



Facultad de Ingeniería
Universidad de Buenos Aires
Algoritmos y programación 2 (95.12)

Trabajo Práctico N°0

1^{er} Cuatrimestre, 2020

Carosella Grau, Juan Manuel	97895	jcarosella@fi.uba.ar
Neumarkt Fernández, Leonardo	97471	lneumarkt@fi.uba.ar
Turqueto, Bernardo	98036	bturqueto@fi.uba.com

Entrega	Fecha	Correcciones
1	28/6	

Índice

1. Introducción	2
2. Desarrollo	2
2.1. Clase complejo	2
2.2. Clase Image	3
2.3. Formato PGM	3
2.4. Implementación	3
2.5. Compilación y ejecución	5
3. Corridas de prueba	6
4. Conclusiones	11
5. Código fuente	11
5.1. main.cpp	11
5.2. main.h	20
5.3. image.cpp	21
5.4. image.h	24
5.5. complejo.cpp	25
5.6. complejo.h	29
5.7. Makefile	30
6. Enunciado	30

1. Introducción

El objetivo de este trabajo práctico es ejercitarse con conceptos básicos de programación C++. Para ello se implementará una herramienta para procesar imágenes. 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.

El manejo de archivos se realizará a través de la línea de comando, pasando como argumentos los archivos de entrada y salida, y se indicará el tipo de función a transformar. En este trabajo se buscará transformar la imagen inicialmente con una función exponencial, y luego se agregarán otras a elección nuestra.

2. Desarrollo

2.1. Clase complejo

Como se va a trabajar con números de tipo complejo, se creó la clase complejo, la cual tiene como atributos privados la parte real y la parte imaginaria. Luego se crearon los métodos para construir, destruir e imprimir el número complejo, así como también las operaciones matemáticas necesarias. Además se creó un método para obtener al número complejo de forma polar de modo que los atributos del complejo pase a ser módulo y fase, así como los setters y getters necesarios para cada atributo.

2.2. Clase Image

La clase *image* tiene como atributos los valores de alto y ancho de la matriz, como también el valor de escala de grises que tenga la imagen utilizada. Esta compuesta de métodos de construcción, el cual pide memoria para una matriz cuadrada de dimensión correspondiente al mayor valor de alto y ancho, y método de destrucción, el cual libera la memoria. También se crearon *getters* y *setters* de los atributos mencionados y también dos métodos para setear y obtener un valor específico de la matriz. Por otro lado, existen dos métodos de impresión, uno de la matriz por pantalla y otro de impresión de la imagen a un archivo de salida. También se creó el atributo para obtener la dimensión mayor de modo de hacer que la imagen de salida resulta cuadrada. Por último se encuentra el método *fill_matrix()* el cual recibe una matriz de enteros como parámetro y a partir de ésta rellena la matriz actual cuadrada de forma tal que la matriz ingresada se encuentre centrada y llenada con ceros para completar las dimensiones.

2.3. Formato PGM

El formato PGM (*Portable Graymap Format*) es un formato de imagen de código abierto el cual consiste de un código identificatorio (**P2**) el cual define que se trata de una imagen en formato *.pgm*, luego un número entre 0 y 255, el cual representa la escala de grises sobre la cual se va a encontrar la imagen. A continuación dos números que representan las dimensiones de la imagen en píxeles, y por último la matriz de números representando cada píxel con el valor de gris correspondiente. Si bien no todas las imágenes poseen los valores de los píxeles ordenados de igual modo, es decir, ordenados de igual manera que dimensión de la imagen, el programa es lo suficientemente robusto como para obtener correctamente todos los valores previamente mencionados.

Ejemplo de formato PGM:

```
P2
# Comentario
3 2
5
0 1 2
3 4 5
```

2.4. Implementación

Una vez implementadas y probadas las clases, paso a realizarse el algoritmo pedido.

Como primer paso, se llama a las funciones de la clase "*cmd_line*" provistas, cuyo objetivo es dividir y guardar los argumentos que se ingresan mediante la linea de comandos. Esta revisa que tipo de argumento se le pasa, buscando el indicador seguido de el argumento en sí, para guardarlo en una estructura provista. Para el caso de las funciones, se creo una nueva estructura y se modifco la función encargada de parsear estas para adaptarlo a este programa.

Se utilizaron variables globales, que corresponden al flujo de entrada y salida de cada archivo, así como también una estructura que se utiliza para las comparaciones de las funciones ingresadas y el parseo de los argumentos ingresados.

Luego, lo que se hizo fue implementar una función que toma como primer y único argumento una variable de la clase "*imagen*" anteriormente explicada. Esta función llamada *read_pgm()* tiene como objetivo leer por la entrada anteriormente especificada, y de carácter global, y completar con la información leída la variable de clase "*imagen*" que se le pasa por referencia. Para lograr esto, guarda en variables auxiliares la información que va leyendo y luego con estas, llama a los setters de la clase *Imagen* con estas variables auxiliares como argumento. A su vez, ademas de guardar los datos, se valida que el formato de la imagen sea correcto, de no ser asi, el programa finaliza mostrando por consola un mensaje de error. Es importante remarcar que a esta imagen se la hace cuadrada y se la centra. En la figura 1a se puede observar que la imagen es cuadrada, pero arriba y abajo hay una franja negra, mostrando que la imagen original es horizontal. Para la figura 1b ocurre lo mismo que se mencionó anteriormente pero en los laterales, mostrando que la imagen original es vertical. Una vez llenada la variable "*imagen*", destruye la memoria pedida.



(a) Imagen horizontal.



(b) Imagen vertical.

Una vez que la imagen de entrada es guardada correctamente, se entra en un *switch* que llama a la función que realiza la transformación. A esta función se la llamó *map_image()*, y recibe por argumento la imagen de entrada, una imagen de salida y como tercer argumento, un puntero a función que le indica la función que debe utilizar. Es por este motivo que se realizó un *switch*, para poder llamar a esta función con la función matemática correspondiente, que fue parseada de la línea de entrada anteriormente. En el siguiente párrafo se detallará el funcionamiento del mapeo.

La función *map_image()* crea una matriz de complejos cuadrada auxiliar a partir de la dimensión máxima de la imagen de entrada. Esta toma valores desde el $-1-i$ hasta el $1+i$, es decir, cubriendo un cuadrado de lado 2, con el centro de este ubicado en el centro de la matriz. Para esto se calculó el paso que debería haber para poder cumplir con estos límites, por lo tanto, en la matriz de complejos no necesariamente se encuentra el origen del plano. Luego se recorre la matriz de complejos, donde se guarda en una variable auxiliar al valor complejo de cada posición y a continuación se le realiza la transformación correspondiente. Se pregunta si el punto transformado cae dentro del rango permitido, y de ser así, se busca mediante búsqueda binaria el punto más cercano en la misma matriz de complejos. Con ese punto, se copia el valor de gris de la imagen de entrada en la posición correspondiente de la imagen de salida. Una vez que finaliza esta función, se borra la memoria pedida para la matriz de complejos. A continuación se muestra un ejemplo:

Para la posición $(0,0)$ de la matriz de la imagen destino:

- Se toma el valor complejo en esa posición a través de la matriz de complejos y se lo guarda en un auxiliar:
complejo aux = matriz_de_complejos[0][0];
- Se transforma el valor:
aux = aux.transformar();
- Se busca la posición donde se encuentra este valor en la matriz de complejos. Si no está dentro de los límites no se hace nada.
pos=busqueda_binaria(aux,matriz_de_complejos);
- Se toma el valor de color de la matriz de entrada en esta posición y se lo guarda en la imagen de salida (en la posición inicial):
matriz_salida[0][0] = matriz_entrada[pos[0]][pos[1]]

Para realizar la búsqueda, se utilizó búsqueda binaria adaptado para una matriz. De esta manera no se la recorre, ya que al estar ordenada, se compara si el valor se encuentra por encima o por debajo del punto del medio y se divide a la mitad el problema. Esto ocurre para cada dimensión, por lo tanto reduce el problema a un cuarto del anterior en cada llamado. Esta estrategia se denomina Dividir y conquistar, que consiste dividir el problema

en problemas más pequeños. Por esta razón, se realizan menos comparaciones y disminuye el tiempo de computo.

Para esta función se utilizó recursividad simple y de cola, de esta manera se optimiza espacio en el stack. En cada llamado se va ajustando los valores de los límites donde se busca el valor, hasta llegar al caso base. Este es cuando se busca en una porción de matriz de 2x2, y ahí se determina cual de las cuatro posiciones es la más próxima al valor buscado.

Por otra parte, se realiza un validación si los límites iniciales son menores a los finales, de ser así retorna NULL.

Por ultimo, se imprime por el output especificado la matriz de salida.

2.5. Compilación y ejecución

La compilación y pruebas del código se realizaron por consola. La compilación se realizo mediante un *makefile* que se ejecuta por consola mediante:

```
$ make all
```

Los contenidos del mismo estarán en la sección de código fuente, pero lo que hace es compilar cada archivo necesario (complejo.h y .cpp, imagen.h y .cpp, main.h y .cpp) para luego generar el ejecutable.

Luego para su ejecución se debe pasar por linea de comando las opciones **-i** y **-o** que permiten seleccionar los archivos de entrada y salida respectivamente. Ambos tienen valores por defecto si seguido de la opción, se encuentra el carácter “-”(o si no se especifican estas opciones), tomando como entrada y salidas las estándar. En caso de especificar un archivo no existente de salida, el mismo se crea. Sin embargo, estas opciones no son obligatorias como sí lo es la opción **-f**, es decir que el programa no podrá ejecutarse si no se la especifica. Esta permite seleccionar qué función se va a utilizar para procesar la imagen. Sin embargo esta no requiere que se especifique un argumento, ya que por defecto la función identidad $f(z) = z$ es la que se ejecutará. Las posibles funciones que puede realizar el programa son las siguientes:

- $f(z) = z$, función identidad, pasando como parámetro **z**
- $f(z) = e^z$, función exponencial, pasando como parámetro **expz**.
- $f(z) = \frac{1}{z}$, función inversa, pasando como parámetro **inversa**
- $f(z) = \log(z)$, función logaritmo, pasando como parámetro **log**
- $f(z) = \sin(z)$, función seno, pasando como parámetro **sin**
- $f(z) = z^2$, función elevar al cuadrado, pasando como parámetro **pow**
- $f(z) = Re(z) - Img(z)j$, función conjugada, pasando como parámetro **conjugar**

En caso de que alguno de los parámetros no sea ingresado debe tomar los valores por defecto, y en caso de que detecte algún problema con los parámetros, se notificará por consola el problema y terminará el programa.

```
$ ./function_image -f function -i in_file.pgm -o out_file.pgm
```

Por ultimo, se puede ejecutar el programa con solo una opción **-h** para que este devuelva como debe utilizarse el programa:

```
$ ./function_image -h
$ function_image [-f function] [-i file] [-o file]
$ funciones: z, expz, conjugar, inversa, log, sin, pow
```

3. Corridas de prueba

En primer lugar, se ejecutó el programa utilizando la herramienta *valgrind*, para detectar si el programa tenía problemas de memoria. El programa fue invocado con valores correctos de argumentos y archivos válidos. Sin embargo, se encontraron errores, ya que no toda la memoria dinámica requerida era liberada correctamente. Por lo tanto, se realizaron las correcciones correspondientes en el destructor de la clase *image* y en las funciones que también requerían memoria dinámica. Luego se corroboró que los errores se hayan solucionado.

Seguido a esto, se ejecutó el programa a través de la herramienta previamente mencionada, pero con archivos corruptos, es decir, con menor y mayor cantidad de datos de lo que indicaba el encabezado. Nuevamente se detectaron errores respecto a la memoria, ya que cuando se detectaba una falla en el archivo, el programa no borraba la memoria pedida, teniendo así fugas de memoria.

Es importante remarcar que para todas las pruebas mencionadas anteriormente se ejecutaron con la siguiente línea utilizando los correspondientes archivos para cada caso.

```
$ valgrind --leak-check=full -v ./function_image.exe -f function -o file -i file
```

A continuación se mostrarán los resultados de las corridas de prueba y ejecución del programa para distintas funciones, así como también los errores que aparecen al tener información incorrecta, ingresar mal la imagen o ingresar mal la línea de comandos.



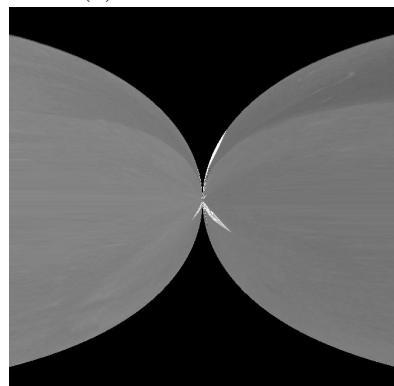
(a) Imagen original.



(b) Función identidad.



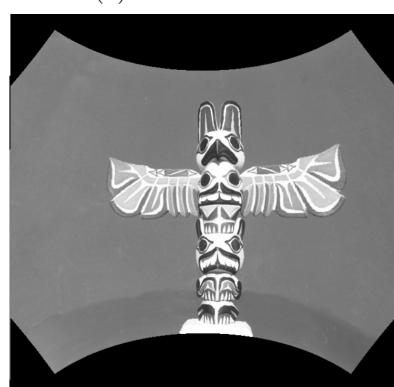
(c) Función exponencial.



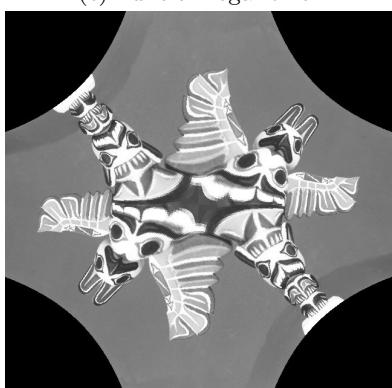
(d) Función inversa.



(e) Función logaritmo.



(f) Función seno.



(g) Función elevar al cuadrado.



(h) Función conjugar.

