Contenido

1 Introducción

El presente informe corresponde al primer trabajo práctico grupal "TP0" de la materia organización de computadoras. A continuación la presentación del enunciado y luego la resolución del mismo.

2 Documentación del diseño e implementación

En el archivo main.c se usa a getopt para analizar los argumentos dados en la línea de comando y en codec.c se encuentra el código de codificación y decodificación.

2.1 Menú (main.c)

En main.c se puede ver la implementación de un menú de ayuda, que posee una opción que ejecuta el programa en modo información "-h" que detalla las variantes de ejecución del programa, estas son -v que imprime la versión del programa y sale del mismo y otra es [options] que muestra las opciones disponibles; estas son -V, -h, -i que está disponible para indicar en donde se encuentra la dirección del archivo de entrada, -o lo mismo pero para el archivo de salida, -a que indica la acción que queremos que ejecute el programa, por default será encode y si quisiéramos que decodifique solo hace falta agregarle un decode.

Si se especifica la opción de codificar o decodificar entonces se podrá también especificar el archivo de entrada y de salida, se le podrá pasar un archivo especifico llamando a la función o también podrá pasárselo desde consola (Stdin y Stdout).

2.2 Codificación

La codificación se hace posible con la agrupación de casos que conforman la solución, para esto se dividen los bytes en bits y se agrupan dependiendo de la llegada de los mismos; a cada caso le corresponde un respectiva máscara, un respectivo delta de desplazamiento, y los bits faltantes para completar con el signo "=". Los casos están ordenados, entonces con un bucle se irá codificando byte a byte. Esta codificación, agarra el caso y sus parámetros, busca en la tabla de base 64 y traduce los bits en el carácter correspondiente. Finalmente, de ser necesario, completa la salida con caracteres de relleno.

2.3 Decodificación

Se incluye una tabla de decodificación generada con la función crear_tabla_de_decodificacion (incluida en el código), con ésta se podrán traducir los caracteres a decodificar a su índice dentro de la tabla de codificación, para de este modo poder concatenarlos y obtener la salida. Se tendrá en cuenta los caracteres inválidos, como por ejemplo '.', '!', etc o un '=' que no se encuentre en el lugar de caracter de relleno. La decodificación se hará de a bloques de 4 bytes, aunque en principio se leen de a 5 bytes para estar seguros de que en los primeros 4 no habrá caracteres de relleno. El programa finaliza decodificando el final de la entrada, verificando que el número de caracteres de relleno sea el correcto.

3 Pruebas

3.1 Código de pruebas.sh

```
#! /bin/sh
```

```
#Codificamos un archivo vacío (cantidad de bytes nula)
touch /tmp/zero.txt #creamos un archivo de texto vacio
./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 caracter ASCII M
echo -n M | ./tp0
#TQ==
echo

#codificamos caracter ASCII M y a
echo -n Ma | ./tp0
#TWE=
echo

echo "Test Codifico y decodifico una imagen. Prueba de binarios"
./tp0 -a encode -i recursos/linux-icon.png | ./tp0 -a decode -o recursos/linux-icon.png.b
      diff -s recursos/linux-icon.png recursos/linux-icon.png.b64

#codificamos Man
echo -n "Man" | ./tp0
#TWFu
echo

#codificamos y decodificamos
echo Man | ./tp0 | ./tp0 -a decode
#Man
echo

#verificamos bit a bit
echo xyz | ./tp0 | ./tp0 -a decode | od -t c
#0000000 x y z \n
#0000004
echo
yes | head -c 1024 | ./tp0 -a encode #codificamos 1024 bytes, comprobamos longitud
#eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
#...
#eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
echo

#verificamos que los bytes sean 1024
yes | head -c 1024 | ./tp0 -a encode | ./tp0 -a decode | wc -c
#1024
echo

#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=\$((\$n+1));

rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
done
```

3.2 Salida de pruebas.sh

```
-rw-r--r-- 1 sebastian sebastian 0 sep 11 16:00 /tmp/zero.txt.b64
TQ==
TWE=
Test Codifico y decodifico una imagen. Prueba de binarios
Los archivos recursos/linux-icon.png y recursos/linux-icon.png.b64 son idénticos
TWFu
Man

0000000   x   y   z  \n
0000004
```


```
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
```

```
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
1024

ok: 1
ok: 2
ok: 3
ok: 4
ok: 5
[...]
```

4 Código en C

4.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "codec.h"

#define ERROR_CODIFICANDO  2
#define ERROR_DECODIFICANDO  3


static char* ENCODE = "encode";
static char* DECODE = "decode";

typedef struct {
       char* accion;
       char* entrada;
       char* salida;
} Parametro;


Parametro manejarArgumentosEntrada(int argc, char** argv)
{
int siguiente_opcion;
int option_index;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hva:i:o:"; /* "hva::i:o:" */

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largas[] =
{
       { "help",     no_argument,   0,  'h'},
```

```
       { "version",    no_argument,   0,  'V'},
{ "action",     required_argument,  0,  'a'}, /*optional_argument*/
{ "input",      required_argument,  0,  'i'},
{ "output",     required_argument,  0,  'o'},
       { 0, 0, 0, 0 }
    };


