



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

Trabajo Práctico N°1

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.ar

Entrega	Fecha	Correcciones
1	18/6	

Índice

1. Introducción	2
2. Desarrollo	2
2.1. Clase complejo	2
2.2. Clase Image	3
2.3. Clase stk	3
2.4. Formato PGM	3
2.5. Librería <i>is_functions</i>	3
2.6. Implementación	3
2.7. Correcciones del TP0	5
2.8. Compilación y ejecución	6
3. Corridas de prueba	7
3.1. Errores	11
3.2. Errores en parser	12
4. Conclusiones	12
5. Código fuente	13
5.1. main_tp1.cpp	13
5.2. main_tp1.h	21
5.3. image.cpp	23
5.4. image.h	26
5.5. complejo.cpp	27
5.6. complejo.h	31
5.7. parser.cpp	33
5.8. parser.h	37
5.9. is_functions.cpp	38
5.10. is_functions.h	40
5.11. stk.h	41
5.12. shunting-yard.cpp	44
5.13. shunting-yard.h	49
5.14. Makefile	49
6. Enunciado	50

1. Introducción

El objetivo de este trabajo práctico es continuar con la implementación del procesamiento de imágenes comenzado en el trabajo práctico 0. A diferencia del trabajo anterior, en este se busca expandir las posibilidades de que función se le puede aplicar a la imagen. Para esto, el usuario puede ingresar una ecuación matemática que dependa de la variable **z**, la cual se utilizará para transformar los píxeles.

Para resolver este problema, se utilizará el algoritmo *Shunting-Yard*, mediante el cual se reacomoda la expresión ingresada por consola para generar una cadena o vector en notación polaca inversa, o RPN (reverse polish notation) por sus siglas en inglés. Para trabajar las expresiones se implementará una clase *stack*.

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, 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 pasen 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 rellenada con ceros para completar las dimensiones.

2.3. Clase stk

Para la realización del algoritmo *Shunting-Yard* es necesario utilizar una estructura de datos del tipo **LIFO** (*Last In, First Out*). Por lo tanto se implemento la clase **stk**. Esta se encuentra compuesta por un puntero a nodo, y el tamaño del *stack*. Nodo es una clase creada dentro de esta clase **stk** y posee un puntero a nodo y el dato T que es tipo *template*, por lo tanto, es posible utilizar esta estructura con cualquier tipo de dato. Por lo tanto esta clase se podría ver como una lista enlazada simple.

2.4. 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.

2.5. Librería *is_functions*

Se creo además una librería llamada *is_functions*, que posee funciones del tipo booleanas. El objetivo de las mismas es recibir caracteres o cadenas de caracteres y compararlos contra caracteres o cadenas de caracteres predefinidos, devolviendo verdadero si son iguales o falso si no lo son. Estas funciones son utilizadas por el algoritmo que parsea la función ingresada por consola, así como también por el algoritmo *Shunting-Yard* y el algoritmo que resuelve la notación polaca.

2.6. Implementación

En primer lugar, 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. Ésta revisa que tipo de argumento se le ingresa como parámetro, buscando el indicador seguido del argumento en sí, para guardarla en una estructura provista. Para el caso de las funciones, se creó una nueva estructura y se modificó la función encargada de parsear para adaptarlo a este programa.

Se utilizaron variables globales, que corresponden al flujo de entrada y salida de cada archivo, así como también una variable *string* donde se guarda la función ingresada por consola.

Luego, lo que se hizo fue implementar una función que toma como primer y único argumento una variable de la clase **image** anteriormente explicada. La 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 **image** 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, además de guardar los datos, se valida que el formato de la imagen sea correcto, de no ser así, 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 finalizado el proceso, se libera la memoria pedida.



(a) Imagen horizontal.



(b) Imagen vertical.

Una vez que la imagen de entrada es guardada correctamente, se pasa a validar y parsear la función ingresada por consola. Se llama a la función *parse_function()*, la cual tiene como objetivo devolver un vector de strings con todas las partes de la expresión matemática ingresada, llamadas *tokens*, para que luego se pueda aplicar el algoritmo de *Shunting-Yard*. La misma valida que la función ingresada este balanceada, no comience con un operador matemático como multiplicar, dividir o elevado (que requieren un elemento a izquierda) y que sea una expresión matemáticamente válida. A medida que valida, separa los elementos en diferentes *tokens* de acuerdo a su tipo:

- Números: pueden ser números tanto números enteros como decimales, o bien con notación exponencial.
- Números negativos: estos números son casos especiales en los que se guarda el numero acompañado del signo menos. Ocurre cuando la función empieza con un numero negativo, luego de un paréntesis abierto y después de un operador.
- Operadores matemáticos: estos siendo el signo de suma, resta, multiplicación, división o potencia.
- Funciones matemáticas: exp, ln, abs(z), phase(z), re(z), im(z).
- Variable "z" y "j": "z" siendo la variable del píxel que se quiere transformar, y "j" la variable de numero imaginario.

Para lograr esto, hace uso de las funciones propias de parseo y la librería creada *is_functions*.

Una vez que sale de la función de parseo, ya se tiene un arreglo de *strings* con los tokens validados de la función ingresada por consola.

Luego de parsear la función ingresada, se pasa a aplicar el algoritmo de *Shunting-Yard* al vector de *strings* generado anteriormente. El mismo se realiza en la función *shunting_yard()*, que recibe por parámetro un *stack* de *strings* donde se guardara el resultado y el vector de *strings* generado junto con su tamaño. Se implementó el mismo en base al pseudocódigo encontrado en la pagina de *Wikipedia*. Luego de aplicar la función, se tiene un *stack* de tokens organizados en notación polaca, listo para utilizar en la transformación de la imagen.

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 punto $-1 - i$ hasta el punto $1 + i$, es decir, cubriendo un cuadrado de lado 2, con el centro del mismo 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.

Para ésta se utiliza la función *solve_rpn()*, que por cada píxel desapila el stack de *tokens* (recursivamente) en notación RPN (Reverse Polish Notation) y hace las operaciones matemáticas correspondientes.

Luego, se pregunta si el punto transformado cae dentro del rango permitido, y de ser así, se busca 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 = solve_rpn(stack, aux);
- 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(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]]

Se utilizo búsqueda binaria pero adaptada a 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 cómputo.

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. Éste 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.

2.7. Correcciones del TP0

En cuanto a las correcciones del trabajo práctico 0, las mismas no han podido aplicarse por una cuestión de tiempos y dado que se trasladan al trabajo práctico 1, han quedado sin corregir. Sin embargo, se realizarán en la próxima versión y se aplicaran tanto al trabajo práctico 0 como al 1. Lo que si se trato de aplicar fue una

búsqueda mas simple que la binaria: dado que es una matriz ordenada de la cual conocemos el paso, se probo realizar la búsqueda con una ecuación simple como se muestra a continuación:

$$-1 + \text{paso} \cdot k_entero \leq \text{valor_a_buscar} \quad (1)$$

Donde nos interesa encontrar el K entero que cumpla esa condición para cada dimensión (dos dimensiones en el caso de una matriz).

Al implementar el algoritmo y poblarlo con la función identidad, se vio que la imagen era la misma pero tenia peor definición que la original, indicio de que la búsqueda de esta forma no era tan precisa como la binaria. Por este motivo se continuo con la implementación de esta última como se explica a continuación.

2.8. 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 para luego generar el ejecutable. Estos archivos son: **main_tp1.cpp**, **shunting_yard.cpp**, **complejo.cpp**, **parser.cpp**, **image.cpp**, **is_functions.cpp**, **main_tp1.h**, **shunting_yard.h**, **complejo.h**, **parser.h**, **image.h**, **is_functions.h** y **stk.h**.

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 la función que se utilizará para transformar a la imagen. Es una función con variable '**z**', donde se pueden operar con números complejos y operaciones matemáticas como exponencial, logaritmo, seno, coseno y mas que serán especificadas a continuación.

Las posibles funciones y operaciones que puede realizar el programa son las siguientes:

- Operaciones matemáticas simples, como sumar (+), restar(-), multiplicar(*), dividir (/) y elevar (^).
- *exp()*, función exponencial.
- *ln()*, función logaritmo.
- *re()*, parte real del parámetro.
- *im()*, parte imaginaria del parámetro, como imaginario puro.
- *abs()*, módulo del parámetro.
- *phase()*, fase del parámetro.

En caso de que detecte algún problema con los parámetros ingresados por línea de comando, se notificará por consola el problema y terminará el programa.

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

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

```
$ ./function_image -h
$ Funciones: exp(z) , ln(z) , re(z) , im(z) , abs(z) , phase(z) .
$ Operadores: +, -, *, /.
```

3. Corridas de prueba

En un primero momento, se ejecuté el programa utilizando Valgrind para comprobar que no tenga fugas de memoria. Luego de reparar una fuga menor de 104 bytes, el programa quedo sin pérdidas y se pasaron a realizar corridas de prueba.

