

TP0 - Mandelbrot

Juan Facundo Tkaczyszyn , *Padrón Nro. 87.931*
`facu.tk@gmail.com`

Santiago Weber, *Padrón Nro. 93.789*
`santiago.weber91@gmail.com`

2do. Cuatrimestre de 2014
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

El set de Mandelbrot es un *fractal*. A lo largo de este trabajo práctico lo analizamos, y construimos un programa que permite dibujarlo centrado y acercado a donde se le indique. Este informe refleja las consideraciones que tomamos al encarar el trabajo práctico, las pruebas y el código fuente entregable.

Índice

1. Introducción	3
1.1. Número Complejo	3
1.2. Mandelbrot	3
2. Análisis	4
2.1. Interfáz	4
2.2. Salida	4
2.2.1. Archivo/Salida Standard	4
2.2.2. Formato PGM	4
3. Diseño	5
3.1. Consideraciones	5
4. Construcción	6
4.1. Makefile	6
4.2. Pruebas Unitarias	6
5. Pruebas	7
5.1. Corridas de prueba	7
5.2. Centrado en 0, ventana de 2	7
5.3. Centrado en $-0.165+1.039i$, ventana de 0.006	8
5.4. Centrado en $-0.027+0.709i$, ventana de 0.009	9
5.5. Pruebas Unitarias	10
5.6. Emulador MIPS	10
6. Código Fuente	11
6.1. default_values.h	11
6.2. main.c	12
6.3. mandelbrot.c	13
6.4. parse_opt.c	15
7. Extras	20
7.1. Render Online	20
7.1.1. Flask	20
7.1.2. jQuery	21
7.2. Repositorio	22
8. Conclusiones	22

1. Introducción

1.1. Número Complejo

Un número complejo[1] es un número, pero diferente de los números normales.

Se puede representar juntado dos números.

La primera parte es un número real. La segunda parte de un número complejo es un número *imaginario*[2].

La parte más importante del número imaginario se la conoce como i , definida como $\sqrt{-1}$. Todos los demás números imaginarios son el número i multiplicado por un número real.

Al número complejo lo podemos escribir como $a+bi$, siendo a y b números reales.

Dado que este número tiene dos componentes, la real y la imaginaria, podemos usar esas componentes para representarlo en un sistema de coordenadas Cartesianas.

Esta representación la conocemos como plano complejo.

1.2. Mandelbrot

El set de Mandelbrot[3][4] es un fractal.

Empieza con la ecuación:

$$Z_{n+1} = Z_n^2 + c$$

Donde c y z son números complejos y n es cero o un número entero positivo.

Empezando en $z_0 = 0$, c está en el set de Mandelbrot si el valor absoluto de Z_n nunca excede cierto número.

Tomando por ejemplo, $c = 1+0i$. La secuencia es 0, 1, 2, 5, 26... que se va a infinito. Por lo tanto $1+0i$ no pertenece al conjunto.

Por otro lado, si tomamos $c = 0+1i$, la secuencia es 0, i , $(-1 + i)$, $-i$, $(-1 + i)$, $-i$, que no se va al infinito, entonces pertenece al conjunto de Mandelbrot.

La intensidad del color está dada por la cantidad de iteraciones que tiene que hacer el algoritmo hasta exceda el valor absoluto, o se alcance una cantidad máxima de iteraciones.

2. Análisis

2.1. Interfáz

El programa tiene que ser capaz de leer argumentos pasados por línea de comandos.

Para parametros como la resolucion (ej: 640x480), o el centro (ej: 1-4.5i), debe validar que se cumpla con el formato correcto y se traiga el tipo de dato correcto.

2.2. Salida

2.2.1. Archivo/Salida Standard

El programa toma el parametro de entrada y debe decidir si tiene que salir a un archivo, o salir por salida standard[5].

En caso que salga por un archivo debe validar que sea posible la escritura al mismo.

2.2.2. Formato PGM

El formato PGM[6] es una formato para almacenar información gráfica en un texto plano.