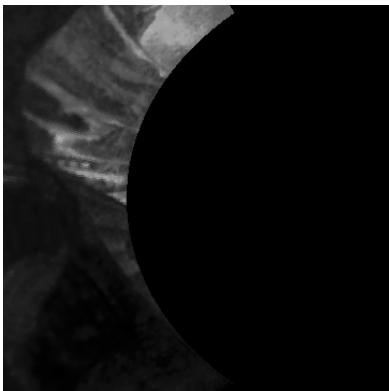
Figura 2: Imagen horizontal transformada.



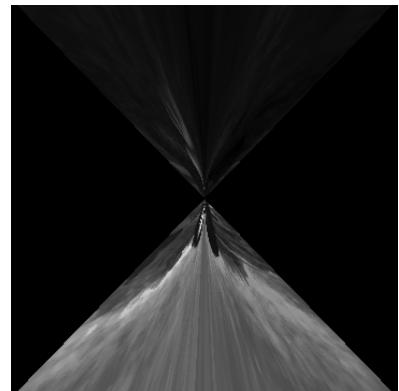
(a) Imagen original.



(b) Función identidad.



(c) Función exponencial.



(d) Función inversa.



(e) Función logaritmo.



(f) Función seno.



(g) Función elevar al cuadrado.



(h) Función conjugar.

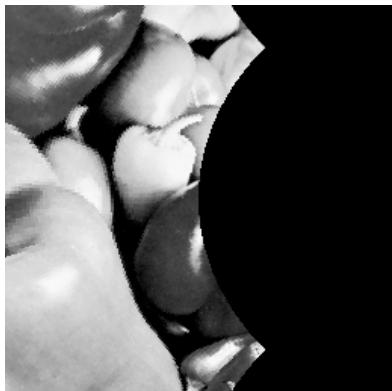
Figura 3: Imagen vertical transformada.



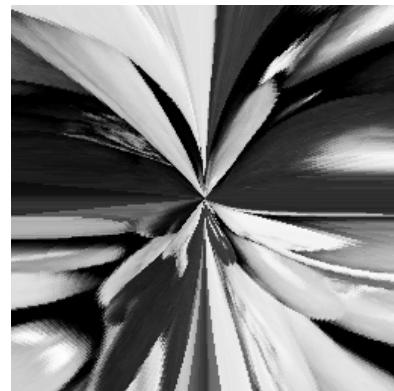
(a) Imagen original.



(b) Función identidad.



(c) Función exponencial.



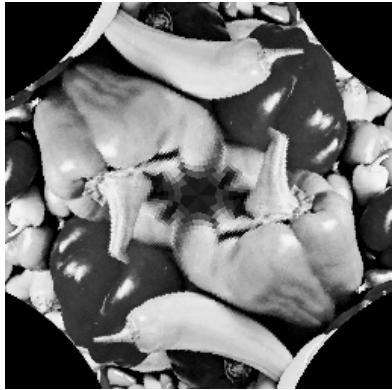
(d) Función inversa.



(e) Función logaritmo.



(f) Función seno.



(g) Función elevar al cuadrado.



(h) Función conjugar.

Figura 4: Imagen cuadrada transformada.

Para realizar la prueba de estrés, se ejecutó el programa con una imagen de dimensión superior a las ejecutadas con anterioridad, siendo esta de 7878 x 4680. El programa se ejecutó de manera correcta, pero como era de esperarse aumentó el tiempo de ejecución.



(a) Imagen original.



(b) Función exponencial.

Figura 5: Imagen que se utilizo para el test de estres.

A continuación se detallarán los distintos mensajes de error que se imprimen por cerr, en este caso por pantalla, teniendo en cuenta el error detectado.

En el caso que una imagen ingresada no tenga el código identificadorio P2, finaliza el programa e imprime:

```
$ No es PGM
```

Si la imagen ingresada no posee el formato de dimensiones correctamente, finaliza el programa e imprime:

```
$ Error de formato.
```

Una vez que comienza a leer los datos que serán guardados en la matriz. En primer lugar analiza si el valor leído esta dentro del rango correspondiente a la escala de grises, en caso contrario imprime el siguiente mensaje:

```
$ Error. Elemento de fuera de rango.
```

En caso que haya menos elementos en el archivo, teniendo en cuenta las dimensiones de la matriz, imprime el siguiente mensaje:

```
$ Error. Cantidad insuficiente de elementos.
```

Por otro lado, si hay mas elementos imprime:

```
$ Error. Cantidad excesiva de elementos.
```

Si ocurre algún error en la búsqueda binaria imprime el siguiente mensaje de error:

```
Error en búsqueda binaria.
```

Cuando se ingresa mal la función a utilizar, termina el programa y devuelve:

```
$ Función invalida
```

Si no se logra abrir la imagen de entrada, finaliza el programa y devuelve:

```
$ No se puede abrir el archivo de entrada: [file]
```

Si no se logra abrir la imagen de salida, finaliza el programa y devuelve:

```
$ No se puede abrir el archivo de salida: [file]
```

En caso que los argumentos sea ingresados erróneamente, termina el programa e imprime:

```
$ Argumento inválido: [arg]
```

Cuando se ingrese una opción desconocida, termina el programa y devuelve:

```
$ Opción desconocida: [option]
```

Si la opción es obligatoria y no fue ingresada, se imprime el siguiente mensaje:

```
La opción requiere el argumento: -[arg]
```

4. Conclusiones

A pesar de tener varias mejoras posibles para aplicar al algoritmo, pudimos resolver el problema dado, transformando imágenes con las funciones pedidas y agregando nuevas. Aplicamos y nos familiarizamos con características del lenguaje C++ que no están presentes en el lenguaje C, como por ejemplo las clases implementadas necesarias para el algoritmo o la forma de pedir y destruir memoria dinámica.

5. Código fuente

5.1. main.cpp

```
1 #include <iostream>
2 #include <fstream>
3 #include <iomanip>
4 #include <sstream>
5 #include <cstdlib>
6 #include <math.h>
7
8 #include "image.h"
9 #include "complejo.h"
10 #include "main.h"
11
12 using namespace std;
13
14 //*****VARIABLES GLOBALES*****//
```

```

16 static istream *iss = 0; // Input Stream (clase para manejo de los flujos de entrada)
17 static ostream *oss = 0; // Output Stream (clase para manejo de los flujos de salida)
18 static fstream ifs; // Input File Stream (derivada de la clase ifstream que deriva
19 de istream para el manejo de archivos)
20 static fstream ofs; // Output File Stream (derivada de la clase ofstream que deriva
21 de ostream para el manejo de archivos)
22
23
24 static option_t options[] = {
25     {1, "i", "input", "-", opt_input, OPT_DEFAULT},
26     {1, "o", "output", "-", opt_output, OPT_DEFAULT},
27     {1, "f", "function", NULL, opt_function, OPT_DEFAULT},
28     {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
29     {0, },
30 };
31
32 enum functions {z, expz, conjugar, inversa, logaritmo, seno, pow2};
33 static functions chosen_function = z;
34
35 // ****MAIN*****
36
37 int main(int argc, char * const argv[]){
38     image input_image;
39
40     cmdline cmdl(options); // Objeto con parametro tipo option_t (struct)
41     // declarado globalmente. Ver linea 51 main.cc
42     cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline
43
44     if(!read_pgm(input_image)){ // Se lee la imagen de entrada
45         cerr<<"Fallo en el archivo"<<endl;
46         return 1;
47     }
48
49     // Se declara la imagen de salida a partir de las dimensiones de la imagen de entrada
50     image output_image(input_image.get_max_dim(),input_image.get_max_dim(),input_image.
51     get_greyscale());
52
53     // Switch que, a partir del parámetro ingresado por consola, define que función
54     // implementar mediante puntero a función
55     switch(chosen_function){
56     case z:
57         input_image.print_image(oss);
58         return 0;
59         break;
60     case expz:
61         map_image(input_image, output_image, &complejo::exponencial);
62         break;
63     case conjugar:
64         map_image(input_image, output_image, &complejo::conjugar);
65         break;
66     case inversa:
67         map_image(input_image, output_image, &complejo::inversa);
68         break;
69     case logaritmo:
70         map_image(input_image, output_image, &complejo::logaritmo);
71         break;
72     case seno:
73         map_image(input_image, output_image, &complejo::seno);
74         break;
75     }

