Trabajo Práctico 0: Infraestructura básica

Santiago Aguilera, *Padrón Nro. 95795*
marquito.santi@gmail.com
Agustina Barbetta, *Padrón Nro. 96528*
agustina.barbetta@gmail.com
Manuel Porto, *Padrón Nro. 96587*
manu.porto94@hotmail.com

2do. Cuatrimestre de 2016
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

13 de Septiembre del 2016

Resumen

El presente trabajo práctico consiste en crear una aplicación, en ISO C, totalmente personalizable capaz de dibujar los conjuntos fractales de Julia sobre una imagen de formato PGM.

El código del trabajo se encuentra en el siguiente repositorio:
https://github.com/saantiaguilera/fiuba-orga-pc-julia-set.

Índice

1. Introducción

El principal objetivo de este trabajo es familiarizarse con las herramientas de software que usaremos en los siguientes prácticos[1][2], implementando un programa y su correspondiente documentación que resuelvan el problema descripto más adelante.

2. Conjunto de Julia

El programa a implementar debe generar distintas imágenes pertenecientes a los conocidos 'conjuntos de Julia'.

> Los conjuntos de Julia, así llamados por el matemático Gaston Julia, son una familia de conjuntos fractales que se obtienen al estudiar el comportamiento de los números complejos al ser iterados por una función holomorfa.
>
> El conjunto de Julia de una función holomorfa f, está constituido por aquellos puntos que bajo la iteración de f, tienen un comportamiento 'caótico'. El conjunto se denota $J(f)$.
>
> En el otro extremo se encuentra el conjunto de Fatou (en honor del matemático Pierre Fatou), que consiste de los puntos que tienen un comportamiento 'estable' al ser iterados. El conjunto de Fatou de una función holomorfa f, se denota $F(f)$ y es el complemento de $J(f)$.[3]

3. Programa

El programa a escribir, en lenguaje C, recibirá al momento de ejecutarse distintos argumentos para modificar el archivo generado o las condiciones del conjunto de Julia.
Los mismos son:

- -o: Obligatorio. Archivo de salida para la imagen. Debe terminar con .pgm. Si el argumento es '-' se utilizara la salida estándar (stdout).
- -r: Opcional. Resolución de la imagen de salida. Por defecto sera 640x480 px.
- -c: Opcional. Centro del set de la partición del plano complejo. Por default sera 0+0i.
- -C: Opcional. Parámetro C de la ecuación de iteración (correspondiente a la familia cuadrática). Por defecto se usara 0.285+0.01i
- -w: Opcional. Ancho del rectángulo del set en el plano real. Por defecto sera 4.
- -H: Opcional. Alto del rectángulo del set en el plano complejo. Por defecto sera 4.
- -h: Opcional. Help. Brinda información detallada del uso de cada parámetro y ejemplos.
- -v: Opcional. Versión del programa.

3.1. Modo de uso

Si no se tiene el ejecutable, para generarlo se debe correr (dentro de donde están los archivos fuente, en caso de clonar el repositorio del trabajo, `cd src/`) `make`.

Una vez que se tenga el archivo ejecutable. Una ejecución básica, usando los parámetros por defecto será:

`./tp0 -o filename.pgm`

Si se desea personalizar los parámetros del conjunto de Julia o de la imagen, se pueden obtener mediante:

`./tp0 -h`

Allí se encuentra toda la documentación detallada para personalizar el programa.

3.2. Diseño

A continuación se describe cada uno de los módulos implementados para la realización del programa.

3.2.1. Complex

En el modulo complex.c se implemento la estructura del numero complejo junto con algunas de sus operaciones básicas ya que la misma no esta soportada de manera nativa en el lenguaje C.

3.2.2. Decoder

En el modulo decoder.c se define la lógica para decodificar una imagen en base a los parámetros recibidos y escribirla sobre un archivo de salida. El mismo creará una imagen de formato PGM (Formato "P2") de dimensiones 640x480 píxeles (el tamaño puede modificarse pasando por parámetro `-r widthxheight`). A su vez, se encontrará centrada en el 0+0i y el recinto sobre el que se iterará para obtener el conjunto de Julia es de 4 tanto de largo como de ancho (Para usar valores distintos para el centro o las dimensiones, se puede ejecutar agregando `-c x+yi -w width -H height`)

El conjunto de Julia se calcula iterando sobre una familia de polinomios cuadrados dados de la forma

$$z_{n+1} = z_n^2 + c$$

$$c = 0{,}285 + 0{,}01i$$

Si se desea cambiar el valor de c. Por parámetro se deberá ingresar `-C x+yi`. La condición de corte utilizada por el decodificador será, que el brillo no sea mayor a 255 (Valor máximo utilizado por la imagen PGM para el brillo) y que $abs(z) <= 2$.

