.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo Practico 0

[66.20/86.37] Organización de Computadoras
Curso 2
Segundo cuatrimestre de 2020

Alumnos	Padrón	Correo electrónico	Slack
Gómez, Joaquín	103735	joagomez@fi.uba.ar	Joaquin Gomez
Grassano, Bruno	103855	bgrassano@fi.uba.ar	Bruno Grassano
Romero, Adrián	103371	adromero@fi.uba.ar	Adrian Romero

Índice

1. Introducción

El propósito del trabajo es familiarizarnos con las herramientas que utilizaremos a lo largo del curso.

Para ello buscamos crear un programa en C que permita la codificación de un archivo a base 64. El programa también debe permitir la decodificación desde base 64.

2. Diseño e implementación

Con el objetivo de crear un codificador de binario codificado en ASCII a base 64 lo primero que tenemos que hacer es tomar los primeros 3 caracteres del archivo de entrada e interpretar la secuencia de bytes (recibida en un archivo) e ir agrupándola de a 6 bits. Una vez que tenemos estos 6 bits utilizamos una tabla de conversión para asignarle el caracter en base 64 correspondiente con esos 6 bits. Finalmente escribimos estos caracteres en base 64 en el archivo de salida. Procedemos a mostrar un ejemplo para clarificar esta idea.

Supongamos que queremos codificar: 'En ' a base 64. Lo primero que debemos hacer es encontrar el byte equivalente al código ASCII para cada uno de los caracteres 'E' 'n' y ' ' y poner los bytes uno seguido de otro. Luego 'leemos' esta secuencia de bits agrupándolos en grupos de 6 bits y realizamos la codificación con una tabla de conversión.

La siguiente tabla muestra gráficamente este proceso:

Input	E								n															
ASCII	69								110								32							
Bits	0	1	0	0	0	1	0	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0	0	0
Índice	17						22						56				32							
Base64	R						W						4				g							

El codificado en base 64 resulta entonces 'RW4g'

Observamos que podemos codificar tres caracteres en 4 dígitos de base 64. Por lo tanto lo que haremos sera agarrar el archivo original, el cual puede venir por stdin, e ir codificando a base 64 de a tres bytes.

Explicamos ahora como logramos la codificación paso por paso. Para ello utilizaremos el mismo ejemplo anterior, es decir el archivo que queremos codificar tiene escrito 'En '

Lo primero que hacemos es poner los 3 bytes (cada caracter es 1 byte) en una variable de tipo int de 4 bytes:

```
caracterCodificandose = textoACodificar[0];
```

caracterCodificandose es una variable de tipo int (4 bytes) y textoACodificar[0] es un caracter (1 byte). Entonces cargamos el primer caracter del texto (E = 69 = 01000101) a la variable caracterCodificandose. Resulta:

$$\text{caracterCodifcandose} = \quad 0x00 \mid 0x00 \mid 0x00 \mid 01000101 \mid$$

Ahora buscamos cargar el segundo caracter a la variable int.

```
caracterCodificandose = (caracterCodificandose << 8 | textoACodificar[1])
```

Lo que hacemos ahora es un desplazamiento de 8 bits del caracterCodificandose y un OR con textoACodificar[1] que es el segundo caracter del texto (n = 110 = 01101110). Es decir:

$$\begin{gathered} 0x00 \mid 0x00 \mid 01000101 \mid 0x00 \mid \\ \text{OR} \\ 0x00 \mid 0x00 \mid 0x00 \mid 01101110 \mid \\ = \\ \text{caracterCodifcandose} = \quad 0x00 \mid 0x00 \mid 01000101 \mid 01101110 \mid \end{gathered}$$

Si repetimos esto una vez mas tendremos los tres bytes que queremos en el caracterCodificandose:

```
caracterCodificandose = (caracterCodificandose << 8 | textoACodificar[2])
```

caracterCodifcandose= 0x00 | 01000101 | 01101110 | 00100000 |

Una vez hecho esto debemos proceder a 'leer' secuencias de 6 bits y asignarle su equivalente en base 64:

```
salidaCodificada[0] = caracteresBase64[(caracterCodificandose >> 18) &
    BINARIO_63];
```

Observamos entonces que salidaCodificada[0] sera el primer valor codificado en base 64. Veamos cual es el resultado de la operación:

caracterCodificandose >> 18 & BINARIO 63

La operación consiste en desplazar 18 bits y luego realizar un AND (bitwise) con 63 BINARIO

0x00 | 0x00 | 0x00 | 000100010 |
AND
0x00 | 0x00 | 0x00 | 00111111 |
=
0x00 | 0x00 | 0x00 | 000100010 |

Observamos que el resultado de la operación son los 6 bits que nos interesaban. Estos bits nos indican que el primer caracter codificado sera el caracter 100010 = 17 de base 64 que es el caracter 'R'

Procedemos ahora a codificar un segundo caracter a base 64:

```
salidaCodificada[1] = caracteresBase64[(caracterCodificandose >> 12) &
    BINARIO_63];
```

Realizamos la siguiente operación:

caracterCodificandose >> 12 & BINARIO 63

Es decir:

0x00 | 00000100 | 01010110 |
AND
0x00 | 0x00 | 0x00 | 00111111 |
=
0x00 | 0x00 | 0x00 | 00010110 |

Esto indica que el segundo caracter codificado sera el caracter 010110 = 22 de base 64 que es el caracter 'W'

Haciendo esto para los siguientes grupos de 6 bits se obtiene que la codificación de 'En ' en base 64 es 'RW4g'

Observamos que cada 3 caracteres del archivo original obtenemos 4 caracteres de base 64. Por lo tanto si el archivo original no tiene una cantidad de caracteres que sea múltiplo de 3 agregaremos símbolos '=' al final del archivo de salida. (ya vienen cargados, y se van remplazando a medida

que se hace la codificación) Esto indica que los bits 0 que se hayan añadido de relleno durante la codificación (al cargar los caracteres leídos en la variable int) no forman parte de los datos codificados.

Explicamos ahora como llevamos a cabo el proceso de decodificación.

Lo primero que hacemos es nuevamente interpretar los caracteres recibidos como su numero ASCII. A cada uno de estos números ASCII obtenidos debemos, mediante una tabla de conversión, asignarle el numero que nos permita recuperar la secuencia de bits inicial mediante ORs entre estos números.

Gráficamente:

Base64	R								W								4								g							
ASCII	82								87								52								103							
Convertido	17								22								56								32							
Bits	0	0	0	1	0	0	0	1	0	0	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0

Donde observamos que los números convertidos son números de 0 a 63 y por lo tanto en su representación binaria los dos dígitos mas significativos de cada uno son 00. Removiendo estos 00 recuperaremos la secuencia binaria inicial de 24 bits que se traducen en los 3 caracteres que generaron la codificación. Veremos que podemos remover los 00 con simples operaciones OR.

Para explicar esto mejor, mostraremos como realizamos la decodificación con un ejemplo. Decodificaremos 'RW4g' que es lo que obtuvimos en la codificación en base 64 de 'En '.

En primer lugar necesitamos una tabla de conversión tal como en el caso anterior. Esta tabla es:

```
int valorAscii[] = { 62, -1, -1, -1, 63, 52, 53, 54, 55, 56, 57, 58, 59,
    60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
     11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1, -1,
    -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
    39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 };
```

Se observa que la tabla tiene muchos -1. Esto se debe a que los caracteres de base 64 no se encuentran contiguos en el código ASCII sino que se encuentran distribuidos entre los caracteres 43 y 122. Esto hace que requiramos una tabla de conversión de 122 - 43 = 79 caracteres en la que solo algunas (64) de las posiciones se corresponden con caracteres de base 64.

Procedemos ahora a cargar en una variable de tipo int llamada caracterDecodificandose un valor de la tabla anterior:

```
caracterDecodificandose = valorAscii[textoADecodificar[0]-43];
```

Para la primera iteración tenemos que textoADecodificar[0] es 'R' que es el numero 82 ASCII por lo tanto al restarle 43 obtenemos el numero 39. La tabla de conversión nos dice que valorAscii[39] es el numero 17 que en binario es: 00010001. Lo cual quiere decir que:

$$\text{caracterDecodificandose} = \quad 0x00 \mid 0x00 \mid 0x00 \mid 00010001 \mid$$

Luego procedemos a realizar un shift de 6 bits de caracterDecodificandose y luego un OR (bitwise) con el valor de la tabla de conversión que se corresponde con el siguiente caracter del texto a decodificar, es decir 'W'

```
caracterDecodificandose = (caracterDecodificandose << 6) | valorAscii[
    textoADecodificar[1]-43];
```

Como ya mencionamos, textoADecodificar[2] = 'W' que es el numero 87 ASCII y por lo tanto la resta resulta: 87 - 43 = 44. La tabla nos indica que valorASCII[44] = 22 que en binario es: 00010110. Por lo tanto:

$$\begin{array}{c} \text{0x00} \mid \text{0x00} \mid \text{00000100} \mid \text{01000000} \mid \\ \text{OR} \\ \text{0x00} \mid \text{0x00} \mid \text{0x00} \mid \text{00010110} \mid \\ = \\ \text{caracterDecodificandose} = \quad \text{0x00} \mid \text{0x00} \mid \text{00000100} \mid \text{01010110} \mid \end{array}$$

Observemos que esto 'solapa' los 00 mas significativos del numero 00010110 con dígitos de caracterLeido. Esto quiere decir que pudimos remover los 00 que deseábamos con operaciones OR.

Realizando esto 2 veces mas caracterDecodificandose resulta:

$$\text{caracterDecodificandose} = \quad \text{0x00} \mid \text{01000101} \mid \text{01101110} \mid \text{00100000} \mid$$

Donde podemos observar que se tiene la secuencia de bits deseada en los 3 bytes menos significativos. Finalmente procedemos a poner estos bytes en la salidaCodificada para recuperar la secuencia original: 'En '

```
salidaDecodificada[0] = (caracterDecodificandose >> 16) & BINARIO_255;
    if (textoADecodificar[2] != '='){
           salidaDecodificada[1] = (caracterDecodificandose >> 8) &
               BINARIO_255;
    }
    if (textoADecodificar[3] != '='){
           salidaDecodificada[2] = caracterDecodificandose & BINARIO_255;
    }
```

Mencionamos las herramientas con las que se realizo el trabajo practico y sus respectivas versiones:

- Lenguaje C para escribir el programa codificador y decodificador
- Debugger GNU gdb. Versión: (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409
- Emulador QEMU para simular la arquitectura MIPS corriendo Debian y poder obtener el archivo que genera el compilador. Versión: 2.11.1 (Debian 1:2.11+dfsg-1ubuntu7.32)
- LaTeX para la redacción del informe.

3. Proceso de compilación

Para poder realizar la compilación del trabajo se recomienda utilizar el *makefile* mediante la siguiente linea en la terminal.

```
make tp
```

Este *makefile* también tiene la opcion de *makeclean*. La cual limpia los archivos generados por el make.

