Estructuras de Datos y sus Aplicaciones

Proyecto Práctico: Implementación de Playlist Musical (Listas Dobles)

Contexto del Problema

Las aplicaciones de reproducción de música (como Spotify, Apple Music o Winamp) utilizan estructuras de datos dinámicas para gestionar la cola de reproducción. La característica fundamental de estas aplicaciones es la capacidad del usuario para navegar en dos direcciones:

- **Siguiente:** Ir a la canción futura.
- **Anterior:** Regresar a la canción que acabamos de escuchar.

Debido a esta bidireccionalidad, la estructura de datos ideal para modelar este comportamiento es una **Lista Doblemente Ligada**.

Objetivo

Implementar una aplicación de consola en C++ que simule el comportamiento de un reproductor de música utilizando una lista doblemente ligada creada desde cero (sin utilizar `std::list`).

Requerimientos Técnicos

1. Estructura del Nodo (Canción)

Cada nodo de la lista representará una canción y debe contener al menos la siguiente información:

```
struct Cancion {
    string titulo;
    string artista;
    int duracionSegundos; // Opcional: Para mostrar formato min:seg

    // Punteros vitales para la lista doble
    Cancion* siguiente;
    Cancion* anterior;
};
```

2. Funcionalidades del Reproductor

El programa debe mostrar un menú interactivo con las siguientes opciones:

1. **Agregar Canción:** Solicita título y artista. La canción debe agregarse siempre al **final** de la lista.
2. **Mostrar Playlist Completa:** Imprime la lista de todas las canciones en orden (de la primera a la última).

3. **Reproducir Actual:** Muestra los detalles de la canción a la que apunta actualmente el puntero del reproductor.

4. **Siguiente (Next):** Mueve el puntero de reproducción al siguiente nodo.
   - *Validación:* Si está en la última canción, debe indicar "Fin de la playlist".

Anterior (« Prev): Mueve el puntero de reproducción al nodo anterior.

»■ ***Validación:* Si está en la primera canción, debe indicar Ïnicio de la playlist".**

»**Eliminar Canción Actual: Elimina de la lista la canción que se está reproduciendo actualmente. El puntero de reproducción debe moverse inteligentemente a la siguiente canción (o a la anterior si no hay siguiente).**

»**Ordenar por Artista (Reto Opcional): Reordena los nodos alfabéticamente por nombre de artista (Bubble Sort o Insertion Sort adaptado a punteros).**

Consideraciones de Implementación

- **Punteros Globales/Clase: Deberá manejar al menos tres punteros principales:**
  - `Head`**: Para saber dónde inicia la lista.**
  - `Tail` **(Opcional pero recomendado): Para insertar al final en** $O(1)$**.**
  - `Reproductor`**: Un puntero auxiliar que indica qué canción se está .**[es]**cuchando.**[en] **ese momento.**
- **Gestión de Memoria: Recuerde usar** `delete` **al eliminar una canción para evitar fugas de memoria (*Memory Leaks*).**

Casos de Prueba (Escenarios a verificar)

1. **Inserción Vacía: Agregar la primera canción. Verificar que** `Head`**,** `Tail` **y** `Reproductor` **apunten a este único nodo.**
2. **Navegación: Agregar 3 canciones (A, B, C).**
   - **Estando en A, presionar .**[A]**nterior"(Debe avisar límite).**
   - **Estando en C, presionar "Siguiente"(Debe avisar límite).**
   - **Ir de A** $\rightarrow$ **B** $\rightarrow$ **C** $\rightarrow$ **B** $\rightarrow$ **A.**
3. **Eliminación Crítica:**
   - **Eliminar la primera canción (A). El** `Head` **debe actualizarse a B.**
   - **Eliminar la última canción (C). El** `Tail` **debe actualizarse a B.**
   - **Eliminar la única canción de la lista. Todos los punteros deben quedar en** `NULL`**.**

Entregable

Subir un archivo comprimido `.zip` conteniendo:

1. **Código fuente (`.cpp`).**
2. **Capturas de pantalla mostrando la funcionalidad de Siguiente, Anterior y Eliminar.**