```

```

72    case pow2:
73        map_image(input_image, output_image, &complejo::pow2);
74        break;
75    default:
76        cerr<< "Error en seleccion de funcion" << endl;
77        return 1;
78    }
79
80    // Se imprime la imagen de salida
81    output_image.print_image(oss);
82
83    return 0;
84}
85
86 //*****FUNCIONES DE CMDLINE*****
87
88 static void opt_input(string const &arg){
89
90    if (arg == "-") {
91        iss = &cin; // Se establece la entrada estandar cin como flujo de entrada
92    }
93    else {
94        ifs.open(arg.c_str(), ios::in);
95        iss = &ifs;
96    }
97    // Verificamos que el stream este OK.
98    if (!iss->good())
99        cerr << "No se puede abrir el archivo de entrada: "<< arg<< endl;
100       exit(1);
101   }
102 }
103
104 static void opt_output(string const &arg){
105
106    if (arg == "-") { // Si el input es -
107        oss = &cout; // se establece la salida estandar cout como flujo de salida
108    } else {
109        ofs.open(arg.c_str(), ios::out);
110        oss = &ofs;
111    }
112    if (!oss->good())
113        cerr << "No se puede abrir el archivo de salida: "<< arg<< endl;
114        exit(1);
115    }
116 }
117
118 static void opt_function(string const &arg){
119
120    if (arg == FUNCTION_Z || arg == "-") {chosen_function = z;} // Se establece la función estandar z
121
122    else if (arg == FUNCTION_EXPZ) {chosen_function = expz; }
123    else if (arg == FUNCTION_CONJUGAR) {chosen_function = conjugar; }
124    else if (arg == FUNCTION_INVERSA) {chosen.function = inversa; }
125    else if (arg == FUNCTION_LOGARITMO) {chosen_function = logaritmo; }
126    else if (arg == FUNCTION_SENO) {chosen_function = seno; }
127    else if (arg == FUNCTION_POW) {chosen_function = pow2; }
128    else {
129        cerr << "Funcion invalida" << endl;
130        exit(1);
131    }

```

```

132 }
133
134 static void opt_help(string const &arg){ // La opción -h imprime el formato de ejecuci
135     ón
136     cout << " cmdline -f function [-i file] [-o file]" << endl;
137     cout << "funciones: z, expz, conjugar, inversa, log, sin, pow" << endl;
138     exit(0);
139 }
140 //*****METODOS DE CMDLINE*****
141
142 cmdline::cmdline(){}
143
144 cmdline::cmdline(option_t *table) : option_table(table){}
145
146 void cmdline::parse(int argc, char * const argv[]) {
147 #define END_OF_OPTIONS(p) \
148     ((p)->short_name == 0 \
149      && (p)->long_name == 0 \
150      && (p)->parse == 0)
151
152     for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
153         op->flags &= ~OPT_SEEN;
154
155     for (int i = 1; i < argc; ++i) {
156
157         if (argv[i][0] != '-')
158             cerr << "Argumento inválido." << argv[i] << endl;
159             exit(1);
160     }
161
162     if (argv[i][1] == '-' \
163         && argv[i][2] == 0)
164         break;
165
166     if (argv[i][1] == '-')
167         i += do_long_opt(&argv[i][2], argv[i + 1]);
168     else
169         i += do_short_opt(&argv[i][1], argv[i + 1]);
170 }
171
172     for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op) {
173 #define OPTION_NAME(op) \
174     (op->short_name ? op->short_name : op->long_name)
175     if (op->flags & OPT_SEEN)
176         continue;
177     if (op->flags & OPT_MANDATORY) {
178         cerr << "Opción " << "-" << OPTION_NAME(op) << " es obligatoria." << "\n";
179         exit(1);
180     }
181     if (op->def_value == 0)
182         continue;
183     op->parse(string(op->def_value));
184 }
185 }
186
187 int cmdline::do_long_opt(const char *opt, const char *arg) {
188     // Se recorre la tabla de opciones, y se busca la entrada larga que se corresponda
189     // con la opción de línea de comandos. De no encontrarse, se indica el error
190     correspondiente.

```

```

191 for (option_t *op = option_table; op->long_name != 0; ++op) {
192     if (string(opt) == string(op->long_name)) {
193         // Se marca esta opción como usada en forma explícita, para evitar tener que
194         inicializarla con el valor por defecto.
195         op->flags |= OPT_SEEN;
196
197         if (op->has_arg) {
198             if (arg == 0) { // Verificamos que se provea un argumento
199                 cerr << "La opción requiere el argumento: "<< "--"<< opt<< "\n";
200                 exit(1);
201             }
202             op->parse(string(arg));
203             return 1;
204         } else {
205             op->parse(string("")); // Opción sin argumento.
206             return 0;
207         }
208     }
209 }
210 // Error: opción no reconocida.
211 cerr << "Opción desconocida: "<< "--"<< opt<< "."<< endl;
212 exit(1);
213 return -1;
214 }
215
216 int cmdline::do_short_opt(const char *opt, const char *arg) {
217     option_t *op;
218
219     for (op = option_table; op->short_name != 0; ++op) {
220
221         if (string(opt) == string(op->short_name)) {
222             op->flags |= OPT_SEEN;
223             if (op->has_arg) {
224                 if (arg == 0) {
225                     cerr << "La opción requiere el argumento: "<< "-"<< opt<< "\n";
226                     exit(1);
227                 }
228                 op->parse(string(arg));
229                 return 1;
230             } else {
231                 op->parse(string(""));
232                 return 0;
233             }
234         }
235     }
236     cerr << "Opción desconocida: "<< "-"<< opt << "." << endl;
237     exit(1);
238     return -1;
239 }
240
241
242 // *****FUNCIONES*****//
243
244
245 // Esta función lee del archivo de input y llena la imagen VACIA que se le pasa como
246 // argumento.
247 bool read_pgm(image & img_arg){
248     int aux_int, aux_size[2], aux_greyscale;
249     int i=0;

