CS 262, ALL SECTIONS

PROJECT 3

DUE SUNDAY, MAY 02 AT 11:59 PM

In this project you will develope a movie managment system using linked lists. You already have many of your favorite movies on your computer, but wish to make an easier way to watch the movie you want, in the order you want. The first step is to come up with a way to create watch lists for your movies. Using the following specifications will get you started on your goal (after which, you can figure out how to actually watch the movies in your watch list.)

The Watchlist

The watchlist will be implemented using a singly linked list. Each node in the linked list represents a movie, and it has several data members corresponding to various attributes of the movie. Users should be able to create a watchlist, add/remove movies to it, search and manipulate the list in convenient ways. Specifically, the watchlist should support the following operations:

- Add a movie:
  - At the end
  - At the begining
  - At a specific position
- Move a movie:
  - up one position
  - down one position
- Remove a movie from the watchlist
- Print the entire watchlist
- Show watchlist duration
- Search a movie by its title in the watchlist
- Read a text file and create a new watchlist
  -->

The structure representing a movie-node will be the following:

```
typedef struct _movienode
{
  char title[35];
  char genre[35];
  double duration;
  struct _movienode *next;
} movieNode;
```

The user will browse a movie library in order to find movies which can be added to the user's watchlist.

The Movie Library

The movie library will be implemented using another singly linked list. The user can browse the movie library by searching a particular movie, or by viewing all movies in alphabetical order. The user may choose to add a movie to the user's watchlist, in which case, the movie will be removed from the library and added to the watchlist. If a user removes a movie from his watchlist, the movie is inserted back to the movie library in proper alphabetical order.

The following operations are supported in the movie library:

- View all movies
- Search by title
- Add movie to watchlist

At the start of the program, a text file will be read to create the movie library. Each movie will constitute 3 lines in the text file, with each line representing title, genre, and duration, respectively. Your program will have to read this text file and create the movie library by inserting each movie information in to the singly linked list, in alphabetical order.

Flow of Operation

At the beginning, the program will create a movie library by reading data from a given text file. The name of the text file will be passed as a command line argument. Your program will read information for each movie from the text file and insert them in alphabetical order in a singly linked list. This singly linked list is your movie library. Initially, the user will have an empty watchlist and the movie library. The user's watchlist will also be a singly linked list. The program will present the user a menu, called the watchlist menu.

watchlist Menu

The watchlist menu will have the following options:

1. Print watchlist
2. Show duration
3. Search by title
4. Move a movie up
5. Move a movie down
6. Remove a movie
7. Save watchlist
8. Load watchlist
9. Go to movie Library
10. Quit

The options 3, 4, 5 and 6 ask the user to input a string as movie title. If the user chooses option 9, " Go to movie Library", then another menu will be diswatched, called the library menu.

Library Menu

The library menu will have the following options:

1. View all movies
2. Search by title
3. Add a movie to watchlist
4. Back to watchlist

The options 2 and 3 will take the movie title as input from user. If the user choose to add a movie to the watchlist, the program will ask for the title of the movie and it will do similar operations as "Search by title". If the movie is found, a third menu will be diswatched, called Add movie Menu.

Add movie Menu

This menu will have the following options:

1. Add movie to the end
2. Add movie at the beginning
3. Insert movie at a specific position

Option 3 will ask the user again to input another non-negative integer value as position. Position values start from 0. If a user enters a negative number as position, the program should ask for a correct value. If the user enter a value larger than the number of movies in the list, then the movie is added to the end of the list. Once a movie has been selected by the user and added to the watchlist, the entire watchlist should be printed with an appropriate message. Then the Library Menu will be diswatched again.

Details of Implementation

There are two main components in this project- the **watchlist** and the **movie library**. We first describe the implementation of the watchlist.

