UNIVERSIDAD DE BUENOS AIRES
FIUBA

66.20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
1^{er} cuatrimestre de 2018

ALUMNOS

Padron	Alumno
92454	ZARAGOZA, MARTIN
92691	SIBIKOWSKI, NICOLAS
91985	DUFAU, EZEQUIEL

Índice

Página

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 incluira el desarrollo de los siguientes puntos:

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

1.1. Enunciado

Univesidad de Buenos Aires - FIUBA
66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
1^{er} cuatrimestre de 2018

`$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 Julia [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 f_0 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, $f_0 = \mathrm{Re}(f_0) + \mathrm{Im}(f_0)i$. A continuación, iteramos sobre $f_{i+1}(c) = f_i(c)^2 + s$, cortando la iteración cuando $|f_i(c)| > 2$, o después de `N` iteraciones.

En pseudo código:

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

Aquí, el parámetro s representa la semilla o *seed* usada para generar el fractal. Se trata de una constante compleja que nos permite parametrizar la forma del fractal, cuyo valor por defecto está especificado en la sección 5.2.

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 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 "-".

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 este enunciado dos fractales de referencia que pueden ser usados para comprobar visualmente el funcionamiento del programa.

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`.

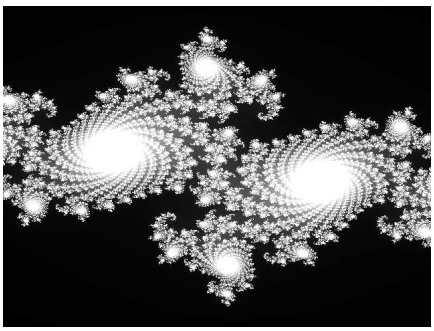


Figura 1: Región barrida por defecto.

A continuación, hacemos *zoom* sobre la región centrada en `0.282-0.007i`, usando un rectángulo de `0.005` unidades de lado.

```
$ tp0 -c 0.282-0.007i -w 0.005 -H 0.005 -o dos.pgm
```

El resultado podemos observarlo en la figura 2.

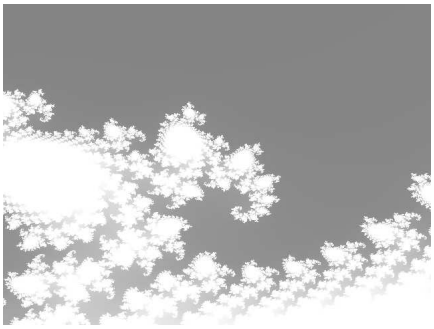


Figura 2: Región comprendida entre $0{,}2795 - 0{,}0045i$ y $0{,}2845 - 0{,}0095i$.

6. Informe

El informe deberá incluir:

- 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.
- Este enunciado.

7. Fecha de entrega

La última fecha de entrega y presentación sería el martes 10/4.

Referencias

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

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

[3] `http://es.wikipedia.org/wiki/Conjunto_de_Julia` (Wikipedia).

[4] `http://mathworld.wolfram.com/JuliaSet.html` (Mathworld).

[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`.

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

```
1  typedef struct {
2  unsigned x;
3  unsigned y;
4  } Pixel;
5
6  typedef struct {
7  unsigned width;
8  unsigned height;
9  } Dimension;
10
11 typedef Dimension Resolution;
12
13 void parseRes(char* str , Resolution* targetRes);
14
15 #endif
```

2.1.2. Complex

Define el TDA para manejar números complejos

```
1  typedef struct {
2  double re;
3  double im;
4  } Complex;
5
6  typedef struct {
7  double left;
8  double right;
9  double top;
10 double bottom;
11 } Boundaries;
12
13 Complex newCpx(double re , double im);
```

```
14
15 /**
16 * Obtiene un numero complejo a partir de un string
17 */
18 void parseCpx(char* str , Complex* targetCpx);
19
20 /**
21 * Obtiene la raiz cuadrada de un complejo
22 */
23 Complex pow2Cpx(Complex* value);
24
25 Complex addCpx(Complex* first , Complex* second);
26
27 /**
28 * Obtiene el modulo de un valor complejo
29   (distancia respecto al origen).
30 */
31 double modCpx(Complex* value);
32
33 /**
34 * Calcula los limites del plano complejo
35    (el rectangulo a mapear del plano complejo).
36 * Dimension dim - Tamano del plano complejo
37 * Complex center - Valor de centro EN EL plano
38 */
39 Boundaries getBoundaries(Complex* dim , Complex* center);
40
41 #endif
```

