Metodología de la Programación.
Grado en Ingeniería Informática.
Grupo A

LA CLASE IMAGEN

Francisco J. Cortijo Bon
cb@decsai.ugr.es

Curso 2019-2020

Índice

1. Introducción

El proyecto que se describe en este documento tiene como objetivo la construcción de la clase `Imagen` y realizar diferentes programas que gestionen objetos de esa clase.

De manera complementaria también se trabajará con ficheros PGM (P2/P5) para leer/guardar datos de tipo imagen.

El proyecto deberá modularizarse en ficheros `.h` y `.cpp` y deberá documentarse.

2. ¿Qué es una imagen digital?

Una imagen digital es el resultado de muestrear una escena y cuantizar una escena (estática), midiendo valores de intensidad de la luz a intervalos regulares horizontal y verticalmente. El resultado es una *matriz* en el que cada casilla -pixel- almacena un valor de luminosidad.

Hay muchos temas relacionados que no vamos a abordar como el **color** (representación del color, imágenes en color) o el **vídeo**, que estudiarán en asignaturas de cursos superiores. En este trabajo seguiremos con imágenes de niveles de gris.

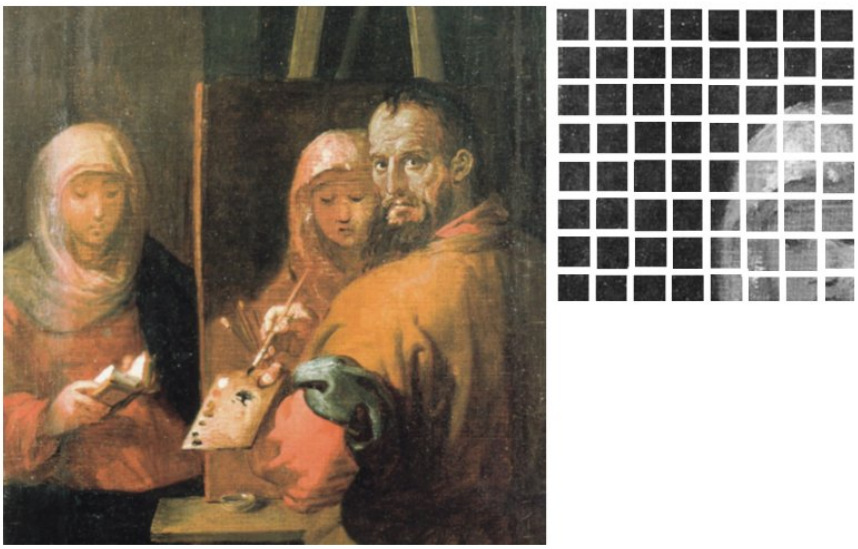

Figura 1: Izquierda: Obtención de una imagen 2D a partir de una escena (Francisco Ribalta, San Lucas evangelista (fragmento), óleo sobre lienzo, 83 x 36 cm. Museo de Bellas Artes de Valencia). Derecha: Una parte de la imagen (en niveles de gris, cada cuadrado es un pixel).

Para manipular una imagen debe residir en la memoria. Una imagen es un ente abstracto que tiene propiedades (por ejemplo, sus dimensiones) y con el que puede operarse (por ejemplo, redimensionarse). Así, la clase `Imagen` servirá para dar soporte a este tipo de entidades en programas que las manipulen.

En este trabajo deberá proponer una representación para los objetos `Imagen` y los métodos que permitan gestionarlos. La lista de métodos solicitados debe entenderse como una lista mínima y orientativa. Si necesitara implementar algún método adicional (público o privado), hágalo.

3. Representación de la clase imagen

Una representación de un objeto `Imagen` debe permitir un procesamiento rápido de su contenido. Es poco frecuente tener que acceder a unos pocos píxeles, siendo las tareas más frecuentes las que necesitan procesar amplias regiones de la imagen

(o la imagen completa).

Las representaciones típicas de una imagen son las que emplean *vectores* o *matrices*. Para aprovechar al máximo la memoria es imprescindible usar memoria dinámica para los píxeles de la imagen. Así, vectores o matrices dinámicos son buenas estructuras de datos candidatas para los píxeles de una imagen.

Analice las propiedades de una imagen y las precondiciones que deberán verificar para declarar los datos privados de los objetos `Imagen`.

4. Comentarios

Para aumentar la legibilidad y trazabilidad se usarán *comentarios* (al estilo de los de los ficheros PGM). Cada operación relevante añade un comentario a la imagen. Esta nota implica que una imagen tiene un dato privado de la clase `Comentario`.

Los comentarios deben gestionarse usando un objeto de una nueva clase, la clase `Comentarios`. Proponemos usar un *array* de datos `string` para gestionar los comentarios.

```
class Comentarios{
   private:
   // PRE: 0 <= num_comentarios
   int num_comentarios;

   string * los_comentarios;
...
};
```

Los método (triviales) para esta clase que debe implementar son:

```
Comentarios (const Comentarios & otro);
```

Constructor de copia.

```
int GetNumComentarios (void) const;
```

Devuelve el número de comentarios guardados.

```
Comentarios & operator = (const Comentarios & otro);
```

Operador de asignación.

```
Comentarios & operator += (string c);
```

Añade un comentario.

```
friend istream & operator >> (istream & in,  Comentarios & c);
```

Operador de extracción de flujo.

```
friend ostream & operator << (ostream & out, const Comentarios & c);
```

Operador de inserción en flujo.

