Guion de prácticas 5

Memoria Dinámica

Abril de 2015

DECSAI

Metodología de la Programación

Curso 2014/2015

Índice

1. Definición del problema

La implementación de la clase Imagen utilizada hasta ahora, almacena los pixeles en un vector cuyo tamaño se fija en tiempo de compilación. En la definición de la clase tenemos

```
typedef unsigned char byte;

class Imagen{
        private:
        //número máximo de píxeles que podemos almacenar
        static const int MAXPIXELS = 1000000;
        byte datos[MAXPIXELS];
        int nfilas;
        int ncolumnas;
        .......
}
```

Como consecuencia de esto, cada vez que se instancia un objeto de la clase Imagen, se ”crea” un vector de tamaño $MAXPIXELS$ aunque la imagen actual pueda tener unos pocos cientos.

Una manera de evitar este problema de desperdicio de memoria es ”solicitar” o ”pedir”, en tiempo de ejecución, solamente la memoria que se va a necesitar (conociendo a priori el tamaño de la imagen), utilizarla y luego ”liberarla”.

Por tanto en esta práctica haremos uso de los conceptos básicos de memoria dinámica a partir de la clase Imagen utilizada en guiones anteriores.

2. Objetivos

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

- Repasar conceptos básicos memoria dinámica.
- Modificar la clase Imagen de prácticas anteriores para incluir gestión de memoria dinámica.
- Implementar una nueva funcionalidad a la clase Imagen.

3. Descripción

En esta práctica realizaremos dos tareas:

1. Extender la clase Imagen para que los datos se almacenen en un vector dinámico.
2. Proveer una funcionalidad de rotación a la clase Imagen.

4. Memoria dinámica en la clase Imagen

En primer lugar, debe hacer una copia de la estructura de directorios del guión anterior (trabajará sobre la copia). Renombre *esteganografia.cpp* a *prueba.cpp*
Luego deberá realizar las siguientes tareas.

- Reimplementar la clase Imagen utilizando como estructura interna un vector dinámico. Ahora, el constructor de la clase debe reservar memoria (Repase los apuntes de teoría para implementar la reserva de memoria asociada a un vector). Más concretamente, el constructor sin parámetros debe crear una imagen vacía (0 filas, 0 columnas y sin memoria reservada), el constructor con parámetros debe reservar la memoria para la imagen. El método *void crear(int filas, int columnas)* debe liberar la memoria que tenga la imagen, si la hubiera, antes de reservar memoria de nuevo. Por último tenga en cuenta que el método de lectura debe crear la imagen antes de leer los datos.

- Implementar un método *destruir* que permita liberar la memoria reservada. Note que al destruir la imagen, además de liberar la memoria, también debe poner el número de filas y columnas a cero. Tenga en cuenta que sólo debe liberar la memoria si tuviera reservada, es decir, destruir una imagen ya destruida o vacía no debe producir ningún efecto. Este método debe llamarse explícitamente al final del programa. Cuando se imparta en teoría el tema de clases, se verá que la manera correcta de realizar esta tarea es mediante la utilización de un método ”destructor”. Por ahora, aplicaremos esta solución de compromiso

5. Extensión de la funcionalidad

Implemente un método privado llamado *rotar()* que permita rotar la imagen 90° en sentido horario. Posteriormente, implemente un método público

void rotación(int grados, bool sentidoHorario)

donde $grados \in \{90, 180, 270\}$ y el giro es en sentido horario si $sentidoHorario = True$ y anti-horario en otro caso. Naturalmente, este método hará las llamadas necesarias al método *rotar()* definido antes.

Al menú de opciones implementado en la Práctica 3, agregue ahora la opción de rotar la imagen. Cuando el usuario elija dicha opción, el programa le solicitará los grados y el sentido de la rotación.

6. Material a entregar

Cuando esté todo listo y probado el alumno empaquetará la estructura de directorios siguiente en un archivo con el nombre **practica5.zip** y lo entregará en la plataforma **decsai** en el plazo indicado. No deben entregarse archivos objeto (**.o**) ni ejecutables. Para asegurarse de esto último conviene ejecutar **make mrproper** antes de proceder al empaquetado.

Escribir un informe donde consten los nombres y DNI de los integrantes del grupo, los problemas que hayan podido surgir durante el desarrollo de la práctica, capturas de pantalla mostrando ejemplos del uso de la rotación, etc. Este informe, en formato pdf, se guardará en la carpeta doc.

El fichero **practica5.zip** debe contener la siguiente estructura:

```
imagen
├── makefile
├── bin
├── doc
│   ├── informe.pdf
│   └── incluya aqui la documentacion de Doxygen
├── imagenes
│   └── lena.pgm
├── include
│   ├── byte.h
│   ├── codificar.h
│   ├── imagen.h
│   └── pgm.h
├── lib
├── obj
└── src
    ├── byte.cpp
    ├── codificar.cpp
    ├── imagen.cpp
    ├── pgm.cpp
    └── prueba.cpp
```

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

```
unzip practica5.zip
cd imagen
make
bin/prueba
```