VNIVERSITAT DE VALÈNCIA

Laboratorio de PROGRAMACIÓN
Grado en Ingeniería Multimedia (1º)
Curso 2021-22

Práctica Nº 2: Análisis de algoritmos de ordenación

Periodo de realización: Semana del **28/02/2022 a 04/03/2022**

Material a entregar

- **(Repositorio) Proyecto compilable con el código fuente necesario para la práctica**.
- **(Tarea AV) Documentación del código**.

Introducción

Cuando se dispone de gran cantidad de información sobre un tema, uno de los principales problemas a los que hay que enfrentarse es la ordenación de los datos. Esta organización permite presentar los datos en un orden determinado, pero también facilita la ejecución de otras operaciones, como pueden ser las búsquedas (que en tal caso pueden hacerse mediante la búsqueda binaria). Es importante que el algoritmo elegido para ordenar sea eficiente, ya que se puede tener que ejecutar muchas veces y sobre grandes conjuntos de datos.

En esta práctica se pretende estudiar y comparar el coste de varios algoritmos de ordenación, en concreto los tres métodos vistos en la asignatura: ordenación por inserción, ordenación por selección y Quicksort. Los vectores de datos con los se van a trabajar son los mismos que en la anterior práctica, vectores de personas vacunadas. Pero en esta ocasión se ha modificado el tipo de datos definido para introducir la representación de las vacunaciones recibidas por las personas, en la que se reflejan los conceptos de programación orientada a objetos vistos en teoría. Los archivos "**`persona.h`**" y "**`persona.cpp`**" (Aula Virtual) incluyen la declaración y la implementación de la clase **Persona**.

El programa "**`pr2_v0_9.cpp`**", que se puede descargar desde el Aula Virtual, leerá los datos de un fichero de personas y sus vacunas y los almacenará en un vector en memoria. Es un esquema general que se deberá modificar para resolver los ejercicios que se plantean a continuación. Este programa hace uso de la clase Persona, por lo cual se deberá emplear un proyecto que combine los diferentes archivos fuente.

Compilación separada en Dev C++ (Proyectos)

Para realizar un programa con diferentes módulos (archivos) en *Dev-C++* se utiliza el concepto de 'proyecto'. Un proyecto es un conjunto de archivos escritos en C++ que una vez compilados conformarán un único programa ejecutable.

La manera de crear un proyecto en *Dev-C++* es utilizando la opción "`Nuevo Proyecto`" que aparece en el menú "`Archivo`". Una vez seleccionada la opción debemos seleccionar el tipo de proyecto y un nombre para guardarlo (p.e. **`proyecto_pr2`**). En este curso es recomendable crear un proyecto vacío (`Empty Project`).

Una vez creado el proyecto podemos añadir o eliminar archivos utilizando el botón "`Proyecto`" del menú o pinchando con el botón de la derecha sobre el árbol de proyecto desplegado de la izquierda del programa. Aunque no es necesario que los ficheros "`.h`" estén incluidos en el proyecto, pues no se compilan, es conveniente añadirlos para facilitar su consulta, escritura y manipulación.

Una vez tengamos escritas las diferentes unidades (archivos) de que conste nuestro programa podemos utilizar "`Reconstruir Todo`" para recompilar todos los módulos escritos del proyecto, o bien, las opciones habituales para "`Compilar`", "`Ejecutar`" o "`Compilar y Ejecutar`" un archivo concreto del proyecto.

Ejercicios

NOTA: Los comentarios etiquetados como SCV indican las tareas a realizar para llevar el seguimiento de revisiones del código generado.

Tarea 1

Antes que nada, debes cambiar el nombre al archivo **"`pr2_ver0_9.cpp`"**, que has descargado. Renómbralo como "**`pr2.cpp`**". Crea ahora un nuevo proyecto "**`proyecto_pr2`**" necesario para desarrollar nuestro programa empleando los diversos archivos que lo conforman. Debemos añadir al proyecto el programa principal **"`pr2.cpp`"**, el interfaz de la clase Persona (**"persona.h"**) y su implementación (**"`persona.cpp`"**), tal y como se indica en el apartado de "Compilación separada Dev-C++ (Proyectos)" de este guion. Si el proyecto tuviera asociado cualquier otro archivo (vacío) utiliza la opción "Quitar del Proyecto" para eliminarlo.

El programa proporcionado lee la información de las personas vacunadas almacenada en un fichero y la guarda en memoria como objetos de la clase Persona dentro de una estructura de datos adecuada para su tratamiento (*array/vector*). El programa también sirve de ejemplo para mostrar cómo utilizar un método de la clase Persona, *`LeerDeFichero`*, para leer los datos de una (sola) persona vacunada de un flujo de entrada y guardarlos en los atributos internos de un objeto de la clase.

SCV: Debes trabajar en la carpeta "Pr2" del repositorio que ya tienes creado para la asignatura. Procede como se indicaba en la tarea 1 de la práctica 2, descargando los archivos disponibles en Aula Virtual y haciendo un primer *commit con el mensaje "Tarea 1"* y un primer *push* con ese material inicial.

Tarea 2

El programa contiene: las definiciones y estructuras de datos, los prototipos de las diferentes funciones que se irán desarrollando y parte de la implementación de algunas de estas funciones. No obstante, contiene muchas partes incompletas o comentadas.

A continuación, debes completar la función *MostrarPersonas*, usando el método *Mostrar* de la clase *Persona* y comprueba el correcto funcionamiento del programa con el fichero "**vacunaciones_small.csv**" que se facilita en Aula Virtual y que contiene tan solo **10** personas vacunadas. El programa deberá mostrar los datos de las 10 personas y mediante la función *Ordenado* (ya implementada) dirá si están ordenados por su identificador de modo creciente o no.

