



U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica

Organización de computadoras 66-20

TRABAJO PRÁCTICO #0

Infraestructura básica

Curso: 2018 - 2do Cuatrimestre

Turno: Martes

GRUPO N°	
Integrantes	Padrón
Verón, Lucas	89341
Gamarra Silva, Cynthia Marlene	92702
Gatti, Nicolás	93570
Fecha de entrega:	18-09-2018
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

Índice	1
1. Enunciado del trabajo práctico	2
1.1. Diseño e implementación	6
1.2. Parámetros del programa	8
1.3. Compilación del programa	8
2. Pruebas realizadas	10
2.1. Pruebas con archivo bash test-automatic.sh	10
2.1.1. Generales	12
3. Conclusiones	14
Referencias	14
A. Código fuente	15
A.0.1. main.c	15
A.0.2. Assembly	17

1. Enunciado del trabajo práctico

66:20 Organización de Computadoras Trabajo práctico #0: Infraestructura básica 2^{do} cuatrimestre de 2018

\$Date: 2018/09/08 23:16:30 \$

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos 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.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 6), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo¹, y se valorarán aquellos escritos usando la herramienta T_EX / L^AT_EX.

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].

En la clase del 21/8 hemos repasado los pasos necesarios para la instalación y configuración del entorno de desarrollo.

¹<http://groups.yahoo.com/group/orga-comp>

5. Programa

Se trata de escribir, en lenguaje C, un programa para codificar y decodificar información en formato base 64: el programa recibirá, por línea de comando, los archivos o *streams* de entrada y salida, y la acción a realizar, codificar (acción por defecto) o decodificar. De no recibir los nombres de los archivos (o en caso de recibir - como nombre de archivo) usaremos los *streams* estándar, **stdin** y **stdout**, según corresponda. A continuación, iremos leyendo los datos de la entrada, generando la salida correspondiente. De ocurrir errores, usaremos **stderr**. Una vez agotados los datos de entrada, el programa debe finalizar adecuadamente, retornando al sistema operativo.

Estrictamente hablando, base 64 es un grupo de esquemas de codificación similares. En nuestra implementación, estaremos siguiendo particularmente el esquema establecido en [3], con el siguiente agregado: si se recibe una secuencia de caracteres inválida en la decodificación, debe asumirse como una condición de error que el programa deberá reportar adecuadamente y detener el procesamiento en ese punto.

5.1. Ejemplos

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

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V, --version      Print version and quit.
  -h, --help         Print this information.
  -i, --input         Location of the input file.
  -o, --output        Location of the output file.
  -a, --action        Program action: encode (default) or decode.
Examples:
  tp0 -a encode -i ~/input -o ~/output
  tp0 -a decode
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b64
$ ls -l /tmp/zero.txt.b64
-rw-r--r-- 1 user group 0 2018-09-08 16:21 /tmp/zero.txt.b64
```

Codificamos el carácter ASCII M,

```
$ echo -n M | tp0
TQ==
```

Codificamos los caracteres ASCII M y a,

```
$ echo -n Ma | tp0
TWE=
```

Codificamos `Man`,

```
$ echo -n Man | tp0
TWFu
```

Codificamos y decodificamos:

```
$ echo Man | tp0 | tp0 -a decode
Man
```

Verificamos bit a bit:

```
$ echo xyz | tp0 | tp0 -a decode | od -t c
0000000  x  y  z  \n
0000004
```

Codificamos 1024 bytes, para verificar que el programa genere líneas de no mas de 76 unidades de longitud:

```
$ yes | head -c 1024 | tp0 -a encode
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
...
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
```

Verificamos que la cantidad de bytes decodificados, sea 1024:

```
$ yes | head -c 1024 | tp0 -a encode | tp0 -a decode | wc -c
1024
```

Generamos archivos de tamaño creciente, y verificamos que el procesamiento de nuestro programa no altere los datos:

```
$ n=1;
$ while ;; do
>     head -c $n </dev/urandom >/tmp/in.bin;
>     tp0 -a encode -i /tmp/in.bin -o /tmp/out.b64;
>     tp0 -a decode -i /tmp/out.b64 -o /tmp/out.bin;
>     if diff /tmp/in.bin /tmp/out.bin; then ;; else
>         echo ERROR: $n;
>         break;
>     fi
>     echo ok: $n;
>     n="expr $n + 1";
>     rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
> done
ok: 1
ok: 2
ok: 3
...
```

6. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas);
- El código MIPS32 generado por el compilador;
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

Referencias

- [1] GXemul, <http://gavare.se/gxemul/>.
- [2] The NetBSD project, <http://www.netbsd.org/>.
- [3] RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies; sección 6.8, Base64 Content-Transfer-Encoding. <http://tools.ietf.org/html/rfc2045#section-6.8>.
- [4] Base64 (Wikipedia). <http://en.wikipedia.org/wiki/Base64>.

1.1. Diseño e implementación

El programa que contiene la lógica de codificador y decodificador se encuentra en el archivo *encode.c*. El codificador transforma expresiones con caracteres ASCII en base64, mientras que el decodificador hace el proceso inverso. Ambas funciones se implementan con diversos ciclos y recorridos, así como utilización de instrucciones aritméticas lógicas (sumas, &, shifts, etc) que relacionan la codificación base 64 con ASCII.

Se creó un archivo *file.c* para el manejo de los archivos a codificar/decodificar. Además se creó otro archivo llamado *command.c* cuya función es tomar los parámetros del programa y realizar las acciones pertinentes a la misma.

Específicamente el programa se estructura en los siguientes pasos:

- **Análisis de las parámetros de la línea de comandos:** se analizan las opciones ingresadas por la línea de comandos utilizando la función `getopt_long()`, la cual puede procesar cada opción que es leída de forma simplificada. Se extraen los argumentos de cada opción y se los guarda dentro de una estructura para su posterior acceso del tipo `CommandOptions` cuya definición es

```

1      typedef struct {
2          File input;
3          File output;
4          const char* input_route;
5          const char* output_route;
6          char error;
7          char encode_opt;
8      } CommandOptions;
9  
```

En caso de que no se encuentre alguna opción, se muestra el mensaje de ayuda al usuario para que identifique el prototipo de cómo debe ejecutar el programa.

- **Validación de opciones:** a medida que se va analizando cada opción de la línea de comandos, se valida cada una de ellas. Si se ingresó algún parámetro no válido para el programa o si se encontró un error se lo informa al usuario por pantalla y se aborta la ejecución del programa. Se utiliza para ello la función `CommandErrArg()` cuyo resultado es:

```

1      fprintf(stderr, "Invalid Arguments\n");
2      fprintf(stderr, "Options:\n");
3      fprintf(stderr, "  -V, --version    Print version and quit.\n");
4      fprintf(stderr, "  -h, --help      Print this information.\n");
5      fprintf(stderr, "  -i, --input      Location of the input file.\n");
6      fprintf(stderr, "  -o, --output     Location of the output file.\n");
7      fprintf(stderr, "  -a, --action     Program action: encode (default) or
decode.\n");
8      fprintf(stderr, "Examples:\n");
9      fprintf(stderr, "  tp0 -a encode -i ~/input -o ~/output\n");
10     fprintf(stderr, "  tp0 -a decode\n");
11  
```

Para el caso en que no hubo errores a la validación de los argumentos se procede a llamar a las funciones correspondientes a:

- **Mensaje de ayuda:** Función `CommandVersion()`
- **Mensaje de versión:** Función `CommandHelp()`
- **Input file :** Función `CommandSetInput()` que guarda la entrada del archivo donde será leído el texto.
- **Output file:** Función `CommandSetOutput()` que guarda la entrada del archivo de salida donde se escribirá el texto codificado.
- **Acción del programa a ejecutar:** Función `CommandSetEncodeOpt()` que setea la variable `opt— > encode_opt` indicando si es una operación de ENCODE o DECODE respectivamente.

- Encode/Decode: una vez que se procesó correctamente las opciones de la línea de comandos se procede a llamar a la función `_CommandEncodeDecode()` que ejecutará la operación de ENCODE o DECODE dependiendo del argumento pasado en la línea de comandos. La operación de DECODE se ejecuta el siguiente código:

```

1 while (!FileEofReached(&opt->input) && !CommandHasError(opt)) {
2     unsigned int read = FileRead(&opt->input, buf_encoded, 4);
3     if (read > 0) { // Solo es 0 si alcance el EOF
4         if (read != 4) { //Siempre debo leer 4 sino el formato es incorrecto
5             fprintf(stderr, "Longitud de archivo no es multiplo de 4\n");
6             CommandSetError(opt);
7         } else {
8             ++count;
9             if (count == 18) { // 19 * 4 = 76 bytes
10                unsigned char aux;
11                FileRead(&opt->input, &aux, 1);
12                count = 0;
13            }
14            if (Decode(buf_encoded, buf_decoded)) {
15                char aux = 0;
16                if (buf_encoded[2] == '=')
17                    ++aux;
18                if (buf_encoded[3] == '=')
19                    ++aux;
20
21                FileWrite(&opt->output, buf_decoded, 3 - aux);
22            } else {
23                fprintf(stderr, "Caracteres invalidos en archivo codificado: ");
24                unsigned int i;
25                for (i = 0; i < 4; ++i)
26                    fprintf(stderr, "%c", buf_encoded[i]);
27                CommandSetError(opt);
28            }
29        }
30    }
31 }
32
33 }
34

```

Básicamente lo que se realiza es la lectura del archivo para procesarlo teniendo en cuenta la longitud del archivo a procesar y el padding a decodificar. La función `Decode()` retorna un buffer de 3 caracteres con el decode de 4 caracteres en base64. Se debe cumplir:

* Pre: el buffer input contiene 4 caracteres. El buffer output tiene por lo menos 3 caracteres * Post: retorna un buffer de 3 byte con los caracteres en ASCII. retorna 0 si error 1 si ok.

La operación de ENCODE se ejecuta el siguiente código:

```

1 while (!FileEofReached(&opt->input)){
2     memset(buf_decoded, 0, 3);
3     unsigned int read = FileRead(&opt->input, buf_decoded, 3);
4     if (read > 0) {
5         Encode(buf_decoded, read, buf_encoded);
6         FileWrite(&opt->output, buf_encoded, 4);
7         ++count;
8         if (count == 18) { // 19 * 4 = 76 bytes
9             FileWrite(&opt->output, (unsigned char *) "\n", 1);
10            count = 0;
11        }
12    }
13 }
14

```


Básicamente lo que se realiza es la lectura del archivo para procesarlo en la función `Encode()` en donde recibe 3 caracteres en buffer y los convierte en 4 caracteres codificados en output. Se debe cumplir:

* Pre: el buffer contiene length caracteres (1 a 3) y todos los caracteres son validos * Post: retorna un buffer de 4 byte con los caracteres en base64.

1.2. Parámetros del programa

Se detallan a continuación los parámetros del programa

- -h: Visualiza la ayuda del programa, en la que se indican los parámetros y sus objetivos.
- -V: Indica la versión del programa.
- -i: Archivo de entrada del programa.
- -o: Archivo de salida del programa.
- -a: Acción a llevar a cabo: codificación o decodificación.

Se indica a continuación detalles respecto a los parámetros:

- Si no se explicitan -i y -o, se utilizarán stdin y stdout, respectivamente.
- -V es una opción “show and quit”. Si se explicita este parámetro, sólo se imprimirá la versión, aunque el resto de los parámetros se hayan explicitado.
- -h también es de tipo “show and quit” y se comporta de forma similar a -V.
- en caso de que se use la entrada estándar (con comando `echo texto | ./tp0 -a encode`) y luego se especifique un archivo de salida con -i, prevalecerá el establecido por parámetro.

1.3. Compilación del programa

Para obtener un ejecutable, se creó un archivo `makefile` cuyo contenido es:

```

1 CC = gcc
2 CFLAGS = -o0 -g -Wall -Werror -pedantic -std=c99
3
4 OBJECTS = command.o encode.o file.o
5 EXEC = tp0
6
7 VALGRIND = valgrind --track-origins=yes --leak-check=full
8 VALGRIND-V = $(VALGRIND) -v
9
10 all: $(EXEC)
11
12 command.o: command.c command.h
13     $(CC) $(CFLAGS) -c command.c -o command.o
14 encode.o: encode.c encode.h
15     $(CC) $(CFLAGS) -c encode.c -o encode.o
16 file.o: file.c file.h
17     $(CC) $(CFLAGS) -c file.c -o file.o
18
19 $(EXEC): $(OBJECTS)
20     $(CC) $(CFLAGS) $(OBJECTS) main.c -o $(EXEC) -lm
21
22 run: $(EXEC)
23     ./$$(EXEC)
24
25 valgrind: $(EXEC)
26     $(VALGRIND) ./$$(EXEC)

```

```
27
28 valgrind-verb: $(EXEC)
29     $(VALGRIND-V) ./$$(EXEC)
30
31 clean:
32     rm -f *.o $(EXEC)
33
```

Para ejecutarlo, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
1 $ make
```

Para proceder a la ejecución del programa, se debe llamar a:

```
1 $ ./tp0
```

seguido de los parámetros que se desee modificar, los cuales se indicaron en la sección 1.2.

En caso de ser entrada estándar (stdin) se podrá ejecutar de la siguiente forma:

```
1 $ echo texto | ./tp0 -a encode
```

También en este caso, se indican a continuación los parámetros a usar.

Para el caso de hacerlo en el emulador GXemul que provee la cátedra, utilizando la máquina virtual que contiene el sistema operativo NetBSD, no se utilizó el archivo Makefile, la compilación se realizó con la herramienta `gcc`.

2. Pruebas realizadas

2.1. Pruebas con archivo bash test-automatic.sh

Para la ejecución del siguiente script se debe copiar, se debe ubicar el archivo ejecutable compilado dentro de la carpeta de test para que se ejecuten correctamente las pruebas. El script sería:

```

1  #!/bin/bash
2
3  echo "#####"
4  echo "##### Tests automaticos #####"
5  echo "#####"
6
7  mkdir ./outputs
8
9  echo "#-----# COMIENZA test ejercicio 0 archivo vacio #-----#"
10 touch ./outputs-aut/zero.txt
11 ./tp0 -a encode -i ./outputs-aut/zero.txt -o ./outputs-aut/zero.txt.b64
12 ls -l ./outputs-aut/zero.txt.b64
13
14 if diff -b ./outputs-aut/zero.txt ./outputs-aut/zero_ok.txt; then
15     echo "[OK]";
16 else echo ERROR;
17 fi
18
19 echo "#-----# FIN test ejercicio 0 archivo vacio #-----#"
20 echo "#-----#"
21 echo "#-----# COMIENZA test ejercicio 1 archivo vacio sin -a #-----#"
22
23 touch ./outputs-aut/zero.txt
24 ./tp0 -i ./outputs-aut/zero.txt -o ./outputs-aut/zero.txt.b64
25 ls -l ./outputs-aut/zero.txt.b64
26
27 if diff -b ./outputs-aut/zero.txt ./outputs-aut/zero_ok.txt; then
28     echo "[OK]";
29 else echo ERROR;
30 fi
31
32 echo "#-----# FIN test ejercicio 1 archivo vacio sin -a #-----#"
33 echo "#-----#"
34 echo "#-----# COMIENZA test ejercicio 2 stdin y stdout #-----#"
35
36 echo -n Man | ./tp0 -a encode > ./outputs/outputEncode.txt
37 if diff -b ./outputs-aut/outputEncode-aut.txt ./outputs/outputEncode.txt; then echo "[OK]";
38     else
39         echo ERROR;
40 fi
41
42 echo "#-----# FIN test ejercicio 2 stdin y stdout #-----#"
43 echo "#-----#"
44 echo "#-----# COMIENZA test ejercicio 3 stdin y stdout #-----#"
45
46 echo -n TWFu | ./tp0 -a decode > ./outputs/outputDecode.txt
47 if diff -b ./outputs-aut/outputDecode-aut.txt ./outputs/outputDecode.txt; then echo "[OK]";
48     else
49         echo ERROR;
50 fi
51
52 echo "#-----# FIN test ejercicio 3 stdin y stdout #-----#"
53 echo "#-----#"
54 echo "#-----# COMIENZA test ejercicio 3 help sin parámetros #-----#"
55
56 ./tp0 > ./outputs/outputMenuHelp.txt
57 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuH.txt; then echo "[OK]";
58     else
59         echo ERROR;