Si se quiere utilizar la linea completa, se puede ingresar lo siguiente.

```
gcc main.c base64.c -o tp0 -g -std=c99 -Wall -Werror -O0
```

Se explican brevemente los parámetros utilizados. Estos se pueden modificar también si se quiere.

* **-o** Escribe lo compilado en un archivo de salida

* **-g** Genera información para debug.

* **-std=c99** Utiliza el estándar de C99.
* **-Wall** Activa todos los mensajes de *warning*.
* **-Werror** Todos los *warnings* son tratados como errores.
* **-O0** No aplica optimizaciones por parte del compilador. Se le puede cambiar el nivel.

4. Portabilidad

El programa fue hecho en su totalidad en el lenguaje C sin el uso de bibliotecas externas al lenguaje. El mismo fue probado en sistemas operativos Ubuntu y en la arquitectura de MIPS emulada por QEMU, por lo que no debería de haber problemas para portabilidad con respecto a sistemas operativos de tipo UNIX.

5. Casos de prueba

Para los casos de prueba, decidimos crear un archivo bash con el objetivo de poder automatizar las pruebas que realizamos. Estas testean diferentes casos y devuelven en la terminal si el resultado de la prueba es exitoso o no. A continuación, mencionamos las pruebas que hacemos y dejamos el archivo de pruebas:

- Se manda a codificar un archivo vacio. (Prueba 1)
- Se intenta decodificar un archivo que no esta en base 64. (Prueba 2)
- Se codifica y decodifica un archivo. (Prueba 3)
- No se envía ningún argumento ni entrada estándar, verificamos mensaje devuelto y que se devuelva distinto de 0. (Pruebas 4 y 12)
- Se pide la versión. (Prueba 5)
- Verificamos el uso de stderr enviando la salida a null. (Prueba 6)
- Se manda una serie de ceros a codificar. (Prueba 7)
- Se manda otro archivo vacio por stdin. (Pruebas 8 y 9)
- Codificamos el main.c, lo decodificamos, lo compilamos, y mandamos a codificar otro archivo. (Prueba 10)
- Se codifica el archivo objeto y lo decodificamos. (Prueba 11)
- Se llama al tp sin pasar el argumento de -i y se verifica que devuelva distinto de 0. (Prueba 13)
- Se manda argumento para -i pero no para -o. (Prueba 14)
- Se manda -i seguido de -o. (Prueba 15)
- Se envía a codificar y decodificar una serie de enters. (Prueba 16)
- Se envía -o seguido de -i sin entrada estándar. (Prueba 17)

Se muestra el archivo de pruebas.

```
    espacios() {
        echo
        echo
    }
    newline=$'\n'

    echo Comienza la ejecucion de las pruebas

espacios

    echo -e "\e[1m PRUEBA 1 \e[0m" - Se codifica un archivo vacio llamado "archivoVacio.txt"
    echo

    echo RESULTADO ESPERADO:
    resultadoEsperado=""
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$(./tp -i archivoVacio.txt -o codificado.txt 2>&1)
    echo
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi


espacios

    echo -e "\e[1m PRUEBA 2 \e[0m" -
    Se manda a decodificar el archivo que no esta en base 64 llamado "noEstoyEn64.txt":
    echo
    echo El archivo "noEstoyEn64.txt":
    cat noEstoyEn64.txt
    echo
    echo RESULTADO ESPERADO:
    resultadoEsperado=$(printf "El archivo enviado no esta en base 64.\n")
    echo "$resultadoEsperado"
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$(./tp -d -i noEstoyEn64.txt -o decodificado.txt 2>&1)
    echo "$resultadoObtenido"
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi


espacios

    echo -e "\e[1m PRUEBA 3 \e[0m" - Codificamos y decodificamos el archivo "quijote.txt"
    y esperamos obtener lo mismo:
    echo
```

```
    echo Archivo "quijote.txt" inicalmente:
    cat quijote.txt
    echo
    ./tp -i quijote.txt -o quijoteCodificado.txt
    echo Codificacion del archivo:
    cat quijoteCodificado.txt
    echo " "
    echo RESULTADO ESPERADO:
    resultadoEsperado="En un lugar de La Mancha de cuyo nombre no quiero acordarme"
    echo "$resultadoEsperado"
    ./tp -d -i quijoteCodificado.txt -o quijoteDecodificado.txt
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$(cat quijoteDecodificado.txt)
    echo "$resultadoObtenido"

    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi


espacios

    echo -e "\e[1m PRUEBA 4 \e[0m" - No mandamos ningun argumento al tp
    echo " "
    echo "Ejecutamos: ./tp"
    echo " "
    echo RESULTADO ESPERADO:
    resultadoEsperado=$(printf "Debe mandar mas argumentos, o mandar por entrada estandar.
                    Puede ver ayuda enviando el parametro -h \n")
    echo "$resultadoEsperado"
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$(./tp 2>&1)
    echo "$resultadoObtenido"
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi


espacios

    echo -e "\e[1m PRUEBA 5 \e[0m" - Pedimos la version y la obtenemos
    echo
    echo "Ejectuamos lo siguiente: ./tp -v"
    echo RESULTADO ESPERADO:
    resultadoEsperado=$(printf "Version 1.0.1")
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$( ./tp -v )
```

```
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 6 \e[0m"- Al usar /dev/null no se muestran errores por stderr
    echo RESULTADO ESPERADO:
    resultadoEsperado=""
    echo $resultadoEsperado
    echo Ejecutamos lo siguiente: "./tp 2> /dev/null"
    echo RESULTADO OBTENIDO:
    resultadoObtenido=$(./tp 2> /dev/null)
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 7 \e[0m" - Codificafamos un archivo con solo ceros
    echo Ejecutamos lo siguiente: "printf '\x00\x00\x00' | base64"
    echo RESULTADO ESPERADO:
    resultadoEsperado="$(printf '\x00\x00\x00' | base64)"
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    resultadoObtenido="$(printf '\x00\x00\x00' | ./tp)"
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 8 \e[0m" - Mandamos un archivo vacio por terminal y
    el error no se muestra al mandarlo a /dev/null
    echo Ejecutamos lo siguiente: "echo -n "" |./tp"
    echo Y luego: "echo -n "" | ./tp 2> /dev/null"
    echo RESULTADO ESPERADO:
    resultadoEsperado=""
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    echo -n "" | ./tp
    resultadoObtenido="$( echo -n "" | ./tp 2> /dev/null )"
```

```
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 9 \e[0m" - Mandamos un archivo vacio por terminal y no se hace nada
    echo Ejecutamos lo siguiente: "echo -n "" |./tp"
    echo RESULTADO ESPERADO:
    resultadoEsperado=""
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    echo
    resultadoObtenido="$(echo -n "" | ./tp)"
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi



espacios

    echo -e "\e[1m PRUEBA 10 \e[0m" - Codificamos el archivo main.c, lo decodificamos,
    luego lo compilamos y codificamos otro archivo
    echo Ejecutamos lo siguiente: "./tp -i main.c -o mainCodificado.txt"
    echo Luego: "./tp -d -i mainCodificado.txt -o main2.c"
    echo Luego: "gcc main2.c base64.c -o tpPrueba10"
    echo Luego: "./tpPrueba10 -i quijote.txt -o quijoteCodificadoPrueba10.txt"
    echo Luego "cat quijoteCodificadoPrueba10.txt"

    ./tp -i main.c -o mainCodificado.txt
    ./tp -d -i mainCodificado.txt -o main2.c
    gcc main2.c base64.c -o tpPrueba10
    ./tpPrueba10 -i quijote.txt -o quijoteCodificadoPrueba10.txt

    echo RESULTADO ESPERADO:
    resultadoEsperado="RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hh
                       IGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29yZGFybWUK"
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    resultadoObtenido="$(cat quijoteCodificadoPrueba10.txt)"
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
```

```
    fi

espacios

    echo -e "\e[1m PRUEBA 11 \e[0m" - Codificamos el archivo objeto tpPrueba11,
    lo decodificamos sin problemas
    echo Ejecutamos lo siguiente: "./tp -i tpPrueba11 -o tpPrueba11Codificado.txt"
    echo Luego: "./tp -d -i tpPrueba11Codificado.txt -o tpPrueba11Decodificado > /dev/null"
    echo Luego: "echo $?"

    echo ""

    ./tp -i tpPrueba11 -o tpPrueba11Codificado.txt
    ./tp -d -i tpPrueba11Codificado.txt -o tpPrueba11Decodificado


    echo RESULTADO ESPERADO:
    resultadoEsperado="0"
    echo $resultadoEsperado
    echo RESULTADO OBTENIDO:
    resultadoObtenido="$(echo $?)"
    echo $resultadoObtenido
    if [ "$resultadoEsperado" == "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 12 \e[0m" - No mandamos ningun argumento al tp y
    devolvemos distinto de 0
    echo " "
    echo "Ejecutamos: ./tp"
    echo "Luego: echo $ ?"
    echo " "
    ./tp
    resultadoObtenido=$(echo $?)
    echo "Ejecutamos y obtenemos:"
    echo "RESULTADO ESPERADO:"
    echo "Distinto de 0"
    echo "RESULTADO OBTENIDO:"
    printf "$resultadoObtenido\n"
    if [ "0" != "$resultadoObtenido" ];
    then
                    echo -e "\e[32m PRUEBA SUPERADA \e[0m"
    else
                    echo -e "\e[31m PRUEBA FALLADA \e[0m"
    fi

espacios

    echo -e "\e[1m PRUEBA 13 \e[0m" - No mandamos el argumento para el parametro -i,
```

```
  esperamos un resultado distinto de 0
  echo " "
  echo "Ejecutamos: ./tp -i"
  echo "Luego: echo $ ?"
  echo " "
  ./tp -i
  resultadoObtenido=$(echo $?)
  echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  echo "Distinto de 0"
  echo "RESULTADO OBTENIDO:"
  printf "$resultadoObtenido\n"
  if [ "0" != "$resultadoObtenido" ];
  then
            echo -e "\e[32m PRUEBA SUPERADA \e[0m"
  else
            echo -e "\e[31m PRUEBA FALLADA \e[0m"
  fi

espacios

  echo -e "\e[1m PRUEBA 14 \e[0m" - No mandamos el argumento para el parametro -o,
  se deberia mostrar ayuda, y se manda a salida estandar
  echo " "
  echo "Ejecutamos: ./tp -i quijote.txt -o"
  echo " "
  resultadoObtenido="$(./tp -i quijote.txt -o)"
  echo " "
  echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  resultadoEsperado="$(printf "RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVy
  printf "$resultadoEsperado\n"
  echo "RESULTADO OBTENIDO:"
  printf "$resultadoObtenido\n"
  if [ "$resultadoEsperado" == "$resultadoObtenido" ];
  then
                echo -e "\e[32m PRUEBA SUPERADA \e[0m"
  else
                echo -e "\e[31m PRUEBA FALLADA \e[0m"
  fi

espacios

  echo -e "\e[1m PRUEBA 15 \e[0m" - Mandamos -i seguido de -o,
  se interpreta como nombre de archivo y no se puede abrir. Se devuelve algo distinto de 0
  echo " "
  echo "Ejecutamos: ./tp -i -o"
  echo " "
  ./tp -i -o
  resultadoObtenido=$(echo $?)
  echo " "
  echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  printf "Algo distinto de 0\n"
```

```
  echo "RESULTADO OBTENIDO:"
  printf "$resultadoObtenido\n"
  if [ "0" != "$resultadoObtenido" ];
  then
                  echo -e "\e[32m PRUEBA SUPERADA \e[0m"
  else
                  echo -e "\e[31m PRUEBA FALLADA \e[0m"
  fi

espacios

  echo -e "\e[1m PRUEBA 16 \e[0m" - Mandamos una serie de enters"(4) y"
  esperamos obtener lo mismo al decodificarlo
  echo " "
  echo "Ejecutamos: printf '\n\n\n\n' | ./tp | ./tp -d"
  echo " "
  resultadoObtenido="$(printf "\n\n\n\n" | ./tp | ./tp -d)"
  echo " "
  echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  resultadoEsperado="$(printf "\n\n\n\n")"
  echo "$resultadoEsperado"
  echo "RESULTADO OBTENIDO:"
  printf "$resultadoObtenido"
  if [ "$resultadoEsperado" == "$resultadoObtenido" ];
  then
                  echo -e "\e[32m PRUEBA SUPERADA \e[0m"
  else
                  echo -e "\e[31m PRUEBA FALLADA \e[0m"
  fi

espacios

  echo -e "\e[1m PRUEBA 17 \e[0m" - Mandamos -o y -i solamente,
  el -i se toma como archivo y devuelve error, ya que no recibe entrada estandar
  echo " "
  echo "Ejecutamos: ./tp -o -i"
  echo " "
  ./tp -o -i
  resultadoObtenido=$(echo $?)
  echo " "
  echo "Ejecutamos y obtenemos:"
  echo "RESULTADO ESPERADO:"
  echo "Algo distinto de 0"
  echo "RESULTADO OBTENIDO:"
  printf "$resultadoObtenido\n"
  if [ "0" != "$resultadoObtenido" ];
  then
                  echo -e "\e[32m PRUEBA SUPERADA \e[0m"
  else
                  echo -e "\e[31m PRUEBA FALLADA \e[0m"
  fi
```

