Examen final – 8/9 de junio de 2016

Tiempo disponible: 3 horas

Se pide construir un programa modular que permita generar selecciones de películas que cumplen cierto criterio a partir de una lista maestra de películas cuya información se carga de un fichero `peliculas.txt`. El programa constará de cuatro módulos: *Reparto*, *Película*, *ListaPelículas* y módulo principal (`main.cpp`).

Módulo *Reparto* (2 puntos) Máx. 10 actores

Declara un tipo de estructura `tReparto` para las listas de actores principales que intervienen en una película (como máximo 10). De cada actor se guarda solamente su nombre (una cadena de caracteres con posibles espacios).

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga un reparto. Para cada reparto, en el archivo primero aparece en una línea el número de actores que aparecerán a continuación, cada uno en una línea (ver el ejemplo de archivo al final del enunciado).
- ✓ `aparece()`: Dado un reparto, el nombre de un actor y una posición, devuelve cierto si el nombre del actor aparece en el reparto a partir de la posición dada, y falso en caso contrario. Debe hacerse de forma **recursiva**.

Módulo *Película* (1,5 puntos)

Declara un tipo de estructura `tPelicula` con 5 campos: el título, el director, el género, la valoración del público (todos de tipo cadena de caracteres, con posibles espacios, salvo valoración, de tipo real) y el reparto.

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga toda la información de una película. En el archivo cada campo ocupa una línea, salvo el reparto que puede ocupar varias, como se ha indicado arriba (ver el ejemplo de archivo al final del enunciado).
- ✓ `interviene()`: Dada una película y el nombre de un actor, devuelve cierto si el actor dado interviene en la película, y falso en otro caso.
- ✓ `mostrar()`: Dada una película muestra por pantalla: título, director, género y valoración, como aparece en el ejemplo de ejecución al final del enunciado.

Módulo *ListaPelículas* (5,5 puntos) Máx. 50 películas

Declara un tipo de estructura `tListaPeliculas` para listas de películas (hasta 50). Esta lista estará implementada con un **array estático de punteros a variables dinámicas** (es decir, un array de punteros a `tPelicula`). No tendrá ningún orden concreto, pudiéndose ordenar por diferentes criterios.

Implementa, al menos, los siguientes subprogramas:

- ✓ `cargar()`: Carga la lista de películas del archivo `peliculas.txt`. El fichero comienza con el número de películas que contiene (en una línea), y a continuación aparece la información de cada una de ellas.
- ✓ `filtrarPorActor()`: Dada una lista de películas y un actor, genera una nueva lista que contiene únicamente enlaces a las películas donde ha intervenido ese actor (no se crean variables dinámicas nuevas, las películas son compartidas). La lista original no se modifica.
- ✓ `mostrar()`: Dada una lista de películas, muestra la información de cada una de las películas, separadas por tres guiones, como aparece en el ejemplo al final del enunciado.
- ✓ `ordenarPorGenero()`: Dada una lista de películas la ordena por género, de menor a mayor.
- ✓ `ordenarPorValoracion()`: Dada una lista de películas la ordena por valoración, de mayor a menor. Debe implementarse una **ordenación por inserción**.
- ✓ `destruir()`: Dada una lista de películas, libera la memoria dinámica que ha demandado la lista.

Módulo principal (1 punto)

El programa principal carga las películas del archivo `peliculas.txt` en una lista de películas, solicita al usuario un nombre de actor, crea a partir de la lista maestra una nueva lista con las películas donde ha intervenido ese actor y la muestra ordenada por géneros, y dentro de cada género, las películas aparecen ordenadas por valoración. Al salir se deberá liberar toda la memoria dinámica utilizada.

