



## **Guion de prácticas 3**

*Planos de bits y arte ASCII*

*Marzo de 2016*



**Metodología de la Programación**

Curso 2015/2016



# Índice

<b>1. Definición del problema</b>	<b>5</b>
<b>2. Objetivos</b>	<b>5</b>
<b>3. Imágenes</b>	<b>5</b>
<b>4. Extracción de un plano de bits</b>	<b>8</b>
<b>5. Arte ASCII</b>	<b>9</b>
<b>6. Material a entregar</b>	<b>11</b>



## 1. Definición del problema

En este guion se plantea el uso de imágenes digitales, concretamente en escala de grises (fotos en blanco y negro), y un par de aplicaciones de las mismas. En la primera aplicación se propone extraer planos de bits de una imagen. La segunda aplicación crea arte ASCII<sup>1</sup> que consiste en representar imágenes con los 95 caracteres imprimibles (de los 128) definidos en el estándar ASCII<sup>2</sup> y, a veces, también con otros caracteres no estándar.

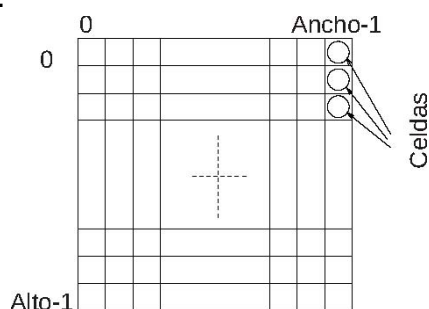
## 2. Objetivos

El desarrollo de esta práctica pretende servir a los siguientes objetivos:

- manejo de cadenas C y arrays
- practicar el paso de parámetros por referencia
- practicar el paso de parámetros de tipo array y cadenas C
- creación de bibliotecas

## 3. Imágenes

Desde un punto de vista práctico, una imagen se puede considerar como una matriz bidimensional de celdas, llamadas píxeles, tal como muestra la siguiente figura:



Cada celda de la matriz almacena la información de un píxel. Para imágenes en blanco y negro, cada píxel se suele representar con un byte<sup>3</sup> (8 bits). El valor del píxel representa su tonalidad de gris que va desde el negro (0) hasta el blanco (255). Un píxel con valor 128 tendrá un gris intermedio entre blanco y negro. En la siguiente imagen se puede observar el valor de los píxeles para una pequeña porción de una imagen.

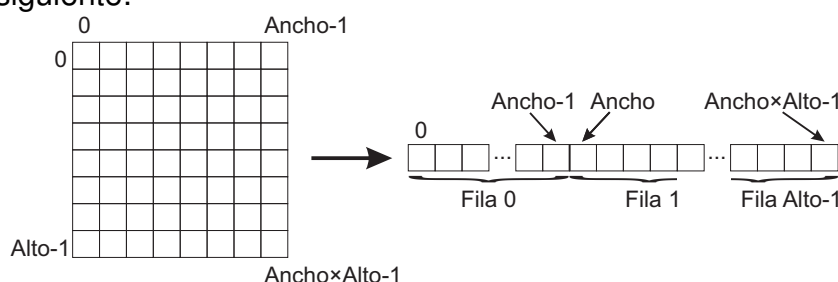
<sup>1</sup>[https://es.wikipedia.org/wiki/Arte\\_ASCII](https://es.wikipedia.org/wiki/Arte_ASCII)

<sup>2</sup><https://es.wikipedia.org/wiki/ASCII>

<sup>3</sup>Recuerde que en C++ un "unsigned char" almacena exactamente un byte.



Pese a que una imagen se trata habitualmente como una matriz bidimensional de bytes, es usual representarla internamente como un vector en el que las filas se van guardando una tras otra, almacenando consecutivos todos los bytes de la imagen. Así, la posición 0 del vector tendrá el píxel de la esquina superior izquierda, la posición 1 el de su derecha, y así hasta el píxel de la esquina inferior derecha, como se muestra en la figura siguiente:



Así se puede acceder fácilmente a las posiciones de la imagen de forma consecutiva pero, para acceder a cada píxel  $(x, y)$  de la imagen es necesario convertir las coordenadas de imagen  $(x, y)$  en la coordenada de vector  $(i)$ . Para ello se aplicará la siguiente fórmula:

$$i = y * \text{columnas} + x.$$

## Ejercicio 1

Usando esta representación, debe crear la clase Imagen según la siguiente especificación:

```
typedef unsigned char byte;

class Imagen{
private:
    static const int MAXPIXELS = 1000000; ///< número máximo de píxeles que podemos almacenar
    byte datos[MAXPIXELS]; ///< datos de la imagen
    int nfilas; ///< número de filas de la imagen
    int ncolumnas; ///< número de columnas de la imagen
public:
    // Construye una imagen vacía (0 filas , 0 columnas)
    Imagen();
    // Construye una imagen negra de tamaño filas x columnas
    Imagen(int filas , int columnas);
```

```
// Crea una imagen negra de tamaño filas x columnas
void crear(int filas , int columnas);
// Devuelve el numero de filas de las imagen
int filas();
// Devuelve el numero de columnas de la imagen
int columnas();
// Asigna el valor v a la posicion (x,y) de la imagen
void set(int y, int x, byte v);
// Devuelve el valor de la posicion (x,y) de la imagen
byte get(int y, int x);
// Asigna el valor v a la posicion i de la imagen
considerada como vector
void setPos(int i, byte v);
// Devuelve el valor de la posicion i de la imagen
considerada como vector
byte getPos(int i);
};
```

