CptS **122 – Data Structures**

Programming Assignment 2: Digital Music Manager & Doubly Linked Lists – Part I

Assigned: Monday, September 3, 2018
Due: Wednesday, September 12, 2018 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:
- Design and implement a dynamic doubly linked list
- Allocate and de-allocate memory at runtime
- Manipulate links in a dynamic list
- Insert items into a dynamic linked list
- Delete items from a dynamic linked list
- Edit items in a dynamic linked list
- Traverse a dynamic linked list

II. Prerequisites:

Before starting this programming assignment, participants should be able to:
- Analyze a basic set of requirements for a problem
- Compose C language programs
- Compile a program using Microsoft Visual Studio 2015
- Create basic test cases for a program
- Apply arrays, strings, and pointers
- Summarize differences between array notation and pointer notation
- Apply pointer arithmetic
- Apply basic string handling library functions
- Define and implement structures in C
- Summarize the operations of a linked list

III. Overview & Requirements:

Many of us have large digital music collections that are not always very well organized. It would be nice to have a program that would manipulate our music collection based on attributes such as artist, album title, song title, genre, song length, number times played, and rating. For this assignment you will write a basic digital music manager (DMM).

Your DMM program must have a text-based interface which allows the user to select from a *main menu* of options including: (1) load, (2) store, (3) display, (4) insert, (5) delete, (6) edit, (7) sort, (8) rate, (9) play, (10) shuffle, and (11) exit. For Part I of the assignment, you will only need to complete the *main menu*, (1) load, (2) store, (3) display, (6) edit, (8) rate, (9) play, and (11) exit features. The other features will be completed in the next part of the assignment.

➢ *What must the main menu contain?*
The *main* menu must display the following commands:

(1) load
(2) store
(3) display
(4) insert
(5) delete
(6) edit
(7) sort
(8) rate
(9) play
(10) shuffle
(11) exit

After a command is selected and completed, your program must display the *main* menu again. This procedure will continue until the "exit" command is selected.

➢ *What must "load" do?*
The "load" command must read all *records* from a file called `musicPlayList.csv` (you may find a sample file here) into a dynamic *doubly* linked list. The doubly linked list is considered the main playlist. As each record is read from the file, it must be inserted at the front of the list. Each *record* consists of the following attributes:

- Artist – a string
- Album title – a string
- Song title – a string
- Genre – a string
- Song length - a struct `Duration` type consisting of seconds and minutes, both integers
- Number times played – an integer
- Rating – an integer (1 – 5)

Each attribute, in a single record, will be separated by a comma in the .csv (comma separated values) file. This means that you will need to design an algorithm to extract the required attributes for each record. Each field in each record will have a value. You do not need to check for null or empty values.

You must define a struct called `Record` to represent the above attributes. Also, do not forget that the *Song Length* must be represented by another struct called `Duration`. `Duration` is defined as follows:

- Minutes – an integer
- Seconds – an integer

Finally, each struct *Node* in the doubly linked list must be defined as follows:

- Data – a `Record`
- Pointer to the next node
- Pointer to the previous node

➢ *What must "store" do?*
The "store" command writes the current *records*, in the dynamic doubly linked list, to the `musicPlayList.csv` file. The *store* will completely overwrite the previous contents in the file.

➢ *What must "display" do?*
The "display" command prints records to the screen. This command must support two methods, one of which is selected by the user:

1. Print all records.
2. Print all records that match an artist.

➢ *What must "edit" do?*
The "edit" command must allow the user to find a record in the list by *artist*. If there are multiple records with the same artist, then your program must prompt the user which one to edit. The user may modify all of the attributes in the record.