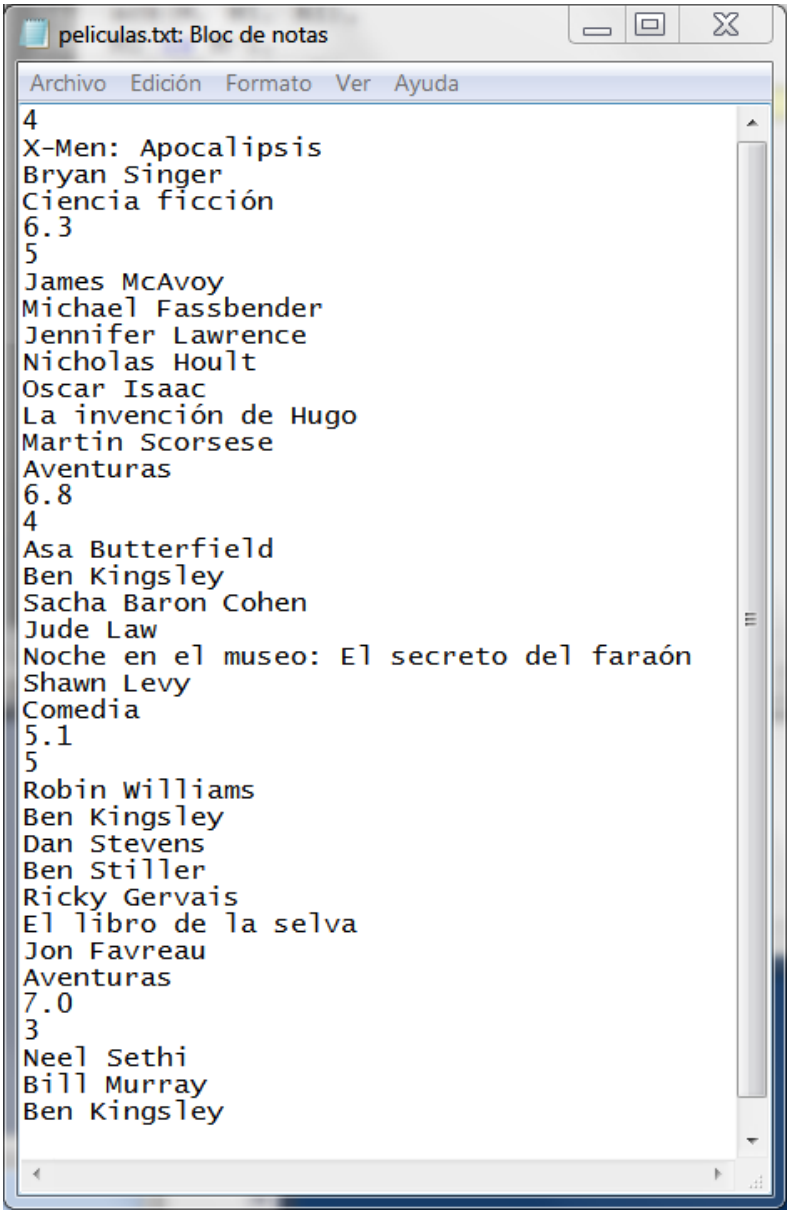
Se valorará la legibilidad, así como el uso adecuado de los esquemas de recorrido y búsqueda, de la comunicación entre subprogramas y de la memoria.

Recuerda: El comando para que se muestre la memoria no liberada es

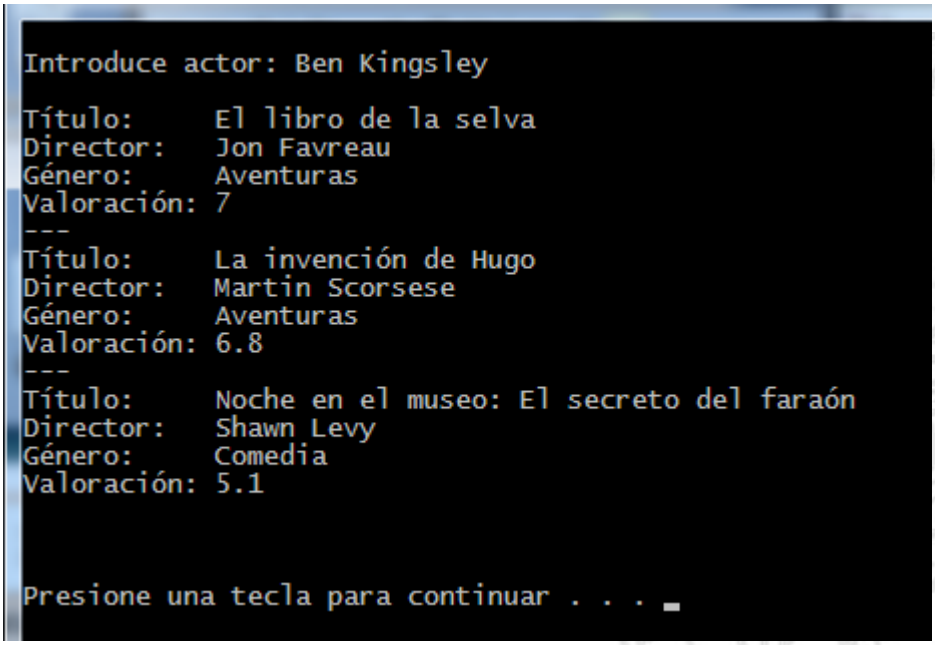
```
_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

Entrega el código del programa a través del Campus Virtual (solo .cpp y .h, comprimidos en un ZIP). ¡Asegúrate de entregar una versión sin errores de compilación!

Ejemplo de archivo `peliculas.txt`:



Ejemplo de ejecución:



Archivo checkML.h

```
#ifdef _DEBUG
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif
```