3.2.3. Main

En el módulo main.c se define la función que da inicio al programa, inmediatamente después, por medio de `getopt_long` se cicla entre todos los *flags* recibidos por parámetro, definiendo el flujo correspondiente del programa. El beneficio de esta función es que permite al usuario escribir las opciones en cualquier orden. Cabe aclarar que, si se utilizan los *flags* de ayuda o versión, el programa devolverá estos datos sin importar que se hayan ingresado los parámetros para una codificación o decodificación.

Además, el módulo contiene funciones para mostrar el mensaje de ayuda (`show_help()`), la versión del programa (`show_version()`) y la función `write_image(...)` encargada de escribir los resultados.

3.3. Compilación

Se provee de un *Makefile* para la compilación del proyecto, el mismo cuenta con un *phony target* para limpiar los archivos generados (`make clean`) y el target *phony all* que (por más que sea *phony*) generará un ejecutable '`tp0`' y los archivos del linker necesarios (`make` o `make all`).

En caso de compilar el proyecto en BSD, se deben realizar los mismos pasos pero reemplazando el comando `make` por `gmake` ya que el `make` de BSD no es el mismo al que se encuentra en los sistemas Linux, que es GNU make. Entonces, para utilizar este ultimo en BSD se debe llamar a `gmake`.

3.4. Pruebas

Se creó una *test-suite* en bash para testear casos borde de los parámetros y que el programa funcione correctamente.

Para correrla simplemente hay que compilar el proyecto previamente, darle permisos al *script* (de ser necesarios, `chmod u+x run_tests.sh`) y correrlo de forma normal (`bash run_tests.sh`). El mismo irá dando información sobre cada test corrido y su estado.

Se borró, de la *test suite* previa, un test que validaba que no se pudiese escribir un archivo protegido (y se retornara información al respecto de la falla), debido a que en BSD el root no es afectado por `chmod`. Estuvimos buscando si podíamos evitar que el usuario lo pueda editar, y es posible pero los comandos son exclusivos de cada OS asi que preferimos ignorar el test.

A continuación se incluye una imagen del caso default pedido por el trabajo. El mismo fue generado mediante `./tp0 -o img.pgm`.