6. Conclusiones

Las conclusiones que obtuvimos realizando el trabajo son las siguientes.

- Pudimos familiarizarnos con algunas de las herramientas que utilizaremos a lo largo del curso.
- Aprendimos las diferentes herramientas que ofrece el lenguaje C para el manejo de bits (como los << y los operadoradores & y |) que vimos en Estructura del Computador.
- Aprendimos las propiedades y diferentes usos que puede tener la conversión a base 64.
- Aprendimos sobre el armado de un script bash para realizar la automatización de las pruebas.
- Se aprendió mas sobre el comportamiento de los diferentes manejos de entrada, tales como stdin, stdout, y stderr. Mas que nada su manejo en cuanto a su uso como un stream de FILE*.

7. Reentrega

Para la reentrega se realizaron los siguientes cambios:

- Se cambio la implementación realizada. Ahora se toman la cantidad de bytes necesarios únicamente y se codifican/decodifican directamente sobre el archivo de salida, sin utilizar memoria dinámica como hacíamos anteriormente.
- Se cambio el manejo de los parámetro. Se utiliza la biblioteca getopt, lo que permite un manejo mas flexible y sencillo de los argumentos.
- Los mensajes de error se escriben en stderr en vez de en stdout, a su vez al ocurrir un error, se devuelve un valor distinto de cero.
- Se agregaron pruebas automatizadas en un archivo bash.
- Se agregaron conclusiones:
  - Aprendimos a utilizar bash.
  - Aprendimos sobre el manejo de stdin, stdout y stderr.
- Se reescribieron los detalles de implementación.
- Se expandió la sección de pruebas.

8. Referencias

Link a la release en GitHub:
https://github.com/brunograssano/Organizacion-de-computadoras-fiuba/releases/tag/v1.0.1
Links de explicaciones de base 64:
https://en.wikipedia.org/wiki/Base64
https://www.lucidchart.com/techblog/2017/10/23/base64-encoding-a-visual-explanation/

9. Apéndice

9.1. Enunciado

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

1. Objetivos

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

2. Alcance

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

3. Requisitos

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

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 7), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa QEMU [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo Debian [2].

5. Base 64

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

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

6. Programa

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

6.1. Ejemplos

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

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information.
  -o, --output      Path to output file.
  -i, --input       Path to input file.
  -d, --decode      Decode a base64-encoded file.
Examples:

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

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

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

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

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

9.2. Código fuente en C

9.2.1. main.c

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <getopt.h>

#include "base64.h"

#define DECODE 'd'
#define INPUT 'i'
#define OUTPUT 'o'
#define VERSION 'v'
#define AYUDA 'h'
#define MAX_NOMBRE_ARCHIVO 256
#define MODO_LECTURA "r"
#define MODO_ESCRITURA "w"

const int LARGO_MAXIMO_ARCHIVO_POR_TERMINAL = 1000;
const int TERMINO = -1, VACIO = 0, ERROR = -1;;

////--------------OTROS----------------////

void imprimirAyuda(){
          printf("Uso: \n");
          printf(" tp -h\n");
          printf(" tp -v\n");
          printf(" tp [opciones]\n");
          printf("Opciones: \n");
          printf(" -v, --version    Imprime la version y termina el programa.\n");
          printf(" -h, --help       Imprime esta informacion.\n");
          printf(" -o, --output     Indica que le sigue la direccion al archivo de salida.\n");
          printf(" -i, --input      Indica que le sigue la direccion al archivo de entrada.\n");
          printf(" -d, --decode     Decodifica un archivo codificado en base 64 .\n");
          printf("Ejemplos: \n");
          printf("        tp -i input.txt -o output.txt\n");
          printf("        tp -d -i inputInBase64.txt -o outputInText.txt\n");
          printf("        cat input.txt | ./tp\n");
    printf("        cat input.txt | ./tp -d\n");
}

void mostrarVersion(){
         printf("Version 1.0.1\n");
}

void manejarParametros(int cantidadArgumentos,char* argumentos[],char archivoInput[MAX_NOMBRE_ARC
    static struct option opcionesLargas[] = {
       {"input", required_argument, 0, 'i'},
       {"decode", no_argument, 0, 'd'},
       {"help", no_argument, 0, 'h'},
```

```
        {"output", required_argument, 0, 'o'},
        {"version", no_argument, 0, 'v'},
        {0, 0, 0, 0}
    };
    int argumento;
    int indiceOpcion = 0;
    bool pidioAyuda = false, pidioVersion = false;

    while((argumento = getopt_long(cantidadArgumentos, argumentos, "i:dho:v",opcionesLargas, &ind
        switch (argumento) {
            case DECODE:
                (*pidioDecode) = true;
                break;
            case INPUT:
                strcpy(archivoInput,optarg);
                break;
            case OUTPUT:
                strcpy(archivoOutput,optarg);
                break;
            case VERSION:
                if(!pidioVersion){
                    (*pidioOtraOpcion) = true;
                    pidioVersion = true;
                    mostrarVersion();
                }
                break;
            case AYUDA:
                if(!pidioAyuda){
                    (*pidioOtraOpcion) = true;
                    pidioAyuda = true;
                    imprimirAyuda();
                }
                break;
            default:
                fprintf(stderr, "Puede ver ayuda enviando el parametro -h \n");
        }
    }

    if(optind < cantidadArgumentos){
        fprintf( stderr, "No son opciones validas las siguientes:\n");
        while (optind < cantidadArgumentos){
            fprintf( stderr, "%s\n", argumentos[optind++]);
        }
    }
}


////--------------DECODIFICACION----------------////

int decodificacion(FILE* entrada, FILE* salida){
    int estado = 0;
    while(!feof(entrada) && estado != ERROR){
        estado = decodificar(entrada,salida);
    }
```

```
    return estado;
}

int manejarDecodificacionArchivos(char archivoInput[MAX_NOMBRE_ARCHIVO],char archivoOutput[MAX_NO
  int estado;
  FILE* inputFile = fopen(archivoInput, MODO_LECTURA);
  if(inputFile == NULL){
    fprintf(stderr, "Hubo un error al abrir el archivo en: %s",archivoInput);
    return ERROR;
  }

  FILE* outputFile = fopen(archivoOutput, MODO_ESCRITURA);
  if(outputFile == NULL){
    fclose(inputFile);
    fprintf(stderr, "Hubo un error al crear el archivo de salida en: %s",archivoOutput);
    return ERROR;
  }

  estado = decodificacion(inputFile,outputFile);

  fclose(inputFile);
  fclose(outputFile);
  return estado;
}

int manejarDecodificacionEntradaArchivoSalidaEstandar(char archivoInput[MAX_NOMBRE_ARCHIVO]){
  int estado = 0;
  FILE* inputFile = fopen(archivoInput, MODO_LECTURA);
  if(inputFile == NULL){
    fprintf(stderr, "Hubo un error al abrir el archivo en: %s",archivoInput);
    return ERROR;
  }

  estado = decodificacion(inputFile,stdout);

  fclose(inputFile);
  return estado;
}

int manejarDecodificacionEntradaEstandarSalidaArchivo(char archivoOutput[MAX_NOMBRE_ARCHIVO]){
  int estado = 0;
  FILE* outputFile = fopen(archivoOutput, MODO_ESCRITURA);
  if(outputFile == NULL){
    fprintf(stderr, "Hubo un error al crear el archivo de salida en: %s",archivoOutput);
    return ERROR;
  }

  estado = decodificacion(stdin,outputFile);

  fclose(outputFile);
  return estado;
}

int manejarDecodificacion(char archivoInput[MAX_NOMBRE_ARCHIVO],char archivoOutput[MAX_NOMBRE_ARC
```

```
  int estado;
  int tamanioInput = strlen(archivoInput);
  int tamanioOutput = strlen(archivoOutput);

  if(tamanioInput > VACIO && tamanioOutput > VACIO){
      estado = manejarDecodificacionArchivos(archivoInput,archivoOutput);
  }
  else if(tamanioInput > VACIO && tamanioOutput == VACIO){
      estado = manejarDecodificacionEntradaArchivoSalidaEstandar(archivoInput);
  }
  else if(tamanioInput == VACIO && tamanioOutput > VACIO && !isatty(0)){
      estado = manejarDecodificacionEntradaEstandarSalidaArchivo(archivoOutput);
  }
  else if((tamanioInput == VACIO) && (tamanioOutput == VACIO) && !isatty(0)){
      estado = decodificacion(stdin,stdout);
  }
  else{
      fprintf(stderr, "Debe mandar mas argumentos, o mandar por entrada estandar. Puede ver ayuda
      estado = ERROR;
  }
  return estado;
}

