Normas para la realización del examen: **Duración: 2:30 horas**

- **El único material permitido durante la realización del examen es un lápiz o bolígrafo (azul o negro).**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

Una imagen digital de niveles de gris puede verse como una matriz bidimensional en la que cada casilla almacena un valor de luminosidad. En este contexto cada casilla se denomina **píxel**. Por simplicidad supondremos que los valores de luminosidad son valores `int` que pertenecen al conjunto $\{0, 1, \ldots, 255\}$ 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). Ver figura 1 (A, B y C).

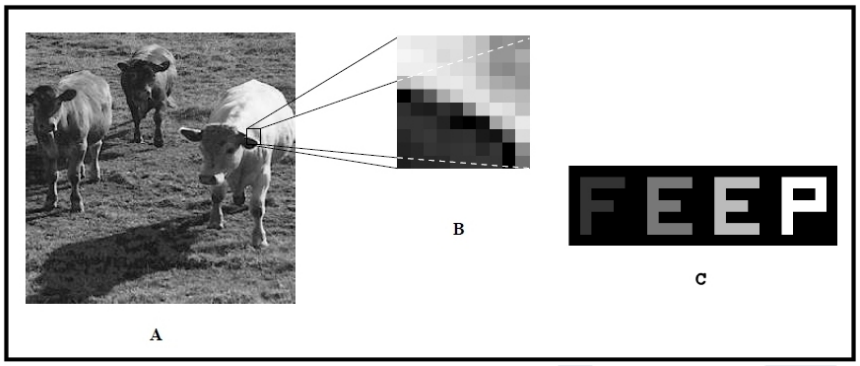

Figure 1: A) Imagen de 256 filas y 256 columnas con 256 niveles de gris. B) Una subimagen de la imagen A con 10 filas y 10 columnas. C) Imagen de 7 filas y 24 columnas con 5 niveles de gris.

La representación de una imagen (clase `Imagen`) debería permitir gestionar imágenes de tamaño arbitrario. Estamos interesados en mantener el historial de las operaciones realizadas y para ello usaremos cadenas de caracteres que describan las operaciones efectuadas que guardaremos en un objeto de la clase `Comentarios`. **Todos los comentarios empiezan por el carácter #**.

```
class Imagen {
  private:
    // PRE: 0 <= filas,columnas
    int filas, columnas;

    // PRE: 0 <= valores[i][j] <= 255
    // 0 <= i < filas, 0 <= j < columnas
    int ** valores; // Valores de luminosidad

    Comentarios comentarios;
```

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

    string * los_comentarios;
```

Suponga que dispone de los siguientes métodos públicos:

- Clase **Comentarios**:

```
Comentarios (const Comentarios & otro);  // Constructor de copia
int GetNumComentarios (void) const; // Devuelve el número de comentarios
Comentarios & operator = (const Comentarios & otro);  // Operador =
friend istream & operator >> (istream & in,  Comentarios & c); // Entrada
friend ostream & operator << (ostream & out, const Comentarios & c); // Salida
```

- Clase **Imagen**:

```
Imagen (const Imagen & otro); // Constructor de copia
int GetNumFilas (void) const; // Devuelve el num. de filas
int GetNumColumnas (void) const; // Devuelve el num. de columnas
friend ostream & operator << (ostream & out, const Imagen & img); // Salida
```

1. **Deberá implementar cualquier otra función/método que pudiera necesitar**.
2. **Si no se especificara la clase a la que pertenece un método deberá determinar cuál es la adecuada**.

◁ Ejercicio 1 ▷ Métodos de la clase Comentarios [0.75 puntos]

1. *(0.25 puntos)* Constructor sin argumentos. ¿Debería implementarse el **destructor**? Si fuera así, hágalo. Si no, explíquelo.
2. *(0.5 puntos)* Implemente el operador += para **añadir** (al final) un nuevo comentario. Éste es un dato `string`.

◁ Ejercicio 2 ▷ Métodos básicos de la clase Imagen [1.25 puntos]

1. *(0.5 puntos)* **Constructores** de la clase **Imagen**:
   (a) sin argumentos, para crear una imagen vacía. Guarda el comentario: `# Creada vacia`
   (b) con dos o tres argumentos: núm. de filas (f), núm. de columnas (c) y opcionalmente un valor de luminosidad (v), para crear una imagen plana (todos los píxeles tienen el mismo valor) con f filas y c columnas. Si se usan dos parámetros la imagen se inicializa al valor 0 y si se usan tres, al valor v. Guarda el comentario: `# Creada` **f** `x` **c** `a valor` **v** donde **f** es el valor del número de filas, **c** es el valor del número de columnas y **v** es el valor con el que ha inicializado la imagen.
2. *(0.5 puntos)* **Operador de asignación** para la clase **Imagen**.
3. *(0.25 puntos)* ¿Debería implementarse el **destructor**? Si fuera así, hágalo. Si no, razone por qué no es preciso.

◁ Ejercicio 3 ▷ Métodos y funciones para E/S [1.25 puntos]

1. *(0.75 puntos)* Sobrecargar el operador >> para tomar de un flujo de texto una serie indeterminada de comentarios, el número de columnas y filas de una imagen, un número entero que indica el valor máximo de luminosidad (que por simplificar el examen será siempre 255) y finalmente, los valores de luminosidad de la imagen.
   El formato de los datos de entrada es el de un fichero de imagen PGM (P2) excepto que este opefrador no espera la cadena mágica P2 (ver descripción detalllada en la figura 3).
2. *(0.5 puntos)* Implemente un constructor de la clase **Imagen** con un argumento que indica el nombre de un fichero *de texto* de tipo **PGM**. Los comentarios que se guardan en el objeto serán los comentarios del fichero.
   En la figura 3 describimos el formato de fichero de imagen PGM. Un ejemplo de fichero PGM con este formato se muestra en la figura 1.D.

◁ Ejercicio 4 ▷ Sobrecarga de operadores [1 punto]

1. *(0.25 puntos)* Sobrecargue el operador () para acceder a un píxel. Debería poder usarse tanto como *lvalue* (p.e. para dar valor a un píxel) como *rvalue* (p.e. para consultar un píxel). **Las filas y columnas se numeran desde 1** (consecuencia: la primera casilla -esquina superior izquierda- es la (1,1))
2. *(0.75 puntos)* 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.
   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 linea 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 4 mostramos un ejemplo del operador $*$. Observe cómo se va actualizando la lista de comentarios.

◁ Ejercicio 5 ▷ Métodos de cálculo y sobrecarga de operadores relacionales [1 punto]

1. *(0.5 puntos)* Implemente el método **SubImagen** que devuelve una nueva imagen, resultado de copiar una zona rectangular de la imagen. Se especificará la coordenada de la esquina superior izquierda de la región de interés (fila f y columna c), así como el alto (h) y el ancho (w). **Se supone que los valores suministrados son correctos: puede obtenerse una subimagen**. La imagen resultante tendrá los comentarios de la imagen, una línea de guiones y el comentario adicional:
   `# Subimagen desde` **f**, **c** `de dimensiones` **h** `x` **w**
2. *(0.25 puntos)* Implemente **ValorMedio**, un método **privado** para calcular y devolver el valor medio de la distribución de los valores de luminosidad de una imagen.
3. *(0.25 puntos)* Implemente el operador relacional > para comparar dos imágenes. El criterio usado para la comparación está basado en el valor calculado por el método **ValorMedio**: una imagen es mayor que otra si el valor medio de la primera es mayor que el de la segunda.

◁ Ejercicio 6 ▷ Aplicación [0.75 puntos]

Escriba un programa *completo* que reciba como argumentos:

- dos números enteros que indican las coordenadas -fila y columna- de la esquina superior izquierda de una subimagen,
- otros dos números enteros que indican el alto y el ancho de la subimagen,
- los nombres de dos archivos PGM con el formato descrito en la pregunta 3.

El programa extraerá una subimagen a partir de cada una de las imágenes leídas de cada fichero PGM y calculará la *mayor* subimagen. Indicará el nombre del fichero PGM en el que se encuentra la mayor subimagen.

Compruebe y decida actuaciones para todas las situaciones de error (coordenadas, dimensiones, existencia de ficheros, . . .).

