UNIVERSIDAD DE BUENOS AIRES
FIUBA

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

ALUMNOS

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

Índice

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

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

```
#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 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

```
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 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

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])

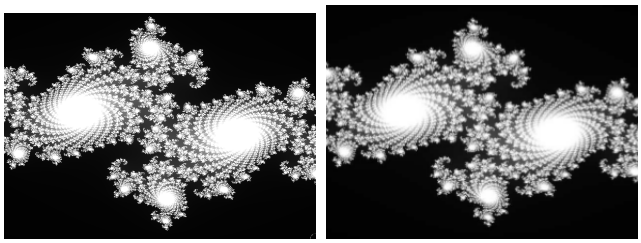
en el archivo pasado como argumento; o por salida estándar -cout- si el argumento es "-".

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

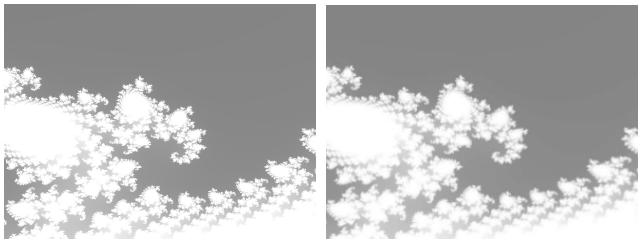
4.2. Zoom en región

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

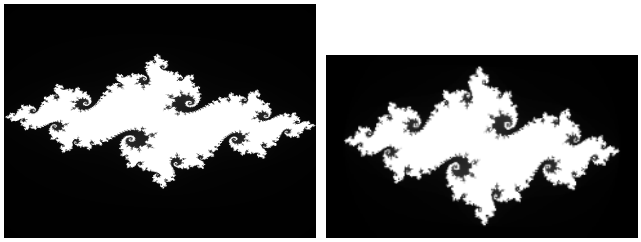
En esta prueba vamos a modificar la semilla utilizando el comando -s. Apoyandonos mediante otra implementación del Algoritmo de Julia, va-



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

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:



(a) nuestro programa (b) Wolframalpha

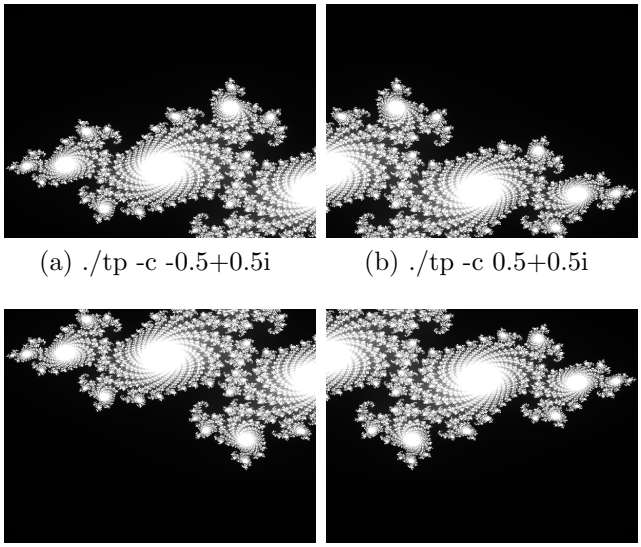
Figura 3: Comparación entre otra implementación del algoritmo de Julia y nuestro programa

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.

4.5. Modificación de la resolución

En esta prueba se modificará la resolución de la pantalla a 200*200 px. utilizando el comando -r

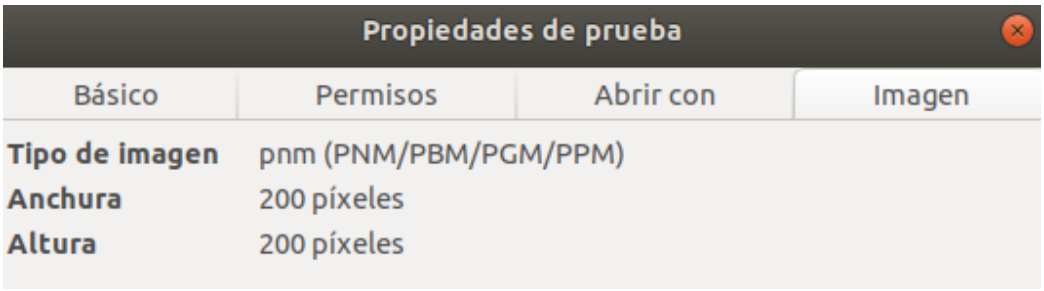


(a) ./tp -c -0.5+0.5i (b) ./tp -c 0.5+0.5i

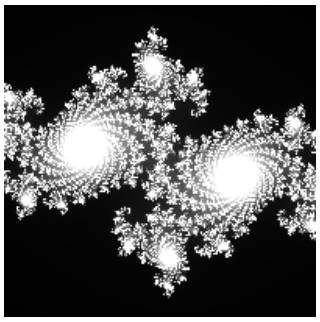
(c) ./tp -c -0.5-0.5i (d) ./tp -c 0.5-0.5i