////--------------CODIFICACION----------------////

int codificacion(FILE* entrada, FILE* salida){
    int estado = 0;
    while(!feof(entrada) && estado != ERROR){
        estado = codificar(entrada,salida);
    }
    return estado;
}

int manejarCodificacionArchivos(char archivoInput[MAX_NOMBRE_ARCHIVO],char archivoOutput[MAX_NOME
  int estado;
  FILE* inputFile = fopen(archivoInput, MODO_LECTURA);
  if(inputFile == NULL){
    fprintf(stderr, "Hubo un error al abrir el archivo de en: %s",archivoInput);
    return ERROR;
  }

  FILE* outputFile = fopen(archivoOutput, MODO_ESCRITURA);
  if(outputFile == NULL){
    fclose(inputFile);
    fprintf(stderr, "Hubo un error al crear el archivo de salida en: %s",archivoOutput);
    return ERROR;
  }

  estado = codificacion(inputFile,outputFile);

  fclose(inputFile);
  fclose(outputFile);
  return estado;
}
```

```
int manejarCodificacionEntradaArchivoSalidaEstandar(char archivoInput[MAX_NOMBRE_ARCHIVO]){
  int estado = 0;
  FILE* inputFile = fopen(archivoInput, MODO_LECTURA);
  if(inputFile == NULL){
    fprintf(stderr, "Hubo un error al abrir el archivo en: %s",archivoInput);
    return ERROR;
  }

  estado = codificacion(inputFile,stdout);

  fclose(inputFile);
  return estado;
}

int manejarCodificacionEntradaEstandarSalidaArchivo(char archivoOutput[MAX_NOMBRE_ARCHIVO]){
  int estado = 0;
  FILE* outputFile = fopen(archivoOutput, MODO_ESCRITURA);
  if(outputFile == NULL){
    fprintf(stderr, "Hubo un error al crear el archivo de salida en: %s",archivoOutput);
    return ERROR;
  }

  estado = codificacion(stdin,outputFile);

  fclose(outputFile);
  return estado;
}

int manejarCodificacion(char archivoInput[MAX_NOMBRE_ARCHIVO],char archivoOutput[MAX_NOMBRE_ARCHI
    int estado = 0;
    int tamanioInput = strlen(archivoInput);
    int tamanioOutput = strlen(archivoOutput);

    if(tamanioInput > VACIO && tamanioOutput > VACIO){
        estado = manejarCodificacionArchivos(archivoInput,archivoOutput);
    }
    else if(tamanioInput > VACIO && tamanioOutput == VACIO){
        estado = manejarCodificacionEntradaArchivoSalidaEstandar(archivoInput);
    }
    else if(tamanioInput == VACIO && tamanioOutput > VACIO && !isatty(0)){
        estado = manejarCodificacionEntradaEstandarSalidaArchivo(archivoOutput);
    }
    else if((tamanioInput == VACIO) && (tamanioOutput == VACIO) && !isatty(0)){
        estado = codificacion(stdin,stdout);
    }
    else{
        fprintf(stderr, "Debe mandar mas argumentos, o mandar por entrada estandar. Puede ver ayu
        estado = ERROR;
    }
    return estado;
}

////--------------MAIN----------------////
```

```
int main(int cantidadArgumentos, char* argumentos[]){
    char archivoInput[MAX_NOMBRE_ARCHIVO] = "";
    char archivoOutput[MAX_NOMBRE_ARCHIVO] = "";
    bool pidioDecode = false, pidioOtraOpcion = false;
    int estado = 0;

    manejarParametros(cantidadArgumentos,argumentos,archivoInput,archivoOutput,&pidioDecode, &pid

    if(pidioDecode && !pidioOtraOpcion){
        estado = manejarDecodificacion(archivoInput,archivoOutput);
    }
    else if(!pidioOtraOpcion){
        estado = manejarCodificacion(archivoInput,archivoOutput);
    }

    return estado;
}
```

9.2.2. base64.c

```
#include "base64.h"
#include <stdbool.h>
#define ERROR -1

#define BINARIO_63 0x3F
#define BINARIO_255 0xFF

const char caracteresBase64[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
const int valorAscii[] = { 62, -1, -1, -1, 63, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, -1, -1, -1
                    6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -
                    29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 4

/* Lee 3 bytes de plain, escribe 4 bytes en encoded */
int codificar(FILE* plain, FILE* encoded){
    if (plain == NULL){
        fprintf(stderr, "No se puede codificar, el metodo de entrada no existe\n");
        return ERROR;
    }

    if (encoded == NULL){
        fprintf(stderr, "No se puede codificar, el metodo de salida no existe\n");
        return ERROR;
    }

    char textoACodificar[3] = "";
    char salidaCodificada[4] = "====";
    int leido1, leido2, leido3;

    leido1 = fread ( &textoACodificar[0], 1, sizeof(char), plain );
    if(leido1<=0){
      return 0;
    }
    leido2 = fread ( &textoACodificar[1], 1, sizeof(char), plain );
```

```
    leido3 = fread ( &textoACodificar[2], 1, sizeof(char), plain );

    int caracterCodificandose = textoACodificar[0];

    caracterCodificandose =  (caracterCodificandose << 8 | textoACodificar[1]);
    caracterCodificandose =  (caracterCodificandose << 8 | textoACodificar[2]);

    salidaCodificada[0]   = caracteresBase64[(caracterCodificandose >> 18) & BINARIO_63];
    salidaCodificada[1] = caracteresBase64[(caracterCodificandose >> 12) & BINARIO_63];

    if(leido2>0){
        salidaCodificada[2] = caracteresBase64[(caracterCodificandose >> 6) & BINARIO_63];
    }
    if(leido3>0){
        salidaCodificada[3] = caracteresBase64[caracterCodificandose & BINARIO_63];
    }

    fwrite(salidaCodificada, sizeof(char), 4, encoded);

    return 0;
}


bool esCaracterValido(char caracter){
    return ((caracter >= '0' && caracter <= '9') || (caracter >= 'A' && caracter <= 'Z') ||
            (caracter >= 'a' && caracter <= 'z') || (caracter == '+' || caracter == '/' || caract
}

/* Lee 4 bytes de encoded, escribe 3 bytes en plain */
int decodificar(FILE* encoded, FILE* plain){
    if (encoded == NULL){
        fprintf(stderr, "No se puede decodificar, el metodo de entrada no existe\n");
        return ERROR;
    }

    if (plain == NULL){
        fprintf(stderr, "No se puede decodificar, el metodo de salida no existe\n");
        return ERROR;
    }

    char textoADecodificar[4] = "";
    char salidaDecodificada[3] = "";
    int leidos = fread(textoADecodificar, 4, sizeof(char), encoded);
    if(leidos<=0){
      return 0;
    }

    for (int i=0; i<4; i++) {
        if (!esCaracterValido(textoADecodificar[i])) {
            fprintf(stderr, "El archivo enviado no esta en base 64.\n");
            return ERROR;
        }
    }
```

```
    int caracterDecodificandose = valorAscii[textoADecodificar[0]-43];
    caracterDecodificandose = (caracterDecodificandose << 6) | valorAscii[textoADecodificar[1]-43
    caracterDecodificandose = textoADecodificar[2]=='=' ? (caracterDecodificandose << 6) : ((cara
    caracterDecodificandose = textoADecodificar[3]=='=' ? (caracterDecodificandose << 6) : ((cara

    salidaDecodificada[0] = (caracterDecodificandose >> 16) & BINARIO_255;
    if (textoADecodificar[2] != '='){
      salidaDecodificada[1] = (caracterDecodificandose >> 8) & BINARIO_255;
    }
    if (textoADecodificar[3] != '='){
      salidaDecodificada[2] = caracterDecodificandose & BINARIO_255;
    }

    fwrite(&salidaDecodificada[0], sizeof(char), 1, plain);
    if(salidaDecodificada[1] != '\0'){
        fwrite(&salidaDecodificada[1], sizeof(char), 1, plain);
    }
    if(salidaDecodificada[2] != '\0'){
        fwrite(&salidaDecodificada[2], sizeof(char), 1, plain);
    }

    return 0;
}
```

9.3. Código generado por el compilador .s

1.3. Código generado por el compilador .s

max width=center

```
        .file   1 "conversionBase64.c"
        .section .mdebug.abi32
        .previous
        .nan    legacy
        .module fp=xx
        .module nooddspreg
        .abicalls
        .globl  CANTIDAD_MINIMA_ARGUMENTOS
        .rdata
        .align  2
        .type   CANTIDAD_MINIMA_ARGUMENTOS, @object
        .size   CANTIDAD_MINIMA_ARGUMENTOS, 4
CANTIDAD_MINIMA_ARGUMENTOS:
        .word   1
        .globl  CANTIDAD_ARGUMENTOS_PARA_CODIFICAR
        .align  2
        .type   CANTIDAD_ARGUMENTOS_PARA_CODIFICAR, @object
        .size   CANTIDAD_ARGUMENTOS_PARA_CODIFICAR, 4
CANTIDAD_ARGUMENTOS_PARA_CODIFICAR:
        .word   5
        .globl  CANTIDAD_ARGUMENTOS_PARA_DECODIFICAR
        .align  2
        .type   CANTIDAD_ARGUMENTOS_PARA_DECODIFICAR, @object
        .size   CANTIDAD_ARGUMENTOS_PARA_DECODIFICAR, 4
CANTIDAD_ARGUMENTOS_PARA_DECODIFICAR:
        .word   6
        .globl  POS_ARCHIVO_INPUT_ENCODE
        .align  2
        .type   POS_ARCHIVO_INPUT_ENCODE, @object
        .size   POS_ARCHIVO_INPUT_ENCODE, 4
POS_ARCHIVO_INPUT_ENCODE:
        .word   2
        .globl  POS_COMANDO_OUTPUT_ENCODE
        .align  2
        .type   POS_COMANDO_OUTPUT_ENCODE, @object
        .size   POS_COMANDO_OUTPUT_ENCODE, 4
POS_COMANDO_OUTPUT_ENCODE:
        .word   3
        .globl  POS_ARCHIVO_OUTPUT_ENCODE
        .align  2
        .type   POS_ARCHIVO_OUTPUT_ENCODE, @object
        .size   POS_ARCHIVO_OUTPUT_ENCODE, 4
POS_ARCHIVO_OUTPUT_ENCODE:
        .word   4
        .globl  POS_ARCHIVO_INPUT_DECODE
        .align  2
        .type   POS_ARCHIVO_INPUT_DECODE, @object
        .size   POS_ARCHIVO_INPUT_DECODE, 4
POS_ARCHIVO_INPUT_DECODE:
        .word   3
        .globl  POS_COMANDO_OUTPUT_DECODE
```

