

UNIVERSIDAD DE BUENOS AIRES  
FIUBA



66.20 Organización de Computadoras  
Trabajo práctico 0: Infraestructura básica  
1<sup>er</sup> cuatrimestre de 2018

**ALUMNOS**

**92454 ZARAGOZA, MARTIN**  
**92691 SIBIKOWSKI, NICOLAS**  
**91985 DUFAU, EZEQUIEL**

# Índice

	Página
<b>1. Introducción</b>	<b>3</b>
<b>2. Diseño e Implementación</b>	<b>3</b>
2.1. Tipos de Datos Abstractos . . . . .	3
2.1.1. <b>Pixel</b> . . . . .	3
2.1.2. <b>Complex</b> . . . . .	4
2.1.3. <b>Arguments</b> . . . . .	5
2.2. Formate PGM . . . . .	5
<b>3. Compilación y ejecución del programa</b>	<b>7</b>
3.1. Compilación . . . . .	7
3.2. Ejecución . . . . .	7
<b>4. Corridas de prueba</b>	<b>8</b>
4.1. Valores por defecto . . . . .	8
4.2. Zoom en región . . . . .	8
4.3. Modificación de la semilla . . . . .	8
4.4. Movimiento del centro . . . . .	9
4.5. Modificación de la resolución . . . . .	9

# 1. Introducción

Este documento representa la documentación técnica del trabajo práctico 0, correspondiente a la materia **66.20 Organización de Computadoras**. En el mismo se incluíra el desarrollo de los siguientes puntos:

- Diseño e Implementación
- Compilación y ejecución del programa
- Corridas de prueba
- Código Fuente

## 2. Diseño e Implementación

Se desarrolló un programa que permite dibujar el conjunto de Julia, en lenguaje C.

### 2.1. Tipos de Datos Abstractos

Se diseñaron los siguientes TDAs:

#### 2.1.1. Pixel

Define un punto en el plano complejo

```
typedef struct {
    unsigned x;
    unsigned y;
} Pixel;

typedef struct {
    unsigned width;
    unsigned height;
} Dimension;

typedef Dimension Resolution;

void parseRes(char* str , Resolution* targetRes);

#endif
```

### 2.1.2. Complex

Define el TDA para manejar números complejos

```
typedef struct {
double re;
double im;
} Complex;

typedef struct {
double left;
double right;
double top;
double bottom;
} Boundaries;

Complex newCpx(double re , double im);

/**
 * Obtiene un numero complejo a partir de un string
 */
void parseCpx(char* str , Complex* targetCpx);

/**
 * Obtiene la raiz cuadrada de un complejo
 */
Complex pow2Cpx(Complex* value);

Complex addCpx(Complex* first , Complex* second);

/**
 * Obtiene el modulo de un valor complejo
 * (distancia respecto al origen).
 */
double modCpx(Complex* value);

/**
 * Calcula los limites del plano complejo
 * (el rectangulo a mapear del plano complejo).
 * Dimension dim – Tamano del plano complejo
 * Complex center – Valor de centro EN EL plano
 */
```

```
Boundaries getBoundaries(Complex* dim , Complex* center );
#endif
```

### 2.1.3. Arguments

Define una estructura que permite manejar los parámetros del programa

```
#include "pixel.h"
#include "complex.h"

typedef struct {
    Resolution resolution; // resolucion de imagen
    Complex center;        // centro del plano complejo
    Complex cpxSize;        // tamano del plano complejo
    Complex seed;          // semilla disparadora
    char* outfile;         // archivo de salida
} Arguments;

Arguments parseArgs(int argc , char** args);

#endif
```

## 2.2. Formate PGM

El formato utilizado para generar la imagen es el PGM. En sus especificaciones establece que ninguna línea debe tener más de 70 caracteres. Debido a esto se tomó la decisión de que el programa genere el número óptimo de caracteres que debe tener cada línea en el archivo. El número máximo de cada valor es 255 aproximadamente 3 caracteres, teniendo en cuenta los 70 caracteres por línea el máximo de valores es 16. Su funcionamiento se puede ver en el siguiente código

```
unsigned getOptIndex(unsigned itemCount) {
    unsigned idx = 16;
    while(itemCount % idx — > 0);
    return idx + 1;
}
```

