Universidad de los Andes
Facultad de Ingeniería

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
ISIS1304
FUNDAMENTOS DE INFRAESTRUCTURA TECNOLÓGICA

Proyecto 1 - 2017-1

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 23 de abril 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 ocultar mensajes en imágenes.

B. DESCRIPCIÓN DEL PROBLEMA

"La esteganografía es la disciplina en la que se estudian y aplican técnicas que permiten el ocultamiento de mensajes u objetos, dentro de otros, llamados portadores, de modo que no se perciba su existencia. Es una mezcla de artes y técnicas que se combinan para conformar la práctica de ocultar y enviar información sensible a través de portador, para que pueda pasar desapercibida". http://es.wikipedia.org/wiki/Esteganografía

En el caso de las imágenes, cuando se tienen tres componentes de color (RGB) de 8 bits, se dispone de más de 16 millones de colores, lo cual supera la capacidad del ojo para diferenciar colores. En consecuencia, pequeños cambios en las componentes de color (en los bits menos significativos) no son detectados por el ojo; estos bits menos significativos se pueden usar para almacenar el mensaje sin afectar notoriamente la imagen.

Se quiere escribir un programa el cual oculte un mensaje (cadena de caracteres) dentro de una imagen en formato bmp, colocando partes del mensaje en los bits menos significativos de los componentes RGB de cada pixel de la imagen.

Formato de la imagen

El formato para el guardado de imágenes a utilizar es bmp de 24 bits, lo que quiere decir que para cada uno de los componentes RGB de un pixel se reserva 1 Byte:

Un pixel en bmp de 24 bits quedaría así:

R	G	B
00000000	00000000	00000000

Y una imagen sería una secuencia de estos pixeles.

Estructuras de datos

La imagen se representará por medio de la siguiente estructura:

```
// La representación de la imagen
typedef struct img {
        int ancho;
        int alto;
        unsigned char *informacion;
} Imagen;
```

Las estructuras de tipo `Imagen` constan de un alto y un ancho, representados en enteros, y un apuntador a la información de la imagen, la cual consiste en una secuencia de pixeles de 3 bytes únicamente (sin más información adicional).

Codificación

Una imagen es un conjunto de pixeles; el método para la inserción consiste en reemplazar los bits menos significativos de los componentes RGB de cada pixel de la imagen con bits del mensaje. Para esto se procede como se ilustra en el siguiente ejemplo:

Se tiene la siguiente secuencia de bytes en una imagen de 2x2:

$Pixel_{1,1}$			**$Pixel_{1,2}$**		
R	**G**	**B**	**R**	**G**	**B**
10011010	10110011	11011000	10110011	00011011	01010110
$Pixel_{2,1}$			**$Pixel_{2,2}$**		
R	**G**	**B**	**R**	**G**	**B**
00111111	00000001	01000011	11100011	00011111	01011000

El mensaje es "am", lo cual en binario, sin tener en cuenta el carácter nulo de final de cadena, es:

01100001	01101101

Para almacenar el mensaje en los tres bits menos significativos de cada componente de color, partimos el mensaje en grupos de 3 bits. Nos queda:

011	000	010	110	110	100

Note que se rellenó con 00 al final, puesto que en el último grupo solo quedaba un bit del mensaje.

Colocando el mensaje en los bits menos significativos de la imagen quedaría entonces en la siguiente forma:

Pixel			**Pixel**		
R	**G**	**B**	**R**	**G**	**B**
10011**011**	10110**000**	11011**010**	10110**110**	00011**110**	01010**100**
Pixel			**Pixel**		
R	**G**	**B**	**R**	**G**	**B**
00111111	00000001	01000011	11100011	00011111	01011000

Note que después de este procedimiento la imagen se puede seguir viendo con cambios menores en los colores de los pixeles modificados que contienen partes del mensaje "am".

El cambio sobre los pixeles de la imagen no debe ser muy grande, por lo general con 1, 2 o 3 bits por byte el cambio en los colores es imperceptible o no muy notorio para el ojo humano.

Ahora bien, para recuperar el mensaje a partir de la secuencia anterior, se debe tener la longitud del mensaje original y el número de bits por byte que se usó; con esto es posible reconstruir el mensaje a partir de los bits menos significativos de los componentes RGB de la imagen.

El programa

En el archivo adjunto (“`main.c`”), encuentra el esquema del programa. El programa se invoca por línea de comando, y recibe un parámetro: el nombre del archivo de imagen sobre el que se va a trabajar.

El programa ya tiene definidos: el procedimiento `main()`, un procedimiento para leer una imagen y uno para escribir una imagen. Deben usar estos procedimientos en el programa. **Estos procedimientos no deben ser modificados**.

El procedimiento `main()` recibe como parámetro la ruta completa del archivo `.bmp` que se da como un argumento de comando. Al correr el programa este permite 2 opciones: insertar un mensaje o extraer un mensaje.

1. Para insertar un mensaje, el programa pide que se escriba el mensaje en cuestión y el número de bits por byte que se quiere usar.

2. Para leer un mensaje, pide la longitud del mensaje y el número de bits por byte que se usó. El programa reporta el mensaje recuperado en la consola.

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

- `void insertarMensaje(Imagen * img , char mensaje[], int n);`

Esta función se encarga de colocar un mensaje en una imagen. Recibe como parámetros un puntero a la imagen, el mensaje que se va a poner en la imagen y el número de bits que se usan en cada componente de color para almacenar el mensaje. Nota: la notación "`char mensaje[]`" es equivalente a "`char * mensaje`": es un apuntador a una cadena de caracteres.

- `void leerMensaje(Imagen * img, char msg[], int l, int n);`

Realiza la operación inversa a la función anterior: se encarga de leer un mensaje guardado previamente en la imagen. Recibe como parámetros: un puntero a la imagen, un vector en el que se guarda la respuesta, la longitud del mensaje que se codificó y el número de bits por byte usado.

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 Visual Studio). Nota importante: los programas se calificarán únicamente usando el ambiente de visual; si el programa no compila en este ambiente, se considerará que no corre (así compile 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 23 de abril a las 23:55.**

E. CASOS DE PRUEBA

Para hacer las pruebas, adjunto al proyecto, se encuentran tres carpetas, cada una con una imagen original (.bmp), el texto a insertar o leer (.txt) y la imagen modificada (.bmp). Al insertar en la imagen original se debe obtener la imagen modificada. Al extraer de la imagen modificada se debe obtener el mensaje. Realice cada una de las pruebas con los siguientes parámetros:

1. Prueba 1 (12 caracteres, bits por Byte: 8)

   Nota: este caso no tiene sentido puesto que reemplaza todo el byte de cada componente (se pierde la imagen); se pone por completitud, para probar el programa en un caso extremo fácil de verificar (puesto que en el vector de información queda literalmente la cadena de caracteres).

2. Prueba 2 (795 caracteres, bits por Byte: 4)
3. Prueba 3 (2130 caracteres, bits por Byte: 2)

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 una 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.