Ejercicio para el laboratorio del 19S

Entrada/salida con archivos

Para operar con archivos en C++ se utilizan las clases `ifstream` (para lectura) y `ofstream` (para escritura) de la cabecera

```
#include <fstream>
```

Un archivo se abre al pasar su ruta como argumento al constructor del flujo de archivo o al usar el método `open`.

```
ifstream entrada("entrada.txt");
ofstream salida;  // archivo sin abrir
salida.open("salida.txt");
```

Se puede comprobar que el archivo se ha abierto correctamente con el método `is_open()` y se interactúa con él como con la entrada/salida estándar.

```
int valor;
salida << "1 + 2 = " << 1 + 2 << '\n';
entrada >> valor;
```

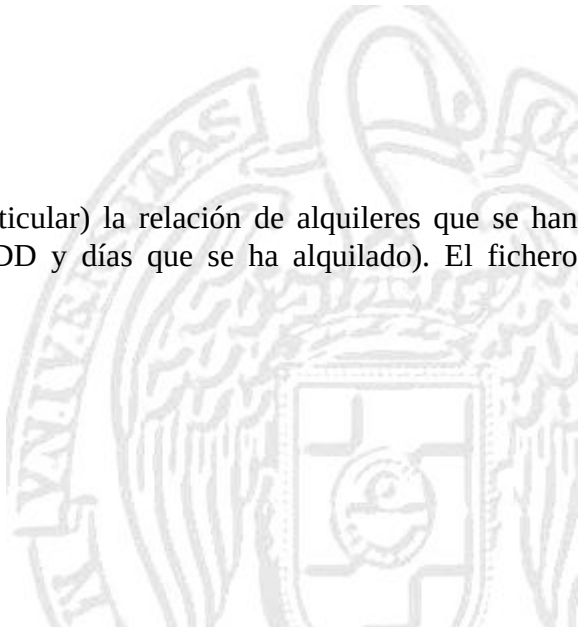
Los archivos se cierran automáticamente cuando su variable sale de ámbito, pero también se pueden cerrar anticipadamente con `.close()`.

Se pide construir un programa que muestre por orden de fecha los alquileres de una agencia de alquiler de coches. Se dispone de dos archivos, uno (`coches.txt`) con la información de códigos (enteros) y nombres (cadenas) de los coches de los que se dispone. El archivo empieza con un entero indicando el número de coches y continua con el código, precio por día y nombre de cada coche (ordenados de menor a mayor código), separados por espacios y saltos de línea cómo se muestra a continuación:

```
10
1325 30 Seat León
1327 25 Seat Arona
1329 35 Seat Toledo
...
```

El otro archivo (`rent.txt`) contiene (en ningún orden particular) la relación de alquileres que se han contratado (código del coche, fecha en formato AA/MM/DD y días que se ha alquilado). El fichero también comienza con el número de alquileres:

```
8
1722 13/03/22 1
1620 01/02/20 7
1332 21/05/21 7
1722 15/07/23 6
...
```



El programa deberá empezar cargando la información de cada archivo en las estructuras `ListaCoches` y `ListaAlquileres`, ambas formadas por un array dinámico (de elementos de tipo `Coche` y `Alquiler` respectivamente), su tamaño y un contador indicando el número real de elementos (el tamaño debe ser algo mayor que el número real de elementos, por ejemplo diez más). La lista de coches quedará ordenada por orden de códigos (como en el archivo). Una vez leídas las listas ordenará la lista de alquileres por fechas. El programa terminará mostrando la información sobre los alquileres (fecha, modelo, días alquilado y precio) como se indica:

```
01/02/15 Ford B-Max 7 día(s) por 210 euros
13/03/17 Toyota Auris 1 día(s) por 32 euros
01/01/18 ERROR: Modelo inexistente
15/07/18 Toyota Auris 6 día(s) por 192 euros
...
```

Recuerda borrar la memoria dinámica creada.

Tildes en la consola de Windows

En la consola de Windows, para que las tildes, eñes y otros caracteres no ASCII se muestren correctamente es preciso incluir la cabecera `#include <windows.h>` y llamar en el `main` a la función `SetConsoleOutputCP(CP_UTF8)`.

El programa deberá hacer uso de los siguientes subprogramas:

- `leerModelos`: carga la información del archivo `coches.txt` en la lista de coches; devuelve true si se ha podido abrir el archivo y false en caso contrario. La lista de coches sólo contendrá la información de este archivo.
- `leerAlquileres`: carga la información del archivo `rent.txt` en la lista de alquileres; devuelve true si se ha podido abrir el archivo y false en caso contrario. Cada alquiler debe incluir un campo de tipo `Coche*` con el puntero al coche al que hace referencia el alquiler (que será `nullptr` en caso de no encontrarse el código en la lista de coches). Para obtener dicho puntero deberás llamar a la función `buscarCoche` (ver abajo).
- `ordenarAlquileres`: ordena la lista de alquileres por orden de fecha (menor a mayor).

  Ordenación en la biblioteca estándar

  La cabecera `algorithm` de la biblioteca estándar incluye una función `sort` que ordena secuencias de elementos. Recibe tres parámetros: un puntero al comienzo del array, un puntero al primer elemento fuera de él y el nombre de la función que implemente el orden entre elementos. El tercer argumento es opcional y en su defecto se utilizará el operador < que esté definido sobre los elementos (si no hay ninguno dará un error de compilación). Puedes definir el operador < sobre el tipo `Alquiler` con

  ```
  bool operator<(const Alquiler& izdo, const Alquiler& dcho) {
          // Definición del orden
  }
  ```

- `buscarCoche`: dada la lista de coches y un código, devuelve la posición (índice) del elemento de la lista de coches con ese código; si no se encuentra el código devuelve -1. Debe implementarse como búsqueda binaria.
- `mostrarAlquileres`: dada la lista de alquileres, muestra la relación de alquileres con el formato mostrado arriba.

Como no hemos visto clases ni el uso de múltiples archivos, puedes implementar todo en el mismo fichero fuente .cpp. Eso sí, ten en cuenta que el compilador procesa el fichero de arriba a abajo y por lo tanto no puedes usar tipos ni funciones si no se han definido más arriba en el fichero. En el CV puedes encontrar un ejemplo de ficheros de entrada y un fichero (`salida.txt`) con la salida que tu programa debería generar para dichos ficheros de entrada. También hay incluida una clase `Date` que representa una fecha del calendario y que puedes utilizar en tu programa principal incluyéndola con `#include "Date.h"` y añadiendo `Date.cpp` a la lista a archivos a compilar en el proyecto.

Entrega: se debe realizar una entrega por grupo en que solo debéis subir vuestro fichero fuente .cpp (incluyendo vuestros nombres y el identificador del grupo, si ya lo tuviérais). La entrega se realiza en el CV, en la pestaña *Laboratorio y Prácticas*.

Ejercicios adicionales: (1) modifica la estructura (y actualiza todo lo necesario) de manera que la estructura `ListaCoches` se represente mediante un array dinámico de punteros a estructuras de tipo `Coche`. (2) Reorganiza el código usando orientación a objetos y separando la declaración de la implementación (aún no lo hemos visto, pero puedes fijarte en la clase `Date`).