5. Constructores y destructor de la clase imagen

En primer lugar, deberá definirse qué es una imagen *vacía* y una imagen válida. Una imagen vacía es una imagen que tiene 0 filas y 0 columnas. Cualquier otra imagen válida deberá tener un número de filas y columnas estrictamente positivo.

Proporcionará un conjunto de constructores que permita crear imágenes válidas:

- Es interesante disponer de un constructor para crear imágenes *vacías*. En este caso, por ejemplo, se añade el comentario `Creada vacia`.
- Evidentemente, deberá proporcionar un constructor de copia. En este caso se añade el comentario `Creada por copia`.
- También se requiere de un constructor para crear una imagen rectangular que inicializa los píxeles de la imagen a un valor común (0 por defecto). En este caso se añade el comentario `Creada` *f* `x` *c* `a valor` *v* (donde *f* es el número de filas, *c* es el número de columnas y *v* es el valor común).
- Finalmente se necesita un constructor que cree una imagen a partir del contenido de un fichero PGM (puede ser P2 ó P5). En este caso se añade el comentario `Creada desde` *fichero* (donde *fichero* es el nombre del fichero PGM)

El destructor de la clase deberá liberar toda la memoria ocupada por el objeto.

6. Métodos y operadores de E/S

El contenido de un objeto imagen puede guardarse en un fichero PGM en cualquier momento. Escriba método(s) para crear un fichero PGM (P5 ó P2) con el contenido de un objeto `Imagen`. El contenido del objeto no se modifica, por supuesto.

Un objeto `Imagen` puede "reutilizarse" y su contenido puede ser sustituido por el de otra imagen guardada en un fichero PGM (P5 ó P2). Escriba un método para esta tarea. En este caso se añade el comentario `Reiniciada desde` *fichero* (donde *fichero* es el nombre del fichero PGM)

Decida la conveniencia de disponer de los operadores << y >> para la clase `Imagen`. Añádalos a la clase si lo estima oportuno.

7. Operador de asignación

Implemente el operador de asignación entre dos objetos de la clase `Imagen`. En este caso se añade el comentario `Reiniciada desde otro objeto de la clase`.

Añada una versión de operador para conseguir una imagen *plana* (todos los valores toman el mismo valor). En este caso se añade el comentario `Reiniciada al valor` *v* (donde *v* es el valor asignado a todos los píxeles)

8. Métodos y/o operadores de acceso y de consulta

Proporcione un operador de acceso que permita leer/modificar el valor de un pixel de la imagen, dado su fila y columna (filas y columnas se numeran desde 1).

Proporcione métodos para consultar las dimensiones de la imagen.

Proporcione métodos para consultar el número de comentarios y para obtener un comentario dado un índice (los comentarios también se numeran desde 1).

9. Sobrecarga de operadores

Sobrecargue el operador unario **!** para la clase `Imagen`. Calcula el negativo de una imagen. Añade el comentario: `Negativo`.

En la figura 2 mostramos un ejemplo del operador `!`

Figura 2: Izquierda: imagen original. Derecha: Negativo

Sobrecargue el operador binario * para la clase `Imagen` usando un método. Se aplica sobre dos objetos de la clase `Imagen` y devuelve un *nuevo* objeto de la clase `Imagen`. Ninguno de los operandos se modifica.

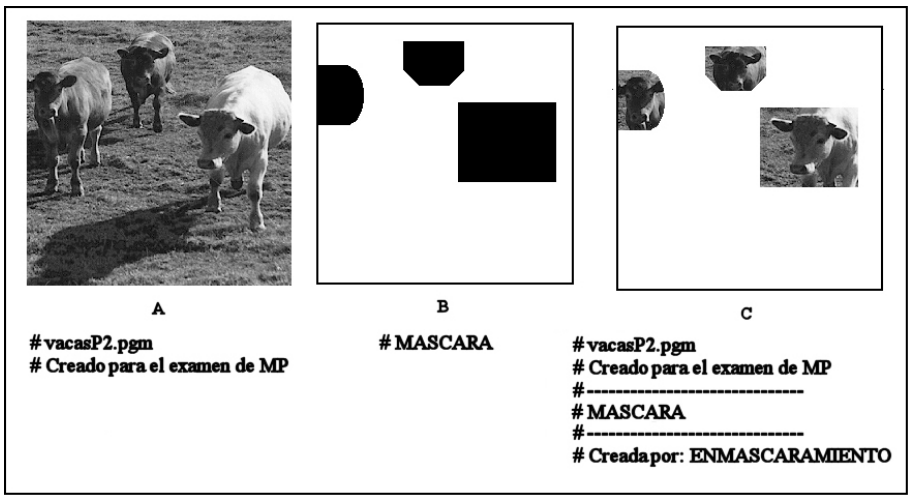

Figura 3: A) Imagen original. B) Una máscara. C) Resultado de enmascarar (operador *) A con B

Un píxel cualquiera de la imagen resultante tendrá el valor de uno de los dos operandos: i) si ese píxel contiene el valor 0 en el segundo operando el resultado será el valor del primer operando. ii) si no, el resultado será el valor del segundo.