Se detalla abajo un ejemplo de un cuadrado negro sobre un fondo blanco.

```
P2                                #Header
4                                 # Cantidad de filas
4                                 # Cantidad de columnas
255                              # Maximo valor que puede tener un punto
255 225 255 255 # Matriz de puntos
255   0   0 255
255   0   0 255
255 225 255 255
```

3. Diseño

3.1. Consideraciones

El primer paso de este desarrollo fue el de discretizar el centro y ventana pedidas a una cantidad de punto finita en el plano complejo. Buscando un poco, encontramos un ejemplo sobre el cual nos basamos en Rossetta Code[7].

Para tomar los parametros que el usuario le pasa a nuestro programa por consola utilizamos `getopt_long`[8]. Luego, para validar que los argumentos pasados cumplan con los formatos esperados usamos `sscanf`[9]. Finalmente, para cada punto procesado que debemos escribir a un archivo, utilizamos `fwrite`[10], ya que su interfaz pide un identificador de archivo y el texto que vamos a escribir. Esto nos permite usar la misma función si estamos escribiendo a un archivo o a salida standard.

El modulo donde calculamos los valores para los puntos y escribimos a un archivo nos planteo un problema de diseño. Lo escribimos de forma tal que no instanciara memoria, simplemente discretiza los puntos, calcula la intensidad para cada punto y lo escribe a al archivo. Este enfoque no era testeable.

Una alternativa era separar la funcionalidad de discretización y cálculo, de la de escritura al archivo.

La alternativa por la que optamos fue aplicar Inversión de dependencias[11] en el módulo que discretiza, calcula y escribe, pasandole por parametro cual es la función que debe usar a la hora de escribir. De esta forma, al compilarlo para la entrega se utiliza la funcion `fwrite`. Pero cuando se compila para pruebas se utiliza una función `fwrite` propia, con la misma firma, pero que en vez de escribir al archivo, escribe a un buffer interno contra el cual despues comparamos los resultados esperados.

4. Construcción

4.1. Makefile

Como primer paso para asegurarnos que siempre se va a compilar igual, usando las mismas fuentes, con los mismos niveles de optimización en las diferentes plataformas donde desarrollaremos y testearamos, escribimos un Makefile[12] con tres targets: all, tests y clean.

all compila el código para la entrega, tests compila y corre las pruebas unitarias, clean borra los ejecutables compilados.

4.2. Pruebas Unitarias

Para validar todos los requisitos funcionales escribimos pruebas unitarias. Como framework de pruebas utilizamos CuTest[13] por su portabilidad al Netbsd de pruebas.

Detallamos las firmas de algunas de las pruebas que escribimos para validar los requisitos. Como el código de estas pruebas escapa al alcance de la entrega no lo incluimos en el código impreso, pero se encuentra disponible en el repositorio[21].

```
/*
 * --width w
 */
// Si le pasamos 10, esperamos 10
test_parse_width_gets_10_returns_10

// Si le pasamos una A, esperamos un error
test_parse_width_gets_A_halts

// Si le pasamos un ancho negativo, esperamos un error
test_parse_width_gets_negative_halts

/*
 * --resolution rx_ry
 */
// Si le pasamos una resolucion 16x12, esperamos 16x12
test_parse_resolution_gets_16x12_returns_16x12

// Si algun componente es negativo, esperamos error
test_parse_resolution_gets_negative_halts

/*
 * --center a+bi
 */
// Si se invoca al parametro pero vuelve vacio, esperamos error
test_parse_center_gets_empty_halts

// Si le pasamos 1-2i, esperamos 1-2i
test_parse_center_gets_1_2i_neg_returns_1_2i_neg
```

5. Pruebas

5.1. Corridas de prueba

Documentamos tres corridas de prueba. Definimos centro y tamaño de ventana y generamos una salida por consola con baja resolución, y luego una con mayor resolución que convertimos en gráfico.

