N.B.K.R.INSTITUTE OF SCIENCE AND TECHNOLOGY

MUSIC PLAYLIST MANAGER

COURSE : Computer science

SECTION: E

YEAR: Ⅰ

SEMISTER: Ⅱ

DATE: MAY-05-2025

SUBMITTED TO MR. ASHOK SELVA KUMAR

SUBMITTED BY:

V.TARUNI (24KB1A05MZ)

S.HITHAISHI (24KB1A05HP)

S.YASMIN (24KB1A05JT)

T.RUPARANI (24KB1A05LC)

ACKNOWLEDGEMENT:

I would like to express my sincere gratitude to everyone who supported and guided me throughout the development of this Music Playlist Manager project. This project, which implements core functionalities such as adding, removing, and displaying songs using a linked list, was a valuable learning experience in data structure application and practical programming.

This project not only deepened my understanding of linked lists in data structures but also improved my skills in algorithm design and problem-solving.

ABSTRACT:

This project presents a Music Playlist Manager application that allows users to manage a dynamic playlist using a linked list data structure. The core functionalities of the system include adding new songs, removing existing songs, and displaying the entire playlist. Each song is represented as a node in the linked list, storing essential information such as the song title .

The use of a linked list enables efficient insertion and deletion operations, particularly suitable for dynamic data management where the size of the playlist may frequently change. This implementation demonstrates practical usage of singly linked lists and strengthens the understanding of data structure concepts in real-world applications.

The project is implemented in a console-based interface, ensuring simplicity and clarity in functionality. It serves as a foundational exercise in both programming logic and the application of data structures in managing collections of data.

INTRODUCTION

In the digital era, managing and organizing music efficiently is a key feature of any audio playback system. This project, titled Music Playlist Manager, focuses on building a basic music playlist application that allows users to add, remove, and view songs using a linked list data structure.

A linked list is a linear data structure where elements are stored in nodes, and each node points to the next. This structure is particularly useful for applications where frequent addition and deletion of elements are required, as it provides flexibility and dynamic memory allocation.

The main aim of this project is to demonstrate the practical application of data structures—specifically linked lists—in software development, enhancing both theoretical knowledge and programming skills. It provides a hands-on approach to understanding node manipulation, memory handling, and traversal techniques in linked list-based systems.

OBJECTIVE:

The primary objective of this project is to design and implement a Music Playlist Manager using a linked list data structure. The system aims to provide the following key functionalities:

Add Songs: Allow users to insert new songs into the playlist dynamically.

Remove Songs: Enable users to delete specific songs from the playlist.

Display Playlist: Traverse and display all songs currently in the playlist in sequential order.

SYSTEM REQUIREMENTS:

To develop and run the Music Playlist Manager application, the following system requirements are recommended

1. Hardware Requirements:

Processor: Intel Core i3 or higher

RAM: Minimum RAM

Storage: At least 100 MB of free disk space

Display: Standard resolution (1024x768) or higher

2. Software Requirements:

Operating System: Windows 10/11, Linux.

Compiler : TURBO C++,CODE BLOCKS.

For C: CodeBlocks/Dev C++

Text Editor: Notepad++, Sublime Text, or any preferred editor

Console/Terminal: Command Prompt, PowerShell, Terminal (Linux/macOS)

3. Programming Language:

C/DATA STRUCTURES…

METHODOLOGY:

The Music Playlist Manager is developed using a structured and modular programming approach, with the core logic centered around the implementation of a singly linked list. The methodology followed in this project consists of the following stages:

1. Problem Analysis:

The requirements were analyzed to identify key operations: adding songs, removing songs, and displaying the playlist. These operations were mapped to linked list functions such as insertion, deletion, and traversal.

2. Design Phase:

A node structure was designed to represent each song, containing fields for song title, artist name, and a pointer to the next node.

3. Implementation Phase:

The application was implemented in using a singly linked list. Key functions include:

addSong(): Creates a new node and inserts it at the end or specified position.

removeSong(): Searches and deletes a song node based on title or position.

displayPlaylist(): Traverses the list and prints details of each song.

4. Testing and Debugging:

Multiple test cases were run to verify the correctness of each function, ensuring proper handling of edge cases like empty lists, duplicate songs, and invalid deletions.

5. User Interaction and Input Validatio:

A menu-driven program was developed for ease of use, with proper prompts, confirmations, and error handling.

PROJECT DISOPTION:

The Music Playlist Manager project successfully demonstrates the use of a singly linked list to manage a dynamic collection of songs. By simulating real-world playlist operations such as adding, removing, and displaying songs, the project highlights the strengths and limitations of using linked lists in application development

However, some limitations were also noted. Searching for a specific song in a singly linked list requires linear traversal, which can be time-consuming with large playlists. Moreover, removing a node requires keeping track of previous nodes, making the deletion process slightly more complex than in other data structures like doubly linked lists.

Throughout the development process, careful consideration was given to edge cases, such as trying to delete from an empty playlist or handling invalid user input. The program structure was kept modular to ensure maintainability and ease of debugging.

Overall, the project provided valuable insights into how data structures like linked lists can be applied to real-world problems. It also reinforced the importance of designing efficient algorithms and user-friendly interfaces in software development.

