Assignment 3 - musicalLL

1.0 Introduction

In this assignment, you will be tasked with implementing a musical playlist management system using linked lists in C. The assignment aims to create and manage a playlist of songs, each containing notes, and implement various functionalities like adding, playing, and removing songs. The playlist system will use a linked list structure, where each node represents a song and contains information about its song ID, name, notes, and a pointer to the next song.

2.0 Your task

With its user-friendly interface and robust functionality, **musicalLL** will be an ideal tool for musicalLLy inclined people looking to streamline their playlists. Via a menu-driven solution, create a program that allows a user to run basic tasks on song data, such as adding a new song to the current playlist, play a song in the playlist, and so on.

Your program must start with the display of a menu that has the following options. Each option must make an appropriate function call (See section 3.0 for the task that each function must perform).

1. Create a new playlist
2. Add a new song to an existing playlist
3. Play all songs in the given playlist
4. Play a song from the playlist, given its id
5. Play a song from the playlist, given its name
6. Count the number of occurrences of a note in a given song
7. Delete a song from the playlist, given its id
8. Delete the entire playlist
9. Exit

For this assignment, **assume that the user always executes menu option 1 when the program is first run**, before executing options 2, 3, 4, 5, 6, 7 or 8. After that, the user can enter any menu option in any random order.

After the completed execution of each of the menu options (except for the Exit option), the user should be brought back to the menu and re-prompted to enter an option. If the user enters a value other than 1 to 9, the user should be notified using an appropriate message of your choice, brought back to the menu and re-prompted to enter a valid option. This should continue until the user enters 9 to exit the program.

Section 3.0 describes the task that each function must perform. All function prototypes are given in section 3.o and also in the header file provided (givenA3.h).

The required files that you have to create and submit for this assignment are:

1. Source files: one for each function – file name must be the same as function name (case-sensitive). For example, create a file called createPlayList.c for menu option 1 – names such as createplaylist.c or CreatePlayList.c will NOT be accepted)

2. Testing file: This file will have your main () function. Name it mainA3.c

3. **makefile:** Your makefile must be saved on the root folder (i.e., folder A3 created by gitcloning your A3 repo). You are required to store the other files using the given folder structure of source, header and executable files – **read Section 4.0 for details on folder structure**. Your program will be compiled using the **make** command. DO NOT change the name of the makefile – you must name it **makefile** (all lowercase). Your code must compile with the following flags: -std=c99 -Wall

3.0 Technical Components

While it is likely that companies use a much more efficient means of storing their data, one of the most basic structures to hold an endless set of data is a **linked list**. In this assignment, we will create a linked list to store the data for all songs that we create, edit, delete, search, load and save. The linked list will rely on a user-defined structure and will follow this structure definition for it (note that MAX_LENGTH is defined in the header file).

The structure is as follows:

```
struct Song
{

   int songId;          // this is auto-generated by the program
                        // must be unique for each song in the playList
   char songName[MAX_LENGTH];       // name of the song
   char songNotes[21][NOTE_LENGTH]; // every song in this playlist has 21 notes
   struct Song *nextSong;
};
typedef struct Song A3Song;
```

Note that member *nextSong is a pointer that will be used to create and link the nodes in the linked list. This pointer will be used to hold the memory address for the next song.

You must write a separate function for each menu option (except the exit option). Each function should be contained in its own function file. You must create a main (**mainA3.c**) that displays the menu and executes the menu options based on user choice. All function **prototypes** are given in a file called **givenA3.h**.

Menu Option1: Create a new playlist

Option1 creates a playlist by calling a function named createPlayList. This C function takes two arguments: a pointer to the head of a linked list of A3Song structures (headLL) and a file name (fileName). Prototype is as given below:

```
int createPlayList(A3Song ** headLL, char fileName[MAX_LENGTH]);
```

This function reads in a .CSV file line by line and creates a linked list structure. The values that are parsed from the CSV file are stored in the Song structure after enough memory has been allocated. Each line contains the song name and exactly 21 notes. The first song should be inserted at the beginning of the linked list, followed by the other songs. If the file fails to open, the program should return -1, otherwise, the program should return the number of songs inserted into the playlist.