```

```

57 fi
58
59 echo "#-----# FIN test ejercicio 3 help sin parámetros #-----#"
60 echo "#-----#"
61 echo "#-----# COMIENZA test menu help (-h) #-----#"
62
63 ./tp0 -h > ./outputs/outputMenuH.txt
64
65 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuH.txt; then echo "[OK]";
66     else
67         echo ERROR;
68 fi
69
70 echo "#-----# FIN test menu version (-h) #-----#"
71 echo "#-----#"
72 echo "#-----# COMIENZA test menu help (--help) #-----#"
73
74 ./tp0 --help > ./outputs/outputMenuHelp.txt
75
76 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuHelp.txt; then echo "[OK
77     ]"; else
78         echo ERROR;
79 fi
80
81 echo "#-----# FIN test menu version (--help) #-----#"
82 echo "#-----#"
83 echo "#-----# COMIENZA test menu version (-V) #-----#"
84
85 ./tp0 -V > ./outputs/outputMenuV.txt
86
87 if diff -b ./outputs-aut/outputMenuVersion-aut.txt ./outputs/outputMenuV.txt; then echo "[OK
88     ]"; else
89         echo ERROR;
90 fi
91
92 echo "#-----# FIN test menu version (-V) #-----#"
93 echo "#-----#"
94 echo "#-----# COMIENZA test menu version (--version) #-----#"
95
96 ./tp0 --version > ./outputs/outputMenuVersion.txt
97
98 if diff -b ./outputs-aut/outputMenuVersion-aut.txt ./outputs/outputMenuVersion.txt; then echo
99     "[OK]"; else
100         echo ERROR;
101 fi
102
103 echo "#-----# FIN test menu version (--version) #-----#"
104 echo "#-----#"
105 echo "#-----# COMIENZA test ejercicio encode/decode #-----#"
106
107 echo xyz | ./tp0 -a encode | ./tp0 -a decode | od -t c
108
109 echo "#-----# FIN test ejercicio encode #-----#"
110 echo "#-----#"
111 echo "#-----# COMIENZA test ejercicio longitud maxima 76 #-----#"
112
113 yes | head -c 1024 | ./tp0 -a encode > ./outputs/outputSize76.txt
114
115 if diff -b ./outputs-aut/outputSize76-aut.txt ./outputs/outputSize76.txt; then echo "[OK]";
116     else
117         echo ERROR;
118 fi
119
120 echo "#-----# FIN test ejercicio longitud maxima 76 #-----#"
121 echo "#-----#"
122 echo "#-----# COMIENZA test ejercicio decode 1024 #-----#"
123
124 yes | head -c 1024 | ./tp0 -a encode | ./tp0 -a decode | wc -c > ./outputs/outputSize1024.txt
125

