UNIVERSIDAD DE GRANADA

Metodología de la Programación

Grado de Ingeniería Informática

Proyecto Informático (MP, Grupo F).

DECSAI

1. Introducción

Una *imagen* es una representación visual de un objeto. En Informática, por imagen, se entiende un archivo codificado que, al abrirlo, muestra una representación visual de algo. El objetivo de

Figura 1: Archivo monigote.jpg

esta práctica consistirá en diseñar e implementar un tipo de dato abstracto (TDA) `imagen`, cuyos objetos se almacenarán en ficheros gráficos muy simples de extensión `.mgp`, y unas acciones básicas a realizar con los objetos de esta clase. Con esto se pretende:

- Introducir al alumno en la abstracción, uso de clases y el manejo de ficheros en C++.
- Que el alumno resuelva un problema en el que es necesaria la modularización, ya que la resolución de esta práctica obliga a la creación de diversos archivos, su compilación y enlace para obtener los ejecutables.
- Que el alumno sea capaz de organizar la compilación mediante un fichero `makefile`

2. Abstracción de una imagen

En esta sección prestaremos atención al TDA `imagen`. Para simplificar, consideraremos que las imágenes son en escala de grises, esto es, cada pixel es de un color gris cuya luminosidad variará entre blanco (la mayor luminosidad) y negro (la menor luminosidad). Matemáticamente, una imagen quedaría definida mediante una función:

$$\text{imagen} = f : \mathbb{R} \times \mathbb{R} \to [0, \infty)$$

Es decir, una imagen queda definida como una asignación de un color (una luminosidad, dada por un valor real) a cada punto del plano. Sin embargo, en un ordenador no podemos representar imágenes de tamaño infinito ni tampoco un subconjunto no discreto del plano. Además, sería recomendable que los posibles colores fueran un número finito. Por lo tanto la representación de la imagen deberá ser una función

$$\text{imagen} = f : \{0, 1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, 256\}$$

Así pues, una imagen digital de niveles de gris puede verse como una matriz bidimensional de puntos (píxeles, en este contexto) en la que cada uno tiene asociado un nivel de luminosidad

cuyos valores están en el conjunto $\{0, 1, ..., 255\}$ de forma que el 0 indica mínima luminosidad (negro) y el 255 la máxima luminosidad (blanco). Los restantes valores indican niveles intermedios de luminosidad (grises), siendo más oscuros cuanto menor sea su valor. Hemos variado la gama de colores entre 0 y 255 para que, con esta representación, cada píxel requiera únicamente un byte, es decir, está representado por un dato de tipo `unsigned char`.

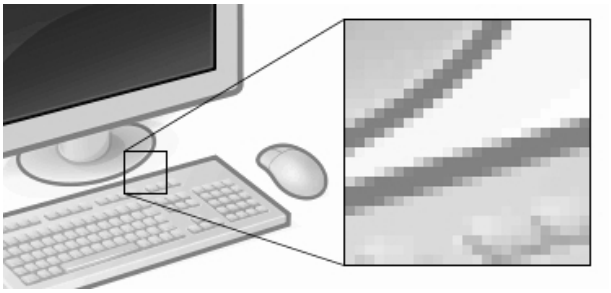


Figura 2: Imagen y ampliación para apreciar los píxeles, fuente: Wikipedia

Las imágenes en color se representan de forma diferente ya que un color no puede representarse usando un único valor, sino que se deben incluir tres números. Existen múltiples propuestas sobre el rango de valores y el significado de cada una de esas componentes. De las más utilizadas es el modelo RGB, donde cada color se representa como como la suma de las tres componentes de una terna: Rojo (en inglés, Red), Verde (en inglés Green) y Azul (en inglés, Blue). De igual manera que con las imágenes en escala de grises es habitual asignarle el rango de números enteros [0, 255], ya que permite representar cada componente con un único byte, y la variedad de posibles colores es suficientemente amplia.