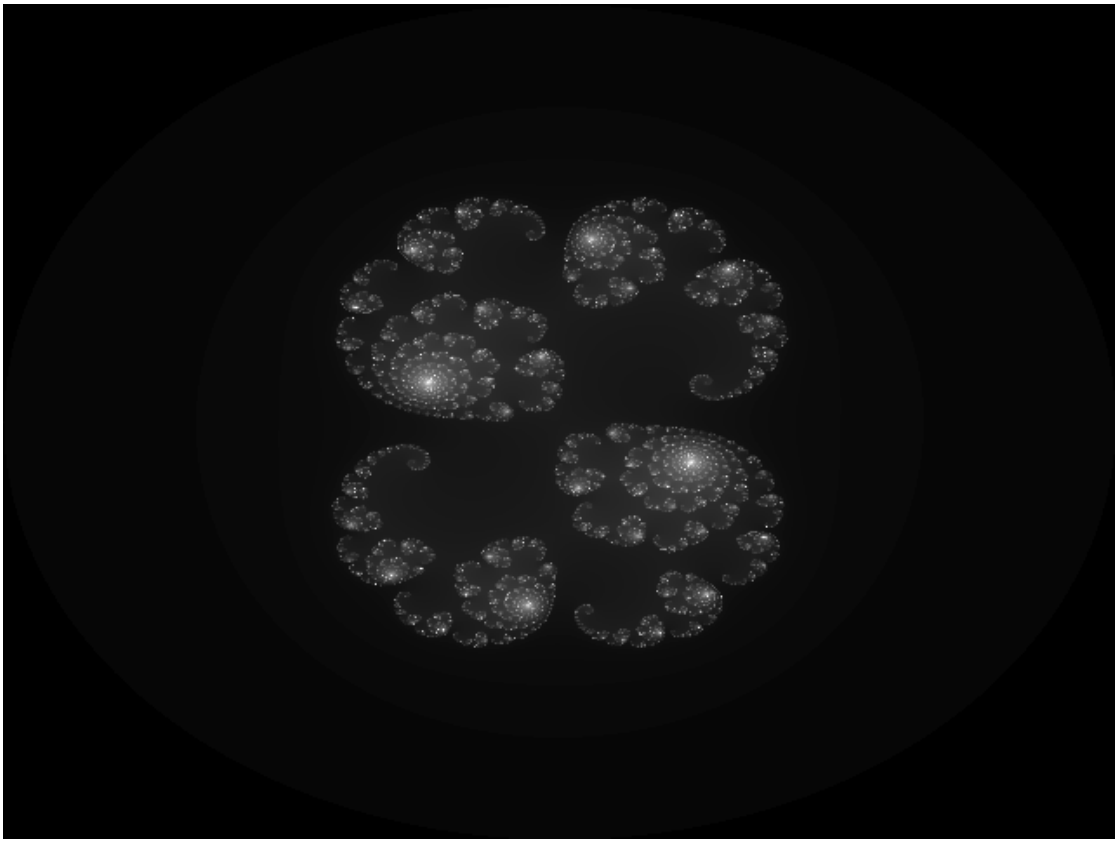

Figura 1: Resultado por default: img.pgm

4. Apéndices

4.1. Código fuente

El código a continuación se encuentra también en el repositorio del trabajo: `https://github.com/saantiaguilera/fiuba-orga-pc-julia-set`.

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

#include "complex.h"
#include "decoder.h"

#define VERSION "2.0.0"
#define BUFFER_LENGTH 1024

#define ERROR_INVALID_RESOLUTION 2
#define ERROR_INVALID_CENTER 3
```

```
#define ERROR_INVALID_C 4
#define ERROR_INVALID_WIDTH 5
#define ERROR_INVALID_HEIGHT 6
#define ERROR_NO_OUTPUT 7
#define ERROR_ILLEGAL_OUTPUT 8

void show_version() {
    printf("v%s\n", VERSION);
}

void show_help() {
    FILE *fp = fopen("julia-set.help", "r");

    char buffer[BUFFER_LENGTH];
    int buflen;
    while ((buflen = fread(buffer, sizeof(char), BUFFER_LENGTH, fp)) > 0)
        puts(buffer);

    fclose(fp);
}

int load_new_resolution(int* resolution_height, int* resolution_width, char optarg
    []) {
    char* end;
    *resolution_width = strtol(optarg, &end, 10);
    *resolution_height = strtol(&end[1], NULL, 10);
    if (!(*resolution_width) || !(*resolution_height))
        return 1;
    return 0;
}

int write_image(char output_file[], int resolution_height,
        int resolution_width, _complex *center, _complex *C,
        double complex_plane_height, double complex_plane_width) {

    int ret_value = EXIT_SUCCESS;
    _decoder decoder;
    decoder_init(&decoder, resolution_width, resolution_height,
            complex_plane_width, complex_plane_height, center, C);
    FILE *fp;

    if (strcmp("-", output_file) == 0) {
        fp = stdout;
    } else {
        fp = fopen(output_file, "wb");
        if (fp == NULL) {
            fprintf(stderr, "fatal: cannot open output file.\n");
            return ERROR_ILLEGAL_OUTPUT;
        }
    }

    if (decoder_decode(&decoder, fp) != 0) {
        fprintf(stderr, "fatal: could not generate julia set.\n");
        ret_value = ERROR_PROCESSING_SET;
    }

    if (strcmp("-", output_file) != 0) fclose(fp);

    return ret_value;
}

int main (int argc, char *argv[]) {
```

```
    bool help, version, resolution, new_center, new_C, width, height, output;
    help = version = resolution = new_center = new_C = width = height = output =
        false;
    int resolution_height = DEFAULT_IMAGE_HEIGHT;
    int resolution_width = DEFAULT_IMAGE_WIDTH;
    _complex center;
    complex_init(&center, DEFAULT_RENDER_CENTER_X, DEFAULT_RENDER_CENTER_Y);
    _complex C;
    complex_init(&C, DEFAULT_RATIO_X, DEFAULT_RATIO_Y);
    double complex_plane_height = DEFAULT_RENDER_HEIGHT;
    double complex_plane_width = DEFAULT_RENDER_WIDTH;
    char* output_file = NULL;

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

    while ((flag = getopt_long(argc, argv, "Vhr:c:C:w:H:o:", opts, NULL)) != -1) {
        switch (flag) {
            case 'V' :
                version = true;
                break;
            case 'h' :
                help = true;
                break;
            case 'r' :
                resolution = true;
                if (load_new_resolution(&resolution_height, &resolution_width,
                    optarg) != 0) {
                    fprintf(stderr, "fatal: invalid resolution specification.\n");
                    return ERROR_INVALID_RESOLUTION;
                }
                break;
            case 'c' :
                new_center = true;
                if (strtoc(&center, optarg) != 0) {
                    fprintf(stderr, "fatal: invalid center specification.\n");
                    return ERROR_INVALID_CENTER;
                }
                break;
            case 'C' :
                new_C = true;
                if (strtoc(&C, optarg) != 0) {
                    fprintf(stderr, "fatal: invalid C specification.\n");
                    return ERROR_INVALID_C;
                }
                break;
            case 'w' :
                width = true;
                complex_plane_width = atof(optarg);
                if (complex_plane_width == 0) {
                    fprintf(stderr, "fatal: invalid complex plane width
                        specification.\n");
```

