INFORMÁTICA APLICADA curso 2006-2007

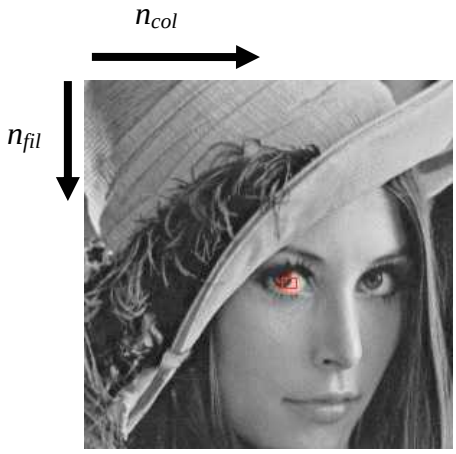
PRÁCTICA 6: Clases en C++

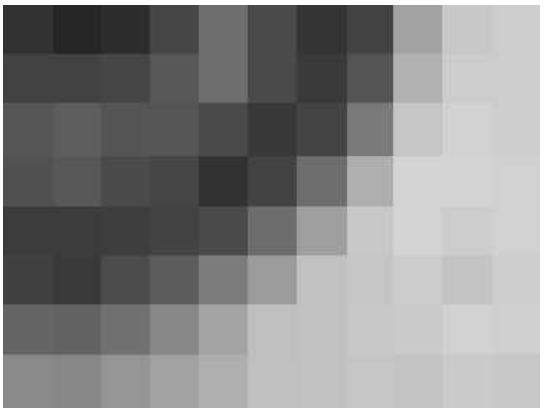
El objetivo de esta práctica es introducir al alumno en la programación orientada a objetos en C++. Para ello deberá implementarse una clase en C++ para manejar imágenes digitales. En esta práctica manejaremos imágenes digitales en niveles de gris, como la que se muestra en la figura 1.

Fig. 1

1. Introducción. Imágenes digitales

Podemos entender una imagen digital como un array bidimensional, en el cual, cada elemento tiene un valor proporcional al brillo de ese punto. Valores cercanos a cero representan elementos oscuros de la imagen, mientras que valores altos representan zonas claras. Cada uno de estos valores se denomina pixel, término anglo-sajon que deriva de *Picture Element*. El array tendrá un número de columnas y filas determinado (n_{fil}, n_{col}). Esto lo podemos ver representado en la figura 2.





50	38	44	70	110	74	52	66	162	200	206
66	66	70	88	110	74	58	84	178	206	206
86	94	84	86	74	56	68	122	198	210	206
80	88	74	70	50	66	110	176	212	212	210
60	60	62	66	74	108	160	200	212	206	210
64	58	76	92	124	156	194	198	204	196	206
100	98	112	136	164	192	194	200	202	210	208
138	136	150	162	176	192	194	198	196	202	200

Fig. 2

2. Formatos

Las imágenes digitales se almacenan en un fichero con un formato determinado. Existen una gran cantidad de formatos, cada uno de ellos está más indicado para una aplicación en concreto. En nuestro caso vamos a trabajar con imágenes en formato `pgm` (*Portable Gray Map*). A continuación se muestra un ejemplo de una imagen en formato `pgm`.

```
P2
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  3  3  3  3 255  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0 255  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0 255  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0 255  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0 255  0  7  7  7  7  0  0 11 11 11 11  0  0 15  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
```

Fig. 3

El formato del fichero es el siguiente:

P2: Es una cadena de caracteres que identifica el formato pgm.
Número de columnas: A continuación se escribe el número de columnas de la imagen (24).
Número de filas: Separado por un espacio se escribe el número de filas de la imagen (7).
Valor máximo: Separado por un retorno de carro se escribe el valor máximo de cada

elemento de la imagen (255, es decir, cada elemento ocupa un byte).
Píxeles: A continuación se escriben separados por espacios cada uno de los pixeles en formato ASCII (1 byte cada uno).
Filas: Las diferentes filas de la imagen están separadas por el carácter '\n'.