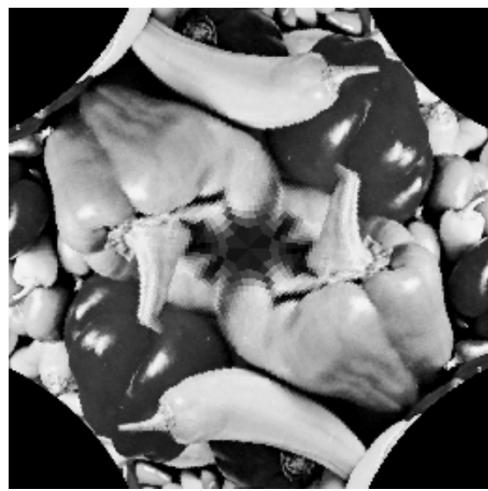
Las pruebas realizadas se ejecutaron con la siguiente linea utilizando los correspondientes archivos para cada caso.

```
$ 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 linea de comandos. En primer lugar, se corrió con una función simple con resultado ya conocido, para corroborar que opere bien:



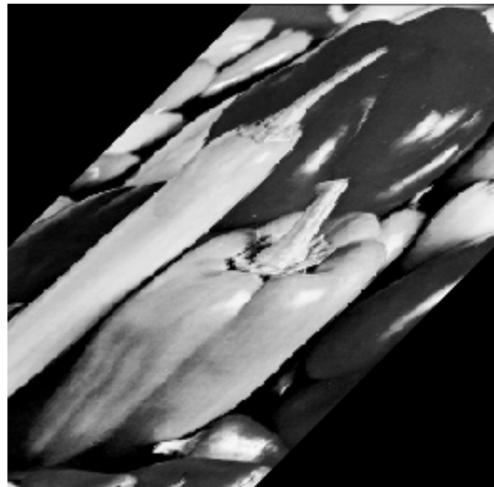
(a) Imagen original.



(b) Elevedo al cuadrado.



(c) $z \wedge \exp(z)$.



(d) $z + j * \text{im}(z)$.

Figura 2: Corridas de prueba para la imágen pepper.pgm.

Por otro lado se realizaron pruebas con funciones básicas realizando rotaciones y cambios en el tamaño.

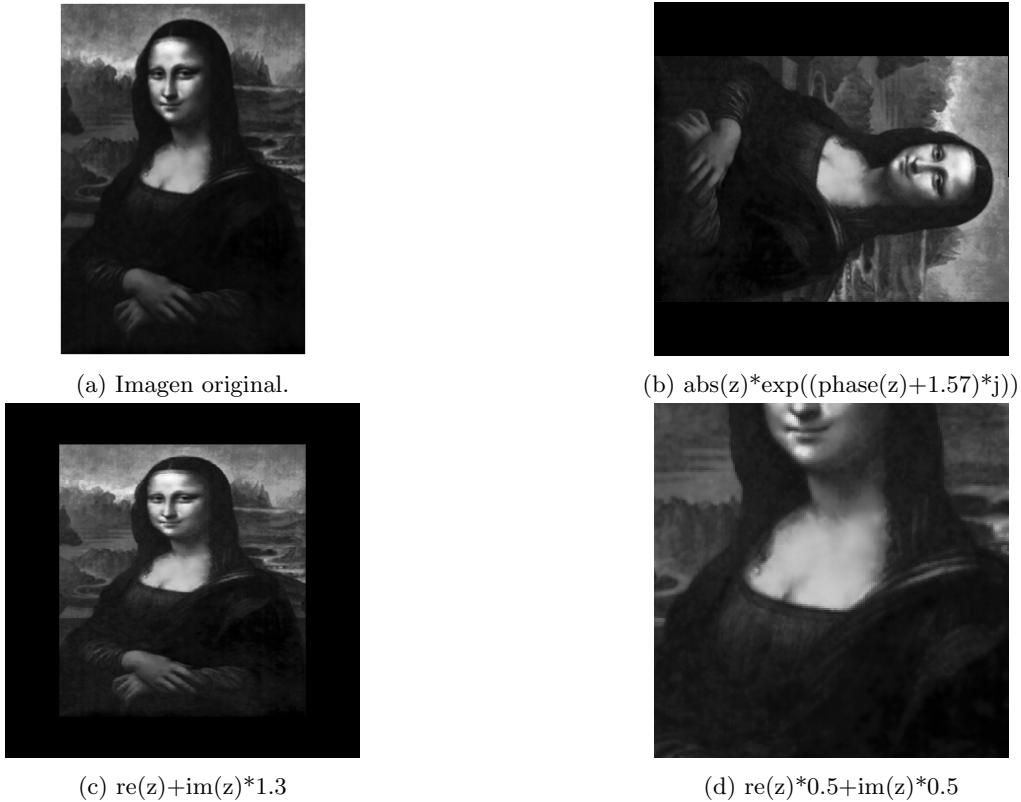


Figura 3: Transformaciones básicas

Para realizar una rotación de la imagen como se ve en la figura (3b), se puede expresar a la función en notación polar y sumarle a la fase el ángulo deseado para rotar. Como se puede observar en la figura (3c) se realizó una reducción en el eje imaginario, intentando llevar una imagen vertical a una cuadrada. Notar que como se menciono anteriormente la función $\text{im}(z)$ devuelve la parte compleja del numero \mathbf{z} , es decir que es un número imaginario puro, por lo tanto no es necesario multiplicarlo por j . Por último se realizo un ampliación de la imagen (figura (3d)).

Para realizar esta acción existen dos formas de hacerlo, multiplicando la parte real y la imaginaria por el valor deseado ($\text{re}(z)*0.5+\text{im}(z)*0.5$) o de manera análoga expresar al complejo en notación polar y multiplicar al modulo por el valor deseado ($0.5*\text{abs}(z)*\exp(j*\text{phase}(z))$). Si se desea desplazar la imagen basta con sumarle un numero real para desplazarla en el eje horizontal y de manera análoga con un numero imaginario puro para el eje vertical.

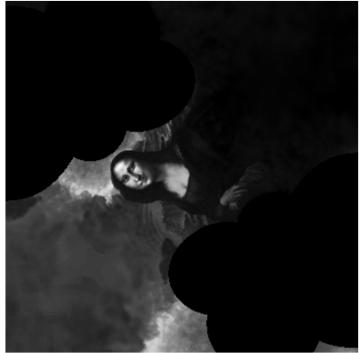
Luego se realizaron más corridas de pruebas, pero en este caso con funciones más extensas:



(a) Imagen original.



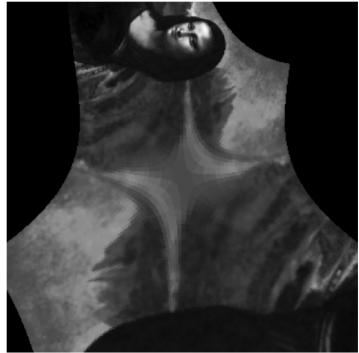
(b) $\exp(z \cdot \ln(z)) - 0.5 - 0.2 \cdot j$



(c) $z / (j \cdot 0.5 + z^2)$



(d) $((z / (j \cdot 0.5 + z^3)) \cdot 0.5) + 0.5 + 0.5 \cdot j$



(e) $((z / (j \cdot 0.5 + 1/z)) \cdot 0.5) + 0.5 + 0.5 \cdot j$



(f) $(z / (j \cdot 0.5 + z^2)) \cdot 0.5$



(g) $((z / (j \cdot 0.5 + 1/z^{(1/2)})) \cdot 0.5) + 0.5 + 0.5 \cdot j$



(h) z^z

Figura 4: Corridas de prueba para la imagen mona_lisa.pgm.

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

3.1. Errores

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.

Si no se puede leer el archivo de entrada ingresado se imprime:

```
Error. Fallo en el archivo.
```

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]
```

3.2. Errores en parser

En caso que la función ingresada no este balanceada, es decir que no estén correctamente utilizado los paréntesis, se imprime el siguiente mensaje:

```
Error. No esta balanceada
```

En caso que la función comience con un operador, se imprime el siguiente mensaje de error:

```
Error, no puede comenzar con -operador
```

Si se ingresa una función matemática inválida, imprime:

```
Error. Funcion matematica erronea.
```

En caso que no se encuentre un paréntesis, se imprime el siguiente mensaje:

```
Error. No encontro parentesis ( parseando.
```

En caso que se quiera dividir por 0, se produce un error y se imprime el siguiente mensaje

```
Error. Division por 0.
```

En caso que el stack se encuentre vacío se imprime el siguiente mensaje:

```
Error. Stack is empty.
```

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_tp1.cpp

```
#include "main_tp1.h"

using namespace std;

//*****VARIABLES GLOBALES*****
static istream *iss = 0; // Input Stream (clase para manejo de los flujos de entrada)
static ostream *oss = 0; // Output Stream (clase para manejo de los flujos de salida)
static fstream ifs; // Input File Stream (derivada de la clase ifstream que deriva
                    de istream para el manejo de archivos)
static fstream ofs; // Output File Stream (derivada de la clase ofstream que deriva
                    de ostream para el manejo de archivos)

static string PGM_IDENTIFIER = "P2";
static char SKIP_LINE_IDENTIFIER = '#';

static option_t options [] = {
    {1, "i", "input", "-", opt_input, OPT_DEFAULT},
    {1, "o", "output", "-", opt_output, OPT_DEFAULT},
    {1, "f", "function", NULL, opt_function, OPT_DEFAULT},
    {0, "h", "help", NULL, opt_help, OPT_DEFAULT},
    {0, },
};

static string entered_function; // Funcion leida directamente de consola

// *****MAIN*****
int main(int argc, char * const argv []) {
    image input_image;
    string * string_array;
    size_t string_array_size= 0;
    stk <string> output_stk;

    cmdline cmdl(options); // Objeto con parametro tipo option_t (struct)
                           // declarado globalmente
    cmdl.parse(argc, argv); // Metodo de parseo de la clase cmdline

    // Se lee la imagen de entrada
    if (!read_pgm(input_image)){
        cerr<<"Error. Fallo en el archivo."<<endl;
        return 1;
    }

    // Se declara la imagen de salida a partir de las dimensiones de la imagen de entrada
    image output_image(input_image.get_max_dim(),input_image.get_max_dim(),input_image.get_greyscale());

    // Se parsea la funcion ingresada para aplicarle el algoritmo Shunting-Yard
    string_array = parse_function(entered_function, string_array_size);

    // Se aplica el algoritmo Shunting-Yard al array de strings y se guarda la funcion en
    // RPN en output_stk
    shunting_yard(output_stk, string_array, string_array_size);
```

```

// Se mapea la imagen de salida resolviendo la expresion ingresada
map_image(input_image, output_image, output_stk);

// Se imprime la imagen de salida
output_image.print_image(oss);

delete [] string_array;

return 0;
}

//*****FUNCIONES DE CMDLINE*****
static void opt_input(string const &arg){

if (arg == "-") {
    iss = &cin; // Se establece la entrada estandar cin como flujo de entrada
}
else {
    ifs.open(arg.c_str(), ios::in);
    iss = &ifs;
}
// Verificamos que el stream este OK.
if (!iss->good())
    cerr << "No se puede abrir el archivo de entrada: "<< arg<< endl;
    exit(1);
}
}

static void opt_output(string const &arg){

if (arg == "-") { // Si el input es -
    oss = &cout; // se establece la salida estandar cout como flujo de salida
}
else {
    ofs.open(arg.c_str(), ios::out);
    oss = &ofs;
}
if (!oss->good())
    cerr << "No se puede abrir el archivo de salida: "<< arg << endl;
    exit(1);
}
}

static void opt_function(string const &arg){
entered_function = arg;
}

static void opt_help(string const &arg){ // La opción -h imprime el formato de ejecuci
ón
    cout << "cmdline -f function [-i file] [-o file]" << endl;
    cout << "Funciones: exp(z), ln(z), re(z), im(z), abs(z), phase(z)." << endl;
    cout << "Operadores: +, -, *, /." << endl;
    exit(0);
}

//*****METODOS DE CMDLINE*****
cmdline::cmdline(){}

