
# Guion de prácticas 3

*Imágenes digitales*

*Marzo de 2017*

DECSAI

Metodología de la Programación

Curso 2016/2017

# Índice

# 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[1] que consiste en representar imágenes con los 95 caracteres imprimibles (de los 128) definidos en el estándar ASCII[2] y, a veces, también con otros caracteres no estándar.

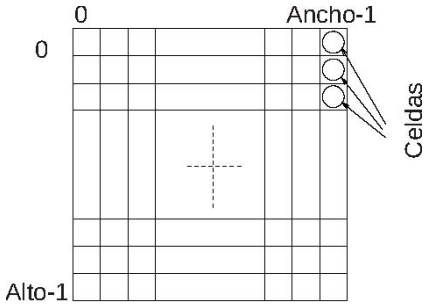
# 2. Objetivos

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

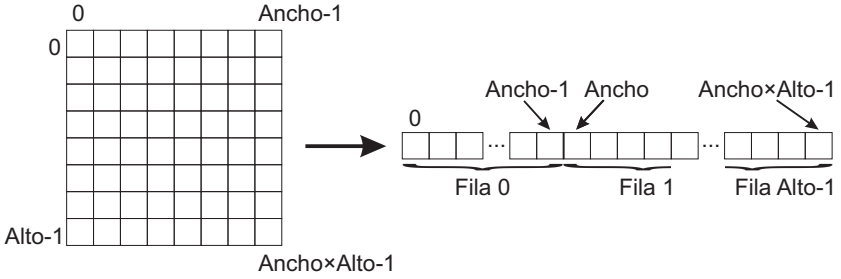
Desde un punto de vista práctico, una imagen se puede considerar como una matriz bidimensional de celdas, llamadas píxeles, tal como muestra la siguiente figura:




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[3] (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.

[1] https://es.wikipedia.org/wiki/Arte_ASCII
[2] https://es.wikipedia.org/wiki/ASCII
[3] 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:

$$i = y * Ancho + x.$$

### Ejercicio 1

*Usando esta representación y las funciones de acceso a memoria bit a bit de la práctica anterior, debe crear la clase* Imagen *según la siguiente especificación:*

```
/** @brief Una imagen en blanco y negro. Cada píxel es un 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 columnsa de la imagen
public:
/** @brief Construye una imagen vacía (0 filas, 0 columnas) */
   Imagen();
/** @brief Construye una imagen negra de tamaño @a filas x @a
    columnas */
   Imagen(int filas, int columnas);
/** @brief Crea una imagen negra de tamaño @a filas x @a
    columnas */
   void crear(int filas, int columnas);
/** @brief Devuelve el número de filas de las imagen */
   int filas() const;
/** @brief Devuelve el número de columnas de las imagen */
   int columnas() const;
/** @brief Asigna el valor @a v a la posición (@a x,@a y) de la
    imagen */
   void set(int y, int x, Byte v);
/** @brief Devuelve el valor de la posición (@a x,@a y) de la
    imagen */
   Byte get(int y, int x) const;
/** @brief Asigna el valor @a v a la posición @a i de la imagen
    considerada como vector */
   void setPos(int i, Byte v);
/** @brief Devuelve el valor de la posición @a i de la imagen
    considerada como vector */
   Byte getPos(int i) const;
};
```

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 TipoImagen {
  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)
TipoImagen 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
```

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

