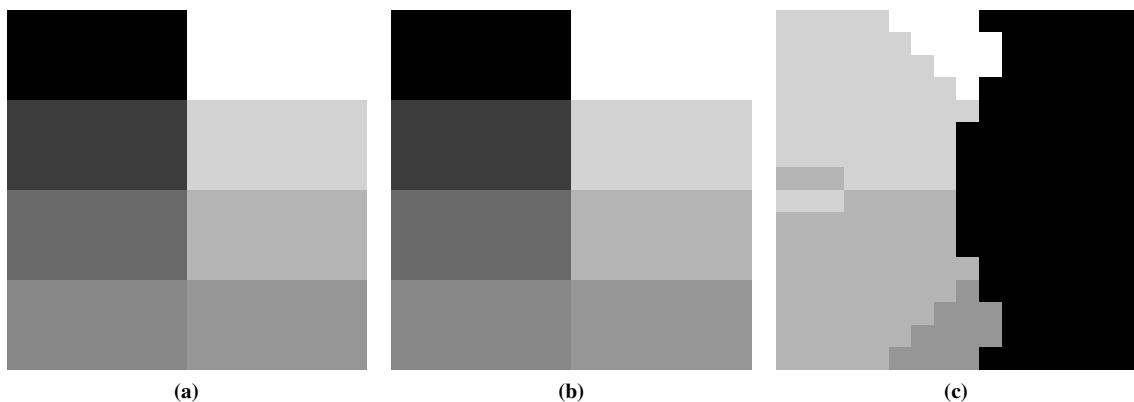


Figura 8.2: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.3. FEEP

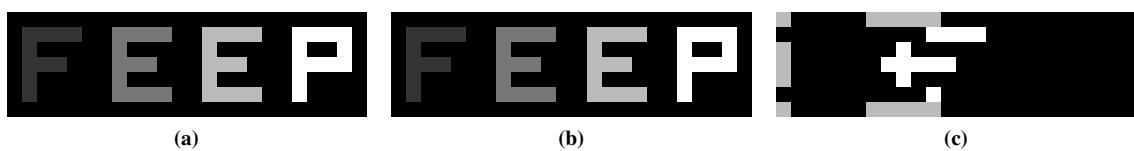


Figura 8.3: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.4. Saturn

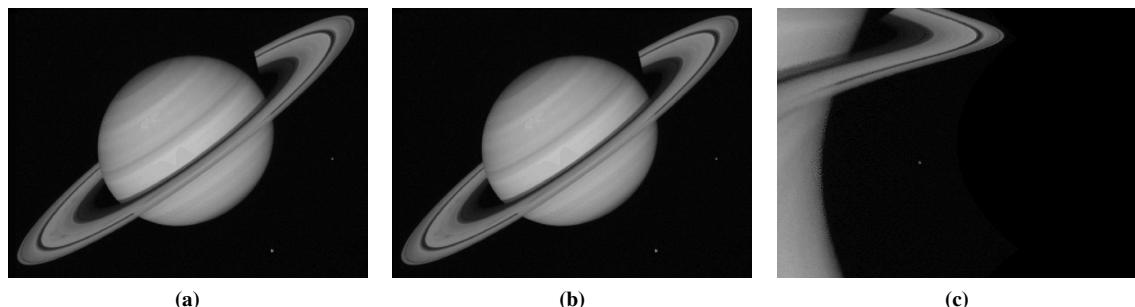


Figura 8.4: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.5. Lena



Figura 8.5: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.6. Marcie



Figura 8.6: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.7. Fractal Tree

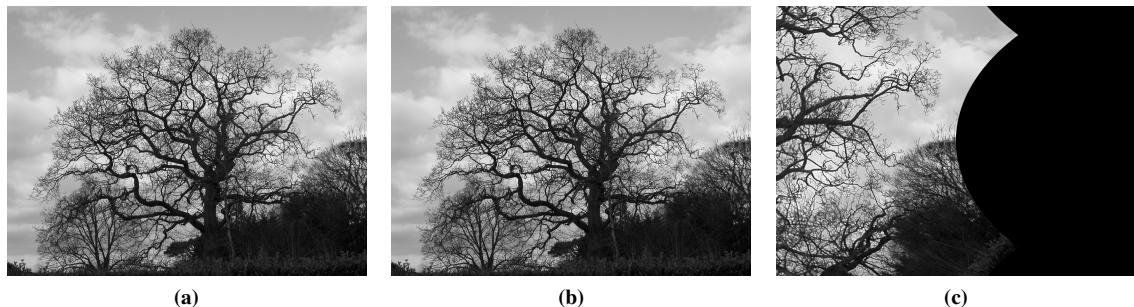


Figura 8.7: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.8. Venus

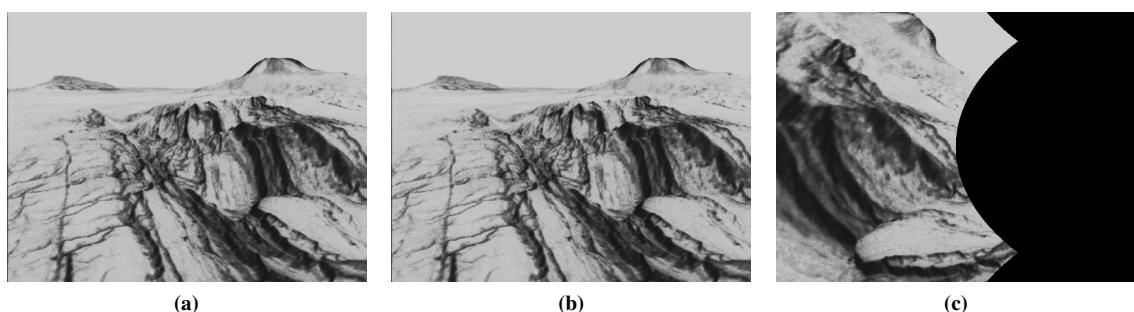


Figura 8.8: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$

8.4.9. Casablanca



Figura 8.9: (a) Entrada (b) $f(z) = z$ (c) $f(z) = \exp(z)$