# 6620. Organización de Computadoras
# Trabajo Práctico 0:
# Infraestructura Básica

Riesgo, Daniela, *Padrón Nro. 95557*
danielap.riesgo@gmail.com

Martin, Débora, *Padrón Nro. 90934*
debbie1mes.world@gmail.com

Constantino, Guillermo, *Padrón Nro. 89776*
guilleconstantino@gmail.com

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

# Índice

# Univesidad de Buenos Aires - FIUBA
# 66:20 Organización de Computadoras
# Trabajo práctico 0: Infraestructura básica
# $2^o$ cuatrimestre de 2014

`$Date: 2014/09/07 14:40:52 $`

## 1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa y su correspondiente documentación que resuelvan el problema descripto 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 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

## 4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

Durante la primera clase del curso presentaremos brevemente los pasos necesarios para la instalación y configuración del entorno de desarrollo.

## 5. Programa

Se trata de un diseñar un programa que permita dibujar el conjunto de Mandelbrot [3] [4] y sus vecindades, en lenguaje C.

El mismo recibirá, por línea de comando, una serie de parámetros describiendo la región del plano complejo y las características del archivo imagen a generar. No deberá interactuar con el usuario, ya que no se trata de un programa interactivo, sino más bien de una herramienta de procesamiento *batch*. Al finalizar la ejecución, y volver al sistema operativo, el programa habrá dibujado el fractal en el archivo de salida.

El formato gráfico a usar es PGM o *portable gray map* [6], un formato simple para describir imágenes digitales monocromáticas.

## 5.1. Algoritmo

.

El algoritmo básico es simple: para algunos puntos $c$ de la región del plano que estamos procesando haremos un cálculo repetitivo. Terminado el cálculo, asignamos el nivel de intensidad del pixel en base a la condición de corte de ese cálculo.

El color de cada punto representa la "velocidad de escape" asociada con ese número complejo: blanco para aquellos puntos que pertenecen al conjunto (y por ende la "cuenta" permanece acotada), y tonos gradualmente más oscuros para los puntos divergentes, que no pertenezcan al conjunto.

Más específicamente: para cada pixel de la pantalla, tomaremos su punto medio, expresado en coordenadas complejas, $c = c_{re} + c_{im}i$. A continuación, iteramos sobre $f_{i+1}(c) = f_i(c)^2 + c$, con $f_0(c) = c$. Cortamos la iteración cuando $|f_i(c)| > 2$, o después de `N` iteraciones.

En pseudo código:

```
para cada pixel $p {
        $f = $c = complejo asociado a $p;
        for ($i = 0; $i < $N - 1; ++$i) {
                if (abs($f) > 2)
                        break;
                $f = $f * $f + $c;
        }
        dibujar el punto p con brillo $i;
}
```

Así tendremos, al finalizar, una representación visual de la cantidad de ciclos de cómputo realizados hasta alcanzar la condición de escape (ver figura 1).

## 5.2. Interfaz

A fin de facilitar el intercambio de código *ad-hoc*, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-r`, o `--resolution`, permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.
- `-c`, o `--center`, para especificar el centro de la imagen, el punto central de la porción del plano complejo dibujada, expresado en forma binómica (i.e. $a + bi$). Por defecto usaremos $0 + 0i$.
- `-w`, o `--width`, especifica el ancho del rectángulo que contiene la región del plano complejo que estamos por dibujar. Valor por defecto: 4.

- `-H`, o `--height`, sirve, en forma similar, para especificar el alto del rectángulo a dibujar. Valor por defecto: 4.
- `-o`, o `--output`, permite colocar la imagen de salida, (en formato PGM [6]) en el archivo pasado como argumento; o por salida estándar `-cout-` si el argumento es "-".

### 5.3. Casos de prueba

Es necesario que el informe trabajo práctico incluya una sección dedicada a verificar el funcionamiento del código implementado.

En el caso del TP 0, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

Incluimos en el apéndice A algunos ejemplos de casos de interés, orientados a ejercitar algunos errores y condiciones de borde.

### 5.4. Ejemplos

Generamos un dibujo usando los valores por defecto, barriendo la región rectangular del plano comprendida entre los vértices $-2 + 2i$ y $+2 - 2i$.

```
$ tp0 -o uno.pgm
```

La figura 1 muestra la imagen `uno.pgm`.

