Univesidad de Buenos Aires - FIUBA
66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica

Barrera Oro, Rafael (83240)
Porras Carhuamaca, Sherly (91076)
García, Renán (100405)

13 de septiembre de 2016

Índice

1. Documentación de diseño

1.1. Estructura del programa

La lógica del programa se compartimentó de manera tal que los tres pasos fundamentales necesarios para resolver el problema queden claramente separados.

1.1.1. Parseo de parámetros

Todo lo relativo a obtener los parámetros de la línea de comando, parsearlos, validarlos y formatear sus valores correctamente.

1.1.2. Cálculo de velocidades de escape

Dependiendo de la resolución indicada por el usuario, la imagen a obtener estará conformada por N pixeles cuyo brillo representará la velocidad de escape de un número complejo z perteneciente al área determinada por los parámetros recibidos. Para obtener la imagen final resulta necesario dividir dicha área del plano complejo en una grilla de dimensiones correspondientes a la resolución de la imagen y por cada casillero obtener el número complejo que corresponde a su punto medio, considerando que el casillero corresponde a un pixel en la imagen a generar y la velocidad de escape del número complejo obtenido a su brillo (a obtenerse a través del algoritmo proveido por el enunciado).

A partir de los parámetros: resolución vertical r_v, resolución horizontal r_h, centro del area c, ancho del área w y alto del área h dedujimos las siguientes fórmulas para obtener las partes real e imaginaria del número complejo z correspondiente al punto medio cuya velocidad de escape determinará el brillo del pixel (i, j) en la imagen. Además teniendo en cuenta que $c, z \in \mathbb{C}$, $r_h, r_v \in \mathbb{N}$, $w, h \in \mathbb{R}$, $i \in < 0, r_h - 1 >$ y que $j \in < 0, r_v - 1 >$:

$$\Delta X = \frac{w}{r_h}$$

$$\Delta Y = \frac{h}{r_v}$$

$$Re(z) = Re(c) - \frac{w}{2} + \frac{\Delta X}{2} \times (2i + 1)$$

$$Im(z) = Im(c) - \frac{h}{2} + \frac{\Delta Y}{2} \times [2(r_v - j - 1) + 1]$$

La varación en la fórmula para $Im(z)$ se debe a que si bien todo pixel de índice vertical 0 es inmediatamente próximo al borde izquierdo de la imagen, todo pixel de índice vertical 0 está inmediatamente próximo al borde superior de la imagen, y debo evitar que un pixel de la porción superior de la imagen quede asociado a un número complejo perteneciente a la parte inferior del área del plano.

Para probar las fórmulas podemos aplicarlas al caso particular de la imagen monopixel, es decir, de resolución vertical y horizontal unitaria ($r_h = 1 \wedge r_v = 1$).

$$\Delta X = \frac{w}{1} = w$$

$$\Delta Y = \frac{h}{1} = h$$

$$Re(z) = Re(c) - \frac{w}{2} + \frac{w}{2} \times (2i + 1)$$

$$Im(z) = Im(c) - \frac{h}{2} + \frac{h}{2} \times [2(1 - j - 1) + 1]$$

Como solo existe un pixel, las únicas coordenadas posibles son $(i, j) = (0, 0)$

$$Re(z) = Re(c) - \frac{w}{2} + \frac{w}{2} \times (2 \times 0 + 1)$$

$$Im(z) = Im(c) - \frac{h}{2} + \frac{h}{2} \times [2 \times (-0) + 1]$$

Finalmente:

$$Re(z) = Re(c)$$

$$Im(z) = Im(c)$$

El punto medio resulta (correctamente) ser el centro de la imagen del área elegida del plano complejo.

1.1.3. Generación de archivo

Confección del archivo en formato pgm y salida del mismo (ya sea por salida estandar o a un archivo definido en disco).

2. Documentación de implementación

2.1. Estructuras

2.1.1. Complejo

Contiene las partes real e imaginaria de un número complejo.

```
typedef struct {
        float real;
        float imag;
} Complejo;
```

2.1.2. Area

Contiene los parámetros que definen un área del plano complejo, centro, ancho y alto.

```
typedef struct {
        Complejo centro;
        float ancho;
        float altura;
} Area;
```

2.1.3. Resolucion

Contiene los parámetros que definen una resolución (horizontal y vertical).

```
typedef struct {
        int horizontal;
        int vertical;
} Resolucion;
```

2.1.4. Parametros

Contiene todos los parámetros que se pueden recibir de la línea de comando.

```
typedef struct {
        Complejo c;
        Area area;
        Resolucion resolucion;
        char* archivo;
} Parametros;
```

2.2. Funciones

2.2.1. error

```
void error(int error)
```

Recibe un código de error, muestra el correspondiente mensaje por salida estandar y finaliza la ejecución con el código de retorno correspondiente.