5.2. Centrado en 0, ventana de 2

```
$ ./tp0 --center 0+0i --width 2 --height 2
    --resolution 14x11 --output -
```

```
P2
14
11
255
  2   2   2   3   3   4  12  44   3   2   2   1   1   1
  2   3   3   3   5   9 255  24   4   3   3   2   1   1
  3   4   5   6  10 255 255 255  30   8   5   2   2   1
  4   5   8 239 255 255 255 255 255 255  6   3   2   2
255 12  52 255 255 255 255 255 255 255  8   3   2   2
255 255 255 255 255 255 255 255 255 255  5   3   2   2
255 255 255 255 255 255 255 255 255 14   5   3   2   2
255 12  52 255 255 255 255 255 255 255  8   3   2   2
  4   5   8 239 255 255 255 255 255 255  6   3   2   2
  3   4   5   6  10 255 255 255  30   8   5   2   2   1
  2   3   3   3   5   9 255  24   4   3   3   2   1   1
```

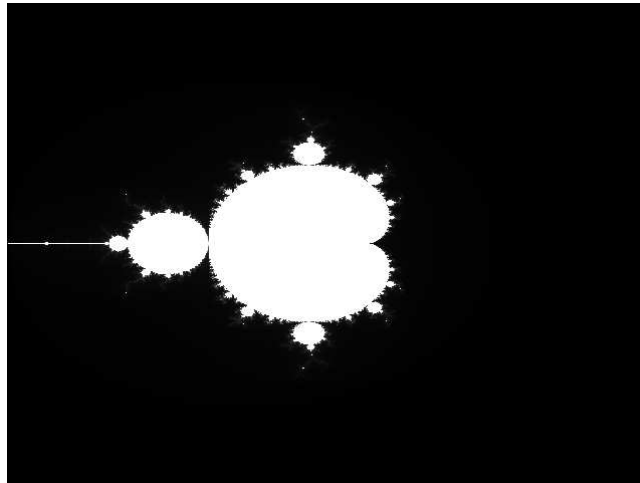


Figura 1: Mandelbrot0

5.3. Centrado en $-0.165+1.039i$, ventana de 0.006

```
$ ./tp0 --width 0.006089755361389781 --height 0.006089755361389781
--center -0.16495019360389762+1.0391402340922113i
--resolution 14x11 --output -
```

```
P2
14
11
255
26 32 26 31 24 22 21 21 23 25 22 21 22 26
47 34 41 29 25 29 24 24 26 29 24 34 25 30
30 54 49 34 31 30 27 26 31 28 27 29 34 58
27 32 35 56 41 35 36 34 36 32 36 34 72 44
22 30 39 33 40 81 45 45 53 45 49 43 55 96
22 24 26 32 37 55 127 119 123 63 69 103 117 255
39 29 28 30 37 47 84 255 255 255 255 255 255 255
22 24 27 30 52 41 103 86 255 255 255 255 255 255
21 22 24 27 35 39 78 105 255 255 255 255 255 255
21 23 31 30 34 40 50 255 255 255 255 255 255 255
23 24 26 32 52 61 76 255 255 255 255 255 255 255
```