2.1.3. Arguments

Define una estructura que permite manejar los parámetros del programa

```
1 #include "pixel.h"
2 #include "complex.h"
3
4 typedef struct {
5 Resolution resolution;  // resolucion de imagen
6 Complex center;         // centro del plano complejo
7 Complex cpxSize;        // tamano del plano complejo
8 Complex seed;           // semilla disparadora
9 char* outfile;          // archivo de salida
10 } Arguments;
11
12 Arguments parseArgs(int argc, char** args);
13
14 #endif
```

2.2. Formato PGM

El formato utilizado para generar la imagen es el PGM. En sus especificaciones establece que ninguna linea 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 linea 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

```
1 unsigned getOptIndex(unsigned itemCount) {
2   unsigned idx = 16;
3   while(itemCount %idx--- > 0);
4   return idx + 1;
5 }
```

Donde itemCount sale de hacer

```
1 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 raiz del proyecto y el programa queda compilado. Se genera un ejecutable llamado app.exe

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

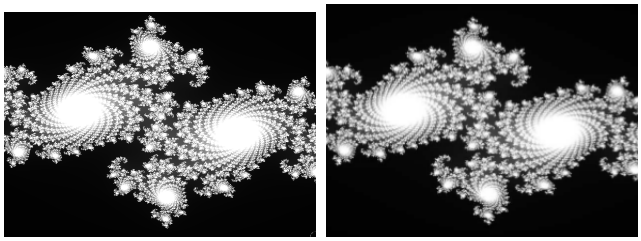
4.1. Valores por defecto

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(Figura1):



(a) nuestro programa (b) trabajo práctico

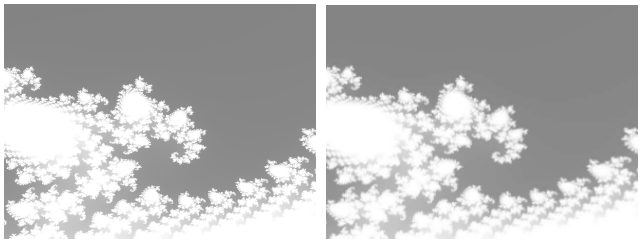
Figura 1: Comparación entre imagen generada por el enunciado del tp, y nuestro programa

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 (Figura 2):



(a) nuestro programa (b) enunciado del trabajo práctico

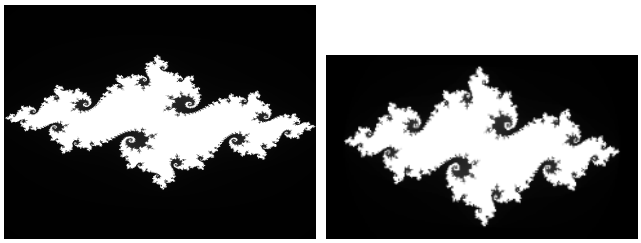
Figura 2: Comparación entre la imagen generada por el enunciado del tp, y nuestro programa

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, vamos a comparar la imagen generada por nuestro programa con 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):



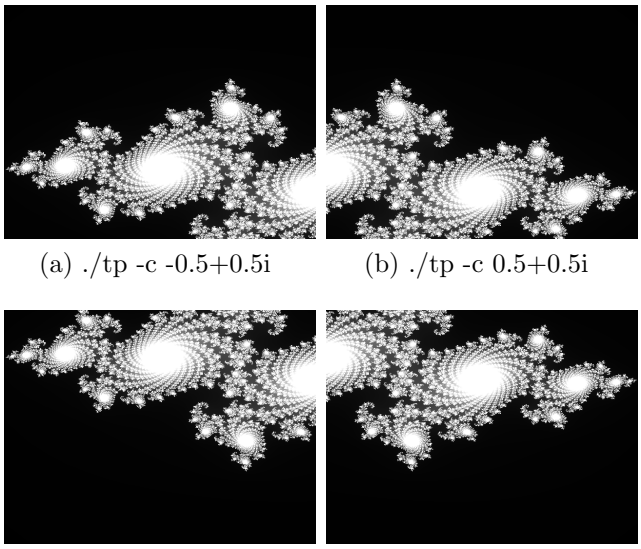
(a) nuestro programa (b) Wolframalpha

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.

Resultado de la prueba (Figura 4).



(a) ./tp -c -0.5+0.5i (b) ./tp -c 0.5+0.5i

(c) ./tp -c -0.5-0.5i (d) ./tp -c 0.5-0.5i

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

4.5. Modificación de la resolución

En esta prueba se modificará la resolución de la pantalla a 200x200 px. utilizando el comando -r. Figura 5.
Prueba ejecutada