```
        .align  2
        .type   POS_COMANDO_OUTPUT_DECODE, @object
        .size   POS_COMANDO_OUTPUT_DECODE, 4
POS_COMANDO_OUTPUT_DECODE:
        .word   4
        .globl  POS_ARCHIVO_OUTPUT_DECODE
        .align  2
        .type   POS_ARCHIVO_OUTPUT_DECODE, @object
        .size   POS_ARCHIVO_OUTPUT_DECODE, 4
POS_ARCHIVO_OUTPUT_DECODE:
        .word   5
        .globl  ARCHIVO_VACIO
        .align  2
        .type   ARCHIVO_VACIO, @object
        .size   ARCHIVO_VACIO, 4
ARCHIVO_VACIO:
        .space  4
        .globl  LARGO_MAXIMO_ARCHIVO_POR_TERMINAL
        .align  2
        .type   LARGO_MAXIMO_ARCHIVO_POR_TERMINAL, @object
        .size   LARGO_MAXIMO_ARCHIVO_POR_TERMINAL, 4
LARGO_MAXIMO_ARCHIVO_POR_TERMINAL:
        .word   1000
        .align  2
$LC1:
        .ascii  "Uso: \000"
        .align  2
$LC2:
        .ascii  " tp0 -h\000"
        .align  2
$LC3:
        .ascii  " tp0 -V\000"
        .align  2
$LC4:
        .ascii  " tp0 [opciones]\000"
        .align  2
$LC5:
        .ascii  "Opciones: \000"
        .align  2
$LC6:
        .ascii  " -V, --version     Imprime la version y termina el progra"
        .ascii  "ma.\000"
        .align  2
$LC7:
        .ascii  " -h, --help        Imprime esta informacion.\000"
        .align  2
$LC8:
        .ascii  " -o, --output      Indica que le sigue la direccion al ar"
        .ascii  "chivo de salida.\000"
        .align  2
$LC9:
        .ascii  " -i, --input       Indica que le sigue la direccion al ar"
        .ascii  "chivo de entrada.\000"
        .align  2
```

```
$LC10:
        .ascii  " -d, --decode      Decodifica un archivo codificado en ba"
        .ascii  "se 64 .\000"
        .align  2
$LC11:
        .ascii  "Ejemplos: \000"
        .align  2
$LC12:
        .ascii  "\011tp0 -i input.txt -o output.txt\000"
        .align  2
$LC13:
        .ascii  "\011tp0 -d -i inputInBase64.txt -o outputInText.txt\000"
        .align  2
$LC14:
        .ascii  "\011cat input.txt | ./tp0  (Unicamente para codificar)\000"
        .text
        .align  2
        .globl  imprimirAyuda
        .set    nomips16
        .set    nomicromips
        .ent    imprimirAyuda
        .type   imprimirAyuda, @function
imprimirAyuda:
        .frame  $fp,32,$31              # vars= 0, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-32
        sw      $31,28($sp)
        sw      $fp,24($sp)
        move    $fp,$sp
        .cprestore      16
        lw      $2,%got($LC1)($28)
        addiu   $4,$2,%lo($LC1)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC2)($28)
        addiu   $4,$2,%lo($LC2)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC3)($28)
        addiu   $4,$2,%lo($LC3)
```

```
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC4)($28)
        addiu   $4,$2,%lo($LC4)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC5)($28)
        addiu   $4,$2,%lo($LC5)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC6)($28)
        addiu   $4,$2,%lo($LC6)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC7)($28)
        addiu   $4,$2,%lo($LC7)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC8)($28)
        addiu   $4,$2,%lo($LC8)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC9)($28)
        addiu   $4,$2,%lo($LC9)
```

```
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC10)($28)
        addiu   $4,$2,%lo($LC10)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC11)($28)
        addiu   $4,$2,%lo($LC11)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC12)($28)
        addiu   $4,$2,%lo($LC12)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC13)($28)
        addiu   $4,$2,%lo($LC13)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got($LC14)($28)
        addiu   $4,$2,%lo($LC14)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        nop
        move    $sp,$fp
```

```
        lw      $31,28($sp)
        lw      $fp,24($sp)
        addiu   $sp,$sp,32
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    imprimirAyuda
        .size   imprimirAyuda, .-imprimirAyuda
        .rdata
        .align  2
$LC15:
        .ascii  "Version 0.0.1\000"
        .text
        .align  2
        .globl  mostrarVersion
        .set    nomips16
        .set    nomicromips
        .ent    mostrarVersion
        .type   mostrarVersion, @function
mostrarVersion:
        .frame  $fp,32,$31              # vars= 0, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-32
        sw      $31,28($sp)
        sw      $fp,24($sp)
        move    $fp,$sp
        .cprestore      16
        lw      $2,%got($LC15)($28)
        addiu   $4,$2,%lo($LC15)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        nop
        move    $sp,$fp
        lw      $31,28($sp)
        lw      $fp,24($sp)
        addiu   $sp,$sp,32
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    mostrarVersion
        .size   mostrarVersion, .-mostrarVersion
```

```
        .align  2
        .globl  esMultiploDeTres
        .set    nomips16
        .set    nomicromips
        .ent    esMultiploDeTres
        .type   esMultiploDeTres, @function
esMultiploDeTres:
        .frame  $fp,8,$31               # vars= 0, regs= 1/0, args= 0, gp= 0
        .mask   0x40000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        addiu   $sp,$sp,-8
        sw      $fp,4($sp)
        move    $fp,$sp
        sw      $4,8($fp)
        lw      $4,8($fp)
        li      $2,1431633920                   # 0x55550000
        ori     $2,$2,0x5556
        mult    $4,$2
        mfhi    $3
        sra     $2,$4,31
        subu    $3,$3,$2
        move    $2,$3
        sll     $2,$2,1
        addu    $2,$2,$3
        subu    $3,$4,$2
        sltu    $2,$3,1
        andi    $2,$2,0x00ff
        move    $sp,$fp
        lw      $fp,4($sp)
        addiu   $sp,$sp,8
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    esMultiploDeTres
        .size   esMultiploDeTres, .-esMultiploDeTres
        .align  2
        .globl  calcularTamanioArchivoSalidaBase64
        .set    nomips16
        .set    nomicromips
        .ent    calcularTamanioArchivoSalidaBase64
        .type   calcularTamanioArchivoSalidaBase64, @function
calcularTamanioArchivoSalidaBase64:
        .frame  $fp,40,$31              # vars= 8, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-40
        sw      $31,36($sp)
```

```
        sw      $fp,32($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,40($fp)
        lw      $2,40($fp)
        sw      $2,24($fp)
        lw      $4,40($fp)
        lw      $2,%got(esMultiploDeTres)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,esMultiploDeTres
1:      jalr    $25
        nop

        lw      $28,16($fp)
        xori    $2,$2,0x1
        andi    $2,$2,0x00ff
        beq     $2,$0,$L6
        nop

        lw      $4,40($fp)
        li      $2,1431633920                   # 0x55550000
        ori     $2,$2,0x5556
        mult    $4,$2
        mfhi    $3
        sra     $2,$4,31
        subu    $3,$3,$2
        move    $2,$3
        sll     $2,$2,1
        addu    $2,$2,$3
        subu    $3,$4,$2
        li      $2,3                    # 0x3
        subu    $2,$2,$3
        lw      $3,24($fp)
        addu    $2,$3,$2
        sw      $2,24($fp)
$L6:
        lw      $2,24($fp)
        li      $3,1431633920                   # 0x55550000
        ori     $3,$3,0x5556
        mult    $2,$3
        mfhi    $3
        sra     $2,$2,31
        subu    $2,$3,$2
        sw      $2,24($fp)
        lw      $2,24($fp)
        sll     $2,$2,2
        sw      $2,24($fp)
        lw      $2,24($fp)
        move    $sp,$fp
        lw      $31,36($sp)
        lw      $fp,32($sp)
        addiu   $sp,$sp,40
        jr      $31
        nop
```

```
        .set    macro
        .set    reorder
        .end    calcularTamanioArchivoSalidaBase64
        .size   calcularTamanioArchivoSalidaBase64, .-calcularTamanioArchivoSalidaBas
        .align  2
        .globl  adicionarCaracteresAlFinalDeLaSecuencia
        .set    nomips16
        .set    nomicromips
        .ent    adicionarCaracteresAlFinalDeLaSecuencia
        .type   adicionarCaracteresAlFinalDeLaSecuencia, @function
adicionarCaracteresAlFinalDeLaSecuencia:
        .frame  $fp,8,$31               # vars= 0, regs= 1/0, args= 0, gp= 0
        .mask   0x40000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        addiu   $sp,$sp,-8
        sw      $fp,4($sp)
        move    $fp,$sp
        sw      $4,8($fp)
        sw      $5,12($fp)
        sw      $6,16($fp)
        sw      $7,20($fp)
        lw      $2,16($fp)
        addiu   $3,$2,1
        lw      $2,28($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L9
        nop

        lw      $2,20($fp)
        addiu   $2,$2,2
        lw      $3,24($fp)
        addu    $2,$3,$2
        lw      $3,12($fp)
        sra     $3,$3,6
        andi    $3,$3,0x3f
        lw      $4,8($fp)
        addu    $3,$4,$3
        lb      $3,0($3)
        sb      $3,0($2)
        b       $L10
        nop

$L9:
        lw      $2,20($fp)
        addiu   $2,$2,2
        lw      $3,24($fp)
        addu    $2,$3,$2
        li      $3,61                   # 0x3d
        sb      $3,0($2)
$L10:
        lw      $2,16($fp)
```

```
        addiu   $3,$2,2
        lw      $2,28($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L11
        nop

        lw      $2,20($fp)
        addiu   $2,$2,3
        lw      $3,24($fp)
        addu    $2,$3,$2
        lw      $3,12($fp)
        andi    $3,$3,0x3f
        lw      $4,8($fp)
        addu    $3,$4,$3
        lb      $3,0($3)
        sb      $3,0($2)
        b       $L13
        nop

$L11:
        lw      $2,20($fp)
        addiu   $2,$2,3
        lw      $3,24($fp)
        addu    $2,$3,$2
        li      $3,61                   # 0x3d
        sb      $3,0($2)
$L13:
        nop
        move    $sp,$fp
        lw      $fp,4($sp)
        addiu   $sp,$sp,8
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    adicionarCaracteresAlFinalDeLaSecuencia
        .size   adicionarCaracteresAlFinalDeLaSecuencia, .-adicionarCaracteresAlFinal
        .rdata
        .align  2
$LC16:
        .ascii  "El archivo esta vacio, no hay nada para codificar.\000"
        .align  2
$LC17:
        .ascii  "Ha ocurrido un error durante la codificacion.\000"
        .align  2
$LC18:
        .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123"
        .ascii  "456789+/\000"
        .text
        .align  2
        .globl  codificarTexto
        .set    nomips16
        .set    nomicromips
```