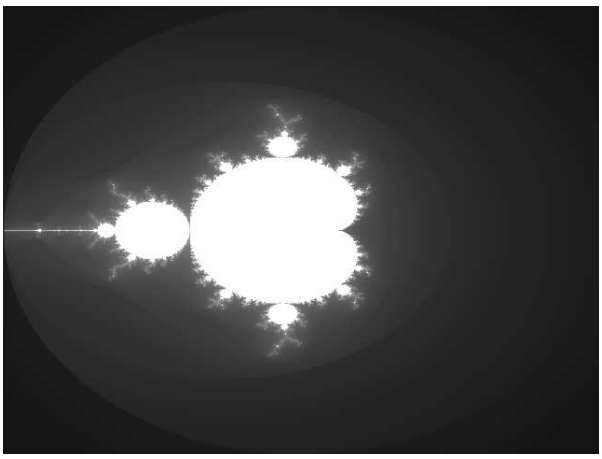

Figura 1: Región barrida por defecto.

A continuación, hacemos *zoom* sobre la región centrada en $+0{,}282 - 0{,}01i$, usando un rectángulo de 0,005 unidades de lado. El resultado podemos observarlo en la figura 2.

```
$ tp0 -c +0.282-0.01i -w 0.005 -H 0.005 -o dos.pgm
```

## 6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.

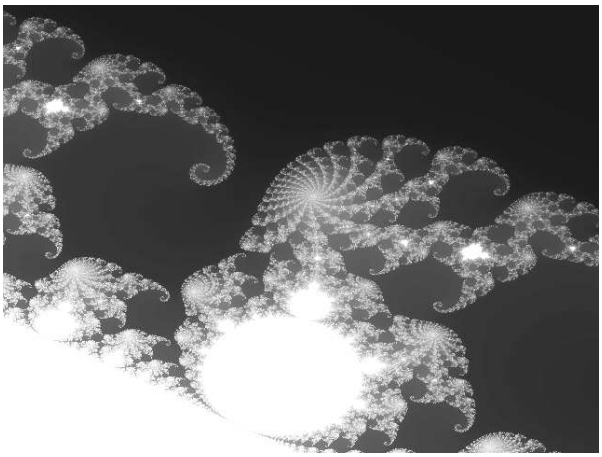

Figura 2: Región comprendida entre $0{,}2795 - 0{,}0075i$ y $0{,}2845 - 0{,}0125i$.

- 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.
- Este enunciado.

## 7. Fecha de entrega

La última fecha de entrega y presentación sería el martes 23/9.

## Referencias


[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project.
`http://www.netbsd.org/`.

[3] `http://en.wikipedia.org/wiki/Mandelbrot_set` (Wikipedia).

[4] Introduction to the Mandelbrot Set.
`http://www.olympus.net/personal/dewey/mandelbrot.html`.

[5] Smooth shading for the Mandelbrot exterior.
`http://linas.org/art-gallery/escape/smooth.html`. Linas Vepstas. October, 1997.

[6] PGM format specification.
`http://netpbm.sourceforge.net/doc/pgm.html`.

# A. Algunos casos de prueba

1. Generamos una imagen de 1 punto de lado, centrada en el orígen del plano complejo:

```
$ tp0 -c 0+0i -r 1x1 -o -
P2
1
1
255
255
```

   Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.

2. Repetimos el experimento, pero nos centramos ahora en un punto que *seguro* no pertenece al conjunto:

```
$ tp0 -c 10+0i -r 1x1 -o -
P2
1
1
255
0
```

   Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.

3. Imagen imposible:

```
$ tp0 -c 0+0i -r 0x1 -o -
Usage:
  tp0 -h
  tp0 -V
...
```

4. Archivo de salida imposible:

```
$ tp0 -o /tmp
fatal: cannot open output file.
```

5. Coordenadas complejas imposibles:

```
$ tp0 -c 1+3 -o -
fatal: invalid center specification.
```

6. Argumentos de línea de comando vacíos,

```
$ tp0 -c "" -o -
fatal: invalid center specification.
```

**Resumen**

El presente trabajo tiene como objetivo familiarizarse con las herramientas de software que serán usadas en los siguientes trabajos, implementando un programa en lenguaje C, y su correspondiente documentación, que resuelva el problema planteado, que permita dibujar el conjunto de Mandelbrot y sus vecindades

