TP 0: Set de Julia

Martín Queija, *Padrón Nro. 96.455*
tinqueija@gmail.com

Estanislao Ledesma, *Padrón Nro. 96.622*
estanislaoledesma@gmail.com

Agustin Luques, *Padrón Nro. 96.803*
agus.luques@hotmail.com

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

1 de noviembre de 2016

Resumen

Graficador de Conjuntos de Julia en formáto PGM

Índice

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

`$Date: 2016/09/20 17:31:41 $`

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`.

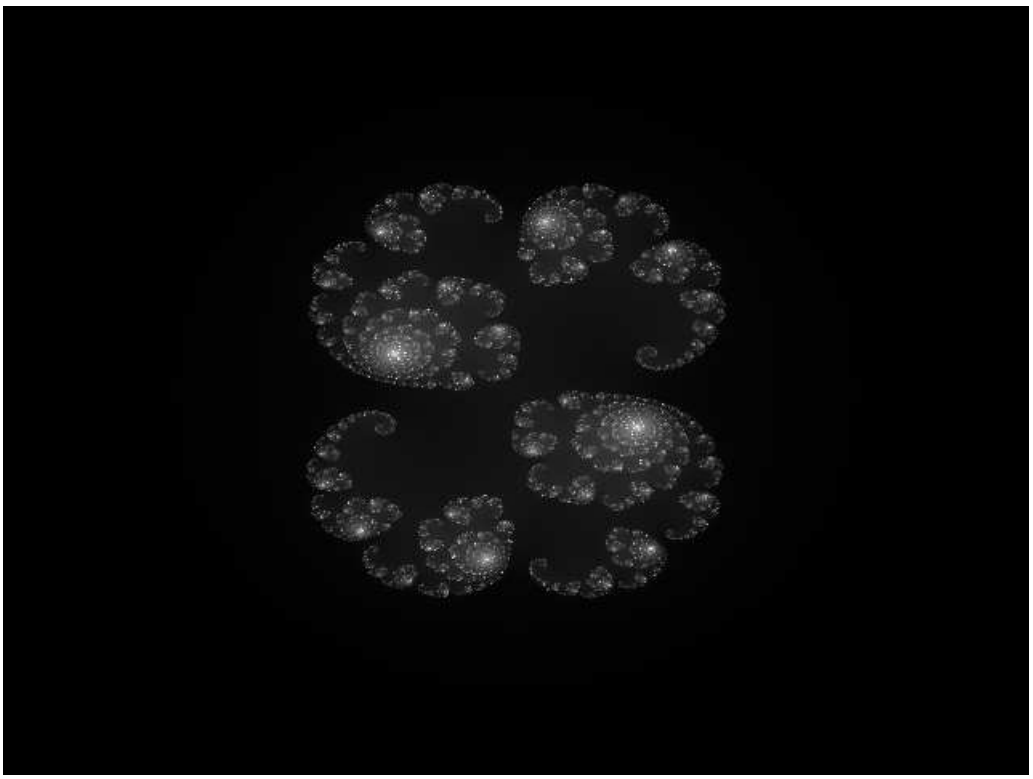

Figura 1: Región barrida por defecto.

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.