```

```

119 if diff -b ./outputs-aut/outputSize1024-aut.txt ./outputs/outputSize1024.txt; then echo "[OK
    ]"; else
120     echo ERROR;
121 fi
122
123 echo "#-----# FIN test ejercicio decode 1024#-----#"
124 echo "#-----#"
125 echo "#-----# COMIENZA test ejercicio encode/decode random #-----#"
126
127 n=1;
128 while ;; do
129     #while [$n -lt 10]; do
130     head -c $n </dev/urandom >/tmp/in.bin;
131     ./tp0 -a encode -i /tmp/in.bin -o /tmp/out.b64;
132     ./tp0 -a decode -i /tmp/out.b64 -o /tmp/out.bin;
133     if diff /tmp/in.bin /tmp/out.bin; then ;; else
134     echo ERROR: $n;
135     break;
136     fi
137     echo [OK]: $n;
138     n=`expr $n + 1`;
139     rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
140 done
141
142 echo "#-----# FIN test ejercicio encode/decode random #-----#"
143 echo "#-----#"
144
145 echo "#####"
146 echo "##### FIN Tests automaticos #####"
147 echo "#####"

```

El cual no presenta errores en ninguna de las corridas llevadas a cabo.

Todas las pruebas que se presentan a continuación, están codificadas en los archivos de prueba `***.txt` de forma que puedan ejecutarse y comprobar los resultados obtenidos.

Se indicaran a continuación lo siguiente: comandos para ejecutarlas, líneas de código que las componen y resultado esperado.

2.1.1. Generales

■ Mensaje de ayuda

```

1 $ ./tp0 -h o ./tp0 --help
2
3 Options:
4   -V, --version      Print version and quit.
5   -h, --help         Print this information.
6   -i, --input         Location of the input file.
7   -o, --output        Location of the output file.
8   -a, --action        Program action: encode (default) or decode.
9 Examples:
10  tp0 -a encode -i ~/input -o ~/output
11  tp0 -a decode

```

■ Mensaje de version

```

1 $ ./tp0 -V o ./tp0 --version
2 Version: 0.1
3

```

■ Archivo de entrada no válido

```
1 $ ./tp0 -i archivoInvalido.txt
2
3 Invalid Arguments
4 Options:
5   -V, --version      Print version and quit.
6   -h, --help         Print this information.
7   -i, --input        Location of the input file.
8   -o, --output       Location of the output file.
9   -a, --action       Program action: encode (default) or decode.
10 Examples:
11   tp0 -a encode -i ~/input -o ~/output
12   tp0 -a decode
13
14
15
```

3. Conclusiones

El trabajo práctico nos permitió desarrollar una API para procesar archivos transformándolos a su equivalente `base64` en lenguaje C. Además, nos permitió familiarizarnos con la arquitectura MIPS32 y el consecuente análisis y desarrollo de código assembler MIPS utilizando el emulador GXemul.

Referencias

- [1] Base64 (Wikipedia) <http://en.wikipedia.org/wiki/Base64>
- [2] The NetBSD project, <http://www.netbsd.org/>
- [3] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.
- [4] *GNU Make* - <https://www.gnu.org/software/make/>
- [5] *Valgrind* - <http://valgrind.org/>

