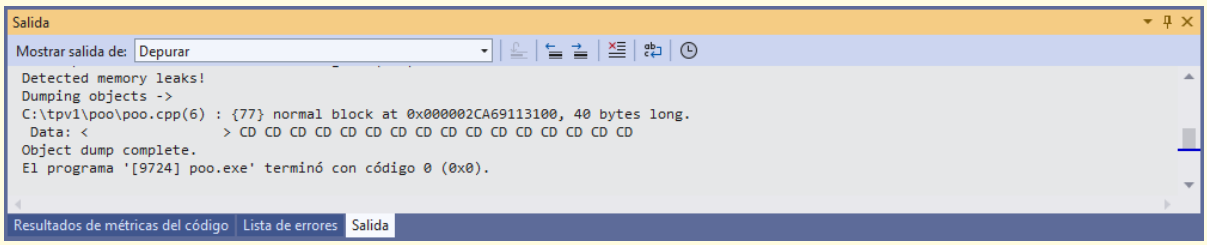
Ejercicio para el laboratorio del 3O

Se trata de rediseñar y extender el programa de hace dos semanas (gestión de alquileres de coches) haciendo uso de orientación a objetos. En particular, se pide lo siguiente:

- Reimplementa todo usando orientación a objetos y separando la declaración de la implementación. Debes implementar las clases `Coche`, `Alquiler`, `ListaCoches` y `ListaAlquileres`, cada una con atributos privados y los métodos públicos que sean necesarios para una buena encapsulación. Para la fecha debes hacer uso de la clase `Date` proporcionada (disponible en el CV) y sus operadores <<, >> y <. También puedes definir estos operadores para tus propios tipos.
- Escribe la función `main` en el fichero `main.cpp`. Esta debe primeramente cargar las listas de coches y alquileres (esta última debe quedar ordenada) y a continuación mostrar un menú con opciones para mostrar los coches, mostrar los alquileres, añadir un nuevo coche, añadir un nuevo alquiler, y salir. Al añadir un coche o alquiler la lista correspondiente debe quedar ordenada para lo cual deberá insertarse en su lugar desplazando elementos si es necesario. En caso de no haber espacio en la lista correspondiente se indicará mediante un mensaje al usuario.
- Comprueba que el programa no deje fugas de memoria.

Comprobación de fugas de memoria en Visual Studio

El archivo `checkML.h` disponible en el campus añade instrumentación a las funciones **`new`** y **`delete`** para detectar fugas de memoria al finalizar el programa. Has de incluir la cabecera al principio de cada archivo que reserve memoria dinámica (asegúrate también de que en la configuración del proyecto esté seleccionado el estándar C++17 o superior). Las fugas de memoria aparecerán en el panel de salida de Visual Studio.



Si usas `g++` o `clang++` en Linux o macOS esto no funcionará, pero basta con pasar la opción `-fsanitize=address` al compilador para que se muestren las fugas de memoria al finalizar la ejecución.

- Cambia la representación de la lista de coches de manera que se haga uso de un array dinámico de punteros a coches. Es recomendable que antes de hacer este cambio pruebes todo lo demás y dejes una versión guardada. Vuelve a comprobar que no haya fugas de memoria.
- Intenta guardar una copia de la lista de alquileres leída en el `main` antes de ordenar.

```
ListaAlquileres alquileres;
alquileres.read(archivo, coches); // o equivalente
ListaAlquileres original = alquileres;
```

¿Qué pasa al finalizar el programa?

Constructor y asignación por copia

En C++ es posible redefinir cómo se copia o asigna un objeto sobrescribiendo el constructor y el operador de asignación por copia

```
Clase(const Clase& otro) { /* rellenar aquí */ }
Clase& operator=(const Clase& otro) { /* rellenar aquí */ }
```

El constructor por copia se utiliza cuando se inicializa un objeto a partir de otro con `Clase uno(otro)` o `Clase uno = otro`. La asignación por copia se utiliza cuando se asigna un objeto a otro existente con `uno = otro`.

Redefine la copia de las clases del ejercicio para evitar los problemas observados.

- (Opcional) Implementa el soporte para que al añadir un nuevo coche o alquiler el array correspondiente se redimensione al doble de capacidad.

Entrega: se debe realizar una entrega por grupo en la que solo debéis subir vuestros ficheros de código fuente (.h y .cpp, incluyendo en el `main.cpp` vuestros nombres y el identificador del grupo). No hagáis un zip, simplemente subid los ficheros directamente. La entrega se realiza en el CV, en la pestaña *Laboratorio y Prácticas*.

Diseño de las clases: los métodos descritos en la anterior versión del ejercicio sin orientación a objetos se han de organizar ahora como métodos de alguna de las cuatro clases del enunciado, respetando el principio de encapsulación. El siguiente diagrama muestra un posible diseño de las clases, con sus atributos y métodos.