Project Title : Music Playlist Manager

❖ Table of Content:

1.Introduction:

In today's digital age, music applications rely heavily on efficient data structures to manage and organize playlists. A "**Music Playlist Manager"** is a software system that allows users to create, modify, and interact with music playlists. In this project, we implement both singly and doubly linked lists to manage songs efficiently.

---

2. 👥 Team Members:

S.NO	TEAM MEMBERS	ROLL NO	RESPONSIBILITY
1	P.Ajay	2403031461725	Project Lead&Documentation
2	K.Vishal	2403031460833	Algorithms&Testing,Debugging
3	M.Shashanksai	2403031460938	System&Future Enhancement
4	K.Siddu Kumar	2403031460216	Module Development(Add/Delete)
5	I.Premsai	2403031461197	Data Structures

---

3. 📌 Objectives

- To implement a **playlist manager** using **linked lists**.
- To understand the difference between **singly** and **doubly** linked lists.
- To perform operations such as:
- Add song or Delete song
- Display playlist & Play next/previous song

- To strengthen the concepts of **dynamic memory allocation** and **pointers** in C.

4. ❓ Problem Statement!

Managing large playlists manually can be cumbersome. This project provides a simple interface to organize music playlists efficiently using basic data structures.Design and implement a music playlist manager in C that allows users to store and manage songs using both singly and doubly linked lists. The program should let users add new songs, delete songs, search for a song, and display the playlist in order. Additionally, provide functionality to navigate forward and backward through the playlist when using the doubly linked list. Ensure that memory management is handled properly when adding or removing songs. Finally, compare the advantages of singly and doubly linked lists in the context.

---

5. 💻 System Requirements

Software:

- C Compiler,Bytexl
- Text Editor
- (DEV C++, VS CODE)
- Windows/Linux/Mac OS

Hardware:

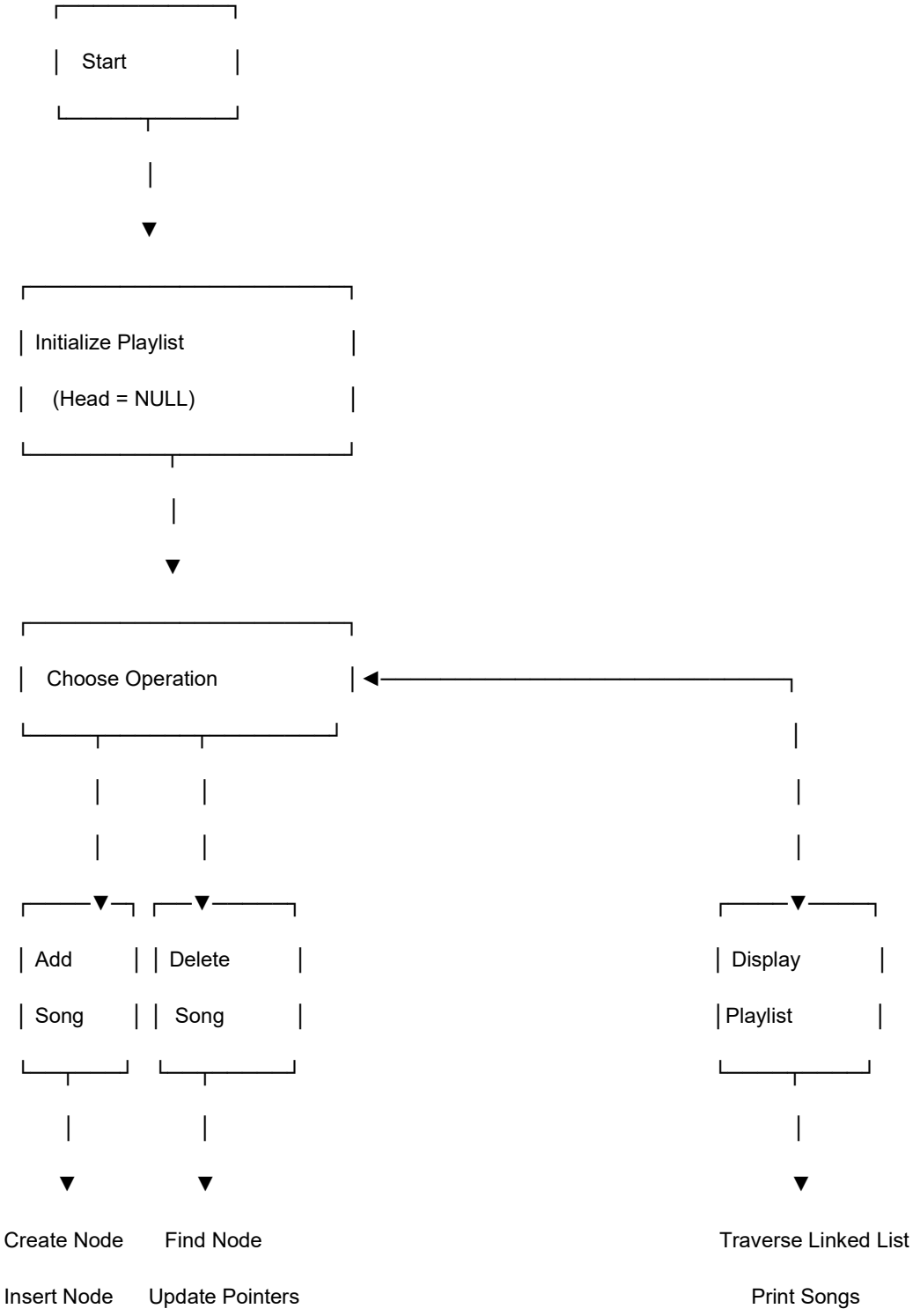
- RAM (1GB or More Required)
- Storage At least 10 MB of free disk space