3. Ficheros PGM

Para almacenar los TDA `imagen` utilizaremos ficheros con extensión `.pgm`. PGM es el acrónimo de *Portable Gray Map file format*. Una imagen PGM almacena una imagen de niveles de gris, es decir, un valor numérico que indica la luminosidad para cada píxel. Un valor de luminosidad viene determinado por un número entero en el rango [0,255]. Por lo tanto, un único byte será suficiente para almacenar el contenido de un píxel. Los ficheros PGM están compuestos de una cabecera y una secuencia binaria de bytes que representan la matriz pixels de la imagen que contiene. Concretamente, la cabecera tiene el siguiente formato (en formato ASCII):

- Una cadena mágica (cadena de caracteres) que, para archivos `.pgm`, es `P5`.
- Un separador (un salto de línea).
- Un número indeterminado de comentarios. Los comentarios son de línea completa y están precedidos por el carácter `#`. La longitud máxima de cada comentario es de 70 caracteres. Obviamente, existe la posibilidad de que no haya ningún comentario.
- El ancho o columnas de la imagen (decimal y formato ASCII), es decir, el número de píxeles de una fila.
- Un separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un espacio en blanco.
- El alto o filas de la imagen (decimal y formato ASCII). Es decir, el número de píxeles de una columna.

- Un separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un salto de linea.
- El valor máximo de gris (decimal y formato ASCII). Indica que el rango de valores para cada valor de luminosidad es [0,M], 0 correspondiente a negro y M a blanco. Para ésta práctica $M = 255$ siempre.
- Un único separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un salto de linea.

A continuación está escrito el contenido de la imagen, es decir, una secuencia binaria de `filas x columnas` valores de luminosidad. Cada uno de estos valores representa el valor de luminosidad de un pixel. El primero corresponde al pixel de la esquina superior izquierda, el segundo al de su derecha, etc... (el último al de la esquina inferior derecha).

```
P5
#Imagen de muestra
#Autor: Pepito Grillo
#Fecha: 13/03/2009
#version: 1.5
10 10
255
(secuencia binaria de 100 valores)
```

3.1. Ficheros PGM de texto

Existe una versión más sencilla de los ficheros PGM, en la que la secuencia de pixeles está escrita en ASCII. Es decir, el archivo es un fichero de texto y la matriz de píxeles viene dada por una matriz de valores enteros (cada fila de la matriz en una línea, cada pixel de una fila separado del siguiente pixel por un espacio en blanco). Estos ficheros tienen la misma estructura, excepto que la cadena mágica es P2.

```
P2
#Imagen de muestra
10 10
255
0 0 0 12 13 12 12 15 15 256
0 0 12 12 13 12 12 15 256 256
 ....
```

Puesto que el tema de entradas/salidas y ficheros en C++ se explicará más adelante, se trabajará mientras tanto con la redirección de la entrada/salida, y utilizando estos ficheros de texto PGM.

En ambos casos, para leer este tipo de archivos podemos utilizar los editores de imágenes *GIMP* (Linux) o *Photoshop* (Windows). Una aplicación gratuita para Windows es *Brennig's*.

4. Implementaciones a realizar en la práctica

4.1. Clase imagen

En base a lo explicado en las secciones anteriores, la clase `imagen`, que se declarará en un fichero `imagen.h` e implementará en un fichero `imagen.cpp`, debería constar, al menos, de una matriz de pixels, donde un píxel es un dato de tipo `unsigned char`. Para abreviar, se recomienda nombrar los datos `unsigned char` mediante la palabra `color`, utilizando la sentencia

```
typedef unsigned char color;
```

En general, debería contar con tantos datos miembro como datos tiene un fichero PGM: anchura, altura, número de comentarios, los comentarios, valor máximo de gris y la matriz de pixeles.

Además, esta clase debe contener los siguientes métodos miembro:

- Un constructor por defecto. Crea una imagen nula e inútil. Los atributos se inicializaran a cero.
- Un constructor con parámetros.
- Un constructor de copia.
- El destructor de la clase.
- Sobrecarga del operador =.
- Por último, otras funciones miembro que el alumno considere necesarias para desarrollar la práctica.

Se valorará el análisis de que algunos elementos de dicha clase deban ser modificados con `static`, `const`, `friend` o `inline`.

4.2. Almacenamiento/lectura en/de ficheros PGM

Para leer y guardar ficheros PGM se necesitará implementar las siguientes funciones:

4.3. Función de escritura

La función `salvar_pgm` debe construir o sobreescribir un fichero PGM con los datos que se dan como argumentos. El prototipo debe ser:

```
bool salvar_pgm(char *name, unsigned char** m, int c, int f,
           char** com, unsigned short num_com);
```

donde:

- `name` es el nombre del fichero PGM donde queremos guardar los datos (extensión `pgm`). Si no existiera, debería crearlo. Y si existe, debería sobreescribirlo.
- `m` es un puntero a puntero a `unsigned char` que contiene la dirección de memoria de la matriz de pixels.
- `c` es el número de columnas de `m`.
- `f` es el número de filas de `m`.
- `com` es un vector de cadenas de caracteres, donde cada componente es un comentario.
- `num_com` es el número de comentarios.