```
        .ent    codificarTexto
        .type   codificarTexto, @function
codificarTexto:
        .frame  $fp,128,$31             # vars= 88, regs= 2/0, args= 24, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-128
        sw      $31,124($sp)
        sw      $fp,120($sp)
        move    $fp,$sp
        .cprestore      24
        sw      $4,128($fp)
        sw      $5,132($fp)
        move    $3,$0
        lw      $2,132($fp)
        bne     $2,$3,$L15
        nop

        lw      $2,%got($LC16)($28)
        addiu   $4,$2,%lo($LC16)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,24($fp)
        move    $2,$0
        b       $L24
        nop

$L15:
        lw      $4,132($fp)
        lw      $2,%got(calcularTamanioArchivoSalidaBase64)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,calcularTamanioArchivoSalidaBase64
1:      jalr    $25
        nop

        lw      $28,24($fp)
        sw      $2,40($fp)
        lw      $2,40($fp)
        addiu   $2,$2,1
        move    $4,$2
        lw      $2,%call16(malloc)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,malloc
1:      jalr    $25
        nop

        lw      $28,24($fp)
```

```
        sw      $2,44($fp)
        lw      $2,44($fp)
        bne     $2,$0,$L17
        nop

        lw      $2,%got($LC17)($28)
        addiu   $4,$2,%lo($LC17)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,24($fp)
        move    $2,$0
        b       $L24
        nop

$L17:
        lw      $2,%got($LC18)($28)
        addiu   $3,$fp,52
        addiu   $2,$2,%lo($LC18)
        li      $4,65                   # 0x41
        move    $6,$4
        move    $5,$2
        move    $4,$3
        lw      $2,%call16(memcpy)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,memcpy
1:      jalr    $25
        nop

        lw      $28,24($fp)
        lw      $2,40($fp)
        lw      $3,44($fp)
        addu    $2,$3,$2
        sb      $0,0($2)
        sw      $0,32($fp)
        sw      $0,36($fp)
        b       $L18
        nop

$L23:
        lw      $2,32($fp)
        lw      $3,128($fp)
        addu    $2,$3,$2
        lbu     $2,0($2)
        sw      $2,48($fp)
        lw      $2,32($fp)
        addiu   $3,$2,1
        lw      $2,132($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L19
        nop
```

```
        lw      $2,48($fp)
        sll     $2,$2,8
        lw      $3,32($fp)
        addiu   $3,$3,1
        lw      $4,128($fp)
        addu    $3,$4,$3
        lbu     $3,0($3)
        or      $2,$2,$3
        b       $L20
        nop

$L19:
        lw      $2,48($fp)
        sll     $2,$2,8
$L20:
        sw      $2,48($fp)
        lw      $2,32($fp)
        addiu   $3,$2,2
        lw      $2,132($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L21
        nop

        lw      $2,48($fp)
        sll     $2,$2,8
        lw      $3,32($fp)
        addiu   $3,$3,2
        lw      $4,128($fp)
        addu    $3,$4,$3
        lbu     $3,0($3)
        or      $2,$2,$3
        b       $L22
        nop

$L21:
        lw      $2,48($fp)
        sll     $2,$2,8
$L22:
        sw      $2,48($fp)
        lw      $2,36($fp)
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,48($fp)
        sra     $3,$3,18
        andi    $3,$3,0x3f
        addiu   $4,$fp,32
        addu    $3,$4,$3
        lb      $3,20($3)
        sb      $3,0($2)
        lw      $2,36($fp)
        addiu   $2,$2,1
        lw      $3,44($fp)
        addu    $2,$3,$2
```

```
        lw      $3,48($fp)
        sra     $3,$3,12
        andi    $3,$3,0x3f
        addiu   $4,$fp,32
        addu    $3,$4,$3
        lb      $3,20($3)
        sb      $3,0($2)
        addiu   $3,$fp,52
        lw      $2,132($fp)
        sw      $2,20($sp)
        lw      $2,44($fp)
        sw      $2,16($sp)
        lw      $7,36($fp)
        lw      $6,32($fp)
        lw      $5,48($fp)
        move    $4,$3
        lw      $2,%got(adicionarCaracteresAlFinalDeLaSecuencia)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,adicionarCaracteresAlFinalDeLaSecuencia
1:      jalr    $25
        nop

        lw      $28,24($fp)
        lw      $2,32($fp)
        addiu   $2,$2,3
        sw      $2,32($fp)
        lw      $2,36($fp)
        addiu   $2,$2,4
        sw      $2,36($fp)
$L18:
        lw      $2,132($fp)
        addiu   $3,$2,-1
        lw      $2,32($fp)
        slt     $2,$2,$3
        bne     $2,$0,$L23
        nop

        lw      $2,44($fp)
$L24:
        move    $sp,$fp
        lw      $31,124($sp)
        lw      $fp,120($sp)
        addiu   $sp,$sp,128
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    codificarTexto
        .size   codificarTexto, .-codificarTexto
        .align  2
        .globl  hacerConversionABase64
        .set    nomips16
        .set    nomicromips
```

```
        .ent    hacerConversionABase64
        .type   hacerConversionABase64, @function
hacerConversionABase64:
        .frame  $fp,48,$31              # vars= 16, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-48
        sw      $31,44($sp)
        sw      $fp,40($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,48($fp)
        sw      $5,52($fp)
        li      $6,2                    # 0x2
        move    $5,$0
        lw      $4,48($fp)
        lw      $2,%call16(fseek)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fseek
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,48($fp)
        lw      $2,%call16(ftell)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,ftell
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,24($fp)
        move    $6,$0
        move    $5,$0
        lw      $4,48($fp)
        lw      $2,%call16(fseek)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fseek
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,24($fp)
        addiu   $2,$2,1
        move    $4,$2
        lw      $2,%call16(malloc)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,malloc
1:      jalr    $25
        nop
```

```
        lw      $28,16($fp)
        sw      $2,28($fp)
        lw      $2,28($fp)
        bne     $2,$0,$L26
        nop

        lw      $2,%got($LC17)($28)
        addiu   $4,$2,%lo($LC17)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L25
        nop

$L26:
        lw      $2,24($fp)
        lw      $7,48($fp)
        move    $6,$2
        li      $5,1                    # 0x1
        lw      $4,28($fp)
        lw      $2,%call16(fread)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fread
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $5,24($fp)
        lw      $4,28($fp)
        lw      $2,%got(codificarTexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,codificarTexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,32($fp)
        lw      $2,32($fp)
        bne     $2,$0,$L28
        nop

        lw      $4,28($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L25
```

```
        nop

$L28:
        lw      $5,52($fp)
        lw      $4,32($fp)
        lw      $2,%call16(fputs)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fputs
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,28($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,32($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
$L25:
        move    $sp,$fp
        lw      $31,44($sp)
        lw      $fp,40($sp)
        addiu   $sp,$sp,48
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    hacerConversionABase64
        .size   hacerConversionABase64, .-hacerConversionABase64
        .rdata
        .align  2
$LC19:
        .ascii  "-o\000"
        .align  2
$LC20:
        .ascii  "r\000"
        .align  2
$LC21:
        .ascii  "El archivo ingresado para codificar no existe.\000"
        .align  2
$LC22:
        .ascii  "w\000"
        .align  2
```

```
$LC23:
        .ascii  "Hubo un error al crear el archivo de salida.\000"
        .align  2
$LC24:
        .ascii  "Faltan argumentos para poder codificar. Se muestra ayuda"
        .ascii  ".\000"
        .align  2
$LC25:
        .ascii  "Se mandaron argumentos de mas. Se muestra ayuda.\000"
        .text
        .align  2
        .globl  convertirABase64
        .set    nomips16
        .set    nomicromips
        .ent    convertirABase64
        .type   convertirABase64, @function
convertirABase64:
        .frame  $fp,40,$31              # vars= 8, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-40
        sw      $31,36($sp)
        sw      $fp,32($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,40($fp)
        sw      $5,44($fp)
        li      $2,5                    # 0x5
        lw      $3,40($fp)
        bne     $3,$2,$L30
        nop

        li      $2,3                    # 0x3
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
        lw      $2,%got($LC19)($28)
        addiu   $5,$2,%lo($LC19)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L30
        nop

        li      $2,2                    # 0x2
```

```
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
        lw      $2,%got($LC20)($28)
        addiu   $5,$2,%lo($LC20)
        move    $4,$3
        lw      $2,%call16(fopen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fopen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,24($fp)
        lw      $2,24($fp)
        bne     $2,$0,$L31
        nop

        lw      $2,%got($LC21)($28)
        addiu   $4,$2,%lo($LC21)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L29
        nop

$L31:
        li      $2,4                    # 0x4
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
        lw      $2,%got($LC22)($28)
        addiu   $5,$2,%lo($LC22)
        move    $4,$3
        lw      $2,%call16(fopen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fopen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,28($fp)
        lw      $2,28($fp)
        bne     $2,$0,$L33
        nop

        lw      $2,%got($LC23)($28)
        addiu   $4,$2,%lo($LC23)
```

```
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,24($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L29
        nop

$L33:
        lw      $5,28($fp)
        lw      $4,24($fp)
        lw      $2,%got(hacerConversionABase64)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,hacerConversionABase64
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,24($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,28($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L29
        nop

$L30:
        li      $2,5                    # 0x5
        lw      $3,40($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L34
        nop
```

```
        lw      $2,%got($LC24)($28)
        addiu   $4,$2,%lo($LC24)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L35
        nop

$L34:
        lw      $2,%got($LC25)($28)
        addiu   $4,$2,%lo($LC25)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
$L35:
        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
$L29:
        move    $sp,$fp
        lw      $31,36($sp)
        lw      $fp,32($sp)
        addiu   $sp,$sp,40
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    convertirABase64
        .size   convertirABase64, .-convertirABase64
        .align  2
        .globl  calcularTamanioArchivoSalidaDeTexto
        .set    nomips16
        .set    nomicromips
        .ent    calcularTamanioArchivoSalidaDeTexto
        .type   calcularTamanioArchivoSalidaDeTexto, @function
calcularTamanioArchivoSalidaDeTexto:
        .frame  $fp,32,$31              # vars= 16, regs= 1/0, args= 0, gp= 8
        .mask   0x40000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
```

```
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        sw      $4,32($fp)
        sw      $5,36($fp)
        lw      $2,36($fp)
        sw      $2,8($fp)
        lw      $2,8($fp)
        bgez    $2,$L37
        nop

        addiu   $2,$2,3
$L37:
        sra     $2,$2,2
        sw      $2,8($fp)
        lw      $3,8($fp)
        move    $2,$3
        sll     $2,$2,1
        addu    $2,$2,$3
        sw      $2,8($fp)
        lw      $2,8($fp)
        sw      $2,12($fp)
        sb      $0,16($fp)
        b       $L38
        nop

$L42:
        lw      $2,12($fp)
        lw      $3,32($fp)
        addu    $2,$3,$2
        lbu     $3,0($2)
        li      $2,61                   # 0x3d
        bne     $3,$2,$L39
        nop

        lw      $2,8($fp)
        addiu   $2,$2,-1
        sw      $2,8($fp)
        b       $L40
        nop