# 1. Introducción

Al comenzar a utilizar nuevas herramientas, en cualquier ámbito, es necesaria una breve introducción al funcionamiento de las mismas: tener una noción de las prestaciones que ofrecen, asi también como de sus limitaciones.
Como primer objetivo en la materia, nos proponemos adentrarnos en el funcionamiento del emulador GXemul. Nuestra meta será emular una plataforma MIPS (ejecutando un sistema operativo NetBSD), para poder desde allí desarrollar programas en lenguaje C. Estos serán compilados y ejecutados haciendo uso de la herramienta GCC (GNU Compiler Collection), mediante el cual también será posible obtener, a posteriori, el código MIPS32 del programa.
Una vez cumplido este objetivo, aprenderemos los rudimientos de LaTeX para generar la documentación relevante al trabajo práctico.

# 2. Programa a implementar

Se trata de un disenar un programa que permita dibujar el conjunto de Mandelbrot y sus vecindades, en lenguaje C. El mismo recibirá por línea de comando, una serie de parámetros describiendo la región del plano complejo y las características del archivo imagen a generar. No deberá interactuar con el usuario, ya que no se trata de un programa interactivo, sino más bien de una herramienta de procesamiento batch. Al finalizar la ejecución, y volver al sistema operativo, el programa habrá dibujado el fractal en el archivo de salida. El formato gráfico a usar es PGM o portable gray map, un formato simple para describir imágenes a digitales monocromáticas.

# 3. Explicación de la Implementación

Las principales funciones y estructuras en nuestra implementación son las siguientes:

### 3.0.1. Main

Esta función se encarga de procesar las opciones ingresadas por línea de comando, verificando validez, y luego de resolver los píxeles a tratar y sus números complejos, la velocidad de escape de ellos y la escritura del archivo.

### 3.0.2. Generarción del archivo

Toma un nombre válido de archivo y sabiendo la cantidad de pixeles en filas y columnas, color de gris máximo, y sus valores de color asociados, va escribiendo en el archivo los valores de forma ordenada.

#### 3.0.3. Velocidad de escape

Para cada par (parte real, parte imaginaria) que representan a un complejo, calcula su velocidad de escape. Este valor define una intensidad de color según la condición de corte.

# 4. Generación de ejecutables y código assembly

Para generar el ejecutable del programa, debe correrse la siguiente sentencia en una terminal:

```
$ gcc -Wall -pedantic --std=c99 -c velocidad_escape.c
$ gcc -Wall -pedantic --std=c99 -c pgm.c
$ gcc -Wall -pedantic --std=c99 velocidad_escape.o pgm.o \
main.c -o tp0
```

Para generar el código MIPS32, debe ejecutarse lo siguiente:

```
$ gcc -Wall -S -pedantic --std=c99 -c velocidad_escape.c
$ gcc -Wall -S -pedantic --std=c99 -c pgm.c
$ gcc -Wall -S -pedantic --std=c99 velocidad_escape.o pgm.o \
main.c -o tp0
```

Nótese que para ambos casos se han activado todos los mensajes de 'Warning' (-Wall). Además, para el caso de MIPS, se ha habilitado '-S', que detiene al compilador luego de generar el assembly.

# 5. Corridas de prueba

En esta sección se presentan algunas de las distintas corridas que se realizaron para probar el funcionamiento del trabajo práctico.

1. Generamos una imagen de 1 punto de lado, centrada en el orígen del plano complejo:

```
> ./tp0 -c 0+0i -r 1x1 -o -
P2
1
1
255
255
```

Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.

2. Repetimos el experimento, pero nos centramos ahora en un punto que seguro no pertenece al conjunto:

```
> ./tp0 -c 10+0i -r 1x1 -o -
P2
1
1
255
0
```

Notar que el resultado es correcto, ya que este punto pertenece al conjunto de Mandelbrot.
3. Imagen imposible:

```
> ./tp0 -c 0+0i -r 0x1 -o -
fatal: invalid resolution specification
Usage:
tp0 -H, for height of the complex plane (4 by default)
tp0 -w, for width of the complex plane (4 by default)
tp0 -r, for resolution of the image: AxB format with A and B natural
numbers (640x480 by default)
tp0 -c, for centre of the complex plane: A+Bi format with A and B
numbers (0+0i by default)
tp0 -o, for pgm output file (stdout by default)
```

