Universidad de los Andes
Facultad de Ingeniería

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
ISIS1304
FUNDAMENTOS DE INFRAESTRUCTURA TECNOLÓGICA

Proyecto 1 - 2017-2

Este proyecto vale 15% de la nota del curso.
Debe ser elaborado en grupo (3 integrantes).
No se permite ningún tipo de consulta entre grupos.
Se debe entregar por Sicua a más tardar el 5 de noviembre a las 23:55

A. OBJETIVOS

- Practicar el lenguaje C y desarrollar un programa de complejidad pequeña.
- Conocer las operaciones de C para el manejo de bits.
- Aplicar lo anterior en un programa que permite comprimir un archivo binario.

B. DESCRIPCIÓN DEL PROBLEMA

La compresión de datos se refiere a codificar información utilizando menos bits de los que utiliza la representación original. Hay diferentes procedimientos para la compresión de datos. Por ejemplo, removiendo la redundancia de datos o información poco relevante. La compresión de datos es útil pues permite reducir los recursos utilizados para almacenar y transmitir información.

Al comprimir y descomprimir información se utilizan recursos computacionales. Sin embargo, como se reduce el espacio utilizado, se produce una solución de compromiso (decisión de ingeniería en la cual se pierde una cualidad a cambio de otra).

En el presente proyecto, se quiere explorar una técnica de compresión de archivos de texto usando un código de tamaño variable. El programa recibe el nombre de un archivo que contiene la información del texto y genera el archivo comprimido correspondiente, en el cual cada carácter es reemplazado por el código asignado.

Formato del archivo

Para este ejercicio sólo se considerarán archivos de texto con mayúsculas (A-Z, sin incluir la letra "Ñ"). Esto implica que son 26 caracteres. No se considerarán tampoco espacios, guiones, puntos ni ningún otro símbolo.

El código que se debe utilizar tiene el siguiente formato (esto es un ejemplo, no implica que el código vaya a ser estrictamente como en la tabla):

Carácter ASCII	**ASCII**	**Bits que lo representan (representación binaria)**	**Longitud actual(bits)**	**Código de compresión**	**Longitud código de compresión (bits)**
A	65	01000001	8	0	1
…	….	…	…	…	…
Z	90	01011010	8	10110	5

Tabla 1 Formato del código a utilizar

Como se puede observar en la tabla 1, mientras que con el código ASCII necesitamos 8 bits (1 byte) para representar un carácter, con el nuevo código necesitamos menos bits para representar el mismo carácter. Adicionalmente, en el código ASCII todos los caracteres tienen el mismo tamaño, mientras que en el nuevo el tamaño varía. Esto implica que, si traducimos un mensaje codificado en ASCII a nuestro nuevo código, su tamaño en bits y por tanto en bytes se puede llegar a reducir eligiendo apropiadamente el tamaño en bits de cada código, por lo que habremos comprimido (y codificado) el mensaje.

Codificación

A continuación, se muestra un ejemplo en el que se ilustra la traducción de un mensaje en ASCII usando un código específico:

Mensaje: `THISISATEST`

Código:

Caracter ASCII	**ASCII**	**Bits que lo representan (representación binaria)**	**Longitud actual(bits)**	**Código de compresión**	**Longitud de código de compresión(bits)**
A	65	01000001	8	110	3
E	69	01000101	8	111	3
H	72	01001000	8	010	3
I	73	01001010	8	011	3
S	83	01010011	8	10	2
T	84	01010100	8	00	2

Dado que el mensaje inicial está codificado en ASCII, se tendrá la siguiente cadena de bits representando el mensaje (se representa cada byte con "|"):

01010100 **|** 01001000 **|** 01001010 **|** 01010011 **|** 01001010 **|** 01010011 **|** 01000001**|** 01010100 **|** 01000101 **|** 01010011 **|** 01010100

Longitud en bits: 8 bits * 11 = 88 bits.