```
                    return ERROR_INVALID_WIDTH;
                }
                break;
            case 'H' :
                height = true;
                complex_plane_height = atof(optarg);
                if (complex_plane_height == 0) {
                    fprintf(stderr, "fatal: invalid complex plane height
                        specification.\n");
                    return ERROR_INVALID_HEIGHT;
                }
                break;
            case 'o' :
                output = true;
                output_file = optarg;
                break;
        }
    }

    if (version) show_version();
    else if (help) show_help();
    else {
        if (!output) {
            fprintf(stderr, "fatal: No output specified.\n");
            return ERROR_NO_OUTPUT;
        }

        fprintf(stderr, "JULIA SET\n resolution_height = %d\n resolution_width = %
            d\n"
                " re_center = %f\n im_center = %f\n re_C = %f\n im_C = %f\n"
                " complex_plane_height = %f\n complex_plane_width = %f\n"
                " output_file = %s\n", resolution_height, resolution_width,
                center.real, center.img, C.real, C.img,
                complex_plane_height, complex_plane_width, output_file);

        return write_image(output_file, resolution_height, resolution_width, &
            center,
                &C, complex_plane_height, complex_plane_width);
    }

    return EXIT_SUCCESS;
}
```

Listing 1: main.c

```
#ifndef COMPLEX_H_
#define COMPLEX_H_

typedef struct _complex {
    double real;
    double img;
} _complex;

int complex_init(_complex *self, double real, double img);
int complex_add(_complex *self, const _complex *other);
int complex_mult(_complex *self, const _complex *other);
int strtoc(_complex *self, char* str);
double complex_abs(const _complex *self);

double complex_getX(const _complex *self);
double complex_getY(const _complex *self);
```

```
#endif
```

Listing 2: complex.h

```
#include <stdlib.h>

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

int complex_init(_complex *self, double real, double img) {
    self->real = real;
    self->img = img;
    return 0;
}

int complex_add(_complex *self, const _complex *other) {
    self->real += other->real;
    self->img += other->img;
    return 0;
}

int complex_mult(_complex *self, const _complex *other) {
    self->real = self->real * other->real -
        self->img * other->img;
    self->img = self->real * other->img +
        self->img * other->real;
    return 0;
}

int strtoc(_complex *self, char* str) {
    char* end;
    self->real = strtod(str, &end);
    self->img = strtod(end, &end);
    if (*end != 'i') return 1;
    return 0;
}

double complex_getX(const _complex *self) {
    return self->real;
}

double complex_getY(const _complex *self) {
    return self->img;
}

double complex_abs(const _complex *self) {
    double real = self->real * self->real;
    double img = self->img * self->img;

    return sqrt(real + img);
}
```

Listing 3: complex.c

```
#ifndef DECODER_H_
#define DECODER_H_

#include <stdio.h>
#include "complex.h"
```

```c
#define DEFAULT_IMAGE_WIDTH 640
#define DEFAULT_IMAGE_HEIGHT 480
#define DEFAULT_RENDER_WIDTH 4.0
#define DEFAULT_RENDER_HEIGHT 4.0
#define DEFAULT_RENDER_CENTER_X 0.0
#define DEFAULT_RENDER_CENTER_Y 0.0

/*
Be careful the fractal in the pdf is not generated with the constant given.
This are the constants for generating the pdf fractal. Below are the
pdf constants for easy access if you want to change them
*/
#define DEFAULT_RATIO_X 0.285
#define DEFAULT_RATIO_Y 0.01

/*
PDF Constants
#define DEFAULT_RATIO_X 0.285
#define DEFAULT_RATIO_Y -0.01
*/

typedef struct _decoder {
    int imageWidth, imageHeight;
    double renderWidth, renderHeight;
    _complex *renderCenter;
    _complex *ratio;
} _decoder;

int decoder_init(_decoder *self,
        int imgWidth, int imgHeight,
        double rndWidth, double rndHeight,
        _complex *rndCenter, _complex *ratio);
int decoder_decode(_decoder *self, FILE *output);

#endif
```

Listing 4: decoder.h

```c
#include "decoder.h"