2.2.2. parsear_complejo

```
void parsear_complejo(char* str_complejo, Complejo* complejo)
```

Recibe una cadena de caracteres describiendo un número complejo y un puntero a una estructura de tipo Complejo, de ser el formato del primero correcto se actualizan las variables de la segunda.

2.2.3. parsear_resolucion

```
void parsear_resolucion(char* str_resolucion, Resolucion* resolucion)
```

Recibe una cadena de caracteres describiendo una resolución y un puntero a una estructura de tipo Resolucion, de ser el formato de la primera correcto se actualizan las variables de la segunda.

2.2.4. parsear_parametros

```
void parsear_parametros(int argc, char** argv, Parametros* parametros)
```

Recibe la cantidad y los valores de los argumentos recibidos de línea de comando más un puntero a una estructura de tipo Parametros, de parsearse correctamente todos los párametros la estructura es llenada con todos los valores recibidos (se utilizan los valores default para los parámetros no recibidos).

2.2.5. modulo

```
float modulo(Complejo* z)
```

Recibe un puntero a una estructura de tipo complejo y devuelve el valor del módulo del número complejo representado por la misma.

2.2.6. sumar

```
void sumar(Complejo* operando1, Complejo* operando2, Complejo* resultado)
```

Recibe tres punteros a estructuras de tipo Complejo, se utilizan los primeros para realizar una suma y el tercero para guardar el resultado.

2.2.7. cuadrado

```
void cuadrado(Complejo* operando, Complejo* resultado)
```

Recibe dos punteros a estructuras de tipo Complejo, se asigna a la segunda estructura el resultado de elevar el complejo representado por la primera al cuadrado.

2.2.8. calcular_complejo

```
void calcular_complejo(int x, int y, Resolucion* resolucion, Area* area, Compl
```

Recibe las coordenadas (x, y) de un pixel, punteros a los parámetros de resolución y área y por último uno a una estructura de tipo Complejo, en esta última se almacena el complejo correspondiente al punto medio representado por el pixel definido por las coordenadas recibidas.

2.2.9. calcular_velocidad

```
void calcular_velocidad(Complejo* z, Complejo* c, int* velocidad)
```

Recibe dos punteros a estructuras de tipo complejo y un puntero a un entero, en este último se almacenará la velocidad de escape del complejo z calculada con el parámetro c.

2.2.10. calcular_velocidades

```
int* calcular_velocidades(Parametros* parametros)
```

Recibe un puntero a una estructura de tipo Parametros y devuelve un array de enteros creado con memoria dinámica (que debe ser liberada antes de que termine la ejecución) con las velocidades de escape de cada número complejo representado por algún pixel de la imagen, es decir, el brillo de cada uno.

2.2.11. generar_pgm

```
void generar_pgm(int* velocidades, Parametros* parametros)
```

Recibe un array con las velocidades de escape (es decir, brillos de cada pixel) y un puntero a una estructura conteniendo los valores de los parámetros que definen la imagen. Con los mismos genera un archivo con la imagen en formato pgm en el destino determinado por el usuario.

2.2.12. main

```
int main(int argc, char** argv)
```

Punto de entrada del programa, se realiza una validación inicial de los parámetros mediante la inspección de la cantidad de los mismos y se verifica si se debe mostrar la leyenda de uso, de lo contrario se prosigue con la ejecución en tres pasos: parseo de parámetros, cálculo de velocidades de escape, generación de archivo y liberación de memoria.