En cuanto a los comentarios, el resultado tendrá los comentarios de la primera imagen, un comentario formado por guiones (una línea formada únicamente por guiones), los comentarios de la segunda imagen, un comentario formado por guiones y finalmente `Creada por: ENMASCARAMIENTO`

Muy importante: Las dos imágenes deben tener el mismo tamaño. Si no fuera así devolverá una imagen vacía

En la figura 3 mostramos un ejemplo del operador `*`. Observe cómo se va actualizando la lista de comentarios.

10. Aplicación 1: estimación de las dimensiones de una imagen

Hemos digitalizado unas fotografías antiguas pero a la hora de guardarlas no hemos indicado el formato de almacenamiento y se ha generado un fichero por cada imagen que contiene únicamente los valores de luminosidad (números enteros, en formato texto) separados por espacios. No es importante la existencia, o no, de saltos de línea que pudieran delimitar cada una de las filas de la imagen. Un ejemplo sería el siguiente:

```
 99   92   76   42  106  121  135  138  139  111  120  109
 85  145   93  129  107  121  145  128  143  158  128  115
122  121  118  133  160  118  123   89  134  124  137  139
...
```

Escriba un programa *completo* que transforma ficheros de valores de luminosidad en ficheros PGM.

El programa recibirá como argumento el nombre de un fichero de texto que contiene un número indeterminado de nombres de archivos (uno por línea). Se presupone que estos archivos contienen únicamente valores de luminosidad. El programa generará un fichero PGM por cada uno de ellos. Su nombre se formará añadiendo el sufijo **.pgm**. Si no pudiera abrirse algún fichero, el programa informará de esta circunstancia y seguirá su trabajo. Al finalizar indicará cuántos ficheros ha podido transformar.

Algoritmo

Dado un fichero con valores de luminosidad debemos **estimar** las dimensiones (filas y columnas) de la imagen.

Si disponemos de un fichero que contiene, por ejemplo, 1000 números enteros, considerando que:

- las imágenes deben ser rectangulares, y
- el número de filas (y columnas) debe ser mayor que uno

podemos encontrar muchas configuraciones posibles: 2×500, 500×2, 4×250, 250×4 ... Si somos capaces de asignar un valor numérico "sensato" a cada configuración podremos usar ese valor para seleccionar la "mejor". Este es el principio de los problemas de *optimización*.

El fundamento del método que proponemos es que **en una imagen real dos filas consecutivas serán muy parecidas**.

Si la configuración sobre la que estamos trabajando consta de F filas y C columnas, el valor numérico asignado a esa configuración, $V_{(F,C)}$, será la media de las desviaciones (al cuadrado) entre cada dos filas consecutivas:

$$V_{(F,C)} = \frac{1}{F-1} \sum_{f=1}^{F-1} \frac{1}{C} \sum_{c=1}^{C} [img(f,c) - img(f+1,c)]^2$$

Observe que la suma interior se calcula para las filas $1, 2, \ldots F-1$ y en cada término de la suma se calcula la diferencia entre los píxeles de la misma columna en dos filas consecutivas (se toma el cuadrado para evitar negativos).

La diferencia entre dos filas consecutivas muy parecidas serán bajas, y cuanto más parecidas sean las filas, menor será el valor de la diferencia. Es de esperar que la imagen "verdadera" sea la que proporcione el mínimo valor entre todas las configuraciones válidas.

Sugerencias para la implementación

Escriba un **constructor** de la clase **Imagen** que reciba el nombre de un fichero con valores de luminosidad y cree una imagen con las dimensiones *estimadas*. El constructor también establecerá los siguientes comentarios en el objeto **Imagen**:

```
# Creada por: ESTIMACION DE DIMENSIONES
# Fichero de datos: fichero
```

si *fichero* es el nombre del fichero que contiene los valores de luminosidad.

Escriba un método para guardar una imagen en un fichero PGM.

Evidentemente, necesitará de métodos adicionales. Escriba todos los que considere oportunos.

11. Aplicación 2: Reescribir programas

Reescribir los programas:

- Redimensionar imagen PGM.
- Binarizar imagen PGM.
- Negativo de una imagen PGM.

usando la clase `Imagen`.

12. Proyecto. Normas

Si está interesado en realizar el proyecto deberá escribirme un correo electrónico (`cb@decsai.ugr.es`). Necesito saber quién está trabajando en el proyecto para poder hacer un seguimiento personalizado.

El proyecto se empaquetará en el fichero `Imagenes.tar` y se entregará usando PRADO.

La entrega se realizará hasta las 0:00 del día en el que se celebre el examen ordinario.