---

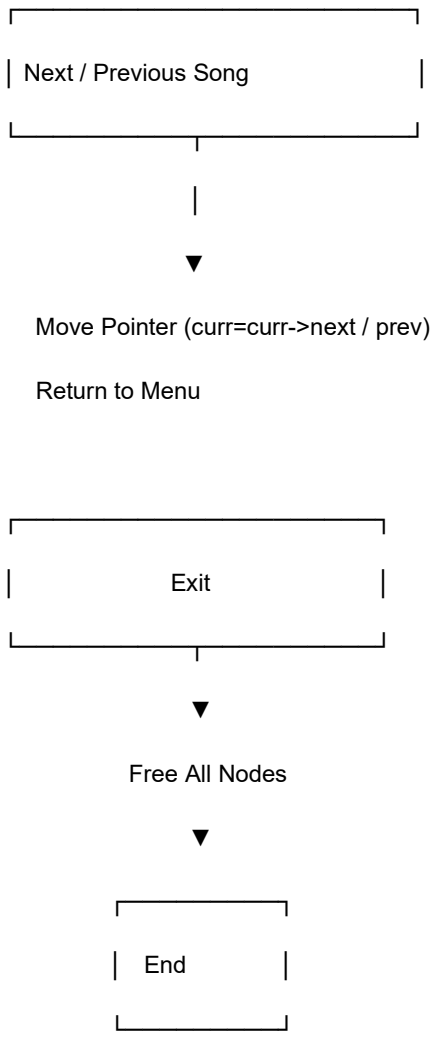
6. 🗃️ Data Structures Used

- Struct for representing a song (title, artist, duration)
- Single Linked List
- Double Linked List

FLOW CHART OF THE CODE:



Return to Menu Free Memory Return to Menu



7. Modules and Description

MODULES	DESCRIPTION
Add Song	Input song details and add to playlist
Delete Song	Remove a song by title
Display Playlist	Show all songs
Search Song	Find song by title
Sort Playlist	Sort songs by title or artist

---

8. ❁ Algorithms Used

- **Traversal**: Iterates through the linked list to display, search, or save songs
- **Insertion**: Adds a new song at the end of the list using iterative traversal
- **Deletion**: Searches for a song by title and adjusts pointers to remove the node
- **Linear Search**: Compares each node's title with the input to find a match

Create a New Song Node

- Allocate memory for a new node.
- Store song details in the node.
- Set pointers (next, prev) appropriately.