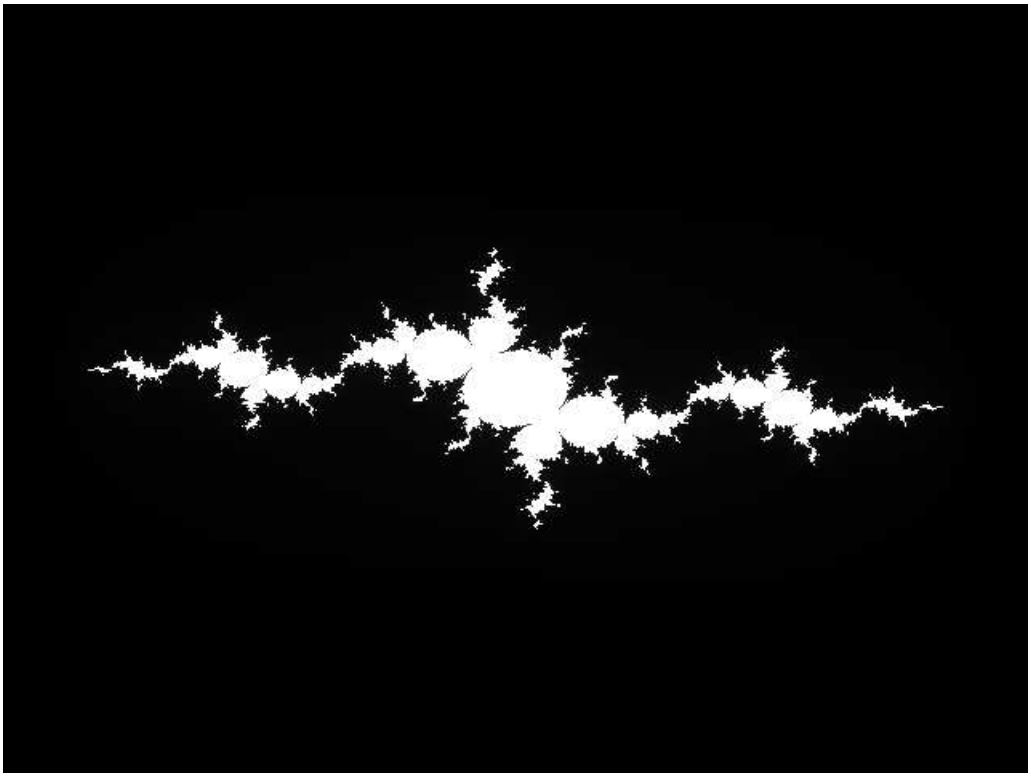

Figura 2: Región por default con $C = -1{,}125 - 0{,}216506350946 11i$.

```
$ tp0 -C -1.125-0.216506350946 11i -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.
- El código MIPS32 generado por el compilador[1];
- Este enunciado.

7. Fechas

Fecha de vencimiento: Martes 26/9.

[1]Por motivos prácticos, en la copia impresa sólo es necesario incluir la primera página del código assembly MIPS32 generado por el compilador.

1. Introducción

Este trabajo práctico tiene como objetivo realizar un graficador de Conjuntos de Julia. Este tipo de fractal se obtiene iterando distintos puntos especificos del plano complejo sobre una función holomorfa. La intención es realizar un sólido programa de linea de comando y que a su vez posea una especifica validación de parámetros. En caso de error deberia indicar al usuario especificamente en que parámetro se equivocó para facilitar su solución. Finalmente el programa debe cumplir con los estandarés de imagenes PGM en escala de grises. Detalles relacionados a PGM estan indicados en el enunciado del TP.

2. Diseño e implementación

Antes que nada, cabe informar que se utilizó una estructura llamada "complejo" para poder representarlos. La misma consiste de dos atributos de tipo float los cuales representan a la parte real e imaginaria del numero. El programa consiste primeramente en encontrar cada argumento indicado (pasado como parámetro al programa por línea de comandos), analizar si éstos son correctos, guardarlos en sus respectivas variables y, en caso de no ser correctos, informar el error y cortar la ejecución del programa.Todos los parámetros fueron guardados en variables estáticas, y en caso de no estar presentes se reemplazaron por aquellos dados en el enunciado por defecto. En caso de no poder crear un archivo de salida, también se sale del programa. Se escribió un programa secuencial completamente, sin el agregado de muchas estructuras adicionales ni funciones auxiliares, para que luego el seguimiento en lenguaje ensamblador sea más simple y entendible. El porgrama comienza con un ciclo switch en el que se almacenan todos los parámetros pasados al programa mediante línea de comandos, a través de la función getopt de la librería unistd. En caso de error, se sale del programa con EXIT-FAILURE. Para la verificación de la validez de los mismos se utilizaron distintas funciones auxiliares, como atoi para la conversión de string a entero, o sscanf para los parámetros que combinaban decimales y otros símbolos (como los complejos). En caso de error, se imprimen mensajes de error y se sale del programa con EXIT-FAILURE.La parte central del algoritmo, donde se itera en el plano complejo de forma discreta, se utilizan dos ciclos definidos for para el eje real y el eje imaginario, manteniendo un acumulador