```

```

250 int ** aux_matrix;
251 string in_string , temp;
252
253
254 getline(*iss , in_string); // Identificador PGM.
255
256 if (in_string[0] == PGM_IDENTIFIER[0]){
257     if (in_string[1] != PGM_IDENTIFIER[1]){
258         cerr<< "No es PGM" << endl; // En caso que el identificador sea incorrecto ,
259         imprime un mensaje de error .
260         return false ;
261     }
262 }else {cerr<< "No es PGM" << endl; return false ;}
263
264 getline(*iss , in_string);
265 if (in_string[0] == SKIP_LINE_IDENTIFIER){ // Se detecta si se leyó un comentario .
266     getline(*iss , in_string); // Se leen las dimensiones de la matriz .
267 }
268
269 stringstream ss (in_string);
270
271 while ( i < 2 && !ss.eof()){
272     ss >> temp;
273     if (stringstream(temp) >> aux_int){ // Si puedo convertir a int , guardo .
274         aux_size[i] = aux_int;
275         i++;
276     }
277     temp = "";
278 }
279 if (i == 1){
280     cout<< "Error en el formato." << endl;
281     return false ;
282 }
283
284 ss >> temp;
285 if (stringstream(temp) >> aux_int){ // Si puedo convertir a int , es un error .
286     cout<< "Error en el formato." << endl;
287     return false ;
288 }
289
290
291 img_arg.set_width(aux_size[0]); // Se guarda el ancho de la matriz .
292 img_arg.set_height(aux_size[1]); // Se guarda el alto de la matriz .
293
294
295 getline(*iss , in_string);
296 aux_greyscale = stoi(in_string);
297 img_arg.set_greyscale(aux_greyscale); // Se guarda el valor de la escala de grises .
298
299 // Crea la matriz de enteros y los llena con ceros .
300 // Como la matriz va a ser cuadrada , se pide dos veces de dimension "max".
301
302 aux_matrix = new int*[aux_size[1]];
303 for (int i = 0; i < aux_size[1]; i++){
304     aux_matrix[i] = new int [aux_size[0]];
305 }
306
307 for (int i = 0; i < aux_size[1]; i++){
308     for (int j = 0; j < aux_size[0]; j++){
309         *iss >> aux_int;

```

```

310     if (!(iss->eof())){ // Se evalúa si los elementos que esta leyendo corresponde a
311         la cantidad de la dimensión
312         if (aux_int <= aux_greyscale && aux_int >= 0)
313         {
314             aux_matrix[i][j] = aux_int;
315         }else{
316             cerr<<"Error. Elemento de fuera de rango."<<endl; // En caso que haya menos
317             elementos,
318             for (int i = 0; i<aux_size[1]; i++)           // se destruye matriz auxiliar
319                 delete [] aux_matrix[i];
320             delete [] aux_matrix;
321             return false;
322         }
323     }else{
324         cerr<<"Error. Cantidad insuficiente de elementos."<<endl; // En caso que haya
325         menos elementos,
326         for (int i = 0; i<aux_size[1]; i++)           // se destruye matriz auxiliar
327             delete [] aux_matrix[i];
328         delete [] aux_matrix;
329         return false;
330     }
331 }
332
333 *iss >> aux_int;
334
335 if (!iss->eof()){ // Se evalúa si el siguiente elemento es eof.
336     cerr<<"Error. Cantidad excesiva de elementos."<<endl; // En caso que haya más
337     elementos,
338     for (int i = 0; i<aux_size[1]; i++)           // Se destruye matriz auxiliar en caso de
339         error
340         delete [] aux_matrix[i];
341     delete [] aux_matrix;
342     return false;
343 }
344
345 img_arg.fill_matrix(aux_matrix); // Se llena la matriz de imagen
346
347 for (int i = 0; i<aux_size[1]; i++) // Se destruye la matriz auxiliar
348     delete [] aux_matrix[i];
349 delete [] aux_matrix;
350
351 return true;
352
353
354 void generate_matrix_c(double max, complejo *** matrix){
355
356     // Esta funcion genera una matriz de complejos con valores que van desde el -1-i
357     hasta el 1+i formando un
358     // rectangulo de lado 2, con el centro del plano complejo en el centro de la matriz.
359     (*matrix) = new complejo*[(int)max]; // Pido memoria para la matriz
360     for (int i = 0; i < max; i++){
361         (*matrix)[i] = new complejo[(int)max];
362     }
363 }
```

```

364 double paso=2/(max-1); // Determina el paso que debe haber debido al salto de una
365     posicion para que en los limites se encuentren los unos
366 double aux_real=-1;
367 double aux_imag=1;
368
369 // Se recorre la matriz y se la va llenando punto a punto con el valor de complejo
370 // correspondiente
371 for (int i = 0; i < max; i++){
372     for (int j = 0; j < max; j++){
373         (*matrix)[i][j]=complejo(aux_real,aux_imag);
374         aux_real=aux_real+paso; // Se ajusta el valor para la proxima posicion
375     }
376     aux_real=-1; // Se reinicia el valor del x ya que recorre por filas
377     aux_imag=aux_imag-paso; // Se ajusta el valor para la proxima posicion
378 }
379
380 int * binary_search(const complejo c, complejo *** matrix, int in_lim[2], int fin_lim
381 [2]){
382     // Esta funcion realiza la busqueda del complejo c en la matriz matrix recibida por
383     // puntero a traves del metodo binario de busqueda de
384     // forma recursiva. in_lim y fin_lim son arreglos de dos posiciones , en la primer
385     // posicion de cada uno se encuentra el valor inicial y final
386     // para las filas y en la segunda los mismos pero para las columnas.
387     // El valor que retorna es la posicion de la matriz de donde se encuentra el valor c o
388     // el mas proximo a este.
389     // Se prueba que los limites iniciales sean menores a los finales , de no ser asi se
390     // devuleve NULL
391     if (in_lim[0]>fin_lim[0] || in_lim[1]>fin_lim[1]){
392         return NULL;
393     }
394
395     // Caso base:
396     // Cuando se llega a una porcion de matriz de 2x2 se fija cual de los cuatro valores
397     // es el mas proximo a c y ajusta los limites para
398     // retornar el vector correspondiente
399     if ((fin_lim[0]-in_lim[0]) == 1 && (fin_lim[1]-in_lim[1]) == 1){
400
401         if (abs(c.get_real() - ((*matrix)[in_lim[1]][in_lim[0]]).get_real()) > abs(c.
402             get_real() - ((*matrix)[fin_lim[1]][fin_lim[0]]).get_real())){
403             in_lim[0] = fin_lim[0];
404         }
405         if (abs(c.get_img() - ((*matrix)[in_lim[1]][in_lim[0]]).get_img()) > abs(c.get_img()
406             () - ((*matrix)[fin_lim[1]][fin_lim[0]]).get_img())){
407             in_lim[1] = fin_lim[1];
408         }
409         return in_lim;
410     } else if ((fin_lim[0]-in_lim[0]) == 0 && (fin_lim[1]-in_lim[1]) == 0){ // Si
411         // encontro el valor exacto lo devuelve. Solo llegar si la matriz
412         // de la
413         // imagen original es de 1x1 porque sino va a terminar en el cb anterior.
414         in_lim[1] = fin_lim[1];
415         return in_lim;
416     }
417
418     // Se calcula la posicion del medio
419     int medio_x = in_lim[0]+(fin_lim[0]-in_lim[0])/2;