#define WHITE 255
#define BLACK 0
#define MAX_ABS_OFFSET 2

#define MAX_LENGTH_SIZE 66

int decoder_init(_decoder *self, int iw, int ih, double rw, double rh,
        _complex *c, _complex *r) {
    if (self == 0 || c == 0 || r == 0)  //Cant be nulls
        return -1;

    if (iw == 0 || ih == 0)
        return -1; //Image cant be 0x0 px

    self->imageWidth = iw;
    self->imageHeight = ih;
    self->renderWidth = rw;
    self->renderHeight = rh;
    self->renderCenter = c;
    self->ratio = r;

    return 0;
```

```
}

int decoder_decode(_decoder *self, FILE *output) {
    double startX, startY, endX, endY, stepX, stepY;

    double halfWidth = self->renderWidth / 2;
    double halfHeight = self->renderHeight /2;
    startX = complex_getX(self->renderCenter) - (halfWidth);
    startY = complex_getY(self->renderCenter) - (halfHeight);
    endX = complex_getX(self->renderCenter) + (halfWidth);
    endY = complex_getY(self->renderCenter) + (halfHeight);

    stepX = self->renderWidth / self->imageWidth;
    stepY = self->renderHeight / self->imageHeight;

    fprintf(output, "P2 ");
    fprintf(output, "%d %d ", self->imageWidth, self->imageHeight);
    fprintf(output, "%d\n", WHITE);

    int lengthCounter = 0;
    for (double indexY = startY ; indexY < endY ; indexY += stepY) {
        for (double indexX = startX ; indexX < endX ; indexX += stepX) {
            //Here im at 1 px of the image.
            _complex point;
            complex_init(&point, indexX, indexY);

            //Using as N = Black in that image format
            int color;
            for (color = BLACK ;
                color < WHITE - 1 && complex_abs(&point) <= MAX_ABS_OFFSET ;
                    ++color) {
                double newX = (complex_getX(&point) * complex_getX(&point)
                        - complex_getY(&point) * complex_getY(&point)
                        + complex_getX(self->ratio));
                double newY = (2 * complex_getX(&point) * complex_getY(&point)
                        + complex_getY(self->ratio));

                complex_init(&point, newX, newY);
            }

            int lengthPrinted = fprintf(output, "%d ", color);

            if (lengthPrinted > 0)
                lengthCounter += lengthPrinted;
            else return ERROR_PROCESSING_SET;

            if (lengthCounter > MAX_LENGTH_SIZE) {
                fprintf(output, "\n");
                lengthCounter = 0;
            }
        }
    }

    return 0;
}
```

Listing 5: decoder.c

4.2. Código MIPS assembly

A continuación adjuntamos el código assembly del módulo `decoder.c`

```
    .file   1 "decoder.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .text
    .align  2
    .globl  decoder_init
    .ent    decoder_init
decoder_init:
    .frame  $fp,24,$31      # vars= 8, regs= 2/0, args= 0, extra= 8
    .mask   0x50000000,-4
    .fmask  0x00000000,0
    .set    noreorder
    .cpload $25
    .set    reorder
    subu    $sp,$sp,24
    .cprestore 0
    sw  $fp,20($sp)
    sw  $28,16($sp)
    move    $fp,$sp
    sw  $4,24($fp)
    sw  $5,28($fp)
    sw  $6,32($fp)
    lw  $2,24($fp)
    beq $2,$0,$L7
    lw  $2,56($fp)
    beq $2,$0,$L7
    lw  $2,60($fp)
    bne $2,$0,$L6
$L7:
    li  $2,-1           # 0xffffffffffffffff
    sw  $2,8($fp)
    b   $L5
$L6:
    lw  $2,28($fp)
    beq $2,$0,$L9
    lw  $2,32($fp)
    bne $2,$0,$L8
$L9:
    li  $2,-1           # 0xffffffffffffffff
    sw  $2,8($fp)
    b   $L5
$L8:
    lw  $3,24($fp)
    lw  $2,28($fp)
    sw  $2,0($3)
    lw  $3,24($fp)
    lw  $2,32($fp)
    sw  $2,4($3)
    lw  $2,24($fp)
    l.d $f0,40($fp)
    s.d $f0,8($2)
    lw  $2,24($fp)
    l.d $f0,48($fp)
    s.d $f0,16($2)
    lw  $3,24($fp)
    lw  $2,56($fp)
    sw  $2,24($3)
    lw  $3,24($fp)
    lw  $2,60($fp)
    sw  $2,28($3)
```