3. Documentación de compilación

3.1. Compilación

Se ha incluido un archivo Makefile para simplificar la obtención del ejecutable, el mismo puede obtenerse simplemente mediante la ejecución del comando *make*, que generará un archivo binario *tp0*:

```
$ make
gcc -std=gnu99 tp0.c -lm -o tp0
$ ls
tp0.c Makefile tp0
```

3.2. Ejecución

Una vez obtenido el ejecutable, el mismo se puede ejecutar con el parámetro *-h* para obtener la leyenda de ayuda:

```
$ ./tp0 -h
Uso: ./tpo [-h] [-r <ancho>x<altura>] [-c <real>+<imag>i] [-C <real
    >+<imag>i] [-w <ancho>] [-H <altura>] [-o <archivo>|-]
```

O utilizando cualquiera de los parámetros requeridos por el enunciado:

```
$ ./tp0 -o uno.pgm
```

4. Casos de prueba

Todos los casos de prueba que producen un archivo pueden ejecutarse utilizando el Makefile:

```
$ make pruebas
gcc -std=gnu99 tp0.c -lm -o tp0
mkdir pruebas
./tp0 -o pruebas/uno.pgm
./tp0 -o pruebas/centro.pgm
./tp0 -c 0+1i -o pruebas/abajo.pgm
./tp0 -c 0-1i -o pruebas/arriba.pgm
./tp0 -c 1+0i -o pruebas/izquierda.pgm
./tp0 -c -1+0i -o pruebas/derecha.pgm
./tp0 -c 1-1i -o pruebas/arriba_izquierda.pgm
./tp0 -c -1-1i -o pruebas/arriba_derecha.pgm
./tp0 -c -1+1i -o pruebas/abajo_derecha.pgm
./tp0 -c 1+1i -o pruebas/abajo_izquierda.pgm
./tp0 -r 1980x1080 -o pruebas/hiref.pgm
./tp0 -w 1 -H 1 -o pruebas/zoom.pgm
./tp0 -C 0.279+0i -w 2.5 -H 2.5 -r 1024x768 -o pruebas/alt.pgm
```

4.1. Validación de parámetros

4.1.1. Resolución

```
$ ./tp0 -r 0x1
Error: resolucion horizontal
$ ./tp0 -r -1x1
Error: resolucion horizontal
$ ./tp0 -r 1x0
Error: resolucion vertical
$ ./tp0 -r 1x-1
Error: resolucion vertical
```

4.1.2. Área

```
$ ./tp0 -w 0
Error: ancho del area
$ ./tp0 -w -1
Error: ancho del area
$ ./tp0 -H 0
Error: altura del area
$ ./tp0 -H -1
Error: altura del area
$ ./tp0 -c -1
Error: formato de numero complejo debe ser <real>+<imag>i
$ ./tp0 -c -1+2
Error: formato de numero complejo debe ser <real>+<imag>i
```

4.1.3. Párametro C

```
$ ./tp0 -C -1
Error: formato de numero complejo debe ser <real>+<imag>i
$ ./tp0 -C -1+2
Error: formato de numero complejo debe ser <real>+<imag>i
```

4.1.4. Archivo de salida

```
$ ./tp0 -o /tmp/
Error: No se pudo crear el archivo
```

4.2. Posición

4.2.1. Área default en el centro de la imagen

```
$ ./tp0 -o centro.pgm
```

4.2.2. Área default a la derecha de la imagen

```
$ ./tp0 -c -1+0i -o derecha.pgm
```

4.2.3. Área default abajo a la derecha de la imagen

```
$ ./tp0 -c -1+1i -o abajo_derecha.pgm
```

4.2.4. Área default abajo de la imagen

```
$ ./tp0 -c 0+1i -o abajo.pgm
```

4.2.5. Área default abajo a la izquierda de la imagen

```
$ ./tp0 -c 1+1i -o abajo_izquierda.pgm
```

4.2.6. Área default a la izquierda de la imagen

```
$ ./tp0 -c 1+0i -o izquierda.pgm
```

4.2.7. Área default arriba a la izquierda de la imagen

```
$ ./tp0 -c 1-1i -o arriba_izquierda.pgm
```

4.2.8. Área default arriba de la imagen

```
$ ./tp0 -c 0-1i -o arriba.pgm
```

4.2.9. Área default arriba a la derecha de la imagen

```
$ ./tp0 -c -1-1i -o arriba_derecha.pgm
```

4.3. Misceláneos

4.3.1. Ayuda

```
$ ./tp0 -h
Uso: ./tpo [-h] [-r <ancho>x<altura>] [-c <real>+<imag>i] [-C <real
    >+<imag>i] [-w <ancho>] [-H <altura>] [-o <archivo>|-]
```

4.3.2. Imagen de un punto de lado centrada en el plano complejo

```
$ ./tp0 -c 0.01+0i -r 1x1 -o -
P2
#Conjunto de Julia#
1 1
255
19
```