```

```

413 int medio_y = in_lim[1]+(fin_lim[1]-in_lim[1])/2;
414
415 // Se evalua si la parte real del complejo es mayor a la parte real del medio, y
416 // luego lo mismo
417 // para la parte imaginaria. Luego se ajustan los limites para el proximo llamado.
418 if (c.get_real()>= ((*matrix)[medio_y][medio_x]).get_real()){ // Se fija si se
419     encuentra en el semiplano derecho
420     in_lim[0] = medio_x;
421     if (c.get_img()>= ((*matrix)[medio_y][medio_x]).get_img()){ // Se fija si se
422         encuentra en el semiplano superior
423         fin_lim[1] = medio_y;
424         return binary_search(c, matrix, in_lim, fin_lim);
425     }else{
426         in_lim[1] = medio_y;
427         return binary_search(c, matrix, in_lim, fin_lim);
428     }
429 }else{
430     fin_lim[0] = medio_x;
431     if (c.get_img()>=((*matrix)[medio_y][medio_x]).get_img()){ // Se fija si se
432         encuentra en el semiplano superior
433         fin_lim[1] = medio_y;
434         return binary_search(c, matrix, in_lim, fin_lim);
435     }else{
436         in_lim[1] = medio_y;
437         return binary_search(c, matrix, in_lim, fin_lim);
438     }
439 }
440
441 void map_image(image & original, image & destino, complejo(complejo::* function_pointer)
442     (void)){
443     int * pos;
444     int in_lim[2];
445     int fin_lim[2];
446     int aux_color;
447     int max = original.get_max_dim();
448     complejo aux;
449     complejo ** complex_matrix;
450
451     // Se genera la matriz de complejos de tamaño max por max
452     generate_matrix_c(original.get_max_dim(), &complex_matrix);
453
454     // Se recorre la matriz de complejos para transformar cada uno de los puntos.
455     // Los indices de la matriz de complejos coinciden con los de la matriz destino, por
456     // lo tanto
457     // alcanza con recorrer solo una de las dos.
458     for(int i=0; i < destino.get_max_dim(); i++){
459         for (int j = 0; j < destino.get_max_dim(); j++)
460         {
461             // Se inicializan los limites para la busqueda binaria
462             in_lim[0]=0;
463             in_lim[1]=0;
464             fin_lim[0]=max-1;
465             fin_lim[1]=max-1;
466
467             // Se guarda el valor de la matriz de complejos para luego realizar la
468             transformacion

```

```

467 aux = complex_matrix[ i ][ j ];
468
469 aux = (aux.*function_pointer)();
470
471 // Se corrobora que el valor c a buscar este dentro de el semiplano que conforman
472 // los puntos (-1+i), (-1-i), (1-i) y (1+i)
473 // sino lo esta, no se hace nada, ya que como la matriz de la imagen destino se
474 // encuentra rellena de ceros (negro)
475 if (abs(aux.get_real()) <= 1 && abs(aux.get_img()) <= 1){
476     pos = binary_search(aux,&complex_matrix,in_lim,fin_lim);
477     if (pos !=NULL){ // Si no se detecta un error se se guarda el color en la
478         imaguen destino
479         aux_color = original.get_matrix_value(pos[1],pos[0]);
480         destino.set_matrix_value(i,j,aux_color);
481     }
482     else {
483         cerr<<" Error en busqueda binaria."<<endl;
484         for (int i = 0; i<max; i++){ // Borra la memoria pedida por
485             generate_matrix_c
486             if (complex_matrix[ i ]){
487                 delete [] complex_matrix[ i ];
488             }
489         }
490     }
491 }
492 for (int i = 0; i<max; i++){ // Borra la memoria pedida por generate_matrix_c
493     if (complex_matrix[ i ]){
494         delete [] complex_matrix[ i ];
495     }
496 }
497 delete [] complex_matrix;
498 }
```

main.cpp

5.2. main.h

```

1 #ifndef MAIN_H
2 #define MAIN_H
3
4 #include <iostream>
5 #include <fstream>
6 #include <iomanip>
7 #include <sstream>
8 #include <cstdlib>
9 #include <string>
10
11 #define OPT_DEFAULT 0
12 #define OPT_SEEN 1
13 #define OPT_MANDATORY 2
14
15 #define FUNCTION_Z "z"
16 #define FUNCTION_EXPZ "expz"
17 #define FUNCTION_CONJUGAR "conjugar"
18 #define FUNCTION_INVERSA "inversa"
19 #define FUNCTION_LOGARITMO "log"
```

```

20 #define FUNCTION_SENO "sin"
21 #define FUNCTION_POW "pow"
22 #define NUL '\0'
23
24 struct option_t {
25     int has_arg;
26     const char *short_name;
27     const char *long_name;
28     const char *def_value;
29     void (*parse)(std::string const &); // Puntero a función de opciones
30     int flags;
31 };
32
33 //*****DECLARACION FUNCIONES*****
34
35 static void opt_input(string const &);
36 static void opt_output(string const &);
37 static void opt_function(string const &);
38 static void opt_help(string const &);
39
40 bool read_pgm(image &);
41 void generate_matrix_c(double, complejo ***);
42 int * binary_search(const complejo, complejo ***, int [2], int [2]);
43 void map_image(image &, image &, complejo (complejo::*function_pointer)());
44
45
46 class cmdline {
47     // Este atributo apunta a la tabla que describe todas las opciones a procesar.
48     // Por el momento, sólo puede ser modificado mediante constructor, y debe finalizar
49     // con un elemento nulo.
50
51     option_t *option_table;
52
53     // El constructor por defecto cmdline::cmdline(), es privado, para evitar construir "
54     // parsers"
55     // (analizador sintáctico, recibe una palabra y lo interpreta en una acción
56     // dependiendo su
57     // significado para el programa sin opciones. Es decir, objetos de esta clase sin
58     // opciones.
59
60     cmdline();
61     int do_long_opt(const char *, const char *);
62     int do_short_opt(const char *, const char *);
63
64 public:
65     cmdline(option_t *);
66     void parse(int, char * const []);
67 };
68
69#endif

```

main.h

5.3. image.cpp

```

1 #include<iostream>
2 #include "image.h"
3
4 using namespace std;
5
6 // Constructor por defecto