$L39:
        li      $2,1                    # 0x1
        sb      $2,16($fp)
$L40:
        lw      $2,12($fp)
        addiu   $2,$2,-1
        sw      $2,12($fp)
$L38:
        lw      $2,12($fp)
        blez    $2,$L41
        nop

        lbu     $2,16($fp)
```

```
        xori    $2,$2,0x1
        andi    $2,$2,0x00ff
        bne     $2,$0,$L42
        nop

$L41:
        lw      $2,8($fp)
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    calcularTamanioArchivoSalidaDeTexto
        .size   calcularTamanioArchivoSalidaDeTexto, .-calcularTamanioArchivoSalidaDe
        .align  2
        .globl  esCaracterValido
        .set    nomips16
        .set    nomicromips
        .ent    esCaracterValido
        .type   esCaracterValido, @function
esCaracterValido:
        .frame  $fp,8,$31               # vars= 0, regs= 1/0, args= 0, gp= 0
        .mask   0x40000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        addiu   $sp,$sp,-8
        sw      $fp,4($sp)
        move    $fp,$sp
        move    $2,$4
        sb      $2,8($fp)
        lb      $2,8($fp)
        slt     $2,$2,48
        bne     $2,$0,$L45
        nop

        lb      $2,8($fp)
        slt     $2,$2,58
        bne     $2,$0,$L46
        nop

$L45:
        lb      $2,8($fp)
        slt     $2,$2,65
        bne     $2,$0,$L47
        nop

        lb      $2,8($fp)
        slt     $2,$2,91
        bne     $2,$0,$L46
        nop
```

```
$L47:
        lb      $2,8($fp)
        slt     $2,$2,97
        bne     $2,$0,$L48
        nop

        lb      $2,8($fp)
        slt     $2,$2,123
        bne     $2,$0,$L46
        nop

$L48:
        lb      $3,8($fp)
        li      $2,43                   # 0x2b
        beq     $3,$2,$L46
        nop

        lb      $3,8($fp)
        li      $2,47                   # 0x2f
        beq     $3,$2,$L46
        nop

        lb      $3,8($fp)
        li      $2,61                   # 0x3d
        bne     $3,$2,$L49
        nop

$L46:
        li      $2,1                    # 0x1
        b       $L50
        nop

$L49:
        move    $2,$0
$L50:
        andi    $2,$2,0x1
        andi    $2,$2,0x00ff
        move    $sp,$fp
        lw      $fp,4($sp)
        addiu   $sp,$sp,8
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    esCaracterValido
        .size   esCaracterValido, .-esCaracterValido
        .rdata
        .align  2
$LC26:
        .ascii  "El archivo esta vacio, no hay nada para decodificar.\000"
        .align  2
$LC27:
```

```
        .ascii  "El archivo enviado no esta en base 64.\000"
        .align  2
$LC28:
        .ascii  "Ha ocurrido un error durante la decodificacion.\000"
        .align  2
$LC0:
        .word   62
        .word   -1
        .word   -1
        .word   -1
        .word   63
        .word   52
        .word   53
        .word   54
        .word   55
        .word   56
        .word   57
        .word   58
        .word   59
        .word   60
        .word   61
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   0
        .word   1
        .word   2
        .word   3
        .word   4
        .word   5
        .word   6
        .word   7
        .word   8
        .word   9
        .word   10
        .word   11
        .word   12
        .word   13
        .word   14
        .word   15
        .word   16
        .word   17
        .word   18
        .word   19
        .word   20
        .word   21
        .word   22
        .word   23
        .word   24
        .word   25
```

```
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   -1
        .word   26
        .word   27
        .word   28
        .word   29
        .word   30
        .word   31
        .word   32
        .word   33
        .word   34
        .word   35
        .word   36
        .word   37
        .word   38
        .word   39
        .word   40
        .word   41
        .word   42
        .word   43
        .word   44
        .word   45
        .word   46
        .word   47
        .word   48
        .word   49
        .word   50
        .word   51
        .text
        .align  2
        .globl  decodificarBase64ATexto
        .set    nomips16
        .set    nomicromips
        .ent    decodificarBase64ATexto
        .type   decodificarBase64ATexto, @function
decodificarBase64ATexto:
        .frame  $fp,376,$31             # vars= 344, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-376
        sw      $31,372($sp)
        sw      $fp,368($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,376($fp)
        sw      $5,380($fp)
        move    $3,$0
```

```
        lw      $2,380($fp)
        bne     $2,$3,$L53
        nop

        lw      $2,%got($LC26)($28)
        addiu   $4,$2,%lo($LC26)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $2,$0
        b       $L68
        nop

$L53:
        lw      $2,380($fp)
        andi    $2,$2,0x3
        beq     $2,$0,$L55
        nop

        lw      $2,%got($LC27)($28)
        addiu   $4,$2,%lo($LC27)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $2,$0
        b       $L68
        nop

$L55:
        sw      $0,24($fp)
        b       $L56
        nop

$L58:
        lw      $2,24($fp)
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $2,0($2)
        seb     $2,$2
        move    $4,$2
        lw      $2,%got(esCaracterValido)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,esCaracterValido
1:      jalr    $25
        nop
```

```
        lw      $28,16($fp)
        xori    $2,$2,0x1
        andi    $2,$2,0x00ff
        beq     $2,$0,$L57
        nop

        lw      $2,%got($LC27)($28)
        addiu   $4,$2,%lo($LC27)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $2,$0
        b       $L68
        nop

$L57:
        lw      $2,24($fp)
        addiu   $2,$2,1
        sw      $2,24($fp)
$L56:
        lw      $3,24($fp)
        lw      $2,380($fp)
        slt     $2,$3,$2
        bne     $2,$0,$L58
        nop

        lw      $5,380($fp)
        lw      $4,376($fp)
        lw      $2,%got(calcularTamanioArchivoSalidaDeTexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,calcularTamanioArchivoSalidaDeTexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,36($fp)
        lw      $2,36($fp)
        addiu   $2,$2,1
        move    $4,$2
        lw      $2,%call16(malloc)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,malloc
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,40($fp)
        lw      $2,40($fp)
        bne     $2,$0,$L59
        nop
```

```
        lw      $2,%got($LC28)($28)
        addiu   $4,$2,%lo($LC28)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $2,$0
        b       $L68
        nop

$L59:
        lw      $2,%got($LC0)($28)
        addiu   $3,$fp,48
        addiu   $2,$2,%lo($LC0)
        li      $4,320                  # 0x140
        move    $6,$4
        move    $5,$2
        move    $4,$3
        lw      $2,%call16(memcpy)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,memcpy
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $0,28($fp)
        sw      $0,32($fp)
        b       $L60
        nop

$L67:
        lw      $2,28($fp)
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $2,0($2)
        addiu   $2,$2,-43
        sll     $2,$2,2
        addiu   $3,$fp,24
        addu    $2,$3,$2
        lw      $2,24($2)
        sw      $2,44($fp)
        lw      $2,44($fp)
        sll     $3,$2,6
        lw      $2,28($fp)
        addiu   $2,$2,1
        lw      $4,376($fp)
        addu    $2,$4,$2
        lbu     $2,0($2)
        addiu   $2,$2,-43
        sll     $2,$2,2
```

```
        addiu   $4,$fp,24
        addu    $2,$4,$2
        lw      $2,24($2)
        or      $2,$3,$2
        sw      $2,44($fp)
        lw      $2,28($fp)
        addiu   $2,$2,2
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $3,0($2)
        li      $2,61                   # 0x3d
        bne     $3,$2,$L61
        nop

        lw      $2,44($fp)
        sll     $2,$2,6
        b       $L62
        nop

$L61:
        lw      $2,44($fp)
        sll     $3,$2,6
        lw      $2,28($fp)
        addiu   $2,$2,2
        lw      $4,376($fp)
        addu    $2,$4,$2
        lbu     $2,0($2)
        addiu   $2,$2,-43
        sll     $2,$2,2
        addiu   $4,$fp,24
        addu    $2,$4,$2
        lw      $2,24($2)
        or      $2,$3,$2
$L62:
        sw      $2,44($fp)
        lw      $2,28($fp)
        addiu   $2,$2,3
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $3,0($2)
        li      $2,61                   # 0x3d
        bne     $3,$2,$L63
        nop

        lw      $2,44($fp)
        sll     $2,$2,6
        b       $L64
        nop

$L63:
        lw      $2,44($fp)
        sll     $3,$2,6
        lw      $2,28($fp)
        addiu   $2,$2,3
```

```
        lw      $4,376($fp)
        addu    $2,$4,$2
        lbu     $2,0($2)
        addiu   $2,$2,-43
        sll     $2,$2,2
        addiu   $4,$fp,24
        addu    $2,$4,$2
        lw      $2,24($2)
        or      $2,$3,$2
$L64:
        sw      $2,44($fp)
        lw      $2,32($fp)
        lw      $3,40($fp)
        addu    $2,$3,$2
        lw      $3,44($fp)
        sra     $3,$3,16
        seb     $3,$3
        sb      $3,0($2)
        lw      $2,28($fp)
        addiu   $2,$2,2
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $3,0($2)
        li      $2,61                   # 0x3d
        beq     $3,$2,$L65
        nop

        lw      $2,32($fp)
        addiu   $2,$2,1
        lw      $3,40($fp)
        addu    $2,$3,$2
        lw      $3,44($fp)
        sra     $3,$3,8
        seb     $3,$3
        sb      $3,0($2)
$L65:
        lw      $2,28($fp)
        addiu   $2,$2,3
        lw      $3,376($fp)
        addu    $2,$3,$2
        lbu     $3,0($2)
        li      $2,61                   # 0x3d
        beq     $3,$2,$L66
        nop

        lw      $2,32($fp)
        addiu   $2,$2,2
        lw      $3,40($fp)
        addu    $2,$3,$2
        lw      $3,44($fp)
        seb     $3,$3
        sb      $3,0($2)
$L66:
        lw      $2,28($fp)
```

```
        addiu   $2,$2,4
        sw      $2,28($fp)
        lw      $2,32($fp)
        addiu   $2,$2,3
        sw      $2,32($fp)
$L60:
        lw      $3,28($fp)
        lw      $2,380($fp)
        slt     $2,$3,$2
        bne     $2,$0,$L67
        nop

        lw      $2,40($fp)
$L68:
        move    $sp,$fp
        lw      $31,372($sp)
        lw      $fp,368($sp)
        addiu   $sp,$sp,376
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    decodificarBase64ATexto
        .size   decodificarBase64ATexto, .-decodificarBase64ATexto
        .align  2
        .globl  hacerConversionATexto
        .set    nomips16
        .set    nomicromips
        .ent    hacerConversionATexto
        .type   hacerConversionATexto, @function
hacerConversionATexto:
        .frame  $fp,48,$31              # vars= 16, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-48
        sw      $31,44($sp)
        sw      $fp,40($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,48($fp)
        sw      $5,52($fp)
        li      $6,2                    # 0x2
        move    $5,$0
        lw      $4,48($fp)
        lw      $2,%call16(fseek)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fseek
1:      jalr    $25
        nop
```