4.3.3. Idem en un punto que no pertenece al conjunto

```
$ ./tp0 -c 10+0i -r 1x1 -o -P2
P2
#Conjunto de Julia#
1 1
255
0
```

4.3.4. Imagen de alta resolución

```
$ ./tp0 -r 1980x1080 -o hiref.pgm
```

4.3.5. Zoom sobre el origen

```
$ ./tp0 -w 1 -H 1 -o zoom.pgm
```

4.3.6. Conjunto de Julia alternativo al default

```
$ ./tp0 -C 0.279+0i -w 2.5 -H 2.5 -r 1024x768 -o alt.pgm
```

5. Código fuente

5.1. C

```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define EXITO 0
#define ERROR_PARAMETROS 1
#define ERROR_ARCHIVO 2
#define ERROR_C 3
#define ERROR_AREA_CENTRO 4
#define ERROR_AREA_ANCHO 5
#define ERROR_AREA_ALTURA 6
#define ERROR_RESOLUCION_HORIZONTAL 7
#define ERROR_RESOLUCION_VERTICAL 8
#define ERROR_MEMORIA 9
#define ERROR_COMPLEJO 10

#define ANCHO_RESOLUCION_DEFECTO 640
#define ALTURA_RESOLUCION_DEFECTO 480
#define CENTRO_DEFECTO_REAL 0
#define CENTRO_DEFECTO_IMAG 0
#define C_DEFECTO_REAL 0.285
#define C_DEFECTO_IMAG -0.01
#define ANCHO_AREA_DEFECTO 4
#define ALTURA_AREA_DEFECTO 4
#define SALIDA_DEFECTO "-"

#define PARAMETRO_RESOLUCION "-r"
#define PARAMETRO_CENTRO "-c"
#define PARAMETRO_C "-C"
#define PARAMETRO_ANCHO_AREA "-w"
#define PARAMETRO_ALTURA_AREA "-H"
#define PARAMETRO_SALIDA "-o"
#define PARAMETRO_AYUDA "-h"

#define MENSAJE_AYUDA "Uso: ./tpo [-h] [-r <ancho>x<altura>] [-c <
    real>+<imag>i] [-C <real>+<imag>i] [-w <ancho>] [-H <altura>]
    [-o <archivo>|-]"

#define MAX_VELOCIDAD 255

#define NUMERO_I 'i'

typedef struct {
        float real;
        float imag;
} Complejo;

typedef struct {
        Complejo centro;
        float ancho;
        float altura;
} Area;

typedef struct {
        int horizontal;
        int vertical;
} Resolucion;
```

```
57
58  typedef struct {
59          Complejo c;
60          Area area;
61          Resolucion resolucion;
62          char* archivo;
63  } Parametros;
64
65  void error(int error) {
66          char* mensaje;
67
68          switch(error) {
69                  case ERROR_PARAMETROS:
70                          mensaje = MENSAJE_AYUDA;
71                          break;
72                  case ERROR_ARCHIVO:
73                          mensaje = "No se pudo crear el archivo";
74                          break;
75                  case ERROR_C:
76                          mensaje = "parametro C";
77                          break;
78                  case ERROR_AREA_CENTRO:
79                          mensaje = "centro del area";
80                          break;
81                  case ERROR_AREA_ANCHO:
82                          mensaje = "ancho del area";
83                          break;
84                  case ERROR_AREA_ALTURA:
85                          mensaje = "altura del area";
86                          break;
87                  case ERROR_RESOLUCION_HORIZONTAL:
88                          mensaje = "resolucion horizontal";
89                          break;
90                  case ERROR_RESOLUCION_VERTICAL:
91                          mensaje = "resolucion vertical";
92                          break;
93                  case ERROR_MEMORIA:
94                          mensaje = "memoria insuficiente";
95                  case ERROR_COMPLEJO:
96                          mensaje = "formato de numero complejo debe
                                ser <real>+<imag>i";
97                          break;
98          }
99
100         printf("Error: %s\n", mensaje);
101         exit(error);
102 }
103
104 void parsear_complejo(char* str_complejo, Complejo* complejo) {
105
106         //si no hay una i al final el numero complejo no esta bien
                formateado
107     if(str_complejo[strlen(str_complejo) - 1] != NUMERO_I) {
108
109                 error(ERROR_COMPLEJO);
110     }
111
112     //busco el + del medio
113     char* str_imag = strstr(str_complejo + 1, "+");
114
115     //si no lo encontre, busco el - del medio
116     if(str_imag == NULL) {
```

```
117
118         str_imag = strstr(str_complejo + 1, "-");