```
./tp -o salida.pgm -r 200x200
```

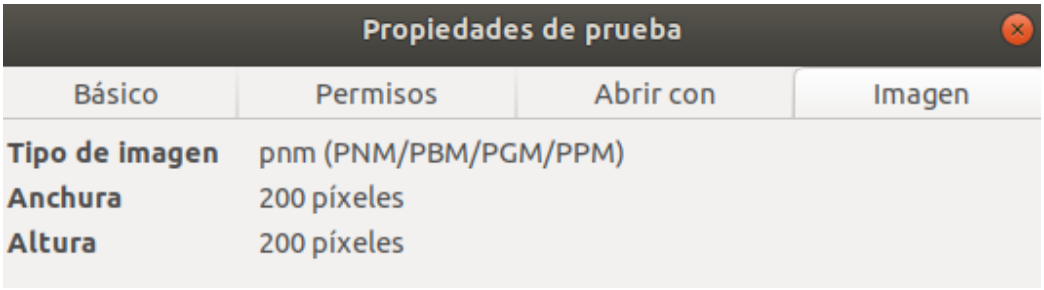
4.6. Salida estandar

Vamos a probar el comando -o -. La idea es obtener por salida estandar la imagen. Para tal fin se ejecutó el siguiente comando

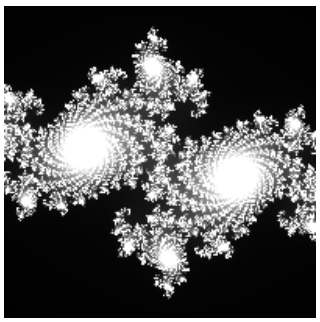
```
./app.exe -o -
```

De esta manera obtenemos por pantalla la salida correspondiente a una imagen del conjunto de Julia con las opciones por defecto. Figura 6.

Otra prueba que hicimos para corroborrar el correcto funcionamiento fue ejecutar



(a) Resolución del archivo creado



(b) Imagen creada

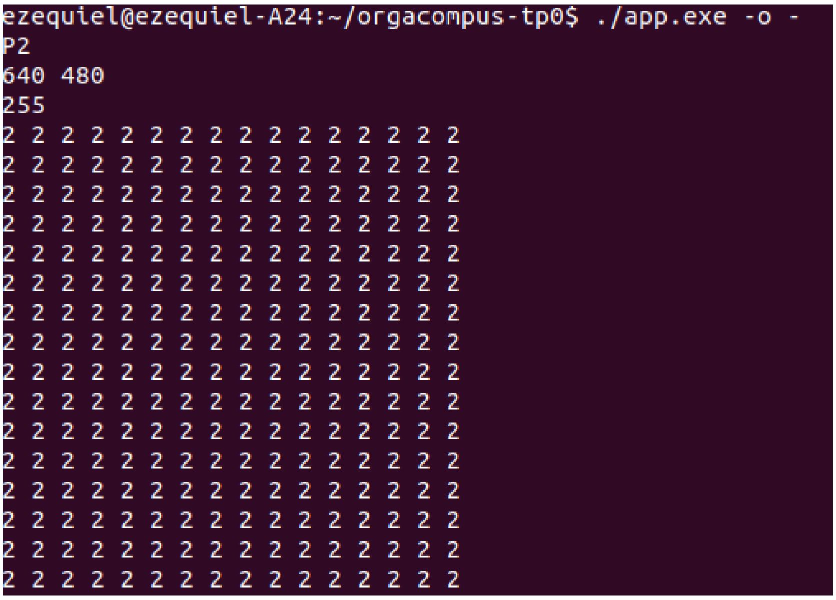
Figura 5: Resultado de la prueba

