ORGANIZACIÓN DE COMPUTADORAS

INFORME
TRABAJO PRÁCTICO 0

Alumnos

93198 - Peña, Maximiliano
maxipenia@gmail.com

93665 - Poggio, Demian
demianpoggio@gmail.com

95505 - Iogha, Octavio
octaviomdq93@gmail.com

Fecha de Vencimiento

Martes 27 de Septiembre

1. Objetivos

El objetivo del trabajo práctico es familiarizarnos con las herramientas que necesitaremos para trabajar a lo largo de la cursada. Estas herramientas son:

- Compilador GCC
- Shell de Unix
- Emulador GXEmul

2. Introducción Teórica

Para poner en práctica estas herramientas, se debe implementar un programa en el lenguaje de programación C que permita, ingresando diversos parametros, generar imágenes en formato pgm correspondientes al conjunto de Julia y sus vecindades. Los parámetros para la generación de la imagen serán ingresados al llamar al programa. A continuación se explican brevemente los conceptos necesarios para llevar a cabo esta tarea:

2.1. Shell de Unix

Un shell es un programa intérprete de comandos. Los comandos utilizados en el presente trabajo son los siguientes:

2.2. Compilador GCC

GCC es un compilador gratuito, Open Source y multiplataforma que permite compilar código C. El mismo es un compilador estándar, es decir, que respeta la norma ISOC99 o ANSI dependiendo de su versión, por lo cual se utilizan las bibliotecas estándar para trabajar con el mismo. En esta asignatura el mismo es de gran utilidad dado que permite tener acceso al código *assembly* equivalente a nuestro programa en C. A continuación se detallan brevemente algunos parámetros que podemos darle al GCC:

- **-Wall:** Activa todos los warnings.
- **-o file** Permite cambiar el nombre del archivo compilado generado.
- **-O0:** Desactiva las optimizaciones, lo cual resulta especialmente útil, porque genera código fácilmente entendible en assembly en comparación al código C.
- **-O3:** Activa todas las optimizaciones, lo cual deseamos evitar, porque el código generado en assembly no se traduce de forma simple al código C correspondiente.
- **-g:** Genera código agregando información de debbugging.
- **-S:** Detiene el compilador luego de generar el código assembly.

- **-mrnames(solo para MIPS):** Indica al compilador que genera la salida utilizando nombre de registro en lugar de número de registro.

Para generar el archivo "main.s" con el código assembly se debe ejecutar el siguiente comando:

```
gcc -std=c99 -S -mrnames -O0 -g tp0.c -o tp0 -lm
```

Para compilar el código se debe ejecutar el siguiente comando:

```
gcc -std=c99 -O0 -g tp0.c -o tp0 -lm
```

2.3. Emulador GXEmul

Es un emulador gratuito y Open Source de la arquitectuca de computadores MIPS. Tiene como ventajas que puede correr algunos sistemas operativos sin modificaciones como netBSD, su portabilidad y velocidad de emulación dado que realiza la traducción binaria en tiempo real.

3. Diseño e implementación

Para la implementación del programa se creó el archivo:

- main.c: Contiene la implementación del software.

El programa utiliza los parámetros recibidos y comienza a realizar los calculos necesarios para generar la imagen del conjunto de Julia. Una vez que realiza los calculos necesarios genera el documento pgm en donde guarda la imagen.

3.1. Compilación

Para compilar el programa basta con ejecutar

```
gcc -g -lm tp0.c -o <nombre salida>
```

3.2. Modo de Operación

La aplicación desarrollada recibe los parametros deseados y los utiliza para la generación de la imagen. Los mismos se describen a continuación:

- `tp0 -h -V -c <a+bi> -H <float> -w <float> -o <out file>`
- **-V** Imprime la versión y finaliza.
- **-h** Imprime información y finaliza.
- **-c** Setea el centro de la imagen.

- **-H** Setea el alto del rectangulo. Valor por defecto=4.
- **-w** Setea el ancho del rectangulo. Valor por defecto=4.
- **-o** Setea el archivo de salida.