A. Código fuente

A.0.1. main.c

```

1  /**
2  * Created by gatti2602 on 12/09/18.
3  * Main
4  */
5
6  #define ERROR 0
7  #define FALSE 0
8  #define TRUE 1
9
10 #include <getopt.h>
11 #include "command.h"
12
13 int main(int argc, char** argv) {
14     struct option arg_long[] = {
15         {"input", required_argument, NULL, 'i'},
16         {"output", required_argument, NULL, 'o'},
17         {"action", required_argument, NULL, 'a'},
18         {"help", no_argument, NULL, 'h'},
19         {"version", no_argument, NULL, 'V'},
20     };
21     char arg_opt_str[] = "i:o:a:hV";
22     int arg_opt;
23     int arg_opt_idx = 0;
24     char should_finish = FALSE;
25
26     CommandOptions cmd_opt;
27     CommandCreate(&cmd_opt);
28
29     if(argc == 1)
30         CommandSetError(&cmd_opt);
31
32     while((arg_opt =
33         getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1 && !
34         should_finish) {
35         switch(arg_opt){
36             case 'i':
37                 CommandSetInput(&cmd_opt, optarg);
38                 break;
39             case 'o':
40                 CommandSetOutput(&cmd_opt, optarg);
41                 break;
42             case 'h':
43                 CommandHelp();
44                 should_finish = TRUE;
45                 break;
46             case 'V':
47                 CommandVersion();
48                 should_finish = TRUE;
49                 break;
50             case 'a':
51                 CommandSetEncodeOpt(&cmd_opt, optarg);
52                 break;
53             default:
54                 CommandSetError(&cmd_opt);
55                 break;
56         }
57     }
58
59     if(should_finish)
60         return 0;
61
62     if(!CommandHasError(&cmd_opt)) {

```



```
62     CommandProcess(&cmd_opt);
63 } else {
64     CommandErrArg();
65     return 1;
66 }
67 return 0;
68 }
```

A.0.2. Assembly

```

1      .file      1 "main.c"
2      .section   .mdebug.abi32
3      .previous
4      .abicalls
5      .rdata
6      .align     2
7  $LC0:
8      .ascii     "input\000"
9      .align     2
10  $LC1:
11     .ascii     "output\000"
12     .align     2
13  $LC2:
14     .ascii     "action\000"
15     .align     2
16  $LC3:
17     .ascii     "help\000"
18     .align     2
19  $LC4:
20     .ascii     "version\000"
21     .data
22     .align     2
23  $LC5:
24     .word      $LC0
25     .word      1
26     .word      0
27     .word      105
28     .word      $LC1
29     .word      1
30     .word      0
31     .word      111
32     .word      $LC2
33     .word      1
34     .word      0
35     .word      97
36     .word      $LC3
37     .word      0
38     .word      0
39     .word      104
40     .word      $LC4
41     .word      0
42     .word      0
43     .word      86
44     .globl     memcpy
45     .rdata
46     .align     2
47  $LC6:
48     .ascii     "i:o:a:hV\000"
49     .text
50     .align     2
51     .globl     main
52     .ent       main
53  main:
54     .frame      $fp,200,$ra          # vars= 152, regs= 3/0, args= 24, extra= 8
55     .mask       0xd0000000,-8
56     .fmask      0x00000000,0
57     .set        noreorder
58     .cpload     $t9
59     .set        reorder
60     subu        $sp,$sp,200
61     .cprestore  24
62     sw          $ra,192($sp)
63     sw          $fp,188($sp)
64     sw          $gp,184($sp)
65     move        $fp,$sp

```

```

66      sw      $a0,200($fp)
67      sw      $a1,204($fp)
68      addu    $v0,$fp,32
69      la      $v1,$LC5
70      move    $a0,$v0
71      move    $a1,$v1
72      li      $a2,80                # 0x50
73      la      $t9,memcpy
74      jal     $ra,$t9
75      lw      $v0,$LC6
76      sw      $v0,112($fp)
77      lw      $v0,$LC6+4
78      sw      $v0,116($fp)
79      lbu     $v0,$LC6+8
80      sb      $v0,120($fp)
81      sw      $zero,132($fp)
82      sb      $zero,136($fp)
83      addu    $v0,$fp,144
84      move    $a0,$v0
85      la      $t9,CommandCreate
86      jal     $ra,$t9
87      lw      $v1,200($fp)
88      li      $v0,1                # 0x1
89      bne     $v1,$v0,$L18
90      addu    $v0,$fp,144
91      move    $a0,$v0
92      la      $t9,CommandSetError
93      jal     $ra,$t9
94 $L18:
95      .set     noreorder
96      nop
97      .set     reorder
98 $L19:
99      addu    $v1,$fp,112
100     addu    $v0,$fp,132
101     sw      $v0,16($sp)
102     lw      $a0,200($fp)
103     lw      $a1,204($fp)
104     move    $a2,$v1
105     addu    $a3,$fp,32
106     la      $t9,getopt_long
107     jal     $ra,$t9
108     sw      $v0,128($fp)
109     lw      $v1,128($fp)
110     li      $v0,-1                # 0xffffffffffffffff
111     beq     $v1,$v0,$L20
112     lb      $v0,136($fp)
113     bne     $v0,$zero,$L20
114     lw      $v0,128($fp)
115     addu    $v0,$v0,-86
116     sw      $v0,180($fp)
117     lw      $v1,180($fp)
118     sltu    $v0,$v1,26
119     beq     $v0,$zero,$L29
120     lw      $v0,180($fp)
121     sll     $v1,$v0,2
122     la      $v0,$L30
123     addu    $v0,$v1,$v0
124     lw      $v0,0($v0)
125     .cpadd   $v0
126     j       $v0
127     .rdata
128     .align   2
129 $L30:
130     .gpword  $L27
131     .gpword  $L29
132     .gpword  $L29