Para comprimir el texto, se tiene que sustituir la letra que se quiere con el código asignado en la tabla de la siguiente manera:

t h i s i s a t e s t

00 010 011 10 011 10 110 00 111 10 00

La longitud en bits de este mensaje es 27 bits. De esta manera se redujo el tamaño del texto de 88 bits a 27 bits. Nótese que, dado que ahora cada letra es representada con menos bits, en un mismo byte pueden caber 1, 2, 3 o más letras, y a su vez, puede que una letra pueda estar representada parcialmente en un byte y el resto en el siguiente. Para ver esto, se marcan los bytes en el resultado de la compresión:

t h i s i s a t e s t

00 010 011 | 10 011 10 1|10 00 111 1| 0 00

En rojo: Casos para este ejemplo en que una letra pueda estar representada parcialmente en un byte y el resto en el siguiente

En azul: Ejemplo de que más de una letra puede estar representada en el mismo byte.

Estructuras de datos

Para guardar los datos de los archivos en este programa se usarán estructuras que tienen el tamaño del arreglo (campo `tamanio`) y un puntero al arreglo con los datos en sí.

```
//-- Estructura de los vectores para procesar los archivos

typedef struct archivo
{
	int tamanio;
	unsigned char *informacion;
} Archivo;
```

Ambos estructuras están declaradas dentro del main para luego ser pasadas como parámetros a los procedimientos del programa.

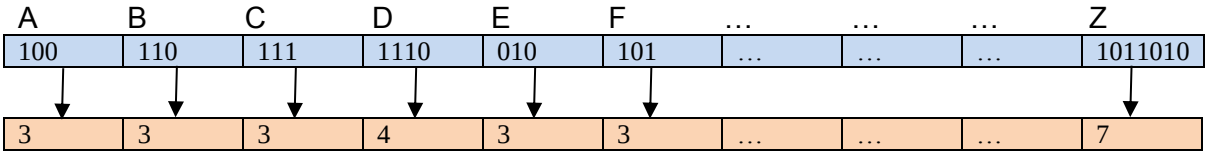
```
Archivo * archivo = (Archivo *) malloc (sizeof (Archivo));

Archivo * archivoCodificado = (Archivo *) malloc (sizeof
(Archivo));
```

En `archivo` el procedimiento de lectura guardará los bytes del archivo que se pretende comprimir. La descripción de este procedimiento se encuentra posteriormente en este documento.

En `archivoCodificado` el procedimiento `codificar` guardará los bytes del archivo comprimido para que luego el procedimiento de escritura lo pueda guardar de nuevo en disco. La descripción de este procedimiento se encuentra posteriormente en este documento.

Además, se tienen las siguientes estructuras para representar el nuevo valor que tendrá una letra según su frecuencia:

Se manejarán 2 arreglos de igual tamaño uno de tipo char y otro de enteros. Uno contiene en valor código de una letra y será de tipo char (`unsigned char codigoCompresion [26]`, en azul) y el otro contiene la longitud de dicho código (`int longitudCodigo[26]`, en naranja). Estos arreglos están declarados al inicio del programa.



Cada posición del arreglo es equivalente a la posición de la letra en el abecedario suponiendo que se empieza desde 0. De esta manera, el código y la longitud del código de la letra A estará en la posición 0 de cada arreglo, de la letra B está en la posición 1 y así sucesivamente.

Note que cada posición del arreglo `codigoCompresion` tiene un byte, pero solo se usan los bits indicados por el arreglo `longitudCodigo`. Por ejemplo, en la posición 0 (la 'A'), el valor guardado es 10000000, pero como la posición `longitudCodigo` correspondiente dice 3, solo se usan los tres primeros bits. Igualmente, en la posición 3 (la 'D'), el valor guardado es 11100000, pero como la posición `longitudCodigo` correspondiente dice 4, solo se usan los cuatro primeros bits.

El programa

En el archivo adjunto ("`main.c`"), encuentra el esquema del programa. El programa se invoca por línea de comando.

El programa ya tiene definidos: el procedimiento `main()`, un procedimiento para cargar el código en memoria principal, un procedimiento para leer un archivo y un procedimiento para escribir el archivo comprimido. Deben usar estos procedimientos en el programa. **Estos procedimientos no deben ser modificados**.