```

```

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

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

for (option_t *op = option_table; !END_OF_OPTIONS(op); ++op)
    op->flags &= ~OPT_SEEN;

for (int i = 1; i < argc; ++i) {

    if (argv[i][0] != '-') {
        cerr << "Argumento inválido." << argv[i] << endl;
        exit(1);
    }

    if (argv[i][1] == '-' \
        && argv[i][2] == 0)
        break;

    if (argv[i][1] == '-')
        i += do_long_opt(&argv[i][2], argv[i + 1]);
    else
        i += do_short_opt(&argv[i][1], argv[i + 1]);
}

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

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

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

        op->flags |= OPT_SEEN;

        if (op->has_arg) {
            if (arg == 0) { // Verificamos que se provea un argumento
                cerr << "La opción requiere el argumento: " << "--" << opt << "\n";
                exit(1);
            }
            op->parse(string(arg));
        }
    }
}
}

```

```

        return 1;
    } else {
        op->parse(string("")); // Opción sin argumento.
        return 0;
    }
}
// Error: opción no reconocida.
cerr << "Opción desconocida: " << "--" << opt << "." << endl;
exit(1);
return -1;
}

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

    for (op = option_table; op->short_name != 0; ++op) {

        if (string(opt) == string(op->short_name)) {
            op->flags |= OPT_SEEN;
            if (op->has_arg) {
                if (arg == 0) {
                    cerr << "La opción requiere el argumento: " << "-" << opt << "\n";
                    exit(1);
                }
                op->parse(string(arg));
                return 1;
            } else {
                op->parse(string(""));
                return 0;
            }
        }
    }
    cerr << "Opción desconocida: " << "-" << opt << "." << endl;
    exit(1);
    return -1;
}

// *****FUNCIONES***** //

// Esta función lee del archivo de input y llena la imagen VACIA que se le pasa como
// argumento
bool read_pgm(image & img_arg){
    int aux_int, aux_size[2], aux_greyscale;
    int i=0;
    int ** aux_matrix;
    string in_string, temp;

    getline(*iss, in_string); // Identificador PGM

    if (in_string[0] == PGM_IDENTIFIER[0]){
        if (in_string[1] != PGM_IDENTIFIER[1]){
            cerr << "No es PGM" << endl; // En caso que el identificador sea incorrecto,
            imprime un mensaje de error
            return false;
        }
    }
    else {cerr << "No es PGM" << endl; return false;}

    getline(*iss, in_string);
}

```

```

if (in_string[0] == SKIP_LINE_IDENTIFIER){ // Se detecta si se leyó un comentario
    getline(*iss, in_string); // Se leen las dimensiones de la matriz
}

stringstream ss (in_string);

while (i < 2 && !ss.eof()){
    ss >> temp;
    if(stringstream(temp) >> aux_int){ // Si se pudo convertir se guarda en el arreglo
        aux_size[i] = aux_int;
        i++;
    }
    temp = "";
}
if (i == 1){
    cout << "Error en el formato." << endl;
    return false;
}

ss >> temp;
if(stringstream(temp) >> aux_int){ // Si se pudo convertir es un error
    cout << "Error en el formato." << endl;
    return false;
}

img_arg.set_width(aux_size[0]); // Se guarda el ancho de la matriz
img_arg.set_height(aux_size[1]); // Se guarda el alto de la matriz

getline(*iss, in_string);
aux_greyscale = stoi(in_string);
img_arg.set_greyscale(aux_greyscale); // Se guarda el valor de la escala de grises

// Crea la matriz de enteros y los llena con ceros
// Como la matriz va a ser cuadrada, se pide dos veces de dimension "max"

aux_matrix = new int*[aux_size[1]];
for (int i = 0; i < aux_size[1]; i++){
    aux_matrix[i] = new int[aux_size[0]];
}

for (int i = 0; i < aux_size[1]; i++){
    for (int j = 0; j < aux_size[0]; j++){
        *iss >> aux_int;
        if (!(iss->eof())){ // Se evalúa si los elementos que esta leyendo corresponde a
            la cantidad de la dimensión
            if (aux_int <= aux_greyscale && aux_int >= 0)
            {
                aux_matrix[i][j] = aux_int;
            } else{
                cerr << "Error. Elemento de fuera de rango." << endl; // En caso que haya menos
                elementos,
                delete_matrix(aux_matrix, aux_size[1]);
                return false;
            }
        } else{
            cerr << "Error. Cantidad insuficiente de elementos." << endl; // En caso que haya
            menos elementos,
            delete_matrix(aux_matrix, aux_size[1]);
        }
    }
}

```

```

        return false;
    }
}

*iss >> aux_int;

if (!iss->eof()){ // Se evalúa si el siguiente elemento es eof.
    cerr<<" Error. Cantidad excesiva de elementos."<<endl; // En caso que haya más
    elementos,
    delete_matrix(aux_matrix, aux_size[1]);
    return false;
}

img_arg.fill_matrix(aux_matrix); // Se llena la matriz de imagen

delete_matrix(aux_matrix, aux_size[1]);
return true;
}

void generate_matrix_c(double max, complejo *** matrix){

// Esta funcion genera una matriz de complejos con valores que van desde el -1-i
// hasta el 1+i formando un
// rectangulo de lado 2, con el centro del plano complejo en el centro de la matriz.

(*matrix) = new complejo*[(int)max]; // Pido memoria para la matriz
for (int i = 0; i < max; i++){
    (*matrix)[i] = new complejo[(int)max];
}

double paso=2/(max-1); // Determina el paso que debe haber debido al salto de una
    posicion para que en los limites se encuentren los unos
double aux_real=-1;
double aux_imag=1;

// Se recorre la matriz y se la va llenando punto a punto con el valor de complejo
// correspondiente
for (int i = 0; i < max; i++){
    for (int j = 0; j < max; j++){
        (*matrix)[i][j]=complejo(aux_real,aux_imag);
        aux_real=aux_real+paso; // Se ajusta el valor para la proxima posicion
    }
    aux_real=-1; // Se reinicia el valor del x ya que recorre por filas
    aux_imag=aux_imag-paso; // Se ajusta el valor para la proxima posicion
}
}

int * binary_search(const complejo c, complejo *** matrix, int in_lim[2], int fin_lim
[2]){

// Esta funcion realiza la busqueda del complejo c en la matriz matrix recibida por
// puntero a traves del metodo binario de busqueda de
// forma recursiva. in_lim y fin_lim son arreglos de dos posiciones, en la primer
// posicion de cada uno se encuentra el valor inicial y final
// para las filas y en la segunda los mismos pero para las columnas.
// El valor que retorna es la posicion de la matriz de donde se encuentra el valor c o
// el mas proximo a este.
}

```

```

// Se prueba que los limites iniciales sean menores a los finales , de no ser asi se
// devuleve NULL

if (in_lim[0]>fin_lim[0] || in_lim[1]>fin_lim[1]){
    return NULL;
}

// Caso base:
// Cuando se llega a una porcion de matriz de 2x2 se fija cual de los cuatro valores
// es el más proximo a c y ajusta los limites para
// retornar el vector correspondiente
if ((fin_lim[0]-in_lim[0]) <= 1 && (fin_lim[1]-in_lim[1]) <= 1){

    if (abs(c.get_real() - ((*matrix)[in_lim[1]][in_lim[0]]).get_real()) > abs(c.
        get_real() - ((*matrix)[fin_lim[1]][fin_lim[0]]).get_real())){
        in_lim[0] = fin_lim[0];
    }
    if (abs(c.get_img() - ((*matrix)[in_lim[1]][in_lim[0]]).get_img()) > abs(c.get_img()
        () - ((*matrix)[fin_lim[1]][fin_lim[0]]).get_img())){
        in_lim[1] = fin_lim[1];
    }
    return in_lim;
} else if ((fin_lim[0]-in_lim[0]) == 0 && (fin_lim[1]-in_lim[1]) == 0){ // Si
    //encontro el valor exacto lo devuelve. Solo llegar si la matri
    //de la imagen original es de 1x1 porque sino va a
    terminar en el cb anterior.
    in_lim[1] = fin_lim[1];
    return in_lim;
}

// Se calcula la posicion del medio
int medio_x = in_lim[0]+(fin_lim[0]-in_lim[0])/2;
int medio_y = in_lim[1]+(fin_lim[1]-in_lim[1])/2;

// Se evalua si la parte real del complejo es mayor a la parte real del medio , y
// luego lo mismo
// para la parte imaginaria. Luego se ajustan los limites para el proximo llamado.

if (c.get_real()>= ((*matrix)[medio_y][medio_x]).get_real()){ // Se fija si se
    encuentra en el semiplano derecho
    in_lim[0] = medio_x;
    if (c.get_img()>= ((*matrix)[medio_y][medio_x]).get_img()){ // Se fija si se
        encuentra en el semiplano superior
        fin_lim[1] = medio_y;
    } else{
        in_lim[1] = medio_y;
    }
} else{
    fin_lim[0] = medio_x;
    if (c.get_img()>= ((*matrix)[medio_y][medio_x]).get_img()){ // Se fija si se
        encuentra en el semiplano superior
        fin_lim[1] = medio_y;
    } else{
        in_lim[1] = medio_y;
    }
}
return binary_search(c, matrix, in_lim, fin_lim);