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

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



(a) Resolución del archivo creado



(b) Imagen creada

Figura 5: Resultado de la prueba

5. Código fuente

En esta sección incluiremos el código fuente de nuestra aplicación .

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "julia.h"
4  #include "complex.h"
5  #include "utils.h"
6  #include "string.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         //printf("\tmod = %f\n" , mod);
17         return res;
18     }
19
20     Complex powCpx;
21     Complex newCpx;
22
23     //printf("mod for complex %f,%lf is %f \n" , cpx.re , cpx.
           im , mod);
24     while(mod < 2 && res++ < 255) {
25         powCpx = pow2Cpx(&itCpx);
26         //printf("\tpow2Cpx: ");
27         //printCpx(&powCpx);
28         //printf("\n");
29
30         newCpx = addCpx(&powCpx , &seed);
31         //printf("\taddCpx: ");
32         //printCpx(&newCpx);
33         //printf("\n");
34
35         mod = modCpx(&newCpx);
36         //printf("\tmod = %f ; " , mod);
37
38         itCpx = newCpx;
39         //printCpx(&itCpx , "newCpx: ");
40         //printf("\n");
41     }
42     return res >= 255 ? 255 : (byte) res;
43 }
```

```
44
45
46 unsigned getOptIndex(unsigned itemCount) {
47     unsigned idx = 16;
48     while(itemCount %idx--- > 0);
49     return idx + 1;
50 }
51
52 void runJulia(Arguments* args) {
53     Complex cpxDim = args->cpxSize;
54     Complex cpxCenter = args->center;
55     Boundaries bound = getBoundaries(&cpxDim , &cpxCenter);
56
57
58     Resolution resolution = args->resolution;
59
60     double stepX = ( (double) cpxDim.re) / ( (double) resolution
           .width);
61     double stepY = ( (double) cpxDim.im) / ( (double) resolution
           .height);
62
63
64     Complex seed = args->seed;
65
66     unsigned itemCount = resolution.width * resolution.height;
67
68
69     int coutMode = 0; //false
70     char* outfilePath = args->outfile;
71     coutMode = strcmp("" , outfilePath) == 0 || strcmp("cout" ,
           outfilePath) == 0;
72     if(coutMode) printf("MODO COUT ACTIVO\n");
73     FILE* outfile = coutMode ? stdout : fopen(outfilePath , "w")
           ;
74
75     if(outfile == NULL) {
76         printf("El archivo %s no existe!\n" , outfilePath);
77         return;
78     }
79
80     fprintf(outfile ,"P2\n");
81     fprintf(outfile , "%d %d\n" , resolution.width , resolution.
           height);
82     fprintf(outfile ,"%d" , 255);
83
84     // indice optimo para introducir un ENTER en el archivo PGM
85     unsigned optIndex = getOptIndex(itemCount);
86
87     double imMap;
```