```
        lw      $28,16($fp)
        lw      $4,48($fp)
        lw      $2,%call16(ftell)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,ftell
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,24($fp)
        move    $6,$0
        move    $5,$0
        lw      $4,48($fp)
        lw      $2,%call16(fseek)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fseek
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,24($fp)
        addiu   $2,$2,1
        move    $4,$2
        lw      $2,%call16(malloc)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,malloc
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,28($fp)
        lw      $2,28($fp)
        bne     $2,$0,$L70
        nop

        lw      $2,%got($LC28)($28)
        addiu   $4,$2,%lo($LC28)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L69
        nop

$L70:
        lw      $2,24($fp)
        lw      $7,48($fp)
        move    $6,$2
        li      $5,1                    # 0x1
        lw      $4,28($fp)
        lw      $2,%call16(fread)($28)
```

```
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fread
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $5,24($fp)
        lw      $4,28($fp)
        lw      $2,%got(decodificarBase64ATexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,decodificarBase64ATexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,32($fp)
        lw      $2,32($fp)
        bne     $2,$0,$L72
        nop

        lw      $4,28($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L69
        nop

$L72:
        lw      $5,52($fp)
        lw      $4,32($fp)
        lw      $2,%call16(fputs)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fputs
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,28($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,32($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
```

```
        nop

        lw      $28,16($fp)
$L69:
        move    $sp,$fp
        lw      $31,44($sp)
        lw      $fp,40($sp)
        addiu   $sp,$sp,48
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    hacerConversionATexto
        .size   hacerConversionATexto, .-hacerConversionATexto
        .rdata
        .align  2
$LC29:
        .ascii  "El archivo ingresado para decodificar no existe.\000"
        .align  2
$LC30:
        .ascii  "Faltan argumentos para poder decodificar. Se muestra ayu"
        .ascii  "da.\000"
        .text
        .align  2
        .globl  decodificarATexto
        .set    nomips16
        .set    nomicromips
        .ent    decodificarATexto
        .type   decodificarATexto, @function
decodificarATexto:
        .frame  $fp,40,$31              # vars= 8, regs= 2/0, args= 16, gp= 8
        .mask   0xc0000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-40
        sw      $31,36($sp)
        sw      $fp,32($sp)
        move    $fp,$sp
        .cprestore      16
        sw      $4,40($fp)
        sw      $5,44($fp)
        li      $2,6                    # 0x6
        lw      $3,40($fp)
        bne     $3,$2,$L74
        nop

        li      $2,4                    # 0x4
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
```

```
        lw      $2,%got($LC19)($28)
        addiu   $5,$2,%lo($LC19)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L74
        nop

        li      $2,3                    # 0x3
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
        lw      $2,%got($LC20)($28)
        addiu   $5,$2,%lo($LC20)
        move    $4,$3
        lw      $2,%call16(fopen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fopen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,24($fp)
        lw      $2,24($fp)
        bne     $2,$0,$L75
        nop

        lw      $2,%got($LC29)($28)
        addiu   $4,$2,%lo($LC29)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L73
        nop

$L75:
        li      $2,5                    # 0x5
        sll     $2,$2,2
        lw      $3,44($fp)
        addu    $2,$3,$2
        lw      $3,0($2)
        lw      $2,%got($LC22)($28)
        addiu   $5,$2,%lo($LC22)
        move    $4,$3
```

```
        lw      $2,%call16(fopen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fopen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,28($fp)
        lw      $2,28($fp)
        bne     $2,$0,$L77
        nop

        lw      $2,%got($LC23)($28)
        addiu   $4,$2,%lo($LC23)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,24($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L73
        nop

$L77:
        lw      $5,28($fp)
        lw      $4,24($fp)
        lw      $2,%got(hacerConversionATexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,hacerConversionATexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,24($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,28($fp)
        lw      $2,%call16(fclose)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,fclose
```

```
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L73
        nop

$L74:
        li      $2,6                    # 0x6
        lw      $3,40($fp)
        slt     $2,$3,$2
        beq     $2,$0,$L78
        nop

        lw      $2,%got($LC30)($28)
        addiu   $4,$2,%lo($LC30)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L79
        nop

$L78:
        lw      $2,%got($LC25)($28)
        addiu   $4,$2,%lo($LC25)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
$L79:
        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
$L73:
        move    $sp,$fp
        lw      $31,36($sp)
        lw      $fp,32($sp)
        addiu   $sp,$sp,40
        jr      $31
        nop

        .set    macro
        .set    reorder
```

```
        .end    decodificarATexto
        .size   decodificarATexto, .-decodificarATexto
        .rdata
        .align  2
$LC31:
        .ascii  "El comando output fue mal utilizado\000"
        .align  2
$LC32:
        .ascii  "-d\000"
        .align  2
$LC33:
        .ascii  "-h\000"
        .align  2
$LC34:
        .ascii  "-V\000"
        .align  2
$LC35:
        .ascii  "-i\000"
        .align  2
$LC36:
        .ascii  "No es un argumento valido\000"
        .align  2
$LC37:
        .ascii  "Faltan argumentos\000"
        .align  2
$LC38:
        .ascii  "%[^\012]\000"
        .text
        .align  2
        .globl  main
        .set    nomips16
        .set    nomicromips
        .ent    main
        .type   main, @function
main:
        .frame  $fp,88,$31              # vars= 24, regs= 10/0, args= 16, gp= 8
        .mask   0xc0ff0000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $25
        .set    nomacro
        addiu   $sp,$sp,-88
        sw      $31,84($sp)
        sw      $fp,80($sp)
        sw      $23,76($sp)
        sw      $22,72($sp)
        sw      $21,68($sp)
        sw      $20,64($sp)
        sw      $19,60($sp)
        sw      $18,56($sp)
        sw      $17,52($sp)
        sw      $16,48($sp)
        move    $fp,$sp
        .cprestore      16
```

```
        sw      $4,88($fp)
        sw      $5,92($fp)
        li      $2,1                    # 0x1
        lw      $3,88($fp)
        slt     $2,$2,$3
        beq     $2,$0,$L81
        nop

        lw      $2,92($fp)
        addiu   $2,$2,4
        lw      $3,0($2)
        lw      $2,%got($LC19)($28)
        addiu   $5,$2,%lo($LC19)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L82
        nop

        lw      $2,%got($LC31)($28)
        addiu   $4,$2,%lo($LC31)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L82:
        lw      $2,92($fp)
        addiu   $2,$2,4
        lw      $3,0($2)
        lw      $2,%got($LC32)($28)
        addiu   $5,$2,%lo($LC32)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
```

```
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L84
        nop

        lw      $5,92($fp)
        lw      $4,88($fp)
        lw      $2,%got(decodificarATexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,decodificarATexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L84:
        lw      $2,92($fp)
        addiu   $2,$2,4
        lw      $3,0($2)
        lw      $2,%got($LC33)($28)
        addiu   $5,$2,%lo($LC33)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L85
        nop

        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L85:
        lw      $2,92($fp)
        addiu   $2,$2,4
        lw      $3,0($2)
        lw      $2,%got($LC34)($28)
        addiu   $5,$2,%lo($LC34)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
```

```
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L86
        nop

        lw      $2,%got(mostrarVersion)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,mostrarVersion
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L86:
        lw      $2,92($fp)
        addiu   $2,$2,4
        lw      $3,0($2)
        lw      $2,%got($LC35)($28)
        addiu   $5,$2,%lo($LC35)
        move    $4,$3
        lw      $2,%call16(strcmp)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strcmp
1:      jalr    $25
        nop

        lw      $28,16($fp)
        bne     $2,$0,$L87
        nop

        lw      $5,92($fp)
        lw      $4,88($fp)
        lw      $2,%got(convertirABase64)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,convertirABase64
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L87:
        lw      $2,%got($LC36)($28)
        addiu   $4,$2,%lo($LC36)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
```

```
        nop

        lw      $28,16($fp)
        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L81:
        move    $4,$0
        lw      $2,%call16(isatty)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,isatty
1:      jalr    $25
        nop

        lw      $28,16($fp)
        beq     $2,$0,$L89
        nop

        lw      $2,%got($LC37)($28)
        addiu   $4,$2,%lo($LC37)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $2,%got(imprimirAyuda)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,imprimirAyuda
1:      jalr    $25
        nop

        lw      $28,16($fp)
        b       $L88
        nop

$L89:
        move    $2,$sp
        sw      $2,40($fp)
        li      $2,1000                 # 0x3e8
        addiu   $2,$2,-1
        sw      $2,24($fp)
        li      $2,1000                 # 0x3e8
        move    $23,$2
        move    $22,$0
        srl     $2,$23,29
```

```
        sll     $18,$22,3
        or      $18,$2,$18
        sll     $19,$23,3
        li      $2,1000                 # 0x3e8
        move    $21,$2
        move    $20,$0
        srl     $2,$21,29
        sll     $16,$20,3
        or      $16,$2,$16
        sll     $17,$21,3
        li      $2,1000                 # 0x3e8
        addiu   $2,$2,7
        srl     $2,$2,3
        sll     $2,$2,3
        subu    $sp,$sp,$2
        addiu   $2,$sp,16
        addiu   $2,$2,0
        sw      $2,28($fp)
        lw      $2,28($fp)
        move    $5,$2
        lw      $2,%got($LC38)($28)
        addiu   $4,$2,%lo($LC38)
        lw      $2,%call16(__isoc99_scanf)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,__isoc99_scanf
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $16,28($fp)
        move    $4,$16
        lw      $2,%call16(strlen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strlen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        addu    $2,$16,$2
        li      $3,10                   # 0xa
        sb      $3,0($2)
        sb      $0,1($2)
        lw      $16,28($fp)
        lw      $2,28($fp)
        move    $4,$2
        lw      $2,%call16(strlen)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,strlen
1:      jalr    $25
        nop

        lw      $28,16($fp)
        move    $5,$2
        move    $4,$16
```

```
        lw      $2,%got(codificarTexto)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,codificarTexto
1:      jalr    $25
        nop

        lw      $28,16($fp)
        sw      $2,32($fp)
        lw      $4,32($fp)
        lw      $2,%call16(puts)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,puts
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $4,32($fp)
        lw      $2,%call16(free)($28)
        move    $25,$2
        .reloc  1f,R_MIPS_JALR,free
1:      jalr    $25
        nop

        lw      $28,16($fp)
        lw      $sp,40($fp)
$L88:
        move    $2,$0
        move    $sp,$fp
        lw      $31,84($sp)
        lw      $fp,80($sp)
        lw      $23,76($sp)
        lw      $22,72($sp)
        lw      $21,68($sp)
        lw      $20,64($sp)
        lw      $19,60($sp)
        lw      $18,56($sp)
        lw      $17,52($sp)
        lw      $16,48($sp)
        addiu   $sp,$sp,88
        jr      $31
        nop

        .set    macro
        .set    reorder
        .end    main
        .size   main, .-main
        .ident  "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```