Ejercicio obligatorio 4

Fecha de entrega: Domingo 21 de noviembre

Introducción

El formato de Windows Bitmap (BMP) es un formato binario con muchas limitaciones pero sencillo de implementar y entender y un formato ampliamente difundido.

Si bien el mismo tiene muchos modos de operación, vamos a centrarnos en la versión de 24 bits de color, no comprimida, sin paleta, etc. para simplificar sus particularidades.

Un archivo BMP consta de tres secciones:

Sección	Tamaño
Encabezado de archivo	14
Encabezado de imagen	40
Pixeles	--

Cada una de estas secciones tiene un formato específico.

Si bien la mayor parte de los campos del archivo son de un solo byte, hay números almacenados en más de un byte y en este caso esos números son números signados almacenados según la convención little-endian. Es decir, si un campo numérico contuviera la secuencia `{0xAA, 0xBB, 0xCC, 0xDD}` el número representado sería el `0xDDCCBBAA`.

Encabezado de archivo

El encabezado de archivo consiste en la siguiente secuencia:

Campo	Tipo	Valor
Tipo	`char[2]`	`"BM"`
Tamaño	`int32_t`	El tamaño en bytes del archivo
Reservado	`int16_t`	`0`
Reservado	`int16_t`	`0`
Offset	`int32_t`	`54`

Como se ve, el único valor no definido del formato es el tamaño en bytes del archivo. El parámetro del offset indica en qué posición comienza la tabla de píxeles en el archivo, pero siendo que estamos usando un formato simplificado donde el encabezado de imagen siempre mide 40 bytes entonces este valor queda fijo.

El tamaño en bytes del archivo puede calcularse previo a la escritura en base al ancho y alto de la imagen, lo abordaremos más adelante.

Encabezado de imagen

El encabezado de imagen consiste en la siguiente secuencia:

Campo	Tipo	Valor
Tamaño	`int32_t`	`40`
Ancho	`int32_t`	El ancho de la imagen
Alto	`int32_t`	El alto de la imagen
Planos	`int16_t`	`1`
Bits de color	`int16_t`	`24`
Compresión	`int32_t`	`0`
Tamaño de imagen	`int32_t`	`0`
Resolución X	`int32_t`	`0`
Resolución Y	`int32_t`	`0`
Tablas de color	`int32_t`	`0`
Colores importantes	`int32_t`	`0`

Como se ve, con todas las simiplificaciones adoptadas, los únicos dos parámetros variables son el ancho y el alto de la imagen. Así como generaremos imágenes no comprimidas, en 24 bits de color y sin paleta de colores al leer un archivo es importante validar que tanto los bits de color, como la compresión como las tablas de color estén en los valores indicados; si no, no sabemos cómo procesar ese archivo.

Cabe remarcar que la altura podría llegar a ser un valor negativo.

Pixeles

En la sección de los píxeles todos los valores ocupan un byte, es decir, pueden considerarse de tipo `uint8_t`. Como estamos usando RGB24 cada pixel se representará con 3 bytes consecutivos.

Ahora bien, como el formato es little-endian, si se leyeran los bytes del RGB de a un byte por vez los mismos estarán en orden BGR, es decir, en el orden inverso. Tanto al leer como al escribir lo haremos, por simplicidad, de a un byte por vez y el orden será azul, verde, rojo.

Los pixeles se almacenan según *scan lines* donde cada scan line es una fila de la imagen recorrida de izquierda a derecha. Ahora bien, cada una de las scan lines tiene que medir un múltiplo de 4 bytes por lo que se completarán con cero los bytes restantes hasta completar los 4.

Por ejemplo, si una imagen tuviera 13 pixeles de ancho, estos 13 pixeles se pueden representar en 39 bytes, pero como eso no es múltiplo de 4 al final de los pixeles se agregará un byte adicional en cero para completar 40 bytes.

Notar que el largo de la scan line puede computarse mirando cuánto falta para que sea múltiplo de 4 la operación de multiplicar por 3 el ancho de la imagen.