Para ejecutarlo se debe hacer:

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

o

```
./tp0 -c +0.282-0.01i -w 0.005 -H 0.005 -o dos.pgm
```

3.3. Ejecución de prueba

Primero, usamos la opcion -h para ver el mensaje de ayuda:

```
Usage:
  tp0 -h  -V -c <a+bi> -C <a+bi> -H <float> -w <float> -o <out_file> -
Options:
    -V     Imprime la version y finaliza.
    -h     Imprime esta informacion y finaliza.
    -c     Setea el centro de la imagen.
    -H     Setea el alto del rectangulo. Valor por defecto=4
    -w     Setea el ancho del rectangulo. Valor por defecto=4
    -o     Setea el archivo de salida
    -C     Setea la constante del algoritmo. Valor por defecto= 0.285+0.01i
Examples:
  tp0 -c +0.282-0.01i -w 0.005 -H 0.005 -o dos.pgm
```

Luego, hacemos una corrida de prueba con la linea:

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

Y el resultado de la ejecución fue:

Los resultados coinciden con los esperados.

4. Corridas de prueba

4.1. Pruebas

4.1.1. example.txt

```
%parametros invalidos y esas cosas
```

La salida es la esperada.

5. Código fuente

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #define ARG_ERR -1
6 typedef struct{
7   double real;
8   float imag;
9 }numcomplex;
10
11 void imprimir_complejo(numcomplex c){
12   printf("%f %+fi \n",c.real,c.imag);
13 }
14
15 double abs_cplx(numcomplex a){
16   return sqrt(pow(a.real,2) + pow(a.imag,2));
17 }
18
19 void sqr_cplx(numcomplex* a){
20   double aux = a->imag;
21   a->imag  = 2 * a->real * a->imag;
22   a->real=pow(a->real,2) - pow(aux,2);
23 }
24
25 void imprimir_error(int status){
26   switch(status){
27     case ARG_ERR:
28       printf("Debe ingresar correctamente los
             argumentos. Abortando ejecucion\n");
29       exit(ARG_ERR);
30       break;
31     default:
32       printf("Error no contemplado. Abortando Ejecucion
           \n");
33   }   exit(status);
34 }
35
36 int main(int argc, char *argv[])
37 {
38   int status = 0;
39   int alto = 640;
40   int ancho = 480;
41   numcomplex constant;
42   numcomplex center;
43   center.real=0;
44   center.imag=0;
45   constant.real=0.285;
```

```
46  constant.imag=-0.01;
47  char *auxc;
48  double H=4,w=4;
49  FILE *salida = stdout;
50
51  if (argc > 1 ) {
52    for(int i=1 ; i<argc ; i+=2){
53      if(argv[i][0] == '-'){
54
55        switch (argv[i][1]) {
56          case 'h': printf( "Usage:\n  tp0 -h  -V -c
                <a+bi> -C <a+bi> -H <float> -w <float> -o
                <out_file> -\n"
57            "Options:\n"
58            "    -V\t    Imprime la version y
                  finaliza.\n"
59            "    -h\t    Imprime esta informacion y
                  finaliza.\n"
60            "    -c\t    Setea el centro de la imagen.\n"
61            "    -H\t    Setea el alto del rectangulo.
                  Valor por defecto=4\n"
62            "    -w\t    Setea el ancho del rectangulo.
                  Valor por defecto=4\n"
63            "    -o\t    Setea el archivo de salida"
64            "    -C\t    Setea la constante del
                  algoritmo. Valor por defecto=
                  0.285+0.01i\n"
65            "Examples:\n  tp0 -c +0.282-0.01i -w 0.005
                -H 0.005 -o dos.pgm\n");
66          //texto con ayuda a completar//
67            return 0;
68            break;
69          case 'V': printf("Conjunto de Julia\nv1.0\n");
70            return 0;
71            break;
72          case 'r':
73            auxc=strtok(argv[i+1],"xX");
74            if (auxc == NULL){
75              status = ARG_ERR;
76              break;
77            }
78            ancho=atoi(auxc);
79            auxc=strtok(NULL, " ");
80            if (auxc == NULL) {
81              status = ARG_ERR;
82              break;
83            }
84            alto=atoi(auxc);
85            break;
86          case 'C':
```