```

```

7 image::image(){
8     width=0;           // Ancho
9     height=0;          // Alto
10    greyscale=0;       // Escala de grises
11    matrix=NULL;       // Matriz
12}
13
14// Constructor por parametro
15image::image(const int w, const int h, const int gs){
16
17    int max = 0;
18    width = w;
19    height = h;
20    greyscale = gs;
21    if(w<h){max = h;} else{max = w;}
22
23    this->matrix = new int*[max+1];
24    for (int i = 0; i < max; i++){ // Pide memoria para crear la matriz cuadrada
25        this->matrix[i] = new int[max+1];
26    }
27
28    for (int i = 0; i < max; i++){
29        for (int j = 0; j < max; j++){
30            this->matrix[i][j] = 0;      // Rellena la matriz con color negro
31        }
32    }
33}
34
35
36// Destructor
37image::~image(){
38
39    int max = this->get_max_dim();
40
41    if (matrix){
42        for (int i = 0; i<max; i++){
43            if (matrix[i]){
44                delete [] matrix[i]; // Libera la memoria pedida para crear la matriz
45            }
46        }
47    }
48    delete [] matrix;
49}
50
51// Setter y getters
52
53void image::set_width(const int A){ // Setea el ancho
54    width = A;
55}
56
57int image::get_width(){ // Obtiene el ancho
58    return width;
59}
60
61void image::set_height(const int A){ // Setea el alto
62    height = A;
63}
64
65int image::get_height(){ // Obtiene el alto
66    return height;
67}

```

```

68
69 void image::set_greyscale(const int A){ // Setea la escala de grises
70     greyscale = A;
71 }
72
73 int image::get_greyscale(){ // Setea la escala de grises
74     return greyscale;
75 }
76
77 int image::get_max_dim(){ // Obtiene la dimension mayor
78     if(width < height) {return height;}
79     else{return width;}
80 }
81
82 void image::set_matrix_value(const int & i,const int & j,const int & aux_color){ // 
    Setea un valor específico de la matriz en la posición i,j
83     matrix[i][j]=aux_color;
84 }
85
86 int image::get_matrix_value(const int & i,const int & j){ // Obtiene un valor especí
    fico de la matriz en la posición i,j
87     return matrix[i][j];
88 }
89
90 void image::print_matrix(){ // Imprime la matriz por cout
91     int max=0;
92
93     if(width<height){max = height;} else{max = width;}
94
95     for(int x=0 ; x<max ; x++){
96         for(int y=0 ; y<max ; y++) {
97             cout << this->matrix[x][y] << " ";
98         }
99         cout << endl;
100    }
101 }
102
103 void image::print_image(ostream *os){ // Imprime la imagen por ostrem
104
105     int max=0;
106     if(width<height){max = height;} else{max = width;}
107
108     *os << "P2" << '\n' << max << " " << max << '\n' << greyscale << '\n';
109     if (os->bad()) {
110         cerr << "cannot write to output stream."
111         << endl;
112         exit(1);
113     }
114
115     for (int x=0;x<max;x++)
116     {
117         for (int y=0;y<max;y++) {
118             *os << this->matrix[x][y];
119             *os << " ";
120             if (os->bad()) {
121                 cerr << "cannot write to output stream."
122                 << endl;
123                 exit(1);
124             }
125     }
126 }
```

```

127     *os<<endl;
128     if (os->bad()) {
129         cerr << "cannot write to output stream."
130         << endl;
131         exit(1);
132     }
133 }
134 }
135
136 // Este método pide memoria y llena la matriz pasada como argumento.
137 void image::fill_matrix(int ** matrix){
138
139     int max=0;
140     bool IS_VERTICAL = true;
141
142     max = get_max_dim();
143     if (width>height)
144         IS_VERTICAL=false;
145
146
147     this->matrix = new int*[max]; // Se pide memoria para la matriz cuadrada de lado
148     // mayor.
149     for (int i=0 ; i<max ; i++){
150         this->matrix[i] = new int[max];
151     }
152
153     if (width == height){ // Para imagenes cuadradas
154         for (int i = 0; i < max; i++){
155             for (int j = 0; j < max; j++){
156                 this->matrix[i][j] = matrix[i][j];
157             }
158         }
159
160         for (int i = 0; i < max; i++){
161             for (int j = 0; j < max; j++){
162                 if (IS_VERTICAL){ // Para imagenes vertical
163                     if ( j<((max-width)/2) || j>((max+width)/2)-1 )
164                         this->matrix[i][j] = 0;
165                     else{
166                         this->matrix[i][j] = matrix[i][j-(max-width)/2];
167                     }
168                 }
169                 else{ // Para imagenes horizontal
170                     if ( i<((max-height)/2) || i>((max+height)/2)-1 )
171                         this->matrix[i][j] = 0;
172                     else
173                         this->matrix[i][j] = matrix[i-(max-height)/2][j];
174                 }
175             }
176         }
177         height = max;
178         width = max;
179     }

```

image.cpp

5.4. image.h

```
1 #ifndef _IMAGE_H_INCLUIDO_
```

```

2 #define _IMAGE_H_INCLUIDO_
3
4 #include <iostream>
5 #include "complejo.h"
6
7 using namespace std;
8
9 class image{
10     private:
11         int width; // Ancho y alto
12         int height;
13         int greyscale;
14         int **matrix;
15
16     public:
17
18     // Constructores
19     image(); // Por defecto
20     image(const int, const int, const int); // Constructor por argumentos
21     ~image(); // Destructor
22
23     // Setters y getters
24     void set_width(const int);
25     int get_width();
26     void set_height(const int);
27     int get_height();
28     void set_greyscale(const int );
29     int get_greyscale();
30     int get_max_dim();
31     void set_matrix_value(const int &,const int &,const int &);
32     int get_matrix_value(const int &,const int &);
33
34     // Printers
35     void print_matrix();
36     void print_image(ostream*);
37
38     void fill_matrix(int **);
39 };
40
41 #endif

```

image.h

5.5. complejo.cpp

```

1 #include <iostream>
2 #include "complejo.h"
3 #include <math.h>
4
5 using namespace std;
6
7 // Constructor por defecto
8 complejo::complejo(){
9     real = 0;
10    img = 0;
11 }
12
13 // Constructor normal, primer argumento es real, segundo es img.
14 complejo::complejo(const double arg_real, const double arg_img){
15    real = arg_real;

```

```

16     img = arg_img;
17 }
18
19 // Constructor por copia, simplemente copia los dos valores de un complejo a otro
20 complejo::complejo(const complejo & arg_complejo){
21     real = arg_complejo.real;
22     img = arg_complejo.img;
23 }
24
25 // Destructor
26 complejo::~complejo(){}
27
28 // Setters
29 void complejo::set_real(const double arg_real){ // Setea la parte real
30     real = arg_real;
31 }
32
33 void complejo::set_img(const double arg_img){ // Setea la parte imaginaria
34     img = arg_img;
35 }
36
37 // Getters
38 double complejo::get_real(void) const{ // Obtiene la parte real
39     return real;
40 }
41
42 double complejo::get_img(void) const{ // Obtiene la parte imaginaria
43     return img;
44 }
45
46 double complejo::get_modulo(){ // Obtiene el módulo
47     return sqrt((this->real)*(this->real) + (this->img)*(this->img));
48 }
49
50 double complejo::get_angulo(){ // Obtiene el angulo
51     return atan((this->img)/(this->real));
52 }
53
54 // Printer
55 void complejo::print_complejo(){
56     cout << "(" << real << "," << img << ")";
57 }
58
59 // Operador SUMA con un complejo
60 complejo complejo::operator + (const complejo & complejo_a_sumar){
61     complejo aux;
62     aux.real = (this->real) + complejo_a_sumar.real;
63     aux.img = (this->img) + complejo_a_sumar.img;
64     return aux;
65 }
66
67 // Operador SUMA con un escalar
68 complejo complejo::operator + (const double & double_a_sumar){
69     complejo aux;
70     aux.real = (this->real) + double_a_sumar;
71     aux.img = (this->img);
72     return aux;
73 }
74
75 // Operador RESTA con un complejo
76 complejo complejo::operator - (const complejo & complejo_a_restar){