119     }
120
121         //si sigue siendo null entonces no habia un numero complejo
                bien escrito
122         if(str_imag == NULL) {
123
124                 error(ERROR_PARAMETROS);
125     }
126
127     //calculo la posicion en el string del + o - encontrado
128     int pos = str_imag - str_complejo;
129
130     //creo un string para la parte real
131     char str_real[pos + 1];
132     str_real[pos] = '\0';
133
134     //copio solo la parte real del string original
135     strncpy(str_real, str_complejo, pos);
136
137     //piso el i del final del string original en la parte
            imaginaria
138     str_imag[strlen(str_imag)] = '\0';
139
140         complejo->real = atof(str_real);
141     complejo->imag = atof(str_imag);
142 }
143
144 void parsear_resolucion(char* str_resolucion, Resolucion*
        resolucion) {
145
146         resolucion->horizontal = atoi(strsep(&str_resolucion, "x"))
                ;
147
148         if(resolucion->horizontal <= 0) {
149
150                 error(ERROR_RESOLUCION_HORIZONTAL);
151         }
152
153         resolucion->vertical = atoi(strsep(&str_resolucion, "x"));
154
155         if(resolucion->vertical <= 0) {
156
157                 error(ERROR_RESOLUCION_VERTICAL);
158         }
159 }
160
161 void parsear_parametros(int argc, char** argv, Parametros*
        parametros) {
162
163         if(argc != 1 && argc != 3 && argc != 5 && argc != 7 && argc
                != 9 && argc != 11) {
164
165                 error(ERROR_PARAMETROS);
166         }
167
168         parametros->c.real = C_DEFECTO_REAL;
169         parametros->c.imag = C_DEFECTO_IMAG;
170
171         parametros->resolucion.horizontal =
                ANCHO_RESOLUCION_DEFECTO;
```

```
172         parametros->resolucion.vertical = ALTURA_RESOLUCION_DEFECTO
                ;
173
174         parametros->area.centro.real = CENTRO_DEFECTO_REAL;
175         parametros->area.centro.imag = CENTRO_DEFECTO_IMAG;
176         parametros->area.ancho = ANCHO_AREA_DEFECTO;
177         parametros->area.altura = ALTURA_AREA_DEFECTO;
178
179         parametros->archivo = SALIDA_DEFECTO;
180
181         for(int i = 0; i < argc; i++) {
182
183                 if(strcmp(argv[i], PARAMETRO_C) == 0) {
184
185                         parsear_complejo(argv[i + 1], &parametros->
                            c);
186                 }
187
188                 if(strcmp(argv[i], PARAMETRO_RESOLUCION) == 0) {
189
190                         parsear_resolucion(argv[i + 1], &parametros
                            ->resolucion);
191                 }
192
193                 if(strcmp(argv[i], PARAMETRO_CENTRO) == 0) {
194
195                         parsear_complejo(argv[i + 1], &parametros->
                            area.centro);
196                 }
197
198                 if(strcmp(argv[i], PARAMETRO_ANCHO_AREA) == 0) {
199
200                         parametros->area.ancho = atof(argv[i + 1]);
201
202                         if(parametros->area.ancho <= 0) {
203
204                                 error(ERROR_AREA_ANCHO);
205                         }
206                 }
207
208                 if(strcmp(argv[i], PARAMETRO_ALTURA_AREA) == 0) {
209
210                         parametros->area.altura = atof(argv[i + 1])
                            ;
211
212                         if(parametros->area.altura <= 0) {
213
214                                 error(ERROR_AREA_ALTURA);
215                         }
216                 }
217
218                 if(strcmp(argv[i], PARAMETRO_SALIDA) == 0) {
219
220                         parametros->archivo = argv[i + 1];
221                 }
222         }
223 }
224
225 float modulo(Complejo* z) {
226         return sqrt(z->real * z->real + z->imag * z->imag);
227 }
228
```

```
229 void sumar(Complejo* operando1, Complejo* operando2, Complejo*
        resultado) {
230
231         resultado->real = operando1->real + operando2->real;
232         resultado->imag = operando1->imag + operando2->imag;