para definir el brillo del pixel, el cual luego es guardado en el archivo de salida. Para enontrar el paso de avance sobre los ejes del plano, se utilizó el cociente entre el alto o ancho y la resolución de la imagen, para el eje imaginario y real, respectivamente. La operación matemática que se realiza en el interior de los ciclos se implementó de la forma más eficiente posible encontrada, generando una única línea de código para la parte real e imaginaria, y usando una única variable temporal.

3. Como obtener el ejecutable y correr pruebas

Para mayor comodidad se dispone de un archivo llamado "script.sh" que al ejecutarlo (esto es, posicionándose sobre el directorio donde se encuentran los archivos y ejecutando el comando ./script.sh) automáticamente compilara el programa y ejecutará una serie de comandos de prueba y sus respectivas salidas.

4. Código

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <unistd.h>
#include <getopt.h>
#define N 255

typedef struct complejo{
float parteReal;
float parteImaginaria;
}complejo;

void print_usage() {
printf("Error de parametros ./tp0 -h para ayuda detallada.\nUso: ./tp0 -r 640x4
}


int main(int argc, char * const argv[]){
char* pathToExe = argv[0];
int option = -1;
int encontroRes = -1;
int encontroAlto = -1;
int encontroAncho = -1;
int encontroConst = -1;
int encontroOutput = -1;
int encontroOrigen = -1;
int salidaConsola = -1;
char* ptrOrigen;
char* constante;
char* output;
char* res;
char controli;
complejo constanteC;
complejo origen;
complejo pixel;
int resolucionHorizontal, resolucionVertical;
float anchoRectangulo;
float altoRectangulo;

while ((option = getopt(argc, argv,"r:c:C:W:H:o:h")) != -1) {
switch (option) {
case 'r' :
encontroRes = 1;
res = optarg;
break;
```

```
case 'c' :
encontroOrigen = 1;
ptrOrigen = optarg;
break;
case 'C' :
encontroConst = 1;
constante = optarg;
break;
case 'W' :
anchoRectangulo = atoi(optarg);
if (anchoRectangulo <= 0){
printf("Ancho plano complejo invalido.\n");
exit(EXIT_FAILURE);
}
encontroAncho = 1;
break;
case 'H' :
altoRectangulo = atoi(optarg);
if (altoRectangulo <= 0){
printf("Alto plano complejo invalido.\n");
exit(EXIT_FAILURE);
}
encontroAlto = 1;
break;
case 'o' :
encontroOutput = 1;
if (strcmp(optarg,"-") == 0) salidaConsola = 1;
output = optarg;
/*
if ((isalpha((char)optarg[0]) != 1) && (isdigit((char)optarg[0]) != 1)){
if (strcmp(optarg, "-") == 0) salidaConsola = 1;
else {
printf("fatal: cannot open output file.\n");
exit(0);
}
}
output = optarg*/
break;
case 'h' :
printf("AYUDA:  Resolucion: -r 640x480\n        Origen: -c 0+0i\n        Constar
return 0;
default:
print_usage();
exit(EXIT_FAILURE);
}
}

if (encontroOutput != 1){    ///UNICA CONDICION NECESARIA PARA CORRER EL PROGRAM
printf("El argumento -o es condición necesaria\n");
exit(EXIT_FAILURE);
```

```
}