```
    sw  $0,8($fp)
$L5:
    lw  $2,8($fp)
    move    $sp,$fp
    lw  $fp,20($sp)
    addu    $sp,$sp,24
    j   $31
    .end    decoder_init
    .size   decoder_init, .-decoder_init
    .rdata
    .align  2
$LC1:
    .ascii  "P2 \000"
    .align  2
$LC2:
    .ascii  "%d %d \000"
    .align  2
$LC3:
    .ascii  "%d\n\000"
    .align  2
$LC4:
    .ascii  "%d \000"
    .align  2
$LC5:
    .ascii  "\n\000"
    .align  3
$LC0:
    .word   0
    .word   1073741824
    .text
    .align  2
    .globl  decoder_decode
    .ent    decoder_decode
decoder_decode:
    .frame  $fp,200,$31     # vars= 136, regs= 3/2, args= 24, extra= 8
    .mask   0xd0000000,-24
    .fmask  0x00f00000,-8
    .set    noreorder
    .cpload $25
    .set    reorder
    subu    $sp,$sp,200
    .cprestore 24
    sw  $31,176($sp)
    sw  $fp,172($sp)
    sw  $28,168($sp)
    s.d $f22,192($sp)
    s.d $f20,184($sp)
    move    $fp,$sp
    sw  $4,200($fp)
    sw  $5,204($fp)
    lw  $2,200($fp)
    l.d $f2,8($2)
    l.d $f0,$LC0
    div.d   $f0,$f2,$f0
    s.d $f0,80($fp)
    lw  $2,200($fp)
    l.d $f2,16($2)
    l.d $f0,$LC0
    div.d   $f0,$f2,$f0
    s.d $f0,88($fp)
    lw  $2,200($fp)
    lw  $4,24($2)
```

```
    la  $25,complex_getX
    jal $31,$25
    mov.d   $f2,$f0
    l.d $f0,80($fp)
    sub.d   $f0,$f2,$f0
    s.d $f0,32($fp)
    lw  $2,200($fp)
    lw  $4,24($2)
    la  $25,complex_getY
    jal $31,$25
    mov.d   $f2,$f0
    l.d $f0,88($fp)
    sub.d   $f0,$f2,$f0
    s.d $f0,40($fp)
    lw  $2,200($fp)
    lw  $4,24($2)
    la  $25,complex_getX
    jal $31,$25
    mov.d   $f2,$f0
    l.d $f0,80($fp)
    add.d   $f0,$f2,$f0
    s.d $f0,48($fp)
    lw  $2,200($fp)
    lw  $4,24($2)
    la  $25,complex_getY
    jal $31,$25
    mov.d   $f2,$f0
    l.d $f0,88($fp)
    add.d   $f0,$f2,$f0
    s.d $f0,56($fp)
    lw  $3,200($fp)
    lw  $2,200($fp)
    l.s $f0,0($2)
    cvt.d.w $f2,$f0
    l.d $f0,8($3)
    div.d   $f0,$f0,$f2
    s.d $f0,64($fp)
    lw  $3,200($fp)
    lw  $2,200($fp)
    l.s $f0,4($2)
    cvt.d.w $f2,$f0
    l.d $f0,16($3)
    div.d   $f0,$f0,$f2
    s.d $f0,72($fp)
    lw  $4,204($fp)
    la  $5,$LC1
    la  $25,fprintf
    jal $31,$25
    lw  $2,200($fp)
    lw  $3,200($fp)
    lw  $4,204($fp)
    la  $5,$LC2
    lw  $6,0($2)
    lw  $7,4($3)
    la  $25,fprintf
    jal $31,$25
    lw  $4,204($fp)
    la  $5,$LC3
    li  $6,255          # 0xff
    la  $25,fprintf
    jal $31,$25
    sw  $0,96($fp)
```

```
    l.d $f0,40($fp)
    s.d $f0,104($fp)
$L11:
    l.d $f2,104($fp)
    l.d $f0,56($fp)
    c.lt.d  $f2,$f0
    bc1t    $L14
    b   $L12
$L14:
    l.d $f0,32($fp)
    s.d $f0,112($fp)
$L15:
    l.d $f2,112($fp)
    l.d $f0,48($fp)
    c.lt.d  $f2,$f0
    bc1t    $L18
    b   $L13
$L18:
    addu    $2,$fp,120
    l.d $f0,104($fp)
    s.d $f0,16($sp)
    move    $4,$2
    lw  $6,112($fp)
    lw  $7,116($fp)
    la  $25,complex_init
    jal $31,$25
    sw  $0,136($fp)
$L19:
    lw  $2,136($fp)
    slt $2,$2,254
    beq $2,$0,$L20
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_abs
    jal $31,$25
    mov.d   $f2,$f0
    l.d $f0,$LC0
    c.le.d  $f2,$f0
    bc1t    $L22
    b   $L20
$L22:
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getX
    jal $31,$25
    mov.d   $f20,$f0
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getX
    jal $31,$25
    mul.d   $f22,$f20,$f0
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getY
    jal $31,$25
    mov.d   $f20,$f0
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getY
    jal $31,$25
    mul.d   $f0,$f20,$f0
    sub.d   $f20,$f22,$f0
```