4. Archivo de salida imposible:

```
> ./tp0 -o /tmp
fatal: cannot open output file
Usage:
tp0 -H, for height of the complex plane (4 by default)
tp0 -w, for width of the complex plane (4 by default)
tp0 -r, for resolution of the image: AxB format with A and B natural
numbers (640x480 by default)
tp0 -c, for centre of the complex plane: A+Bi format with A and B
```

```
numbers (0+0i by default)
tp0 -o, for pgm output file (stdout by default)
```

5. Coordenadas complejas imposibles:

```
> ./tp0 -c 1+3 -o -
fatal: invalid center specification
Usage:
tp0 -H, for height of the complex plane (4 by default)
tp0 -w, for width of the complex plane (4 by default)
tp0 -r, for resolution of the image: AxB format with A and B natural
numbers (640x480 by default)
tp0 -c, for centre of the complex plane: A+Bi format with A and B
numbers (0+0i by default)
tp0 -o, for pgm output file (stdout by default)
```

6. Argumentos de línea de comando vacíos,

```
> ./tp0 -c "" -o -
fatal: invalid center specification
Usage:
tp0 -H, for height of the complex plane (4 by default)
tp0 -w, for width of the complex plane (4 by default)
tp0 -r, for resolution of the image: AxB format with A and B natural
numbers (640x480 by default)
tp0 -c, for centre of the complex plane: A+Bi format with A and B
numbers (0+0i by default)
tp0 -o, for pgm output file (stdout by default)
```

7. Imagen PGM

```
> ./tp0 -o por_default.pgm
```

Genera la siguiente imagen:

8. Imagen PGM con región no centrada y un rectángulo de 0,005 unidades de lado.

```
> ./tp0 -c +0.282-0.01i -w 0.005 -H 0.005 -o zoom.pgm
```

Genera la siguiente imagen:

9. Se puede ver en la sección del código fuente otras pruebas unitarias hechas para probar la función de velocidad de escape en el archivo pruebas_escape.c.

```
> gcc -Wall -pedantic --std=c99 pruebas_escape.c \
-o prueba_escape
> ./prueba_escape
Vel de escape de 0+0i es 255: OK
Vel de escape de 10+0i es 0: OK
Vel de escape de -1+0i es 255: OK
Vel de escape de 0,5+0i es 4: OK
Vel de escape de 1+0i es 2: OK
```

10. De la misma forma para pruebas unitarias respecto a la lógica del escalamiento de la imagen, se tiene el archivo pruebas_escala.c.

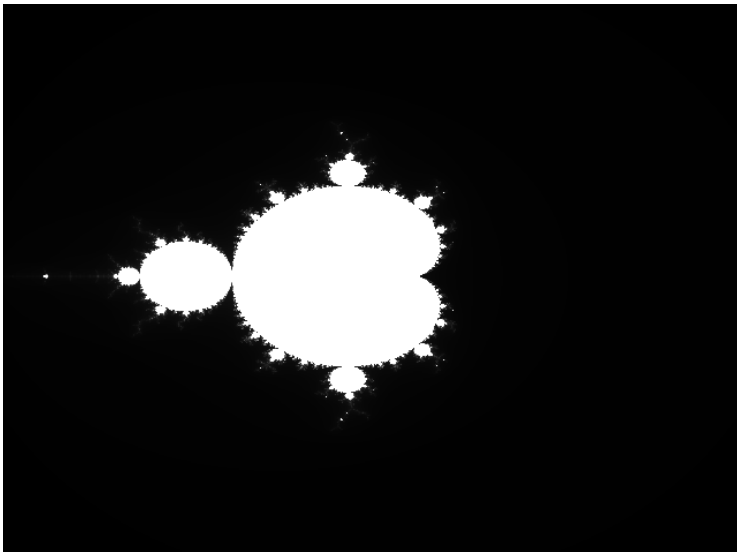

Figura 1:

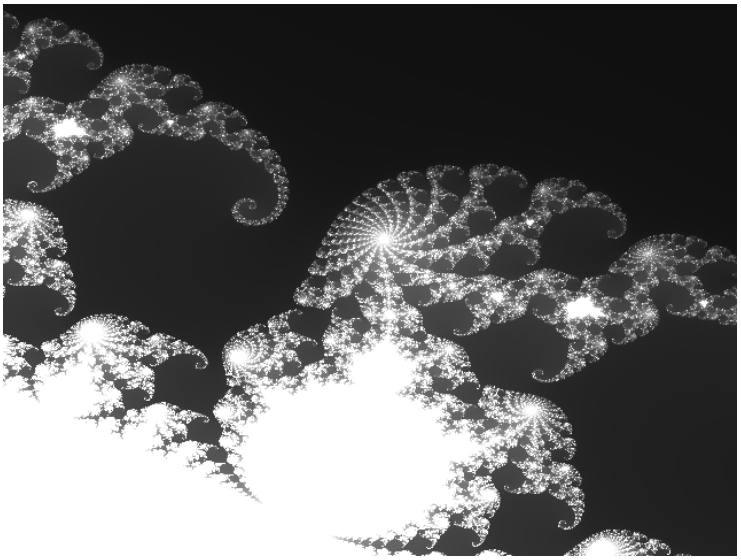

Figura 2:

```
> gcc -Wall -pedantic --std=c99 pruebas_escala.c \
-o prueba_escala
> ./prueba_escala
Complejos para -c 0+0i -r 2x2 -w 4 -H 4:
-1.000000 + 1.000000 i
1.000000 + 1.000000 i
-1.000000 + -1.000000 i
1.000000 + -1.000000 i
El primero es correcto: OK
El segundo es correcto: OK
El tercero es correcto: OK
El cuarto es correcto: OK
Complejos para -c 10+0i -r 1x1 -w 4 -H 4:
10.000000 + 0.000000 i
El primero es correcto: OK
Complejos para -c -3.5+1i -r 1x1 -w 4 -H 4:
-3.500000 + 1.000000 i
El primero es correcto: OK
Complejos para -c 1-3i -r 2x1 -w 2 -H 3:
0.500000 + -3.000000 i
1.500000 + -3.000000 i
El primero es correcto: OK
El segundo es correcto: OK
```

# 6. Código fuente C

## 6.1. main.c

```
#include <stdlib.h>
#include <getopt.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#define DEFAULT_RESOLUTION_WIDTH 640
#define DEFAULT_RESOLUTION_HEIGHT 480
#define DEFAULT_CENTER_REAL 0
#define DEFAULT_CENTER_IMAG 0
#define DEFAULT_PLANE_WIDTH 4
#define DEFAULT_PLANE_HEIGHT 4
#define DEFAULT_MAX_GRAY 255
#define ARG_DEFAULT_OUT "-"

int main(int argc, char* argv[]){
    static struct option long_options[] =
    {
        {"resolution", required_argument, 0, 'r'},
        {"center", required_argument, 0, 'c'},
        {"width", required_argument, 0, 'w'},
```

```
        {"height", required_argument, 0, 'H'},
        {"output", required_argument, 0, 'o'},
        {0, 0, 0, 0}
    };

    int resolution[2];
    float center[2];
    float plane[2];
int max_gray;
    FILE* output;

// La inicializo por default
resolution[0] = DEFAULT_RESOLUTION_WIDTH;
    resolution[1] = DEFAULT_RESOLUTION_HEIGHT;
    center[0] = DEFAULT_CENTER_REAL;
    center[1] =  DEFAULT_CENTER_IMAG;
    plane[0] = DEFAULT_PLANE_WIDTH;
    plane[1] = DEFAULT_PLANE_HEIGHT;
max_gray = DEFAULT_MAX_GRAY;
    output = stdout;


// Parseo y verifico argumentos.
    bool need_close = false;
    char option, i;
    int option_index;

    while ((option = getopt_long(argc, argv, "o:r:c:w:H:", long_options,
      &option_index)) != -1) {
        switch (option){
            case 'r':
                if (sscanf(optarg, "%d%*c%d", &resolution[0],
                 &resolution[1]) != 2){
                    printf("fatal: invalid resolution specification\n");
                    goto usage;
                }
                if (resolution[0] <= 0 || resolution[1] <= 0){
                    printf("fatal: invalid resolution specification\n");
                    goto usage;
                }

                break;
            case 'c':
                if (sscanf(optarg, "%f%*c%f%c", &center[0], &center[1], &i) != 3
                  || i!='i'){
                    printf("fatal: invalid center specification\n");
                    goto usage;
                }
                break;
            case 'w':
```

```
                plane[0] = atof(optarg);
                if (plane[0] <= 0){
                    printf("fatal: invalid width specification\n");
                    goto usage;
                }
                break;
            case 'H':
                plane[1] = atof(optarg);
                if (plane[1] <= 0){
                    printf("fatal: invalid height specification\n");
                    goto usage;
                }
                break;
            case 'o':
                if (strcmp(ARG_DEFAULT_OUT, optarg) != 0){
                    output = fopen(optarg, "w");
                    if (! output){
                        printf("fatal: cannot open output file\n");
                        goto usage;
                    }
                    need_close = true;
                }
                break;
            default:
            usage:
                printf("Usage:\n\ttp0 -H, for height of the complex plane
                (4 by default)\n
                \ttp0 -w, for width of the complex plane (4 by default)\n
                \ttp0 -r, for resolution of the image: AxB format with A and
                B natural numbers (640x480 by default)\n
                \ttp0 -c, for centre of the complex plane: A+Bi format with
                A and B numbers
                (0+0i by default)\n
                \ttp0 -o, for pgm output file (stdout by default)\n");
                return 1;
        }
    }

// Calculo y escribo el archivo:

// Armo escala.
double first_real_value = center[0] - plane[0]/2;
    double first_imaginary_value = center[1] + plane[1]/2;
    double width_scale = (plane[0] / resolution[0]);
    double height_scale =  - (plane[1] / resolution[1]);
    first_real_value += width_scale/2;
    first_imaginary_value += height_scale/2;

// Escribo parámetros de la imagen.
fprintf(output, "P2\n");
```

```
fprintf(output, "%d\n%d\n", resolution[0], resolution[1]);
fprintf(output, "%d\n", max_gray);

// Escribo valor de cada pixel.
int j, k;
for (j = 0; j < resolution[1]; ++j){
for (k = 0; k < resolution[0]; ++k) {

double real = first_real_value + k * width_scale;
double imag = first_imaginary_value + j * height_scale;

// Calculo velocidad de escape
int vel;
for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
//Módulo al cuadrado > 4
break;
// Sino, elevo al cuadrado y le sumo a si mismo
real = real * real - imag * imag + real;
imag = 2 * real * imag + imag;
}

fprintf(output, "%d ", vel);
}
fprintf(output, "\n");
}
// Terminé de escribir el archivo.


    if (need_close) fclose(output);
    return 0;
}
```