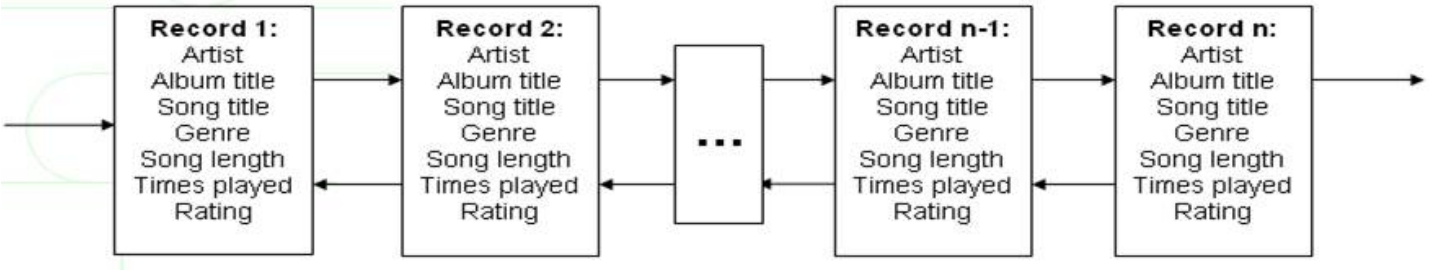
➢ *What must "rate" do?*
The "rate" command must allow the user to assign a value of 1 – 5 to a song; 1 is the lowest rating and 5 is the highest rating. The rating will *replace* the previous rating.

➢ *What must "play" do?*
The "play" command must allow the user to select a song, and must start "playing" each song in order from the current song. "Playing" the song for this assignment means displaying the contents of the record that represents the song for a short period of time, clearing the screen and showing the next record in the list, etc. This continues until all songs have been played.

➢ *What must "exit" do?*
The "exit" command saves the most recent list to the `musicPlayList.csv` file. This command will completely *overwrite* the previous contents in the file.

IV. Logical Block Diagram

The logical block diagram for your doubly linked list should look like the following:



As you can see from the illustration a doubly linked list has a pointer to the next node and the previous node in the list. The first node's previous node pointer is always NULL and the last node's next pointer is always NULL. When you insert and delete nodes from a doubly linked list, you must always carefully link the previous and next pointers.

V. Submitting Assignments:

1. Using the OSBLE+ MS VS plugin, please submit your solution. Please visit https://github.com/WSU-HELPLAB/OSBLE/wiki/Submitting-an-Assignment for more information about submitting using OSBLE+.
2. Your project must contain at least one header file (a .h file), two C source files (which must be .c files), and a local copy of the .csv file.
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

VI. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 5 pts – Appropriate top-down design, style, and commenting according to class standards
- 4 pts – For correct definition of struct `Record`
- 2 pts – For correct definition of struct `Duration`
- 3 pts - For correct definition of struct `Node`
- 5 pts – For correctly displaying the *main* menu, getting the command from the user, and executing the command
- 3 pts – For *looping* back to main menu after a command is executed
- 21 pts – For correctly constructing a *doubly* linked list, including:
    1. (6 pts) For correct implementation a `makeNode()` function, which allocates space for a struct `Node` on the heap, and initializes the node
    2. (9 pts) For correct implementation of `insertFront()` function, which calls `makeNode()` and returns 1 for successfully allocating space for a node; 0 otherwise
    3. (6 pts) For correct implementation of `printList()`, which visits each node in the list and prints out the contents of the record
- 15 pts – Correct "load" command implementation
    1. (2 pts) For correctly opening `musicPlayList.csv` for mode "read"
    2. (6 pts) For correctly extracting each attribute from each record in the file
    3. (5 pts) For correctly using `insertFront()`
    4. (2 pts) For correctly closing `musicPlayList.csv`
- 13 pts – Correct "store" command implementation, which writes the records in the list to the `musicPlayList.csv` file.
    1. (3 pts) For opening `musicPlayList.csv` for mode "write".
    2. (10 pts) For correctly writing all the *records* in the list to the file, maintaining the .csv format
- 7 pts – For correct "display" command implementation
    1. (2 pts) For displaying all records by using printList ()
    2. (5 pts) For *searching* for specific records based on *artist* and displaying matching record - should be able to use the same search function as used in the "edit" command
- 7 pts – Correct "edit" command implementation
    1. (2 pts) For *searching* for specific records based on *artist* – should be able to use the same search function as used in the "display" command
    2. (5 pts) For *editing* the record specified by the user
- 3 pts – Correct "rate" command implementation
- 7 pts – Correct "play" command implementation
    1. (2 pts) For playing all songs in order until the end of the list

2. (5 pts) For *searching* for specific song based on *song title* and playing all songs until the end of the list has been reached

- 5 pts – Correct "exit" command implementation, which writes the records in the list to the `musicPlayList.csv` file, and exits the program.