Assignment 1

You have recently been hired as a backend systems engineer for a startup developing "RetroPlayer," a lightweight console-based music player for desktop applications. Because the application is designed to use minimal system resources, the engineering team cannot rely on bloated, inefficient, or standard automated frameworks.
Efficient applications are built by carefully analyzing the problem and choosing the most suitable data structure for the required operations. The system must manage a dynamic playlist of audio tracks that grows and shrinks dynamically as the user interacts with it.

The Problem
The user interface team has handed you the functional requirements. The playlist size is never fixed, and users demand the ability to insert or remove tracks at any position on the fly. Furthermore, users need to seamlessly jump to the next or previous track while listening.
Your task is to design the internal data structure that efficiently supports these operations without causing memory leaks or lag.

Part 1: Architectural Design & Defense
Before writing any code, you must critically analyze the problem.

1. **Select the Data Structure:** Choose the most appropriate data structure to implement the playlist. (Hint: Consider how the requirements for dynamic sizing, middle-insertions, and bidirectional navigation impact your choice).

2. **Justification:** Write a brief paragraph explaining *why* you chose this data structure over an array or a singly linked list.

Part 2: Backend Implementation
Implement your chosen data structure using C++. You are encouraged to reuse the header (.h) files you created during your laboratory sessions.
Track Metadata: Each node (track) in your structure must store the following data:

- Track ID
- Title
- Artist
- Duration (in seconds)

Required System Operations: Your system must successfully execute the following commands:

- **Add an audio track:** At the beginning, at the end, or at a specific target position.
- **Remove a track:** Delete a track safely using its Track ID.
- **Search:** Find and display a track by its title.
- **Display:** View the entire playlist in its current order.
- **Navigate:** Move to the next track or return to the previous track.
- **Metrics:** Count and display the total number of tracks currently in the playlist.

Part 3: Bringing it to Life (Audio Integration)

To turn your data structure into a real music player, you will integrate the **SFML (Simple and Fast Multimedia Library)** for audio playback.

Setup Instructions:

1. Download and install SFML via the official getting-started guide for Visual Studio:
   https://www.sfml-dev.org/download/sfml/3.0.2/

   https://www.sfml-dev.org/tutorials/3.0/getting-started/visual-studio/#creating-and-configuring-an-sfml-project

2. Ensure you have a .mp3 or .wav file in your project directory named song.mp3.
   SFML Test Code: Use the following starter code to verify your configuration before integrating it into your playlist structure:

```
#include <SFML/Audio.hpp>

int main()
{
    sf::Music music;

    if (!music.openFromFile("song.mp3"))
        return -1;

    music.play();

    while (music.getStatus() == sf::SoundSource::Status::Playing)
    {
        // wait while music plays
    }

    return 0;

}
```

Rubrics

Rubric Category	Weightage
Data Structure Utilization & Implementation	30%
Object-Oriented Design (OOP)	20%
Audio Library Integration (SFML)	15%
Documentation & Defense	15%
Robustness & Error Handling	10%
User Interface & Console Experience	10%
Total	**100%**