233 }
234
235 void cuadrado(Complejo* operando, Complejo* resultado) {
236
237         resultado->real = operando->real * operando->real -
                operando->imag * operando->imag;
238         resultado->imag = 2 * operando->real * operando->imag;
239 }
240
241 void calcular_complejo(int x, int y, Resolucion* resolucion, Area*
        area, Complejo* complejo) {
242
243         float deltaX = area->ancho / resolucion->horizontal;
244         float deltaY = area->altura / resolucion->vertical;
245
246         complejo->real = area->centro.real - (area->ancho / 2) + (
                deltaX / 2) * (2 * x + 1);
247         complejo->imag = area->centro.imag - (area->altura / 2) + (
                deltaY / 2) * (2 * (resolucion->vertical - y - 1) + 1);
248 }
249
250 void calcular_velocidad(Complejo* z, Complejo* c, int* velocidad) {
251         Complejo z_cuadrado;
252
253         for (*velocidad = 0; *velocidad <= MAX_VELOCIDAD; ++(*
                velocidad)) {
254
255                 if (modulo(z) > 2) {
256                         break;
257                 }
258
259                 cuadrado(z, &z_cuadrado);
260
261                 sumar(&z_cuadrado, c, z);
262         }
263 }
264
265 int* calcular_velocidades(Parametros* parametros) {
266
267         int* velocidades = (int*)malloc(sizeof(int) * parametros->
                resolucion.horizontal * parametros->resolucion.vertical
                );
268
269         for(int y = 0; y < parametros->resolucion.vertical; y++) {
270                 for(int x = 0; x < parametros->resolucion.
                        horizontal; x++) {
271
272                         Complejo z;
273
274                         calcular_complejo(x, y, &parametros->
                                resolucion, &parametros->area, &z);
275
276                         int velocidad;
277
278                         calcular_velocidad(&z, &parametros->c, &
                                velocidad);
279
```

```
280                         velocidades[x + y * parametros->resolucion.
                                horizontal] = velocidad;
281                 }
282         }
283
284         return velocidades;
285 }
286
287 void generar_pgm(int* velocidades, Parametros* parametros) {
288
289         FILE* archivo;
290
291         if(strcmp(parametros->archivo, SALIDA_DEFECTO) == 0) {
292                 archivo = stdout;
293         }
294         else {
295
296                 archivo = fopen(parametros->archivo, "w");
297
298                 if(archivo == NULL) {
299
300                         error(ERROR_ARCHIVO);
301                 }
302         }
303
304         fprintf(archivo, "P2\n");
305         fprintf(archivo, "#Conjunto de Julia#\n");
306         fprintf(archivo, "%d %d\n", parametros->resolucion.
                horizontal, parametros->resolucion.vertical);
307         fprintf(archivo, "255\n");
308
309         for(int y = 0; y < parametros->resolucion.vertical; y++) {
310                 for(int x = 0; x < parametros->resolucion.
                        horizontal; x++) {
311
312                         fprintf(archivo, "%d ", velocidades[x + y *
                                parametros->resolucion.horizontal]);
313                 }
314
315                 fprintf(archivo, "\n");
316         }
317
318         fclose(archivo);
319 }
320
321 int main(int argc, char** argv) {
322
323         if(argc == 2 && strcmp(PARAMETRO_AYUDA, argv[1]) == 0) {
324                 printf(MENSAJE_AYUDA);
325                 printf("\n");
326         }
327         else {
328
329                 Parametros parametros;
330
331                 parsear_parametros(argc, argv, &parametros);
332
333                 int* velocidades = calcular_velocidades(&parametros
                        );
334
335                 generar_pgm(velocidades, &parametros);
336
```

```
337                     free(velocidades);
338             }
339
340             return EXITO;
341     }
```