> *Por simplicidad para esta práctica, consideraremos que los identificadores están ordenados por comparación alfabética directa, de modo que el identificador 0000ZZZ es anterior al 9999AAA.*

SCV: Registra el cambio de "Pr2.cpp" (*commit*) con el mensaje "Completada Tarea 2". Si vas a continuar trabajando no hace falta que hagas *push*.

Tarea 3

El programa realizado, mediante la función *ProbarOrdenaciones*, debe **ordenar** el conjunto de datos **según el identificador de las personas vacunadas**. Para ello hay que "habilitar" la llamada a dicha función en el programa principal (main) eliminando la marca `//` de comentarios que anteceden a su llamada. Se debe también habilitar la llamada inmediatamente anterior a la función *ResetContadores*.

La función *ProbarOrdenaciones* empieza por desordenar pseudoaleatoriamente el vector, guarda ese vector desordenado para poder utilizar el mismo como entrada para los tres algoritmos de ordenación y a continuación llama a las funciones *OrdenarInsercion*, *OrdenarSeleccion* y *OrdenarQuicksort*, y después de cada una de ellas comprueba si el vector está ordenado o no, y muestra el número de pasos contabilizado.

Comprobarás que las funciones *OrdenarInsercion*, *OrdenarSeleccion* y *OrdenarQuicksort* no ordenan porque tienen partes comentadas. Se deben quitar las marcas de comentarios `/*` y `*/` y modificar los algoritmos a fin de que las comparaciones necesarias para la ordenación del vector se hagan utilizando los **identificadores** de las distintas personas vacunadas (objetos de la clase) que forman el vector.

SCV: Registra el cambio de "Pr2.cpp" (*commit*) con el mensaje "Completada Tarea 3". Si vas a continuar trabajando no hace falta que hagas *push*.

Tarea 4

Para analizar el coste de los algoritmos de ordenación contaremos un paso por cada sentencia que realice una comparación o asignación que involucre a elementos del vector.

La función *ProbarOrdenaciones* ya muestra los pasos contabilizados por cada uno de los algoritmos, de momento 0, por lo cual tendremos que situar los incrementos de los contadores de pasos adecuados en los lugares oportunos de cada uno de los algoritmos (a modo de ejemplo, *OrdenarInsercion* ya tiene este contador correctamente actualizado).

Una vez comprobados los resultados con los datos del fichero pequeño, cambia el programa para leer el fichero grande "**vacunaciones_big.svc**" (Aula Virtual) y comenta las líneas que llamen a la función *MostrarPersonas* (emplea mucho tiempo y además es inútil mostrar datos de miles de vacunaciones).

SCV: Registra el cambio de "Pr2.cpp" (*commit*) con el mensaje "Completada Tarea 4". Si vas a continuar trabajando no hace falta que hagas *push*.

Tarea 5

Para un análisis completo del coste de los algoritmos nos interesará realizar varias ordenaciones sobre diferentes organizaciones del vector para obtener resultados promedio, y además hacerlo para diferentes tallas del problema (número de elementos que se ordenan).

Para ello, el programa solicitará una talla del problema (que obviamente no puede ser mayor que el número de personas vacunadas en el fichero) y un número de repeticiones[1] y llamaremos a la función *RealizarExperimentos* (para lo cual habilitaremos la llamada a dicha función y de paso inhabilitaremos la llamada a la función *ProbarOrdenaciones*).

Hay que completar la función *RealizarExperimentos* para repetir varias veces lo realizado en la función *ProbarOrdenaciones*: (i) desordenar el vector para ordenar -el mismo vector- con los tres algoritmos de ordenación, (ii) acumular separadamente los pasos empleados por cada uno de ellos, (iii) calcular al final el número medio de pasos empleado por cada algoritmo para la talla analizada y (iv) mostrar los resultados finales (ya implementado).

En todos los casos, para poder trabajar con diferentes tallas del problema, se ordenará el número de elementos del vector indicado como talla, que en la función es el argumento **n** (y no el número de personas vacunadas contenidas en el fichero, que siempre es el mismo).

Todo ello debe hacerse dentro de la función *RealizarExperimentos*, al principio de la cual se puede inicializar una sola vez la semilla de números aleatorios con la función *srand* (con *srand(time(NULL))*).

Queremos poder probar distintas tallas en una sola ejecución del programa, así que el programa pedirá una talla al usuario y mostrará los resultados por pantalla hasta que el usuario introduzca como valor de la talla el número 0.

Una vez realizado el programa, se probará en una sola ejecución para 50 repeticiones con la siguiente secuencia de tallas: 100, 200, 400, 800, 1600, 3200, y 0 para terminar.

SCV: Registra el cambio de "Pr2.cpp" (*commit*) con el mensaje "Completada Tarea 5" y realiza un último *push* para subir la versión final de la práctica.

[1] Cuanto mayor sea el número de repeticiones más fiabilidad en los resultados, pero valores elevados hará incrementar el tiempo de ejecución del programa (precaución!).

Tarea 5: Generación de documentación

El código realizado deberá estar correctamente documentado con Doxygen, de acuerdo con la guía de estilo. Una vez realizada la versión final del programa, se deberá ejecutar Doxygen para generar toda la documentación definitiva. La documentación deberá ser generada exclusivamente en formato en *html* (utilizar como referencia, convenientemente personalizado para esta práctica, el archivo de configuración de Doxygen disponible en el Aula Virtual). Solo la documentación deberá subirse al Aula Virtual (como archivo comprimido). **El programa NO se debe subir al AV, ya que será evaluado directamente desde el repositorio de control de versiones**.

La documentación generada con Doxygen NO debe subirse al repositorio de control de versiones Se debe mantener exclusivamente en el ordenador personal y subirlos al aula virtual.