NATIONAL INSTITUTE OF TECHONOLOGY ROURKELA

PROJECT REPORT
ON
Music Playlist Manager

Under the guidance of

Prof.Bibhudatta Sahoo,

HOD (Computer Science and Engineering)

Introduction:

The evolution of digital music platforms has brought about a need for efficient music playlist management systems. Managing a music playlist involves adding new songs, removing unwanted tracks, shuffling, and organizing songs in various ways. This project addresses these needs by implementing a **Music Playlist Manager** in **C++**, utilizing a **singly linked list** as the core data structure.

The program is a **console-based menu-driven application** that offers a variety of features to manage a playlist effectively. It provides a practical example of how linked lists can be used to store and manipulate sequential data dynamically.

The project combines theoretical knowledge of linked lists with practical programming skills, offering users a lightweight, functional tool for playlist management.

Features and Functionalities:

The **Music Playlist Manager** supports a comprehensive range of features to make playlist management seamless for users. These features include:

1 Add a Song

- Allows users to add a song to the playlist by specifying its title, artist, and duration.
- Each song is represented as a Song object, which is stored in a linked list node.
- The new song is appended to the end of the playlist.

2 Remove a Song

- Enables users to delete a song by its title.
- If the specified song does not exist in the playlist, the program notifies the user.

3 Display Playlist

- Lists all songs in the playlist along with their respective details (title, artist, and duration).

- Displays the total duration of all songs combined.
- Example output:

4 Reverse Playlist

- Reverses the order of songs in the playlist by re-linking the nodes.
- Updates the playlist structure dynamically.

5 Shuffle Playlist

- Rearranges the songs in a random order.
- The implementation uses a vector to hold the nodes temporarily for efficient shuffling.

6 Search Song

- Allows users to search for a song by its title.
- Displays the song details if found or notifies the user if the song is absent.

7 Enable Repeat Mode

- Converts the playlist into a circular linked list, enabling continuous playback.
- The last node's next pointer is updated to point to the head.

8 Disable Repeat Mode

- Breaks the circular linked list by setting the last node's next pointer to NULL, restoring a standard singly linked list structure.

9 Exit

- Safely exits the application and deallocates all dynamically allocated memory using the ~Playlist destructor.

Technical Details:

Programming Language

The program is implemented in **C++**, a versatile programming language that provides control over low-level operations like memory management while offering high-level features like object-oriented programming.

Data Structure: Singly Linked List:

- Each node represents a song in the playlist.
- Nodes are linked together dynamically, allowing efficient memory usage and flexibility in operations like insertion, deletion, and traversal.

Class Design:

Class 1: Song

Represents individual songs.

Attributes:

- title: The name of the song.
- artist: The artist of the song.
- duration: The song's duration in seconds.

Methods:

- Constructor for initializing a song with its details.

Class 2: Node

Represents a node in the linked list.

Attributes:

- song: A pointer to a Song object.
- next: A pointer to the next node in the linked list.

Class 3: Playlist

Manages the entire playlist.

Implementation Details:

1 Adding Songs

- Dynamically allocates memory for new Song and Node objects.
- Appends the node to the end of the linked list.

2. Memory Management

- The program ensures all dynamically allocated memory is freed upon exit using the ~Playlist destructor.
- Each Node and Song is explicitly deleted to avoid memory leaks.

3 Error Handling

- Handles edge cases like empty playlists, invalid song titles, and repeat mode toggling on single-song playlists.