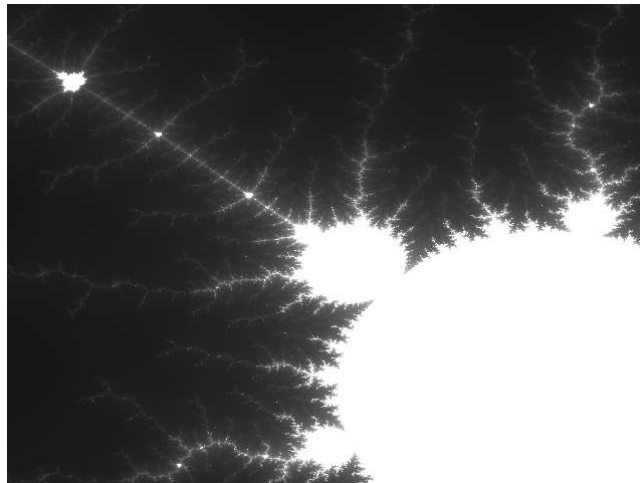


Figura 2: Mandelbrot1

5.4. Centrado en $-0.027+0.709i$, ventana de 0.009

```
$ ./tp0 --width 0.00913463304208467 --height 0.00913463304208467
--center -0.027010582808902495+0.7093001367538602i
--resolution 14x11 -output -
```

```
P2
14
11
255
255 255 255 255 255 255 107 84 88 52 39 36 39 64
255 255 255 255 255 255 86 125 61 47 40 39 66 52
255 255 255 255 255 255 255 63 59 64 42 42 44 63
255 255 255 255 255 255 228 255 230 53 45 44 45 46
255 255 255 255 255 255 255 202 140 54 110 85 49 52
255 255 255 255 255 255 255 179 74 61 58 64 81 90
255 255 255 224 255 255 164 139 105 85 100 196 105 117
255 149 217 144 73 230 158 71 154 175 92 75 62 49
255 90 78 58 53 56 81 58 81 74 97 93 115 61
194 255 255 118 48 43 45 49 70 66 97 83 49 55
58 64 84 72 47 48 38 55 52 45 43 123 38 34
```

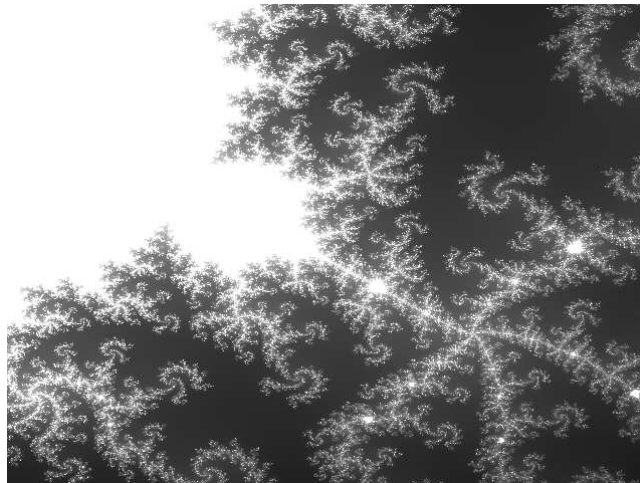


Figura 3: Mandelbrot2

5.5. Pruebas Unitarias

Para correr las pruebas unitarias, invocamos al target test de nuestro makefile. Abajo exponemos el resultado de dicha corrida.

```
make tests

./tests
.....

OK (21 tests)
```

5.6. Emulador MIPS

Para correr las pruebas sobre el NetBSD[14], corremos el GXemul[15] tal como se nos explicó en clase y luego copiamos la carpeta mediante SSH[16], con el comando SCP. Navegamos hasta la carpeta del makefile y escribimos.

```
make tests
```

6. Código Fuente

Se expone el código fuente del programa. El código fuente de las pruebas unitarias se encuentra en el repositorio[21]

6.1. default_values.h

```
/*
=====
Name      : default_values.h
Author    : Tkaczyszyn, Facundo
Version   : 1.0
Description : Default Values
=====
*/

#ifndef DEFAULT_VALUES_H
#define DEFAULT_VALUES_H

const int    default_res_x  = 640;
const int    default_res_y  = 480;
const double default_width  = 4;
const double default_height = 4;
const double default_c_re   = 0;
const double default_c_im   = 0;

#endif
```

6.2. main.c

```
/*
=====
Name      : main.c
Author    : Tkaczyszyn, Facundo
Version   : 1.0
Description : 66.20 TPO - Mandelbrot, Version 1.0
=====
*/

#include <stdio.h>
#include "default_values.h"

int main(int argc, char** argv) {
    int    res_x = default_res_x;
    int    res_y = default_res_y;
    double width = default_width;
    double height = default_height;
    double c_re  = default_c_re;
    double c_im  = default_c_im;
    FILE * output;

    int parse_result = parse_opts( argc,
                                    argv,
                                    &res_x,
                                    &res_y,
                                    &c_re,
                                    &c_im,
                                    &width,
                                    &height,
                                    &output );

    if ( parse_result == 0 ) {
        mandelbrot( res_x,
                    res_y,
                    c_re,
                    c_im,
                    width,
                    height,
                    output,
                    fprintf );

        return 0;
    }
    return 1;
}
```