Note that each row in the given CSV file contains the name and notes for a song. However, song Id for each song is auto-generated by your code using the following formula. Note that song id must be unique for every song in the playlist.

length of the song name + a random integer between 1 - 1000 (both inclusive)

Hint: Use functions srand() and rand() to generate a random number

<Reprint the menu>

Menu Option2: Add a new song to the given playlist

This option adds a new song to a given playlist. Prototype is as given below:

```
bool addNewSong(A3Song ** headLL, int beginOrEnd);
```

This function adds a new song to the beginning or end of the linked list, depending on if beginOrEnd is passed as 1 or 2. It returns True for success and False if it fails for any reason. To populate the Song fields, the following must be done:

1. The program must prompt the user to enter the song name
2. The program must generate a unique song Id, using the same formula as before: length of the song name + a random integer between 1 - 1000 (both inclusive) **Hint:** Use functions srand() and rand() to generate a random number
3. The program must generate 21 random notes, the notes must not be anything other than **do, re, mi, fa, sol, la, ti. Hint:** Use functions srand() and rand() to generate a

random note

<Reprint the menu>

Menu Option3: Play all songs in the given playlist

Prototype is as given below:

```
void playPlayList(A3Song * headLL);
```

This function plays all songs in a given playlist. Each song should be displayed to the user, printing its Song ID, Song Name, and Notes. The Notes should be printed out on one line and separated by a "." character.

For example:

Song ID: 123
Song Name: Happy Birthday
Notes: Do.Re.Mi. and so on

<Reprint the menu>

Menu Option4: Play a song from the playlist, given its id

Prototype is as given below:

```
int playSongGivenId(A3Song *headLL, int givenSongId);
```

This function plays a song given its givenSongId. This function should print out the song that is being played: If the song is not found, return -1, otherwise return 1 if found.

For example:

Song ID: 321
Song Name: Blinding Lights
Notes: Do.Re.Mi. and so on

<Reprint the menu>

Menu Option5: Play a song from the playlist, given its name

Prototype is as given below:

```
int playSongGivenName(A3Song * headLL, char givenSongName[MAX_LENGTH]);
```

This function plays a song given its givenSongName. This function should print out the song that is being played: If the song is not found, return -1, otherwise return 1 if found.

For example:

Song ID: 523
Song Name: Sky
Notes: Do.Re.Mi. and so on

<Reprint the menu>

Menu Option6: Count the number of occurrences of a note in a given song

Prototype is as given below:

```
int countNotesInASong(A3Song * headLL, int givenSongId, char whichNote[4]);
```

This function counts and returns the number of times a given note appears in the song identified by givenSongId. If the song is not found, return -1

<Reprint the menu>

Menu Option7: Delete a song from the playlist, given its id

Prototype is as follows:

```
int deleteASongGivenId(A3Song ** headLL, int givenSongId);
```

This function removes the song identified by givenSongId. Returns -1 if the song doesn't exist, 1 otherwise.

<Reprint the menu>

Menu Option8: Delete the entire playlist

Prototype is given as follows:

```
void deletePlayList(A3Song ** headLL);
```

This function removes every song from the playlist.
<Reprint the menu>

4.0 Submission - Required Files and Folder structure

Required files to submit on A3 repo of gitlab.socs.uoguelph.ca

- All 8 function implementation files. You must submit these.
- `givenA3.h` — this file is given to you with the function definitions/prototypes, constants, library imports. You must add your student info, and declaration of academic integrity to this file. You must submit this.
- `mainA3.c` — your testing file/driver program used to verify outputs/test your functions. This file must contain your `main()` function. In `mainA3.c` file, you are still strongly encouraged to test throughout the entirety of the development lifecycle of this program. You must submit this.
- `makefile -` You must submit this.

  The main target in your makefile must be called `musicalLL.` Note that your makefile is required to have a target called `clean`, that allows to remove all object files and the executable file (`musicalLL`).

Required folder structure for A3 (same as A2)

```
A3 $
```

src include bin

Folder `src` must contain all your source files.

Folder `include` must contain your header file

Folder `bin` must contain your executable file (i.e. `musicalLL`)

Note that the makefile should be located in the top-level directory (e.g. in folder A3 create as a result of git clone command). The commands in makefile must call your source, header and executable files via relative path.

Running such a program would be done as follows:

```
./bin/musicalLL
```

Header Comment: Your `givenA3.h` header file will contain a program header comment signifying your academic integrity. It will be of the following form:

```
/*
  Student Name: Firstname Lastname
  Student ID: #######
  Due Date: Mon Day, Year
  Course: CIS*2500

  I have exclusive control over this submission via my password.
  By including this header comment, I certify that:
   1) I have read and understood the policy on academic integrity.
   2) I have completed Moodle's module on academic integrity.
   3) I have achieved at least 80% on the academic integrity quiz
  I assert that this work is my own. I have appropriate acknowledged
  any and all material that I have used, be it directly quoted or
  paraphrased. Furthermore, I certify that this assignment was written
  by me in its entirety.

*/
```

5.0 Marking

- Programs that don't compile receive an immediate zero (0).
- Programs that produce compilation warnings will receive a deduction of 1 mark per unique warning (i.e., 3 'unused variable' warnings will only deduct 1 mark).
- Loss of marks may also result from poor style (e.g., lack of comments, structure)
- Programs that use goto statements receive an immediate zero (0)
- Programs that use global variables statements receive an immediate zero (0)