```
P2
# FEEP.pgm
# Creado para el examen de MP
24 7
255
0 0   0   0   0   0   0 0   0   0   0   0   0 0   0   0   0   0   0 0   0   0   0   0
0 51  51  51  51  0   0 119 119 119 119 0   0 187 187 187 187 0   0 255 255 255 255 0
0 51  0   0   0   0   0 119 0   0   0   0   0 187 0   0   0   0   0 255 0   0   255 0
0 51  51  51  0   0   0 119 119 119 0   0   0 187 187 187 0   0   0 255 255 255 255 0
0 51  0   0   0   0   0 119 0   0   0   0   0 187 0   0   0   0   0 255 0   0   0   0
0 51  0   0   0   0   0 119 119 119 119 0   0 187 187 187 187 0   0 255 0   0   0   0
0 0   0   0   0   0   0 0   0   0   0   0   0 0   0   0   0   0   0 0   0   0   0   0
```

Figure 2: Fichero PGM (P2) que corresponde a la imagen C de la figura 1

PGM es el acrónimo de formato de ficheros **P***ortable* **G***ray* **M***ap*. Estamos interesados únicamente en ficheros PGM de tipo texto (**P2**) por lo que cuando nos refiramos a PGM siempre será a esta clase de ficheros PGM. El formato (simplificado) es:

- Una *cadena mágica* que identifica el tipo de fichero. En nuestro caso será la cadena **P2** y un separador (salto de línea). `P2` indica que el fichero contiene una imagen de *niveles de gris* con valores de luminosidad codificados en *texto*.
- Un número indeterminado de comentarios. Los comentarios son de línea completa. El primer carácter de una línea de comentario siempre será el carácter #. La longitud máxima es de 70 caracteres.
- El ancho o número de columnas (c), un separador y el alto o número de filas (f), y un separador (salto de línea).
- El máximo nivel de luminosidad de la imagen (m) y un separador (salto de línea). Por simplicidad valdrá 255.
- Los $c \times f$ valores de luminosidad, que son números enteros en formato texto. Entre cada dos valores hay un separador. El primero corresponde al pixel de la esquina superior izquierda, el segundo al pixel que está a su derecha, etc. (el último byte corresponde al píxel de la esquina inferior derecha de la imagen). En definitiva, los valores de luminosidad se organizan según el orden de lectura (izquierda a derecha y de arriba abajo).

Figure 3: Descripción del formato de imagen PGM (P2)

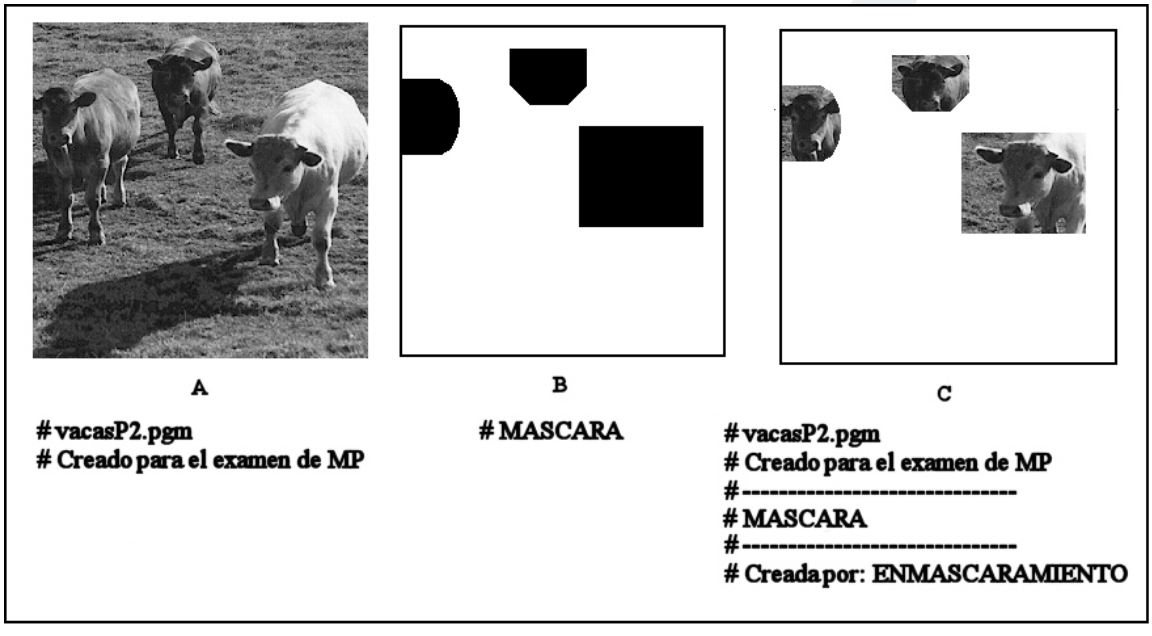

Figure 4: A) Imagen original. B) Una máscara. C) Resultado de enmascarar (operador *) A con B