```
/** @brief Carga una imagen desde un fichero */
   bool leerImagen(const char nombreFichero[]);
/** @brief Guarda una imagen en un fichero */
   bool escribirImagen(const char nombreFichero[], bool
    esBinario) const;
```

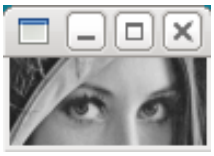
*Tanto la lectura como la escritura tratará sólo con imágenes PGM binario. Para leer, el alumno tiene que asegurarse de que la imagen es de tipo* `IMG_PGM_BINARIO` *(usando la función* `infoPGM()`*) y que su tamaño es inferior a* `MAXPIXELS` *antes de leer la imagen usando* `leerPGMbinario()`*.*

*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.

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 3** *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:*

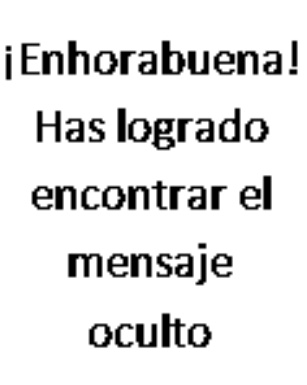
```
Imagen plano(int k);
```

*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 carácter "`@`", los valores en el intervalo $[64, 127]$ se sustituirán por el carácter "`x`", los valores en el intervalo $[128, 191]$ se sustituirán por el carácter "o" y, finalmente, los valores en el intervalo $[192, 255]$ por el carácter "`.`". En general, si el conjunto de caracteres tiene cardinal $cardinal$, el carácter de salida será el que ocupe la posición entera $valorDelPixel * cardinal/256$ dentro del conjunto de caracteres.

**Ejercicio 4** *Ampliar la clase anterior con un método cuya cabecera sera:*

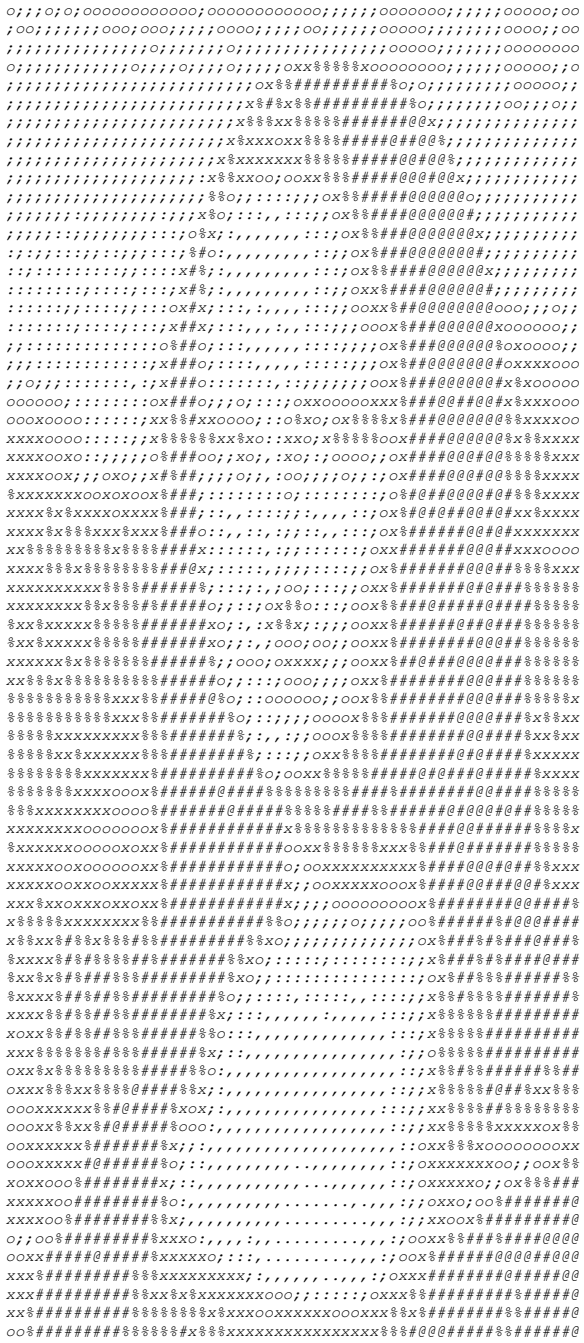
```
bool aArteASCII(const char grises[], char arteASCII[], int
    maxlong);
```

*donde* `grises` *es un cstring, que contiene el conjunto de caracteres que se usarán en el dibujo,* `arteASCII` *es otro cstring que contendrá la imagen convertida a arte ASCII y* `maxlong` *es el número máximo de caracteres que puede almacenar* `arteASCII`*. La conversión se realizará según el proceso descrito anteriormente, teniendo en cuenta que tras convertir una fila completa de la imagen hay que incluir un carácter* `\n` *en la cadena de salida. El método devolverá* `false` *si la imagen convertida no cabe en el cstring de salida y* `true` *en caso contrario.*

*Probar la corrección de clase compilando y ejecutando el programa* `testarteASCII.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/testarteASCII > gio.txt
```

*será un fichero llamado* `gio.txt` *cuyo contenido es*



El programa de prueba puede extenderse para usar cualquier conjunto de caracteres. Se sugiere probar con “`#$:.`”, o con “`"mQt]?'␣`”, o con “`"@%#*+=-:.␣`”, o con la extremadamente larga: “`$B%8&WM#*oahkbdpqwmZ0OQLCJUYXzcvunxrjft /|()1{}[]?-_+<>i!lI;:,"^'‘.␣`”[4].