Donde itemCount sale de hacer

```
itemCount = resolution.width * resolution.height;
```

## 3. Compilación y ejecución del programa

Utilizamos el programa GXemul para simular un entorno de desarrollo. El mismo consta de una máquina MIPS corriendo una versión del sistema operativo BSD. Debido a esto, el compilador utilizado es el que trae instalada la imagen provista por la cátedra. El mismo es el **(GCC) 3.3.3 (NetBSD nb3 20040520)**

### 3.1. Compilación

Para facilitar el proceso de compilación, programamos un Makefile, seteando las opciones correspondientes. Se debe ejecutar make en el directorio raíz del proyecto y el programa queda compilado. Se genera un ejecutable llamado app

### 3.2. Ejecución

Se debe ejecutar  
./app.exe [OPCIONES]

#### Opciones:

-r, o -resolution, permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.

-c, o -center, para especificar las coordenadas correspondientes al 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 de la región del plano complejo que estamos por dibujar. Valor por defecto: 2.

-H, o -height, sirve, en forma similar, para especificar el alto del rectángulo a dibujar. Valor por defecto: 2.

-s, o -seed, para configurar el valor complejo de la semilla usada para generar el fractal. Valor por defecto:  $-0.726895347709114071439+0.188887129043845954792i$ .

-o, o -output, permite colocar la imagen de salida, (en formato PGM [6])

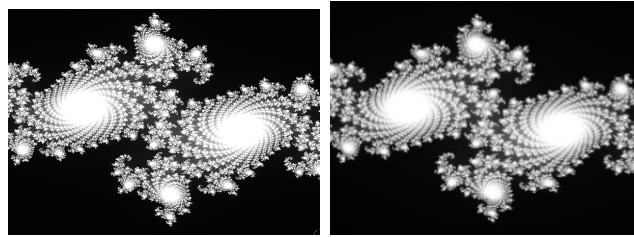
en el archivo pasado como argumento; o por salida estándar `-cout` si el argumento es “-”.

## 4. Corridas de prueba

### 4.1. Valores por defecto

Se generó un dibujo usando los valores por defecto, barriendo la región rectangular del plano comprendida entre los vértices  $-2+2i$  y  $+2-2i$ . La validación fue visual. Comparando contra un dibujo patrón disponible en el enunciado.

**Comando ejecutado:** `tp0 -o uno.pgm`      Resultado de la prueba:



(a) nuestro programa

(b) trabajo práctico

Figura 1: Comparación entre imagen generada por el enunciado del tp, y nuestro programa

### 4.2. Zoom en región

Hacemos zoom sobre la región centrada en  $0.282-0.007i$ , usando un rectángulo de 0.005 unidades de lado.

**Comando ejecutado:** `tp0 -c 0.282-0.007i -w 0.005 -H 0.005 -o dos.pgm`  
Resultado de la prueba:

### 4.3. Modificación de la semilla

En esta prueba vamos a modificar la semilla utilizando el comando `-s`. Apoyandonos mediante otra implementación del Algoritmo de Julia, va-



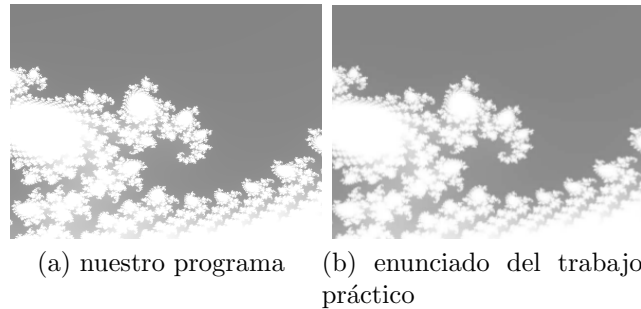


Figura 2: Comparación entre la imagen generada por el enunciado del tp, y nuestro programa

mos a comparar la imagen generada por nuestro programa coon la generada por la otra implementación. La implementación de Julia la obtuvimos en <https://www.wolframalpha.com/juliaset.html> Utilizamos la semilla  $-0.742+0.1i$ .