Parametro parametro;
parametro.accion  = ENCODE;
parametro.entrada  = "";
parametro.salida  = "";

    while (1)
    {
        siguiente_opcion = getopt_long (argc, argv, op_cortas, op_largas, &option_index);

        if (siguiente_opcion == -1)
         break;

        switch (siguiente_opcion)
        {
            case 'h' :

                printf("Usage:\n");
                printf("\ttp0 -h\n");
                printf("\ttp0 -V\n");
                printf("\ttp0 [ options ]\n");

                printf("Options:\n");
                printf("\t-V, --version       Print version and quit.\n");
                printf("\t-h, --help          Print this information.\n");
                printf("\t-i, --input         Location of the input file.\n");
                printf("\t-o, --output        Location of the output file.\n");
                printf("\t-a, --action        Program action: encode (default) or decode.

                printf("Examples:\n");
                printf("\ttp0 -a encode -i ~/input -o ~/output\n");
                printf("\ttp0 -a encode\n");
                exit(0);
            break;

            case 'v' :
                printf("Tp0:Version_0.1:Grupo: Bárbara Mesones Miret, Nestor Huallpa, Seb
                exit(0);
             break;

            case 'a' :
                if ( optarg )
                parametro.accion = optarg;
```

```
            break;

           case 'i' :
               if ( optarg )
parametro.entrada = optarg;
            break;

           case 'o' :
               if ( optarg )
parametro.salida = optarg;
            break;
        }
    }

    return parametro;

}

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

int returnCode = 0;

Parametro p = manejarArgumentosEntrada(argc, argv);

int isEntradaArchivo = strcmp(p.entrada,"");
int isSalidaArchivo = strcmp(p.salida,"");

//Si la entrada esta vacia lee stdin (teclado)
FILE* archivoEntrada = (isEntradaArchivo!=0)?fopen(p.entrada, "rb"):stdin;
//Si la salida esta vacia escribe stdout (pantalla)
FILE* archivoSalida = (isSalidaArchivo!=0)?fopen ( p.salida, "w" ):stdout;

if (strcmp(p.accion, ENCODE) == 0) {
codificar (archivoEntrada, archivoSalida);
} else if (strcmp(p.accion, DECODE) == 0) {
decodificar (archivoEntrada, archivoSalida);
} else {
fprintf(stderr, "ERROR: SE DEBE INGRESAR UN ARGUMENTO CORRECTO PARA LA OPCION i.\n");
}

if (isEntradaArchivo!=0) {
fclose(archivoEntrada);
}
if (isSalidaArchivo!=0) {
fclose(archivoSalida);
}
if(returnCode!=0){
  exit(1);
}
return returnCode;
```

```
}
```

4.2 codec.h

```
#ifndef CODEC_H
#define CODEC_H

#include <stdio.h>

void codificar (FILE *entrada, FILE *salida);
void decodificar (FILE *entrada, FILE *salida);