```

```

77    complejo aux;
78    aux.real = (this->real) - complejo_a_restar.real;
79    aux.img = (this->img) - complejo_a_restar.img;
80    return aux;
81 }
82
83 // Operador RESTA con un escalar
84 complejo complejo::operator - (const double & double_a_restar){
85     complejo aux;
86     aux.real = (this->real) - double_a_restar;
87     aux.img = (this->img);
88     return aux;
89 }
90
91 // Operador MULTIPLICACION por un complejo
92 complejo complejo::operator * (const complejo & A){
93     complejo aux;
94     aux.real = (this->real * A.real) - (this->img * A.img);
95     aux.img = (this->real * A.img) + (this->img * A.real);
96     return aux;
97 }
98
99 // Operador MULTIPLICACION por un escalar
100 complejo complejo::operator * (const double & A){
101     complejo aux;
102     aux.real = (this->real * A);
103     aux.img = (this->img * A);
104     return aux;
105 }
106
107 // Operador DIVISION por un complejo
108 complejo complejo::operator / (const complejo & divisor){
109     complejo aux;
110     double a = this->real;
111     double b = this->img;
112     double c = divisor.real;
113     double d = divisor.img;
114
115     aux.real = (a*c)/(c*c + d*d) + (b*d)/(c*c + d*d);
116     aux.img = (b*c)/(c*c + d*d) - (a*d)/(c*c + d*d);
117     return aux;
118 }
119
120 // Operador DIVISION por un escalar
121 complejo complejo::operator / (const double & divisor){
122     complejo aux;
123     aux.real = this->real / divisor;
124     aux.img = this->img / divisor;
125     return aux;
126 }
127
128 // Operador IGUAL
129 complejo & complejo::operator = (const complejo & complejo_a_igualar){
130     real=complejo_a_igualar.real;
131     img=complejo_a_igualar.img;
132     return *this;
133 }
134
135 // Funciones
136
137 // EXPONENCIAL

```

```

138 complejo complejo::exponencial(){
139
140     complejo aux;
141
142     double modulo = exp(this->real);
143     double angulo = this->img;
144
145     aux.real = modulo*cos(angulo);
146     aux.img = modulo*sin(angulo);
147
148     return aux;
149 }
150
151 // CONJUGAR
152 complejo complejo::conjugar(){
153     complejo aux;
154     aux.real = this->real;
155     aux.img = -(this->img);
156     return aux;
157 }
158
159 // INVERSA
160 complejo complejo::inversa(){
161     complejo aux;
162     aux=this->conjugar();
163     aux.real=aux.real/(aux.get_modulo());
164     aux.img=aux.img/(aux.get_modulo());
165     return aux;
166 }
167
168 // LOGARITMO
169 complejo complejo::logaritmo(){
170
171     complejo aux;
172
173     double modulo = this->get_modulo();
174
175     aux.real = log(modulo);
176     aux.img = atan2(this->img, this->real);
177
178     return aux;
179 }
180
181 // SENO
182 complejo complejo::seno(){
183
184     complejo aux;
185
186     aux.real = sin(real)*cosh(img);
187     aux.img = cos(real)*sinh(img);
188
189     return aux;
190 }
191
192 // CUADRADO
193
194 complejo complejo::pow2(){
195
196     complejo aux;
197
198     aux.real = real*real-img*img;

```

```

199     aux.img = real*img*2;
200
201     return aux;
202 }
```

complejo.cpp

5.6. complejo.h

```

1 #ifndef _COMPLEJO_H_INCLUIDO_
2 #define _COMPLEJO_H_INCLUIDO_
3
4 #include <iostream>
5
6 using namespace std;
7
8 class complejo{
9
10 public:
11
12     // Constructores y destructor
13     complejo(); // Constructor por defecto
14     complejo(const double , const double ); // Constructor normal, usa doubles
15     complejo(const complejo &); // Constructor por copia, simplemente copia los dos
16         valores de un complejo a otro
17     ~complejo(); // Destructor, no tiene que destruir nada porque no pedimos memoria
18
19     // Setters y Getters
20     void set_real(const double );
21     void set_img(const double );
22     double get_real() const;
23     double get_img() const;
24     double get_modulo();
25     double get_angulo();
26
27     // Printer
28     void print_complejo();
29
30     // Operadores
31     complejo operator + (const complejo &); // Suma de complejos
32     complejo operator - (const complejo &); // Resta de complejos
33     complejo operator * (const complejo &); // Multiplicacion de complejos
34     complejo operator / (const complejo &); // Division de complejos
35     complejo & operator = (const complejo &); // Operador =
36
37     // Funciones
38     complejo exponencial ();
39     complejo conjugar (); // Conjuga el complejo
40     complejo inversa (); // Calcula la inversa
41     complejo logaritmo();
42     complejo seno();
43     complejo pow2();
44
45 private:
46     double real; // Dos atributos, real e imaginario, se explica solo
```

```
51     double img;  
52 };  
54  
55 #endif
```

complejo.h

5.7. Makefile

```
1 CXXARGS = -g -Wall  
2 CXXFLAGS = -I. $(CXXARGS)  
3  
4 all: function_image  
5  
6 function_image: main.cpp image.cpp complejo.cpp main.h image.h complejo.h  
7   $(CXX) $(CXXFLAGS) -o function_image complejo.cpp image.cpp main.cpp  
8  
9 clean:  
10   $(RM) -vf *.o *.exe *.t *.out *.err
```

Makefile

6. 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 19:35:23 \$

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. 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 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 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 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 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 0 11 11 11 11 0 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
```

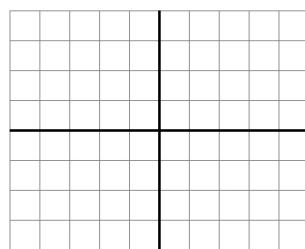


Figura 1: `grid.pgm`

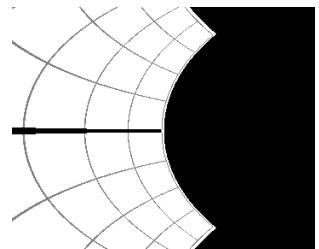


Figura 2: `grid-exp.pgm`

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)
```

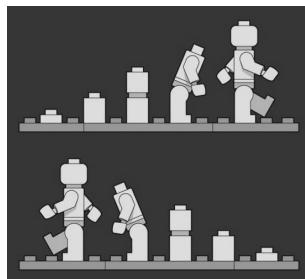


Figura 3: `evolution.pgm`

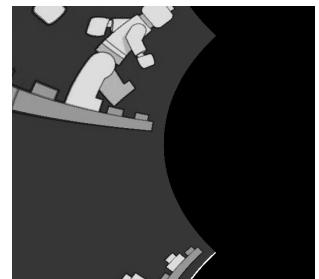


Figura 4: `evolution-exp.pgm`

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 se 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>