6.3. mandelbrot.c

```
/*
=====
Name      : mandelbrot.c
Author    : Tkaczyszyn, Facundo
Version   : 1.0
Description : 66.20 TPO - Mandelbrot, Version 1.0
=====
*/

#include <stdio.h>
#include <math.h>

int mandelbrot( int res_x,
                int res_y,
                double c_re,
                double c_im,
                double width,
                double height,
                FILE * output,
                int (*pfprintf)(FILE * , const char *, ...) ) {

    // hack to solve issue when Resolution == 1
    if ( res_x == 1) width = 0; if ( res_y == 1) height = 0;

    const int    max_it = 255;
    const double escape_radius = 2;

    int it_x;
    int it_y;
    double c_x;
    double c_y;
    double c_x_min = c_re - ( width / 2 );
    double c_y_min = c_im - ( height / 2 );

    double px_width  = ( width )/res_x;
    double px_height = ( height )/res_y;

    double z_x, z_y;
    double z_x_sq, z_y_sq;

    int it;

    double er_sq = escape_radius*escape_radius;

    // PGM header
    (*pfprintf)( output, "P2\n%d\n%d\n%d\n", res_x, res_y, max_it );

    // iterate over the coordinates and write the data
    for( it_y = res_y; it_y > 0 ; it_y-- ) {
        c_y = c_y_min + it_y * px_height;
        if( fabs( c_y ) < px_height / 2 ) c_y = 0.0;
        for( it_x = 0 ; it_x < res_x ; it_x++ ) {
```

```

        c_x      = c_x_min + it_x * px_width;
        z_x      = c_x;
        z_y      = c_y;
        z_x_sq   = z_x * z_x;
        z_y_sq   = z_y * z_y;
        for ( it = 0; it < max_it && ((z_x_sq + z_y_sq )< er_sq ); it++) {
            z_y     = 2 * z_x * z_y + c_y;
            z_x     = z_x_sq - z_y_sq + c_x;
            z_x_sq  = z_x * z_x;
            z_y_sq  = z_y * z_y;
        };
        (*pfprintf)( output, "%3d_", it);
    }
    (*pfprintf)( output, "\n");
}
return 0;
}

```

6.4. parse_opt.c

```
/*
=====
Name      : parse_opt.c
Author    : Tkaczyszyn, Facundo
Version   : 1.0
Description : Utility module to handle command line user interface
=====
*/

#include <stdio.h>
#include <string.h>
#include <getopt.h>

const char msg_invalid_width[]      = "invalid_width_specification.";
const char msg_invalid_height[]     = "invalid_height_specification.";
const char msg_invalid_resolution[] = "invalid_resolution_specification.";
const char msg_invalid_center[]     = "invalid_center_specification.";
const char msg_output_error[]       = "Output_file_error.";

const char* const op_cortas = "r:c:w:H:o:hV";

const struct option op_largas[] = {
    { "resolution", required_argument, NULL, 'r' },
    { "center",     required_argument, NULL, 'c' },
    { "width",      required_argument, NULL, 'w' },
    { "height",     required_argument, NULL, 'H' },
    { "output",     required_argument, NULL, 'o' },
    { "help",       no_argument,      NULL, 'h' },
    { "version",    no_argument,      NULL, 'V' },
    { NULL,         no_argument,      NULL, 0 }
};

int print_stderr = 1;

void disable_error_output() {
    print_stderr = 0;
}

void print_error(const char * message ) {
    if ( print_stderr )
        fprintf( stderr, "fatal: %s\n", message );
}

int parse_width( char * param, double * result ) {
    double width;
    int scanned = sscanf( param, "%lf", &width );
    if ( scanned == 1 ) {
        if ( width > 0 ) {
            *result = width;
            return 0;
        }
    }
}
```

```

        print_error( msg_invalid_width );
        return 1;
    }

    int parse_height( char * param, double * result ) {
        double height;
        int scanned = sscanf( param, "%lf", &height );
        if ( scanned == 1 ) {
            if ( height > 0 ) {
                *result = height;
                return 0;
            }
        }
        print_error( msg_invalid_height );
        return 1;
    }

    int parse_resolution( char * param, int * res_x, int * res_y ) {
        int _res_x;
        int _res_y;

        int scanned = sscanf( param, "%dx%d", &_res_x, &_res_y );
        if ( scanned == 2 ) {
            if ( ( _res_x > 0 ) && ( _res_y > 0 ) ) {
                *res_x = _res_x;
                *res_y = _res_y;
                return 0;
            }
        }
        print_error( msg_invalid_resolution );
        return 1;
    }

    int parse_center( char * param, double * c_re, double * c_im ) {
        double _c_re;
        double _c_im;
        char _c_im_sign;

        int scanned = sscanf( param, "%lf%c%lfi" , &_c_re, &_c_im_sign, &_c_im );
        if ( scanned == 3 ) {

            if ( _c_im_sign == '-' )
                _c_im = _c_im * -1;
            *c_re = _c_re;
            *c_im = _c_im;

            return 0;
        }
        print_error( msg_invalid_center );
        return 1;
    }

    int parse_output( char * param, FILE ** output ) {
        FILE * _output = 0;