Para almacenar las imágenes en el disco duro usaremos el formato *Portable Gray Map* —PGM (<http://netpbm.sourceforge.net/doc/pgm.html>)— y, para no tener que ocuparnos del formato, se proporcionan los ficheros `pgm.h` y `pgm.cpp` con las funciones básicas de lectura y escritura de imágenes en este formato. El fichero de cabecera `pgm.h` contiene lo siguiente:

```
#ifndef _PGM_H_
#define _PGM_H_
// Tipo de imagen
enum TipolImagen {
    IMG_DESCONOCIDO,    ///< Tipo de imagen desconocido
    IMG_PGM_BINARIO,    ///< Imagen tipo PGM Binario
    IMG_PGM_TEXTO      ///< Imagen tipo PGM Texto
};

// Información sobre la imagen (tipo , filas y columnas)
TipolImagen infoPGM (const char nombre[], int &filas ,
    int &columnas);
// Lee de disco una imagen en formato PGM binario
bool leerPGMBinario (const char nombre[], unsigned char
    datos[], int &filas , int &columnas);
// Escribe en disco una imagen en formato PGM binario
bool escribirPGMBinario (const char nombre[], const
    unsigned char datos[], int filas , int columnas);
#endif
```

Además, se incluye documentación en formato doxygen para que sirva de muestra y pueda ser usada como referencia para estas funciones. Ejecute “make documentacion” en el paquete que se le ha entregado para obtener la salida de esa documentación en formato HTML (use un navegador para consultarla).

**Ejercicio 2** Ampliar la clase Imagen antes creada con estos dos métodos:

```
// Carga una imagen desde el fichero
bool leerImagen(const char nombreFichero[]);
// Guarda la imagen en fichero. El parámetro esBinario,
// que siempre tomará el valor true, se añade para
// su uso en el futuro.
bool escribirImagen(const char nombreFichero[], bool
esBinario);
```

Tanto la lectura como la escritura tratará sólo con imágenes PGM binario. Para leer, el alumno tiene que asegurarse de que la imagen es de tipo IMG\_PGM\_BINARIO (usando la función infoPGM()) y que su tamaño es inferior a MAXPIXELS antes de leer la imagen usando leerPGMbinario().

Crear la biblioteca libimagen.a con los ficheros imagen.o y pgm.o.

Probar la corrección de clase compilando y ejecutando el programa testimagen.cpp proporcionado en el directorio src. Se debe crear un makefile que compile los fuentes, cree la biblioteca y cree el ejecutable. El resultado de ejecutar bin/testimage será:

```
$ bin/testimagen
degradado.pgm guardado correctamente
usa: display degradado.pgm para ver el resultado
trozo.pgm guardado correctamente
usa: display trozo.pgm para ver el resultado
```



degradado.pgm



trozo.pgm

## 4. Extracción de un plano de bits

El ojo humano no es capaz de distinguir un gran número de tonos de gris. De hecho, en escenas relativamente complejas, no es capaz de captar pequeñas variaciones de tono. Podemos aprovecharnos de esta característica para guardar información de una imagen dentro de otra alterando alguno de sus bits.

**Ejercicio 3** Cree los archivos byte.h y byte.cpp copiando el módulo bloqueLed definido en la practica 2, y reemplazando los términos “bloqueLed” y “LED” por “byte” y “bit”, respectivamente. Para evitar posibles



confusiones renombraremos la función “get” como “getbit”. De esta forma, se podrán utilizar las operaciones definidas para manipular LEDs en `bloqueLed` para manipular bits en los bytes que componen las imágenes.

Definimos el plano  $k$ -ésimo de una imagen como una nueva imagen con un tamaño idéntico, en la que el valor de cada píxel se obtiene colocando en el bit más significativo (el que ocupa la posición 7) el bit  $k$ -ésimo del píxel correspondiente en la imagen original y el resto de bits a cero.

**Ejercicio 4** Ampliar la clase anterior con un método que dado un número,  $k$ , extraiga el plano de bits  $k$ -ésimo de la imagen actual y lo devuelva como una nueva imagen. Su cabecera será:

```
Imagen plano(int k);
```

Probar la corrección de clase compilando y ejecutando el programa `testplano.cpp` proporcionado en el directorio `src`. Se debe ampliar el `makefile` para que compile los fuentes, cree la biblioteca y cree el ejecutable. El resultado de ejecutar `bin/testplano` será:

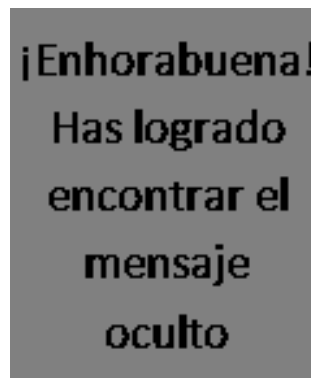
```
$ bin/testplano
plano6.pgm guardado correctamente
usa: display plano6.pgm para ver el resultado
plano0.pgm guardado correctamente
usa: display plano0.pgm para ver el resultado
```



*giotexto.pgm*



*plano6.pgm*



*plano0.pgm*

## 5. Arte ASCII

El arte ASCII trata de hacer imágenes con caracteres ASCII. Un método consiste en representar cada píxel de una imagen con un carácter que representa su nivel de gris. Así si el píxel es casi blanco, se puede sustituir por un punto (.), si es gris claro con una o (o), si es más oscuro con una equis (x), y si es casi negro con una arroba (@). Para visualizar la imagen en arte ASCII, el texto resultante se visualiza con una fuente de ancho fijo (como Courier).

Más formalmente, si el conjunto de caracteres de salida es “@xo.”, todos los píxeles con valores en el intervalo  $[0, 63]$  se sustituirán por el



<sup>4</sup>Estas cadenas están en el fichero `grises.txt` en el directorio `imagenes`.