Data Structures and Algorithms
SE-F22
LAB-09

Issue Date: May 03, 2024

Start Time: 10:00 AM
Completion Time: 01:00 PM

Total Marks: 60

Objective:

The objective of this lab is to reinforce understanding and practical implementation of linked lists, particularly doubly linked lists, in the context of managing a music playlist. Students will develop a MusicPlayer class that utilizes a doubly linked list to handle song operations and playlist management.

Instructions:
1) Follow the question instructions very carefully, no changes in function prototypes are allowed.
2) You could solve the following growth functions on paper.
3) Anyone caught in an act of plagiarism would be awarded an "F" grade in this Lab.

Music Playlist Management Lab

In this scenario-based lab, students will develop a Music Playlist Management system using a doubly linked list. The system will allow users to add, remove, and rearrange songs in a playlist, as well as play the playlist in both forward and reverse orders. Each song in the playlist will be represented by a Song object, containing attributes such as song ID, title, artist, album, and duration. Students will implement the MusicPlayer class, leveraging the doubly linked list implementation for efficient playlist management.

Task 01 (Basic Structure) [5 Marks]

The Song class would hold the information related to a song. Following is the structure for song class:

```
class Song {
public:
    int songID;
    string title;
    string artist;
    string album;
    int durationInSeconds;

    Song(int id, string t, string a, string al, int dur);
};
```

Use your doubly linked list created in class but make sure it is implemented using templates as each node of the list would be holding a song object with attributes described above.

For instance your Linked List class should be similar to:

```
template <typename T>
class DLLNode {
public:
    T data;
    DLLNode* prev;
    DLLNode* next;

    DLLNode(T d, DLLNode* n, DLLNode* p) : data(d), prev(p), next(n) {}
};
```

```
template <typename T>
class DoubleLinkedList {
private:
    DLLNode<T>* head;
    DLLNode<T>* tail;

public:
    DoubleLinkedList() : head(nullptr), tail(nullptr) {}

    void addToListTail(T element);

    T deleteFromDLListTail();

    void addToDLListHead(T element);

    T deleteFromHead();

    void display() const;
};
```

Task 02 (Music Player Impementation) [10 Marks]

Implement a class MusicPlayer that uses the provided doubly linked list implementation to manage a playlist of songs. The MusicPlayer class should include the following functionalities:

- Add a song to the playlist
- Remove a song from the playlist by song ID
- Play the playlist (forward from the current song)
- Move forward/backward in the playlist (move the current song to next and previous accordingly)
- Display the current playlist
- Display the current song

This class should have two data members, a doubly linked list to store the playlist of songs and a node to store the current song being played. If the current song is NULL set it to the head of playlist. If current song reaches to the end of playlist, set it the head again.

```
class MusicPlayer {
private:
    DoubleLinkedList<Song> playlist; // Playlist using a doubly linked list
    DLLNode<Song>* currentSongNode; // Pointer to the current song in the playlist

public:
    MusicPlayer(); // Constructor

    void addSongToPlaylist(const Song& song); // Add a song to the playlist
    void removeSongFromPlaylist(int songID); // Remove a song from the playlist by ID
    void playPlaylist(); // Play the playlist (forward from the current song)
    void moveForward(); // Move to the next song in the playlist
    void moveBackward(); // Move to the previous song in the playlist
    void displayPlaylist(); // Display the current playlist
    void displayCurrentSong(); // Display the current song in the playlist
};
```

Task 03 (Shuffle Playlist) [20 Marks]

The shufflePlaylist function in the MusicPlayer class is designed to shuffle the songs in the playlist in pairs, creating a distinct and rhythmic listening experience. This method of shuffling ensures that consecutive songs are swapped in pairs, maintaining a coherent flow while introducing variation to the playlist order.

Example:

If the playlist is like A → B → C → D.
Then after the shuffling it would be like B → A → D → C.

```
void shufflePlaylist(); // Shuffle the playlist
```

Task 04 (Merge Playlist) [15 Marks]

The mergePlaylists function in the MusicPlayer class is designed to merge two playlists into a single playlist while maintaining a sorted order based on song ID. This function enables users to combine multiple playlists seamlessly and create a unified and organized playlist.

Example:

Playlist 1: **3 → 8 → 9 → 14**
Playlist 2: **1 → 5 → 17 → 20**

Merged: **1 → 3 → 5 → 8 → 9 → 14 → 17 → 20**

```
void mergePlaylists(DoubleLinkedList<Song>& playlist2); // Merge two playlists
```

Note: You can assume that both playlists are already sorted based on the Song Id.

Task 05 (Delete Song from the End of Playlist) [10 Marks]

The deleteNthSongFromEnd function in the MusicPlayer class is designed to remove the nth song from the end of the playlist. This function allows users to selectively delete songs from the end of the playlist, providing flexibility in playlist management.

Example:

Playlist: A → B → C → D → E → F
n: 3

After deletion: A → B → C → E → F

```
void deleteNthSongFromEnd(int n); // Delete the nth song from the end of the playlist
```