```



```

    if ( strcmp(param, "-") == 0 ) {
        _output = stdout;
    } else {
        _output = fopen( param, "wb" );
    }

    if ( _output ) {
        *output = _output;
        return 0;
    }

    print_error( msg_output_error );
    return 1;
}

void print_help( char * binary_name ) {
    printf(
        "Usage:\n"
        "%%s[options]\n"
        "\n"
        "Options:\n"
        "%%-r,%%--resolution_%%(WxH)%%Image_resolution_%%(default:%%640x480).\\n"
        "%%-c,%%--center_%%(a+bi)%%Complex_plane_center_%%(default:%%0+0i).\\n"
        "%%-w,%%--width_%%(w)%%Complex_plane_width_%%(default:%%4).\\n"
        "%%-H,%%--height_%%(h)%%Complex_plane_height_%%(default:%%4).\\n"
        "%%-o,%%--output_%%[destination]%%Path_to_output_file_%%(PGM_format).\\n"
        "%%-o,%%--output_%%[destination]%%If_%%[destination]%%is_%%-,%%outputs_%%to_%%stdout\\n"
        "%%-h,%%--help%%Print_%%this_%%message_%%and_%%quit.\\n"
        "%%-V,%%--version%%Print_%%version_%%and_%%quit.\\n"
        "\n",
        binary_name );
}

void print_version() {
    printf("66.20_%%TP0_%%Mandelbrot, %%Version_%%1.0\\n");
}

int parse_opts( int argc,
                char * const * argv,
                int * res_x,
                int * res_y,
                double * c_re,
                double * c_im,
                double * width,
                double * height,
                FILE ** output ) {

    int output_defined = 0;
    int result;

    // getopt does not print over stderr
    opterr = 0;

```

```

// every argument processed
int next_opt = 0;

while (1) {

    next_opt = getopt_long( argc,
                           argv,
                           op_cortas,
                           op_largas,
                           NULL);

    if (next_opt == -1) {
        break;
    }

    switch (next_opt) {

        case 'r': {
            if ( parse_resolution( optarg,
                                   res_x,
                                   res_y ) > 0 )
                return 1;
            break;
        }

        case 'c': {
            if ( parse_center( optarg,
                               c_re,
                               c_im ) > 0 )
                return 1;
            break;
        }

        case 'w': {
            if( parse_width( optarg,
                             width ) > 0 )
                return 1;
            break;
        }

        case 'H': {
            if( parse_height( optarg,
                              height ) > 0 )
                return 1;
            break;
        }

        case 'o': {
            if( parse_output( optarg,
                              output ) == 0 ) {
                output_defined = 1;
            } else {
                return 1;
            }
        }
    }
}

```

```

        break;
    }

    case 'h': {
        print_help( argv[0] );
        return 1;
        break;
    }

    case 'V': {
        print_version();
        return 1;
        break;
    }

    default: {
        print_help( argv[0] );
        return 1;
        break;
    }
}

if ( !output_defined ) return 1;

return 0;
}

```

7. Extras

7.1. Render Online

Luego que concluimos con la construcción y las pruebas, fuimos un paso mas y desarrollamos una interfáz para tomar parámetros via web, y la matriz de salida generada convertirla en una imagen. Se encuentra disponible en <http://home.facu.tk/mandelbrot> y el código en la carpeta del repositorio[21].