# 6. Material a entregar

Cuando esté todo listo y probado el alumno empaquetará la estructura de directorios anterior en un archivo con el nombre **practica3.zip** y lo entregará en la plataforma **decsai** en el plazo indicado. No deben entre-

[4] Estas cadenas están en el fichero `grises.txt` en el directorio `data`.

garse archivos objeto (**.o**) ni ejecutables. Para asegurarse de esto último conviene ejecutar **make zip** antes de proceder al empaquetado.

El fichero **practica3.zip** debe contener la siguiente estructura:

```
./
├── makefile
├── include
│   ├── byte.h
│   ├── imagen.h
│   └── pgm.h
├── src
│   ├── byte.cpp
│   ├── imagen.cpp
│   ├── pgm.cpp
│   ├── testimagen.cpp
│   ├── testplano.cpp
│   └── testarteASCII.cpp
├── bin
├── obj
├── lib
├── data
│   ├── lena.pgm
│   ├── giotexto.pgm
│   ├── gio.pgm
│   └── grises.txt
└── zip
```

El alumno debe asegurarse de que ejecutando las siguientes órdenes se compila y ejecuta correctamente su proyecto:

```
unzip practica3.zip
make
bin/testimagen
bin/testplano
bin/testarteASCII
```

Gradebook Practicas Grupo F MP1516 - Práctica 3

| | Tipo de error | Frecuen |
|---|---|---|
| EC | Error makefile | 7 |
| EE | Errores de compilación | 2 |
| EI | Faltan #ifndef en .h | 0 |
| EM | Faltan los .h en el cuerpo de algunas de las reglas de makefile | 0 |
| ES | Reglas mal formadas en makefile | 0 |
| EU | ZIP mal | 3 |
| EV | Faltan macros en makefile | 0 |
| EW | Errores de ejecución | 8 |
| EX | main() manipulado | 0 |
| EY | Copia o deja copiar | 0 |
| EZ | Falta resumen | 3 |
| E7 | Error de acceso a memoria (SEGFAULT) | 0 |
| ED | Errores de casuística en funciones con return (posible return aleatorio) | 4 |
| EK | Faltan comprobaciones de seguridad en acceso a vectores | 15 |
| EL | Faltan gestión de errores en apertura/grabación de ficheros o funciones booleanas | 2 |
| EF | Métodos de clase redundantes | 8 |
| EG | Error en arteASCII | 14 |
| | | |
| E6 | Entregado fuera de plazo | 2 |

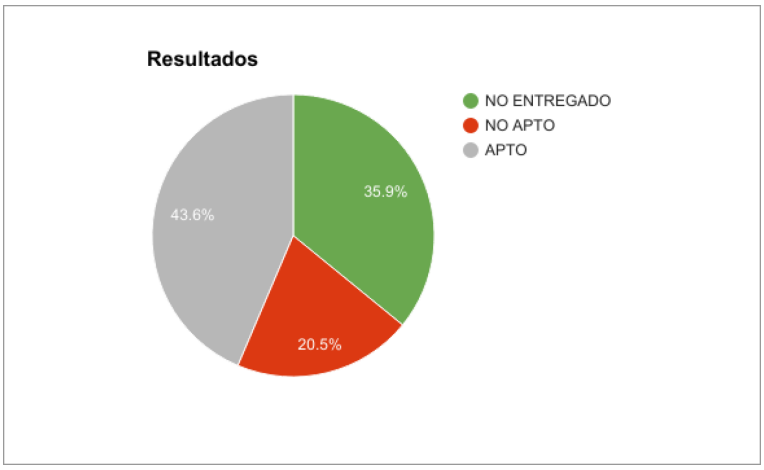




Figura 1: Estadísticas y principales errores del curso 2015-2016