Junto con la práctica se proporcionan algunas imágenes de prueba. Podemos abrir cada una de ellas utilizando un visor estándar de imágenes. En el siguiente link se encuentra un visor muy versátil: http://www.stratopoint.com/download/11view.exe

3. Compresión de imágenes

El formato que hemos elegido para almacenar la información de la imagen no es demasiado eficiente. Dependiendo del número de filas, de columnas de la imagen, ésta ocupará una cantidad de memoria. El número de bytes que ocupa la imagen lo podemos calcular de la siguiente manera:

Nº de Bytes = $n_{fil} \cdot n_{col} \cdot P$

Donde P es el número de bytes que ocupa cada pixel de la imagen (1 byte en nuestro caso). Por ejemplo, si una imagen tiene 500 filas x 500 columnas y cada uno de sus píxeles ocupa 3 bytes, en total tendremos: nº de bytes = 750.000 Bytes = 732 KB.

Podemos pensar en una manera de almacenar la imagen mucho más eficiente, lo que comprimiría la información. La idea fundamental se basa en almacenar el valor de un píxel y, justo después, el número de veces consecutivas que se repite el pixel en la imagen. Por ejemplo, si tenemos esta secuencia de píxeles: 0 0 0 0 50 50 128 128 128 128 128 50 50 50 0 0 0 0. Esta secuencia se sustituye por: 0 4 50 2 128 4 50 3 0 4. Es decir, el 0 se repite 4 veces, el 50 se repite 2 veces… etc. En la figura 4 podemos ver un ejemplo más completo de cómo funciona el método:

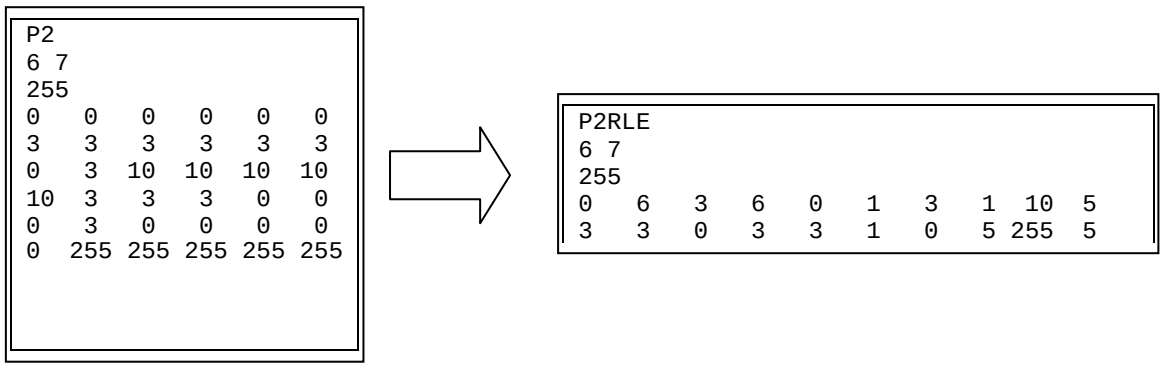


Fig. 4

Nótese como, en este caso, la imagen ha pasado de ocupar 6x7x1Byte = 42 Bytes a ocupar 20 Bytes. Se debe comentar, por último, que el método es capaz de comprimir la información siempre y cuando existan píxeles consecutivos que se repitan.

4. Tareas a realizar

1) Crear y configurar un proyecto en Dev C++ que se llame **practica7**. El fichero de código fuente en el que se implementará la clase CImagen se llamará *CImagen.cpp* y el fichero cabecera en el que se declarará la clase, *CImagen.h*. El fichero fuente en el que se creará el *main* para probar el funcionamiento de la clase se denominará *MainImagen.cpp*.

2) Definir la siguiente clase C++.

```
class CImagen{

private:
        unsigned char *imagen; //Elementos de la imagen
        int nfil;              //Número de filas de la imagen
        int ncol;              //Número de columnas de la imagen
        int maxVal;            //Valor máximo de cada pixel
        int imagCargada;       // 0 si no se ha cargado la imagen

public:
       CImagen(int nfil=320, int ncol=240, int maxval=255);
       CImagen(CImagen &imag_orig);
       ~CImagen(void);

       //Lee una imagen desde un fichero
       int LeerFichero(char *nombrefichero);
       int LeerFicheroComprimido(char *nombrefichero);

       int GuardarFichero(char *nombrefichero);
       int GuardarFicheroComprimido(char *nombrefichero);

       int Multiplicar(int K);
       int Dividir(int K);
};
```

3) Programe los siguientes constructores:

- `CImagen(int nfil=320, int ncol=240, int maxval=255);`

Constructor con número de filas y columnas por defecto. Reserva memoria para una imagen de tamaño nfil x ncol.