```

```
}
```

```
void map_image(image & original, image & destino, stk <string> output_stk){

    int * pos;
    int aux_color;
    int in_lim[2];
    int fin_lim[2];
    double max = original.get_max_dim();
    complejo aux;
    complejo ** complex_matrix;

    // Se genera la matriz de complejos de tamaño max por max
    generate_matrix_c(max, &complex_matrix);

    double paso=2/(max-1); // Determina el paso que debe haber debido al salto de una
                           // posicion para que en los limites se encuentren los unos
    double aux_real=-1;
    double aux_imag=1;

    // Se recorre la matriz de complejos para transformar cada uno de los puntos.
    // Los indices de la matriz de complejos coinciden con los de la matriz destino, por
    // lo tanto
    // alcanza con recorrer solo una de las dos.
    for(int i=0; i < max; i++){

        for (int j = 0; j < max; j++)
        {

            // Se inicializan los limites para la busqueda binaria
            in_lim[0]=0;
            in_lim[1]=0;
            fin_lim[0]=max-1;
            fin_lim[1]=max-1;

            // Se crea un complejo a el cual se le va a realizar la transformacion
            complejo aux (aux_real,aux_imag);

            // Se iguala el stack en RPN a uno vacio para no perder la funcion cuando se
            desapile
            stk <string> stk_to_solve = output_stk;

            // Se transforma el complejo
            solve_rpn(stk_to_solve, aux);

            string aux_string;

            if (!stk_to_solve.peek(aux_string)){
                cerr << "Stack is empty" << endl;
                exit(1);
            }

            stringstream s1 (aux_string);
            s1 >> aux;

            // Se corrobora que el valor c a buscar este dentro de el semiplano que conforman
            los puntos (-1+i), (-1-i), (1-i) y (1+i)
            // sino lo esta, no se hace nada, ya que como la matriz de la imagen destino se
            encuentra rellena de ceros (negro)
```

```

if (abs(aux.get_real()) <= 1 && abs(aux.get_img()) <= 1){

    pos = binary_search(aux,&complex_matrix,in_lim,fin_lim);

    if (pos !=NULL){ // Si no se detecta un error se guarda el color en la
        imagen destino
        aux_color = original.get_matrix_value(pos[1],pos[0]);
        destino.set_matrix_value(i,j,aux_color); // Se guarda el valor del pixel en
        la imagen destino
    }
    else {
        cerr<<"Error en busqueda binaria."<<endl;
        for (int i = 0; i<max; i++){ // Borra la memoria pedida por
        generate_matrix_c
            if (complex_matrix[i]){
                delete [] complex_matrix[i];
            }
        }
        delete [] complex_matrix;
    }
}
aux_real=aux_real+paso; // Se ajusta el valor para la proxima posicion
}
aux_real=-1; // Se reinicia el valor del x ya que recorre por filas
aux_imag=aux_imag-paso; // Se ajusta el valor para la proxima posicion

}

for (int i = 0; i<max; i++){ // Borra la memoria pedida por generate_matrix_c
    if (complex_matrix[i]){
        delete [] complex_matrix[i];
    }
}
delete [] complex_matrix;

}

// Se destruye la matriz
bool delete_matrix(int **&matrix, int size){
    for (int i = 0; i<size; i++)
        delete [] matrix[i];
    delete [] matrix;

    return true;
}

```

main_tp1.cpp

5.2. main_tp1.h

```

#ifndef MAIN_H
#define MAIN_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <cstdlib>
#include <string>

```

```

#include <cmath>

#include "is_functions.h"
#include "complejo.h"
#include "stk.h"
#include "parser.h"
#include "image.h"
#include "shunting_yard.h"

using namespace std;

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

#define NUL '\0'

struct option_t {
    int has_arg;
    const char *short_name;
    const char *long_name;
    const char *def_value;
    void (*parse)(std::string const &); // Puntero a función de opciones
    int flags;
};

//*****DECLARACION FUNCIONES*****//

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

bool read_pgm(image &);
void generate_matrix_c(double, complejo ***);

int * binary_search(const complejo, complejo ***, int [2], int [2]);
void search(int *, const complejo, const double);

void map_image(image &, image &, stk <string>);
bool delete_matrix(int ** &, int);

//*****//

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

    option_t *option_table;

    // El constructor por defecto cmdline::cmdline(), es privado, para evitar construir "
    // parsers"
    // (analizador sintáctico, recibe una palabra y lo interpreta en una acción
    // dependiendo su
    // significado para el programa sin opciones. Es decir, objetos de esta clase sin
    // opciones.

    cmdline();
    int do_long_opt(const char *, const char *);
    int do_short_opt(const char *, const char *);
};

```

```

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

#endif

```

main_tp1.h

5.3. image.cpp

```

#include<iostream>
#include "image.h"

using namespace std;

// Constructor por defecto
image::image(){
    width=0;           // Ancho
    height=0;          // Alto
    greyscale=0;       // Escala de grises
    matrix=NULL;       // Matriz
}

// Constructor por parametro
image::image(const int w, const int h, const int gs){

    int max = 0;
    width = w;
    height = h;
    greyscale = gs;
    if(w<h){max = h;} else{max = w;}

    this->matrix = new int*[max+1];
    for (int i = 0; i < max; i++){ // Pide memoria para crear la matriz cuadrada
        this->matrix[i] = new int[max+1];
    }

    for (int i = 0; i < max; i++){
        for (int j = 0; j < max; j++){
            this->matrix[i][j] = 0;      // Rellena la matriz con color negro
        }
    }
}

// Destructor
image::~image(){

    int max = this->get_max_dim();

    if (matrix){
        for (int i = 0; i<max; i++){
            if (matrix[i]){
                delete [] matrix[i]; // Libera la memoria pedida para crear la matriz
            }
        }
        delete [] matrix;
    }
}

```

```

// Setter y getters

void image::set_width(const int A){ // Setea el ancho
    width = A;
}

int image::get_width(){ // Obtiene el ancho
    return width;
}

void image::set_height(const int A){ // Setea el alto
    height = A;
}

int image::get_height(){ // Obtiene el alto
    return height;
}

void image::set_greyscale(const int A){ // Setea la escala de grises
    greyscale = A;
}

int image::get_greyscale(){ // Setea la escala de grises
    return greyscale;
}

int image::get_max_dim(){ // Obtiene la dimension mayor
    if(width < height) {return height;}
    else{return width;}
}

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
    matrix[i][j]=aux_color;
}

int image::get_matrix_value(const int & i,const int & j){ // Obtiene un valor específico
    fico de la matriz en la posición i,j
    return matrix[i][j];
}

void image::print_matrix(){ // Imprime la matriz por cout
    int max=0;

    if(width<height){max = height;} else{max = width;}

    for(int x=0 ; x<max ; x++){
        for(int y=0 ; y<max ; y++) {
            cout << this->matrix[x][y] << " ";
        }
        cout << endl;
    }
}

void image::print_image(ostream *os){ // Imprime la imagen por ostream

    int max=0;
    if(width<height){max = height;} else{max = width;}

    *os << "P2"<<'n'<<max<<" "<<max<<'n'<<greyscale<<'n';
}

```

```

if (os->bad()) {
    cerr << "cannot write to output stream."
    << endl;
    exit(1);
}

for (int x=0;x<max;x++)
{
    for (int y=0;y<max;y++) {
        *os << this->matrix[x][y];
        *os << " ";
        if (os->bad()) {
            cerr << "cannot write to output stream."
            << endl;
            exit(1);
        }
    }

    *os<<endl;
    if (os->bad()) {
        cerr << "cannot write to output stream."
        << endl;
        exit(1);
    }
}
}

// Este método pide memoria y llena la matriz pasada como argumento.
void image::fill_matrix(int ** matrix){

    int max=0;
    bool IS_VERTICAL = true;

    max = get_max_dim();
    if (width>height)
        IS_VERTICAL=false;

    this->matrix = new int*[max]; // Se pide memoria para la matriz cuadrada de lado
    mayor.
    for (int i=0 ; i<max ; i++){
        this->matrix[i] = new int [max];
    }

    if (width == height){ // Para imagenes cuadradas
        for (int i = 0; i < max; i++){
            for (int j = 0; j < max; j++){
                this->matrix[i][j] = matrix[i][j];
            }
        }
    }

    for (int i = 0; i < max; i++){
        for (int j = 0; j < max; j++){
            if (IS_VERTICAL){ // Para imagenes vertical
                if ( j<((max-width)/2) || j>((max+width)/2)-1 )
                    this->matrix[i][j] = 0;
                else{
                    this->matrix[i][j] = matrix[i][j-(max-width)/2];
                }
            }
        }
    }
}

```

```

        else{    // Para imagenes horizontal
            if ( i<((max-height)/2) || i>((max+height)/2)-1 )
                this->matrix[ i ][ j ] = 0;
            else
                this->matrix[ i ][ j ] = matrix[ i-(max-height)/2 ][ j ];
        }
    }
height = max;
width = max;
}