tp0.c

5.2. Assembly (MIPS)

El siguiente es un extracto del código assembly generado con los mismos flags de compilación utilizados para generer al ejecutable más *-O0* para evitar cualquier optimización que pueda alterar el resultado (dentro del entorno MIPS emulado mediante el gxemul y la imagen netbsd):

```
root@:/tmp# make asm
gcc -O0 -S -std=gnu99 tp0.c -lm -o tp0.s
gcc: -lm: linker input file unused because linking not done
root@:/tmp# ls
Makefile tp0.c     tp0.s
```

```
    .file   1 "tp0.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2
$LC0:
    .ascii  "Uso: ./tpo [-h] [-r <ancho>x<altura>] [-c <real>+<imag>i"
    .ascii  "] [-C <real>+<imag>i] [-w <ancho>] [-H <altura>] [-o <ar"
    .ascii  "chivo>|-]\000"
    .align  2
$LC1:
    .ascii  "No se pudo crear el archivo\000"
    .align  2
$LC2:
    .ascii  "parametro C\000"
    .align  2
$LC3:
    .ascii  "centro del area\000"
    .align  2
$LC4:
    .ascii  "ancho del area\000"
    .align  2
$LC5:
    .ascii  "altura del area\000"
    .align  2
$LC6:
    .ascii  "resolucion horizontal\000"
    .align  2
$LC7:
    .ascii  "resolucion vertical\000"
    .align  2
$LC8:
    .ascii  "memoria insuficiente\000"
    .align  2
$LC9:
    .ascii  "formato de numero complejo debe ser <real>+<imag>i\000"
    .align  2
$LC10:
    .ascii  "Error: %s\n\000"
    .text
    .align  2
    .globl  error
    .ent    error
```

```
45 error:
46     .frame  $fp,48,$31      # vars= 8, regs= 3/0, args= 16, extra=
           8
47     .mask   0xd0000000,-8
48     .fmask  0x00000000,0
49     .set    noreorder
50     .cpload $25
51     .set    reorder
52     subu    $sp,$sp,48
53     .cprestore 16
54     sw  $31,40($sp)
55     sw  $fp,36($sp)
56     sw  $28,32($sp)
57     move    $fp,$sp
58     sw  $4,48($fp)
59     lw  $2,48($fp)
60     sltu    $2,$2,11
61     beq $2,$0,$L18
62     lw  $2,48($fp)
63     sll $3,$2,2
64     la  $2,$L29
65     addu    $2,$3,$2
66     lw  $2,0($2)
67     .cpadd  $2
68     j   $2
69     .rdata
70     .align  2
71 $L29:
72     .gpword $L18
73     .gpword $L19
74     .gpword $L20
75     .gpword $L21
76     .gpword $L22
77     .gpword $L23
78     .gpword $L24
79     .gpword $L25
80     .gpword $L26
81     .gpword $L27
82     .gpword $L28
83     .text
84 $L19:
85     la  $2,$LC0
86     sw  $2,24($fp)
87     b   $L18
88 $L20:
89     la  $2,$LC1
90     sw  $2,24($fp)
91     b   $L18
92 $L21:
93     la  $2,$LC2
94     sw  $2,24($fp)
95     b   $L18
96 $L22:
97     la  $2,$LC3
98     sw  $2,24($fp)
99     b   $L18
100 $L23:
101     la  $2,$LC4
102     sw  $2,24($fp)
103     b   $L18
104 $L24:
105     la  $2,$LC5
```

```
106     sw   $2,24($fp)
107     b    $L18
108 $L25:
109     la   $2,$LC6
110     sw   $2,24($fp)
111     b    $L18
112 $L26:
113     la   $2,$LC7
114     sw   $2,24($fp)
115     b    $L18
116 $L27:
117     la   $2,$LC8
118     sw   $2,24($fp)
119 $L28:
120     la   $2,$LC9
121     sw   $2,24($fp)
122 $L18:
123     la   $4,$LC10
124     lw   $5,24($fp)
125     la   $25,printf
126     jal  $31,$25
127     lw   $4,48($fp)
128     la   $25,exit
129     jal  $31,$25
130     .end    error
131     .size   error, .-error
132     .rdata
133     .align  2
134 $LC11:
135     .ascii  "+\000"
136     .align  2
137 $LC12:
138     .ascii  "-\000"
139     .text
140     .align  2
141     .globl  parsear_complejo
142     .ent    parsear_complejo
143 parsear_complejo:
144     .frame  $fp,64,$31      # vars= 16, regs= 5/0, args= 16, extra=
            8
145     .mask   0xd0030000,-8
146     .fmask  0x00000000,0
147     .set    noreorder
148     .cpload $25
149     .set    reorder
150     subu    $sp,$sp,64
151     .cprestore 16
152     sw   $31,56($sp)
153     sw   $fp,52($sp)
154     sw   $28,48($sp)
155     sw   $17,44($sp)
156     sw   $16,40($sp)
157     move    $fp,$sp
158     sw   $4,64($fp)
159     sw   $5,68($fp)
160     sw   $sp,32($fp)
161     lw   $4,64($fp)
162     la   $25,strlen
163     jal  $31,$25
164     ...
```

tp0_red.s