Esta función debe devolver `true` si la escritura se produce correctamente. Y `false` si hay algún error. Además, en este caso, debe mostrar un mensaje informativo sobre el error. Si se prefiere, se puede utilizar la clase `string` en vez de cadenas de caracteres.

4.4. Función de lectura

La función `leer_pgm` debe leer los datos de un fichero PGM y guardarlos en las variables que se pasan como argumentos. El prototipo debe ser:

```
bool leer_pgm(char *name, unsigned char** &m, int &c,
          int &f, char** &com, unsigned short &num_com);
```

donde:

- `name` es el nombre del fichero PGM de donde leemos los datos (extensión `.pgm`).
- `m` es donde se guarda la sucesión binaria de valores de luminosidad de la imagen contenida en el fichero.
- `c` es donde se guarda el ancho de la imagen.
- `f` es donde se guarda el alto de la imagen.
- `com` es donde se guardan los comentarios.
- `num_com` es donde se guarda el número de comentarios.

Esta función debe devolver `true` si la lectura se produce correctamente. Y `false` si hay algún error. Además, en este caso, debe mostrar un mensaje informativo sobre el error. Si se prefiere, se puede utilizar la clase `string` en vez de cadenas de caracteres.

Las funciones de lectura y escritura de ficheros PGM podrán implementarse con la matriz de pixeles como números en formato texto. Es decir, considerando los ficheros PGM como ficheros de texto. Sin embargo, no podrá obtenerse la calificación máxima.

4.5. Función para cambiar comentarios (para subir nota)

Implementar un programa `camb_coment`, que reciba argumentos desde la linea de comandos, y que sobreescriba los comentarios dados como argumentos en los comentarios de una función PGM. Por ejemplo, la orden

`camb_coment prueba.pgm _imagen de muestra _autor: Pepito Grillo`

elimina los comentarios de `prueba.pgm` y coloca los dos que se dan como argumentos. Para simplificar la implementación, se puede suponer que cada comentario se limita a una única palabra (es decir, no tiene espacios en blanco). Por ejemplo, la orden anterior sería

`camb_coment prueba.pgm imagen_de_muestra autor:_Pepito_Grillo`

Las funciones `contraste`, `rellenar` y `collage` deben seguir funcionando sin haber realizado ninguna modificación.

Nota: En Linux no podemos dar un argumento empezando por `#` ya que entonces lo ignorará.

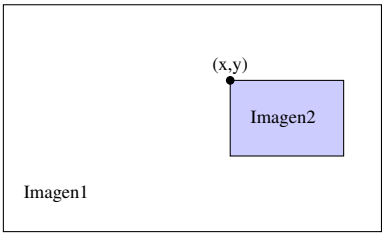
4.6. Acciones con ficheros PGM

Se deben implementar tres programas, **que se deben ejecutar desde la linea de comandos**, que realicen las siguientes funciones:

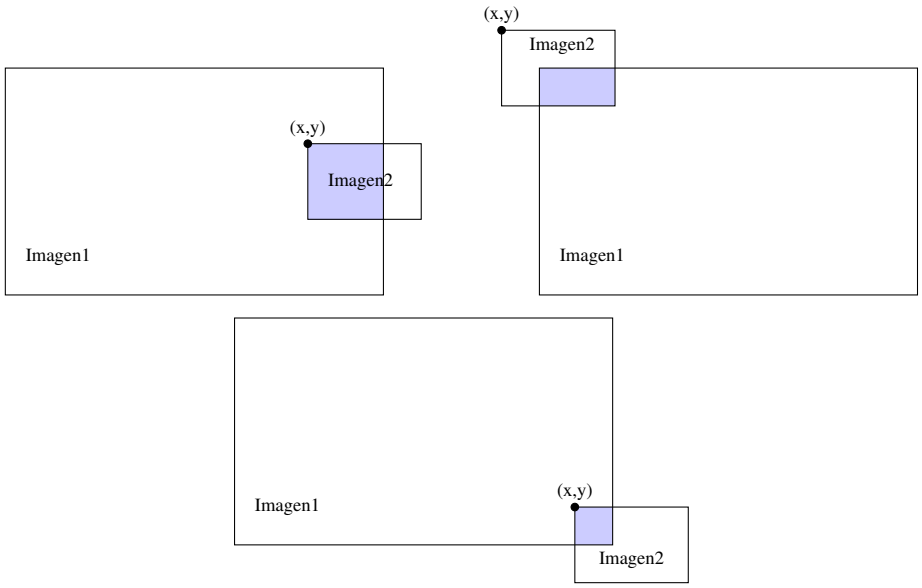
- **Rellenar una imagen de un color concreto** (`rellenar`). Esta función no requiere más explicación, simplemente debe dar a cada pixel del objeto un color concreto. Por ejemplo, la orden
  `rellenar ima.pgm 150`
  da el color `150` a todos los pixels de la imagen guardada en el fichero `ima.pgm`. El resultado sería el que se refleja en la Figura 3.