7.1.1. Flask

Desarrollamos un wrapper en Python[17] para tomar los parametros de un query string HTTP, y mapearlos a un comando de linea de comandos.

```
http://SERVER/?opcion=argumento
```

lo mapeamos a:

```
./tp0 -opcion argumento
```

Para la parte web utilizamos Flask[18], un framework de desarrollo web liviano escrito en Python.

Se detalla abajo la parte relevante del código.

```
...
@app.route("/mandelbrot.gif")
def mandelbrot():
    ...
    subprocess.call( "./tp0",
                      "-o_salida.out",
                      "-r_%s"%(request.args.get('res', '')) )
    ...
    Image.open( "salida.out" ).convert("RGB").save( "salida.gif" )
    return send_file( "salida.gif", mimetype='image/gif' )
...
```

7.1.2. jQuery

Utilizamos el framework de javascript jQuery [19] para manejar el click del usuario sobre la imagen.
Exponemos la parte relevante.

```
var res = "320x240";
var z = 4;
var zFactor = 1.5;
var c_re = 0;
var c_im = 0;
$(document).ready(function(){
    $( "#mandelmap" ).on( "click", function(e) {

        c_re = ( xpos*( z / this.width ) + ( c_re - (z / 2) ) );
        c_im = ( ( c_im + (z / 2) ) - z*( ypos / this.height ) );
        c_im_sign = ( c_im < 0 )? ':' '+';
        z = z / zFactor;

        $("#mandelmap").attr(
            "src",
            "http://localhost:5000/mandelbrot.gif" +
            "?" +
            "res=" + res +
            "&w=" + z +
            "&h=" + z + "// );
            "&center=" + c_re + c_im_sign + c_im + "i" );
    });
});
```

7.2. Repositorio

El código fuente del tp, el wrapper y este documento esta alojado en github[20].
<https://github.com/facutk/66.20>

8. Conclusiones

A lo largo del desarrollo del trabajo práctico nos fuimos familiarizando con diversas funciones y librerías standard de C que nos facilitaron la entrada y validación de parámetros. Pudimos aplicar conceptos de análisis, desarrollo y calidad aprendidos en otras materias. Ejercitamos la práctica con túneles SSH hacia un emulador. Además aprendimos mucho acerca de LaTeX[22] para realizar la redacción del informe del trabajo práctico.

Referencias

- [1] Complex Number, http://en.wikipedia.org/wiki/Complex_number
- [2] Imaginary Number, http://en.wikipedia.org/wiki/Imaginary_number
- [3] Introduction to the Mandelbrot Set, <http://www.ddewey.net/mandelbrot/>
- [4] Mandelbrot set, http://en.wikipedia.org/wiki/Mandelbrot_set
- [5] Standard streams, http://en.wikipedia.org/wiki/Standard_streams
- [6] PGM format, http://en.wikipedia.org/wiki/Netpbm_format
- [7] Mandelbrot C Renderer, http://rosettacode.org/wiki/Mandelbrot_set
- [8] getopt_long(3), http://linux.die.net/man/3/getopt_long
- [9] sscanf(3), <http://linux.die.net/man/3/sscanf>
- [10] fwrite(3), <http://man7.org/linux/man-pages/man3/fwrite.3.html>
- [11] Dependency inversion principle, http://en.wikipedia.org/wiki/Dependency_inversion_principle
- [12] Makefile, <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>
- [13] CuTest: C Unit Testing Framework, <http://cutest.sourceforge.net/>
- [14] The NetBSD Project, <http://www.netbsd.org/>
- [15] GXemul, <http://gxemul.sourceforge.net/>
- [16] Secure Shell (SSH), http://en.wikipedia.org/wiki/Secure_Shell
- [17] Python, <https://www.python.org/>
- [18] Flask Quickstart, <http://flask.pocoo.org/docs/0.10/quickstart/>
- [19] jQuery, <http://jquery.com/>
- [20] git - the simple guide, <http://rogerdudler.github.io/git-guide/>
- [21] Repositorio, <https://github.com/facutk/66.20>
- [22] LaTeX, <http://www.latex-project.org/>