```

image.cpp

5.4. image.h

```

#ifndef _IMAGE_H_INCLUIDO_
#define _IMAGE_H_INCLUIDO_

#include <iostream>
#include "complejo.h"

using namespace std;

class image{
private:
    int width; // Ancho y alto
    int height;
    int greyscale;
    int **matrix;

public:
    // Constructores
    image(); // Por defecto
    image(const int, const int, const int); // Constructor por argumentos
    ~image(); // Destructor

    // Setters y getters
    void set_width(const int);
    int get_width();
    void set_height(const int);
    int get_height();
    void set_greyscale(const int );
    int get_greyscale();
    int get_max_dim();
    void set_matrix_value(const int &,const int &,const int &);
    int get_matrix_value(const int &,const int &);

    // Printers
    void print_matrix();
    void print_image(ostream*);

    void fill_matrix(int **);
};

#endif

```

image.h

5.5. complejo.cpp

```
#include "complejo.h"

using namespace std;

// Constructor por defecto
complejo::complejo(){
    real = 0;
    img = 0;
}

// Constructor normal, primer argumento es real, segundo es img.
complejo::complejo(const double arg_real, const double arg_img){
    real = arg_real;
    img = arg_img;
}

// Constructor por copia, simplemente copia los dos valores de un complejo a otro
complejo::complejo(const complejo & arg_complejo){
    real = arg_complejo.real;
    img = arg_complejo.img;
}

// Destructor
complejo::~complejo(){}

// Setters
void complejo::set_real(const double arg_real){ // Setea la parte real
    real = arg_real;
}

void complejo::set_img(const double arg_img){ // Setea la parte imaginaria
    img = arg_img;
}

// Getters
double complejo::get_real(void) const{ // Obtiene la parte real
    return real;
}

double complejo::get_img(void) const{ // Obtiene la parte imaginaria
    return img;
}

double complejo::get_abs() const{ // Obtiene el módulo
    return sqrt(((this->real)*(this->real)) + ((this->img)*(this->img)));
}

double complejo::get_phase() const{ // Obtiene el ángulo
    return atan2((this->img),(this->real));
}

// Combierte un complejo a una string con el formato "real imaginario j"
string complejo::to_string(){

    ostringstream x_convert;
    x_convert << *this;
    return x_convert.str();
}
```

```

// Compara dos numeros complejos
bool complejo::operator== (const complejo& a){

    if (real == a.real && img == a.img)
        return true;
    return false;
}

// Compara un complejo con un double
bool complejo::operator== (const double a){

    if (real == a && img == 0)
        return true;
    return false;
}

// Compara un complejo con un int
bool complejo::operator== (const int a){

    if (real == a && img == 0)
        return true;
    return false;
}

// Operador SUMA con un complejo
complejo complejo::operator + (const complejo & complejo_a_sumar){
    complejo aux;
    aux.real = (this->real) + complejo_a_sumar.real;
    aux.img = (this->img) + complejo_a_sumar.img;
    return aux;
}

// Operador SUMA con un escalar
complejo complejo::operator + (const double & double_a_sumar){
    complejo aux;
    aux.real = (this->real) + double_a_sumar;
    aux.img = (this->img);
    return aux;
}

// Operador RESTA con un complejo
complejo complejo::operator - (const complejo & complejo_a_restar){
    complejo aux;
    aux.real = (this->real) - complejo_a_restar.real;
    aux.img = (this->img) - complejo_a_restar.img;
    return aux;
}

// Operador RESTA con un escalar
complejo complejo::operator - (const double & double_a_restar){
    complejo aux;
    aux.real = (this->real) - double_a_restar;
    aux.img = (this->img);
    return aux;
}

// Operador MULTIPLICACION por un complejo
complejo complejo::operator * (const complejo & A){
    complejo aux;

    aux.real = (this->real * A.real) - (this->img * A.img);

```

```

    aux.img = (this->real * A.img) + (this->img * A.real);
    return aux;
}

// Operador MULTIPLICACION por un escalar
complejo complejo::operator * (const double & A){
    complejo aux;
    aux.real = (this->real * A);
    aux.img = (this->img * A);
    return aux;
}

// Operador DIVISION por un complejo
complejo complejo::operator / (const complejo & divisor){
    complejo aux;
    double a = this->real;
    double b = this->img;
    double c = divisor.real;
    double d = divisor.img;

    aux.real = (a*c)/(c*c + d*d) + (b*d)/(c*c + d*d);
    aux.img = (b*c)/(c*c + d*d) - (a*d)/(c*c + d*d);
    return aux;
}

// Operador DIVISION por un escalar
complejo complejo::operator / (const double & divisor){
    complejo aux;
    aux.real = this->real / divisor;
    aux.img = this->img / divisor;
    return aux;
}

// Operador IGUAL
complejo & complejo::operator = (const complejo & complejo_a_igualar){

    real=complejo_a_igualar.real;
    img=complejo_a_igualar.img;
    return *this;
}

complejo complejo::operator = (const double num_a_igualar){

    return complejo (num_a_igualar,0);
}

// Funciones

// EXPONENCIAL
complejo complejo::exponencial(){

    complejo aux;

    double modulo = exp(this->real);
    double angulo = this->img;

    aux.real = modulo*cos(angulo);
    aux.img = modulo*sin(angulo);

    return aux;
}

```

```

// CONJUGAR
complejo complejo::conjugar (){
    complejo aux;
    aux.real = this->real;
    aux.img = -(this->img);
    return aux;
}

// INVERSA
complejo complejo::inversa (){
    complejo aux;
    aux=this->conjugar();
    aux.real=aux.real/(aux.get_abs());
    aux.img=aux.img/(aux.get_abs());
    return aux;
}

// LOGARITMO
complejo complejo::logaritmo (){

    complejo aux;

    double modulo = this->get_abs();

    aux.real = log(modulo);
    aux.img = atan2(this->img, this->real);

    return aux;
}

// SENO
complejo complejo::seno (){

    complejo aux;

    aux.real = sin(real)*cosh(img);
    aux.img = cos(real)*sinh(img);

    return aux;
}

// Complejo elevado a complejo
complejo complejo::complex_pow(const complejo a){

    double r = this->get_abs();
    double th = this->get_phase();

    double r2 = pow(r, a.real)*exp(-th*a.img);

    double th2 = (log(r)*a.img) + (th*a.real);

    return complejo(r2*cos(th2), r2*sin(th2));
}

// Complejo elevado a un numero
complejo complejo::complex_pow(const double a){

    double r = this->get_abs();
    double th = this->get_phase();

```

```

double r2 = pow(r ,a);
double th2 = (th*a);

    return complejo(r2*cos(th2) ,r2*sin(th2));
}

// Devuelve la parte real como un complejo real puro
complejo complejo::re(){
    return complejo (real ,0);
}

// Devuelve la parte imaginaria de un complejo como un imaginario puro
complejo complejo::im(){
    return complejo (0,img);
}

// Imprime el numero complejo en un ostream
ostream & operator << (ostream &out, const complejo &c){
    out << c.get_real();
    out << " " << c.get_img() << " j";
    return out;
}

// Lee un numero complejo con formato "real imaginario j"
istream & operator >> (istream &in ,  complejo &c){
    double aux;
    string temp;

    in >> temp;           // Se lee parte real

    stringstream(temp) >> aux; // Se combierte el numero a double
    c.set_real(aux);        // Se guarda el valor en la parte real

    in >> temp;           // Se lee el proximo dato del istream

    if (!in.eof()){         // Si hay mas datos
        stringstream(temp) >> aux; // Se combierte el numero a double
        c.set_img(aux);        // Se guarda el valor imaginario en el complejo
    }else{
        c.set_img(0);        // Si no se le asigna 0 a la parte imaginaria
    }
    return in;
}

```

complejo.cpp

5.6. complejo.h

```

#ifndef _COMPLEJO_H_INCLUIDO_
#define _COMPLEJO_H_INCLUIDO_

#include <iostream>
#include <string>
#include <sstream>
#include <cmath>

using namespace std;

class complejo{

```

```

public:

    // Constructores y destructor
    complejo(); // Constructor por defecto
    complejo(const double , const double ); // Constructor normal, usa doubles
    complejo(const complejo &); // Constructor por copia, simplemente copia los dos
        valores de un complejo a otro
    ~complejo(); // Destructor, no tiene que destruir nada porque no pedimos memoria

    // Setters y Getters
    void set_real(const double );
    void set_img(const double );
    double get_real() const;
    double get_img() const;
    double get_abs() const;
    double get_phase() const;

    // Printer
    string to_string();

    // Operadores
    complejo operator + (const complejo &); // Suma de complejos
    complejo operator + (const double &);
    complejo operator - (const complejo &); // Resta de complejos
    complejo operator - (const double &);
    complejo operator * (const complejo &); // Multiplicacion de complejos
    complejo operator * (const double &);
    complejo operator / (const complejo &); // Division de complejos
    complejo operator / (const double &);

    complejo & operator = (const complejo &);
    complejo operator = (const double );
    bool operator== (const complejo &);
    bool operator== (const double a);
    bool operator== (const int a);

    // Funciones
    complejo exponencial ()// Calcula la exponencial
    complejo conjugar (); // Conjugua el complejo
    complejo inversa (); // Calcula la inversa
    complejo logaritmo(); // Calcula el logaritmo
    complejo seno(); // Calcula el seno
    complejo complex_pow(const complejo); // Eleva un complejo a otro
    complejo complex_pow(const double); // Eleva un complejo a un double
    complejo re (); // Devuelve la parte real
    complejo im (); // Devuelve la parte imaginaria como imaginario puro

private:
    double real; // Dos atributos, real e imaginario
    double img;

};

ostream & operator << (ostream &, const complejo &); // Imprimie complejo en ostream
istream & operator >> (istream &, complejo &); // Lee complejo desde istream

#endif