La sección de píxeles consistirá en una secuencia de tantas scan lines como altura tenga la imagen. Si el parámetro de altura se hubiera indicado como un número positivo las scan lines están ordenadas de abajo hacia arriba, en cambio si fuera negativo estarán ordenadas de arriba hacia abajo (esta última es la convención que veníamos usando en PPM).

Tamaño del archivo

Ahora conociendo la especificación completa podemos ver que el tamaño del archivo estará dado por el tamaño de los encabezados, que son 14 para el encabezado de archivo y 40 para el encabezado de imagen, y que la sección de píxeles será el tamaño de la scan line multiplicado por el alto de la imagen.

Entonces, el tamaño será: 14 + 40 + alto * scanline. Donde la scanline será ancho * 3 más lo que haga falta para que ese número sea múltiplo de 4.

Trabajo

`imagen_t`

Se provee ya implementado un TDA `imagen_t` el cual sirve para almacenar los pixeles de una imagen, cada pixel es de tipo `color_t`.

Este TDA provee las siguientes primitivas:

- `imagen_t *imagen_crear(size_t ancho, size_t alto);`
- `void imagen_destruir(imagen_t *im);`
- `void imagen_dimensiones(const imagen_t *im, size_t *ancho, size_t *alto);`
- `bool imagen_set_pixel(imagen_t *im, size_t x, size_t y, color_t color);`
- `color_t imagen_get_pixel(const imagen_t *im, size_t x, size_t y);`

las cuales son suficientes para crear y destruir una imagen, consultar sus dimensiones, almacenar y recuperar pixeles.

`computar_intensidad()`

Se provee ya implementada una función `computar_intensidad()` para no depender de la implementada por el alumno en el EJ3 la cual funciona con la misma lógica de las ya implementadas. (Se provee además un `main()` para mostrar su uso.)

Escritura de PPM

Se pide implementar una función `void escribir_PPM(const imagen_t *imagen, FILE *f);`, la misma recibirá un descriptor de archivo `f` abierto en modo texto (que podrá eventualmente ser `stdout`) y escribirá la `imagen` en dicho archivo en formato PPM.

Esta función no es una primitiva del TDA `imagen_t` por lo que tendrá que acceder a dicha imagen a través de la interfaz de primitivas del tipo.

Endianness

Escribir una función `void escribir_int16_little_endian(FILE *f, int16_t v);` que reciba un archivo `f` y un entero `v` de 16 bits y lo escriba en el archivo en formato little-endian.

Escribir una función `void escribir_int32_little_endian(FILE *f, int32_t v);` similar a la anterior pero que escriba un entero de 32 bits.

> **Nota**
>
> No puede asumirse que la plataforma es big/little-endian. Las funciones deben operar con los bytes a bajo nivel y ser portables a cualquier arquitectura.

Escritura de BMP

Escribir una función `void escribir_BMP(const imagen_t *imagen, FILE *f);` que reciba un archivo `f` abierto en modo binario y escriba en el mismo la `imagen` en formato BMP.

Aplicación

Se pide implementar una aplicación que se ejecute como:

```
$ ./20212_ej4 [ancho] [alto] [nombrearchivo]
```

que genere en el archivo de nombre `nombrearchivo` una imagen de `ancho` x `alto`.

Dependiendo de si el nombre de archivo termine en `.ppm` o en `.bmp` deberá generarse la imagen en formato PPM o BMP respectivamente.