- El procedimiento de lectura recibe como parámetro una estructura `Archivo * archivo` en la cual carga la información que luego será codificada en otros procedimientos.
- El procedimiento de cargar del código de compresión El procedimiento lee el archivo y carga el código en el arreglo de correspondiente, una letra por cada posición.
- El procedimiento `readFileCode` usado por el procedimiento que carga el código de compresión.
- El procedimiento `writeFile` de escritura del archivo ya codificado.

Complete los procedimientos restantes según lo indicado a continuación y en el esqueleto del programa adjunto.

- `int codificar(Archivo * archivo, Archivo * archivocodificado )`
  Esta función se encarga de codificar cada uno de los valores que se encuentran en la estructura `archivo`, y pasa su código a la información referenciada en la estructura `archivoCodificado`. Para ello almacena la secuencia de unos y ceros que componen el código (como números) en un vector de `char`, el cual será procesado por la función `agregarAlArreglo`, como se indica a continuación. Nota: la notación "`char datos[]`" es equivalente a "`char * datos`": es un apuntador a una cadena de caracteres.

- `void agregarAlArreglo( unsigned char datosCodificados[], unsigned char codigo, int longitud, int posicionBit, int nuevoTamanio)`

  Esta función recibe como parámetros el vector de datos codificados (`datosCodificados`), el código que se debe insertar (`codigo`), la longitud de este último (l`ongitud`) la posición donde debe insertar el `codigo` en el vector codificado (`posicionBit`) y `nuevoTamanio, que` es el tamanio del archivo comprimido.

  Se encarga de convertir los datos de `codigo` a bits e insertarlos en la posición que les corresponde en `datosCodificados`.

Para desarrollar el programa pueden crear las funciones adicionales que necesiten, las cuales deben estar debidamente comentadas y documentadas.

C. ESPECIFICACIONES

- Los programas se deben escribir en C (en el compilador de Visual C++). Nota importante: los programas se calificarán únicamente usando el compilador de Visual Studio 2015; si el programa no compila con este, se considerará que no corre (así compile con otros compiladores o en otros ambientes).
- Legibilidad del programa: indentar el programa; escribir comentarios explicando el código.
- Debe respetar la estructura del código entregado. En particular, debe usar los procedimientos y variables del esqueleto.

D. CONDICIONES DE ENTREGA

- Entregar el código fuente junto con el ejecutable en un archivo `*.zip`. **Al comienzo del archivo fuente escriba los nombres de los miembros, sus códigos y correos, de lo contrario no será evaluado (ver esqueleto)**. Si su programa no funciona o si su solución tiene particularidades, puede enviar un archivo .doc explicando por qué cree que no funciona o qué fue lo que hizo.
- El trabajo se realiza en grupos de **3** personas. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes).
- Se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. Dicha sustentación puede afectar la nota de todos los miembros.
- El proyecto debe ser entregado por Sicua por uno solo de los integrantes.
- **Se debe entregar por Sicua a más tardar el 5 de noviembre a las 23:55.**

E. CASOS DE PRUEBA

Para hacer las pruebas, adjunto al proyecto, se encuentran tres ejemplos completos, cada

uno con el archivo original y el archivo comprimido.

1. Prueba 1
2. Prueba 2
3. Prueba 3

F. CRITERIOS DE CALIFICACIÓN PARA LOS PROGRAMAS

La calificación consta de dos partes:

- Ejecución (50%). Para las funciones propuestas se harán 5 pruebas: tres de los casos de prueba entregados y otros dos nuevos con un mensaje diferente para codificar. Para cada uno de los mensajes se revisará si la salida es correcta o no según los requerimientos establecidos en el enunciado. Cada una de las pruebas vale 10%.
- Inspección del código (50%). Se consideran tres aspectos:
    - 10% - legibilidad (nombres dicientes para variables, comentarios e indentación)
    - 20% - manejo de bits (uso de los operadores de bits de c: `>>`, `&`, etc.)
    - 20% - manejo de la estructura de datos (recorrido, manejo de los elementos).

G. RECOMENDACIONES

- Recuerden que los trabajos hechos en grupo y entregados individualmente (o en grupos diferentes al original) son una forma de fraude académico.