```
87              auxc=argv[i+1];
88              if(auxc == NULL){
89                status=ARG_ERR;
90                break;
91              }
92              constant.real=atof(auxc);
93              if(argv[i+1][0]=='-' || argv[i+1][0]=='+'){
                          //si el primer numero tiene signo,
                    lo saltea
94                auxc=strpbrk(auxc+1,"-+");
95              }
96              else auxc=strpbrk(auxc,"-+");
97              constant.imag=atof(auxc);
98              break;
99            case 'c':
100
101             auxc=argv[i+1];
102             if (auxc == NULL){
103               status=ARG_ERR;
104               break;
105             }
106             center.real=atof(auxc);
                  //toma la parte real
107             if(argv[i+1][0]=='-' || argv[i+1][0]=='+'){
                          //si el primer numero tiene signo,
                    lo saltea
108               auxc=strpbrk(auxc+1,"-+");
109             }
110             else auxc=strpbrk(auxc,"-+");
111             center.imag=atof(auxc);
                  //toma la parte imaginaria
112             break;
113           case 'H':
114             if (argv[i+1] == NULL){
115               status=ARG_ERR;
116               break;
117             }
118             H=atof(argv[i+1]);
119             break;
120           case 'o':
121             if(argv[i+1] == NULL){
122               status=ARG_ERR;
123               break;
124             }
125             if(strcmp(argv[i+1],"-") == 0){
126               break;
127             };
128
129             salida = fopen(argv[i+1], "w");
130             break;
```

```
131             case 'w':
132               w=atof(argv[i+1]);
133               break;
134             default: printf("Argumento desconocido: prueba
                    con -h para ver la ayuda.\n");
135           }
136         }
137         else{
138           printf("Error! Formato desconocido. Prueba con
                  -h para ver la ayuda. \n");
139           return 1;
140           //return error de argumento
141         }
142       }
143     }
144     else{
145       printf("Se correra el programa con los valores por
             DEFAULT. \n");
146     }
147     if (status != 0) imprimir_error(status);
148     fputs("P2 \n",salida);
149     fputs("#TP0 Vecindades de Julia \n",salida);
150     char alto_str[10];
151     char ancho_str[10];
152     char concat[20];
153     sprintf(alto_str,"%d \n",alto);
154     sprintf(ancho_str,"%d",ancho);
155     strcpy(concat,ancho_str);
156     strcat(concat," ");
157     strcat(concat,alto_str);
158     fputs(concat,salida);
159     fputs("255 \n",salida);
160     int matrix_PGM[alto][ancho];
161     double aux_im;
162     int n;
163     for (int im=0; im<alto;im++){
164        aux_im= (H - (im*2*H/alto));
165        numcomplex zeta;
166       for(int re=0;re<ancho;re++){
167         zeta.real=(-w + (re*2*w/ancho)) - center.real;
168         zeta.imag=aux_im - center.imag;
169         while(abs_cplx(zeta) < 2 && n<255){
170           sqr_cplx(&zeta);
171           zeta.real+=constant.real;
172           zeta.imag+=constant.imag;
173           n++;
174         }
175         matrix_PGM[im][re]= n;
176         fprintf(salida,"%d ",matrix_PGM[im][re]);
177         n=0;
```

```
178         }
179         fprintf(salida,"\n");
180     }
181
182     return 0;
183 }
```

tp0.c

6. Código MIPS

aca va el codigo mips generado.

7. Conclusión

En el presente trabajo se aprendió a utilizar las herramientas que utilizaremos en los próximos trabajos prácticos. Se aprendió a utilizar el emulador GXEmul y compilar programas para arquitectura MIPS, tanto a forma binaria como assembly MIPS.