```
    lw  $2,200($fp)
    lw  $4,28($2)
    la  $25,complex_getX
    jal $31,$25
    add.d   $f0,$f20,$f0
    s.d $f0,144($fp)
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getX
    jal $31,$25
    add.d   $f20,$f0,$f0
    addu    $2,$fp,120
    move    $4,$2
    la  $25,complex_getY
    jal $31,$25
    mul.d   $f20,$f20,$f0
    lw  $2,200($fp)
    lw  $4,28($2)
    la  $25,complex_getY
    jal $31,$25
    add.d   $f0,$f20,$f0
    s.d $f0,152($fp)
    addu    $2,$fp,120
    l.d $f0,152($fp)
    s.d $f0,16($sp)
    move    $4,$2
    lw  $6,144($fp)
    lw  $7,148($fp)
    la  $25,complex_init
    jal $31,$25
    lw  $2,136($fp)
    addu    $2,$2,1
    sw  $2,136($fp)
    b   $L19
$L20:
    lw  $4,204($fp)
    la  $5,$LC4
    lw  $6,136($fp)
    la  $25,fprintf
    jal $31,$25
    sw  $2,160($fp)
    lw  $2,160($fp)
    blez    $2,$L24
    lw  $2,96($fp)
    lw  $3,160($fp)
    addu    $2,$2,$3
    sw  $2,96($fp)
    b   $L25
$L24:
    li  $2,9            # 0x9
    sw  $2,164($fp)
    b   $L10
$L25:
    lw  $2,96($fp)
    slt $2,$2,67
    bne $2,$0,$L17
    lw  $4,204($fp)
    la  $5,$LC5
    la  $25,fprintf
    jal $31,$25
    sw  $0,96($fp)
$L17:
```

```
    l.d $f0,112($fp)
    l.d $f2,64($fp)
    add.d   $f0,$f0,$f2
    s.d $f0,112($fp)
    b   $L15
$L13:
    l.d $f0,104($fp)
    l.d $f2,72($fp)
    add.d   $f0,$f0,$f2
    s.d $f0,104($fp)
    b   $L11
$L12:
    sw  $0,164($fp)
$L10:
    lw  $2,164($fp)
    move    $sp,$fp
    lw  $31,176($sp)
    lw  $fp,172($sp)
    l.d $f22,192($sp)
    l.d $f20,184($sp)
    addu    $sp,$sp,200
    j   $31
    .end    decoder_decode
    .size   decoder_decode, .-decoder_decode
    .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

Listing 6: decoder.s

4.3. Enunciado

Univesidad de Buenos Aires - FIUBA
66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica
2^o cuatrimestre de 2016

`$Date: 2016/08/30 04:13:03 $`

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 diseñar un programa que permita dibujar el conjunto de Julia [3] y sus vecindades, en lenguaje C, correspondiente a un polinomio cuadrático.

El mismo recibirá, por línea de comando, una serie de parámetros describiendo la región del plano complejo, las características del archivo imagen a generar, y el parámetro c.

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* [4], un formato simple para describir imágenes digitales monocromáticas.

5.1. Algoritmo

El algoritmo básico es simple: para algunos puntos z 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, $z = z_{re} + z_{im}i$. A continuación, iteramos sobre $z_{n+1} = z_n^2 + c$, con $z_0 = z$. Cortamos la iteración cuando $|z_n| > 2$, o después de `N` iteraciones.

En pseudo código:

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

Notar que c es un parámetro del programa.

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`: permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.
- `-c`: 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 $0 + 0i$.
- `-C`: determina el parámetro c, también expresado en forma binómica. El valor por defecto será $0{,}285 - 0{,}01i$.
- `-w`: especifica el ancho del rectángulo que contiene la región del plano complejo que estamos por dibujar. Valor por defecto: 4.