- `CImagen(CImagen &imag_orig);`

Constructor copia. Deberá copiar las dimensiones de la imagen así como los datos de ella.

- `~CImagen(void);`

Destructor. Deberá liberar la memoria asignada a la imagen.

4) Programe los siguientes métodos de la clase CImagen:

- int LeerFichero(char *nombrefichero);

Este método leerá una imagen en formato pgm y lo almacenará en la variable miembro `imagen`. El nombre del fichero se pasará como argumento al método. Por simplicidad, todos los elementos de la imagen se almacenarán en un array unidimensional. Como ejemplo se da el código de este método:

```
int CImagen::LeerFichero(char *nombrefichero)
{
    FILE *fp;
    char linea[50], val;
    int res, i=0, dat, num;

    fp = fopen(nombrefichero, "r");

    if(fp == NULL)
      return -1;

    //Leemos el encabezado del fichero
    fgets(linea, 49 ,fp);

    cout << "\nLeyendo fichero sin comprimir: " <<endl;


    fscanf(fp, "%d", &ncol);
    fscanf(fp, "%d", &nfil);
    fscanf(fp, "%d", &maxVal);
    cout << "\nNumero filas: " << nfil;
    cout << "\nNumero columnas: " << ncol;

    //Leemos del fichero y lo guardamos como datos de la imagen
    while(!feof(fp)&& i<nfil*ncol)
    {
        fscanf(fp, "%d", &dat);
        imagen[i] = dat;
        i++;
    }
    cout << endl << i << " datos leidos " << endl;

      //Hemos cargado la imagen
    imagCargada = 1;

    cout << "\nCerrando fichero: " <<endl;

    fclose(fp);

    return 0;
}
```

- int GuardarFichero(char *nombrefichero);

Guarda los datos de una imagen en un fichero con formato pgm. El nombre del fichero se pasará como argumento al método.

- int LeerFicheroComprimido(char *nombrefichero);

Lee los datos de una imagen a partir de un fichero comprimido (apartado 3). Almacena los datos en la variable miembro `imagen`.

- int GuardarFicheroComprimida(char *nombrefichero);

Guarda los datos de la variable miembro `imagen` en un fichero comprimido (apartado 3). Para ello, se deberá contar el número de elementos de la imagen que se repiten, almacenando consecutivamente en el fichero el valor y el número de veces que se repite.

- int Multiplicar(int K);

Multiplica todos los datos de una imagen por un factor K.

- int Dividir(int K);

Multiplica todos los datos de una imagen por un factor K.

En la función `main()`, la clase se utiliza de la siguiente manera:

```
CImagen im1(7, 24); // columnas, filas de la imagen

 im1.LeerFichero("imagen1.pgm");

 im1.GuardarFicheroComprimida("imagen1.rle");

 im1.LeerFicheroComprimida("imagen1.rle");

 im1.GuardarFichero("imagen1descomprimida.pgm");

 system("PAUSE");
 return EXIT_SUCCESS;
```

En este ejemplo se lee imagen1.pgm, se guarda comprimida, a continuación se abre según su formato y se descomprime. Finalmente, se guardan los datos de la imagen descomprimida en otro fichero. El alumno deberá comprobar que imagen1.pgm e imagen1descomprimida.pgm son iguales.

5) Finalmente, se deberán probar las funciones programando una función main que ofrezca las siguientes opciones:

1. Cargar imagen.
2. Cargar imagen comprimida.
3. Guardar imagen.
4. Guardar imagen comprimida.
5. Multiplicar
6. Dividir
7. FIN

A continuación se describen las acciones a realizar en cada opción:

1.- Deberá solicitar el nombre del fichero imagen a abrir. A continuación llamará a la función `LeerFichero`.
2.- Deberá solicitar el nombre del fichero imagen a abrir. A continuación llamará a la función `LeerFicheroComprimido`.
3.- Deberá solicitar el nombre del fichero imagen a guardar. A continuación llamará al método `GuardarFichero`.
4.- Deberá solicitar el nombre del fichero imagen a guardar. A continuación llamará al método `GuardarFicheroComprimido`.
5.- Invocará al método `Multiplicar`.
6.- Invocará al método `Dividir`.

7.- Finaliza el programa.

Nota: Con la práctica se suministran dos ficheros en formato pgm (imagen1.pgm e imagen2.pgm). Podemos ver los datos abriéndolos con un editor de texto (notepad p.e.) o con un visor de imágenes (http://www.stratopoint.com/download/11view.exe).