```

complejo.h

5.7. parser.cpp

```
#include "parser.h"

using namespace std;

// Se parsea la funcion ingresada
string * parse_function(const string function, size_t & string_array_size){
    size_t i = 0;
    string * string_array;

    if (!check_balance(function)){ // Se valida que la expresion este balanceada
        cerr << "Error. No esta balanceada" << endl;
        exit(0);
    }
    if (!check_operator_at_begining(function)){ // Se valida que la funcion no comience
        con un operador * o /
        cerr << "Error, no puede comenzar con " << function[0] << endl;
        exit(0);
    }

    while (i < function.length()){ // Se recorre el string de entrada

        // Se guardan las expresiones matematicas (exp, ln, phase, abs, re, im)
        if (is_math_function_initial(function[i]) && // Si es la inicial de una funcion
            matematica y
            function[i+1]!='-' && // si el siguiente no es un guion y
            !isdigit(function[i+1])){ // si el siguiente no es un digito

            if (!parse_math_expression(function, string_array, string_array_size, i)){
                cerr << "Error. Funcion matematica erronea." << endl;
                exit(0);
            }
        }

        // Guardo numeros o numeros que empiecen con punto
        else if (isdigit(function[i]) || // Si es un digito o
                  (function[i]=='.') && isdigit(function[i+1])) // es un punto Y el
            siguiente un digito

            if (!parse_number(function, string_array, string_array_size, i)){ // Trato de
                parsear el numero
                cerr << "Error. Formato numero." << endl;
                exit(0);
            }
        }

        // Guardo numeros con el signo menos como string:
        else if (is_negative_number(function[i], function[i-1], function[i+1], i) ){

            if (!parse_negative_number(function, string_array, string_array_size, i)){
                cerr << "Error. Formato numero negativo." << endl;
                exit(0);
            }
        }

        else if (is_negative_j(function[i], function[i-1], function[i+1], i) ){

            if (!parse_negative_j(function, string_array, string_array_size, i)){
                cerr << "Error. Formato j." << endl;
                exit(0);
            }
        }
    }
}
```

```

        }

    }

    // Guardo si es operador o parentesis: +-*/^()
    else if ( is_operator(function[i]) ||           // Si es operador o
              is_parenthesis(function[i]) ) {      // es parentesis

        string aux_string = "";
        aux_string.append(1, function[i]);
        add_string_to_array(string_array, string_array_size, aux_string);
    }

    // Guardo la variable z y la j de imaginario
    else if ( function[i]=='z' ||                  // Si es 'z' o
              function[i]=='j' ){                   // si es 'j'

        string aux_string = "";
        aux_string.append(1, function[i]);
        add_string_to_array(string_array, string_array_size, aux_string);
    }
    i++;
}

return string_array;
}

// Comprueba la funcion matematica y la guarda en string array
bool parse_math_expression (const string function, string *& string_array, size_t &
                           string_array_size, size_t & position){

    string aux_string = "";

    while (function[position] != '(') { // Todas las funciones matematicas terminan con
        '('

        if (position >= function.length()) { // Si la posicion supero a el largo de la
            string
            cerr << "Error. No encontro parentesis ( parseando." << endl;
            exit (0);
        }

        aux_string.append(1, function[position]); // Se copia letra por letra
        position++;
    }

    position--;

    if (is_math_function(aux_string)){ // Se verifica que sea una funcion
        add_string_to_array(string_array, string_array_size, aux_string);
        return true;
    }
    return false;
}

// Se parsea el numero
bool parse_number (const string function, string *& string_array, size_t &
                  string_array_size, size_t & position){

    string aux_string = "";

    while( isdigit(function[position]) ||           // Mientras lo
          q leo sea un digito ||

```

```

        function[position] == '.' || // un punto ||
        (function[position] == '-' && ((function[position-1] == 'e' || function[
position-1] == 'E'))|| // un signo - precedido por una e ||
        ( (function[position] == 'e' || function[position] == 'E') &&
// (una 'e' seguida de un
        (function[position+1] == '-' || isdigit(function[position+1])) )
// signo menos o digito)
) {
    aux_string.append(1, function[position++]);
}

position--;
add_string_to_array(string_array, string_array_size, aux_string);
return true;
}

// Se pasea el numero negativo y se lo guarda
bool parse_negative_number (const string function, string *& string_array, size_t &
string_array_size, size_t & position){

string aux_string = "";
aux_string.append(1, function[position++]); // Guarda el signo menos

while( isdigit(function[position]) || // Mientras sea digito ||
function[position] == '.') { // sea un punto

if (position >= function.length()){
    cerr << "Error. No encontro parentesis ( parseando." << endl;
    return false;
}
aux_string.append(1, function[position++]);
}
position--;
add_string_to_array(string_array, string_array_size, aux_string);
return true;
}

// Se parsea la j negativa
bool parse_negative_j (const string function, string *& string_array, size_t &
string_array_size, size_t & position){

string aux_string = "";
aux_string.append(1, function[position++]); // Guarda el signo menos
aux_string.append(1, function[position++]);

position--;
add_string_to_array(string_array, string_array_size, aux_string);
return true;
}

// Funcion que agrega una string a string_array
void add_string_to_array(string *& string_array, size_t & string_array_size, const
string & str2add){

resize_string_array(string_array, string_array_size, 1);
}

```

```

    string_array[ string_array_size - 1 ] = str2add;

}

// Funcion que le agrega i lugares a string_array
bool resize_string_array (string *& string_array , size_t & string_array_size , size_t i)
{
    // pido memoria para string array de tamaño i
    string * aux_string_array;

    if (string_array_size == 0){ // Primer caso
        string_array = new string [ i ];
        string_array_size += i;
        return true;
    }

    else if (string_array_size >0){ // Luego del primer caso

        aux_string_array = new string [ string_array_size ]; // Pido memoria para el string
        array aux

        for (size_t j = 0; j < string_array_size ; j++){ // Copio el string array a un
        auxiliar
            aux_string_array [j] = string_array [j];
        }

        delete [] string_array;
        string_array = NULL;

        string_array = new string [ string_array_size + i ]; // Pido memoria para i lugar
        mas

        for (size_t j = 0; j < string_array_size ; j++){ // Recupero la informacion que
        tenia antes
            string_array [j] = aux_string_array [j];
        }

        //for (size_t j = string_array_size ; j < string_array_size+i ; j++){ // Seteo las
        nuevas posiciones en vacio
        //    string_array [j] = "";
        //}
        string_array_size += i;

        delete [] aux_string_array;
        aux_string_array = NULL;

        return true;
    }

    return true;
}

// Funcion que verifica la expresion esta balanceada
bool check_balance (const string function){
    stk <char> stack;
    bool balanced = true;
    char aux;

    for (size_t i = 0; i < function.length() && balanced == true; ++i)
{

```

```

switch(function[i]){
    case '{':
    case '[':
    case '(':
        stack.push(function[i]);
        break;
    case '}':
        if (!stack.is_empty() && (stack.peek(aux) && aux == '{'))
            stack.pop();
        else
            balanced = false;
        break;
    case ']':
        if (!stack.is_empty() && (stack.peek(aux) && aux == '['))
            stack.pop();
        else
            balanced = false;
        break;
    case ')':
        if (!stack.is_empty() && (stack.peek(aux) && aux == '('))
            stack.pop();
        else
            balanced = false;
        break;
}
if (!stack.is_empty())
    balanced = false;
return balanced;
}

// Funcion que checkea que lo ingresado no haya empezado con un operador matematico
// prohibido
bool check_operator_at_beginning(const string function){

    if (function[0]=='*' || function[0]== '/' || function[0]=='^')
    {
        return false;
    }
    return true;
}

```

parser.cpp

5.8. parser.h

```

#ifndef _PARSER_H_INCLUIDO_
#define _PARSER_H_INCLUIDO_

#include <iostream>
#include <string>
#include "is_functions.h"
#include "stk.h"

using namespace std;

string * parse_function(const string , size_t &);

bool parse_math_expression(const string , string *&, size_t &, size_t &);

```

```

bool parse_number(const string , string *&, size_t &, size_t &);
bool parse_negative_number(const string , string *&, size_t &, size_t &);
bool parse_negative_j(const string , string *&, size_t &, size_t &);

bool check_operator_at_begining(const string );
bool check_balance (const string );

bool resize_string_array (string *& , size_t &, size_t );
void add_string_to_array(string *& , size_t &, const string &);

#endif

```

parser.h

5.9. is_functions.cpp

```

#include "is_functions.h"

/* Para agregar funciones , modificar los dos vectores de funciones de abajo y modificar
LA
CANTIDAD EN IS_FUNCTIONS.H */
static char functions_initials[] = {'l', 'r', 'i', 'e', 'a', 'p', 'c', 's'};
static string math_functions[] = {"exp", "ln", "re", "im", "abs", "phase", "cos", "sin"
};

// Compara contra las iniciales de las funciones
bool is_math_function_initial (const char letter){
    size_t size = sizeof(functions_initials) / sizeof(functions_initials[0]);

    for (size_t i=0 ; i< size ; i++){
        if (letter == functions_initials[i]){
            return true;
        }
    }
    return false;
}

// Se detecta si el numero es negativo
bool is_negative_number(const char actual , const char anterior , const char sig , size_t
vuelta){
    if (
        (actual=='-' && vuelta==0 && isdigit(sig)) || // Encuentro un menos Y esta
        al principio Y lo sig es un numero O
        (actual=='-' && anterior=='(' && isdigit(sig)) || // encuentro un menos Y el
        anterior es un parentesis abierto Y lo sig es un numero O
        (actual=='-' && is_operator(anterior) && isdigit(sig)) ){ // encuentro un menos
        Y el anterior es un operador Y lo sig es un numero
        return true;
    }
    else return false;
}

// Se detecta si es una j negativa
bool is_negative_j(const char actual , const char anterior , const char sig , size_t
vuelta){
    if (
        (actual=='-' && vuelta==0 && sig == 'j') || // Encuentro un menos Y esta
        al principio O
        (actual=='-' && anterior=='(' && sig=='j') || // encuentro un menos Y el
        anterior es un parentesis abierto O