if (encontroAlto != 1)  altoRectangulo = 4;
if (encontroAncho != 1) anchoRectangulo = 4;

int scanResult;
if (encontroConst == 1){
scanResult = sscanf(constante, "%f%f%c", &constanteC.parteReal, &constanteC.par
if (scanResult < 3){
printf("Error obteniendo constante compleja.\n");
print_usage();
exit(EXIT_FAILURE);
}
} else {
constanteC.parteReal = 0.285;
constanteC.parteImaginaria = -0.01;
}

if (encontroOrigen == 1){
scanResult = sscanf(ptrOrigen, "%f%f%c", &origen.parteReal, &origen.parteImagin
if (scanResult < 3 || controli != 'i'){
printf("Error obteniendo origen complejo.\n");
print_usage();
exit(EXIT_FAILURE);
}
} else {
origen.parteReal = 0;
origen.parteImaginaria = 0;
}

if (encontroRes == 1){
int scanResult = sscanf(res, "%d%c%d", &resolucionHorizontal, &controli, &resol
if (scanResult < 2 || resolucionVertical <= 0 || resolucionHorizontal <= 0 || c
{
if (resolucionVertical <= 0) printf("Resolucion vertical invalida\n");
else if (resolucionHorizontal <= 0) printf("Resolucion horizontal invalida\n");
else printf("Resolucion invalida\n");
print_usage();
exit(EXIT_FAILURE);
}
} else {
resolucionHorizontal = 640;
resolucionVertical = 480;
}


float pasoHorizontal = ((anchoRectangulo) / (float)(resolucionHorizontal))/2;
float pasoVertical = ((altoRectangulo) / (float)(resolucionVertical))/2;
int contadorBrillo;
float temp, valorAbsoluto;
```

```
FILE *fp;
fprintf(stdout, "P2 \n%d %d \n255 \n", resolucionHorizontal, resolucionVertical
if (salidaConsola != 1) {
fp=fopen(output, "w");
if(fp == NULL) {
printf("fatal: cannot open output file\n");
print_usage();
exit(EXIT_FAILURE);
}
fprintf(fp, "P2 \n%d %d \n255 \n", resolucionHorizontal, resolucionVertical);
}

//debug printf("constantR %f constanteI %f orinenR %f origenI %f \n", constante

for (int y = 1; y <= resolucionVertical; y++){
for (int x = 1; x <= resolucionHorizontal; x++){
pixel.parteReal = pasoHorizontal * (2 * x -1) - anchoRectangulo / 2 + origen.pa
pixel.parteImaginaria = -pasoVertical * (2 * y -1) + altoRectangulo / 2 + orige

for (contadorBrillo = 0; contadorBrillo <= N; contadorBrillo++){
valorAbsoluto = sqrtf((pixel.parteReal * pixel.parteReal) + (pixel.parteImagina
if (valorAbsoluto > 2) break;
temp = ((pixel.parteReal)*(pixel.parteReal))-((pixel.parteImaginaria)*(pixel.pa
pixel.parteImaginaria = ((pixel.parteImaginaria)*(pixel.parteReal))+((pixel.par
pixel.parteReal = temp;
}
if (salidaConsola == 1) fprintf(stdout, "%d ", contadorBrillo); else fprintf(fp
}
if (salidaConsola == 1) fprintf(stdout, "\n"); else fprintf(fp, "\n");
}

if (salidaConsola != 1){
rewind(fp);
fclose(fp);
printf("Archivo guardado en %s/%s\n", pathToExe, output);
}
/*
if (salidaConsola == 1){

fp=fopen(output, "rb+");
if(fp == NULL) return 1;
int c;

while((c=fgetc(fp)) != EOF){
printf( "%c", c );
}

fclose(fp);
remove(output);
```

```
}*/
return 0;
}
```

