

Examen final – 8/9 de junio de 2016

Tiempo disponible: 3 horas

Se pide construir un programa modular que permita elaborar listas de reproducción de temas musicales a partir de una lista maestra de temas. El programa constará de cuatro módulos: *Tema*, *ListaTemas*, *ListaReproducción* y módulo principal (`main.cpp`).

Módulo *Tema* (0.75 puntos)

Declara un tipo de estructura `tTema` con los campos: título, intérprete y segundos (dos cadenas de caracteres con posibles espacios y un entero).

Implementa, al menos, las siguientes funciones:

- ✓ `cargar()`: Carga un tema. En el archivo cada tema consta de 3 líneas: título, intérprete y segundos (ver ejemplo de archivo al final del enunciado).
- ✓ `mostrar()`: Dado un tema lo muestra en una línea, como aparece en el ejemplo al final del enunciado.

Módulo *ListaTemas* (1.75 puntos)

Máx. 50 temas

Declara un tipo de estructura `tListaTemas` para la lista de temas disponibles (hasta 50). Esta lista estará implementada con un **array estático de punteros a variables dinámicas**.

Implementa, al menos, las siguientes funciones:

- ✓ `cargar()`: Carga la lista de temas del archivo `temas.txt`. El fichero comienza con el número de temas que contiene (en una línea), y a continuación aparece la información de cada uno de ellos.
- ✓ `destruir()`: Dada una lista de temas, la libera la memoria dinámica que utiliza.

Módulo *ListaReproducción* (5 puntos)

Define un tipo de estructura `tElemento` con un puntero a un tema (de la lista de temas) y una valoración (un entero del 0 al 10) sobre su pertenencia a la lista de reproducción. Declara un tipo de estructura `tListaReproduccion` para listas de `tElemento` implementadas con **array dinámico**. Además guardará el nombre de la lista de reproducción (una cadena de caracteres con posibles espacios).

Implementa, al menos, las siguientes funciones:

- ✓ `nueva()`: Dado un nombre y una capacidad (`dim`), devuelve una lista de reproducción vacía, adecuadamente inicializada para poder contener `dim` elementos.
- ✓ `insertar()`: Dada una lista y un elemento lo inserta al final de la lista (no es necesario redimensionar la lista).
- ✓ `buscar()`: Dada una lista de reproducción, el título de un tema y una posición, determina si el tema se encuentra en la lista a partir de la posición dada. Debe implementarse de forma **recursiva**.
- ✓ `mostrar()`: Dada una lista muestra por pantalla su nombre, los temas, numerados en el orden en que se encuentren, y el tiempo total de reproducción (en formato mm:ss). Sigue el formato del ejemplo al final del enunciado.
- ✓ `modificarOrden()`: Dada una lista y dos posiciones, una origen y otra destino, modifica las posiciones de los elementos de la lista, de forma que el elemento de la posición origen debe quedar en la posición destino, sin modificar el orden relativo de los demás elementos. Para ello, desplaza a la izquierda o derecha, según corresponda, los elementos de la lista para hacer hueco en la posición destino al elemento de la posición origen. Ver ejemplo de ejecución al final del enunciado.
- ✓ `destruir()`: Dada una lista de reproducción, libera la memoria dinámica que utiliza.

Módulo principal (2.5 puntos)

Carga los temas musicales del archivo `temas.txt` en una lista de temas (lista maestra), solicita al usuario un nombre y un número de temas `numTemas`, crea una nueva lista de reproducción con ese nombre y `numTemas` temas distintos, elegidos aleatoriamente de la lista maestra (la valoración también se decide de forma aleatoria), y realiza las siguientes modificaciones en el orden (mostrando la lista para comprobar las modificaciones): mover el último tema al primero, mover el segundo al último y mover el segundo al cuarto.

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.

Nota: Al inicio del `main`, pon la instrucción `srand(1)` para que al hacer las pruebas siempre se obtenga la misma lista de reproducción aleatoria.

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** (sólo `.cpp` y `.h`, comprimidos en un ZIP). ¡Asegúrate de entregar una versión sin errores de compilación!

Ejemplo de archivo temas.txt:

```
Nombre: examen
Número de temas: 5

Lista: examen
1- titulo2 interprete2      274 seg. (7)
2- titulo5 interprete5      60 seg. (0)
3- titulo4 interprete4      60 seg. (8)
4- titulo3 interprete3     180 seg. (4)
5- titulo1 interprete1     240 seg. (5)
Tiempo de reproducción: 13:34

Cambio de orden: último al primero

Lista: examen
1- titulo1 interprete1     240 seg. (5)
2- titulo2 interprete2     274 seg. (7)
3- titulo5 interprete5      60 seg. (0)
4- titulo4 interprete4      60 seg. (8)
5- titulo3 interprete3     180 seg. (4)
Tiempo de reproducción: 13:34

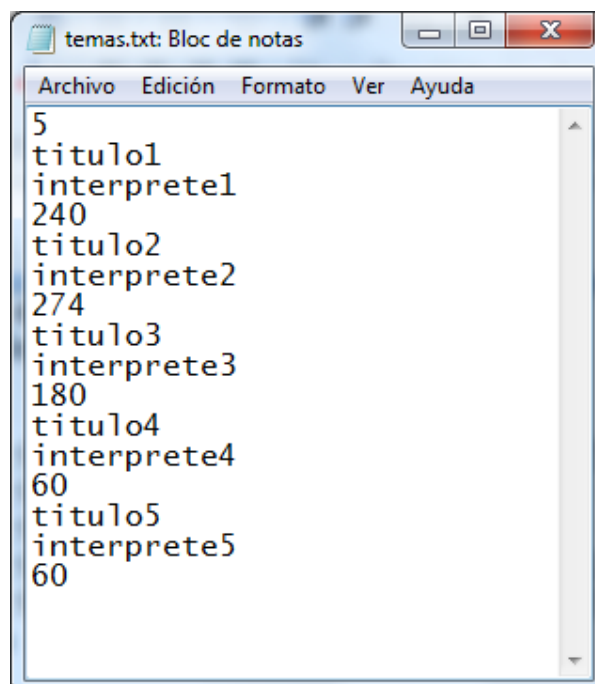
Cambio de orden: segundo al último

Lista: examen
1- titulo1 interprete1     240 seg. (5)
2- titulo5 interprete5      60 seg. (0)
3- titulo4 interprete4      60 seg. (8)
4- titulo3 interprete3     180 seg. (4)
5- titulo2 interprete2     274 seg. (7)
Tiempo de reproducción: 13:34

Cambio de orden: segundo al cuarto

Lista: examen
1- titulo1 interprete1     240 seg. (5)
2- titulo5 interprete5      60 seg. (0)
3- titulo3 interprete3     180 seg. (4)
4- titulo2 interprete2     274 seg. (7)
5- titulo4 interprete4      60 seg. (8)
Tiempo de reproducción: 13:34
```

Ejemplo de ejecución:



```
temas.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
5
titulo1
interprete1
240
titulo2
interprete2
274
titulo3
interprete3
180
titulo4
interprete4
60
titulo5
interprete5
60
```

Archivo checkML.h

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