Ejecutamos: `./tp -o salida.pgm -s -0.742+0.1i`

Resultado de la prueba:

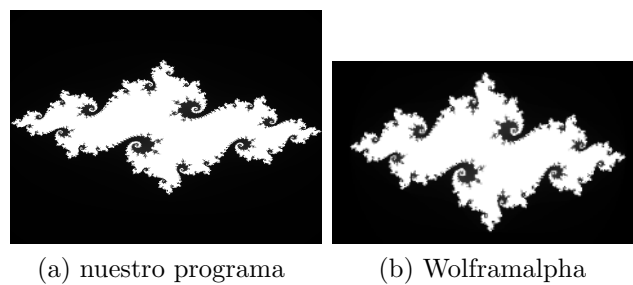


Figura 3: Comparación entre otra implementación del algoritmo de Julia y nuestro programa

#### 4.4. Movimiento del centro

En esta prueba vamos a generar cuatro imagenes(una para cada cuadrante) de la imagen utilizada en la prueba de valores por defecto.

#### 4.5. Modificación de la resolución

En esta prueba se modificará la resolución de la pantalla a  $200 \times 200$  px. utilizando el comando `-r`

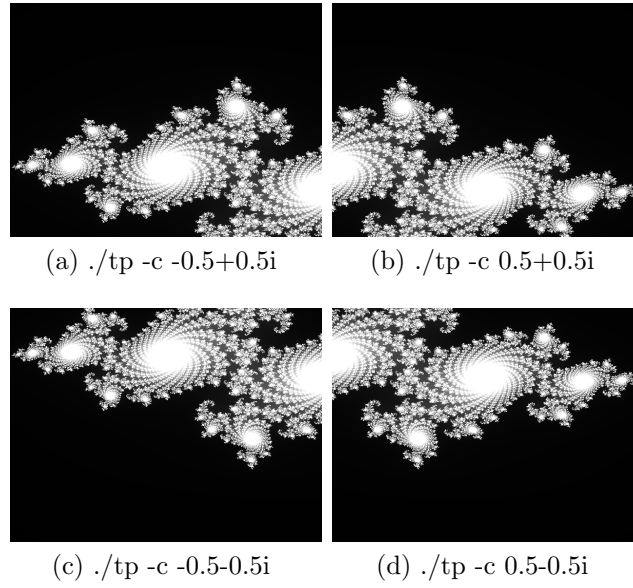
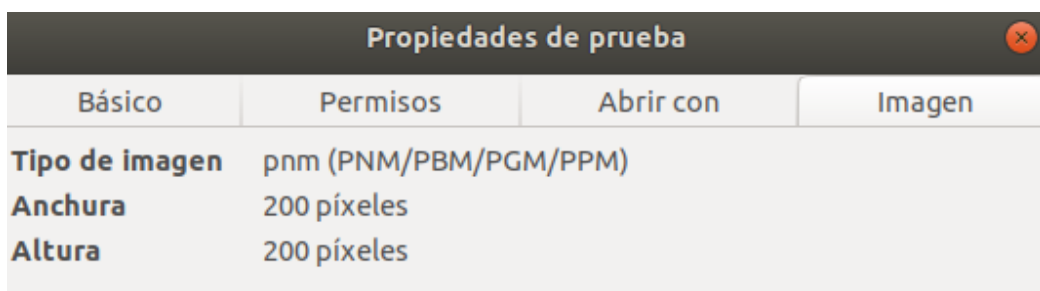
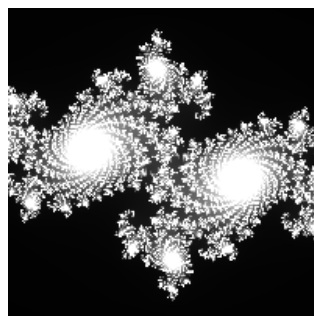


Figura 4: Diferentes imagenes que permiten observar como se fue moviendo el centro según cada situación

Prueba ejecutada Ejecutamos:  $./tp -o salida.pgm -r 200*200$



(a) Resolución del archivo creado



(b) Imagen creada

Figura 5: Resultado de la prueba