CODE:

```
#include <iostream>
#include <string>
#include <time.h>
#include <vector>
using namespace std;

class Song
{
public:
   string title;
   string artist;
   int duration; // duration in seconds

   Song(string t, string a, int d) : title(t), artist(a), duration(d) {}
};

class Node
{
public:
   Song *song;
   Node *next;

   Node(Song *s) : song(s), next(NULL) {}
```

```
};

class Playlist
{
private:
   Node *head;

public:
   Playlist() : head(NULL) {}

   // Add a song
   void addSong(string title, string artist, int duration)
   {
      Song *newSong = new Song(title, artist, duration);
      Node *temp = new Node(newSong);

      if (head == NULL)
      {
         // If the list is empty
         head = temp;
         return;
      }

      Node *curr = head;
```

```
    Node *prev = NULL;
    while (curr)
    {
        prev = curr;
        curr = curr->next;
    }
    if (prev == NULL)
    {
        // Insert at the head
        temp->next = head;
        head = temp;
    }
    else
    {
        prev->next = temp;
        temp->next = curr;
    }
}

// Remove a song by title
void removeSong(string title)
{
    if (head == NULL)
    {
```

```
    cout << "Playlist is empty!" << endl;
    return;
}

Node *curr = head;
Node *prev = NULL;

while (curr != NULL)
{
    if (curr->song->title == title)
    {
        if (prev == NULL)
        {
            head = curr->next; // Remove head
        }
        else
        {
            prev->next = curr->next;
        }
        delete curr->song;
        delete curr;
        cout << "Song removed successfully!" << endl;
        return;
    }
```

```
            prev = curr;
            curr = curr->next;
        }
        cout << "Song not found!" << endl;
    }

    // Display the playlist
    void display()
    {
        if (head == NULL)
        {
            cout << "Playlist is empty!" << endl;
            return;
        }

        Node *curr = head;
        int index = 1;
        int totalDuration = 0;

        while (curr != NULL)
        {
            cout << index++ << ". " << curr->song->title << " by " << curr->song->artist
                 << " (" << curr->song->duration << " seconds)" << endl;
```

```
        totalDuration += curr->song->duration;
        curr = curr->next;
      }
      cout << "Total Duration: " << totalDuration << " seconds" <<
endl;
   }

   // Reverse the playlist
   void rev_playlist()
   {
      Node *prev = NULL;
      Node *curr = head;
      Node *next = NULL;

      while (curr != NULL)
      {
         next = curr->next;
         curr->next = prev;
         prev = curr;
         curr = next;
      }
      head = prev;
      cout << "Playlist reversed!" << endl;
   }
```

```
// shuffle playlist

void shuffle_playlist()
{
   if (!head || !head->next)
   {
      cout << "Playlist is too small to shuffle!" << endl;
      return;
   }

   srand(static_cast<unsigned int>(time(0)));

   vector<Node *> nodes;
   Node *curr = head;
   while (curr)
   {
      nodes.push_back(curr);
      curr = curr->next;
   }

   for (int i = nodes.size() - 1; i > 0; --i)
   {
      int j = rand() % (i + 1);
```

```
        swap(nodes[i], nodes[j]);
    }

    head = nodes[0];
    curr = head;
    for (int i = 1; i < nodes.size(); ++i)
    {
        curr->next = nodes[i];
        curr = curr->next;
    }
    curr->next = NULL;

    cout << "Playlist shuffled!" << endl;
}

// Search for a song by title
void search(string title)
{
    if (head == NULL)
    {
        cout << "Playlist is empty!" << endl;
        return;
    }
```

```
    Node *curr = head;

    while (curr != NULL)
    {
        if (curr->song->title == title)
        {
            cout << "Song found " << ": " << curr->song->title
                << " by " << curr->song->artist << endl;
            return;
        }
        curr = curr->next;
    }
    cout << "Song not found!" << endl;
}

void Repeat_on()
{
    if (!head)
    {
        cout << "Playlist is empty!" << endl;
        return;
    }

    Node *curr = head;
```

```
    while (curr->next)
    {
        curr = curr->next;
    }
    curr->next = head; // Create a circular linked list
    cout << "Repeat mode enabled!" << endl;
}

// To break the repeat mode
void Repeat_off()
{
    if (!head)
    {
        cout << "Playlist is empty!" << endl;
        return;
    }

    Node *curr = head;
    while (curr->next && curr->next != head)
    {
        curr = curr->next;
    }
    curr->next = NULL; // Break the circular reference
    cout << "Repeat mode disabled!" << endl;
```

```
    }

    // Destructor to free memory
    ~Playlist()
    {
        Node *curr = head;
        while (curr != NULL)
        {
            Node *temp = curr;
            curr = curr->next;
            delete temp->song;
            delete temp;
        }
    }
};

int main()
{
    Playlist playlist;
    int choice;

    do
    {
        cout << "\n--- Music Playlist Manager ---\n";
```

```
cout << "1. Add Song\n";
cout << "2. Remove Song\n";
cout << "3. Display Playlist\n";
cout << "4. Reverse Playlist\n";
cout << "5. Shuffle Playlist\n";
cout << "6. Search Song\n";
cout << "7. Enable Repeat Mode\n";
cout << "8. Disable Repeat Mode\n";
cout << "9. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

switch (choice)
{
case 1:
{
   string title, artist;
   int duration;
   cout << "Enter song title: ";
   cin.ignore();
   getline(cin, title);
   cout << "Enter artist name: ";
   getline(cin, artist);
   cout << "Enter duration (in seconds): ";
```

```
            cin >> duration;
            playlist.addSong(title, artist, duration);
            cout << "Song added successfully!" << endl;
            break;
        }
        case 2:
        {
            string title;
            cout << "Enter the title of the song to remove: ";
            cin.ignore();
            getline(cin, title);
            playlist.removeSong(title);
            break;
        }
        case 3:
            playlist.display();
            break;
        case 4:
            playlist.rev_playlist();
            break;
        case 5:
            playlist.shuffle_playlist();
            break;
        case 6:
```

```
        {
            string title;
            cout << "Enter the title of the song to search: ";
            cin.ignore();
            getline(cin, title);
            playlist.search(title);
            break;
        }
        case 7:
            playlist.Repeat_on();
            break;
        case 8:
            playlist.Repeat_off();
            break;
        case 9:
            cout << "Exiting... Thank you for using the playlist manager!"
<< endl;
            break;
        default:
            cout << "Invalid choice! Please try again." << endl;
        }
    } while (choice != 9);


    return 0;
```