```
./app.exe -o salida.pgm
```

Y luego:

```
./app.exe -o - > salida2.pgm.
```

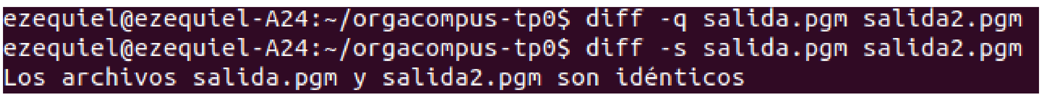
Con esta última ejecución redireccionamos la salida estandar a un archivo. Solo basta utilizar el comando diff utilizando el comando -s para obtener si son identicos. Figura 7



```
ezequiel@ezequiel-A24:~/orgacompus-tp0$ ./app.exe -o -
P2
640 480
255
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

(a) Salida por pantalla

Figura 6: Prueba de la salida por pantalla



```
ezequiel@ezequiel-A24:~/orgacompus-tp0$ diff -q salida.pgm salida2.pgm
ezequiel@ezequiel-A24:~/orgacompus-tp0$ diff -s salida.pgm salida2.pgm
Los archivos salida.pgm y salida2.pgm son idénticos
```

(a) Prueba de la igualdad de los archivos

Figura 7: Resultado de la prueba

5. Código fuente

En esta sección incluiremos el código fuente de nuestra aplicación .

```
1  #include "julia.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "complex.h"
5  #include "string.h"
6  #include "utils.h"
7
8  typedef unsigned char byte;
9
10 byte iterateCpx(Complex cpx, Complex seed) {
11   double mod = modCpx(&cpx);
12   int res = 0;
13   Complex itCpx = cpx;
14
15   if (mod >= 2.0) {
16     return res;
17   }
18
19   Complex powCpx;
20   Complex newCpx;
21
22   while (mod < 2 && res++ < 255) {
23     powCpx = pow2Cpx(&itCpx);
24     newCpx = addCpx(&powCpx, &seed);
25
26     mod = modCpx(&newCpx);
27
28     itCpx = newCpx;
29
30   }
31   return res >= 255 ? 255 : (byte)res;
32 }
33
34 unsigned getOptIndex(unsigned itemCount) {
35   unsigned idx = 16;
36   while (itemCount %idx--> 0)
37     ;
38   return idx + 1;
39 }
40
41 void runJulia(Arguments* args) {
42   Complex cpxDim = args->cpxSize;
43   Complex cpxCenter = args->center;
44   Boundaries bound = getBoundaries(&cpxDim, &cpxCenter);
```

```
45
46    Resolution resolution = args->resolution;
47
48    double stepX = ((double)cpxDim.re) / ((double)resolution.width
          );
49    double stepY = ((double)cpxDim.im) / ((double)resolution.
          height);
50
51    Complex seed = args->seed;
52
53    unsigned itemCount = resolution.width * resolution.height;
54
55    int coutMode = 0;  // false
56    char* outfilePath = args->outfile;
57    coutMode = strcmp("", outfilePath) == 0 || strcmp("cout",
          outfilePath) == 0;
58
59    FILE* outfile = coutMode ? stdout : fopen(outfilePath, "w");
60
61    if (outfile == NULL) {
62      printf("El archivo %s no existe!\n", outfilePath);
63      return;
64    }
65
66    fprintf(outfile, "P2\n");
67    fprintf(outfile, "%d %d\n", resolution.width, resolution.
          height);
68    fprintf(outfile, "%d", 255);
69
70    // indice optimo para introducir un ENTER en el archivo PGM
71    unsigned optIndex = getOptIndex(itemCount);
72
73    double imMap;
74    double reMap;
75    int y;
76    int x;
77    unsigned index = 0;
78    for (y = 0; y < resolution.height; y++) {
79      imMap = ((double)y) * stepY * (-1.0) + bound.top;
80
81      for (x = 0; x < resolution.width; x++) {
82        reMap = ((double)x) * stepX + bound.left;
83
84        Complex cpx = newCpx(reMap, imMap);
85
86        byte value = iterateCpx(cpx, seed);
87
88        if (index % optIndex == 0) fprintf(outfile, "\n");
89        fprintf(outfile, "%d ", value);
```

```
90
91          ++index;
92        }
93    }
94
95    fprintf(outfile, "\n");
96    fflush(outfile);
97    if (!coutMode) fclose(outfile);
98
99 }
```