```

```

        (actual=='-' && is_operator(anterior) && sig=='j') ){ // encuentro un menos Y el
        anterior es un operador
        return true;
    }
    else return false;
}

// Se detecta si es un operador para char
bool is_operator(char token){
    return token == '+' || token == '-' ||
           token == '*' || token == '/' ||
           token == '^';
}

// Se detecta si es un operador para string
bool is_operator(string token){
    return token == "+" || token == "- " ||
           token == "*" || token == "/" ||
           token == "^";
}

// Se detecta si es un parentesis
bool is_parenthesis(char token){
    return token == ')' || token == '(' ||
           token == ']' || token == '[' ||
           token == '{' || token == '}';
}

// Se detecta si es un parentesis izquierdo
bool is_left_parenthesis(string token){
    return token == "(" ||
           token == "[" ||
           token == "{";
}

// Se detecta si es un parentesis derecho
bool is_right_parenthesis(string token){

    return token == ")" ||
           token == "]" ||
           token == "}";
}

// Se verifica si el operador tiene asociatividad izquierda
bool is_left_associative(string token){
    bool left_assoc = true;

    if (token == "+" || token == "-" ||
        token == "*" || token == "/"){
        return left_assoc;
    }
    else if (token == "^"){
        left_assoc = false;
    }

    return left_assoc;
}

```

```

// Se detecta si es una funcion matematica
bool is_math_function (string funct){

    bool aux;

    for (size_t i=0 ; i< FUNCTIONS_AMOUNT ; i++){
        if (funct == math_functions[ i ]){
            aux=true;
            return aux;
        }
    }
    return false;
}

// Se verifica si una string es un digito o una j
bool is_string_digit (string str){

    if (str == "j" || str == "-j")
        return true;

    string temp;
    stringstream ss (str);
    double aux;

    ss >> temp;
    if(stringstream(temp) >> aux){ // Si puedo convertir a double
        return true;
    }
    return false;
}

```

is_functions.cpp

5.10. is_functions.h

```

#ifndef _IS_FUNCTIONS_H_INCLUIDO_
#define _IS_FUNCTIONS_H_INCLUIDO_

#include <iostream>
#include <string>
#include <sstream>

using namespace std;

/* Modificar este valor si se desean agregar mas funciones en is_functions.cpp*/
#define FUNCTIONS_AMOUNT 8 // Cantidad de funciones matematicas implementadas (exp, ln, re..)

/* En esta libreria se encuentran las funciones utilizadas por el algoritmo de
parseo y el shunting-yard. En la misma hay funciones simples que reciben char
o strings y devuelven un booleano dependiendo lo recibido. */

bool is_math_function_initial (const char);
bool is_math_function (string);

bool is_negative_number(const char, const char, const char, size_t);
bool is_negative_j(const char , const char , const char , size_t);

```

```

bool is_operator(char);
bool is_operator(string);

bool is_parenthesis(char);
bool is_left_parenthesis(string);
bool is_right_parenthesis(string);

bool is_left_associative(string);
bool is_string_digit (string );

#endif

```

is_functions.h

5.11. stk.h

```

#ifndef _STK_H_INCLUIDO_
#define _STK_H_INCLUIDO_

#include <iostream>

using namespace std;

template<typename T>
class stk{
private:

    class nodo
    {

    friend class stk;

    nodo *sig_;
    T dato_;

public:
    nodo(T const&); 
    ~nodo(); 
};

nodo *pri_;
size_t tam_;


public:
    stk();
    stk (const stk<T>&); 
    ~stk(); 

    size_t stk_size() const;

    void pop();
    void push(T);
    bool peek(T &); 
    bool is_empty(); 
    stk<T>& operator= (const stk<T>&);


```

```

};

// Constructor de nodo
template<typename T>
stk<T>::nodo::nodo(const T &t) : sig_(0), dato_(t)
{
}

// Destructor de nodo
template<typename T>
stk<T>::nodo::~nodo()
{
}

// Constructor por defecto del stk
template<typename T>
stk<T>::stk() : pri_(0), tam_(0)
{
}

// Constructor por copia del stk
template <typename T>
stk<T>::stk (const stk<T>& stk2copy){

    nodo * aux;
    nodo * aux2;
    if (stk2copy.tam_>0)
    {
        pri_ = new nodo((stk2copy.pri_)->dato_);
        aux = (stk2copy.pri_)->sig_;
        aux2 = pri_;

        while (aux){
            aux2->sig_ = new nodo(aux->dato_);
            aux = aux->sig_;
            aux2 = aux2->sig_;
        }
        tam_=stk2copy.tam_;
    }
    else{
        this->pri_=0;
        this->tam_=0;
    }
}

// Destructor
template<typename T>
stk<T>::~stk()
{
    for (nodo *p = pri_; p; )
    {
        nodo *q = p->sig_;
        delete p;
        p = q;
    }
}

// Operador= : se la asigna a un stack el contenido del otro. Se
// crean dinamicamente los nuevos nodos
template <typename T>

```

```

stk<T>& stk<T>::operator= (const stk<T>& stk2copy){

    for (nodo *p = pri_; p; )
    {
        nodo *q = p->sig_;
        delete p;
        p = q;
    }

    nodo * aux;
    nodo * aux2;
    if (stk2copy.tam_>0)
    {
        pri_ = new nodo((stk2copy.pri_-)->dato_);
        aux = (stk2copy.pri_-)->sig_;
        aux2 = pri_;

        while (aux){
            aux2->sig_ = new nodo(aux->dato_);
            aux = aux->sig_;
            aux2 = aux2->sig_;
        }
        tam_=stk2copy.tam_;
    }
    else{
        this->pri_=0;
        this->tam_=0;
    }
    return *this;
}

// Funciones

// Pop: Desapila el ultimo elemento del stack, reasigna el puntero
// para no perder a los demas elementos y libera la memoria correspondiente
template<typename T>
void stk<T>::pop()
{
    if (pri_)
    {
        nodo * aux;
        aux = pri_->sig_;
        delete pri_;
        pri_ = aux;
        tam_--;
    }else
        return;
}

// Push: Apila un elemento en el stack, creando un nuevo nodo y
// reasignando los punteros
template<typename T>
void stk<T>::push(T t)
{
    nodo *p = new nodo(t);
    p->sig_ = pri_;

    pri_ = p;
}

```

```

        tam_++;
    }

// Peek: Toma el ultimo elemento en el stack sin desapilarlo ,
// en caso de que el stack este vacio devuelve false si no
// retorna true
template<typename T>
bool stk<T>::peek(T & t)
{
    if (!is_empty()){
        t = pri_->dato_;
        return true;
    }
    return false;
}

// Is_empty: Corrobora el estado del stack , si se encuentra o
// no vacio
template<typename T>
bool stk<T>::is_empty()
{
    if (!pri_-)
    {
        return true;
    }
    return false;
}

// Stk_size: Devuelve el tamaño del stack
template<typename T>
size_t stk<T>::stk_size() const{
    return tam_;
}

#endif

```

stk.h

5.12. shunting_yard.cpp

```

#include "shunting-yard.h"

using namespace std;

void shunting_yard(stk <string> & output_stack, string entered_function[], size_t tamano )
{
    stk <string> op_stack; // Donde se almacenaran los operadores

    string aux;

    for (int i = 0; i < (int)tamano; ++i) // Para cada token parseado anteriormente
    {

        if (is_string_digit(entered_function[i]) || entered_function[i]==”z”) // Se fija
            si es un numero o una z para ponerlo en el
        {
            // output stack
            output_stack.push(entered_function[i]);
        }
    }
}

```

```

else if(is_math_function(entered_function[i]))
{                                     // Se fija si es una funcion para ponerlo en
operator stack
    op_stack.push(entered_function[i]);
}

else if (is_operator(entered_function[i]))
{
    op_stack.peek(aux);

    bool lower_precedence = precedence(entered_function[i]) < precedence(aux); // Se
determina si el token tiene menor precedencia que el ultimo en el stack
    bool equal_precedence =precedence(entered_function[i]) == precedence(aux); // Se
determina si el token tiene igual precedencia que el ultimo en el stack

    while ((!op_stack.is_empty() && is_operator(aux)) &&
           (lower_precedence || (equal_precedence && is_left_associative(entered_function
[i]))))
        && (!is_left_parenthesis(aux)))
    {

        output_stack.push(aux);                                // Se pasa al operador o funcion al
output stack
        op_stack.pop();                                         // Se desapila el operador

        op_stack.peek(aux);

        lower_precedence = precedence(entered_function[i]) < precedence(aux); // Se
determina si el token tiene menor precedencia que el ultimo en el stack
        equal_precedence =precedence(entered_function[i]) == precedence(aux); // Se
determina si el token tiene igual precedencia que el ultimo en el stack
    }
    op_stack.push(entered_function[i]);                      // Se apila el nuevo operador en la
pila de operadores
}

else if (is_left_parenthesis(entered_function[i]))
{
    op_stack.push(entered_function[i]);
}
else if (is_right_parenthesis(entered_function[i]))
{
    if (!op_stack.peek(aux)){
        cout << "Error. Stack is empty." << endl;
        exit(1);
    }

    while ( !is_left_parenthesis(aux) ){                  // Mientras que no sea un
parentesis dereche se pasan los operadores
        // al output_stack
        output_stack.push(aux);
        op_stack.pop();
        op_stack.peek(aux);
    }
    if (!op_stack.peek(aux)){
        cout << "Error. Stack is empty." << endl;
        exit(1);
    }
    if (is_left_parenthesis(aux))
}