}

OUTPUT:

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 1

Enter song title: song1

Enter artist name: sr1

Enter duration (in seconds): 1

Song added successfully!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 1

Enter song title: song2

Enter artist name: ar2

Enter duration (in seconds): 2

Song added successfully!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 1

Enter song title: song3

Enter artist name: ar3

Enter duration (in seconds): 5

Song added successfully!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 1

Enter song title: song4

Enter artist name: ar4

Enter duration (in seconds): 9

Song added successfully!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 3

1. song1 by sr1 (1 seconds)

2. song2 by ar2 (2 seconds)

3. song3 by ar3 (5 seconds)

4. song4 by ar4 (9 seconds)

Total Duration: 17 seconds

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 4

Playlist reversed!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 3

1. song4 by ar4 (9 seconds)

2. song3 by ar3 (5 seconds)

3. song2 by ar2 (2 seconds)

4. song1 by sr1 (1 seconds)

Total Duration: 17 seconds

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 5

Playlist shuffled!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 3

1. song1 by sr1 (1 seconds)

2. song2 by ar2 (2 seconds)

3. song4 by ar4 (9 seconds)

4. song3 by ar3 (5 seconds)

Total Duration: 17 seconds

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 6

Enter the title of the song to search: song3

Song found : song3 by ar3

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 2

Enter the title of the song to remove: song3

Song removed successfully!

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 3

1. song1 by sr1 (1 seconds)

2. song2 by ar2 (2 seconds)

3. song4 by ar4 (9 seconds)

Total Duration: 12 seconds

--- Music Playlist Manager ---

1. Add Song

2. Remove Song

3. Display Playlist

4. Reverse Playlist

5. Shuffle Playlist

6. Search Song

7. Enable Repeat Mode

8. Disable Repeat Mode

9. Exit

Enter your choice: 9

Exiting... Thank you for using the playlist manager!

Conclusion:

The Music Playlist Manager is a comprehensive application that demonstrates the practical utility of linked lists. The project highlights the flexibility and efficiency of dynamic data structures while offering users a practical tool for playlist management.

References:

- **C++ Documentation: https://cplusplus.com**
- **GeeksforGeeks: Linked List**
- **Random Shuffling Algorithm**

SUBMITTED BY:

SAI CHARAN GOKA,

123CS0168