FACULTAD DE INGENIERIA
Universidad de Buenos Aires

[66.20] ORGANIZACIÓN DE COMPUTADORAS

TRABAJO PRÁCTICO 0

2DO CUATRIMESTRE 2018

Infraestructura básica

AUTORES

Husain, Ignacio Santiago. - #90.117
<santiago.husain@gmail.com>
Pesado, Lucía. - #98.275
<luupesado@gmail.com>
Verstraeten, Federico. - #92.892
<federico.verstraeten@gmail.com>

CÁTEDRA

Dr. Ing. Hamkalo, José Luis.

CURSO

Ing. Santi, Leandro.
Ing. Perez Masci, Hernán.
Ing. Natale, Luciano.

FECHA DE ENTREGA 24 de septiembre de 2018

FECHA DE APROBACIÓN

CALIFICACIÓN

FIRMA DE APROBACIÓN

Índice

1. Enunciado del trabajo práctico **2**

2. Objetivos **6**

3. Diseño e implementación del programa **6**

4. Compilación del programa **8**

5. Pruebas **9**
5.1. Pruebas en las opciones de programa 9
5.1.1. Pruebas de codificación y decodificación 12
5.1.2. Tiempos de codificación y decodificación 14

6. Herramientas de hardware y software utilizadas **17**

7. Conclusiones **17**

Referencias **18**

A. Makefile **19**
A.0.1. makefile 19

B. Tests **21**
B.0.1. runTests.sh 21

C. Código fuente **35**
C.0.1. main.c 35
C.0.2. common.h 37
C.0.3. decoder.h 39
C.0.4. decoder.c 41
C.0.5. encoder.h 46
C.0.6. encoder.c 48
C.0.7. parser.h 52
C.0.8. parser.c 54
C.0.9. messages.h 59
C.0.10. main.s 61
C.0.11. parser.s 63
C.0.12. encoder.s 79
C.0.13. decoder.s 87

1. Enunciado del trabajo práctico

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

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

1. Objetivos

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

2. Alcance

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

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

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

El informe deberá respetar el modelo de referencia que se encuentra en el 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 21/8 hemos repasado los pasos necesarios para la instalación y configuración del entorno de desarrollo.

[1] `http://groups.yahoo.com/group/orga-comp`

5. Programa

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

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

5.1. Ejemplos

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

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

Codificamos un archivo vacío (cantidad de bytes nula):

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

Codificamos el 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.

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`.

2. Objetivos

El presente trabajo tiene los siguientes objetivos:

- Diseñar un codificador/decodificador (*codec*) de información en formato base 64 utilizando el lenguaje de programación C.
- Compilar dicho codec en el sistema operativo netBSD en una máquina con arquitectura MIPS32, y producir el código assembly MIPS32 del mismo.
- Realizar pruebas de caja negra para verificar que el programa está funcionando de manera correcta.
- Utilizar la infraestructura básica que será utilizada en trabajos posteriores para programar en assembly MIPS32.

3. Diseño e implementación del programa

Se diseñó un programa en lenguaje ANSI C que implementa el codec descripto en el documento RFC2045 *Base64 Content-Transfer-Encoding* [4], [5], con la diferencia de que si en la etapa de decodificación se encuentra un carácter no válido, la ejecución del programa se suspende, notificando al usuario del error producido.

El programa se estructura de la siguiente manera:

- Análisis gramatical de la línea de comandos: se analizan las opciones ingresadas por la línea de comandos haciendo uso de la función `parseCmdline()`. La misma se encarga de inicializar una estructura del tipo `params_t` utilizada para almacenar las opciones que ingresó el cliente, y cuya defición es

```
1        typedef struct params_t
2        {
3          char *action;
4          FILE *inputStream;
5          FILE *outputStream;
6        } params_t;
```

  Además, hace uso de la función `getopt_long()` de la biblioteca `getopt.h`. Dicha función provee una forma simple de procesar cada opción que es leída, extrayendo los argumentos de cada una. En caso de que no se encuentre alguna opción, se utiliza su valor por defecto según las especificaciones del trabajo.

- Validación de opciones: a medida que se va analizando cada opción de la línea de comandos, se valida cada una de ellas utilizando las funciones

```
1        outputCode optAction()
2        outputCode optOutput()
```

```
3         outputCode optInput()
4         void optHelp()
5         void optVersion()
```

Además, las mismas realizan la correcta inicialización de las diferentes variables dentro de la estructura `params_t` descripta en el punto anterior, o en caso de que el usuario ingresó las opciones de ayuda e indicación de versión del programa, se imprime por el flujo `stderr` dicha información.

En caso de que se encuentre algún error en el argumento de alguna de las opciones, el usuario es informado por el flujo `stderr`, y se aborta la ejecución del programa utilizando la función `exit()`. Para ello, se creó un tipo enumerativo para simplificar el manejo de errores, definiendo los códigos que pueden devolver las funciones desarrolladas:

```
1     typedef enum outputCodes_ {
2     outOK,
3     outERROR
4     } outputCode;
```

- Codificación/Decodificación: una vez realizada la validación de las opciones y carga de configuración del programa, se utilizan las funciones `base256ToBase64()` o `base64ToBase256()` para codificar o decodificar el flujo de entrada, dependiendo de qué fue lo que el cliente le solicitó al programa.

  El proceso de codificación se realiza tomando cada carácter de entrada, y guardando los dos bits más y menos significativos para poder concatenarlos con los próximos caracteres que vayan ingresando por el flujo de entrada. Se realiza la extracción de los bits utilizando una máscara de bits y desplazándola adecuadamente dependiendo cuántos bytes se fueron leyendo. Con esta información, se extrae el índice de la tabla de codificación, y se almacenan los caracteres codificados en un arreglo de 4 bytes que posteriormente son impresos en el flujo de salida especificado. Cuando por el flujo de entrada ingresa "EOF", se agrega el carácter de padding "=" dependiendo de cuántos caracteres hayan sido codificados en los pasos previos para que el arreglo de salida quede con 4 bytes completos.

  Por otro lado, la función `encode()` se encarga no solo de codificar el flujo de entrada, si no de llevar un registro de la cantidad de caracteres codificados para incluir un salto de línea cada 76 caracteres como se establece en la especificación del codec.

  El proceso de decodificación se realiza en tres pasos y de a bloques de 4 bytes. Para cada carácter a decodificar, primero se busca en la tabla traducción `translationTableB64` el índice correspondiente al carácter codificado que ingresó por el flujo de entrada. Luego se lo posiciona dentro de un entero de 4 bytes, en una posición que depende de cuál de los 4 bytes del bloque de entrada fue leído. Por último, se utiliza una máscara de bits para ir extrayendo cada uno de los caracteres de 1 byte correspondientes, y guardándolos en un array de salida, que será escrito en el flujo de salida posteriormente por la función `decode()`.

- Terminación del programa: una vez finalizada la codificación o decodificación, se cierran los flujos de entrada y salida, y se retorna al sistema operativo.

El código fuente se encuentra en el apéndice, tanto en lenguaje C (C.0.1) como en MIPS32 (C.0.10).

4. Compilación del programa

Debido al requerimiento de utilizar el programa en una computadora con arquitectura MIPS32, se utiliza el emulador `GXemul` que provee la cátedra, utilizando una máquina virtual que contiene el sistema operativo NetBSD con las herramientas `gcc` y `make` para compilar el programa desarrollado.

Para obtener un ejecutable, se creó un archivo `makefile` cuyo contenido se puede ver en la sección A.0.1. Para ejecutarlo, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
1    $ make
```

donde el mismo generará el ejecutable con nombre `tp0`. En caso de requerir que el código assembly MIPS32 de cada archivo con código fuente, ejecutar el siguiente comando:

```
1    $ make assembly ARGS=-mrnames
```

5. Pruebas

Según Patton en [2], una prueba de caja negra (o *black-box test*) es aquella donde se realizan pruebas al programa sin acceder al código fuente. Es decir, la única información a la que se tiene acceso es aquella que define el comportamiento del programa según las especificaciones del cliente. En esta sección se muestran los resultados de las distintas pruebas de caja negra que se realizaron sobre el programa para determinar su robustez y fiabilidad ante diferentes tipos de entradas.

Se creó un script en lenguaje `Bash` para automatizar las pruebas del programa. El código del script se encuentra en la sección B.0.1, y está compuesto por 17 tests junto con 2 para medir tiempos de ejecución.

La salida del script se divide en 2 secciones, cada una con un encabezado indicando el inicio del nuevo test y su nombre, y varias líneas por cada test. La primer línea del test es el comando ejecutado, indicado con la etiqueta `Testing`. La segunda indica si el test fue exitoso o no mediante la etiqueta `PASSED/FAILED` en color verde o rojo respectivamente, y las siguientes líneas son los resultados que produce el programa (mensajes de error, etc...). Por ejemplo, para la prueba de la opción "-i", se tiene lo siguiente:

```
1 ------------------------------------------------------
2 TEST1: inexistent 'input' stream.
3 ------------------------------------------------------
4 Testing: ./tp0 -i 1
5 PASSED
6   PROGRAM OUTPUT:
7     ERROR: Can't open input stream.
```

donde se ve que el test fue satisfactorio ya que se introdujo un nombre de flujo de entrada inválido.

Para ejecutar el script con las pruebas, posicionarse en el directorio `src/` y ejecutar el comando `/.runTests.sh` en la terminal de Linux.

Si todos los tests pasan, entonces al final de la ejecución se debe obtener el siguiente mensaje

```
1 ------------------------------------------------------
2 Test suite ended.
3 ------------------------------------------------------
4   All tests passed.
```

En caso de que no pasen todos los tests, el script indicará cuántos no lo hicieron en color rojo.

5.1. Pruebas en las opciones de programa

En el script de tests se prueban diferentes combinaciones de las opciones de entrada para verificar si el programa es capaz de detectar errores. Los 7 primeros son validaciones utilizando opciones y parámetros inválidos, donde se verifica que al intentar ejecutarlo, el pro-

grama termina y retorna un mensaje que indique el motivo de la ejecución fallida. El test `test4_valid_parameters` se corresponde con ejecuciones que retornan un código de éxito.

Las salidas arrojadas por el script fueron las siguientes:

```
-----------------------------------------------------
TEST1: inexistent 'input' stream.
-----------------------------------------------------
Testing: ./tp0 -i 1
PASSED
  PROGRAM OUTPUT:
    ERROR: Can't open input stream.
Testing: ./tp0 -i hola.bin
PASSED
  PROGRAM OUTPUT:
    ERROR: Can't open input stream.
-----------------------------------------------------
TEST11: no 'input' option parameters.
-----------------------------------------------------
Testing: ./tp0 -i
PASSED
  PROGRAM OUTPUT:
    ./tp0: option requires an argument -- 'i'
-----------------------------------------------------
TEST12: invalid 'input' stream.
-----------------------------------------------------
Testing: ./tp0 -i .
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid input stream.
Testing: ./tp0 -i ..
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid input stream.
Testing: ./tp0 -i /
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid input stream.
Testing: ./tp0 -i //
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid input stream.
-----------------------------------------------------
TEST2: invalid 'output' stream.
-----------------------------------------------------
Testing: ./tp0 -i ../tests/leviathan.input -o .
PASSED
  PROGRAM OUTPUT:
```

```
    ERROR: Invalid output stream.
Testing: ./tp0 -i ../tests/leviathan.input -o ..
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid output stream.
Testing: ./tp0 -i ../tests/leviathan.input -o /
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid output stream.
Testing: ./tp0 -i ../tests/leviathan.input -o //
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid output stream.
-----------------------------------------------------
TEST21: no 'output' option parameters.
-----------------------------------------------------
Testing: ./tp0 -o
PASSED
  PROGRAM OUTPUT:
    ./tp0: option requires an argument -- 'o'
-----------------------------------------------------
TEST3: invalid 'action' parameters.
-----------------------------------------------------
Testing: ./tp0 -a bad_action
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
Testing: ./tp0 -a 1
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
Testing: ./tp0 -a .
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
Testing: ./tp0 -a ..
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
Testing: ./tp0 -a /
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
Testing: ./tp0 -a //
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
```

```
Testing: ./tp0 -a $
PASSED
  PROGRAM OUTPUT:
    ERROR: Invalid action argument.
-----------------------------------------------------
TEST31: no 'action' option parameters.
-----------------------------------------------------
Testing: ./tp0 -a
PASSED
  PROGRAM OUTPUT:
    ./tp0: option requires an argument -- 'a'
-----------------------------------------------------
TEST4: all options with correct parameters.
-----------------------------------------------------
Testing: ./tp0 -a encode -i ../tests/test1.bin -o ../tests/↩
   test1_out.bin
PASSED
  PROGRAM OUTPUT:

Testing: ./tp0 -a decode -i ../tests/test1_out.bin -o ../tests↩
   /test2.bin
PASSED
  PROGRAM OUTPUT:
```

5.1.1. Pruebas de codificación y decodificación

Los tests cuyo nombre son del estilo `test5x_IO_validation` se crearon para probar si el codec funciona correctamente, ya sea ingresando palabras y/o archivos cuya codificación es conocida, como también archivos aleatorios que son codificados y decodificados para probar si el programa realiza una operación de identidad. También se probó ingresar al decodificador caracteres que no estén en la tabla de codificación para determinar si el programa cortaba la ejecución como se encuentra especificado en el enunciado.

Además, la prueba `test57_IO_validation` verifica que el programa produzca saltos de línea cada 76 caracteres según la especificación del enunciado, y además que si se codificaron N caracteres, la decodificación produzca N caracteres.

Las salidas del script para dichas pruebas fue la siguiente:

```
-----------------------------------------------------
TEST5: input-output should be the same.
-----------------------------------------------------
PASSED:  n = 1
PASSED:  n = 2
PASSED:  n = 4
PASSED:  n = 8
PASSED:  n = 16
PASSED:  n = 32
```

```
PASSED:  n = 64
PASSED:  n = 128
PASSED:  n = 256
PASSED:  n = 512
PASSED:  n = 1024
PASSED:  n = 2048
PASSED:  n = 4096
PASSED:  n = 8192
PASSED:  n = 16384
PASSED:  n = 32768
PASSED:  n = 65536
PASSED:  n = 131072
PASSED:  n = 262144
PASSED:  n = 524288
------------------------------------------------------
TEST51: input known text with known encoding.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST52: Encode letter 'M'.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST53: Encode 'Ma'.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST54: Encode 'Man'.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST55: Encode and decode 'Man'.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST56: Check bit by bit.
------------------------------------------------------
PASSED:  No differences.
------------------------------------------------------
TEST57: Check max line length and number of encoded bytes.
------------------------------------------------------
PASSED:  No differences in line count.
PASSED:  No differences in word count.
------------------------------------------------------
TEST58: decoding input with chars not in B64 table should ↩
   produce error.
------------------------------------------------------
```

```
PASSED
  PROGRAM OUTPUT:
    ERROR: Character is not in Base64 Table.
```

5.1.2. Tiempos de codificación y decodificación

Los tests `test6_encoding_execution_times` y `test7_decoding_execution_times` se utilizan para evaluar los tiempos de ejecución del programa completo cuando se codifican y decodifican archivos binarios aleatorios de varios tamaños.

Las salidas de los mismos se muestran a continuación, donde la letra `n` indica la cantidad de bytes del archivo binario.

```
-------------------------------------------------------
TEST6: encoding execution times.
-------------------------------------------------------
n: 1                        1,00 [ms]
n: 2                        1,00 [ms]
n: 4                        2,00 [ms]
n: 8                        1,00 [ms]
n: 16                       2,00 [ms]
n: 32                       2,00 [ms]
n: 64                       2,00 [ms]
n: 128                      5,00 [ms]
n: 256                      1,00 [ms]
n: 512                      1,00 [ms]
n: 1024                     6,00 [ms]
n: 2048                     1,00 [ms]
n: 4096                     1,00 [ms]
n: 8192                     2,00 [ms]
n: 16384                    3,00 [ms]
n: 32768                    3,00 [ms]
n: 65536                    6,00 [ms]
n: 131072                   12,00 [ms]
n: 262144                   19,00 [ms]
n: 524288                   49,00 [ms]
n: 1048576                  78,00 [ms]
n: 2097152                  121,00 [ms]
n: 4194304                  214,00 [ms]
n: 8388608                  429,00 [ms]
-------------------------------------------------------
TEST7: decoding execution times.
-------------------------------------------------------
n: 1                        1,00 [ms]
n: 2                        2,00 [ms]
n: 4                        1,00 [ms]
n: 8                        1,00 [ms]
```

```
n: 16                    4,00 [ms]
n: 32                    2,00 [ms]
n: 64                    3,00 [ms]
n: 128                   1,00 [ms]
n: 256                   3,00 [ms]
n: 512                   4,00 [ms]
n: 1024                  3,00 [ms]
n: 2048                  2,00 [ms]
n: 4096                  2,00 [ms]
n: 8192                  3,00 [ms]
n: 16384                 8,00 [ms]
n: 32768                 15,00 [ms]
n: 65536                 15,00 [ms]
n: 131072                25,00 [ms]
n: 262144                49,00 [ms]
n: 524288                88,00 [ms]
n: 1048576               170,00 [ms]
n: 2097152               332,00 [ms]
n: 4194304               665,00 [ms]
n: 8388608               1342,00 [ms]
```

Los gráficos correspondientes a los tiempos de ejecución se muestran en las figuras 5.1 y 5.2.

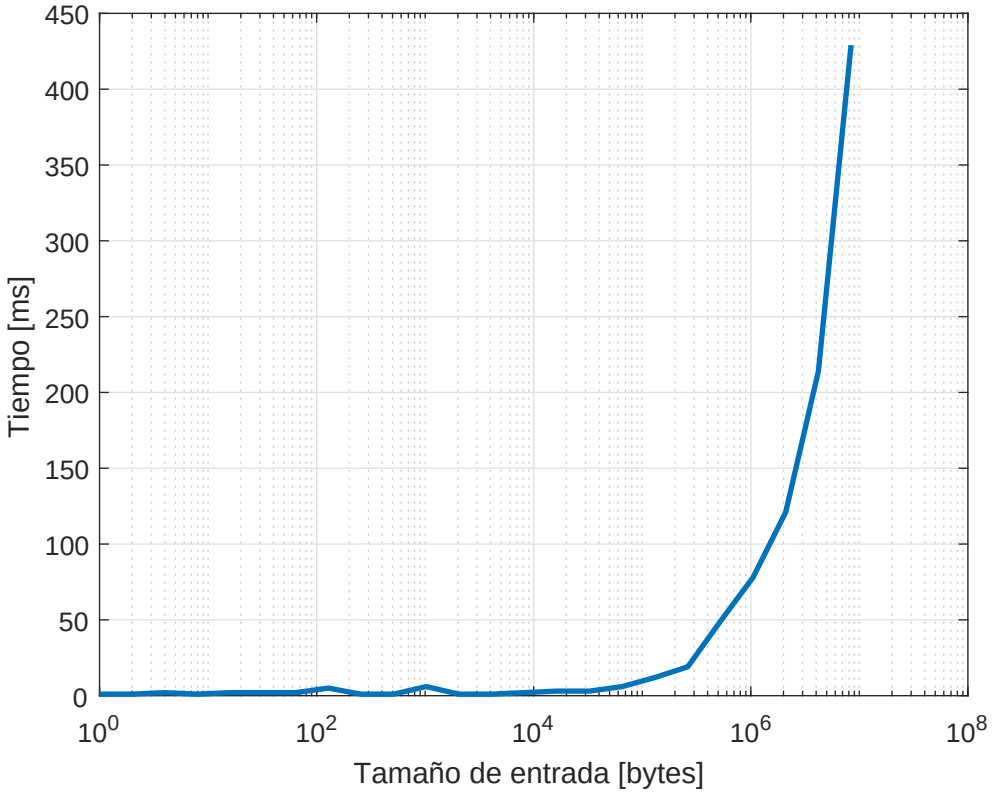

Figura 5.1: Tiempos de ejecución para codificación.

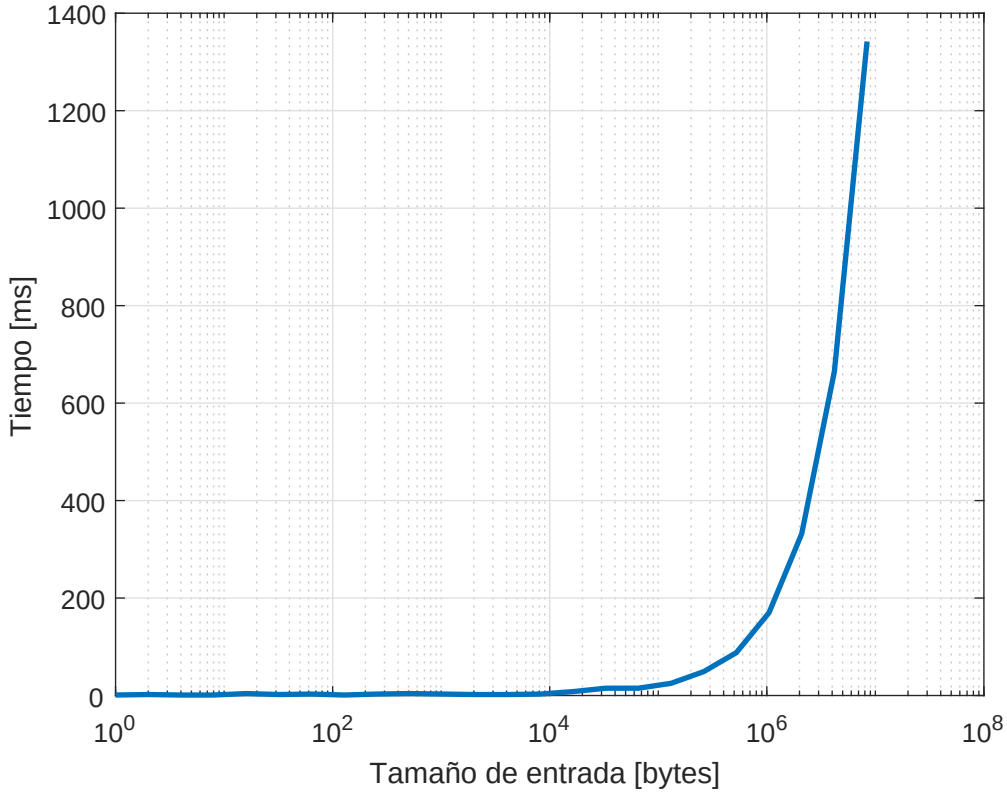

Figura 5.2: Tiempos de ejecución para decodificación.

6. Herramientas de hardware y software utilizadas

La computadora utilizada para realizar el desarrollo y las pruebas tiene las siguientes especificaciones:

- Procesador: Intel i3-6100.
- Memoria: 8GB RAM DDR4.
- Almacenamiento: Disco magnético de 200GB de 7200RPM.

Los programas se desarrollaron en el sistema operativo Linux Ubuntu, cuyos datos de distribución son

```
Distributor ID: Ubuntu
Description: Ubuntu 16.04.2 LTS
Release: 16.04
Codename: xenial
```

Además, se utilizaron las siguientes herramientas:

- Compilador del proyecto: GNU gcc (Ubuntu 5.4.0-6ubuntu1 16.04.4) 5.4.0 20160609 [7].
- Control del proceso de compilación: GNU Make 4.1 [6].
- Compilador del presente informe: Latex pdfTeX 3.14159265-2.6-1.40.16 (TeX Live 2015/Debian) [8].
- Edición de código fuente: VIM - Vi IMproved 7.4 (2013 Aug 10, compiled Nov 24 2016 16:44:48) y Atom 1.15.0 [9] [10].
- Depuración del programa: gdb [11].

7. Conclusiones

Se creó un programa en lenguaje C que permite codificar y decodificar información en formato base 64, cumpliendo con las especificaciones del enunciado del trabajo. Se describió el flujo del programa y las funciones y estructuras más importantes del mismo, junto con el manejo de errores. Además, se describió el modo de compilación del programa utilizando las herramientas `gcc` y `make`.

Por otro lado, se crearon casos de prueba de caja negra para verificar el correcto funcionamiento del programa. Las mismas cubren tests de validación de opciones del programa ingresando combinaciones de opciones válidas e inválidas para determinar si este es capaz de detectar errores. Además, se crearon pruebas de codificación y decodificación de archivos, tamaños máximos de líneas, y tiempos de ejecución.

De esta forma se deja lista la infraestructura básica para los siguientes proyectos donde se trabajará con la arquitectura MIPS32 y el consecuente análisis y desarrollo de código assembly MIPS32.

Referencias

[1] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.

[2] Patton, R. - *Software Testing* - 2nd edition - Sams Indianapolis, IN, USA 2005.

[3] *Apuntes del curso 66.20 Organización de Computadoras* - Cátedra Hamkalo - Facultad de Ingeniería de la Universidad de Buenos Aires.

[4] *RFC 2045* - https://tools.ietf.org/html/rfc2045#section-6.8

[5] *Base 64* - https://en.wikipedia.org/wiki/Base64

[6] *GNU Make* - https://www.gnu.org/software/make/

[7] *GNU Gcc* - https://gcc.gnu.org/

[8] LaTeX- https://www.latex-project.org/

[9] VIM - https://vim.sourceforge.io/

[10] Atom - https://atom.io/

[11] GNU gdb - https://www.gnu.org/software/gdb/

A. Makefile

A.0.1. makefile

```
# ----------------------------------------------------------------
# @Title:   FIUBA - 66.20 Organizacion de Computadoras.
# @Project: TP0 - Infraestructura basica.
# ----------------------------------------------------------------
# @Filename: makefile
# ----------------------------------------------------------------
# @Authors:
#    Husain, Ignacio Santiago.
#          santiago.husain at gmail dot com
#        Pesado, Lucia.
#               luupesado at gmail dot com
#    Verstraeten, Federico.
#          federico.verstraeten at gmail dot com
#
# @Date:               07-Sep-2018 12:57:50 pm
# @Last modified by:   Ignacio Santiago Husain
# @Last modified time: 24-Sep-2018 8:07:37 am
#
# @Copyright (C):
#    This file is part of 'TP0 - Infraestructura basica.'.
#    Unauthorized copying or use of this file via any medium
#    is strictly prohibited.
# ----------------------------------------------------------------
#
# The source files must have .c extension.
# The object code must have .o extension.
# The header files must have .h extension.
#
# To compile, execute 'make'.
# To clean all the compilation files, issue the command
# 'make clean'.
#
# ----------------------------------------------------------------
# List all the header and object files separated by a blank
# space.
_DEPS = messages.h common.h parser.h encoder.h decoder.h
_SRC = main.c parser.c encoder.c decoder.c
_OBJ = main.o parser.o encoder.o decoder.o
# ----------------------------------------------------------------
# Configuration.
CC = gcc
CFLAGS = -ansi -Wall -I. -O0
```

```
43 OUTPUT1 = tp0
44 # ------------------------------------------------------------
45 all: $(OUTPUT1)
46
47 $(OUTPUT1): $(_OBJ)
48     $(CC) $(CFLAGS) -o $(OUTPUT1) $(_OBJ)
49
50 assembly:
51     $(CC) $(CFLAGS) -S $(_SRC) $(ARGS)
52
53 .PHONY: clean assembly
54
55 clean:
56     rm -f ./*.o *~ core ./*~ ./*.s
57     rm -f $(OUTPUT1)
```

B. Tests

B.0.1. runTests.sh

```
#!/bin/bash

# ------------------------------------------------------------
# @Title:   FIUBA - 66.20 Organizacion de Computadoras.
# @Project: TP0 - Infraestructura basica.
# ------------------------------------------------------------
# @Filename: runTests.sh
# ------------------------------------------------------------
# @Authors:
#    Husain, Ignacio Santiago.
#          santiago.husain at gmail dot com
#        Pesado, Lucia.
#            luupesado at gmail dot com
#    Verstraeten, Federico.
#          federico.verstraeten at gmail dot com
#
# @Date:               07-Sep-2018 2:12:07 pm
# @Last modified by:   Ignacio Santiago Husain
# @Last modified time: 24-Sep-2018 12:16:29 pm
#
# @Copyright (C):
#    This file is part of 'TP0 - Infraestructura basica.'.
#    Unauthorized copying or use of this file via any medium
#    is strictly prohibited.
# ------------------------------------------------------------
#
# Script to test errors in the program arguments.
#
# To remove newlines from a textfile, use
# printf %s "$(cat file)" > file
#
# To print contents of a file, including control characters, ↩
   do:
# oc -c file
#
# ------------------------------------------------------------

# Program name to test.
PROGRAM_NAME='./tp0'

# Failed tests counter.
failedTests=0;
```

```
42
43 # Colors to be used.
44 RED="\e[31m";
45 GREEN="\e[32m";
46 CYAN="\e[96m";
47 YELLOW="\e[93m";
48 BOLD="\033[1m";
49 DEFAULT="\e[0m";
50
51 # Helper and formatting functions definitions.
52 function header() {
53   echo -e "$CYAN↩
       ------------------------------------------------------↩
       $DEFAULT";
54   echo -e "$CYAN$1$DEFAULT";
55   echo -e "$CYAN↩
       ------------------------------------------------------↩
       $DEFAULT";
56 }
57
58 function msg_true() {
59   echo -e "$GREEN\0PASSED \n $DEFAULT PROGRAM OUTPUT:\n\t$1";
60 }
61
62 function msg_false() {
63   echo -e "$RED\0FAILED \n $DEFAULT PROGRAM OUTPUT:\n\t$1";
64 }
65
66 function msg_testing() {
67   echo -e "Testing: $BOLD$1$DEFAULT";
68 }
69
70 function success_msg() {
71   echo -e "  $GREEN$1$DEFAULT";
72 }
73
74 function error_msg() {
75   echo -e "  $RED$1$DEFAULT";
76 }
77
78 # ------------------------------------------------------------
79 # Input parameters tests.
80 # ------------------------------------------------------------
81
82 # Define the expected outputs of each of the test cases with ↩
     its associated
83 # test functions.
```

```
84  EXPECTED_OUTPUT_INEXISTENT_INPUT_STREAM=("ERROR: Can't open ↩
       input stream.")
85
86  function test1_parameter_input_inexistent_stream(){
87    header "TEST1: inexistent 'input' stream."
88
89      commands=(
90    "-i 1"
91    "-i hola.bin"
92    )
93
94      for i in "${commands[@]}"
95      do
96
97          msg_testing "$PROGRAM_NAME $i"
98
99          PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
100
101     if [[ "$EXPECTED_OUTPUT_INEXISTENT_INPUT_STREAM" == "↩
           $PROGRAM_OUTPUT" ]]; then
102       msg_true "$PROGRAM_OUTPUT"
103     else
104       msg_false "$PROGRAM_OUTPUT"
105       failedTests=$(($failedTests+1));
106     fi
107
108     done
109 }
110
111 EXPECTED_OUTPUT_INPUT_NO_ARGUMENT=("./tp0: option requires an ↩
       argument -- 'i'")
112
113 function test11_parameter_input_no_argument(){
114   header "TEST11: no 'input' option parameters."
115
116     commands=("-i ")
117
118     for i in "${commands[@]}"
119     do
120
121         msg_testing "$PROGRAM_NAME $i"
122
123         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
124
125     if [[ "$EXPECTED_OUTPUT_INPUT_NO_ARGUMENT" == "↩
           $PROGRAM_OUTPUT" ]]; then
126       msg_true "$PROGRAM_OUTPUT"
```

```
127     else
128       msg_false "$PROGRAM_OUTPUT"
129       failedTests=$(($failedTests+1));
130     fi
131
132     done
133 }
134
135 EXPECTED_OUTPUT_INPUT_INVALID_STREAM=("ERROR: Invalid input ↩
    stream.")
136
137 function test12_parameter_input_invalid_stream(){
138   header "TEST12: invalid 'input' stream."
139
140     commands=(
141   "-i ."
142   "-i .."
143   "-i /"
144   "-i //"
145   )
146
147     for i in "${commands[@]}"
148     do
149
150         msg_testing "$PROGRAM_NAME $i"
151
152         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
153
154     if [[ "$EXPECTED_OUTPUT_INPUT_INVALID_STREAM" == "↩
        $PROGRAM_OUTPUT" ]]; then
155       msg_true "$PROGRAM_OUTPUT"
156     else
157       msg_false "$PROGRAM_OUTPUT"
158       failedTests=$(($failedTests+1));
159     fi
160
161     done
162 }
163
164 EXPECTED_OUTPUT_INVALID_OUTPUT_STREAM=("ERROR: Invalid output ↩
    stream.")
165
166 function test2_parameter_output_stream(){
167   header "TEST2: invalid 'output' stream."
168
169     commands=(
170   "-i ../tests/leviathan.input -o ."
```

```
171   "-i ../tests/leviathan.input -o .."
172   "-i ../tests/leviathan.input -o /"
173   "-i ../tests/leviathan.input -o //"
174   )
175
176     for i in "${commands[@]}"
177     do
178
179         msg_testing "$PROGRAM_NAME $i"
180
181         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
182
183     if [[ "$EXPECTED_OUTPUT_INVALID_OUTPUT_STREAM" == "↩
        $PROGRAM_OUTPUT" ]]; then
184       msg_true "$PROGRAM_OUTPUT"
185     else
186       msg_false "$PROGRAM_OUTPUT"
187       failedTests=$(($failedTests+1));
188     fi
189
190     done
191 }
192
193 EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT=("./tp0: option requires an↩
    argument -- 'o'")
194
195 function test21_parameter_output_no_argument(){
196   header "TEST21: no 'output' option parameters."
197
198     commands=("-o ")
199
200     for i in "${commands[@]}"
201     do
202
203         msg_testing "$PROGRAM_NAME $i"
204
205         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
206
207     if [[ "$EXPECTED_OUTPUT_OUTPUT_NO_ARGUMENT" == "↩
        $PROGRAM_OUTPUT" ]]; then
208       msg_true "$PROGRAM_OUTPUT"
209     else
210       msg_false "$PROGRAM_OUTPUT"
211       failedTests=$(($failedTests+1));
212     fi
213
214     done
```

```
215 }
216
217 EXPECTED_OUTPUT_INVALID_ACTION=("ERROR: Invalid action ↩
      argument.")
218
219 function test3_parameter_action(){
220   header "TEST3: invalid 'action' parameters."
221
222     commands=(
223   "-a bad_action"
224   "-a 1"
225   "-a ."
226   "-a .."
227   "-a /"
228   "-a //"
229   "-a $"
230   )
231
232     for i in "${commands[@]}"
233     do
234
235         msg_testing "$PROGRAM_NAME $i"
236
237         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
238
239     if [[ "$EXPECTED_OUTPUT_INVALID_ACTION" == "↩
          $PROGRAM_OUTPUT" ]]; then
240       msg_true "$PROGRAM_OUTPUT"
241     else
242       msg_false "$PROGRAM_OUTPUT"
243       failedTests=$(($failedTests+1));
244     fi
245
246     done
247 }
248
249 EXPECTED_OUTPUT_ACTION_NO_ARGUMENT=("./tp0: option requires an↩
      argument -- 'a'")
250
251 function test31_parameter_action_no_argument(){
252   header "TEST31: no 'action' option parameters."
253
254     commands=("-a ")
255
256     for i in "${commands[@]}"
257     do
258
```

```
259         msg_testing "$PROGRAM_NAME $i"
260 
261         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
262 
263     if [[ "$EXPECTED_OUTPUT_ACTION_NO_ARGUMENT" == "↩
        $PROGRAM_OUTPUT" ]]; then
264       msg_true "$PROGRAM_OUTPUT"
265     else
266       msg_false "$PROGRAM_OUTPUT"
267       failedTests=$(($failedTests+1));
268     fi
269 
270     done
271 }
272 
273 EXPECTED_OUTPUT_VALID_PARAMETERS=()
274 
275 function test4_valid_parameters(){
276   header "TEST4: all options with correct parameters."
277 
278     commands=(
279   "-a encode -i ../tests/test1.bin -o ../tests/test1_out.bin"
280   "-a decode -i ../tests/test1_out.bin -o ../tests/test2.bin")
281 
282     for i in "${commands[@]}"
283     do
284 
285         msg_testing "$PROGRAM_NAME $i"
286 
287         PROGRAM_OUTPUT=$($PROGRAM_NAME $i 2>&1)
288 
289 
290     if [[ "$EXPECTED_OUTPUT_VALID_PARAMETERS" == "↩
        $PROGRAM_OUTPUT" ]]; then
291       msg_true "$PROGRAM_OUTPUT"
292     else
293       msg_false "$PROGRAM_OUTPUT"
294       failedTests=$(($failedTests+1));
295     fi
296 
297     done
298 }
299 
300 # ------------------------------------------------------------
301 # Input-output validation tests.
302 # ------------------------------------------------------------
303 function IO_validation_passed(){
```

```
304     echo -e "$GREEN\0PASSED: $DEFAULT $1"
305 }
306 function IO_validation_failed(){
307     echo -e "$RED\0FAILED $DEFAULT $1"
308 }
309
310 TESTS_DIR="../tests";
311 mkdir $TESTS_DIR;
312
313 function test5_IO_validation(){
314   header "TEST5: input-output should be the same."
315
316   n=1;
317   nLimit=$((1024*1000));
318
319   while [ $n -le $nLimit ]
320   do
321      head -c $n </dev/urandom >$TESTS_DIR/in.bin;
322     $PROGRAM_NAME -a encode -i $TESTS_DIR/in.bin -o $TESTS_DIR↩
          /out.b64;
323     $PROGRAM_NAME -a decode -i $TESTS_DIR/out.b64 -o ↩
          $TESTS_DIR/out.bin;
324
325     diff_result="$(diff $TESTS_DIR/in.bin $TESTS_DIR/out.bin)"↩
          ;
326
327     if [[ -z ${diff_result} ]]; then :;
328         IO_validation_passed "n = $n";
329     else
330         IO_validation_failed "n = $n";
331       error_msg "in.bin";
332       cat $TESTS_DIR/in.bin | od -A x -t x1z -v;
333       error_msg "out.b64";
334       cat $TESTS_DIR/out.b64 | od -A x -t x1z -v;
335       error_msg "out.bin";
336       cat $TESTS_DIR/out.bin | od -A x -t x1z -v;
337       failedTests=$(($failedTests+1));
338         break;
339     fi
340
341     n=$(($n*2));
342
343     rm -f $TESTS_DIR/in.bin $TESTS_DIR/out.b64 $TESTS_DIR/out.↩
          bin
344   done
345 }
346
```

```
347 function test51_IO_validation(){
348   header "TEST51: input known text with known encoding."
349
350   $PROGRAM_NAME -a encode -i $TESTS_DIR/leviathan.input -o ↩
        $TESTS_DIR/leviathan_out.b64;
351
352   diff_result="$(diff $TESTS_DIR/leviathan_out.b64 $TESTS_DIR/↩
        leviathan_out_truth.b64)";
353
354   if [[ -z ${diff_result} ]]; then :;
355     IO_validation_passed "No differences.";
356   else
357     IO_validation_failed "Differences: \n${diff_result}";
358     failedTests=$(($failedTests+1));
359   fi
360 }
361
362 function test52_IO_validation(){
363   header "TEST52: Encode letter 'M'."
364
365   program_output="$(echo -n M | $PROGRAM_NAME)";
366   correct_output="TQ==";
367   diff_result="$(diff  <(echo "$program_output" ) <(echo "↩
        $correct_output"))";
368
369   if [[ -z ${diff_result} ]]; then :;
370     IO_validation_passed "No differences.";
371   else
372     IO_validation_failed "Differences: \n${diff_result}";
373     failedTests=$(($failedTests+1));
374   fi
375 }
376
377 function test53_IO_validation(){
378   header "TEST53: Encode 'Ma'."
379
380   program_output="$(echo -n Ma | $PROGRAM_NAME)";
381   correct_output="TWE=";
382   diff_result="$(diff  <(echo "$program_output" ) <(echo "↩
        $correct_output"))";
383
384   if [[ -z ${diff_result} ]]; then :;
385     IO_validation_passed "No differences.";
386   else
387     IO_validation_failed "Differences: \n${diff_result}";
388     failedTests=$(($failedTests+1));
389   fi
```

```
390 }
391
392 function test54_IO_validation(){
393   header "TEST54: Encode 'Man'."
394
395   program_output="$(echo -n Man | $PROGRAM_NAME)";
396   correct_output="TWFu";
397   diff_result="$(diff  <(echo "$program_output" ) <(echo "↩
        $correct_output"))";
398
399   if [[ -z ${diff_result} ]]; then :;
400     IO_validation_passed "No differences.";
401   else
402     IO_validation_failed "Differences: \n${diff_result}";
403     failedTests=$(($failedTests+1));
404   fi
405 }
406
407 function test55_IO_validation(){
408   header "TEST55: Encode and decode 'Man'."
409
410   program_output="$(echo Man | $PROGRAM_NAME | $PROGRAM_NAME -↩
        a decode)";
411   correct_output="Man";
412   diff_result="$(diff  <(echo "$program_output" ) <(echo "↩
        $correct_output"))";
413
414   if [[ -z ${diff_result} ]]; then :;
415     IO_validation_passed "No differences.";
416   else
417     IO_validation_failed "Difference↩
          ----------------------------------------------------
418 Test suite ended.
419 ----------------------------------------------------
420   All tests passed.
421 s: \n${diff_result}";
422     failedTests=$(($failedTests+1));
423   fi
424 }
425
426 function test56_IO_validation(){
427   header "TEST56: Check bit by bit."
428
429   program_output="$(echo -E xyz | $PROGRAM_NAME | ↩
        $PROGRAM_NAME -a decode | od -t c)";
430   correct_output="0000000   x   y   z  \n
431 0000004";
```

```
432   diff_result="$(diff  <(echo -E "$program_output" ) <(echo -E↩
          "$correct_output"))";
433
434   if [[ -z ${diff_result} ]]; then :;
435     IO_validation_passed "No differences.";
436   else
437     IO_validation_failed "Differences: \n${diff_result}";
438     failedTests=$(($failedTests+1));
439   fi
440 }
441
442 function test57_IO_validation(){
443   header "TEST57: Check max line length and number of encoded ↩
          bytes."
444
445   program_output_line_count="$(echo -n "$(yes | head -c 1024 |↩
          $PROGRAM_NAME -a encode)" | wc -l)";
446
447   # 1024 bytes[base256] => (8190+2) bits => 1365 bytes[base64]↩
          + 2 bits
448   # 1365 bytes[base64] + 2 bits + '==' =>  1366 bytes[base64]
449   # floor(1366 bytes[base64] / 76 charEachLine) => 17 lines
450   correct_output_line_count="17";
451   diff_result_line_count="$(diff  <(echo "↩
          $program_output_line_count" ) <(echo "↩
          $correct_output_line_count"))";
452
453   if [[ -z ${diff_result_line_count} ]]; then :;
454     IO_validation_passed "No differences in line count.";
455   else
456     IO_validation_failed "Differences in line count:
457     Program output:${program_output_line_count}
458     Correct output:${correct_output_line_count}";
459     failedTests=$(($failedTests+1));
460   fi
461
462   program_output_word_count="$(yes | head -c 1024 | ↩
          $PROGRAM_NAME -a encode | $PROGRAM_NAME -a decode | wc -c↩
          )";
463   correct_output_word_count="1024";
464   diff_result_word_count="$(diff  <(echo "↩
          $program_output_word_count" ) <(echo "↩
          $correct_output_word_count"))";
465
466   if [[ -z ${diff_result_word_count} ]]; then :;
467     IO_validation_passed "No differences in word count.";
468   else
```

```
469     IO_validation_failed "Differences in word count:
470     Program output:${program_output_word_count}
471     Correct output:${correct_output_word_count}";
472     failedTests=$(($failedTests+1));
473   fi
474 }
475
476 EXPECTED_OUTPUT_INVALID_DECODE_INPUT=("ERROR: Character is not in Base64 Table.")
477 function test58_IO_validation(){
478   header "TEST58: decoding input with chars not in B64 table should produce error."
479
480   n=1000;
481
482     head -c $n </dev/urandom >$TESTS_DIR/in_notvalid_b64.b64;
483     PROGRAM_OUTPUT=$($PROGRAM_NAME -a decode -i $TESTS_DIR/in_notvalid_b64.b64 -o $TESTS_DIR/out.bin 2>&1)
484
485     if [[ "$EXPECTED_OUTPUT_INVALID_DECODE_INPUT" == "$PROGRAM_OUTPUT" ]]; then
486       msg_true "$PROGRAM_OUTPUT"
487     else
488       msg_false "$PROGRAM_OUTPUT"
489       failedTests=$(($failedTests+1));
490     fi
491
492     rm -f $TESTS_DIR/in_notvalid_b64.b64 $TESTS_DIR/out.bin
493 }
494
495 # ------------------------------------------------------------
496 # Encoding-Decoding execution times tests.
497 # ------------------------------------------------------------
498 function test6_encoding_execution_times(){
499   header "TEST6: encoding execution times."
500
501   n=1;
502   nLimit=$((1024*1000));
503
504   while [ $n -le $nLimit ]
505   do
506     head -c $n </dev/urandom >$TESTS_DIR/in.bin;
507     ts=$(date +%s%N);
508     $PROGRAM_NAME -a encode -i $TESTS_DIR/in.bin -o $TESTS_DIR/out.b64;
509     tt=$((($(date +%s%N) - $ts)/1000000));
510
```

```
511     printf 'n: %-10d %10s %.2f [ms]\n' "$n" " " "$tt"
512     printf '%-10d %10s %.2f\n' "$n" " " "$tt" >> ../tests/↩
            encodingTimes.txt
513 
514     n=$((n*2));
515 
516     rm -f $TESTS_DIR/in.bin $TESTS_DIR/out.b64 $TESTS_DIR/out.↩
            bin
517   done
518 }
519 
520 function test7_decoding_execution_times(){
521   header "TEST7: decoding execution times."
522 
523   n=1;
524   nLimit=$((1024*1000));
525 
526   while [ $n -le $nLimit ]
527   do
528     head -c $n </dev/urandom >$TESTS_DIR/in.bin;
529     $PROGRAM_NAME -a encode -i $TESTS_DIR/in.bin -o $TESTS_DIR↩
            /out.b64;
530     ts=$(date +%s%N);
531     $PROGRAM_NAME -a decode -i $TESTS_DIR/out.b64 -o ↩
            $TESTS_DIR/out.bin;
532     tt=$((($(date +%s%N) - $ts)/1000000));
533 
534     printf 'n: %-10d %10s %.2f [ms]\n' "$n" " " "$tt"
535     printf '%-10d %10s %.2f\n' "$n" " " "$tt" >> ../tests/↩
            decodingTimes.txt
536 
537     n=$((n*2));
538 
539     rm -f $TESTS_DIR/in.bin $TESTS_DIR/out.b64 $TESTS_DIR/out.↩
            bin
540   done
541 }
542 
543 # ------------------------------------------------------------
544 # Run the tests.
545 # ------------------------------------------------------------
546 test1_parameter_input_inexistent_stream
547 test11_parameter_input_no_argument
548 test12_parameter_input_invalid_stream
549 test2_parameter_output_stream
550 test21_parameter_output_no_argument
551 test3_parameter_action
```

```
552 test31_parameter_action_no_argument
553 test4_valid_parameters
554 test5_IO_validation
555 test51_IO_validation
556 test52_IO_validation
557 test53_IO_validation
558 test54_IO_validation
559 test55_IO_validation
560 test56_IO_validation
561 test57_IO_validation
562 test58_IO_validation
563 test6_encoding_execution_times
564 test7_decoding_execution_times
565
566 header "Test suite ended."
567
568 if [[ $failedTests -eq $zero ]]; then
569   success_msg "All tests passed.";
570 else
571   error_msg "Failed tests: $failedTests";
572 fi
```

C. Código fuente

C.0.1. main.c

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: main.c
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               07-Sep-2018 3:46:28 pm
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 10:48:11 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Program entry point.

------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "common.h"
#include "decoder.h"
#include "encoder.h"
#include "parser.h"

int main(int argc, char **argv)
{
  params_t params;

  /* We parse the command line and check for errors. */
```

```
42    outputCode cmdParsingState = parseCmdline(argc, argv, &↩
          params);
43    if (cmdParsingState == outERROR)
44    {
45      exit(EXIT_FAILURE);
46    }
47
48    outputCode transformationState;
49    if (strcmp(params.action, ENCODE_STR_TOKEN) == 0)
50    {
51      transformationState = encode(&params);
52    }
53    else
54    {
55      transformationState = decode(&params);
56    }
57    if (transformationState == outERROR)
58    {
59      exit(EXIT_FAILURE);
60    }
61
62    /* Close and free what is left. */
63    fclose(params.inputStream);
64    fclose(params.outputStream);
65
66    return EXIT_SUCCESS;
67  }
```

C.0.2. common.h

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: common.h
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:36:31 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 18-Sep-2018 3:28:10 pm

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Common program structures.

------------------------------------------------------------ */
#ifndef COMMON__H
#define COMMON__H

#ifndef VERSION
#define VERSION "1.0.0"
#endif

#define SIZETABLEB64 64

static const char translationTableB64[SIZETABLEB64] =
    "↩
       ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789↩
       +/";

typedef struct params_t
{
  char *action;
```

```
43    FILE *inputStream;
44    FILE *outputStream;
45  } params_t;
46
47  typedef enum outputCodes_ { outOK, outERROR } outputCode;
48
49  #endif
```

C.0.3. decoder.h

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: decoder.h
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:53:31 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 11:27:38 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Decoder definitions and declarations.

------------------------------------------------------------ */
#ifndef DECODER__H
#define DECODER__H

#include <stdio.h>
#include <string.h>
#include "common.h"
#include "messages.h"

#define DECODER_MASK 0xFF000000
#define B64_CHARS_PER_BLOCK 4
#define PADDING_DEC '='
#define PADD_INDEX 0
#define OUTPUT_BLOCK_SIZE 3
#define BITS_PER_BYTE 8

outputCode base64ToBase256(unsigned char *outBlock, unsigned ↩
   char *inBlock,
```

```
44                              unsigned char *decCount);
45 outputCode decode(params_t *params);
46
47 #endif
```

C.0.4. decoder.c

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: decoder.c
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:21:30 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 11:27:31 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Decoder implementation.

------------------------------------------------------------ */
#include "decoder.h"

outputCode base64ToBase256(unsigned char *outBlock, unsigned char *inBlock,
                           unsigned char *decCount)
{
  unsigned int bitMask = DECODER_MASK;
  unsigned char index1 = 0, index2 = 0;
  unsigned char indexTable[B64_CHARS_PER_BLOCK] = {0, 0, 0, 0};
  unsigned int charHolder = 0;
  /* Variable to hold the output block of 4 bytes. */
  unsigned int bitPattern = 0;
  unsigned char accumBit = 0;

  /* Search char index in translationTableB64 */
  for (index1 = 0; index1 < B64_CHARS_PER_BLOCK; index1++)
```

```
43  {
44    if (inBlock[index1] == PADDING_DEC)
45    {
46      indexTable[index1] = PADD_INDEX;
47      continue;
48    }
49
50    for (index2 = 0; index2 < SIZETABLEB64; ++index2)
51    {
52      if (inBlock[index1] == translationTableB64[index2])
53      {
54        indexTable[index1] = index2;
55        (*decCount)++;
56        break;
57      }
58    }
59    if (index2 >= SIZETABLEB64)
60    {
61      fprintf(stderr, ERROR_B64_CHAR_NOT_FOUND_MSG);
62      return outERROR;
63    }
64  }
65
66  /* bitPattern generation using indexTable */
67  for (index1 = 0; index1 < B64_CHARS_PER_BLOCK; index1++)
68  {
69    accumBit += 2;
70    charHolder = (unsigned int)indexTable[index1];
71    charHolder <<= (B64_CHARS_PER_BLOCK - 1 - index1) * sizeof↩
       (unsigned char) *
72                  BITS_PER_BYTE;
73    charHolder <<= accumBit;
74    bitPattern = (bitPattern | charHolder);
75  }
76
77  index1 = 0;
78  do
79  {
80    /* Extract the decoded character from bitPattern */
81    charHolder = (bitPattern & bitMask);
82
83    /* Shift right the decoded character to the correct ↩
       position. */
84    charHolder >>= (B64_CHARS_PER_BLOCK - 1 - index1) * sizeof↩
       (unsigned char) *
85                  BITS_PER_BYTE;
86
```

```
87      /* Store in outBlock */
88      outBlock[index1] = (unsigned char)charHolder;
89
90      /* Shift right 0,8,16...bits the bitMask */
91      bitMask >>= sizeof(unsigned char) * BITS_PER_BYTE;
92      index1++;
93    } while (index1 < OUTPUT_BLOCK_SIZE);
94
95    return outOK;
96  }
97
98  outputCode decode(params_t *params)
99  {
100   unsigned char readChar;
101   unsigned char inBlock[B64_CHARS_PER_BLOCK] = {};
102   unsigned char outBlock[OUTPUT_BLOCK_SIZE] = {};
103   unsigned char index1, index2 = 0;
104   unsigned char decodedCharsCount = 0;
105
106   while (1)
107   {
108     decodedCharsCount = 0;
109
110     /* Load input buffer */
111     for (index1 = 0; index1 < B64_CHARS_PER_BLOCK; ++index1)
112     {
113       readChar = getc(params->inputStream);
114       if (ferror(params->inputStream))
115       {
116         fprintf(stderr, ERROR_INPUT_STREAM_READING_MSG);
117         return outERROR;
118       }
119
120       /* Discard detected whitespaces. */
121       if (readChar == '\n' || readChar == '\t' || readChar == ↩
            ' ')
122       {
123         index1--;
124         continue;
125       }
126       else
127       {
128         inBlock[index1] = readChar;
129       }
130
131       /* EOF */
132       if (feof(params->inputStream))
```

```
133       {
134         /* If there are still chars in the buffer, we flush it. */
135         if (index1 != 0)
136         {
137           if (base64ToBase256(outBlock, inBlock, &decodedCharsCount) ==
138               outERROR)
139           {
140             return outERROR;
141           }
142
143           for (index2 = 0; index2 < decodedCharsCount - 1; ++index2)
144           {
145             fputc(outBlock[index2], params->outputStream);
146           }
147           if (ferror(params->outputStream))
148           {
149             fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);
150             return outERROR;
151           }
152         }
153         return outOK;
154       }
155     }
156
157     /* Translate inBlock into base256 */
158     if (base64ToBase256(outBlock, inBlock, &decodedCharsCount) == outERROR)
159     {
160       return outERROR;
161     }
162
163     for (index2 = 0; index2 < decodedCharsCount - 1; ++index2)
164     {
165       fputc(outBlock[index2], params->outputStream);
166     }
167
168     if (ferror(params->outputStream))
169     {
170       fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);
171       return outERROR;
172     }
173   }
174
175   return outERROR;
```

```
176 }
```

C.0.5. encoder.h

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: encoder.h
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:52:58 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 11:29:36 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Encoder definitions and declarations.

------------------------------------------------------------ */
#ifndef ENCODER__H
#define ENCODER__H

#include <stdio.h>
#include <string.h>
#include "common.h"
#include "messages.h"

#define ENCODER_MASK 0xFC
#define TAIL_MAX_BITS_TO_SHIFT 6
#define PADDING "="
#define ENCODER_OUTPUT_CHARS 4
#define MAX_LINE_LENGHT 76

unsigned char base256ToBase64(char *outChar, unsigned int ↩
   inChar);
outputCode encode(params_t *params);
```

```
44
45 #endif
```

C.0.6. encoder.c

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: encoder.c
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:21:26 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 11:29:21 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Encoder implementation.

------------------------------------------------------------ */
#include "encoder.h"

unsigned char base256ToBase64(char *outBlock, unsigned int ↩
   inChar)
{
  unsigned char headByte = 0, prevByte = 0;
  static unsigned char tailByte = 0;
  static unsigned char bitMask = ENCODER_MASK;
  static unsigned int shiftRightBit = 2;
  unsigned char encodedCharsCount = 0;

  /* Backup the previous tailByte. */
  prevByte = tailByte;

  /* Padding: The last encoded block contains less than 6 bits↩
     . */
  if ((inChar == EOF))
```

```
43  {
44    if (shiftRightBit == 6)
45    {
46      headByte = (prevByte | 0);
47      strncpy(outBlock, &translationTableB64[headByte], 1);
48      strncat(outBlock, PADDING, 1);
49      return (encodedCharsCount + 2);
50    }
51    else if (shiftRightBit == 4)
52    {
53      headByte = (prevByte | 0);
54      strncpy(outBlock, &translationTableB64[headByte], 1);
55      strncat(outBlock, PADDING, 1);
56      strncat(outBlock, PADDING, 1);
57      return (encodedCharsCount + 3);
58    }
59    else
60      return encodedCharsCount;
61  }
62
63  /* Save the head of input char. */
64  headByte = inChar & bitMask;
65
66  /* Shift right 2, 4 or 6bit the headByte. */
67  headByte >>= shiftRightBit;
68
69  /* Save the tail input char. */
70  tailByte = inChar & (~bitMask);
71
72  /* Shift left 4, 2 or 0bit the tailByte. */
73  tailByte <<= (TAIL_MAX_BITS_TO_SHIFT - shiftRightBit);
74
75  /* Merge previous tailByte and current headByte. */
76  headByte = (prevByte | headByte);
77
78  /*Print translation in outBlock*/
79  strncpy(outBlock, &translationTableB64[headByte], 1);
80  encodedCharsCount++;
81
82  shiftRightBit += 2;
83
84  /* Shift left 2 bits the mask. */
85  if (!(bitMask <<= 2))
86  {
87    /* Restart mask at the beginning. */
88    bitMask = ENCODER_MASK;
89    shiftRightBit = 2;
```

```
90
91      /* Print tailByte and clear. */
92      strncat(outBlock, &translationTableB64[tailByte], 1);
93      encodedCharsCount++;
94
95      tailByte = 0;
96    }
97
98    return encodedCharsCount;
99  }
100
101 outputCode encode(params_t *params)
102 {
103   unsigned int inChar;
104   char outBlock[ENCODER_OUTPUT_CHARS] = {};
105   unsigned char totalEncodedCharsCount = 0, encodedCharsCount ↩
        = 0;
106
107   do
108   {
109     /* Clear outBlock. */
110     memset(outBlock, 0, sizeof(outBlock));
111
112     /* Read inputStream and store as integer. */
113     inChar = getc(params->inputStream);
114
115     if (ferror(params->inputStream))
116     {
117       fprintf(stderr, ERROR_INPUT_STREAM_READING_MSG);
118       return outERROR;
119     }
120
121     /* Encoding to Base64. */
122     encodedCharsCount = base256ToBase64(outBlock, inChar);
123
124     if ((totalEncodedCharsCount + encodedCharsCount) <= ↩
          MAX_LINE_LENGHT)
125     {
126       totalEncodedCharsCount += encodedCharsCount;
127     }
128     else
129     {
130       fputs("\n", params->outputStream);
131       if (ferror(params->outputStream))
132       {
133         fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);
134         return outERROR;
```

```
135         }
136         totalEncodedCharsCount = encodedCharsCount;
137       }
138
139       /* Print output stream. */
140       fputs(outBlock, params->outputStream);
141       if (ferror(params->outputStream))
142       {
143         fprintf(stderr, ERROR_OUTPUT_STREAM_WRITING_MSG);
144         return outERROR;
145       }
146     } while (inChar != EOF);
147
148     return outOK;
149 }
```

C.0.7. parser.h

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: parser.h
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:50:53 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 10:50:34 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

CLA parser definitions and declarations.

------------------------------------------------------------ */
#ifndef PARSER__H
#define PARSER__H

#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "common.h"
#include "messages.h"

#ifndef STD_STREAM_TOKEN
#define STD_STREAM_TOKEN "-"
#endif

/* Actions definitions */
#ifndef ENCODE_STR_TOKEN
#define ENCODE_STR_TOKEN "encode"
```

```
45 #endif
46 #ifndef DECODE_STR_TOKEN
47 #define DECODE_STR_TOKEN "decode"
48 #endif
49
50 extern struct option cmdOptions[];
51
52 outputCode parseCmdline(int argc, char **argv, params_t *↩
      params);
53 outputCode optAction(char *arg, params_t *params);
54 outputCode optOutput(char *arg, params_t *params);
55 outputCode optInput(char *arg, params_t *params);
56 void optHelp(char *arg);
57 void optVersion(void);
58
59 #endif
```

C.0.8. parser.c

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: parser.c
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:21:22 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 10:50:18 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

CLA parser implementation.

------------------------------------------------------------ */
#include "parser.h"

/* The options are distinguished by the ASCII code of the
 * 'char' variables. */
struct option cmdOptions[] = {{"version", no_argument, NULL, 'V'},
                              {"help", no_argument, NULL, 'h'},
                              {"input", required_argument, NULL, 'i'},
                              {"output", required_argument, NULL, 'o'},
                              {"action", required_argument, NULL, 'a'},
                              {0, 0, 0, 0}};

void optVersion(void)
```

```
40 {
41   fprintf(stderr, "%s\n", VERSION);
42 
43   exit(EXIT_SUCCESS);
44 }
45 
46 void optHelp(char *arg)
47 {
48   if (arg == NULL)
49   {
50     fprintf(stderr, ERROR_ACTION_INVALID_ARGUMENT);
51     exit(EXIT_FAILURE);
52   }
53   fprintf(stderr, "Usage:\n");
54   fprintf(stderr, "  %s -h\n", arg);
55   fprintf(stderr, "  %s -V\n", arg);
56   fprintf(stderr, "  %s [options]\n", arg);
57   fprintf(stderr, "Options:\n");
58   fprintf(stderr, "-V, --version\tPrint version and quit.\n");
59   fprintf(stderr, "-i, --input\tLocation of the input file.\n"↩
       );
60   fprintf(stderr, "-o, --output\tLocation of the output file.\↩
       n");
61   fprintf(stderr,
62           "-a, --action\tProgram action: encode (default) or ↩
              decode.\n");
63   fprintf(stderr, "Examples:\n");
64   fprintf(stderr, "  %s -a encode -i ~/input -o ~/output\n", ↩
       arg);
65   fprintf(stderr, "  %s -a decode\n", arg);
66 
67   exit(EXIT_SUCCESS);
68 }
69 
70 outputCode validateStreamName(char *streamName)
71 {
72   if (streamName == NULL)
73   {
74     return outERROR;
75   }
76 
77   if (!strcmp(streamName, ".") || !strcmp(streamName, "..") ||
78       !strcmp(streamName, "/") || !strcmp(streamName, "//"))
79   {
80     return outERROR;
81   }
82 
```

```
83    return outOK;
84  }
85
86  outputCode optInput(char *arg, params_t *params)
87  {
88    if (validateStreamName(arg) == outERROR)
89    {
90      fprintf(stderr, ERROR_INVALID_INPUT_STREAM);
91      return outERROR;
92    }
93
94    if (strcmp(arg, STD_STREAM_TOKEN) == 0)
95    {
96      params->inputStream = stdin;
97    }
98    else
99    {
100     params->inputStream = fopen(arg, "rb");
101   }
102
103   if ((params->inputStream) == NULL)
104   {
105     fprintf(stderr, ERROR_OPENING_INPUT_STREAM);
106     return outERROR;
107   }
108
109   return outOK;
110 }
111
112 outputCode optOutput(char *arg, params_t *params)
113 {
114   if (validateStreamName(arg) == outERROR)
115   {
116     fprintf(stderr, ERROR_INVALID_OUTPUT_STREAM);
117     return outERROR;
118   }
119
120   if (strcmp(arg, STD_STREAM_TOKEN) == 0)
121   {
122     params->outputStream = stdout;
123   }
124   else
125   {
126     params->outputStream = fopen(arg, "wb");
127   }
128
129   if ((params->outputStream) == NULL)
```

```
130   {
131     fprintf(stderr, ERROR_OPENING_OUTPUT_STREAM);
132     return outERROR;
133   }
134
135   return outOK;
136 }
137
138 outputCode optAction(char *arg, params_t *params)
139 {
140   if (arg == NULL)
141   {
142     fprintf(stderr, ERROR_ACTION_INVALID_ARGUMENT);
143     return outERROR;
144   }
145
146   if (strcmp(arg, ENCODE_STR_TOKEN) == 0)
147   {
148     params->action = ENCODE_STR_TOKEN;
149   }
150   else if (strcmp(arg, DECODE_STR_TOKEN) == 0)
151   {
152     params->action = DECODE_STR_TOKEN;
153   }
154   else
155   {
156     fprintf(stderr, ERROR_ACTION_INVALID_ARGUMENT);
157     return outERROR;
158   }
159
160   return outOK;
161 }
162
163 outputCode parseCmdline(int argc, char **argv, params_t *↩
    params)
164 {
165   int indexptr = 0;
166   int optCode;
167
168   outputCode optOutCode = outERROR;
169   char *programName = argv[0];
170
171   /* Set the default values. */
172   params->action = ENCODE_STR_TOKEN;
173   params->inputStream = stdin;
174   params->outputStream = stdout;
175
```

```
176   /* 'version' and 'help' have no arguments. The rest, do
177    * have, and are mandatory.*/
178   char *shortOpts = "Vhi:o:a:";
179
180   while ((optCode =
181               getopt_long(argc, argv, shortOpts, cmdOptions, &↩
                    indexptr)) != -1)
182   {
183     switch (optCode)
184     {
185       case 'V':
186         optVersion();
187         break;
188       case 'h':
189         optHelp(programName);
190         break;
191       case 'i':
192         optOutCode = optInput(optarg, params);
193         break;
194       case 'o':
195         optOutCode = optOutput(optarg, params);
196         break;
197       case 'a':
198         optOutCode = optAction(optarg, params);
199         break;
200       case '?':
201         /* getopt_long already printed an error message. */
202         optOutCode = outERROR;
203         break;
204     }
205     if (optOutCode == outERROR)
206     {
207       return outERROR;
208     }
209   }
210
211   return outOK;
212 }
```

C.0.9. messages.h

```
/* ------------------------------------------------------------
 @Title:   FIUBA - 66.20 Organizacion de Computadoras.
 @Project: TP0 - Infraestructura basica.
------------------------------------------------------------
 @Filename: messages.h
------------------------------------------------------------
 @Authors:
    Husain, Ignacio Santiago.
          santiago.husain at gmail dot com
    Pesado, Lucia.
          luupesado at gmail dot com
    Verstraeten, Federico.
          federico.verstraeten at gmail dot com

 @Date:               12-Sep-2018 11:32:42 am
 @Last modified by:   Ignacio Santiago Husain
 @Last modified time: 24-Sep-2018 10:50:07 am

 @Copyright(C):
    This file is part of 'TP0 - Infraestructura basica.'.
    Unauthorized copying or use of this file via any medium
    is strictly prohibited.
------------------------------------------------------------

Program's common messages.

------------------------------------------------------------ */
#ifndef MESSAGES__H
#define MESSAGES__H

#ifndef ERROR_INVALID_INPUT_STREAM
#define ERROR_INVALID_INPUT_STREAM "ERROR: Invalid input ↩
   stream.\n"
#endif
#ifndef ERROR_OPENING_INPUT_STREAM
#define ERROR_OPENING_INPUT_STREAM "ERROR: Can't open input ↩
   stream.\n"
#endif
#ifndef ERROR_INVALID_OUTPUT_STREAM
#define ERROR_INVALID_OUTPUT_STREAM "ERROR: Invalid output ↩
   stream.\n"
#endif
#ifndef ERROR_OPENING_OUTPUT_STREAM
```

```
41 #define ERROR_OPENING_OUTPUT_STREAM "ERROR: Can't open output ↩
      stream.\n"
42 #endif
43 #ifndef ERROR_ACTION_INVALID_ARGUMENT
44 #define ERROR_ACTION_INVALID_ARGUMENT "ERROR: Invalid action ↩
      argument.\n"
45 #endif
46 #ifndef ERROR_OUTPUT_STREAM_WRITING_MSG
47 #define ERROR_OUTPUT_STREAM_WRITING_MSG "Output error when ↩
      writing stream.\n"
48 #endif
49 #ifndef ERROR_INPUT_STREAM_READING_MSG
50 #define ERROR_INPUT_STREAM_READING_MSG "Input error when ↩
      reading stream.\n"
51 #endif
52 #ifndef ERROR_B64_CHAR_NOT_FOUND_MSG
53 #define ERROR_B64_CHAR_NOT_FOUND_MSG \
54   "ERROR: Character is not in Base64 Table.\n"
55 #endif
56
57 #endif
```

C.0.10. main.s

```
    .file   1 "main.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2
    .type   translationTableB64, @object
    .size   translationTableB64, 64
translationTableB64:
    .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123"
    .ascii  "456789+/"
    .align  2
$LC0:
    .ascii  "encode\000"
    .text
    .align  2
    .globl  main
    .ent    main
main:
    .frame  $fp,64,$31      # vars= 24, regs= 3/0, args= 16, extra= 8
    .mask   0xd0000000,-8
    .fmask  0x00000000,0
    .set    noreorder
    .cpload $25
    .set    reorder
    subu    $sp,$sp,64
    .cprestore 16
    sw  $31,56($sp)
    sw  $fp,52($sp)
    sw  $28,48($sp)
    move    $fp,$sp
    sw  $4,64($fp)
    sw  $5,68($fp)
    lw  $4,64($fp)
    lw  $5,68($fp)
    addu    $6,$fp,24
    la  $25,parseCmdline
    jal $31,$25
    sw  $2,40($fp)
    lw  $3,40($fp)
    li  $2,1            # 0x1
```

```
42      bne $3,$2,$L18
43      li  $4,1            # 0x1
44      la  $25,exit
45      jal $31,$25
46  $L18:
47      lw  $4,24($fp)
48      la  $5,$LC0
49      la  $25,strcmp
50      jal $31,$25
51      bne $2,$0,$L19
52      addu    $4,$fp,24
53      la  $25,encode
54      jal $31,$25
55      sw  $2,44($fp)
56      b   $L20
57  $L19:
58      addu    $4,$fp,24
59      la  $25,decode
60      jal $31,$25
61      sw  $2,44($fp)
62  $L20:
63      lw  $3,44($fp)
64      li  $2,1            # 0x1
65      bne $3,$2,$L21
66      li  $4,1            # 0x1
67      la  $25,exit
68      jal $31,$25
69  $L21:
70      lw  $4,28($fp)
71      la  $25,fclose
72      jal $31,$25
73      lw  $4,32($fp)
74      la  $25,fclose
75      jal $31,$25
76      move    $2,$0
77      move    $sp,$fp
78      lw  $31,56($sp)
79      lw  $fp,52($sp)
80      addu    $sp,$sp,64
81      j   $31
82      .end    main
83      .size   main, .-main
84      .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

C.0.11. parser.s

```
    .file   1 "parser.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2
    .type   translationTableB64, @object
    .size   translationTableB64, 64
translationTableB64:
    .ascii  "↩
      ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123↩
      "
    .ascii  "456789+/"
    .align  2
$LC0:
    .ascii  "version\000"
    .align  2
$LC1:
    .ascii  "help\000"
    .align  2
$LC2:
    .ascii  "input\000"
    .align  2
$LC3:
    .ascii  "output\000"
    .align  2
$LC4:
    .ascii  "action\000"
    .globl  cmdOptions
    .data
    .align  2
    .type   cmdOptions, @object
    .size   cmdOptions, 96
cmdOptions:
    .word   $LC0
    .word   0
    .word   0
    .word   86
    .word   $LC1
    .word   0
    .word   0
    .word   104
    .word   $LC2
    .word   1
```

```
43      .word   0
44      .word   105
45      .word   $LC3
46      .word   1
47      .word   0
48      .word   111
49      .word   $LC4
50      .word   1
51      .word   0
52      .word   97
53      .word   0
54      .word   0
55      .word   0
56      .word   0
57      .rdata
58      .align  2
59  $LC5:
60      .ascii  "%s\n\000"
61      .align  2
62  $LC6:
63      .ascii  "1.0.0\000"
64      .text
65      .align  2
66      .globl  optVersion
67      .ent    optVersion
68  optVersion:
69      .frame  $fp,40,$31      # vars= 0, regs= 3/0, args= 16, ↩
           extra= 8
70      .mask   0xd0000000,-8
71      .fmask  0x00000000,0
72      .set    noreorder
73      .cpload $25
74      .set    reorder
75      subu    $sp,$sp,40
76      .cprestore 16
77      sw  $31,32($sp)
78      sw  $fp,28($sp)
79      sw  $28,24($sp)
80      move    $fp,$sp
81      la  $4,__sF+176
82      la  $5,$LC5
83      la  $6,$LC6
84      la  $25,fprintf
85      jal $31,$25
86      move    $4,$0
87      la  $25,exit
88      jal $31,$25
```

```
89      .end    optVersion
90      .size   optVersion, .-optVersion
91      .rdata
92      .align  2
93  $LC7:
94      .ascii  "ERROR: Invalid action argument.\n\000"
95      .align  2
96  $LC8:
97      .ascii  "Usage:\n\000"
98      .align  2
99  $LC9:
100     .ascii  "  %s -h\n\000"
101     .align  2
102 $LC10:
103     .ascii  "  %s -V\n\000"
104     .align  2
105 $LC11:
106     .ascii  "  %s [options]\n\000"
107     .align  2
108 $LC12:
109     .ascii  "Options:\n\000"
110     .align  2
111 $LC13:
112     .ascii  "-V, --version\tPrint version and quit.\n\000"
113     .align  2
114 $LC14:
115     .ascii  "-i, --input\tLocation of the input file.\n\000"
116     .align  2
117 $LC15:
118     .ascii  "-o, --output\tLocation of the output file.\n\000"
119     .align  2
120 $LC16:
121     .ascii  "-a, --action\tProgram action: encode (default) or↩
            decode"
122     .ascii  ".\n\000"
123     .align  2
124 $LC17:
125     .ascii  "Examples:\n\000"
126     .align  2
127 $LC18:
128     .ascii  "  %s -a encode -i ~/input -o ~/output\n\000"
129     .align  2
130 $LC19:
131     .ascii  "  %s -a decode\n\000"
132     .text
133     .align  2
134     .globl  optHelp
```

```
135     .ent    optHelp
136 optHelp:
137     .frame  $fp,40,$31      # vars= 0, regs= 3/0, args= 16, ↩
            extra= 8
138     .mask   0xd0000000,-8
139     .fmask  0x00000000,0
140     .set    noreorder
141     .cpload $25
142     .set    reorder
143     subu    $sp,$sp,40
144     .cprestore 16
145     sw  $31,32($sp)
146     sw  $fp,28($sp)
147     sw  $28,24($sp)
148     move    $fp,$sp
149     sw  $4,40($fp)
150     lw  $2,40($fp)
151     bne $2,$0,$L19
152     la  $4,__sF+176
153     la  $5,$LC7
154     la  $25,fprintf
155     jal $31,$25
156     li  $4,1            # 0x1
157     la  $25,exit
158     jal $31,$25
159 $L19:
160     la  $4,__sF+176
161     la  $5,$LC8
162     la  $25,fprintf
163     jal $31,$25
164     la  $4,__sF+176
165     la  $5,$LC9
166     lw  $6,40($fp)
167     la  $25,fprintf
168     jal $31,$25
169     la  $4,__sF+176
170     la  $5,$LC10
171     lw  $6,40($fp)
172     la  $25,fprintf
173     jal $31,$25
174     la  $4,__sF+176
175     la  $5,$LC11
176     lw  $6,40($fp)
177     la  $25,fprintf
178     jal $31,$25
179     la  $4,__sF+176
180     la  $5,$LC12
```

```
181     la  $25,fprintf
182     jal $31,$25
183     la  $4,__sF+176
184     la  $5,$LC13
185     la  $25,fprintf
186     jal $31,$25
187     la  $4,__sF+176
188     la  $5,$LC14
189     la  $25,fprintf
190     jal $31,$25
191     la  $4,__sF+176
192     la  $5,$LC15
193     la  $25,fprintf
194     jal $31,$25
195     la  $4,__sF+176
196     la  $5,$LC16
197     la  $25,fprintf
198     jal $31,$25
199     la  $4,__sF+176
200     la  $5,$LC17
201     la  $25,fprintf
202     jal $31,$25
203     la  $4,__sF+176
204     la  $5,$LC18
205     lw  $6,40($fp)
206     la  $25,fprintf
207     jal $31,$25
208     la  $4,__sF+176
209     la  $5,$LC19
210     lw  $6,40($fp)
211     la  $25,fprintf
212     jal $31,$25
213     move    $4,$0
214     la  $25,exit
215     jal $31,$25
216     .end    optHelp
217     .size   optHelp, .-optHelp
218     .rdata
219     .align  2
220 $LC20:
221     .ascii  ".\000"
222     .align  2
223 $LC21:
224     .ascii  "..\000"
225     .align  2
226 $LC22:
227     .ascii  "/\000"
```

```
228     .align  2
229 $LC23:
230     .ascii  "//\000"
231     .text
232     .align  2
233     .globl  validateStreamName
234     .ent    validateStreamName
235 validateStreamName:
236     .frame  $fp,48,$31      # vars= 8, regs= 3/0, args= 16, ↩
            extra= 8
237     .mask   0xd0000000,-8
238     .fmask  0x00000000,0
239     .set    noreorder
240     .cpload $25
241     .set    reorder
242     subu    $sp,$sp,48
243     .cprestore 16
244     sw  $31,40($sp)
245     sw  $fp,36($sp)
246     sw  $28,32($sp)
247     move    $fp,$sp
248     sw  $4,48($fp)
249     lw  $2,48($fp)
250     bne $2,$0,$L21
251     li  $2,1            # 0x1
252     sw  $2,24($fp)
253     b   $L20
254 $L21:
255     lw  $4,48($fp)
256     la  $5,$LC20
257     la  $25,strcmp
258     jal $31,$25
259     beq $2,$0,$L23
260     lw  $4,48($fp)
261     la  $5,$LC21
262     la  $25,strcmp
263     jal $31,$25
264     beq $2,$0,$L23
265     lw  $4,48($fp)
266     la  $5,$LC22
267     la  $25,strcmp
268     jal $31,$25
269     beq $2,$0,$L23
270     lw  $4,48($fp)
271     la  $5,$LC23
272     la  $25,strcmp
273     jal $31,$25
```

```
274     bne $2,$0,$L22
275 $L23:
276     li  $2,1            # 0x1
277     sw  $2,24($fp)
278     b   $L20
279 $L22:
280     sw  $0,24($fp)
281 $L20:
282     lw  $2,24($fp)
283     move    $sp,$fp
284     lw  $31,40($sp)
285     lw  $fp,36($sp)
286     addu    $sp,$sp,48
287     j   $31
288     .end    validateStreamName
289     .size   validateStreamName, .-validateStreamName
290     .rdata
291     .align  2
292 $LC24:
293     .ascii  "ERROR: Invalid input stream.\n\000"
294     .align  2
295 $LC25:
296     .ascii  "-\000"
297     .align  2
298 $LC26:
299     .ascii  "rb\000"
300     .align  2
301 $LC27:
302     .ascii  "ERROR: Can't open input stream.\n\000"
303     .text
304     .align  2
305     .globl  optInput
306     .ent    optInput
307 optInput:
308     .frame  $fp,48,$31      # vars= 8, regs= 4/0, args= 16, ↩
          extra= 8
309     .mask   0xd0010000,-4
310     .fmask  0x00000000,0
311     .set    noreorder
312     .cpload $25
313     .set    reorder
314     subu    $sp,$sp,48
315     .cprestore 16
316     sw  $31,44($sp)
317     sw  $fp,40($sp)
318     sw  $28,36($sp)
319     sw  $16,32($sp)
```

```
320     move    $fp,$sp
321     sw  $4,48($fp)
322     sw  $5,52($fp)
323     lw  $4,48($fp)
324     la  $25,validateStreamName
325     jal $31,$25
326     move    $3,$2
327     li  $2,1            # 0x1
328     bne $3,$2,$L25
329     la  $4,__sF+176
330     la  $5,$LC24
331     la  $25,fprintf
332     jal $31,$25
333     li  $2,1            # 0x1
334     sw  $2,24($fp)
335     b   $L24
336 $L25:
337     lw  $4,48($fp)
338     la  $5,$LC25
339     la  $25,strcmp
340     jal $31,$25
341     bne $2,$0,$L26
342     lw  $3,52($fp)
343     la  $2,__sF
344     sw  $2,4($3)
345     b   $L27
346 $L26:
347     lw  $16,52($fp)
348     lw  $4,48($fp)
349     la  $5,$LC26
350     la  $25,fopen
351     jal $31,$25
352     sw  $2,4($16)
353 $L27:
354     lw  $2,52($fp)
355     lw  $2,4($2)
356     bne $2,$0,$L28
357     la  $4,__sF+176
358     la  $5,$LC27
359     la  $25,fprintf
360     jal $31,$25
361     li  $2,1            # 0x1
362     sw  $2,24($fp)
363     b   $L24
364 $L28:
365     sw  $0,24($fp)
366 $L24:
```

```
367     lw  $2,24($fp)
368     move    $sp,$fp
369     lw  $31,44($sp)
370     lw  $fp,40($sp)
371     lw  $16,32($sp)
372     addu    $sp,$sp,48
373     j   $31
374     .end    optInput
375     .size   optInput, .-optInput
376     .rdata
377     .align  2
378 $LC28:
379     .ascii  "ERROR: Invalid output stream.\n\000"
380     .align  2
381 $LC29:
382     .ascii  "wb\000"
383     .align  2
384 $LC30:
385     .ascii  "ERROR: Can't open output stream.\n\000"
386     .text
387     .align  2
388     .globl  optOutput
389     .ent    optOutput
390 optOutput:
391     .frame  $fp,48,$31      # vars= 8, regs= 4/0, args= 16, ↩
          extra= 8
392     .mask   0xd0010000,-4
393     .fmask  0x00000000,0
394     .set    noreorder
395     .cpload $25
396     .set    reorder
397     subu    $sp,$sp,48
398     .cprestore 16
399     sw  $31,44($sp)
400     sw  $fp,40($sp)
401     sw  $28,36($sp)
402     sw  $16,32($sp)
403     move    $fp,$sp
404     sw  $4,48($fp)
405     sw  $5,52($fp)
406     lw  $4,48($fp)
407     la  $25,validateStreamName
408     jal $31,$25
409     move    $3,$2
410     li  $2,1            # 0x1
411     bne $3,$2,$L30
412     la  $4,__sF+176
```

```
413     la  $5,$LC28
414     la  $25,fprintf
415     jal $31,$25
416     li  $2,1            # 0x1
417     sw  $2,24($fp)
418     b   $L29
419 $L30:
420     lw  $4,48($fp)
421     la  $5,$LC25
422     la  $25,strcmp
423     jal $31,$25
424     bne $2,$0,$L31
425     lw  $3,52($fp)
426     la  $2,__sF+88
427     sw  $2,8($3)
428     b   $L32
429 $L31:
430     lw  $16,52($fp)
431     lw  $4,48($fp)
432     la  $5,$LC29
433     la  $25,fopen
434     jal $31,$25
435     sw  $2,8($16)
436 $L32:
437     lw  $2,52($fp)
438     lw  $2,8($2)
439     bne $2,$0,$L33
440     la  $4,__sF+176
441     la  $5,$LC30
442     la  $25,fprintf
443     jal $31,$25
444     li  $2,1            # 0x1
445     sw  $2,24($fp)
446     b   $L29
447 $L33:
448     sw  $0,24($fp)
449 $L29:
450     lw  $2,24($fp)
451     move    $sp,$fp
452     lw  $31,44($sp)
453     lw  $fp,40($sp)
454     lw  $16,32($sp)
455     addu    $sp,$sp,48
456     j   $31
457     .end    optOutput
458     .size   optOutput, .-optOutput
459     .rdata
```

```
460     .align  2
461 $LC31:
462     .ascii  "encode\000"
463     .align  2
464 $LC32:
465     .ascii  "decode\000"
466     .text
467     .align  2
468     .globl  optAction
469     .ent    optAction
470 optAction:
471     .frame  $fp,48,$31      # vars= 8, regs= 3/0, args= 16, ↩
           extra= 8
472     .mask   0xd0000000,-8
473     .fmask  0x00000000,0
474     .set    noreorder
475     .cpload $25
476     .set    reorder
477     subu    $sp,$sp,48
478     .cprestore 16
479     sw  $31,40($sp)
480     sw  $fp,36($sp)
481     sw  $28,32($sp)
482     move    $fp,$sp
483     sw  $4,48($fp)
484     sw  $5,52($fp)
485     lw  $2,48($fp)
486     bne $2,$0,$L35
487     la  $4,__sF+176
488     la  $5,$LC7
489     la  $25,fprintf
490     jal $31,$25
491     li  $2,1            # 0x1
492     sw  $2,24($fp)
493     b   $L34
494 $L35:
495     lw  $4,48($fp)
496     la  $5,$LC31
497     la  $25,strcmp
498     jal $31,$25
499     bne $2,$0,$L36
500     lw  $3,52($fp)
501     la  $2,$LC31
502     sw  $2,0($3)
503     b   $L37
504 $L36:
505     lw  $4,48($fp)
```

```
506     la  $5,$LC32
507     la  $25,strcmp
508     jal $31,$25
509     bne $2,$0,$L38
510     lw  $3,52($fp)
511     la  $2,$LC32
512     sw  $2,0($3)
513     b   $L37
514 $L38:
515     la  $4,__sF+176
516     la  $5,$LC7
517     la  $25,fprintf
518     jal $31,$25
519     li  $2,1            # 0x1
520     sw  $2,24($fp)
521     b   $L34
522 $L37:
523     sw  $0,24($fp)
524 $L34:
525     lw  $2,24($fp)
526     move    $sp,$fp
527     lw  $31,40($sp)
528     lw  $fp,36($sp)
529     addu    $sp,$sp,48
530     j   $31
531     .end    optAction
532     .size   optAction, .-optAction
533     .rdata
534     .align  2
535 $LC33:
536     .ascii  "Vhi:o:a:\000"
537     .text
538     .align  2
539     .globl  parseCmdline
540     .ent    parseCmdline
541 parseCmdline:
542     .frame  $fp,80,$31      # vars= 32, regs= 3/0, args= 24, ↩
          extra= 8
543     .mask   0xd0000000,-8
544     .fmask  0x00000000,0
545     .set    noreorder
546     .cpload $25
547     .set    reorder
548     subu    $sp,$sp,80
549     .cprestore 24
550     sw  $31,72($sp)
551     sw  $fp,68($sp)
```

```
552     sw  $28,64($sp)
553     move    $fp,$sp
554     sw  $4,80($fp)
555     sw  $5,84($fp)
556     sw  $6,88($fp)
557     sw  $0,32($fp)
558     li  $2,1            # 0x1
559     sw  $2,40($fp)
560     lw  $2,84($fp)
561     lw  $2,0($2)
562     sw  $2,44($fp)
563     lw  $3,88($fp)
564     la  $2,$LC31
565     sw  $2,0($3)
566     lw  $3,88($fp)
567     la  $2,__sF
568     sw  $2,4($3)
569     lw  $3,88($fp)
570     la  $2,__sF+88
571     sw  $2,8($3)
572     la  $2,$LC33
573     sw  $2,48($fp)
574 $L41:
575     addu    $2,$fp,32
576     sw  $2,16($sp)
577     lw  $4,80($fp)
578     lw  $5,84($fp)
579     lw  $6,48($fp)
580     la  $7,cmdOptions
581     la  $25,getopt_long
582     jal $31,$25
583     sw  $2,36($fp)
584     lw  $3,36($fp)
585     li  $2,-1           # 0xffffffffffffffff
586     bne $3,$2,$L43
587     b   $L42
588 $L43:
589     lw  $2,36($fp)
590     addu    $2,$2,-63
591     sw  $2,56($fp)
592     lw  $3,56($fp)
593     sltu    $2,$3,49
594     beq $2,$0,$L44
595     lw  $2,56($fp)
596     sll $3,$2,2
597     la  $2,$L51
598     addu    $2,$3,$2
```

```
599     lw  $2,0($2)
600     .cpadd  $2
601     j   $2
602     .rdata
603     .align  2
604 $L51:
605     .gpword $L50
606     .gpword $L44
607     .gpword $L44
608     .gpword $L44
609     .gpword $L44
610     .gpword $L44
611     .gpword $L44
612     .gpword $L44
613     .gpword $L44
614     .gpword $L44
615     .gpword $L44
616     .gpword $L44
617     .gpword $L44
618     .gpword $L44
619     .gpword $L44
620     .gpword $L44
621     .gpword $L44
622     .gpword $L44
623     .gpword $L44
624     .gpword $L44
625     .gpword $L44
626     .gpword $L44
627     .gpword $L44
628     .gpword $L45
629     .gpword $L44
630     .gpword $L44
631     .gpword $L44
632     .gpword $L44
633     .gpword $L44
634     .gpword $L44
635     .gpword $L44
636     .gpword $L44
637     .gpword $L44
638     .gpword $L44
639     .gpword $L49
640     .gpword $L44
641     .gpword $L44
642     .gpword $L44
643     .gpword $L44
644     .gpword $L44
645     .gpword $L44
```

```
646     .gpword $L46
647     .gpword $L47
648     .gpword $L44
649     .gpword $L44
650     .gpword $L44
651     .gpword $L44
652     .gpword $L44
653     .gpword $L48
654     .text
655 $L45:
656     la  $25,optVersion
657     jal $31,$25
658     b   $L44
659 $L46:
660     lw  $4,44($fp)
661     la  $25,optHelp
662     jal $31,$25
663     b   $L44
664 $L47:
665     lw  $4,optarg
666     lw  $5,88($fp)
667     la  $25,optInput
668     jal $31,$25
669     sw  $2,40($fp)
670     b   $L44
671 $L48:
672     lw  $4,optarg
673     lw  $5,88($fp)
674     la  $25,optOutput
675     jal $31,$25
676     sw  $2,40($fp)
677     b   $L44
678 $L49:
679     lw  $4,optarg
680     lw  $5,88($fp)
681     la  $25,optAction
682     jal $31,$25
683     sw  $2,40($fp)
684     b   $L44
685 $L50:
686     li  $2,1            # 0x1
687     sw  $2,40($fp)
688 $L44:
689     lw  $3,40($fp)
690     li  $2,1            # 0x1
691     bne $3,$2,$L41
692     li  $2,1            # 0x1
```

```
693     sw  $2,52($fp)
694     b   $L40
695 $L42:
696     sw  $0,52($fp)
697 $L40:
698     lw  $2,52($fp)
699     move    $sp,$fp
700     lw  $31,72($sp)
701     lw  $fp,68($sp)
702     addu    $sp,$sp,80
703     j   $31
704     .end    parseCmdline
705     .size   parseCmdline, .-parseCmdline
706     .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

C.0.12. encoder.s

```
    .file   1 "encoder.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2
    .type   translationTableB64, @object
    .size   translationTableB64, 64
translationTableB64:
    .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123"
    .ascii  "456789+/"
    .local  tailByte.0
    .comm   tailByte.0,1,1
    .data
    .type   bitMask.1, @object
    .size   bitMask.1, 1
bitMask.1:
    .byte   -4
    .align  2
    .type   shiftRightBit.2, @object
    .size   shiftRightBit.2, 4
shiftRightBit.2:
    .word   2
    .rdata
    .align  2
$LC0:
    .ascii  "=\000"
    .text
    .align  2
    .globl  base256ToBase64
    .ent    base256ToBase64
base256ToBase64:
    .frame  $fp,48,$31      # vars= 8, regs= 3/0, args= 16, extra= 8
    .mask   0xd0000000,-8
    .fmask  0x00000000,0
    .set    noreorder
    .cpload $25
    .set    reorder
    subu    $sp,$sp,48
    .cprestore 16
    sw  $31,40($sp)
```

```
42      sw  $fp,36($sp)
43      sw  $28,32($sp)
44      move    $fp,$sp
45      sw  $4,48($fp)
46      sw  $5,52($fp)
47      sb  $0,24($fp)
48      sb  $0,25($fp)
49      sb  $0,26($fp)
50      lbu $2,tailByte.0
51      sb  $2,25($fp)
52      lw  $3,52($fp)
53      li  $2,-1           # 0xffffffffffffffff
54      bne $3,$2,$L6
55      lw  $3,shiftRightBit.2
56      li  $2,6            # 0x6
57      bne $3,$2,$L7
58      lbu $2,25($fp)
59      sb  $2,24($fp)
60      lbu $3,24($fp)
61      la  $2,translationTableB64
62      addu    $2,$3,$2
63      lw  $4,48($fp)
64      move    $5,$2
65      li  $6,1            # 0x1
66      la  $25,strncpy
67      jal $31,$25
68      lw  $4,48($fp)
69      la  $5,$LC0
70      li  $6,1            # 0x1
71      la  $25,strncat
72      jal $31,$25
73      lbu $2,26($fp)
74      addu    $2,$2,2
75      andi    $2,$2,0x00ff
76      sw  $2,28($fp)
77      b   $L5
78  $L7:
79      lw  $3,shiftRightBit.2
80      li  $2,4            # 0x4
81      bne $3,$2,$L9
82      lbu $2,25($fp)
83      sb  $2,24($fp)
84      lbu $3,24($fp)
85      la  $2,translationTableB64
86      addu    $2,$3,$2
87      lw  $4,48($fp)
88      move    $5,$2
```

```
89      li  $6,1            # 0x1
90      la  $25,strncpy
91      jal $31,$25
92      lw  $4,48($fp)
93      la  $5,$LC0
94      li  $6,1            # 0x1
95      la  $25,strncat
96      jal $31,$25
97      lw  $4,48($fp)
98      la  $5,$LC0
99      li  $6,1            # 0x1
100     la  $25,strncat
101     jal $31,$25
102     lbu $2,26($fp)
103     addu    $2,$2,3
104     andi    $2,$2,0x00ff
105     sw  $2,28($fp)
106     b   $L5
107 $L9:
108     lbu $2,26($fp)
109     sw  $2,28($fp)
110     b   $L5
111 $L6:
112     lbu $3,52($fp)
113     lbu $2,bitMask.1
114     and $2,$3,$2
115     sb  $2,24($fp)
116     lbu $3,24($fp)
117     lw  $2,shiftRightBit.2
118     sra $2,$3,$2
119     sb  $2,24($fp)
120     lbu $2,bitMask.1
121     nor $2,$0,$2
122     lbu $3,52($fp)
123     and $2,$3,$2
124     sb  $2,tailByte.0
125     lbu $4,tailByte.0
126     li  $3,6            # 0x6
127     lw  $2,shiftRightBit.2
128     subu    $2,$3,$2
129     sll $2,$4,$2
130     sb  $2,tailByte.0
131     lbu $3,24($fp)
132     lbu $2,25($fp)
133     or  $2,$3,$2
134     sb  $2,24($fp)
135     lbu $3,24($fp)
```

```
136     la  $2,translationTableB64
137     addu    $2,$3,$2
138     lw  $4,48($fp)
139     move    $5,$2
140     li  $6,1            # 0x1
141     la  $25,strncpy
142     jal $31,$25
143     lbu $2,26($fp)
144     addu    $2,$2,1
145     sb  $2,26($fp)
146     lw  $2,shiftRightBit.2
147     addu    $2,$2,2
148     sw  $2,shiftRightBit.2
149     lbu $2,bitMask.1
150     sll $2,$2,2
151     sb  $2,bitMask.1
152     andi    $2,$2,0x00ff
153     bne $2,$0,$L11
154     li  $2,-4           # 0xfffffffffffffffc
155     sb  $2,bitMask.1
156     li  $2,2            # 0x2
157     sw  $2,shiftRightBit.2
158     lbu $3,tailByte.0
159     la  $2,translationTableB64
160     addu    $2,$3,$2
161     lw  $4,48($fp)
162     move    $5,$2
163     li  $6,1            # 0x1
164     la  $25,strncat
165     jal $31,$25
166     lbu $2,26($fp)
167     addu    $2,$2,1
168     sb  $2,26($fp)
169     sb  $0,tailByte.0
170 $L11:
171     lbu $2,26($fp)
172     sw  $2,28($fp)
173 $L5:
174     lw  $2,28($fp)
175     move    $sp,$fp
176     lw  $31,40($sp)
177     lw  $fp,36($sp)
178     addu    $sp,$sp,48
179     j   $31
180     .end    base256ToBase64
181     .size   base256ToBase64, .-base256ToBase64
182     .rdata
```

```
183     .align  2
184 $LC1:
185     .ascii  "Input error when reading stream.\n\000"
186     .align  2
187 $LC2:
188     .ascii  "\n\000"
189     .align  2
190 $LC3:
191     .ascii  "Output error when writing stream.\n\000"
192     .text
193     .align  2
194     .globl  encode
195     .ent    encode
196 encode:
197     .frame  $fp,72,$31      # vars= 32, regs= 3/0, args= 16, ↩
          extra= 8
198     .mask   0xd0000000,-8
199     .fmask  0x00000000,0
200     .set    noreorder
201     .cpload $25
202     .set    reorder
203     subu    $sp,$sp,72
204     .cprestore 16
205     sw  $31,64($sp)
206     sw  $fp,60($sp)
207     sw  $28,56($sp)
208     move    $fp,$sp
209     sw  $4,72($fp)
210     sw  $0,32($fp)
211     sb  $0,40($fp)
212     sb  $0,41($fp)
213 $L13:
214     addu    $2,$fp,32
215     move    $4,$2
216     move    $5,$0
217     li  $6,4            # 0x4
218     la  $25,memset
219     jal $31,$25
220     lw  $2,72($fp)
221     lw  $3,4($2)
222     lw  $2,72($fp)
223     lw  $2,4($2)
224     lw  $2,4($2)
225     addu    $2,$2,-1
226     sw  $2,4($3)
227     bgez    $2,$L16
228     lw  $2,72($fp)
```

```
229     lw  $4,4($2)
230     la  $25,__srget
231     jal $31,$25
232     sw  $2,48($fp)
233     b   $L17
234 $L16:
235     lw  $2,72($fp)
236     lw  $4,4($2)
237     lw  $2,0($4)
238     move    $3,$2
239     lbu $3,0($3)
240     sw  $3,48($fp)
241     addu    $2,$2,1
242     sw  $2,0($4)
243 $L17:
244     lw  $2,48($fp)
245     sw  $2,24($fp)
246     lw  $2,72($fp)
247     lw  $2,4($2)
248     lhu $2,12($2)
249     srl $2,$2,6
250     andi    $2,$2,0x1
251     beq $2,$0,$L18
252     la  $4,__sF+176
253     la  $5,$LC1
254     la  $25,fprintf
255     jal $31,$25
256     li  $2,1            # 0x1
257     sw  $2,44($fp)
258     b   $L12
259 $L18:
260     addu    $2,$fp,32
261     move    $4,$2
262     lw  $5,24($fp)
263     la  $25,base256ToBase64
264     jal $31,$25
265     sb  $2,41($fp)
266     lbu $3,40($fp)
267     lbu $2,41($fp)
268     addu    $2,$3,$2
269     slt $2,$2,77
270     beq $2,$0,$L19
271     lbu $2,40($fp)
272     lbu $3,41($fp)
273     addu    $2,$2,$3
274     sb  $2,40($fp)
275     b   $L20
```

```
276 $L19:
277     lw  $2,72($fp)
278     la  $4,$LC2
279     lw  $5,8($2)
280     la  $25,fputs
281     jal $31,$25
282     lw  $2,72($fp)
283     lw  $2,8($2)
284     lhu $2,12($2)
285     srl $2,$2,6
286     andi    $2,$2,0x1
287     beq $2,$0,$L21
288     la  $4,__sF+176
289     la  $5,$LC3
290     la  $25,fprintf
291     jal $31,$25
292     li  $2,1            # 0x1
293     sw  $2,44($fp)
294     b   $L12
295 $L21:
296     lbu $2,41($fp)
297     sb  $2,40($fp)
298 $L20:
299     addu    $2,$fp,32
300     lw  $3,72($fp)
301     move    $4,$2
302     lw  $5,8($3)
303     la  $25,fputs
304     jal $31,$25
305     lw  $2,72($fp)
306     lw  $2,8($2)
307     lhu $2,12($2)
308     srl $2,$2,6
309     andi    $2,$2,0x1
310     beq $2,$0,$L15
311     la  $4,__sF+176
312     la  $5,$LC3
313     la  $25,fprintf
314     jal $31,$25
315     li  $2,1            # 0x1
316     sw  $2,44($fp)
317     b   $L12
318 $L15:
319     lw  $3,24($fp)
320     li  $2,-1           # 0xffffffffffffffff
321     bne $3,$2,$L13
322     sw  $0,44($fp)
```

```
323 $L12:
324     lw  $2,44($fp)
325     move    $sp,$fp
326     lw  $31,64($sp)
327     lw  $fp,60($sp)
328     addu    $sp,$sp,72
329     j   $31
330     .end    encode
331     .size   encode, .-encode
332     .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

C.0.13. decoder.s

```
    .file   1 "decoder.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2
    .type   translationTableB64, @object
    .size   translationTableB64, 64
translationTableB64:
    .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
    .align  2
$LC0:
    .ascii  "ERROR: Character is not in Base64 Table.\n\000"
    .text
    .align  2
    .globl  base64ToBase256
    .ent    base64ToBase256
base64ToBase256:
    .frame  $fp,72,$31      # vars= 32, regs= 3/0, args= 16, extra= 8
    .mask   0xd0000000,-8
    .fmask  0x00000000,0
    .set    noreorder
    .cpload $25
    .set    reorder
    subu    $sp,$sp,72
    .cprestore 16
    sw  $31,64($sp)
    sw  $fp,60($sp)
    sw  $28,56($sp)
    move    $fp,$sp
    sw  $4,72($fp)
    sw  $5,76($fp)
    sw  $6,80($fp)
    li  $2,-16777216            # 0xffffffffff000000
    sw  $2,24($fp)
    sb  $0,28($fp)
    sb  $0,29($fp)
    sw  $0,32($fp)
    sw  $0,40($fp)
    sw  $0,44($fp)
```

```
42      sb  $0,48($fp)
43      sb  $0,28($fp)
44  $L6:
45      lbu $2,28($fp)
46      sltu    $2,$2,4
47      bne $2,$0,$L9
48      b   $L7
49  $L9:
50      lbu $3,28($fp)
51      lw  $2,76($fp)
52      addu    $2,$3,$2
53      lbu $3,0($2)
54      li  $2,61           # 0x3d
55      bne $3,$2,$L10
56      lbu $2,28($fp)
57      addu    $3,$fp,32
58      addu    $2,$3,$2
59      sb  $0,0($2)
60      b   $L8
61  $L10:
62      sb  $0,29($fp)
63  $L11:
64      lbu $2,29($fp)
65      sltu    $2,$2,64
66      bne $2,$0,$L14
67      b   $L12
68  $L14:
69      lbu $3,28($fp)
70      lw  $2,76($fp)
71      addu    $2,$3,$2
72      lbu $3,0($2)
73      lbu $2,29($fp)
74      lb  $2,translationTableB64($2)
75      bne $3,$2,$L13
76      lbu $3,28($fp)
77      addu    $2,$fp,32
78      addu    $3,$2,$3
79      lbu $2,29($fp)
80      sb  $2,0($3)
81      lw  $3,80($fp)
82      lw  $2,80($fp)
83      lbu $2,0($2)
84      addu    $2,$2,1
85      sb  $2,0($3)
86      b   $L12
87  $L13:
88      lbu $2,29($fp)
```

```
89      addu    $2,$2,1
90      sb  $2,29($fp)
91      b   $L11
92  $L12:
93      lbu $2,29($fp)
94      sltu    $2,$2,64
95      bne $2,$0,$L8
96      la  $4,__sF+176
97      la  $5,$LC0
98      la  $25,fprintf
99      jal $31,$25
100     li  $2,1            # 0x1
101     sw  $2,52($fp)
102     b   $L5
103 $L8:
104     lbu $2,28($fp)
105     addu    $2,$2,1
106     sb  $2,28($fp)
107     b   $L6
108 $L7:
109     sb  $0,28($fp)
110 $L17:
111     lbu $2,28($fp)
112     sltu    $2,$2,4
113     bne $2,$0,$L20
114     b   $L18
115 $L20:
116     lbu $2,48($fp)
117     addu    $2,$2,2
118     sb  $2,48($fp)
119     lbu $3,28($fp)
120     addu    $2,$fp,32
121     addu    $2,$2,$3
122     lbu $2,0($2)
123     sw  $2,40($fp)
124     lbu $3,28($fp)
125     li  $2,3            # 0x3
126     subu    $2,$2,$3
127     sll $3,$2,3
128     lw  $2,40($fp)
129     sll $2,$2,$3
130     sw  $2,40($fp)
131     lbu $3,48($fp)
132     lw  $2,40($fp)
133     sll $2,$2,$3
134     sw  $2,40($fp)
135     lw  $3,44($fp)
```

```
136     lw  $2,40($fp)
137     or  $2,$3,$2
138     sw  $2,44($fp)
139     lbu $2,28($fp)
140     addu    $2,$2,1
141     sb  $2,28($fp)
142     b   $L17
143 $L18:
144     sb  $0,28($fp)
145 $L21:
146     lw  $3,44($fp)
147     lw  $2,24($fp)
148     and $2,$3,$2
149     sw  $2,40($fp)
150     lbu $3,28($fp)
151     li  $2,3            # 0x3
152     subu    $2,$2,$3
153     sll $3,$2,3
154     lw  $2,40($fp)
155     srl $2,$2,$3
156     sw  $2,40($fp)
157     lbu $3,28($fp)
158     lw  $2,72($fp)
159     addu    $3,$3,$2
160     lbu $2,40($fp)
161     sb  $2,0($3)
162     lw  $2,24($fp)
163     srl $2,$2,8
164     sw  $2,24($fp)
165     lbu $2,28($fp)
166     addu    $2,$2,1
167     sb  $2,28($fp)
168     lbu $2,28($fp)
169     sltu    $2,$2,3
170     bne $2,$0,$L21
171     sw  $0,52($fp)
172 $L5:
173     lw  $2,52($fp)
174     move    $sp,$fp
175     lw  $31,64($sp)
176     lw  $fp,60($sp)
177     addu    $sp,$sp,72
178     j   $31
179     .end    base64ToBase256
180     .size   base64ToBase256, .-base64ToBase256
181     .rdata
182     .align  2
```

```
183 $LC1:
184     .ascii  "Input error when reading stream.\n\000"
185     .align  2
186 $LC2:
187     .ascii  "Output error when writing stream.\n\000"
188     .text
189     .align  2
190     .globl  decode
191     .ent    decode
192 decode:
193     .frame  $fp,80,$31      # vars= 40, regs= 3/0, args= 16, ↩
          extra= 8
194     .mask   0xd0000000,-8
195     .fmask  0x00000000,0
196     .set    noreorder
197     .cpload $25
198     .set    reorder
199     subu    $sp,$sp,80
200     .cprestore 16
201     sw  $31,72($sp)
202     sw  $fp,68($sp)
203     sw  $28,64($sp)
204     move    $fp,$sp
205     sw  $4,80($fp)
206     sw  $0,32($fp)
207     sh  $0,40($fp)
208     sb  $0,42($fp)
209     sb  $0,49($fp)
210     sb  $0,50($fp)
211 $L26:
212     sb  $0,50($fp)
213     sb  $0,48($fp)
214 $L29:
215     lbu $2,48($fp)
216     sltu    $2,$2,4
217     bne $2,$0,$L32
218     b   $L30
219 $L32:
220     lw  $2,80($fp)
221     lw  $3,4($2)
222     lw  $2,80($fp)
223     lw  $2,4($2)
224     lw  $2,4($2)
225     addu    $2,$2,-1
226     sw  $2,4($3)
227     bgez    $2,$L33
228     lw  $2,80($fp)
```

```
229     lw  $4,4($2)
230     la  $25,__srget
231     jal $31,$25
232     sb  $2,56($fp)
233     b   $L34
234 $L33:
235     lw  $2,80($fp)
236     lw  $4,4($2)
237     lw  $2,0($4)
238     move    $3,$2
239     lbu $3,0($3)
240     sb  $3,56($fp)
241     addu    $2,$2,1
242     sw  $2,0($4)
243 $L34:
244     lbu $2,56($fp)
245     sb  $2,24($fp)
246     lw  $2,80($fp)
247     lw  $2,4($2)
248     lhu $2,12($2)
249     srl $2,$2,6
250     andi    $2,$2,0x1
251     beq $2,$0,$L35
252     la  $4,__sF+176
253     la  $5,$LC1
254     la  $25,fprintf
255     jal $31,$25
256     li  $2,1            # 0x1
257     sw  $2,52($fp)
258     b   $L25
259 $L35:
260     lbu $3,24($fp)
261     li  $2,10           # 0xa
262     beq $3,$2,$L37
263     lbu $3,24($fp)
264     li  $2,9            # 0x9
265     beq $3,$2,$L37
266     lbu $3,24($fp)
267     li  $2,32           # 0x20
268     beq $3,$2,$L37
269     b   $L36
270 $L37:
271     lbu $2,48($fp)
272     addu    $2,$2,-1
273     sb  $2,48($fp)
274     b   $L31
275 $L36:
```

```
276     lbu $3,48($fp)
277     addu    $2,$fp,32
278     addu    $3,$2,$3
279     lbu $2,24($fp)
280     sb  $2,0($3)
281     lw  $2,80($fp)
282     lw  $2,4($2)
283     lhu $2,12($2)
284     srl $2,$2,5
285     andi    $2,$2,0x1
286     beq $2,$0,$L31
287     lbu $2,48($fp)
288     beq $2,$0,$L40
289     addu    $2,$fp,40
290     addu    $3,$fp,32
291     addu    $6,$fp,50
292     move    $4,$2
293     move    $5,$3
294     la  $25,base64ToBase256
295     jal $31,$25
296     move    $3,$2
297     li  $2,1            # 0x1
298     bne $3,$2,$L41
299     li  $2,1            # 0x1
300     sw  $2,52($fp)
301     b   $L25
302 $L41:
303     sb  $0,49($fp)
304 $L42:
305     lbu $3,49($fp)
306     lbu $2,50($fp)
307     addu    $2,$2,-1
308     slt $2,$3,$2
309     bne $2,$0,$L45
310     b   $L43
311 $L45:
312     lbu $3,49($fp)
313     addu    $2,$fp,40
314     addu    $2,$2,$3
315     lbu $2,0($2)
316     lw  $3,80($fp)
317     move    $4,$2
318     lw  $5,8($3)
319     la  $25,fputc
320     jal $31,$25
321     lbu $2,49($fp)
322     addu    $2,$2,1
```

```
323     sb  $2,49($fp)
324     b   $L42
325 $L43:
326     lw  $2,80($fp)
327     lw  $2,8($2)
328     lhu $2,12($2)
329     srl $2,$2,6
330     andi    $2,$2,0x1
331     beq $2,$0,$L40
332     la  $4,__sF+176
333     la  $5,$LC2
334     la  $25,fprintf
335     jal $31,$25
336     li  $2,1            # 0x1
337     sw  $2,52($fp)
338     b   $L25
339 $L40:
340     sw  $0,52($fp)
341     b   $L25
342 $L31:
343     lbu $2,48($fp)
344     addu    $2,$2,1
345     sb  $2,48($fp)
346     b   $L29
347 $L30:
348     addu    $2,$fp,40
349     addu    $3,$fp,32
350     addu    $6,$fp,50
351     move    $4,$2
352     move    $5,$3
353     la  $25,base64ToBase256
354     jal $31,$25
355     move    $3,$2
356     li  $2,1            # 0x1
357     bne $3,$2,$L47
358     li  $2,1            # 0x1
359     sw  $2,52($fp)
360     b   $L25
361 $L47:
362     sb  $0,49($fp)
363 $L48:
364     lbu $3,49($fp)
365     lbu $2,50($fp)
366     addu    $2,$2,-1
367     slt $2,$3,$2
368     bne $2,$0,$L51
369     b   $L49
```

```
370 $L51:
371     lbu $3,49($fp)
372     addu    $2,$fp,40
373     addu    $2,$2,$3
374     lbu $2,0($2)
375     lw  $3,80($fp)
376     move    $4,$2
377     lw  $5,8($3)
378     la  $25,fputc
379     jal $31,$25
380     lbu $2,49($fp)
381     addu    $2,$2,1
382     sb  $2,49($fp)
383     b   $L48
384 $L49:
385     lw  $2,80($fp)
386     lw  $2,8($2)
387     lhu $2,12($2)
388     srl $2,$2,6
389     andi    $2,$2,0x1
390     beq $2,$0,$L26
391     la  $4,__sF+176
392     la  $5,$LC2
393     la  $25,fprintf
394     jal $31,$25
395     li  $2,1            # 0x1
396     sw  $2,52($fp)
397 $L25:
398     lw  $2,52($fp)
399     move    $sp,$fp
400     lw  $31,72($sp)
401     lw  $fp,68($sp)
402     addu    $sp,$sp,80
403     j   $31
404     .end    decode
405     .size   decode, .-decode
406     .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```