```
88      double reMap;
89      int y;
90      int x;
91      unsigned index = 0;
92      for(y = 0 ; y < resolution.height ; y++) {
93          imMap = ((double)y) * stepY * (-1.0) + bound.top;
94
95          for(x = 0 ; x < resolution.width ; x++) {
96              reMap = ((double)x) * stepX + bound.left;
97
98              Complex cpx = newCpx(reMap , imMap);
99              //printCpx(&cpx , "this step: ");
100             //printf("\n");
101
102             byte value = iterateCpx(cpx , seed);
103
104             if(index % optIndex == 0) fprintf(outfile , "\n");
105             fprintf(outfile , "%d " , value);
106
107             ++index;
108         }
109     }
110
111     fflush(outfile);
112     if(!coutMode) fclose(outfile);
113
114     printf("stepX: %lf , stepY: %lf \n" , stepX , stepY);
115     printf("Boundaries: left:%lf,right:%lf,top:%lf,bottom:%lf \n
            " , bound.left , bound.right , bound.top , bound.bottom);
116 }
```

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     //Resolution res = { 1 , 1 };
13     //parseRes("10x10" , &res);
14     //
15     //Complex seed = { 0.0 , 0.0 };
16     //parseCpx("-0.7268953477091140714 39+0.188887129043845954792
```

```
        i" , &seed);
17
18 //     int Digs = 21;
19 //     double OneSeventh = 1.0/7.0;
20 //     printf("%.*f\n", Digs, OneSeventh);
21
22     /*
23     Resolution resolution;  // resolucion de imagen
24     Complex center;         // centro del plano complejo
25     Dimension cpxSize;      // tamano del plano complejo
26     Complex seed;           // semilla disparadora
27     char* outfile;          // archivo de salida
28     */
29
30     //Complex center = newCpx(0.0 , 0.0);
31     //Complex cpxSize = {2.0,2.0};
32     //char* outfile = "out.pgm";
33     //Arguments args = {res , center , cpxSize , seed , outfile
           };
34
35     runJulia(&args);
36
37     printCpx(&args.center , "Centro: ");
38     printf("\n");
39     printCpx(&args.seed , "Seed: ");
40     printf("\n");
41     printf("Resolution: %ux%u\n" , args.resolution.width , args.
           resolution.height);
42     printCpx(&args.cpxSize , "Tamano plano complejo: ");
43     printf("\n");
44     printf("Salida: %s\n" , args.outfile);
45
46     return 0;
47 }
```

Listing 2: app.c

```
1 #include <getopt.h>
2 #include <stdio.h> /* for printf */
3 #include "args.h"
4 #include "pixel.h"
5 #include "utils.h"
6
7 Arguments parseArgs(int argc, char** argv) {
8   int c;
9
10  Resolution resolution = { 640 , 480 };
11  Complex center = {0.0 , 0.0};
12  double cpxw = 2.0;
```

```
13  double cpxh = 2.0;
14  Complex seed = { -0.726895347709114071439 ,
        0.188887129043845954792 };
15  char* outfile;
16
17  while (1) {
18    int option_index = 0;
19
20    // SETEANDO opterr = 0 se logra que getopt no imprima sus
          propios mensajes
21    opterr = 0;
22    static struct option long_options[] = {{"resolution",
          optional_argument, 0, 0},
23                                           {"center",
                                               optional_argument,
                                                0, 0},
24                                           {"width",
                                               optional_argument,
                                                0, 0},
25                                           {"height",
                                               optional_argument,
                                                0, 0},
26                                           {"seed",
                                               optional_argument,
                                                0, 0},
27                                           {"output",
                                               optional_argument,
                                                0, 0},
28                                           {0, 0, 0, 0}};
29
30    c = getopt_long(argc, argv, "r:c:w:H:s:o:", long_options, &
          option_index);
31    if (c == -1) break;
32
33    switch (c) {
34      case 0:
35        switch(option_index) {
36          case 0:
37            parseRes(optarg , &resolution); break;
38          case 1:
39            parseCpx(optarg , &center); break;
40          case 2:
41            sscanf(optarg , "%lf" , &cpxw); break;
42          case 3:
43            sscanf(optarg , "%lf" , &cpxh); break;
44          case 4:
45            parseCpx(optarg , &seed); break;
46          case 5:
47            outfile = optarg; break;
```

```
48            default: break;
49          }
50          break;
51
52        case 'r':
53          parseRes(optarg , &resolution);
54          break;
55
56        case 'w':
57          sscanf(optarg , "%lf" , &cpxw);
58          break;
59
60        case 'H':
61          sscanf(optarg , "%lf" , &cpxh);
62          break;
63
64        case 's':
65          parseCpx(optarg , &seed);
66          break;
67
68        case 'o':
69          outfile = optarg;
70          break;
71
72        case 'c':
73          parseCpx(optarg , &center);
74          break;
75
76        case '?':
77          printf("option ? with value '%s'\n", optarg);
78          break;
79
80        default:
81          printf("?? getopt returned character code 0%o ??\n", c);
82      }
83    }
84
85    if (optind < argc) {
86      printf("non-option ARGV-elements: ");
87      while (optind < argc) printf("%s ", argv[optind++]);
88      printf("\n");
89    }
90
91    //  Resolution resolution;
92    //  Complex center;
93    //  Complex cpxSize;
94    //  Complex seed;
95    //  char* outfile;
96    Complex cpxSize = {cpxw , cpxh};
```

```
97    Arguments arguments = {resolution , center , cpxSize , seed ,
          outfile};
98
99    return arguments;
100 }
```

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