ALGORITHM:

Step 1: Start the program

Step 2: Initialize the head of the linked list to NULL

Step 3: Display a menu with options:

1. Add Song

2. Remove Song

3. Show Playlist

4. Exit

Step 4:

Accept user's choicStep 5: Based on choice:

If 1 (Add Song):

Create a new node

Input song title and artist

Insert the node at the end of the list

If 2 (Remove Song):

Input song title (or position)

Search the list

If found, delete the node

Else, display "Song not found"

If 3 (Show Playlist):

Traverse the linked list

Display each song's title and artist

If 4 (Exit):

Terminate the program

Step 6: Repeat steps 3–5 until user chooses Exit

Step 7: End.

PROGRAM CODE:

"https://onlinegdb.com/sX\_SXrloR"

OUTPUT:

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Show All Songs

4. Exit

Enter your choice: 1

Enter song title to add: Shape of You

Added: Shape of You

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Show All Songs

4. Exit

Enter your choice: 3

Playlist:

- Shape of You

- Blinding Lights

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Show All Songs

4. Exit

Enter your choice: 2

Enter song title to remove: Shape of You

Removed: Shape of You

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Show All Songs

4. Exit

Enter your choice: 3

Playlist:

- Blinding Lights

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Show All Songs

4. Exit

Enter your choice: 4

Exiting... `

TESTING AND VALIDATION:

Validation Methods:

Unit Testing: Each function (add, remove, display) was tested individually to verify that it performs as expected.

Boundary Testing: Tested behavior when the playlist is empty or full of entries.

Invalid Input Handling: Verified system's response to unexpected or invalid inputs, such as deleting a song from an empty list or choosing an invalid menu option.

Memory Management : Ensured that dynamically allocated memory is properly released upon node deletion.

Here’s a well-defined Limitations section for your Music Playlist Manager using linked list implementation:

LIMITATIONS:

While the Music Playlist Manager effectively demonstrates the use of a linked list for managing a playlist, it has several limitations:

1. Linear Search Time:

Searching for a song requires linear traversal through the linked list, which can become inefficient as the playlist grows larger.

2. No Sorting or Organization:

Songs are stored in the order they are added, with no built-in sorting by title, artist, or other criteria, making navigation less user-friendly.

3. Limited User Interface:

The application is console-based with a text interface, which may not be as intuitive or visually appealing as a graphical user interface (GUI).

4. Single Linked List Usage:

Since a singly linked list is used, operations like removing the last node require traversal from the beginning, increasing complexity. Doubly linked lists could provide more efficient operations in some cases.

5.Basic Input Handling:

Input validation is minimal. For instance, the system doesn't prevent adding duplicate songs or invalid characters unless explicitly coded.

6. No Persistent Storage:

The playlist exists only during the runtime of the program. Once the program is closed, all data is lost as there is no file or database integration.

7. Limited Features:

Advanced features found in modern music apps (like search filters, shuffle, repeat, duration display, or playback) are not implemented.

FUTURE ENHANCEMENT:

To improve the functionality and user experience of the Music Playlist Manager, the following enhancements can be considered for future development:

1. Graphical User Interface (GUI):

Integrate a GUI using tools like Tkinter (Python), JavaFX (Java), or Qt (C++) to provide a more user-friendly and interactive experience.

2. Persistent Storage:

Add file handling or database support to save the playlist permanently, allowing users to retrieve their songs even after closing the program.

3. Search Functionality:

Implement search features to find songs by title, artist, or keywords, reducing the time to locate specific entries.

4. Sorting Options:

Allow users to sort songs alphabetically by title or artist, or by the order of addition, for better playlist organization.

5. Use of Doubly Linked List:

Upgrade the data structure to a doubly linked list to allow easier backward traversal and more efficient deletion from both ends.

6. Playlist Management:

Include features to create, delete, and switch between multiple playlists.

7. Duplicate Check:

Implement checks to prevent the addition of duplicate songs based on title and artist

8. Playback Simulation:

Add simulated playback controls like play, pause, next, and previous to mimic the behavior of an actual music playlist.

CONCLUSION:

The Music Playlist Manager project effectively demonstrates the practical application of singly linked lists in managing dynamic data structures. By implementing core functionalities such as adding, removing, and displaying songs, the project illustrates how linked lists can be used to build flexible and efficient systems.

Through this project, key programming concepts such as dynamic memory allocation, pointer manipulation, and traversal algorithms were reinforced. While the application is basic and text-based, it provides a solid foundation for understanding the structure and behavior of linked lists in real-world scenarios.

Despite some limitations—such as the lack of persistent storage and graphical interface—the project successfully achieves its objectives and opens the door for future enhancements, including GUI integration, advanced search, and persistent data handling.

Overall, the project has been a valuable learning experience in both software development and data structure implementation.

REFERENCE:

Programming Books & Textbooks:

Balagurusamy, E. Programming in ANSI C, McGraw-Hill Education.

Lafore, R. Data Structures and Algorithms in C++, Sams Publishing.

Horowitz, E., Sahni, S. Fundamentals of Data Structures in C, Universities Press.

Development Tools & Environments:

GCC Compiler / Code::Blocks / Dev C++ / Visual Studio Code