```

```

    {
        op_stack.pop();
        op_stack.peek(aux);

        if (is_math_function(aux)){           // En caso de que sea una funcion se la
            pasa al output_stack

            output_stack.push(aux);
            op_stack.pop();
        }
    }

}

while (!op_stack.is_empty()){           // Se pasan todos los operadores a el
    output_stack

    if (!op_stack.peek(aux)){
        cout << "Error. Stack is empty." << endl;
        exit(1);
    }
    output_stack.push(aux);
    op_stack.pop();
}
}

// Esta funcion devuelve la presedencia del operador que se le pasa por copia
int precedence (string token){
    int p = 0;

    if (token == "+" || token == "-")
        p=2;
    else if (token == "*" || token == "/")
        p=3;
    else if (token == "^")
        p=4;

    return p;
}

// Esta funcion resuelve la funcion en notacion polaca inversa. Recibe el stack con la
// funcion y el complejo
// c, que será reemplazado por z. Es importante remarcar que el resultado lo devuelve
// en el stack
void solve_rpn(stk<string> & stack , complejo c){
    string temp;
    string aux;

    if (stack.is_empty()) // Caso base
    {
        return;
    }

    else if (stack.peek(aux) && is_string_digit(aux)) // Se fija si es un numero o una
        j se crea el numero complejo
    {
        if (stack.peek(aux) && aux == "j")           // Complejo puro positivo
        {
            stack.pop();

```

```

        complejo x (0,1);
        stack.push(x.to_string());
    }
    if (stack.peek(aux) && aux == "-j")           // Complejo puro negativo
    {
        stack.pop();
        complejo x (0,-1);
        stack.push(x.to_string());
    }
    return;
}

else if (stack.peek(aux) && aux == "z")           // Se fija si es una z y la reemplaza por
{
    stack.pop();
    stack.push(c.to_string());
    return;
}

else if (stack.peek(aux) && is_operator(aux)){ // Se fija si es un operador

    string token;
    if (!stack.peek(token)){                   // Se guarda el operador en token
        cerr << "Error. Stack is empty." << endl;
        exit(1);
    }

    stack.pop();                            // Se quita el operador de la pila

    complejo x, y;

    solve_rpn(stack,c);                  // Como todos los operadores son binarios reciben
    un elemento a izquierda             // y otro a derecha, por lo tanto se llama recursivamente a
    solve_rpn                         // y se guarda lo que retorna en la cima de la pila a la
    variable right                     // (elemento de la derecha de la operacion)

    string right;

    if (!stack.peek(right)){            // Se valida que haya algo en el stack
        cerr << "Error. Stack is empty." << endl;
        exit(1);
    }

    stack.pop();                      // Se quita el valor del stack

    stringstream s1 (right);
    s1 >> x;                         // Se combierte al valor a un numero complejo

    string left;

    if (stack.is_empty()){            // Si no hay elemento a izquierda significa que
        la funcion comienza          // - o +, y de ser asi se le asigna 0 al elemento de la
        izquierda                     // izquierda
        left = "0";
    }
    else{

```

```

    solve_rpn(stack,c);           // Se llama recursivamente para obtener el valor a
izquierda del operador

    if (!stack.peek(left)){
        cerr << "Error. Stack is empty." << endl;
        exit(1);
    }

    stack.pop();

stringstream s2 (left);           // Se combierte el operando de la izquierda a
complejo
s2 >> y;                         // Segun que operador se desea aplicar se hace la cuenta
if      (token == "+") x = y+x;
else if (token == "-") x = y-x;
else if (token == "*") x = y*x;
else if (token == "^") x = y.complex_pow(x);
else if (token == "/"){           // Se valida de que no se divida por cero
    if (x == 0)
    {
        cerr << "Error. Division por 0." << endl;
        exit(1);
    }
    else
        x = y/x;
}

right = x.to_string();           // El resultado obtenido se lo combierte a string
para
stack.push(right);              // pushearlo en el stack
return;
}
else if (stack.peek(aux) && is_math_function(aux)){           // Se fija si es una funcion matematica de un solo operando
    string function;

    if (!stack.peek(function)){
        cerr << "Error. Stack is empty." << endl;
        exit(1);
    }

    stack.pop();                     // Se quita la funcion del stack

    solve_rpn(stack,c);           // Se obtiene recursivamente el operando de la
funcion

    string right;
    if (!stack.peek(right)){
        cerr << "Error. Stack is empty." << endl;
        exit(1);
    }
    stack.pop();                   // Se saca este numero del stack para ser evaluado

    complejo y;
}

```

```

stringstream s1 (right);           // Se combierte al string que estaba en el stack
(el numero)
s1 >> y;                      // a un numero complejo para evaluarlo en la funcion

                // Segun la funcion requerida se evalua el numero
if      (function == "exp")    y = y.exponencial();
else if (function == "ln")     y = y.logaritmo();
else if (function == "re")     y = y.re();
else if (function == "im")     y = y.im();
else if (function == "abs")    y = complejo(y.get_abs(),0);
else if (function == "phase")  y = complejo(y.get_phase(),0);

right = y.to_string();           // Se combierte al resultado en un string para
stack.push(right);              // pusheralo en el stack
return;

}

}

```

shunting-yard.cpp

5.13. shunting-yard.h

```

#ifndef _SHUNTING_H_INCLUIDO_
#define _SHUNTING_H_INCLUIDO_

#include <iostream>
#include <string>
#include <sstream>
#include <string>
#include "is_functions.h"
#include "stk.h"
#include "complejo.h"

static const int precedence_2 = 2;
static const int precedence_3 = 3;
static const int precedence_4 = 4;

//ostream & operator << (ostream &, const complejo &);

void shunting_yard(stk <string> &, string [], size_t);

int precedence (string token);

void solve_rpn(stk <string> &, complejo);

#endif

```

shunting-yard.h

5.14. Makefile

```

CXXARGS  = -g -Wall
CXXFLAGS = -I. $(CXXARGS)

all: image_transformer

```

```
image_transformer: main_tp1.cpp shunting_yard.cpp complejo.cpp parser.cpp image.cpp  
    is_functions.cpp main_tp1.h shunting_yard.h complejo.h parser.h image.h  
    is_functions.h stk.h  
$(CXX) $(CXXFLAGS) -o ../image_transformer main_tp1.cpp shunting_yard.cpp complejo.  
    cpp parser.cpp image.cpp is_functions.cpp  
  
clean:  
    $(RM) -vf *.o *.exe *.t *.out *.err
```

Makefile

6. Enunciado

75.04/95.12 Algoritmos y Programación II

Trabajo práctico 1: Estructuras de datos simples (DRAFT)

Universidad de Buenos Aires - FIUBA
Primer cuatrimestre de 2020
\$Date: 2020/05/28 15:49:17 \$

1. Objetivos

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

2. Alcance

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

3. Requisitos

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

4. Introducción

En este trabajo continuaremos desarrollando nuestra herramienta para procesar imágenes, de forma tal que el programa pueda recibir funciones de transformación arbitrarias.

Con esto en mente, nuestro programa deberá poder:

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

4.1. Interfaz

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

Formato. Por simplicidad se usará un formato de imágenes basado en archivos de texto: *portable graymap* o PGM, con codificación ASCII[1].

Este formato define un mapa de grises: cada pixel va a tener un valor que define su intensidad entre 0 (negro) y cierto número **max** (blanco).

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

2. Luego puede haber comentarios identificados con # al inicio de la línea. Estos comentarios deben ser ignorados por el programa.
3. Después se presenta el tamaño de la imagen. En el ejemplo de más abajo, 24 pixels de ancho y 7 de alto.
4. Una vez definido el tamaño encontramos el máximo valor de intensidad de la imagen. En el ejemplo, 15.
5. Por último está la imagen en sí: cada número define la intensidad de un pixel, comenzando en el margen superior izquierdo de la imagen y barriéndola por líneas hacia abajo.

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



Y a continuación el contenido del archivo correspondiente:

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

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

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

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

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

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

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

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

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

4.2. Línea de comando

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

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

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

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

4.3. Ejemplos

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

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

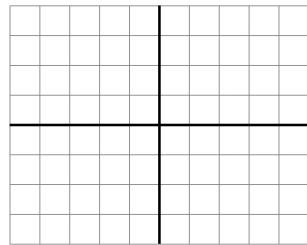


Figura 1: `grid.pgm`

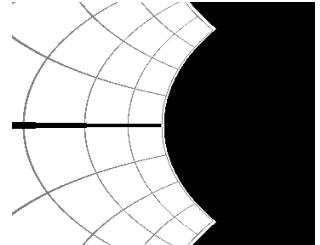


Figura 2: `grid-exp.pgm`

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

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

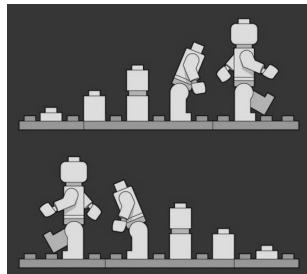


Figura 3: `evolution.pgm`

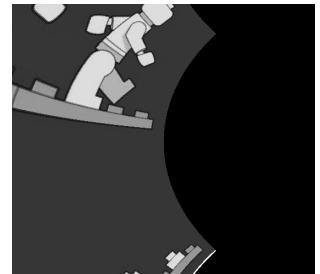


Figura 4: `evolution-exp.pgm`

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

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

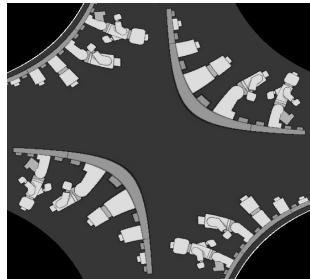


Figura 5: evolution-sqr.pgm

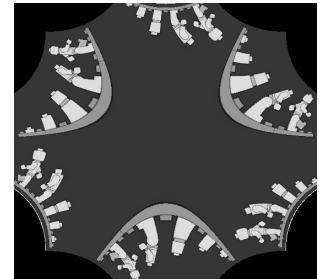


Figura 6: evolution-cube.pgm

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

4.4. Portabilidad

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

5. Informe

El contenido mínimo del informe deberá incluir:

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

6. Fechas

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

Referencias

- [1] Netpbm format (Wikipedia).
http://en.wikipedia.org/wiki/Netpbm_format
- [2] Holomorphic function (Wikipedia).
http://en.wikipedia.org/wiki/Holomorphic_function
- [3] Shunting yard algorithm (Wikipedia). http://en.wikipedia.org/wiki/Shunting-yard_algorithm