## 6.2. pruebas_escala.c

```
#include <stdio.h>

void imprimir_complejos_del_plano(double centro_real, double centro_imag,
double ancho_plano, double alto_plano, int ancho_res, int alto_res, double* v){

double first_real_value = centro_real - ((float)ancho_plano)/2;
    double first_imaginary_value = centro_imag + ((float)alto_plano)/2;
    double width_scale = (((float) ancho_plano) / ancho_res);
    double height_scale =  - (((float) alto_plano) / alto_res);
    first_real_value += width_scale/2;
    first_imaginary_value += height_scale/2;

int contador = 0;
    for(int i = 0; i < alto_res; i++){
        for(int j = 0; j < ancho_res; j++){
```

```
double real = first_real_value + j * width_scale;
double imag = first_imaginary_value + i * height_scale;

printf("%f + %f i\n", real, imag);

v[contador] = real;
contador++;
v[contador] = imag;
contador++;
}
}

}

int main(){

printf("Complejos para -c 0+0i -r 2x2 -w 4 -H 4:\n");
    double v1[8];
    imprimir_complejos_del_plano(0, 0, 4, 4, 2, 2, v1);
printf("El primero es correcto: %s\n", ((v1[0] == -1) & (v1[1] == 1))?
 "OK" : "ERROR");
printf("El segundo es correcto: %s\n", ((v1[2] == 1) & (v1[3] == 1))?
"OK" : "ERROR");
printf("El tercero es correcto: %s\n", ((v1[4] == -1) & (v1[5] == -1))?
 "OK" : "ERROR");
printf("El cuarto es correcto: %s\n", ((v1[6] == 1) & (v1[7] == -1))?
"OK" : "ERROR");

printf("Complejos para -c 10+0i -r 1x1 -w 4 -H 4:\n");
    double v2[2];
    imprimir_complejos_del_plano(10, 0, 4, 4, 1, 1, v2);
printf("El primero es correcto: %s\n", ((v2[0] == 10) & (v2[1] == 0))?
 "OK" : "ERROR");

printf("Complejos para -c -3.5+1i -r 1x1 -w 4 -H 4:\n");
    double v3[2];
    imprimir_complejos_del_plano(-3.5, 1, 4, 4, 1, 1, v3);
printf("El primero es correcto: %s\n", ((v3[0] == - 3.5) & (v3[1] == 1))?
 "OK" : "ERROR");

printf("Complejos para -c 1-3i -r 2x1 -w 2 -H 3:\n");
    double v4[4];
    imprimir_complejos_del_plano(1, -3, 2, 3, 2, 1, v4);
printf("El primero es correcto: %s\n", ((v4[0] == 0.5) & (v4[1] == -3))?
 "OK" : "ERROR");
printf("El segundo es correcto: %s\n", ((v4[2] == 1.5) & (v4[3] == -3))?
 "OK" : "ERROR");

return 0;
}
```