```

```

133      .gpword $L29
134      .gpword $L29
135      .gpword $L29
136      .gpword $L29
137      .gpword $L29
138      .gpword $L29
139      .gpword $L29
140      .gpword $L29
141      .gpword $L28
142      .gpword $L29
143      .gpword $L29
144      .gpword $L29
145      .gpword $L29
146      .gpword $L29
147      .gpword $L29
148      .gpword $L26
149      .gpword $L24
150      .gpword $L29
151      .gpword $L29
152      .gpword $L29
153      .gpword $L29
154      .gpword $L29
155      .gpword $L25
156      .text
157  $L24:
158      addu    $v0,$fp,144
159      move    $a0,$v0
160      lw      $a1,optarg
161      la      $t9,CommandSetInput
162      jal     $ra,$t9
163      b       $L19
164  $L25:
165      addu    $v0,$fp,144
166      move    $a0,$v0
167      lw      $a1,optarg
168      la      $t9,CommandSetOutput
169      jal     $ra,$t9
170      b       $L19
171  $L26:
172      la      $t9,CommandHelp
173      jal     $ra,$t9
174      li      $v0,1                # 0x1
175      sb      $v0,136($fp)
176      b       $L19
177  $L27:
178      la      $t9,CommandVersion
179      jal     $ra,$t9
180      li      $v0,1                # 0x1
181      sb      $v0,136($fp)
182      b       $L19
183  $L28:
184      addu    $v0,$fp,144
185      move    $a0,$v0
186      lw      $a1,optarg
187      la      $t9,CommandSetEncodeOpt
188      jal     $ra,$t9
189      b       $L19
190  $L29:
191      addu    $v0,$fp,144
192      move    $a0,$v0
193      la      $t9,CommandSetError
194      jal     $ra,$t9
195      b       $L19
196  $L20:
197      lb      $v0,136($fp)
198      beq     $v0,$zero,$L31
199      sw      $zero,176($fp)

```

```
200      b      $L17
201 $L31:
202      addu    $v0,$fp,144
203      move    $a0,$v0
204      la      $t9,CommandHasError
205      jal     $ra,$t9
206      bne     $v0,$zero,$L32
207      addu    $v0,$fp,144
208      move    $a0,$v0
209      la      $t9,CommandProcess
210      jal     $ra,$t9
211      b      $L33
212 $L32:
213      la      $t9,CommandErrArg
214      jal     $ra,$t9
215      li      $v0,1                # 0x1
216      sw      $v0,176($fp)
217      b      $L17
218 $L33:
219      sw      $zero,176($fp)
220 $L17:
221      lw      $v0,176($fp)
222      move    $sp,$fp
223      lw      $ra,192($sp)
224      lw      $fp,188($sp)
225      addu    $sp,$sp,200
226      j      $ra
227      .end    main
228      .size   main,.-main
229      .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```