Figura 3: Fichero ima.pgm

- **Plasmar una imagen en otra haciendo un collage** (`collage`). Esta función consiste en superponer una imagen encima de otra a partir de ciertas coordenadas. Por ejemplo,



Entonces, la `Imagen1` varía en esa región. Hay que prestar atención a la posibilidad de que `Imagen2` se coloque de tal manera que tan sólo una parte se solape con `Imagen1`. Por ejemplo, podrían darse los siguientes casos:



El programa se podría ejecutar, por ejemplo, con la siguiente orden
`collage ima1.pgm ima2.pgm 50 50`
provocando que la imagen en `ima1.pgm` se copie en la imagen en `ima2.pgm` a partir de la coordenada (50,50). Cuidado!!! las coordenadas podrían ser negativas.

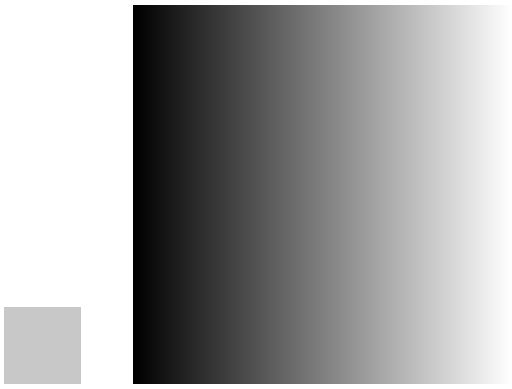


Figura 4: Ficheros ima.pgm (50x50) y ima2.pgm (256x256)

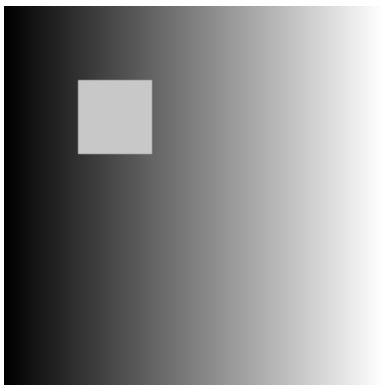


Figura 5: Ficheros ima2.pgm después de la orden collage

- **Cambiar el contraste de una imagen** (`contraste`). Esta función aumenta o disminuye la luminosidad de la imagen siguiendo para cada pixel la fórmula:

$$\text{nueva} = \min + \frac{(\max - \min)}{(b-a)}(\text{vieja} - a)$$

  donde:

  - `vieja` es el color original del pixel.
  - `[a,b]` es el intervalo de color de la imagen. Es decir, `a` es el mínimo de luminosidad de todos los píxeles de la imagen y `b` es el máximo.
  - `[min,max]` es el nuevo intervalo color de la imagen. Es decir, `min` será el mínimo de luminosidad para un pixel de la imagen (aunque siempre será al menos 0) y `max`, el máximo (aunque siempre será menos de 255).
  - `nueva` es el color que se aplica al pixel.

  Por ejemplo, esta orden se podría aplicar de la siguiente forma
  `contraste ima.pgm 255 0 ima2.pgm`
  cambiando la luminosidad máxima y mínima de la imagen en ima.pgm a 255 y 0, respectivamente, y guardando los cambios en `ima2.pgm`. Para que la fórmula funcione correctamente, la longitud del nuevo intervalo de luminosidad `[min,max]` debe ser mayor que la longitud del intervalo actual `[a,b]`. Es decir, ésta formula sirve solamente para aumentar el contraste.

