

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 12 de mayo a las 23:50

Si tienen alguna duda sobre el enunciado o el desarrollo del proyecto pueden comunicarse con los profesores del curso o los monitores:

- ✓ Andrés Hernández – af.hernandezl@uniandes.edu.co
- ✓ Laura Prieto – lv.prieto@uniandes.edu.co
- ✓ Santiago Salazar – s.salazara@uniandes.edu.co

A. OBJETIVOS

- Practicar con el lenguaje ensamblador.
- Practicar con programación mixta C-ensamblador.
- 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 del portador, para que pueda pasar desapercibida"¹.

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 humana 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.

De manera que, se quiere escribir un programa que permita ocultar 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 guardar las imágenes es bmp de 24 bits. Esto quiere decir que para cada uno de los componentes RGB que forman 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.

¹ Obtenido de: <http://es.wikipedia.org/wiki/Esteganografía>

Estructuras de datos

La imagen se representará por medio de la siguiente estructura:

```
typedef struct img
{
    int ancho;
    int alto;
    unsigned char * información;
} 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 información consiste de 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 (serían 3 bits por byte). 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
10011011	10110000	11011010	10110110	00011110	01010100
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 ("esqueleto (ASM) .c"), encuentra el esquema del programa. El programa se invoca por línea de comando con una diferencia al proyecto 1: en este caso, se deberán indicar los parámetros para el funcionamiento del programa en el momento que se llama la ejecución del archivo, **no mientras que se está ejecutando** (el programa no lee nada de la consola; recibe todo como parámetro por la línea de comando).

Los parámetros necesarios para la ejecución del programa son los siguientes:

1. La ruta completa del archivo.bmp
2. Indicar si se está ejecutando el programa para insertar (se utiliza el argumento 'w') o leer un mensaje (se utiliza el argumento 'r').
3. El número de bits por Byte que se utilizaron o que se utilizarán para guardar el mensaje, según corresponda.
4. El mensaje que será guardado en la imagen o la longitud del mensaje a ser leído.

A continuación se ejemplifica el uso de los parámetros necesarios para correr el programa:

- `/path/to/image.bmp w 4 thisIsATestMessage`
- `/path/to/image.bmp w 2 "this test message has spaces between words"`
- `/path/to/image.bmp r 5 18`

Note que los parámetros están separados por un espacio (' '). Cualquier parámetro que tenga espacios debe ir entre comillas dobles (" ") para que se reconozca como uno solo.

El programa ya tiene definidos: el procedimiento `main(int argc, char* argv[])`, un procedimiento para leer una imagen `cargarBMP24()` y uno para escribir una imagen `guardarBMP24()`. Deben usar estos procedimientos en el programa. **Estos procedimientos no deben ser modificados.** Adicionalmente, una vez que se guarda la imagen modificada, esta queda con el nombre de la imagen original más `"_modificado.bmp"` con el fin de que no se sobrescriba la imagen original. Es importante resaltar que esto **ya lo realiza el programa**, los estudiantes no deben encargarse de la modificación del nombre.

De manera que, se deben completar 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 estructura de 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. Para el desarrollo de este método **NO está permitido el uso de nombres simbólicos.**

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 estructura de 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 el desarrollo de este método **está permitido el uso de nombres simbólicos.**

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

Finalmente, se propone el desarrollo de un método **opcional** `sacarNbits(unsigned char secuencia[], int bitpos, int n)`. La descripción de su funcionamiento esta adjunta en la documentación del esqueleto. Para el desarrollo de este método **está permitido el uso de nombres simbólicos**.

El desarrollo de estos procedimientos debe realizarse haciendo uso del lenguaje ensamblador.

C. ESPECIFICACIONES

- El programa se deben escribir en Ensamblador (en Visual Studio). Adicionalmente **se deben usar únicamente los registros que se ven en clase**, es decir, registros de **sistemas de 32 bits**. No se permite el uso de registros de sistemas de 64 bits como por ejemplo `rax`, `rbx`, etc.

Nota importante: Los programas se calificarán únicamente usando el ambiente de Visual Studio (Compilador de Visual Studio 2017 de C). 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** coherentes explicando el código de manera que sea comprensible para el evaluador.
- 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 `.docx` o `.pdf` explicando por qué cree que no funciona o qué fue lo que hizo.
- El trabajo se realiza en grupos de **máximo 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 en Sicua+ por **un solo** integrante.
- **Se debe entregar por Sicua+ a más tardar el 12 de mayo a las 23:50**

E. CASOS DE PRUEBA

Para hacer las pruebas, adjunto al proyecto se encuentra un archivo comprimido que contiene tres carpetas. Cada una de estas con una imagen original (`.bmp`), el texto a insertar o leer (`.txt`) y la imagen modificada (`.bmp`). Al insertar el mensaje dado en la imagen original se debe obtener la imagen modificada. Al extraer el mensaje de la imagen modificada se debe obtener el mensaje original. Realice cada una de las pruebas con los

siguientes parámetros:

1. Prueba 1 (138 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 evalúa este caso 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 (898 caracteres, bits por Byte: 5)

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 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:
 - o 10% - legibilidad (nombres dicientes para variables, comentarios e indentación)
 - o 20% - manejo de bits (uso de los operadores de bits de Ensamblador: SHR, AND, etc.)
 - o 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 y serán tratados según el Reglamento General de Estudiantes de Pregrado.
- Este proyecto se basa enteramente en el primer proyecto del curso, por ende es recomendable que si se tenían errores en el primer proyecto se corrijan oportunamente.
- No está permitido utilizar programas que realicen la conversión del proyecto anterior a lenguaje Ensamblador. La finalidad del proyecto es que los estudiantes practiquen los conocimientos adquiridos en clase. Estos casos serán tratados como fraude académico según el Reglamento General de Estudiantes de Pregrado.