### 6.3. pruebas_escala.c

```
#include <stdio.h>

int main(){
        double real, imag;
        int vel;

        real = 0;
        imag = 0;

        for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
            //Módulo al cuadrado > 4
break;
            // Sino, elevo al cuadrado y le sumo a si mismo
            real = real * real - imag * imag + real;
            imag = 2 * real * imag + imag;
        }

        printf("%s: %s\n", "Vel de escape de 0+0i es 255", (vel == 255)?
         "OK" : "ERROR");

        real = 10;
        imag = 0;

        for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
            //Módulo al cuadrado > 4
break;
            // Sino, elevo al cuadrado y le sumo a si mismo
            real = real * real - imag * imag + real;
            imag = 2 * real * imag + imag;
        }

        printf("%s: %s\n", "Vel de escape de 10+0i es 0", (vel == 0)?
         "OK" : "ERROR");

        real = -1;
        imag = 0;

        for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
            //Módulo al cuadrado > 4
break;
            // Sino, elevo al cuadrado y le sumo a si mismo
            real = real * real - imag * imag + real;
            imag = 2 * real * imag + imag;
        }
```

```
        printf("%s: %s\n", "Vel de escape de -1+0i es 255", (vel == 255)?
         "OK" : "ERROR");

        real = 0.5;
        imag = 0;

        for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
            //Módulo al cuadrado > 4
break;
            // Sino, elevo al cuadrado y le sumo a si mismo
            real = real * real - imag * imag + real;
            imag = 2 * real * imag + imag;
        }


        printf("%s: %s\n", "Vel de escape de 0,5+0i es 4", (vel == 4)?
         "OK" : "ERROR");

        real = 1;
        imag = 0;

        for (vel = 0; vel < max_gray; ++vel) {
if ((real * real + imag * imag) > 4)
            //Módulo al cuadrado > 4
break;
            // Sino, elevo al cuadrado y le sumo a si mismo
            real = real * real - imag * imag + real;
            imag = 2 * real * imag + imag;
        }


        printf("%s: %s\n", "Vel de escape de 1+0i es 2", (vel == 2)?
         "OK" : "ERROR");
}
```

# 7. Conclusiones

El trabajo práctivo motivo de este informe ha presentado al equipo diversos desafíos. En primer lugar cabe mencionar la adaptación a un ambiente basado en GNU/Linux, no dominado por todos sus integrantes de igual manera. También es destacable la dificultad inicial que trajo la correcta configuración del ambiente virtual utilizado para emular la máquina MIPS. Finalmente, también fue invertida una cantidad considerable de tiempo en aprender a utilizar e investigar sobre archivos PGM y principalmente LaTeX, ya que si bien permite obtener muy buenos resultados, se requiere de mucha lectura para poder aprovechar todo su potencial. Por las razones expuestas, consideramos muy necesario un trabajo práctico introductorio de esta naturaleza, para nivelar e introducir los elementos a utilizar en las demás actividades prácticas que se desarrollarán en el curso.
Como conclusión final, podemos considerar que hemos logrado obtener un buen manejo de las herramientas introducidas en este primer proyecto.

# Referencias


[1] GXemul, http://gxemul.sourceforge.net/

[2] The NetBSD Project, http://www.netbsd.org/

[3] Mandelbrot, http://en.wikipedia.org/wiki/Mandelbrot_set/

[4] Introduction to the Mandelbrot Set, http://www.olympus.net/personal/dewey/mandelbrot.html.

[5] Smooth shading for the Mandelbrot exterior. http://linas.org/art-gallery/escape/smooth.html. Linas Vepstas. October, 1997.

[6] PGM format specification. http://netpbm.sourceforge.net/doc/pgm.html

[7] Oetiker, Tobias, "The Not So Short Introduction To LaTeX2", http://www.physics.udel.edu/$\sim$dubois/lshort2e/