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

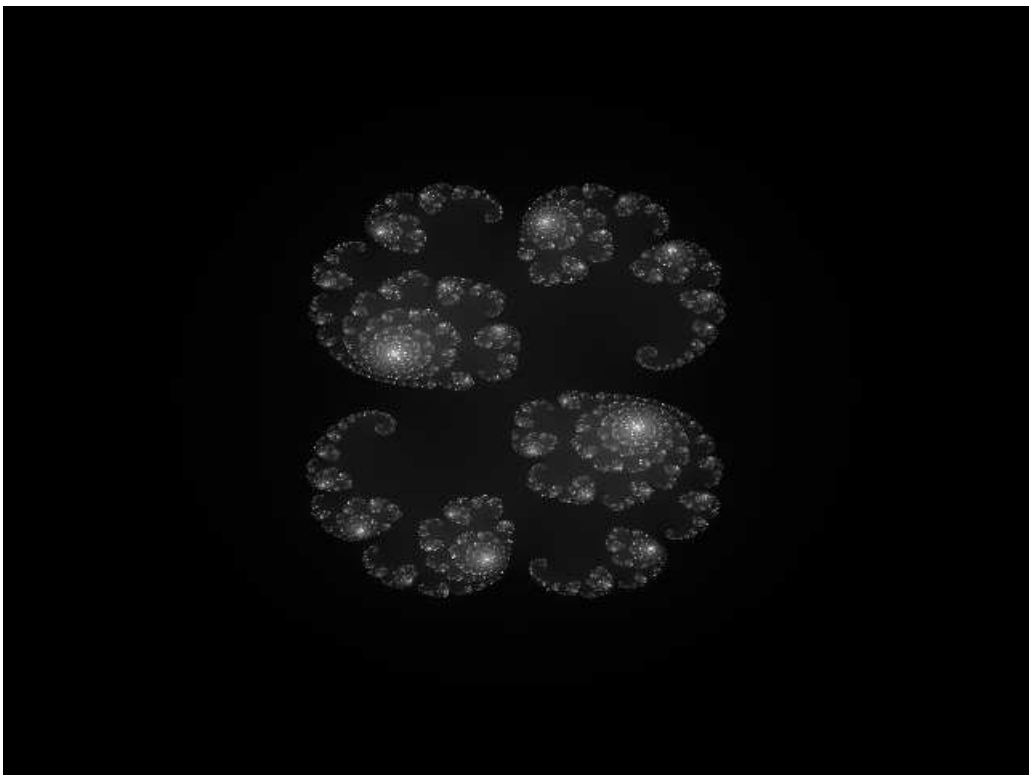

Figura 1: Región barrida por defecto.

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.

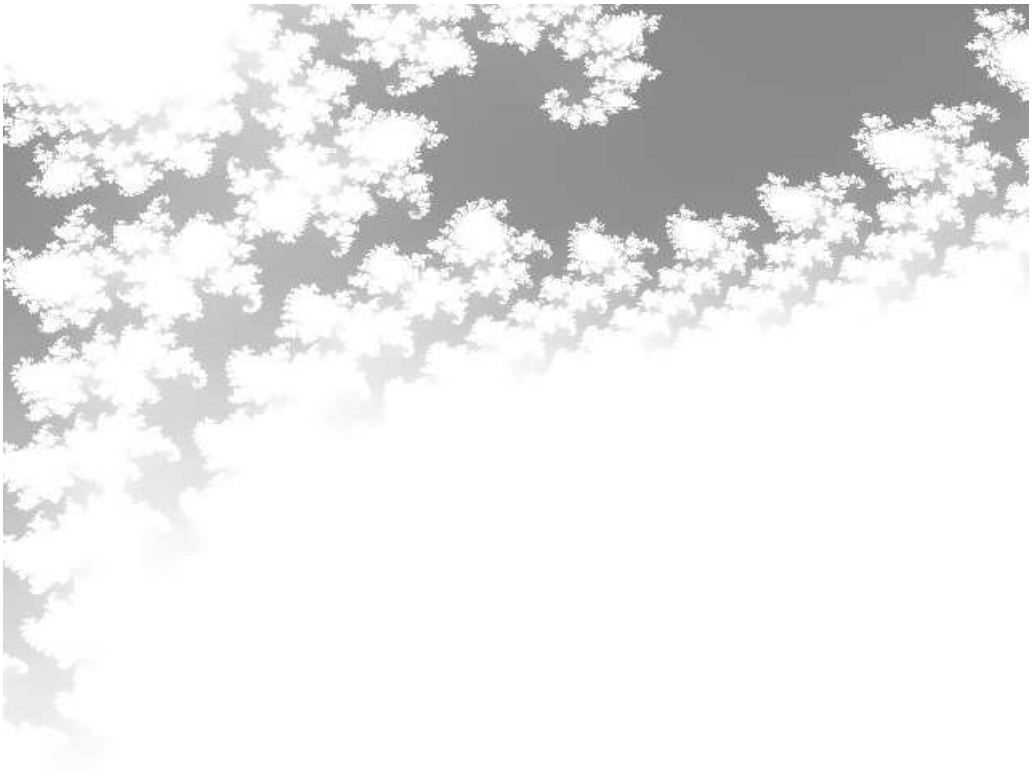

Figura 2: Región comprendida entre $0{,}2795 - 0{,}0075i$ y $0{,}2845 - 0{,}0125i$.

```
$ 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.
- 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. Fechas

Fecha de vencimiento: Martes 26/9.

Referencias

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

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

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

[4] 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 0.01+0i -r 1x1 -o -
P2
1
1
255
255
```

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

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 no pertenece al conjunto de Julia.

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.
```

Referencias

[1] *GXemul*
http://gavare.se/gxemul/

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

[3] *Conjunto de Julia*
https://es.wikipedia.org/wiki/Conjunto_de_Julia

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