Listing 1: julia.c

```
 1 #include <stdio.h>
 2
 3 #include "pixel.h"
 4 #include "complex.h"
 5 #include "args.h"
 6 #include "julia.h"
 7 #include "utils.h"
 8
 9 int main(int argc, char** argv) {
10     Arguments args = parseArgs(argc, argv);
11
12     runJulia(&args);
13
14     return 0;
15 }
```

Listing 2: app.c

```
 1 #include "args.h"
 2 #include <getopt.h>
 3 #include <stdio.h> /* for printf */
 4 #include <string.h>
 5 #include "pixel.h"
 6 #include "utils.h"
 7
 8 Arguments parseArgs(int argc, char** argv) {
 9   int c;
10
11   Resolution resolution = {640, 480};
12   Complex center = {0.0, 0.0};
13   double cpxw = 2.0;
14   double cpxh = 2.0;
15   Complex seed = {-0.726895347709114071439,
         0.188887129043845954792};
16   char* outfile = "cout";
17
18   while (1) {
```

```
19      int option_index = 0;
20
21      // SETEANDO opterr = 0 se logra que getopt no imprima sus
            propios mensajes
22      opterr = 0;
23      static struct option long_options[] = {
24          {"resolution", optional_argument, 0, 0},
25          {"center", optional_argument, 0, 0},
26          {"width", optional_argument, 0, 0},
27          {"height", optional_argument, 0, 0},
28          {"seed", optional_argument, 0, 0},
29          {"output", optional_argument, 0, 0},
30          {0, 0, 0, 0}};
31
32      c = getopt_long(argc, argv, "r:c:w:H:s:o:", long_options, &
            option_index);
33      if (c == -1) break;
34
35      switch (c) {
36        case 0:
37          switch (option_index) {
38            case 0:
39              parseRes(optarg, &resolution);
40              break;
41            case 1:
42              parseCpx(optarg, &center);
43              break;
44            case 2:
45              sscanf(optarg, "%lf", &cpxw);
46              break;
47            case 3:
48              sscanf(optarg, "%lf", &cpxh);
49              break;
50            case 4:
51              parseCpx(optarg, &seed);
52              break;
53            case 5:
54              outfile = optarg;
55              break;
56            default:
57              break;
58          }
59          break;
60
61        case 'r':
62          parseRes(optarg, &resolution);
63          break;
64
65        case 'w':
```

```
66          sscanf(optarg, "%lf", &cpxw);
67          break;
68
69        case 'H':
70          sscanf(optarg, "%lf", &cpxh);
71          break;
72
73        case 's':
74          parseCpx(optarg, &seed);
75          break;
76
77        case 'o':
78          outfile = optarg;
79          break;
80
81        case 'c':
82          parseCpx(optarg, &center);
83          break;
84
85        case '?':
86          printf("option ? with value '%s'\n", optarg);
87          break;
88
89        default:
90          printf("?? getopt returned character code 0%o ??\n", c);
91      }
92    }
93
94    if (optind < argc) {
95      printf("non-option ARGV-elements: ");
96      while (optind < argc) printf("%s ", argv[optind++]);
97      printf("\n");
98    }
99
100   int argi;
101   for (argi = 0; argi < argc;) {
102     if (strcmp("-o", argv[argi]) == 0) {
103       argi++;
104       if (argi == argc) break;
105       if (strcmp("-", argv[argi]) == 0) {
106         outfile = "cout";
107         break;
108       }
109     }
110     argi++;
111   }
112
113   Complex cpxSize = {cpxw, cpxh};
114   Arguments arguments = {resolution, center, cpxSize, seed,
```

```
        outfile};
115
116   return arguments;
117 }
```

Listing 3: args.c

```
 1 #include <stdio.h>
 2
 3 #include "pixel.h"
 4
 5 void parseRes(char* str , Resolution* targetRes) {
 6     unsigned w;
 7     unsigned h;
 8     int scanResult = sscanf(str , "%ux%u" , &w , &h);
 9     if(scanResult == 2) {
10         targetRes->height = h;
11         targetRes->width = w;
12     }
13 }
```

Listing 4: pixel.c