Add Song at End (Doubly Linked List)

- Create a new node.
- If list is empty, set head = newNode.

- Else, traverse to the last node.
- Link new node at the end and update prev.

Delete a Song

- Traverse list until song title is found.
- Adjust previous and next pointers.
- Free memory of deleted node.

Display Playlist

- Traverse from head to NULL.
- Print song details.
- For reverse display (DLL), traverse back using prev.

---

9. Sample I/O

```
Welcome to Playlist Manager
1. Add Song
2. Delete Song
3. Display Playlist
4. Search Song
5. Sort Playlist
6. Exit
```

```
Enter choice: 1

Enter song title: Summertime Sadness
Enter artist name: Lana Del Rey
Enter duration (seconds): 4:25
Song added successfully!

Enter choice: 3
Playlist:
1. Summertime Sadness - Lana Del Rey [4:25]
```

10. ☑ Test Cases

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for each song (node)
typedef struct Song {
    char name[50];
    struct Song *next;
} Song;

Song *head = NULL;  // Playlist head
// Function to add song at end
void addSong(char name[]) {
    Song *newSong = (Song *)malloc(sizeof(Song));
    strcpy(newSong->name, name);
    newSong->next = NULL;

    if (head == NULL) {
        head = newSong;
    } else {
        Song *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newSong;
    }
    printf("✅ Song '%s' added to playlist.\n", name);
}

// Function to display playlist
void displayPlaylist() {
    if (head == NULL) {
        printf("📂 Playlist is empty.\n");
        return;
    }
```

```
    Song *temp = head;
    printf("\n🎶 Current Playlist:\n");
    while (temp != NULL) {
        printf(" - %s\n", temp->name);
        temp = temp->next;
    }
}
// Function to delete a song by name
void deleteSong(char name[]) {
    if (head == NULL) {
        printf("⚠️ Playlist is empty. Nothing to delete.\n");
        return;
    }
    Song *temp = head, *prev = NULL;
    // If head node itself holds the song
    if (strcmp(temp->name, name) == 0) {
        head = temp->next;
        free(temp);
        printf("🗑️ Song '%s' deleted.\n", name);
        return;
    }

    // Search for the song
    while (temp != NULL && strcmp(temp->name, name) != 0) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("⚠️ Song '%s' not found in playlist.\n", name);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("🗑️ Song '%s' deleted.\n", name);
}

// Main function (Menu driven)
int main() {
```

```
int choice;
char songName[50];

do {
    printf("\n==== Music Playlist Menu ====\n");
    printf("1. Add Song\n");
    printf("2. Display Playlist\n");
    printf("3. Delete Song\n");
    printf("4. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);
    getchar(); // clear newline from buffer

    switch (choice) {
        case 1:
            printf("Enter song name: ");
            fgets(songName, sizeof(songName), stdin);
            songName[strcspn(songName, "\n")] = 0; // remove newline
            addSong(songName);
            break;

        case 2:
            displayPlaylist();
            break;

        case 3:
            printf("Enter song name to delete: ");
            fgets(songName, sizeof(songName), stdin);
            songName[strcspn(songName, "\n")] = 0;
            deleteSong(songName);
            break;

        case 4:
            printf("👋 Exiting... Goodbye!\n");
            break;

        default:
            printf("⚠️ Invalid choice.\n");
    }
} while (choice != 4);
```

```
return 0;
 }
```

---

11. 🔧 Future Enhancements

- Implement **shuffle and repeat modes** for dynamic playback.
- Provide **playlist saving and loading** from files for persistence.
- Add a **graphical user interface (GUI)** for better user interaction.
- Extend to **multiple playlists management** with import/export options.
- Support **sorting** (alphabetically, by duration, or by most played).

---

12. 🎯 Conclusion

The ***Music Playlist Manager*** project successfully demonstrates the use of **singly and doubly linked lists** to manage a dynamic collection of songs. It provides core playlist features such as adding, deleting, searching, and traversing songs. This project not only enhances understanding of **linked list data structures** but also bridges the gap between theory and practical implementation.

---

13. 🌐 References

- Tutorials From Trainer(Mr. Jashwanth Sir).
- Online resources for Better Implementation (ChatGpt).