5. Código MIPS (Primeras hojas)

```
.section .mdebug.abi32
        .previous
        .abicalls
        .file 1 "tp0f.c"
        .section        .debug_abbrev,"",@progbits
$Ldebug_abbrev0:
        .section        .debug_info,"",@progbits
$Ldebug_info0:
        .section        .debug_line,"",@progbits
$Ldebug_line0:
        .text
$Ltext0:
        .file 2 "/usr/include/mips/int_types.h"
        .file 3 "/usr/include/mips/ansi.h"
        .file 4 "/usr/include/string.h"
        .file 5 "/usr/include/mips/types.h"
        .file 6 "/usr/include/sys/ansi.h"
        .file 7 "/usr/include/sys/types.h"
        .file 8 "/usr/include/sys/endian.h"
        .file 9 "/usr/include/pthread_types.h"
        .file 10 "/usr/include/stdlib.h"
        .file 11 "/usr/include/stdio.h"
        .file 12 "/usr/include/math.h"
        .file 13 "/usr/include/unistd.h"
        .file 14 "/usr/include/getopt.h"
        .rdata
        .align  2
$LC0:
        .ascii  "Error de parametros ./tp0 -h para ayuda detallada.\n"
        .ascii  "Uso: ./tp0 -r 640x480 -c 0+0i -C 0.285-0,01i -w 4 -H 4 -"
        .ascii  "o /home/user/julia.pgm (or -o - for stdout)\n\000"
        .text
        .align  2
        .globl  print_usage
$LFB29:
        .loc 1 15 0
        .ent    print_usage
print_usage:
        .frame  $fp,40,$31              # vars= 0, regs= 3/0, args= 16, extra=
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    reorder
```

```
        subu    $sp,$sp,40
        .cprestore 16
$LCFI0:
        sw      $31,32($sp)
$LCFI1:
        sw      $fp,28($sp)
$LCFI2:
        sw      $28,24($sp)
$LCFI3:
        move    $fp,$sp
$LCFI4:
        .loc 1 16 0
        la      $4,$LC0
        la      $25,printf
        jal     $31,$25
        .loc 1 17 0
        move    $sp,$fp
        lw      $31,32($sp)
        lw      $fp,28($sp)
        addu    $sp,$sp,40
        j       $31
        .end    print_usage
$LFE29:
        .size   print_usage, .-print_usage
        .rdata
        .align  2
$LC1:
        .ascii  "r:c:C:W:H:o:h\000"
        .align  2
$LC2:
        .ascii  "Ancho plano complejo invalido.\n\000"
        .align  2
$LC3:
        .ascii  "Alto plano complejo invalido.\n\000"
        .align  2
$LC4:
        .ascii  "-\000"
        .align  2
$LC5:
        .ascii  "AYUDA:\tResolucion: -r 640x480\n"
        .ascii  "\tOrigen: -c 0+0i\n"
        .ascii  "\tConstante imaginaria: -C 0.285-0,01i\n"
        .ascii  "\tAncho del plano complejo: -W 4\n"
        .ascii  "\tAlto del plano complejo: -H 4\n"
        .ascii  "\tFormato de salida: -o /home/julia.pgm (PARAMETRO IMPRE"
        .ascii  "SCINDIBLE) (or -o - for stdout)\n\000"
        .align  2
$LC6:
        .ascii  "El argumento -o es condici\303\263n necesaria\n\000"
        .align  2
```

```
$LC8:
        .ascii  "%f %f %c\000"
        .align  2
$LC9:
        .ascii  "Error obteniendo constante compleja.\n\000"
        .align  2
$LC12:
        .ascii  "Error obteniendo origen complejo.\n\000"
        .align  2
$LC13:
        .ascii  "%dx%d\000"
        .align  2
$LC14:
        .ascii  "Resolucion vertical invalida\n\000"
        .align  2
$LC15:
        .ascii  "Resolucion horizontal invalida\n\000"
        .align  2
$LC17:
        .ascii  "w\000"
        .align  2
$LC18:
        .ascii  "fatal: cannot open output file\n\000"
        .align  2
$LC19:
        .ascii  "P2 \n"
        .ascii  "%d %d \n"
        .ascii  "255 \n\000"
        .align  2
$LC20:
        .ascii  "%d \000"
        .align  2
$LC21:
        .ascii  "\n\000"
        .align  2
$LC22:
        .ascii  "Archivo guardado en %s/%s\n\000"
        .align  2
$LC7:
        .word   1082130432
        .align  2
$LC10:
        .word   1049750405
        .align  2
$LC11:
        .word   -1138501878
        .align  2
$LC16:
        .word   1073741824
        .text
```

```
        .align  2
        .globl  main
$LFB31:
        .loc 1 20 0
        .ent    main
main:
        .frame  $fp,192,$31             # vars= 144, regs= 3/0, args= 24, extra
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    reorder
        subu    $sp,$sp,192
        .cprestore 24
$LCFI5:
        sw      $31,184($sp)
$LCFI6:
        sw      $fp,180($sp)
$LCFI7:
        sw      $28,176($sp)
$LCFI8:
        move    $fp,$sp
$LCFI9:
        sw      $4,192($fp)
        sw      $5,196($fp)
        .loc 1 21 0
$LBB2:
        lw      $2,196($fp)
        lw      $2,0($2)
        sw      $2,32($fp)
        .loc 1 22 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,36($fp)
        .loc 1 23 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,40($fp)
        .loc 1 24 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,44($fp)
        .loc 1 25 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,48($fp)
        .loc 1 26 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,52($fp)
        .loc 1 27 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,56($fp)
        .loc 1 28 0
        li      $2,-1                   # 0xffffffffffffffff
```

```
        sw      $2,60($fp)
        .loc 1 29 0
        li      $2,-1                   # 0xffffffffffffffff
        sw      $2,64($fp)
        .loc 1 42 0
$L19:
        lw      $4,192($fp)
        lw      $5,196($fp)
        la      $6,$LC1
        la      $25,getopt
        jal     $31,$25
        sw      $2,36($fp)
        lw      $3,36($fp)
        li      $2,-1                   # 0xffffffffffffffff
        bne     $3,$2,$L21
        b       $L20
$L21:
        .loc 1 43 0
        lw      $2,36($fp)
        addu    $2,$2,-67
        sw      $2,168($fp)
        lw      $3,168($fp)
        sltu    $2,$3,48
        beq     $2,$0,$L35
        lw      $2,168($fp)
        sll     $3,$2,2
        la      $2,$L36
        addu    $2,$3,$2
        lw      $2,0($2)
        .cpadd  $2
        j       $2
        .rdata
        .align  2
```

6. Conclusiones

Se tarda poco en realizarse que el verdadero desafío que se presenta durante la codificación del programa es la validación de los párametros. El algorítmo para obtener el set de Julia es fácil de implementar y no debería de requerir funciones adicionales. Resultó interesante incrementar la resolución para la misma superficie compleja y observar cuan profundo se puede *zoomear* en las imagenes resultantes.

Referencias

[1] Conjunto de Julia, WikiPedia, https://es.wikipedia.org/wiki/Conjunto_de_Julia.

[2] The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie.