FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TP0 - Infraestructura básica

[66.20] Organización del Computador
Segundo cuatrimestre de 2020

Mariotti, Franco	102223	fmariotti@fi.uba.ar	Franco Mariotti
Fabbiano, Fernando	102464	ffabbiano@fi.uba.ar	Fernado Fabbiano
Alasino, Franco Federico	102165	falasino@fi.uba.ar	Franco Alasino

Índice

1. Introducción

En el presente trabajo, además de familiarizarnos con el entorno de desarrollo a utilizar durante la cursada, se desarrollo un programa que permite convertir archivos de texto a archivos en base64. A su vez, este mismo programa nos permite hacer el camino inverso, realizando una decodificación para pasar de base64 a texto.

Además de lo anteriormente mencionado, se utilizan los descriptores de archivo: `stdin, stdout y stderr`. El primero correspondiente a la entrada estándar, el segundo a la salida estándar, y el último correspondiente al error estándar.

Por último, se llevo a cabo una investigación y posterior utilización de la función `'getopt long'` para el manejo de las distintas opciones que se le brindan al usuario para interactuar con el programa en cuestión.

2. Diseño e Implementación

El programa consiste en un módulo principal `-main-` que se encarga de handlear las distintas opciones que se le brindan al usuario, sabiendo como responder a los comandos requeridos en el enunciado, pero también a casos en el que el usuario quiere vulnerar el sistema ingresando, por ejemplo, una opción inválida. Esta función main la podemos encontrar en el archivo `'tp0.c'`, junto con una función para abrir, leer, escribir y cerrar los archivos correspondientes.

Luego tenemos el archivo `'utils.c'` el cual contiene las funciones para realizar la codificación o decodificación de los textos adjuntados.

Por último, el archivo `'utils.h'` contiene la declaración de los métodos implementados en el archivo mencionado mencionado inmediatamente antes.

3. Proceso de Compilación

Para la compilación del programa, basta con ingresar en una consola, en el directorio correspondiente al proyecto, el comando `'make'`. Esto generará automáticamente los archivos binarios correspondientes, y los ejecutables (tanto el principal, como el de pruebas). Este último se guarda con el nombre de `'tp0'`.

Para la ejecución y posterior utilización del programa, se debe ingresar `'./tp0 -comando'` donde `'comando'` puede ser una de la siguientes opciones:

- -V, - -version –>Print version and quit
- -h, - -help –>Print información de ayuda acerca de los comandos
- -o, - -output –>Path para el output file donde se escribirá la salida del programa
- -i, - -input –>Path para el input file que se desea codificar a base 64
- -d, - -decode –>Decodificar un archivo encodeado en base64

4. Casos de Prueba

A continuación se muestra la salida por consola que se genera luego de ejecutar el paquete de tests pensado para este trabajo. Para generar dicha salida, será necesario correr el comando `'./test_utils'`. El código de las mismas se encuentra en la sección Código Fuente, en lenguaje C

Figura 1: Salida por consola luego de ejecutar las pruebas correspondientes.

5. Conclusiones

Con este trabajo pudimos entender como funciona y se configura un emulador dentro de nuestro mismo computador. A través de las problemáticas que surgieron a partir de estas instalaciones, fuimos capaces de indagar más y entender, por ejemplo, cómo se hace un túnel entre una máquina host y una guest.

Por otro lado, yendo al núcleo del trabajo, fuimos capaces de entender como funciona el sistema de base 64, y lograr llevarlo a código C. Para esto manipulamos arrays de caracteres a bajo nivel, y usamos, por ejemplo, funciones de desplazamiento de bits (o shifteos) para conseguir lo necesario para codificar y decodificar. Esto último nos parecio muy interesante, dado que hasta este momento de la carrera no habíamos utilizado en detalle dichos operandos en C, y por tanto no estabamos tan familiarizados con los mismos. De alguna forma lo pudimos relacionar con la aplicación de conocimientos mas bien teóricos vistos en la materia Estructura del Computador.

6. Referencias

- base64decode.org –>para codificar y decodificar, usado para comprobar los valores de referencia utilizados en las pruebas
- Wikipedia –>información general sobre el sistema de base64
- mycplusplus –>información acerca del algoritmo de decode y encode del sistema base64.
- Repositrio tp0 –>repositorio de github donde se puede encontrar el código fuente, el presente informe, imágenes y demás recursos utilizados para desarrollar el trabajo.

7. Apéndices

7.1. Código Fuente, en lenguaje C

tp0.c

```c
#define  _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```c
#include <getopt.h>
#include <stdbool.h>
#include "utils.h"

#define V_OPTION 'V'
#define H_OPTION 'h'
#define O_OPTION 'o'
#define I_OPTION 'i'
#define D_OPTION 'd'

#define MAX_LONGITUD 256
#define HELP_MESSAGE "Options:\n-V, --version\tPrint version and quit.\n-h, --help\t\
Print this information.\n-o, --output\tPath to output file.\n-i, \
--input\tPath to input file.\n-d, --decode\tDecode a base64-encoded\
file.\n"

#define INVALID_MESSAGE "Invalid option , use -h or --help to list valid commands\n"

enum operacion {DECODE,ENCODE,HELP,VERSION};

void imprimir_salida(const char* mensaje) {
  fprintf(stdout,"%s\n",mensaje);
}

void imprimir_error(const char* error) {
  fprintf(stderr,"%s\n",error);
}

int main(int argc, char **argv){
  int c;
  int operacion = ENCODE;
  const char* inputFileptr = NULL;
  const char* outputFileptr = NULL;
  char inputFileName[MAX_LONGITUD];
  char outputFileName[MAX_LONGITUD];
  memset(inputFileName,0,MAX_LONGITUD);
  memset(outputFileName,0,MAX_LONGITUD);

  while (true) {
    int option_index = 0;

    static struct option long_options[] = {
      {"version", no_argument,0, 'V'},
      {"help", no_argument, 0, 'h'},
      {"input", required_argument, 0, 'i'},
      {"output", required_argument, 0, 'o'},
      {"decode", no_argument, 0, 'd'}
    };

    c = getopt_long(argc, argv, "Vhdi:o:", long_options, &option_index);
    if (c == -1)
      break;

    switch (c) {
```

```
61        case V_OPTION:
62          operacion = VERSION;
63          break;
64        case H_OPTION:
65          operacion = HELP;
66          break;
67        case I_OPTION:
68          memcpy(inputFileName,optarg,strlen(optarg));
69          inputFileptr = inputFileName;
70          break;
71        case O_OPTION:
72          memcpy(outputFileName,optarg,strlen(optarg));
73          outputFileptr = outputFileName;
74          break;
75        case D_OPTION:
76          operacion = DECODE;
77          break;
78
79        default:
80          imprimir_salida(INVALID_MESSAGE);
81      }
82    }
83
84    if(operacion == ENCODE || operacion == DECODE) {
85      FILE* outfd = stdout;
86      FILE* infd = stdin;
87
88      if (inputFileptr) {
89        infd = fopen(inputFileName,"r");
90        if (!infd) {
91          imprimir_error("El archivo de input no existe\n");
92          return EXIT_FAILURE;
93        }
94      }
95
96      if (outputFileptr) {
97        outfd = fopen(outputFileName,"w");
98      }
99
100     if(operacion) {
101       encodeFileToBase64(infd,outfd);
102     } else {
103       decodeFileFromBase64(infd,outfd);
104     }
105
106     fclose(infd);
107     fclose(outfd);
108   } else if (operacion == HELP) {
109     imprimir_salida(HELP_MESSAGE);
110   } else if(operacion == VERSION) {
111     imprimir_salida("Version: 1.0.0\n");
112   }
113
114   return EXIT_SUCCESS;
```

```
}
```

utils.h

```h
#ifndef UTILS_H
#define UTILS_H
#include <stdio.h>
#include <stdlib.h>
#define ENCODE_SIZE 3 // o un multiplo de 3
#define DECODE_SIZE 4 // o un multiplo de 4
#define EXIT_FAILURE 1
#define SUCCESS 0

int encodeFileToBase64(FILE* fdin,FILE* fdout);
int decodeFileFromBase64(FILE* fdin,FILE* fdout);
char* encodeBase64(const char *data,size_t lenInput,size_t* lenOutput);
char* decodeBase64(const char *data,size_t lenInput,size_t* lenOutput);
#endif
```

utils.c

```c
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include "utils.h"

#define MASK_SEXTET 0x3F
#define MASK_OCTET 0xFF
#define SHIFT_SEXTET 6
#define BYTE_NULO 0
#define CARACTER_IGUAL '='

static int modTable[] = {0,2,1}; // cant de veces que tiene que iterar
static const unsigned char base64_table[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
//INICIO FUNCIONES PRIVADAS
static bool __b64_isvalidchar(char c) {
  bool valid = false;
  if (c >= '0' && c <= '9')
    valid = true;
  if (c >= 'A' && c <= 'Z')
    valid = true;
  if (c >= 'a' && c <= 'z')
    valid = true;
  if (c == '+' || c == '/' || c == '=')
    valid = true;
  return valid;
}

static size_t __len_base64_decode_output(const char *input,size_t lenInput) {
  if (!input) return 0;

  size_t len = 0;
  len = lenInput / 4 * 3;

  for (size_t i = 0; i < lenInput; i++) {
```

```
35      if (input[i] == CARACTER_IGUAL) {
36        len--;
37      }
38    }
39
40    return len;
41  }
42
43  static size_t __len_base64_encode_output(size_t inputLen) {
44    size_t ouputLen;
45    ouputLen =  inputLen;
46    if (inputLen % 3 != 0)
47      ouputLen += 3 - (inputLen % 3);
48    ouputLen /= 3;
49    ouputLen *= 4;
50
51    return ouputLen;
52  }
53
54  static int __write(FILE* file, char* data, size_t lenData) {
55    if(!file || !data) return EXIT_FAILURE;
56    fwrite(data,sizeof(char),lenData,file);
57    return SUCCESS;
58  }
59
60  static int __processFile(FILE* fdin,FILE* fdout,char* buffer,size_t lenBuffer,bool encode) {
61    char* output = NULL;
62    size_t len = 0;
63    size_t nread = 0;
64    while ((nread = fread(buffer,sizeof(char),lenBuffer,fdin)) > 0) {
65      if(encode) {
66        output = encodeBase64(buffer, nread,&len);
67      } else {
68        output = decodeBase64(buffer, nread,&len);
69      }
70      if(output){
71        __write(fdout,output,len);
72        free(output);
73      }
74      memset(buffer,0,lenBuffer);
75    }
76
77    if(nread == -1) return EXIT_FAILURE;
78
79    return SUCCESS;
80  }
81
82  //FIN FUNCIONES PRIVADAS
83
84  //INICIO FUNCIONES PUBLICAS
85  char* encodeBase64(const char *data,
86                     size_t lenInput,
87                     size_t *lenOutput) {
88    if(data == NULL || lenInput < 0) return NULL;
```

```
89
90    size_t maxlenOutput = __len_base64_encode_output(lenInput);
91    char* output = (char*)calloc(sizeof(char), maxlenOutput + 1);
92    if(!output) return NULL;
93
94    size_t i,j;
95
96    for (i = 0,j=0; i < lenInput; j+=4) {
97      uint32_t octet0 = (unsigned char) data[i++];
98      uint32_t octet1 = i < lenInput ?  data[i++] : BYTE_NULO;
99      uint32_t octet2 = i < lenInput ?  data[i++] : BYTE_NULO;
100
101     uint32_t bits = (octet0 << 16) + (octet1 << 8) + octet2;
102
103     uint32_t sextetA = (bits >> 18) & MASK_SEXTET;
104     uint32_t sextetB = (bits >> 12) & MASK_SEXTET;
105     uint32_t sextetC = (bits >> 6) & MASK_SEXTET;
106     uint32_t sextetD = (bits) & MASK_SEXTET;
107
108     output[j]   = base64_table[sextetA];
109     output[j+1] = base64_table[sextetB];
110     output[j+2] = base64_table[sextetC];
111     output[j+3] = base64_table[sextetD];
112
113   }
114
115   for(int i=0; i < modTable[lenInput%3]; i++)
116     output[maxlenOutput - 1 -i] = CARACTER_IGUAL;
117
118   *lenOutput = strlen(output);
119   return output;
120 }
121
122 char* decodeBase64(const char *data,
123                              size_t lenInput,
124                              size_t *lenOutput) {
125   if (!data) return NULL;
126   unsigned char* dataPtr = (unsigned char*)data;
127
128   unsigned char dTable[256];
129   memset(dTable, -1, 256);
130   for (size_t i = 0; i < sizeof(base64_table) - 1; i++)
131     dTable[base64_table[i]] = (unsigned char) i;
132
133   size_t lenOutputAux = __len_base64_decode_output(data,lenInput);
134
135   char *output = (char*)calloc(sizeof(char), lenOutputAux + 1);
136   if(!output) return NULL;
137
138   uint32_t decode;
139
140   size_t i;
141   size_t j;
142   for (i=0, j=0; i<lenInput; i+=4, j+=3) {
```

```
143     if (!__b64_isvalidchar(data[i])) continue;
144     decode = dTable[dataPtr[i]];
145     decode = (decode << 6) | dTable[dataPtr[i+1]];
146     decode = data[i+2] == CARACTER_IGUAL ? decode << 6 : (decode << 6) | dTable[dataPtr[i+2]];
147     decode = data[i+3] == CARACTER_IGUAL ? decode << 6 : (decode << 6) | dTable[dataPtr[i+3]];
148 
149     output[j] = (decode >> 16) & MASK_OCTET;
150     if (data[i+2] != '=')
151       output[j+1] = (decode >> 8) & MASK_OCTET;
152     if (data[i+3] != '=')
153       output[j+2] = decode & MASK_OCTET;
154   }
155   *lenOutput = strlen(output);
156 
157   return output;
158 }
159 
160 int encodeFileToBase64(FILE* fdin,FILE* fdout) {
161   if (!fdin || !fdout) return EXIT_FAILURE;
162   char buffer[ENCODE_SIZE];
163   memset(buffer,0,ENCODE_SIZE);
164   return __processFile(fdin,fdout,buffer,ENCODE_SIZE,true);
165 }
166 
167 int decodeFileFromBase64(FILE* fdin,FILE* fdout) {
168   if (!fdin || !fdout) return EXIT_FAILURE;
169   char buffer[DECODE_SIZE];
170   memset(buffer,0,DECODE_SIZE);
171   return __processFile(fdin,fdout,buffer,DECODE_SIZE,false);
172 }
173 
174 //FIN FUNCIONES PUBLICAS
```

tests

```
1  #include "utils.h"
2  #include "string.h"
3  
4  #define MAX_LONGITUD 256
5  
6  //Pruebas de la funcion encodeBase64
7  
8  int test00PasoABase64UnaFrase() {
9    size_t outputLen = 0;
10   char* frase = "En un lugar de La Mancha de cuyo nombre no quiero acordarme";
11   char* fraseEnBase64 = "RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29y
12   char* resultado = encodeBase64(frase,strlen(frase),&outputLen);
13 
14   if (strcmp(fraseEnBase64,resultado) == 0) {
15     printf("Paso una frase a base64 : OK\n");
16     return SUCCESS;
17   } else {
18     printf("Paso una frase a base64 : ERROR\n");
19     return EXIT_FAILURE;
```

```
20    }
21  }
22
23  int test01PasoABase64UnaFraseVacia() {
24    size_t outputLen = 0;
25    char* frase = "";
26    char* resultado = encodeBase64(frase,strlen(frase),&outputLen);
27
28    if (strcmp(frase,resultado) == 0) {
29      printf("Paso una frase vacia a base64 : OK\n");
30      return SUCCESS;
31    } else {
32      printf("Paso una frase vacia a base64 : ERROR\n");
33      return EXIT_FAILURE;
34    }
35  }
36
37  int test02PasoABase64UnaFraseInvalida() {
38    size_t outputLen = 0;
39    char* frase = NULL;
40    char* resultado = encodeBase64(frase,-1,&outputLen);
41
42    if (!resultado) {
43      printf("Paso una frase invalida a base64 : OK\n");
44      return SUCCESS;
45    } else {
46      printf("Paso una frase invalida a base64 : ERROR\n");
47      return EXIT_FAILURE;
48    }
49  }
50
51  //Pruebas de la funcion decodeBase64
52
53  int test00DecodificoUnaFraseEnBase64() {
54    size_t outputLen = 0;
55    char* frase = "En un lugar de La Mancha de cuyo nombre no quiero acordarme";
56    char* fraseEnBase64 = "RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVyby BhY29y
57    char* resultado = decodeBase64(fraseEnBase64,strlen(fraseEnBase64),&outputLen);
58
59
60    if (strcmp(frase,resultado) == 0) {
61      printf("Decodifico frase en base64 : OK\n");
62      return SUCCESS;
63    } else {
64      printf("Decodifico frase en base64 : ERROR\n");
65      return EXIT_FAILURE;
66    }
67  }
68
69  int test01DecodificoUnaFraseVacia() {
70    size_t outputLen = 0;
71    char* frase = "";
72    char* resultado = decodeBase64(frase,strlen(frase),&outputLen);
73
```

```
74    if (strcmp(frase,resultado) == 0) {
75      printf("Decodifico una frase vacia : OK\n");
76      return SUCCESS;
77    } else {
78      printf("Decodifico una frase vacia : ERROR\n");
79      return EXIT_FAILURE;
80    }
81  }
82
83  int test02DecodificoUnaFraseInvalida() {
84    size_t outputLen = 0;
85    char* frase = NULL;
86    char* resultado = decodeBase64(frase,-1,&outputLen);
87
88    if (!resultado) {
89      printf("Decodifico una frase invalida : OK\n");
90      return SUCCESS;
91    } else {
92      printf("Decodifico una frase invalida : ERROR\n");
93      return EXIT_FAILURE;
94    }
95  }
96
97  //Pruebas de la funcion encodeFileToBase64
98
99  int test00CodificoUsandoDosArchivosValidos() {
100   FILE* inputFile = fopen("entrada.txt", "r+");
101   FILE* outputFile = fopen("textoCodificado.txt","r+");
102
103   int resultado = encodeFileToBase64(inputFile,outputFile);
104   char* inputFileEnconding = "RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVyb
105
106   if (resultado == SUCCESS) {
107     char outputFileContent [MAX_LONGITUD];
108     memset(outputFileContent,0,MAX_LONGITUD);
109     fseek(outputFile, 0, SEEK_SET);
110     fread(outputFileContent,sizeof(char),MAX_LONGITUD,outputFile);
111
112     if (strcmp(inputFileEnconding,outputFileContent) == 0) {
113       printf("Codifico usando dos archivos validos : OK\n");
114       fclose(inputFile);
115       fclose(outputFile);
116       return SUCCESS;
117     } else {
118       printf("Codifico usando dos archivos validos : ERROR\n");
119       fclose(inputFile);
120       fclose(outputFile);
121       return EXIT_FAILURE;
122     }
123
124
125   } else {
126     printf("Codifico usando dos archivos validos : ERROR\n");
127     fclose(inputFile);
```

```
128     fclose(outputFile);
129     return EXIT_FAILURE;
130   }
131 }
132
133 int test01CodificoUsandoArchivoDeInputInexistente() {
134   FILE* inputFile = fopen("noExiste.txt", "r+");
135   FILE* outputFile = fopen("textoCodificado.txt","r+");
136
137   int resultado = encodeFileToBase64(inputFile,outputFile);
138
139   if (resultado == EXIT_FAILURE) {
140     printf("Codifico usando archivo de input inexistente : OK\n");
141     fclose(outputFile);
142     return SUCCESS;
143   } else {
144     printf("Codifico usando archivo de input inexistente : ERROR\n");
145     fclose(inputFile);
146     fclose(outputFile);
147     return EXIT_FAILURE;
148   }
149 }
150
151 int test02CodificoUsandoArchivoDeOutputInexistente() {
152   FILE* inputFile = fopen("entrada.txt", "r+");
153   FILE* outputFile = fopen("noExiste.txt","r+");
154
155   int resultado = encodeFileToBase64(inputFile,outputFile);
156
157   if (resultado == EXIT_FAILURE) {
158     printf("Codifico usando archivo de output inexistente : OK\n");
159     fclose(inputFile);
160     return SUCCESS;
161   } else {
162     printf("Codifico usando archivo de output inexistente : ERROR\n");
163     fclose(inputFile);
164     fclose(outputFile);
165     return EXIT_FAILURE;
166   }
167 }
168
169 int test03CodificoUnArchivoBinario() {
170   FILE* inputFile = fopen("pruebaBinario.bin", "r+");
171   FILE* outputFile = fopen("textoBinarioCodificado.bin","r+");
172
173   int resultado = encodeFileToBase64(inputFile,outputFile);
174   char* inputFileEnconding = "YmluYXJpbwo=";
175
176   if (resultado == SUCCESS) {
177     char outputFileContent [MAX_LONGITUD];
178     memset(outputFileContent,0,MAX_LONGITUD);
179     fseek(outputFile, 0, SEEK_SET);
180     fread(outputFileContent,sizeof(char),MAX_LONGITUD,outputFile);
181
```

```
182
183     if (strcmp(inputFileEnconding,outputFileContent) == 0) {
184       printf("Codifico un archivo binario: OK\n");
185       fclose(inputFile);
186       fclose(outputFile);
187       return SUCCESS;
188     } else {
189       printf("Codifico un archivo binario : ERROR\n");
190       fclose(inputFile);
191       fclose(outputFile);
192       return EXIT_FAILURE;
193     }
194
195
196   } else {
197     printf("Codifico un archivo binario : ERROR\n");
198     fclose(inputFile);
199     fclose(outputFile);
200     return EXIT_FAILURE;
201   }
202 }
203
204
205 //Pruebas de la funcion decodeFileToBase64
206
207 int test00DecodificoUsandoDosArchivosValidos() {
208   FILE* inputFile = fopen("textoCodificado.txt","r+");
209   FILE* outputFile = fopen("entrada.txt", "r+");
210
211   int resultado = decodeFileFromBase64(inputFile,outputFile);
212   char* inputFileDecoding = "En un lugar de La Mancha de cuyo nombre no quiero acordarme hola com
213
214   if (resultado == SUCCESS) {
215     char outputFileContent [MAX_LONGITUD];
216     memset(outputFileContent,0,MAX_LONGITUD);
217     fseek(outputFile, 0, SEEK_SET);
218     fread(outputFileContent,sizeof(char),MAX_LONGITUD,outputFile);
219
220     if (strcmp(inputFileDecoding,outputFileContent) == 0) {
221       printf("Codifico usando dos archivos validos : OK\n");
222       fclose(inputFile);
223       fclose(outputFile);
224       return SUCCESS;
225     } else {
226       printf("Codifico usando dos archivos validos : ERROR\n");
227       fclose(inputFile);
228       fclose(outputFile);
229       return EXIT_FAILURE;
230     }
231
232
233   } else {
234     printf("Codifico usando dos archivos validos : ERROR\n");
235     fclose(inputFile);
```

```
236     fclose(outputFile);
237     return EXIT_FAILURE;
238   }
239 }
240
241 int test01DecodificoUsandoArchivoDeInputInexistente() {
242   FILE* inputFile = fopen("noExiste.txt","r+");
243   FILE* outputFile = fopen("entrada.txt", "r+");
244
245   int resultado = decodeFileFromBase64(inputFile,outputFile);
246
247   if (resultado == EXIT_FAILURE) {
248     printf("Deodifico usando archivo de input inexistente : OK\n");
249     fclose(outputFile);
250     return SUCCESS;
251   } else {
252     printf("Decodifico usando archivo de input inexistente : ERROR\n");
253     fclose(inputFile);
254     fclose(outputFile);
255     return EXIT_FAILURE;
256   }
257 }
258
259 int test02DecodificoUsandoArchivoDeOutputInexistente() {
260   FILE* inputFile = fopen("textoCodificado.txt","r+");
261   FILE* outputFile = fopen("noExiste.txt", "r+");
262
263   int resultado = decodeFileFromBase64(inputFile,outputFile);
264
265   if (resultado == EXIT_FAILURE) {
266     printf("Decodifico usando archivo de output inexistente : OK\n");
267     fclose(inputFile);
268     return SUCCESS;
269   } else {
270     printf("Decodifico usando archivo de output inexistente : ERROR\n");
271     fclose(inputFile);
272     fclose(outputFile);
273     return EXIT_FAILURE;
274   }
275 }
276
277 int test03DecodificoUnArchivoBinario() {
278   FILE* inputFile = fopen("textoBinarioCodificado.bin","r+");
279   FILE* outputFile = fopen("pruebaBinario.bin", "r+");
280
281   int resultado = decodeFileFromBase64(inputFile,outputFile);
282   char* inputFileDecoding = "binario\n";
283
284   if (resultado == SUCCESS) {
285     char outputFileContent [MAX_LONGITUD];
286     memset(outputFileContent,0,MAX_LONGITUD);
287     fseek(outputFile, 0, SEEK_SET);
288     fread(outputFileContent,sizeof(char),MAX_LONGITUD,outputFile);
289
```

```
290
291     if (strcmp(inputFileDecoding,outputFileContent) == 0) {
292       printf("Decodifico un archivo binario: OK\n");
293       fclose(inputFile);
294       fclose(outputFile);
295       return SUCCESS;
296     } else {
297       printf("Decodifico un archivo binario : ERROR\n");
298       fclose(inputFile);
299       fclose(outputFile);
300       return EXIT_FAILURE;
301     }
302
303
304   } else {
305     printf("Codifico un archivo binario : ERROR\n");
306     fclose(inputFile);
307     fclose(outputFile);
308     return EXIT_FAILURE;
309   }
310 }
311
312
313 //Pruebas de uso
314
315 int test00PasoABase64UnaFraseYLuegoLaDecodifico() {
316   size_t outputLen = 0;
317   char* frase = "En un lugar de La Mancha de cuyo nombre no quiero acordarme";
318
319   char* encode = encodeBase64(frase,strlen(frase),&outputLen);
320   char* resultado = decodeBase64(encode,strlen(encode),&outputLen);
321
322   if (strcmp(frase,resultado) == 0) {
323     printf("Paso una frase a base64 y luego la decodifico : OK\n");
324     return SUCCESS;
325   } else {
326     printf("Paso una frase a base64 y luego la decodifico : ERROR\n");
327     return EXIT_FAILURE;
328   }
329 }
330
331
332
333 int tests() {
334   int resultado = 0;
335   printf("---Incio pruebas de la funcion encodeBase64---\n\n");
336   resultado += test00PasoABase64UnaFrase();
337   resultado += test01PasoABase64UnaFraseVacia();
338   resultado += test02PasoABase64UnaFraseInvalida();
339
340   printf("\n---Incio pruebas de la funcion decodeBase64---\n\n");
341   resultado += test00DecodificoUnaFraseEnBase64();
342   resultado += test01DecodificoUnaFraseVacia();
343   resultado += test02DecodificoUnaFraseInvalida();
```

```
344
345   printf("\n---Incio pruebas de la funcion encodeFileToBase64---\n\n");
346   resultado += test00CodificoUsandoDosArchivosValidos();
347   resultado += test01CodificoUsandoArchivoDeInputInexistente();
348   resultado += test02CodificoUsandoArchivoDeOutputInexistente();
349   resultado += test03CodificoUnArchivoBinario();
350
351   printf("\n---Incio pruebas de la funcion decodeFileToBase64---\n\n");
352   resultado += test00DecodificoUsandoDosArchivosValidos();
353   resultado += test01DecodificoUsandoArchivoDeInputInexistente();
354   resultado += test02DecodificoUsandoArchivoDeOutputInexistente();
355   resultado += test03DecodificoUnArchivoBinario();
356
357   printf("\n---Incio pruebas de uso---\n\n");
358   resultado += test00PasoABase64UnaFraseYLuegoLaDecodifico();
359
360   if (resultado == SUCCESS) return SUCCESS;
361   else return EXIT_FAILURE;
362 }
363
364 int main() {
365   return tests();
366 }
```

7.2. Código MIPS32, generado por el compilador

códigoMips32 - utils.c

```
1   .file  1 "utils.c"
2   .section .mdebug.abi32
3   .previous
4   .nan  legacy
5   .module  fp=xx
6   .module  nooddspreg
7   .abicalls
8   .data
9   .align  2
10  .type  modTable, @object
11  .size  modTable, 12
12 modTable:
13  .word  0
14  .word  2
15  .word  1
16  .rdata
17  .align  2
18  .type  base64_table, @object
19  .size  base64_table, 65
20 base64_table:
21  .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123"
22  .ascii  "456789+/\000"
23  .text
24  .align  2
25  .set  nomips16
26  .set  nomicromips
```

```
27    .ent  __b64_isvalidchar
28    .type  __b64_isvalidchar, @function
29 __b64_isvalidchar:
30    .frame  $fp,24,$31    # vars= 8, regs= 1/0, args= 0, gp= 8
31    .mask  0x40000000,-4
32    .fmask  0x00000000,0
33    .set  noreorder
34    .set  nomacro
35    addiu  $sp,$sp,-24
36    sw  $fp,20($sp)
37    move  $fp,$sp
38    move  $2,$4
39    sb  $2,24($fp)
40    sb  $0,8($fp)
41    lb  $2,24($fp)
42    slt  $2,$2,48
43    bne  $2,$0,$L2
44    nop
45
46    lb  $2,24($fp)
47    slt  $2,$2,58
48    beq  $2,$0,$L2
49    nop
50
51    li  $2,1      # 0x1
52    sb  $2,8($fp)
53 $L2:
54    lb  $2,24($fp)
55    slt  $2,$2,65
56    bne  $2,$0,$L3
57    nop
58
59    lb  $2,24($fp)
60    slt  $2,$2,91
61    beq  $2,$0,$L3
62    nop
63
64    li  $2,1      # 0x1
65    sb  $2,8($fp)
66 $L3:
67    lb  $2,24($fp)
68    slt  $2,$2,97
69    bne  $2,$0,$L4
70    nop
71
72    lb  $2,24($fp)
73    slt  $2,$2,123
74    beq  $2,$0,$L4
75    nop
76
```

```
77    li   $2,1       # 0x1
78    sb   $2,8($fp)
79  $L4:
80    lb   $3,24($fp)
81    li   $2,43      # 0x2b
82    beq  $3,$2,$L5
83    nop
84
85    lb   $3,24($fp)
86    li   $2,47      # 0x2f
87    beq  $3,$2,$L5
88    nop
89
90    lb   $3,24($fp)
91    li   $2,61      # 0x3d
92    bne  $3,$2,$L6
93    nop
94
95  $L5:
96    li   $2,1       # 0x1
97    sb   $2,8($fp)
98  $L6:
99    lbu  $2,8($fp)
100   move  $sp,$fp
101   lw  $fp,20($sp)
102   addiu  $sp,$sp,24
103   jr  $31
104   nop
105
106   .set  macro
107   .set  reorder
108   .end  __b64_isvalidchar
109   .size  __b64_isvalidchar, .-__b64_isvalidchar
110   .align  2
111   .set  nomips16
112   .set  nomicromips
113   .ent  __len_base64_decode_output
114   .type  __len_base64_decode_output, @function
115 __len_base64_decode_output:
116   .frame  $fp,24,$31    # vars= 8, regs= 1/0, args= 0, gp= 8
117   .mask  0x40000000,-4
118   .fmask  0x00000000,0
119   .set  noreorder
120   .set  nomacro
121   addiu  $sp,$sp,-24
122   sw  $fp,20($sp)
123   move  $fp,$sp
124   sw  $4,24($fp)
125   sw  $5,28($fp)
126   lw  $2,24($fp)
```

```
127   bne   $2,$0,$L9
128   nop
129
130   move   $2,$0
131   b   $L10
132   nop
133
134 $L9:
135   sw   $0,8($fp)
136   lw   $2,28($fp)
137   srl   $3,$2,2
138   move   $2,$3
139   sll   $2,$2,1
140   addu   $2,$2,$3
141   sw   $2,8($fp)
142   sw   $0,12($fp)
143   b   $L11
144   nop
145
146 $L13:
147   lw   $3,24($fp)
148   lw   $2,12($fp)
149   addu   $2,$3,$2
150   lb   $3,0($2)
151   li   $2,61        # 0x3d
152   bne   $3,$2,$L12
153   nop
154
155   lw   $2,8($fp)
156   addiu   $2,$2,-1
157   sw   $2,8($fp)
158 $L12:
159   lw   $2,12($fp)
160   addiu   $2,$2,1
161   sw   $2,12($fp)
162 $L11:
163   lw   $3,12($fp)
164   lw   $2,28($fp)
165   sltu   $2,$3,$2
166   bne   $2,$0,$L13
167   nop
168
169   lw   $2,8($fp)
170 $L10:
171   move   $sp,$fp
172   lw   $fp,20($sp)
173   addiu   $sp,$sp,24
174   jr   $31
175   nop
```

```
176
177   .set  macro
178   .set  reorder
179   .end  __len_base64_decode_output
180   .size  __len_base64_decode_output, .-__len_base64_decode_output
181   .align  2
182   .set  nomips16
183   .set  nomicromips
184   .ent  __len_base64_encode_output
185   .type  __len_base64_encode_output, @function
186 __len_base64_encode_output:
187   .frame  $fp,24,$31    # vars= 8, regs= 1/0, args= 0, gp= 8
188   .mask  0x40000000,-4
189   .fmask  0x00000000,0
190   .set  noreorder
191   .set  nomacro
192   addiu  $sp,$sp,-24
193   sw  $fp,20($sp)
194   move  $fp,$sp
195   sw  $4,24($fp)
196   lw  $2,24($fp)
197   sw  $2,8($fp)
198   lw  $4,24($fp)
199   li  $2,-1431699456      # 0xffffffffaaaa0000
200   ori  $2,$2,0xaaab
201   multu  $4,$2
202   mfhi  $2
203   srl  $3,$2,1
204   move  $2,$3
205   sll  $2,$2,1
206   addu  $2,$2,$3
207   subu  $3,$4,$2
208   beq  $3,$0,$L15
209   nop
210
211   lw  $4,24($fp)
212   li  $2,-1431699456      # 0xffffffffaaaa0000
213   ori  $2,$2,0xaaab
214   multu  $4,$2
215   mfhi  $2
216   srl  $3,$2,1
217   move  $2,$3
218   sll  $2,$2,1
219   addu  $2,$2,$3
220   subu  $3,$4,$2
221   lw  $2,8($fp)
222   subu  $2,$2,$3
223   addiu  $2,$2,3
224   sw  $2,8($fp)
```

```
225 $L15:
226   lw  $3,8($fp)
227   li  $2,-1431699456      # 0xffffffffaaaa0000
228   ori  $2,$2,0xaaab
229   multu  $3,$2
230   mfhi  $2
231   srl  $2,$2,1
232   sw  $2,8($fp)
233   lw  $2,8($fp)
234   sll  $2,$2,2
235   sw  $2,8($fp)
236   lw  $2,8($fp)
237   move  $sp,$fp
238   lw  $fp,20($sp)
239   addiu  $sp,$sp,24
240   jr  $31
241   nop
242
243   .set  macro
244   .set  reorder
245   .end  __len_base64_encode_output
246   .size  __len_base64_encode_output, .-__len_base64_encode_output
247   .align  2
248   .set  nomips16
249   .set  nomicromips
250   .ent  __write
251   .type  __write, @function
252 __write:
253   .frame  $fp,32,$31    # vars= 0, regs= 2/0, args= 16, gp= 8
254   .mask  0xc0000000,-4
255   .fmask  0x00000000,0
256   .set  noreorder
257   .cpload  $25
258   .set  nomacro
259   addiu  $sp,$sp,-32
260   sw  $31,28($sp)
261   sw  $fp,24($sp)
262   move  $fp,$sp
263   .cprestore  16
264   sw  $4,32($fp)
265   sw  $5,36($fp)
266   sw  $6,40($fp)
267   lw  $2,32($fp)
268   beq  $2,$0,$L18
269   nop
270
271   lw  $2,36($fp)
272   bne  $2,$0,$L19
273   nop
274
```

```
275 $L18:
276   li   $2,1       # 0x1
277   b   $L20
278   nop
279
280 $L19:
281   lw   $7,32($fp)
282   lw   $6,40($fp)
283   li   $5,1       # 0x1
284   lw   $4,36($fp)
285   lw   $2,%call16(fwrite)($28)
286   move   $25,$2
287   .reloc   1f,R_MIPS_JALR,fwrite
288 1:  jalr   $25
289   nop
290
291   lw   $28,16($fp)
292   move   $2,$0
293 $L20:
294   move   $sp,$fp
295   lw   $31,28($sp)
296   lw   $fp,24($sp)
297   addiu   $sp,$sp,32
298   jr   $31
299   nop
300
301   .set   macro
302   .set   reorder
303   .end   __write
304   .size   __write, .-__write
305   .align   2
306   .set   nomips16
307   .set   nomicromips
308   .ent   __processFile
309   .type   __processFile, @function
310 __processFile:
311   .frame   $fp,48,$31    # vars= 16, regs= 2/0, args= 16, gp= 8
312   .mask   0xc0000000,-4
313   .fmask   0x00000000,0
314   .set   noreorder
315   .cpload   $25
316   .set   nomacro
317   addiu   $sp,$sp,-48
318   sw   $31,44($sp)
319   sw   $fp,40($sp)
320   move   $fp,$sp
321   .cprestore   16
322   sw   $4,48($fp)
323   sw   $5,52($fp)
324   sw   $6,56($fp)
```

```
325   sw   $7,60($fp)
326   sw   $0,24($fp)
327   sw   $0,32($fp)
328   sw   $0,28($fp)
329   b    $L22
330   nop
331
332 $L26:
333   lbu  $2,67($fp)
334   beq  $2,$0,$L23
335   nop
336
337   addiu  $2,$fp,32
338   move   $6,$2
339   lw   $5,28($fp)
340   lw   $4,56($fp)
341   lw   $2,%got(encodeBase64)($28)
342   move   $25,$2
343   .reloc  1f,R_MIPS_JALR,encodeBase64
344 1:  jalr   $25
345   nop
346
347   lw   $28,16($fp)
348   sw   $2,24($fp)
349   b    $L24
350   nop
351
352 $L23:
353   addiu  $2,$fp,32
354   move   $6,$2
355   lw   $5,28($fp)
356   lw   $4,56($fp)
357   lw   $2,%got(decodeBase64)($28)
358   move   $25,$2
359   .reloc  1f,R_MIPS_JALR,decodeBase64
360 1:  jalr   $25
361   nop
362
363   lw   $28,16($fp)
364   sw   $2,24($fp)
365 $L24:
366   lw   $2,24($fp)
367   beq  $2,$0,$L25
368   nop
369
370   lw   $2,32($fp)
371   move   $6,$2
372   lw   $5,24($fp)
373   lw   $4,52($fp)
```

```
374   lw   $2,%got(__write)($28)
375   addiu   $2,$2,%lo(__write)
376   move   $25,$2
377   .reloc   1f,R_MIPS_JALR,__write
378 1:   jalr   $25
379   nop
380
381   lw   $28,16($fp)
382   lw   $4,24($fp)
383   lw   $2,%call16(free)($28)
384   move   $25,$2
385   .reloc   1f,R_MIPS_JALR,free
386 1:   jalr   $25
387   nop
388
389   lw   $28,16($fp)
390 $L25:
391   lw   $6,60($fp)
392   move   $5,$0
393   lw   $4,56($fp)
394   lw   $2,%call16(memset)($28)
395   move   $25,$2
396   .reloc   1f,R_MIPS_JALR,memset
397 1:   jalr   $25
398   nop
399
400   lw   $28,16($fp)
401 $L22:
402   lw   $7,48($fp)
403   lw   $6,60($fp)
404   li   $5,1       # 0x1
405   lw   $4,56($fp)
406   lw   $2,%call16(fread)($28)
407   move   $25,$2
408   .reloc   1f,R_MIPS_JALR,fread
409 1:   jalr   $25
410   nop
411
412   lw   $28,16($fp)
413   sw   $2,28($fp)
414   lw   $2,28($fp)
415   bne   $2,$0,$L26
416   nop
417
418   lw   $3,28($fp)
419   li   $2,-1       # 0xffffffffffffffff
420   bne   $3,$2,$L27
421   nop
422
423   li   $2,1       # 0x1
```

```
424   b  $L29
425   nop
426
427 $L27:
428   move  $2,$0
429 $L29:
430   move  $sp,$fp
431   lw  $31,44($sp)
432   lw  $fp,40($sp)
433   addiu  $sp,$sp,48
434   jr  $31
435   nop
436
437   .set  macro
438   .set  reorder
439   .end  __processFile
440   .size  __processFile, .-__processFile
441   .align  2
442   .globl  encodeBase64
443   .set  nomips16
444   .set  nomicromips
445   .ent  encodeBase64
446   .type  encodeBase64, @function
447 encodeBase64:
448   .frame  $fp,88,$31    # vars= 56, regs= 2/0, args= 16, gp= 8
449   .mask  0xc0000000,-4
450   .fmask  0x00000000,0
451   .set  noreorder
452   .cpload  $25
453   .set  nomacro
454   addiu  $sp,$sp,-88
455   sw  $31,84($sp)
456   sw  $fp,80($sp)
457   move  $fp,$sp
458   .cprestore  16
459   sw  $4,88($fp)
460   sw  $5,92($fp)
461   sw  $6,96($fp)
462   lw  $2,88($fp)
463   bne  $2,$0,$L31
464   nop
465
466   move  $2,$0
467   b  $L32
468   nop
469
470 $L31:
471   lw  $4,92($fp)
472   lw  $2,%got(__len_base64_encode_output)($28)
473   addiu  $2,$2,%lo(__len_base64_encode_output)
474   move  $25,$2
```

```
475   .reloc  1f,R_MIPS_JALR,__len_base64_encode_output
476 1:  jalr  $25
477   nop
478
479   lw  $28,16($fp)
480   sw  $2,36($fp)
481   lw  $2,36($fp)
482   addiu  $2,$2,1
483   move  $5,$2
484   li  $4,1        # 0x1
485   lw  $2,%call16(calloc)($28)
486   move  $25,$2
487   .reloc  1f,R_MIPS_JALR,calloc
488 1:  jalr  $25
489   nop
490
491   lw  $28,16($fp)
492   sw  $2,40($fp)
493   lw  $2,40($fp)
494   bne  $2,$0,$L33
495   nop
496
497   move  $2,$0
498   b  $L32
499   nop
500
501 $L33:
502   sw  $0,24($fp)
503   sw  $0,28($fp)
504   b  $L34
505   nop
506
507 $L39:
508   lw  $2,24($fp)
509   addiu  $3,$2,1
510   sw  $3,24($fp)
511   lw  $3,88($fp)
512   addu  $2,$3,$2
513   lb  $2,0($2)
514   andi  $2,$2,0x00ff
515   sw  $2,44($fp)
516   lw  $3,24($fp)
517   lw  $2,92($fp)
518   sltu  $2,$3,$2
519   beq  $2,$0,$L35
520   nop
521
522   lw  $2,24($fp)
523   addiu  $3,$2,1
```

```
524   sw  $3,24($fp)
525   lw  $3,88($fp)
526   addu  $2,$3,$2
527   lb  $2,0($2)
528   b  $L36
529   nop
530
531 $L35:
532   move  $2,$0
533 $L36:
534   sw  $2,48($fp)
535   lw  $3,24($fp)
536   lw  $2,92($fp)
537   sltu  $2,$3,$2
538   beq  $2,$0,$L37
539   nop
540
541   lw  $2,24($fp)
542   addiu  $3,$2,1
543   sw  $3,24($fp)
544   lw  $3,88($fp)
545   addu  $2,$3,$2
546   lb  $2,0($2)
547   b  $L38
548   nop
549
550 $L37:
551   move  $2,$0
552 $L38:
553   sw  $2,52($fp)
554   lw  $2,44($fp)
555   sll  $3,$2,16
556   lw  $2,48($fp)
557   sll  $2,$2,8
558   addu  $3,$3,$2
559   lw  $2,52($fp)
560   addu  $2,$3,$2
561   sw  $2,56($fp)
562   lw  $2,56($fp)
563   srl  $2,$2,18
564   andi  $2,$2,0x3f
565   sw  $2,60($fp)
566   lw  $2,56($fp)
567   srl  $2,$2,12
568   andi  $2,$2,0x3f
569   sw  $2,64($fp)
570   lw  $2,56($fp)
571   srl  $2,$2,6
```

```
572   andi  $2,$2,0x3f
573   sw  $2,68($fp)
574   lw  $2,56($fp)
575   andi  $2,$2,0x3f
576   sw  $2,72($fp)
577   lw  $3,40($fp)
578   lw  $2,28($fp)
579   addu  $2,$3,$2
580   lw  $3,%got(base64_table)($28)
581   addiu  $4,$3,%lo(base64_table)
582   lw  $3,60($fp)
583   addu  $3,$4,$3
584   lbu  $3,0($3)
585   seb  $3,$3
586   sb  $3,0($2)
587   lw  $2,28($fp)
588   addiu  $2,$2,1
589   lw  $3,40($fp)
590   addu  $2,$3,$2
591   lw  $3,%got(base64_table)($28)
592   addiu  $4,$3,%lo(base64_table)
593   lw  $3,64($fp)
594   addu  $3,$4,$3
595   lbu  $3,0($3)
596   seb  $3,$3
597   sb  $3,0($2)
598   lw  $2,28($fp)
599   addiu  $2,$2,2
600   lw  $3,40($fp)
601   addu  $2,$3,$2
602   lw  $3,%got(base64_table)($28)
603   addiu  $4,$3,%lo(base64_table)
604   lw  $3,68($fp)
605   addu  $3,$4,$3
606   lbu  $3,0($3)
607   seb  $3,$3
608   sb  $3,0($2)
609   lw  $2,28($fp)
610   addiu  $2,$2,3
611   lw  $3,40($fp)
612   addu  $2,$3,$2
613   lw  $3,%got(base64_table)($28)
614   addiu  $4,$3,%lo(base64_table)
615   lw  $3,72($fp)
616   addu  $3,$4,$3
617   lbu  $3,0($3)
618   seb  $3,$3
```

```
619   sb  $3,0($2)
620   lw  $2,28($fp)
621   addiu  $2,$2,4
622   sw  $2,28($fp)
623 $L34:
624   lw  $3,24($fp)
625   lw  $2,92($fp)
626   sltu  $2,$3,$2
627   bne  $2,$0,$L39
628   nop
629
630   sw  $0,32($fp)
631   b  $L40
632   nop
633
634 $L41:
635   lw  $2,32($fp)
636   lw  $3,36($fp)
637   subu  $2,$3,$2
638   addiu  $2,$2,-1
639   lw  $3,40($fp)
640   addu  $2,$3,$2
641   li  $3,61       # 0x3d
642   sb  $3,0($2)
643   lw  $2,32($fp)
644   addiu  $2,$2,1
645   sw  $2,32($fp)
646 $L40:
647   lw  $4,92($fp)
648   li  $2,-1431699456      # 0xffffffffaaaa0000
649   ori  $2,$2,0xaaab
650   multu  $4,$2
651   mfhi  $2
652   srl  $3,$2,1
653   move  $2,$3
654   sll  $2,$2,1
655   addu  $2,$2,$3
656   subu  $3,$4,$2
657   lw  $2,%got(modTable)($28)
658   sll  $3,$3,2
659   addiu  $2,$2,%lo(modTable)
660   addu  $2,$3,$2
661   lw  $3,0($2)
662   lw  $2,32($fp)
663   slt  $2,$2,$3
664   bne  $2,$0,$L41
665   nop
666
```

```
667   lw  $4,40($fp)
668   lw  $2,%call16(strlen)($28)
669   move  $25,$2
670   .reloc  1f,R_MIPS_JALR,strlen
671 1:  jalr  $25
672   nop
673
674   lw  $28,16($fp)
675   move  $3,$2
676   lw  $2,96($fp)
677   sw  $3,0($2)
678   lw  $2,40($fp)
679 $L32:
680   move  $sp,$fp
681   lw  $31,84($sp)
682   lw  $fp,80($sp)
683   addiu  $sp,$sp,88
684   jr  $31
685   nop
686
687   .set  macro
688   .set  reorder
689   .end  encodeBase64
690   .size  encodeBase64, .-encodeBase64
691   .align  2
692   .globl  decodeBase64
693   .set  nomips16
694   .set  nomicromips
695   .ent  decodeBase64
696   .type  decodeBase64, @function
697 decodeBase64:
698   .frame  $fp,320,$31    # vars= 288, regs= 2/0, args= 16, gp= 8
699   .mask  0xc0000000,-4
700   .fmask  0x00000000,0
701   .set  noreorder
702   .cpload  $25
703   .set  nomacro
704   addiu  $sp,$sp,-320
705   sw  $31,316($sp)
706   sw  $fp,312($sp)
707   move  $fp,$sp
708   .cprestore  16
709   sw  $4,320($fp)
710   sw  $5,324($fp)
711   sw  $6,328($fp)
712   lw  $2,320($fp)
713   bne  $2,$0,$L43
714   nop
715
716   move  $2,$0
717   b  $L57
```

```
718   nop
719
720 $L43:
721   lw  $2,320($fp)
722   sw  $2,36($fp)
723   addiu  $2,$fp,52
724   li  $6,256      # 0x100
725   li  $5,-1       # 0xffffffffffffffff
726   move  $4,$2
727   lw  $2,%call16(memset)($28)
728   move  $25,$2
729   .reloc  1f,R_MIPS_JALR,memset
730 1:  jalr  $25
731   nop
732
733   lw  $28,16($fp)
734   sw  $0,24($fp)
735   b  $L45
736   nop
737
738 $L46:
739   lw  $2,%got(base64_table)($28)
740   addiu  $3,$2,%lo(base64_table)
741   lw  $2,24($fp)
742   addu  $2,$3,$2
743   lbu  $2,0($2)
744   move  $4,$2
745   lw  $2,24($fp)
746   andi  $3,$2,0x00ff
747   addiu  $2,$fp,24
748   addu  $2,$2,$4
749   sb  $3,28($2)
750   lw  $2,24($fp)
751   addiu  $2,$2,1
752   sw  $2,24($fp)
753 $L45:
754   lw  $2,24($fp)
755   sltu  $2,$2,64
756   bne  $2,$0,$L46
757   nop
758
759   lw  $5,324($fp)
760   lw  $4,320($fp)
761   lw  $2,%got(__len_base64_decode_output)($28)
762   addiu  $2,$2,%lo(__len_base64_decode_output)
763   move  $25,$2
764   .reloc  1f,R_MIPS_JALR,__len_base64_decode_output
765 1:  jalr  $25
766   nop
```

```
767
768   lw  $28,16($fp)
769   sw  $2,40($fp)
770   lw  $2,40($fp)
771   addiu  $2,$2,1
772   move  $5,$2
773   li  $4,1        # 0x1
774   lw  $2,%call16(calloc)($28)
775   move  $25,$2
776   .reloc  1f,R_MIPS_JALR,calloc
777 1:  jalr  $25
778   nop
779
780   lw  $28,16($fp)
781   sw  $2,44($fp)
782   lw  $2,44($fp)
783   bne  $2,$0,$L47
784   nop
785
786   move  $2,$0
787   b  $L57
788   nop
789
790 $L47:
791   sw  $0,28($fp)
792   sw  $0,32($fp)
793   b  $L48
794   nop
795
796 $L56:
797   lw  $3,320($fp)
798   lw  $2,28($fp)
799   addu  $2,$3,$2
800   lb  $2,0($2)
801   move  $4,$2
802   lw  $2,%got(__b64_isvalidchar)($28)
803   addiu  $2,$2,%lo(__b64_isvalidchar)
804   move  $25,$2
805   .reloc  1f,R_MIPS_JALR,__b64_isvalidchar
806 1:  jalr  $25
807   nop
808
809   lw  $28,16($fp)
810   xori  $2,$2,0x1
811   andi  $2,$2,0x00ff
812   bne  $2,$0,$L58
813   nop
814
815   lw  $3,36($fp)
816   lw  $2,28($fp)
```

```
817   addu  $2,$3,$2
818   lbu  $2,0($2)
819   move  $3,$2
820   addiu  $2,$fp,24
821   addu  $2,$2,$3
822   lbu  $2,28($2)
823   sw  $2,48($fp)
824   lw  $2,48($fp)
825   sll  $2,$2,6
826   lw  $3,28($fp)
827   addiu  $3,$3,1
828   lw  $4,36($fp)
829   addu  $3,$4,$3
830   lbu  $3,0($3)
831   move  $4,$3
832   addiu  $3,$fp,24
833   addu  $3,$3,$4
834   lbu  $3,28($3)
835   or  $2,$2,$3
836   sw  $2,48($fp)
837   lw  $2,28($fp)
838   addiu  $2,$2,2
839   lw  $3,320($fp)
840   addu  $2,$3,$2
841   lb  $3,0($2)
842   li  $2,61       # 0x3d
843   bne  $3,$2,$L51
844   nop
845
846   lw  $2,48($fp)
847   sll  $2,$2,6
848   b  $L52
849   nop
850
851 $L51:
852   lw  $2,48($fp)
853   sll  $2,$2,6
854   lw  $3,28($fp)
855   addiu  $3,$3,2
856   lw  $4,36($fp)
857   addu  $3,$4,$3
858   lbu  $3,0($3)
859   move  $4,$3
860   addiu  $3,$fp,24
861   addu  $3,$3,$4
862   lbu  $3,28($3)
863   or  $2,$2,$3
```

```
$L52:
  sw  $2,48($fp)
  lw  $2,28($fp)
  addiu  $2,$2,3
  lw  $3,320($fp)
  addu  $2,$3,$2
  lb  $3,0($2)
  li  $2,61       # 0x3d
  bne  $3,$2,$L53
  nop

  lw  $2,48($fp)
  sll  $2,$2,6
  b  $L54
  nop

$L53:
  lw  $2,48($fp)
  sll  $2,$2,6
  lw  $3,28($fp)
  addiu  $3,$3,3
  lw  $4,36($fp)
  addu  $3,$4,$3
  lbu  $3,0($3)
  move  $4,$3
  addiu  $3,$fp,24
  addu  $3,$3,$4
  lbu  $3,28($3)
  or  $2,$2,$3
$L54:
  sw  $2,48($fp)
  lw  $3,44($fp)
  lw  $2,32($fp)
  addu  $2,$3,$2
  lw  $3,48($fp)
  srl  $3,$3,16
  seb  $3,$3
  sb  $3,0($2)
  lw  $2,28($fp)
  addiu  $2,$2,2
  lw  $3,320($fp)
  addu  $2,$3,$2
  lb  $3,0($2)
  li  $2,61       # 0x3d
  beq  $3,$2,$L55
  nop

  lw  $2,32($fp)
```

```
912   addiu  $2,$2,1
913   lw  $3,44($fp)
914   addu  $2,$3,$2
915   lw  $3,48($fp)
916   srl  $3,$3,8
917   seb  $3,$3
918   sb  $3,0($2)
919 $L55:
920   lw  $2,28($fp)
921   addiu  $2,$2,3
922   lw  $3,320($fp)
923   addu  $2,$3,$2
924   lb  $3,0($2)
925   li  $2,61       # 0x3d
926   beq  $3,$2,$L50
927   nop
928
929   lw  $2,32($fp)
930   addiu  $2,$2,2
931   lw  $3,44($fp)
932   addu  $2,$3,$2
933   lw  $3,48($fp)
934   seb  $3,$3
935   sb  $3,0($2)
936   b  $L50
937   nop
938
939 $L58:
940   nop
941 $L50:
942   lw  $2,28($fp)
943   addiu  $2,$2,4
944   sw  $2,28($fp)
945   lw  $2,32($fp)
946   addiu  $2,$2,3
947   sw  $2,32($fp)
948 $L48:
949   lw  $3,28($fp)
950   lw  $2,324($fp)
951   sltu  $2,$3,$2
952   bne  $2,$0,$L56
953   nop
954
955   lw  $4,44($fp)
956   lw  $2,%call16(strlen)($28)
957   move  $25,$2
958   .reloc  1f,R_MIPS_JALR,strlen
959 1:  jalr  $25
```

```
960   nop
961
962   lw  $28,16($fp)
963   move  $3,$2
964   lw  $2,328($fp)
965   sw  $3,0($2)
966   lw  $2,44($fp)
967 $L57:
968   move  $sp,$fp
969   lw  $31,316($sp)
970   lw  $fp,312($sp)
971   addiu  $sp,$sp,320
972   jr  $31
973   nop
974
975   .set  macro
976   .set  reorder
977   .end  decodeBase64
978   .size  decodeBase64, .-decodeBase64
979   .align  2
980   .globl  encodeFileToBase64
981   .set  nomips16
982   .set  nomicromips
983   .ent  encodeFileToBase64
984   .type  encodeFileToBase64, @function
985 encodeFileToBase64:
986   .frame  $fp,48,$31    # vars= 8, regs= 2/0, args= 24, gp= 8
987   .mask  0xc0000000,-4
988   .fmask  0x00000000,0
989   .set  noreorder
990   .cpload  $25
991   .set  nomacro
992   addiu  $sp,$sp,-48
993   sw  $31,44($sp)
994   sw  $fp,40($sp)
995   move  $fp,$sp
996   .cprestore  24
997   sw  $4,48($fp)
998   sw  $5,52($fp)
999   lw  $2,48($fp)
1000  beq  $2,$0,$L60
1001  nop
1002
1003  lw  $2,52($fp)
1004  bne  $2,$0,$L61
1005  nop
1006
1007 $L60:
1008  li  $2,1      # 0x1
1009  b  $L63
1010  nop
```

```
1011
1012 $L61:
1013   li  $6,3      # 0x3
1014   move  $5,$0
1015   addiu  $2,$fp,32
1016   move  $4,$2
1017   lw  $2,%call16(memset)($28)
1018   move  $25,$2
1019   .reloc  1f,R_MIPS_JALR,memset
1020 1:  jalr  $25
1021   nop
1022
1023   lw  $28,24($fp)
1024   li  $2,1      # 0x1
1025   sw  $2,16($sp)
1026   li  $7,3      # 0x3
1027   addiu  $2,$fp,32
1028   move  $6,$2
1029   lw  $5,52($fp)
1030   lw  $4,48($fp)
1031   lw  $2,%got(__processFile)($28)
1032   addiu  $2,$2,%lo(__processFile)
1033   move  $25,$2
1034   .reloc  1f,R_MIPS_JALR,__processFile
1035 1:  jalr  $25
1036   nop
1037
1038   lw  $28,24($fp)
1039 $L63:
1040   move  $sp,$fp
1041   lw  $31,44($sp)
1042   lw  $fp,40($sp)
1043   addiu  $sp,$sp,48
1044   jr  $31
1045   nop
1046
1047   .set  macro
1048   .set  reorder
1049   .end  encodeFileToBase64
1050   .size  encodeFileToBase64, .-encodeFileToBase64
1051   .align  2
1052   .globl  decodeFileFromBase64
1053   .set  nomips16
1054   .set  nomicromips
1055   .ent  decodeFileFromBase64
1056   .type  decodeFileFromBase64, @function
1057 decodeFileFromBase64:
1058   .frame  $fp,48,$31    # vars= 8, regs= 2/0, args= 24, gp= 8
1059   .mask  0xc0000000,-4
1060   .fmask  0x00000000,0
```

```
1061    .set  noreorder
1062    .cpload  $25
1063    .set  nomacro
1064    addiu  $sp,$sp,-48
1065    sw  $31,44($sp)
1066    sw  $fp,40($sp)
1067    move  $fp,$sp
1068    .cprestore  24
1069    sw  $4,48($fp)
1070    sw  $5,52($fp)
1071    lw  $2,48($fp)
1072    beq  $2,$0,$L65
1073    nop
1074
1075    lw  $2,52($fp)
1076    bne  $2,$0,$L66
1077    nop
1078
1079  $L65:
1080    li  $2,1      # 0x1
1081    b  $L68
1082    nop
1083
1084  $L66:
1085    li  $6,4      # 0x4
1086    move  $5,$0
1087    addiu  $2,$fp,32
1088    move  $4,$2
1089    lw  $2,%call16(memset)($28)
1090    move  $25,$2
1091    .reloc  1f,R_MIPS_JALR,memset
1092  1:  jalr  $25
1093    nop
1094
1095    lw  $28,24($fp)
1096    sw  $0,16($sp)
1097    li  $7,4      # 0x4
1098    addiu  $2,$fp,32
1099    move  $6,$2
1100    lw  $5,52($fp)
1101    lw  $4,48($fp)
1102    lw  $2,%got(__processFile)($28)
1103    addiu  $2,$2,%lo(__processFile)
1104    move  $25,$2
1105    .reloc  1f,R_MIPS_JALR,__processFile
1106  1:  jalr  $25
1107    nop
1108
1109    lw  $28,24($fp)
1110  $L68:
```

```
1111   move   $sp,$fp
1112   lw   $31,44($sp)
1113   lw   $fp,40($sp)
1114   addiu   $sp,$sp,48
1115   jr   $31
1116   nop
1117
1118   .set  macro
1119   .set  reorder
1120   .end  decodeFileFromBase64
1121   .size  decodeFileFromBase64, .-decodeFileFromBase64
1122   .ident  "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```

7.3. Enunciado

66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica

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 7), 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, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa QEMU [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 Debian [2].

5. Base 64

La codificación base 64 [3] se creó para poder transmitir archivos binarios en medios que sólo admitían texto: 64 es la mayor potencia de 2 que se podía

representar sólo con caracteres ASCII imprimibles. Básicamente se tiene una tabla de conversión de combinaciones de 6 bits a caracteres ASCII, se 'corta' el archivo en secuencias de 6 bits y se transmiten los caracteres correspondientes a esas secuencias. Cada tres bytes de la secuencia original se generan cuatro caracteres base64; cuando la cantidad de bytes original no es múltiplo de tres, se adicionan caracteres '=' al final en cantidad necesaria.

6. Programa

El programa a escribir, en lenguaje C, recibirá un nombre de archivo (o el archivo mismo por `stdin`) y devolverá ese mismo archivo codificado en `base64` [3], o bien decodificado desde base64 si se utiliza la opción `-d`.

6.1. 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.
  -o, --output      Path to output file.
  -i, --input       Path to input file.
  -d, --decode      Decode a base64-encoded file.
Examples:

  tp0 -i input.txt -o output.txt
```

Luego, lo usamos para codificar un pequeño fragmento de texto:

```
$ cat quijote.txt
En un lugar de La Mancha de cuyo nombre no quiero acordarme
$ ./tp0 -i quijote.txt -o qb64
$ cat qb64
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29yZGFy
bWUK
```

Otra manera de ejecutarlo es a través de `stdin` y/o `stdout`:

```
cat quijote.txt | ./tp0
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29yZGFy
bWUK
```

También se puede usar para decodificar:

```
$ ./tp0 -d -i qb64 -o texto
$ cat texto
En un lugar de La Mancha de cuyo nombre no quiero acordarme
```

7. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C;
- El código MIPS32 generado por el compilador;
- Este enunciado.

8. Fecha de entrega

La fecha de entrega es el jueves 22 de Octubre de 2020.

Referencias

[1] QEMU, `https://www.qemu.org/`.

[2] Debian, the Universal Operating System, `https://www.debian.org/`.

[3] Codificación base64, `https://es.wikipedia.org/wiki/Base64`