#endif
```

4.3 codec.c

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include "codec.h"

void escribir (int c, FILE *salida)
{
if (fputc (c, salida) == EOF) {
fprintf (stderr, "%s\n", strerror (errno));
exit (1);
}
}

/*
 * Ejemplo de entrada dividida en bloques de 6 bits: xxxxxx00 0000xxxx xx000000 xxxxxx00
 *
 * |     mascaras | deltas | Faltan (=)
 * caso | previo   actual | <<  >> |
 * 0 | 00000000 xxxxxx00 00 fc |  0  2  | 2
 * 1 | 000000xx xxxx0000 03 f0 |  4  4  | 1
 * 2 | 0000xxxx xx000000 0f c0 |  2  6  | 0
 *                     |
 * 3 | 00000000 00xxxxxx 00 3f |  0  0  | 0
 * luego repite
 */

void codificar (FILE *entrada, FILE *salida)
{
const char *tabla = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
const int previo = 0;
const int actual = 1;
const int mascara[2][4] = {{0x00, 0x03, 0x0f, 0x00}, {0xfc, 0xf0, 0xc0, 0x3f}};
const int delta[2][4] = {{0, 4, 2, 0}, {2, 4, 6, 0}};
const int faltantes[3] = {0, 2, 1};
```

```
int b[2] = {0};
int caso = 0;
while ((b[actual] = fgetc (entrada)) != EOF) {
int i = ((b[previo] & mascara[previo][caso]) << delta[previo][caso]) |
((b[actual] & mascara[actual][caso]) >> delta[actual][caso]);
escribir (tabla[i], salida);
caso++;
if (caso == 3) {
i = (b[actual] & mascara[actual][caso]);
escribir (tabla[i], salida);
caso = 0;
}
b[previo] = b[actual];
}
if (faltantes[caso] == 1) {
int i = (b[previo] & 0x0f) << 2;
escribir (tabla[i], salida);
} else if (faltantes[caso] == 2) {
int i = (b[previo] & 0x03) << 4;
escribir (tabla[i], salida);
}
for (int i = 0; i < faltantes[caso]; i++) {
escribir ('=', salida);
}
}

void crear_tabla_de_decodificacion ()
{
const char *tabla_codificacion = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012

const int largo = 256;
char tabla[largo];
for (int i = 0; i < largo; i++) {
tabla[i] = -1;
}
unsigned char c;
int i = 0;
while ((c = tabla_codificacion[i]) != '\0') {
tabla[c] = i++;
}
for (int i = 0; i < largo; i++) {
if ( i % 16 == 0 ) fprintf (stderr, "\n");
fprintf (stderr, "0x%02x, ", tabla[i]);
}
fprintf (stderr, "\n");
}

void resolver (char *b, int largo)
{
const char tabla[] = {
```

```
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3e, 0xff, 0xff, 0xff,
0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d,
0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0xff, 0xff, 0xff, 0xff,
0xff, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, 0x30, 0x31, 0x32, 0x33, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
};
for (int i = 0; i < largo; i++) {
unsigned char c = b[i];
if (b[i] == -1) {
fprintf (stderr, "Error: Caracter '%c' ilegal en la entrada\n", c);
exit (1);
}
b[i] = tabla[c];
}
}

void decodificar_finales (char * b, int finales, FILE *salida)
{
if (finales == 1) {
resolver (b, 2);
escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
} else if (finales == 2) {
resolver (b, 3);
escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);

} else if (finales == 3) {
resolver (b, 4);
escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);
escribir (((b[2] & 0x03) << 6) | (b[3] & 0x3f), salida);
}
}

void decodificar (FILE *entrada, FILE *salida)
{
char b[5];
int leidos;
if ((leidos = fread (b, 1, 5, entrada)) == 5) {
```

```
do {
resolver (b, 4);
escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);
escribir (((b[2] & 0x03) << 6) | (b[3] & 0x3f), salida);
b[0] = b[4];
} while ((leidos = fread (b+1, 1, 4, entrada)) == 4);
leidos += 1;
}
if (leidos) {
int finales ;
if (leidos == 4) {
finales = b[3] == '=' ? (b[2] == '=' ? 1 : 2) : 3;
} else if (leidos == 3) {
fprintf (stderr, "Error: Faltan 1 caracter de relleno\n");
finales = b[2] == '=' ? 1 : 2;
} else if (leidos == 2) {
fprintf (stderr, "Error: Faltan 2 caracteres de relleno\n");
finales = 1;
} else {
fprintf (stderr, "Error: Faltan 2 caracteres de relleno\n");
fprintf (stderr, "Error: Faltan 1 caracter de informacion\n");
return ;
}
decodificar_finales (b, finales, salida);
}
}
```

5 Codigo de ensamblado MIPS32

5.1 main.s

```
.file 1 "main.c"
.section .mdebug.abi32
.previous
.abicalls
.rdata
.align 2
$LC0:
.ascii "encode\000"
.data
.align 2
.type ENCODE, @object
.size ENCODE, 4
ENCODE:
.word $LC0
.rdata
.align 2
$LC1:
.ascii "decode\000"
```

```
.data
.align 2
.type DECODE, @object
.size DECODE, 4
DECODE:
.word $LC1
.rdata
.align 2
$LC3:
.ascii "help\000"
.align 2
$LC4:
.ascii "version\000"
.align 2
$LC5:
.ascii "action\000"
.align 2
$LC6:
.ascii "input\000"
.align 2
$LC7:
.ascii "output\000"
.data
.align 2
$LC8:
.word $LC3
.word 0
.word 0
.word 104
.word $LC4
.word 0
.word 0
.word 86
.word $LC5
```

5.2 codec.s

```
.file 1 "codec.c"
.section .mdebug.abi32
.previous
.abicalls
.rdata
.align 2
$LC0:
.ascii "%s\n\000"
.text
.align 2
.globl escribir
.ent escribir
escribir:
.frame $fp,40,$31 # vars= 0, regs= 3/0, args= 16, extra= 8
```

```
.mask 0xd0000000,-8
.fmask 0x00000000,0
.set noreorder
.cpload $25
.set reorder
subu $sp,$sp,40
.cprestore 16
sw $31,32($sp)
sw $fp,28($sp)
sw $28,24($sp)
move $fp,$sp
sw $4,40($fp)
sw $5,44($fp)
lw $4,40($fp)
lw $5,44($fp)
la $25,fputc
jal $31,$25
move $3,$2
li $2,-1 # 0xffffffffffffffff
bne $3,$2,$L17
la $25,__errno
jal $31,$25
lw $4,0($2)
la $25,strerror
jal $31,$25
la $4,__sF+176
la $5,$LC0
move $6,$2
la $25,fprintf
jal $31,$25
li $4,1 # 0x1
la $25,exit
jal $31,$25
$L17:
move $sp,$fp
lw $31,32($sp)
lw $fp,28($sp)
addu $sp,$sp,40
```

6 Enunciado

6.1 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.

6.2 Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

6.3 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 grupo 1, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

6.4 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.

6.5 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.

6.6 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.

6.7 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: For- mat 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.