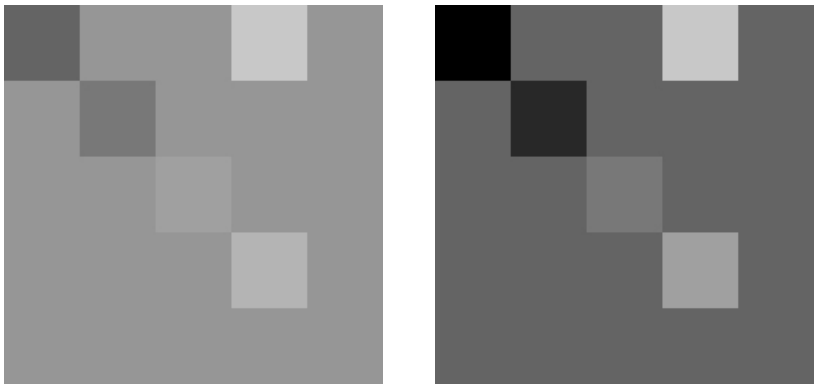


Figura 6: Ficheros ima.pgm (max:200, min:100) y ima2.pgm (max:255, min:0)

Importante: Estos comandos deben informar al usuario si se ha producido un error al realizar el proceso, o si algún parámetro no se ha introducido en el formato adecuado y desplegar un mensaje de ayuda. Para terminar el programa por alguno de éstos motivos se puede utilizar la función `exit`.

5. Práctica

La práctica consistirá en implementar los elementos explicados en la sección anterior. Dividir el código en diferentes ficheros tal y como se ha explicado en las clases de prácticas de compilación separada y realizar un fichero `makefile` que compile todo el proyecto. Es obligatorio seguir los siguientes puntos:

- Construir tantos archivos de cabecera como se considere necesario.
- Los archivos de cabecera deben estar perfectamente documentados para ser interpretado con el programa Doxygen.
- Los archivos `.cpp` también deben estar documentados mediante comentarios que expliquen las operaciones que va realizando cada función.
- Construir, al menos, una biblioteca con las funciones para manipulación de ficheros PGM. Se pueden crear otras bibliotecas que el alumno considere oportuno. Las funciones anteriores deben enlazarse (para construir los ejecutables) a través de dichas bibliotecas.
- Todos los pasos de la compilación y enlazamiento deberán estar contenidos en un fichero `makefile` tal que, al ejecutarlo con el programa `make`, en cada paso muestre por pantalla el proceso que está realizando, aunque no debe mostrar la linea de comando que realiza esa acción. Por ejemplo, si está generando el fichero objeto del archivo `principal.cpp`, debe mostrar por pantalla
  `Generando el fichero objeto de principal.cpp`,
  pero no debe mostrar por pantalla
  `g++ -c principal.o principal.cpp`.
- El fichero `makefile` debe contener la orden de que, al finalizar todo el proceso de compilación, borre los ficheros `.o` correspondientes a las funciones que se encuentran en las bibliotecas (únicamente esos).
- Para unificar la disposición de los archivos, seguiremos el siguiente árbol de carpetas:

```
ProyectoInformatico/
      |
      |-------- bin/
      |-------- src/
      |-------- include/
                  |
                  |------- operar_pgm.h
      |-------- obj/
      |-------- lib/
                  |
                  |------- liboper_pgm.a
      |-------- doc/
      |
      |-------- makefile
```

donde,

- En la carpeta `ProyectoInformatico` estará el archivo `makefile` que se ha realizado para la práctica.
- En la carpeta `bin` se generarán los ficheros ejecutables mediante el programa `g++`.
- En la carpeta `src` se guardarán los ficheros de código `.cpp` que se realicen para la práctica.
- En la carpeta `include` se guardarán los ficheros de cabecera `.h` que se realicen para la práctica.
- En la carpeta `obj` se generarán los ficheros objeto `.o` mediante el programa `g++`
- En la carpeta `lib` se generarán las librerías `.a` mediante el programa `ar`
- En la carpeta `doc` se guardará la memoria de la práctica (archivo word, pdf, ...).

El objetivo es que al ejecutar el programa `make` en la carpeta `ProyectoInformatico`, el proyecto debe compilarse generando los ficheros en su carpeta correspondiente.

- Para entregar el código se debe empaquetar en un fichero `.zip` el árbol de directorios y subirlo a la plataforma de DECSAI a través del acceso identificado. La fecha límite de entrega electrónica de la práctica es el 12 de Junio de 2018.