66.20 Organización de Computadoras
Trabajo Práctico #0:
Infraestructura básica

97043 - González Pérez, Ailén Y.
96260 - Mariotti, María Eugenia
95457 - Raña, Cristian Ezequiel

18 de Abril de 2017 - Reentrega

1. Diseño e implementación

1.1. Diseño del programa

El programa se divide en dos funciones principales: codificador y decodificador. 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 aritmético lógicas (sumas, , shifts, etc) que relacionan la codificación base 64 con ASCII.

A su vez, se cuenta con una función readerçuyo objetivo es obtener, de la entrada, un string que sirva como parámetro para encode y decode.

Se definen las funciones auxiliares show_help y show_version, las cuales tienen como objetivo clarificar el código en main, evitando poner en esta función todas las líneas que componen estas funciones.

Se diseñan tres archivo de pruebas. Esta decisión tiene como objetivo probar de forma clara y rápidamente legible diversos resultados de nuestro programa, agrupando de acuerdo al tipo de prueba.

1.2. Parámetros

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. Si no se explicita -a, se realizará una codificación, ya que encode es la opción por defecto
- -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. La única excepción es la ayuda (-h); si este parámetro se especifica antes que -V, primero se imprimirá la ayuda y luego se procederá al "show and quit"version.
- -h también es de tipo "show and quitz se comporta de forma similar a -V.
- en caso de que se use la entrada estándar (con comando echo texto — ./tp0 etc) y luego se especifique un archivo de salida con -i, prevalecerá el establecido por parámetro.

2. Documentación del proceso de compilación y ejecución

2.1. Compilación y ejecución

La compilación del archivo fuente se realiza de la siguiente manera

```
$ make
```

Para proceder a la ejecución del programa, se debe llamar a (según el nombre elegido al compilar):

```
$ ./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:

```
$ echo textoDeEntrada | ./tp0
```

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

3. Prueba con archivo bash test.sh

Se realizan corridas de prueba con el siguiente script

```
#!/bin/bash

n=1
while :; do
        head -c $n </dev/urandom >in.bin;
        ./tp0 -a encode -i in.bin -o - | tr -d "\n" > out.b64
        base64 in.bin | tr -d "\n" > out-ref.b64;
        if diff -b out.b64 out-ref.b64; then :; else
                echo ERROR encoding: $n;
                break;
        fi

        ./tp0 -a decode -i out.b64 -o out.bin;
        if diff in.bin out.bin; then :; else
                echo ERROR decoding: $n;
                break;
        fi
        echo ok: $n;
        n="`expr $n + 1`";
        rm -f in.bin out.b64 out.bin out-ref.b64;
done
```

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

4. Corridas de prueba

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.[1]

Presentaremos: identificación de las pruebas, comandos para ejecutarlas, líneas de código que las componen y resultado esperado.

4.0.1. Generales

Se van a ejecutar cuando hagamos

```
$chmod +x pruebas_basicas.txt
$./pruebas_basicas.txt
```

- Muestra de ayuda

```
$ ./tp0 -h o ./tp0 --help

Usage:
  tp0 -h
  tp0 -V
  tp0 [options]

Options:
  -V, --version    Print version and quit.
  -h, --help     Print this information and quit.
  -i, --input    Location of the input file.
```

[1]Las distintas decodificaciones/codificaciones de referencia, se obtuvieron con el comando base64 de Linux, simulando exactamente la misma ejecución que con nuestro programa (archivos y entradas iguales).

```
  -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
```

- Muestra de version

```
$ ./tp0 -V o ./tp0 --version
Organizacion de Computadoras - TP0

Encoder/Decoder Base64 - v1.3

Group Members:
Gonzalez Perez, Ailen    Padron: 97043
Mariotti, Maria Eugenia  Padron: 96260
Ra[U+FFFD]a, Cristian Ezequiel  Padron: 95457
```

- Archivo de entrada no válido

```
$ ./tp0 -i fail.txt

Error de apertura de archivo de entrada. No se puede continuar.
```

4.0.2. Decodificador

La ejecución de los siguientes comandos debería dar como resultado (leído de corrido) un fragmento de **El amor en tiempos de cólera**[2]

- Ejecución de las pruebas

```
$chmod +x pruebas_decode.txt
$ ./pruebas_decode.txt
```

- Comandos que la componen

```
echo TGUgcm9nw7MgYSBEaW9zIHF1ZSBsZSBjb25jZWRpZXJhIGFsIG1lbm9zIHV
uIGluc3RhbnRlIHBhcmEgcXVlIMOpbCBubyBzZSBmdWVyYSBzaW4gc2FiZXIgY3X
DoW50byBsbyBoYWLDrWEgcXVlcmlkbyBwb3IgZW5jaW1hIGRlIGxhcyBkdWRhcyB
kZSBhbWJvcywgeSBzaW50acOzIHVuIGFwcmVtaW8gaXJyZXNpc3RpYmxlIGRlIGV
tcGV6YXIgbGEgdmlkYSBjb24gw6lsIG90cmEgdmV6IGRlc2RlIGVsIHByaW5jaXB
pbyBwYXJhIGRlY2lyc2UgdG9kbyBsbyBxdWUgc2UgbGVzIHF1ZWTDsyBzaW4gZGV
jaXIsIHkgdm9sdmVyIGEgaGFjZXIgYmllbiBjdWFscXVpZXIgY29zYSBxdWUgaHV
iaWVyYW4gaGVjaG8gbWFsIGVuIGVsIHBhc2Fkby4gUGVybyB0dXZvIHF1ZSByZW5
kaXJzZSBhbnRlIGxhIGludHJhbnNpZ2VuY2lhIGRlIGxhIG11ZXJ0ZS4gU3UgZG9
sb3Igc2UgZGVzY29tcHVzbyBlbiB1bmEgY8OzbGVyYSBjaWVnYSBjb250cmEgZWw
gbXVuZG8sIHkgYXVuIGNvbnRyYSBlbGxhIG1pc21hLCB5IGVzbyBsZSBpbmZ1bmR
pw7MgZWwgZG9taW5pbyB5IGVsIHZhbG9yIHBhcmEgZW5mcmVudGFyc2Ugc29sYSB
hIHN1IHNvbGVkYWQuCg==| .tp0 -a decode
```

- Texto experado

[2]Libro de Gabriel García Marquez cuya publicación original fue en 1985

Le rogó a Dios que le concediera al menos un instante para que él no se fuera sin saber cuánto lo había querido por encima de las dudas de ambos, y sintió un apremio irresistible de empezar la vida con él otra vez desde el principio para decirse todo lo que se les quedó sin decir, y volver a hacer bien cualquier cosa que hubieran hecho mal en el pasado. Pero tuvo que rendirse ante la intransigencia de la muerte. Su dolor se descompuso en una cólera ciega contra el mundo, y aun contra ella misma, y eso le infundió el dominio y el valor para enfrentarse sola a su soledad.

4.0.3. Codificador

La ejecución de los siguientes comandos debería dar como resultado (leído de corrido) la codificación en base 64 de un fragmento de **Rayuela**[3]

- Ejecución de las pruebas

```
$chmod +x pruebas_encode.txt
$ ./pruebas_encode.txt
```

- Líneas del texto original a codificar[4]

  Cada vez iré sintiendo menos y recordando más, pero qué es el recuerdo sino el idioma de los sentimientos, un diccionario de caras y días y perfumes que vuelven como los verbos y los adjetivos en el discurso.

- Comando que la compone [5]

```
echo Cada vez ire sintiendo menos y recordando mas, pero que es el recuerdo sino el idioma de
    los sentimientos, un diccionario de caras y dias y perfumes que vuelven como los verbos
    y los adjetivos en el discurso.| ./tp0
```

- Resultado esperado

```
Q2FkYSB2ZXogaXJlIHNpbnRpZW5kbyBtZW5vcyB5IHJlY29
yZGFuZG8gbWFzLCBwZXJvIHF1ZSBlcyBlbCByZWN1ZXJkby
BzaW5vIGVsIGlkaW9tYSBkZSBsb3Mgc2VudGltaWVudG9zL
CB1biBkaWNjaW9uYXJpbyBkZSBjYXJhcyB5IGRpYXMgeSBw
ZXJmdW1lcyBxdWUgdnVlbHZlbiBjb21vIGxvcyB2ZXJib3M
geSBsb3MgYWRqZXRpdm9zIGVuIGVsIGRpc2N1cnNvLgo=
```

4.0.4. Input-Output files

Todas se van a ejecutar cuando hagamos

```
$chmod +x pruebas_files.txt
$./pruebas_files.txt
```

- Comandos que la componen

```
./tp0 -i dText1.txt
./tp0 -i eText1.txt -a decode
./tp0 -i dText2.txt -o prueba3.txt
./tp0 -i eText2.txt -a decode -o prueba4.txt
echo Man | ./tp0 -o prueba5.txt
echo TWFu | ./tp0 -a decode -o prueba6.txt
```

[3]Libro de Julio Cortázar cuya publicación original fue en 1963
[4]Este texto se va a leer en lìneas separadas, ya que no se pasó entero al codificador, sino por partes.
[5]Se presentan aquí las palabras sin tilde, por cuestiones de formato, pero en el archivo se encuentran correctamente ecritas.

- Resultado esperado

  TWFuCg==
  Man
  Y la generación de cuatro archivos de salida, cuyo contenido será:

  - prueba3.txt: T3JnYW5pemFjacOzbiBkZSBjb21wdXRhZG9yYXMK
  - prueba4.txt: Organización de computadoras
  - prueba5.txt: TWFuCg==
  - prueba6.txt: Man

5. Conclusiones

- La ejecución de las pruebas, cuyos resultados se ven por consola, tiene una tardanza casi igual en ambos sistemas operativos.
- Es posible, con un correcto manejo de archivos, llevar el contenido de los mismos a cualquier formato que se requiera para otra función.

6. Anexo

6.1. Análisis de la decodificación

De aquí en adelante y hasya terminar esta sección, vamos a suponer que el ingreso fue TWFu (ejemplo del enunciado) y por lo que debemos obtener Man como resultado. Procederemos a desarrollar paso a paso las líneas correspondientes a la decodificación (explicando cada una de sus partes) y los valores obtenidos paso a paso [6][7].

```
//Primer paso
buf[0] = (tmp[0] << 2) + ((tmp[1] & 0x30) >> 4)

tmp[0] = 19 = 00010011
tmp[0] << 2 = 01001100 = 76

tmp[1] = 22 = 00010110
tmp[1] & 0x30 = 00010110 & 00110000 = 00010000 = 16
(tmp[1] & 0x30) >> 4 = 00000001 = 1

tmp[0] << 2 + (tmp[1] & 0x30) >> 4 = 76 + 1 = 77

Por lo tanto, buf[0] = 77.


//Segundo paso
buf[1] = ((tmp[1] & 0xf) << 4) + ((tmp[2] & 0x3c) >> 2)

tmp[1] = 22 = 00010110
tmp[1] & 0xf = 00010110 & 00001111 = 00000110 = 6
(tmp[1] & 0xf) << 4 = 01100000 = 96

tmp[2] = 00000101 = 5
tmp[2] & 0x3c = 00000101 & 00111100 = 00000100 = 4
(tmp[2] & 0x3c) >> 2 = 00000001 = 1

(tmp[1] & 0xf) << 4 + (tmp[2] & 0x3c) >> 2 = 97

Por lo tanto, buf[1] = 97.

//Tercer paso
buf[2] = ((tmp[2] & 0x3) << 6) + tmp[3];

tmp[2] = 00000101 = 5
tmp[2] & 0x3 = 00000101 & 00000011 = 00000001
tmp[2] & 0x3 << 6 = 01000000 = 64

tmp[3] = 46 = 00101110


tmp[2] & 0x3) << 6 + tmp[3] = 01000000 + 00101110 = 01101110 = 110


Por lo tanto, buf[2] = 110 .
```

Si buscamos los valores de buf[i] con i=0,1,2 en la tabla de caracteres ASCII observamos que: 77 corresponde al caracter M
97 corresponde al caracter a

[6] Los valores de tmp[i] se pueden verificar con la ejecución del código.
[7] Tener en cuenta: 0x30 = 48 0x3c = 60 0xf= 15 0x3 = 3

110 corresponde al caracter n

Por lo que la salida es, tal como se esperaba, **Man**.

6.2. Enunciado

Se adjunta a continuación el enunciado provisto por el equipo docente, en el cual se especifican los requerimientos del trabajo práctico

66:20 Organización de Computadoras
Trabajo práctico #0: Infraestructura básica
1^{er} cuatrimestre de 2017

`$Date: 2017/03/21 22:54:33 $`

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

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 7/3 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 2017-03-19 15:14 /tmp/zero.txt.b64
```

Codificamos el caracter ASCII `M`,

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

Codificamos los caracteres ASCII `M` y `a`,

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

Codificamos `M a n`,

```
$ 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
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
...
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
```

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.

7. Fechas

- Entrega: 28/3/2017;
- Vencimiento: 10/4/2017.

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`.

6.3. Código fuente

Se presenta a continuación el código fuente presente en el archivo tp0.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "encoder_decoder_t.h"

#define COMPARATOR 0
#define EXIT_SUCCESS 0
#define ERROR_ACTION 1
#define ENCODE 2
#define DECODE 3
#define BUFF_SZ 100
#define VERSION "version.txt"
#define HELP "help.txt"

int print_file(char *fname){
    char buff[BUFF_SZ + 1];
    FILE *file = fopen(fname, "r+");

    while(!feof(file)){
        memset(buff, '\0',(BUFF_SZ + 1)*sizeof(char));
        fread(buff, sizeof(char), BUFF_SZ, file);
        printf("%s",buff);
    }
    fclose(file);
    return EXIT_SUCCESS;
}

int show_version(){
    return print_file(VERSION);
}

int show_help(){
    return print_file(HELP);
}

bool free_mem(FILE* input, FILE* output){
    return (fclose(input) != EOF) && (fclose(output) != EOF);
}

int menu(int argc, char** argv, FILE **input, FILE **output){
    int exit = ENCODE;

    for (int counter = 1; counter < argc; counter++){

        if (strncmp(argv[counter],"-h", strlen("-h")) == COMPARATOR ||
            strncmp(argv[counter],"--help",strlen("--help")) == COMPARATOR){
            show_help();
            return EXIT_SUCCESS;
        }
        else if (strncmp(argv[counter],"-V", strlen("-V")) == COMPARATOR ||
            strncmp(argv[counter],"--version", strlen("--version")) == COMPARATOR){
            show_version();
            return EXIT_SUCCESS;
        }
        else if (strncmp(argv[counter],"-a",strlen("-a")) == COMPARATOR ||
            strncmp(argv[counter],"--action",strlen("--action")) == COMPARATOR){
            counter++;
```

```
            if (strncmp(argv[counter],"decode", strlen("decode")) == COMPARATOR){
                exit = DECODE;
            }
            else if (strncmp(argv[counter],"encode", strlen("encode")) == COMPARATOR){
                exit = ENCODE;
            }
            else{
                return ERROR_ACTION;
            }
        }
        else if (strncmp(argv[counter],"-o",strlen("-o")) == COMPARATOR ||
             strncmp(argv[counter],"--output",strlen("--output")) == COMPARATOR){
            counter++;
            if (strncmp(argv[counter],"-",strlen("-")) != COMPARATOR){
                FILE* salida = fopen(argv[counter],"wb+");
                if (!salida){
               printf("Error de apertura de archivo de salida. No se puede continuar.\n");
               return -1;
            }
            else{
               *output = salida;
            }
            }
        }
        else if (strncmp(argv[counter],"-i",strlen("-i")) == COMPARATOR ||
             strncmp(argv[counter],"--input",strlen("--input")) == COMPARATOR){
            counter++;
            if (strncmp(argv[counter],"-",strlen("-")) != COMPARATOR){
                FILE* entrada = fopen(argv[counter],"rb+");
                if (!entrada){
               printf("Error de apertura de archivo de entrada. No se puede continuar.\n");
               return -1;
            }
            else{
               *input = entrada;
            }

            }
        }
    }
    return exit;
}

int main(int argc, char* argv[]){
    EncDec_t encdec;
    FILE* input = stdin;
    FILE* output = stdout;

    int action_code = menu(argc, argv, &input, &output);
    if (action_code == -1){
         return -1;
   }
    init_encdec(&encdec, input, output);

    switch (action_code){
        case ENCODE:
            encode_text(&encdec);
            break;
        case DECODE:
            decode_text(&encdec);
```

```
            break;
        case EXIT_SUCCESS:
            return EXIT_SUCCESS;
    }
    return free_mem(input, output);
}
```

Se presenta a continuación el código de encoder_decoder_t.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <unistd.h>
#include "encoder_decoder_t.h"

//#define BYTE_GROUP 3
#define BYTE_SZ 8
#define GROUP_SZ 6.0
#define FILL_CHAR_POS 64
#define SYMBOL_POS 62
#define ENCODED_GROUP_SZ 4
#define DECODED_GROUP_SZ 3
//#define SUCCESS 0
#define NEW_LINE '\n'
#define MAX_LEN 76

#define MASK 0x3F
#define DMASK_1 0x30
#define DMASK_2 0xf
#define DMASK_3 0x3c
#define DMASK_4 0x3

#define DELTA_UPP 65
#define DELTA_LOW 71
#define DELTA_NUM 4
#define DELTA_SYM 18

static const char letters[] = {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','j','k',
'l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5',
'6','7','8','9','+','/','='};

int init_encdec(EncDec_t *self, FILE *input, FILE *output){
    self->input_file = input;
    self->output_file = output;
    self->state = SUCCESS;
    return SUCCESS;
}

char get_fill_char(EncDec_t *self){
    //Obtengo el caracter de relleno ('=')
    return letters[FILL_CHAR_POS];
}

void set_input(EncDec_t *self, FILE *input){
    self->input_file = input;
}

void set_output(EncDec_t *self, FILE *output){
```

```
    self->output_file = output;
}

char encode(EncDec_t *self, unsigned int letter_index){
    return letters[letter_index];
}

bool at_stdin_end(EncDec_t *self){
    return (self->input_file == stdin) && feof(self->input_file);
}

bool at_file_end(EncDec_t *self, int pos, int len){
    return (self->input_file != stdin) && (pos == len);
}

bool error_ocurred(EncDec_t *self){
    bool ioerr = ferror(self->input_file) || ferror(self->output_file);
    self->state = ioerr ? IO_ERROR : self->state;
    return self->state;
}

int concantenate_binary_to_int(unsigned char *characters){
    int number = 0;
    for(int i = 0; i < sizeof(int); ++i){
        number = number | (characters[i] << (sizeof(int) -1 -i)*BYTE_SZ);
    }
    return number;
}

int file_len(FILE *file){
    int len = 0, pos = ftell(file);
    if (file != stdin){
        fseek(file, 0, SEEK_END);
        len = ftell(file);
        fseek(file, pos, SEEK_SET);
    }
    return len;
}

int encode_text_to_output(EncDec_t *self, unsigned char *read_letters, int tot_read){
    //group_qty: la cantidad de grupos de 6 bits que puedo formar
    //con los bytes que leo:
    int max_group_qty = (DECODED_GROUP_SZ * BYTE_SZ) / GROUP_SZ;
    int group_qty = (int)ceil((double)((tot_read * BYTE_SZ) / GROUP_SZ));

    int read_bytes = 0;
    unsigned int index = 0, shift_count = 0;
    unsigned char encoded_chars[group_qty + 1];
    memset(&encoded_chars, '\0', (group_qty + 1)*sizeof(char));
    memset(&encoded_chars, get_fill_char(self), max_group_qty*sizeof(char));

    read_bytes = concantenate_binary_to_int(read_letters);

    //Ahora aplico operaciones logicas y obtengo un index:
    for (int j = 0; j < group_qty; ++j){
        shift_count = (max_group_qty - j - 1)*GROUP_SZ + BYTE_SZ;
        index = (read_bytes >> shift_count) & MASK;
        //Uso el index para obtener un caracter del
        //archivo de caracteres posibles:
        encoded_chars[j] = encode(self, index);
```

```
    }
    if(!error_ocurred(self)){
        fwrite(encoded_chars, sizeof(char), max_group_qty, self->output_file);
        self->state = ferror(self->output_file) ? IO_ERROR : self->state;
    }
    return self->state;
}

int encode_text(EncDec_t *self){
    int input_len = file_len(self->input_file);
    unsigned char read_letters[sizeof(int)];
    memset(&read_letters, '\0', sizeof(int));

    //Leo de a 3 bytes y codifico:
    int len = fread(read_letters, sizeof(char), DECODED_GROUP_SZ, self->input_file);
    while(!error_ocurred(self) && !at_stdin_end(self) && !at_file_end(self,
         ftell(self->input_file), input_len)){
        encode_text_to_output(self, read_letters, len);
        memset(&read_letters, '\0', sizeof(int));
        len = fread(read_letters, sizeof(char), DECODED_GROUP_SZ, self->input_file);
    }
    if(!error_ocurred(self) && (len > 0)){
        return encode_text_to_output(self, read_letters, len);
    }
    return self->state;
}

int decode_to_output_file(EncDec_t *self, char *letter_indexes, int padding){
    char buff[DECODED_GROUP_SZ + 1];
    memset(buff, '\0', (DECODED_GROUP_SZ + 1)*sizeof(char));

    //Teniendo los n[U+FFFD]meros asociados se hace la decodificaci[U+FFFD]n en s[U+FFFD] misma
    //relacionando los valores en b64 y ascii
    //A considerar: 0x30 = 48 0x3c = 60 0xf= 15 0x3 = 3
    //Se encuentran explicados los c[U+FFFD]lculos en el informe
    buff[0] = (letter_indexes[0] << 2) | ((letter_indexes[1] & DMASK_1) >> 4);
    buff[1] = ((letter_indexes[1] & DMASK_2) << 4) | ((letter_indexes[2] & DMASK_3) >> 2);
    buff[2] = ((letter_indexes[2] & DMASK_4) << 6) | letter_indexes[3];
    //Esto es para que si se codifico un caracter nulo
    //en el medio de los otros caracteres, este se copie
    //al archivo decodificado.
    size_t len = DECODED_GROUP_SZ - padding;
    if(!error_ocurred(self)){
        fwrite(buff, sizeof(char), len, self->output_file);
        self->state = ferror(self->output_file) ? IO_ERROR : self->state;
    }
    return self->state;
}

bool issymbol(EncDec_t *self, unsigned char *c, char *index){
    for (int i = SYMBOL_POS; letters[i] && i < FILL_CHAR_POS; ++i){
        if (*c == letters[i]){
            *index = i;
            return true;
        }
    }
    return false;
}

int decode(EncDec_t *self, unsigned char *letters, char fill_character, int count){
```

```
    char indexes[ENCODED_GROUP_SZ + 1];
    memset(indexes, '\0', (ENCODED_GROUP_SZ + 1)*sizeof(char));
    int padding = 0;
    for (int i = 0; i < count; ++i) {
        //Si no, tenemos que buscar el [U+FFFD]ndice de la letra.
        //en b64 y guardar su posici[U+FFFD]n.
        //TODO: esta pared de ifs hay que emprolijarla:
        if (letters[i] == fill_character){
            //Si la letra leida es el caracter de relleno,
            //llenamos con 0 la posicion de ese caracter.
            padding++;
            indexes[i] = 0;
            continue;
        }
        if(isupper(letters[i])){
            indexes[i] = letters[i] - DELTA_UPP;
            continue;
        }
        if(islower(letters[i])){
            indexes[i] = letters[i] - DELTA_LOW;
            continue;
        }
        if(isdigit(letters[i])){
            indexes[i] = letters[i] + DELTA_NUM;
            continue;
        }
        if(!issymbol(self, letters +i, indexes + i)){
            self->state = INVALID_CHARACTER;
            return INVALID_CHARACTER;
        }
    }
    return decode_to_output_file(self, indexes, padding);
}

int decode_text(EncDec_t *self){
    int input_len = file_len(self->input_file);
    self->state = SUCCESS;
    char fill_character = get_fill_char(self);
    unsigned char read_letters[ENCODED_GROUP_SZ + 1];
    memset(read_letters, '\0', (ENCODED_GROUP_SZ + 1)*sizeof(char));

    //Inicializo el array de caracteres leidos con el caractere
    //de relleno por si no se completa la cantidad de bytes esperados.
    //Esto solo agrega un '\0' al resultado decodificado.
    int qty_read = fread(read_letters, sizeof(char), ENCODED_GROUP_SZ, self->input_file);
    while(!error_ocurred(self) && !at_stdin_end(self) && !at_file_end(self,
        ftell(self->input_file), input_len)){
        decode(self, read_letters, fill_character, ENCODED_GROUP_SZ);
        memset(read_letters, '\0', ENCODED_GROUP_SZ*sizeof(char));
        qty_read = fread(read_letters, sizeof(char), ENCODED_GROUP_SZ, self->input_file);
    }
    if (!error_ocurred(self) && (qty_read > 0)){
        return decode(self, read_letters, fill_character, qty_read);
    }
    return self->state;
}
```

6.4. Código MIPS

Se presenta a continuación parte del código assembly MIPS obtenido al compilar en netBSD.

```
   .file 1 "tp0.c"
   .section .mdebug.abi32
   .previous
   .abicalls
   .rdata
   .align  2
$LC0:
   .ascii  "r+\000"
   .align  2
$LC1:
   .ascii  "%s\000"
   .text
   .align  2
   .globl  print_file
   .ent print_file
print_file:
   .frame  $fp,152,$31   # vars= 112, regs= 3/0, args= 16, extra= 8
   .mask 0xd0000000,-8
   .fmask  0x00000000,0
   .set noreorder
   .cpload $25
   .set reorder
   subu $sp,$sp,152
   .cprestore 16
   sw $31,144($sp)
   sw $fp,140($sp)
   sw $28,136($sp)
   move $fp,$sp
   sw $4,152($fp)
   lw $4,152($fp)
   la $5,$LC0
   la $25,fopen
   jal  $31,$25
   sw $2,128($fp)
$L18:
   lw $2,128($fp)
   lhu  $2,12($2)
   srl  $2,$2,5
   andi $2,$2,0x1
   beq  $2,$0,$L20
   b  $L19
$L20:
   addu $4,$fp,24
   move $5,$0
   li $6,101       # 0x65
   la $25,memset
   jal  $31,$25
   addu $4,$fp,24
   li $5,1      # 0x1
   li $6,100       # 0x64
   lw $7,128($fp)
   la $25,fread
   jal  $31,$25
   la $4,$LC1
   addu $5,$fp,24
   la $25,printf
   jal  $31,$25
```

```
   b  $L18
$L19:
   lw $4,128($fp)
   la $25,fclose
   jal  $31,$25
   move  $2,$0
   move  $sp,$fp
   lw $31,144($sp)
   lw $fp,140($sp)
   addu  $sp,$sp,152
   j  $31
   .end  print_file
   .size print_file, .-print_file
   .rdata
   .align  2
$LC2:
   .ascii  "version.txt\000"
   .text
   .align  2
   .globl  show_version
   .ent  show_version
show_version:
   .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
   la $4,$LC2
   la $25,print_file
   jal  $31,$25
   move  $sp,$fp
   lw $31,32($sp)
   lw $fp,28($sp)
   addu  $sp,$sp,40
   j  $31
   .end  show_version
   .size show_version, .-show_version
   .rdata
   .align  2
$LC3:
   .ascii  "help.txt\000"
   .text
   .align  2
```

Referencias

[1] Base64 (Wikipedia) http://en.wikipedia.org/wiki/Base64

[2] RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies; sección 6.8, Base64 Content-TransferEncoding. http://tools.ietf.org/html/rfc2045section-6.8

[3] 15 Practical Examples of 'echo' command in Linux http://www.tecmint.com/echo-command-in-linux/

[4] GXemul, http://gavare.se/gxemul/

[5] The NetBSD project, http://www.netbsd.org/

[6] https://www.base64decode.org/

[7] http://fm4dd.com/programming/base64/base64_algorithm.htm

[8] http://ascii.cl/es/

[9] https://es.wikipedia.org/wiki/Entrada_est