Material

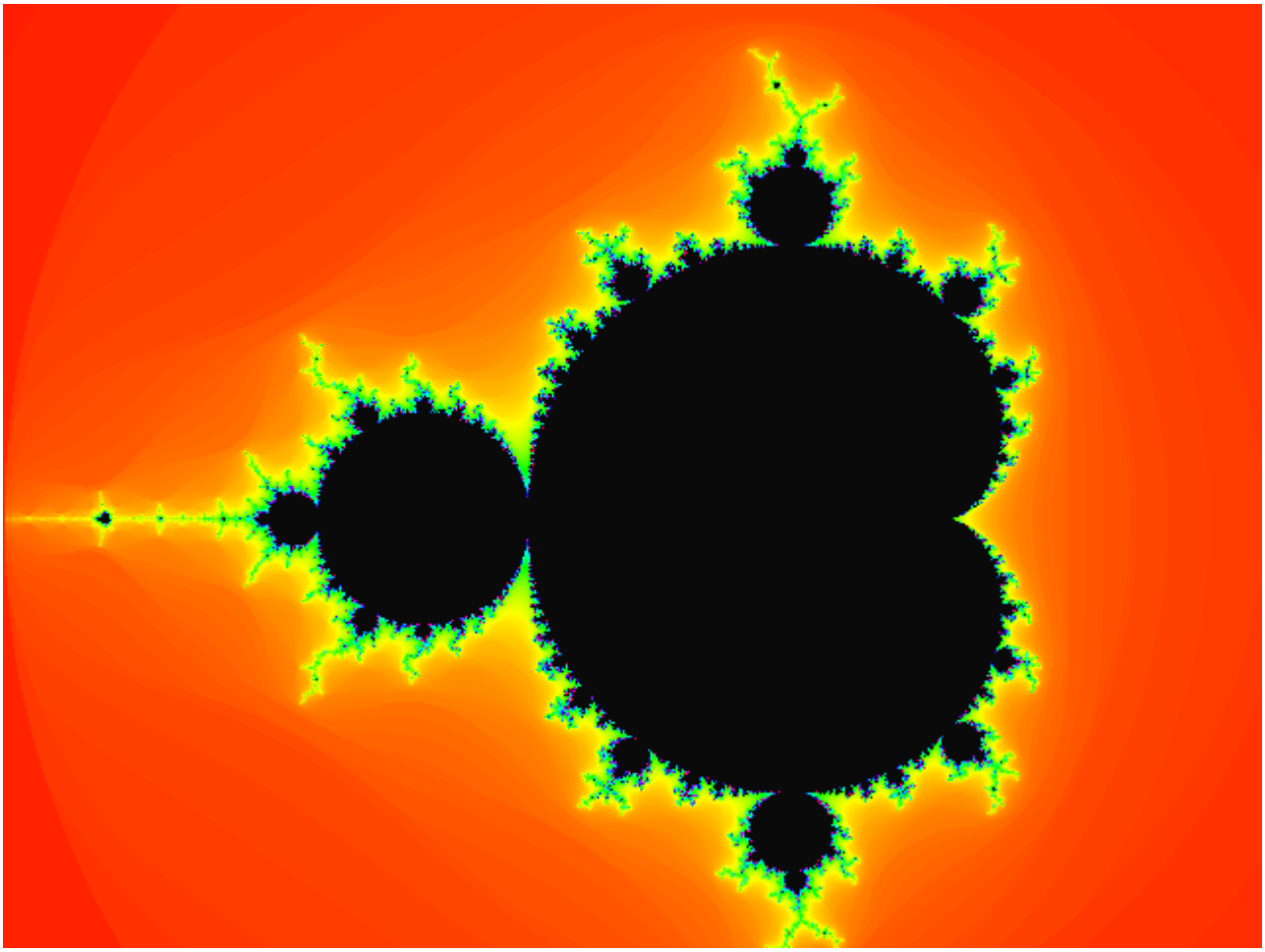
El TDA imagen y la función computar intensidad se descargan de acá:
archivos_20212_ej4.tar.gz

Validación

Para la ejecución:

```
$ ./20212_ej4 640 480 imagen.bmp
```

se genera una imagen como la siguiente



validar que la salida generada sea consistente con respecto a esta imagen de referencia.

Entrega

Deberá entregarse el código fuente del programa desarrollado.

El programa debe:

1. Compilar correctamente con los flags:

```
-Wall -Werror -std=c99 -pedantic
```

2. pasar Valgrind correctamente,
3. validar la imagen esperada.

Nota

Si bien no es obligatorio para esta entrega se permite que la misma se haga modularizada. En dicho caso deberá entregarse el archivo `Makefile` correspondiente.

La entrega se realiza a través del sistema de entregas.

El ejercicio es de entrega individual.