- The watchlist will be a linked list of structures named `movieNode`, as given above. A pointer to `movieNode` is used to point to the first element in the list" called a `head` pointer. You could setup the initial empty linked list with a dummy node, or by making `head` a NULL pointerâ€" it is a design decision that affects the way you would write the functions for linked list operations, e.g., insertion, deletion, etc.
- A function **createmovieNode()** will take the movie attributes as input parameters and create a `struct` for the movie by allocating memory dynamically. This function returns a pointer to the newly created linked list node, which can be passed to a function called **insertmovie()** as a parameter. This function will insert the new movieNode in to the watchlist, in an appropriate position.
- A function called **printList()** will take the `head` pointer as input and it will traverse each `movieNode` in order. It will print detailed information about each movie as it traverses.
- A function called **computeDuration()** will take the `head` pointer as input and it will traverse each `movieNode` in order. It will sum the duration of all movies in the watchlist and print it to the console.
- Another function, **searchByTitle()** will take a string as input and search the linked list node which has the same title. When a user selects the "Search by title" option, they will be prompted for an input from the console. This input string will be passed to the function and the results will be diswatched. In order to make the search easier, the movie titles are always stored in all capital letters. Also for simplicity, it is guaranteed that each of the movie titles will be unique.
- Another function, called **removemovie()** will take a pointer to the `movieNode` to be removed as input and search for it in the list. If the node is found it is removed from the linked list, and returned by the function. If the node is not found, the function returns NULL.
- You will also need a function called **getNodePosition()** which takes the movie title as input and returns the position (index) of the corresponding movieNode in the linked list, or -1 if the movie is not found. The menu option for "move up" and "move down" can be implemented using the functions described above.
- For each menu option, a separate function should be written which calls the above described functions appropriately and in order, possibly handles any required user-input gracefully, and then diswatchs appropriate messages on the console.

If the user selects "Go to movie Library", then the second menu is diswatched, which shows operations on the movie library. All of the functionalities in the library can be implemented by using the functions described above (or by implementing close variants of them). Therefore, be sure to test and debug your basic linked list functions thoroughly and carefullyâ€" you will have to use these building blocks in order to create more complicated functionalities. If the user selects "Add a movie to watchlist", then the user will be prompted for a name of the movie, and the third menu will be shown. Based on the user's selection, the movie will be removed from the movie library (function removemovie()) and inserted to the watchlist (function insertmovie()) at the appropriate position.

The exact specification for each required function is given below:

- **movieNode *createmovieNode(char *title, char *genre, double duration)**
  Allocates a movie node, initializes it with the input parameters and returns a pointer to it.
- **int insertmovie(movieNode *head, movieNode *newNode, int position)**
  Inserts *newNode* at the specified *position* of the linked list pointed to by *head*.
- **void printList(movieNode *head)**
  Prints information for each movie from the linked list pointed to by *head*, in a nice formatted way.
- **double computeDuration(movieNode *head)**
  Computes total duration from each movie in the linked list pointed to by *head*, and returns it.
- **movieNode *searchByTitle(movieNode *head, char *search)**
  Takes a search-string called *search*, and capitalizes it. Then, it traverses the entire linked list starting from *head*, and returns the node when it find a match. If a match is not found, it returns NULL.
- **movieNode *removemovie(movieNode *head, movieNode *remNode)**
  Searches the movieNode pointed to by *remNode*, and if found, removes it from the linked list. It returns the node that is removed. If the node is not found, it returns NULL.
- **int deletemovie(movieNode *head, movieNode *delNode)**
  This function removes the node *delNode* from the linked list, then deallocates memory for it, saving us from any memory leak. It returns 0 on success, and -1 on failure. It can be implemented by calling removemovie(), and then free() on that node. You can use this function to clean up memory before your program exits.
- **int getNodePosition(movieNode *head, char *search)**
  Searches the linked list pointed to by *head* for the node with title matching the string in *search*, then returns the *position* (index number) of the node. Returns -1 if the node is not found.
- **int savewatchlist(movieNode *head)**
  This function prompts the user for a filename, writes each movieNode from the linked list (*head*) to the file. It returns 0 on success, and -1 on error.
- **movieNode *loadwatchlist()**
  This function prompts the user for a filename. Then it reads the file in order to obtain information for each movie in the watchlist, and creates the watchlist. The format of the file is left as a design choice. The only requirement is that your program should be able to read a file that was previous written by it using savewatchlist() function. On success, the function returns the head pointer to the new linked list. On error, it returns NULL.

The above functions must be present in your submitted code and they should perform the specified actions. In case of insertmovie(), removemovie() and deletemovie(), you may need to pass a movieNode **head, based on your design choice. Of course, you can (and should) write additional helper functions based on your design.

Suggested Steps for Implementation

The strategy for implementing this project successfully is based on modularizing the components, and individually test them thoroughly. For every component of your project, we suggest that you write a function, and then test it properly before integrating it with the main project. You may need to write some extra "test code" for this purpose. If you can find bugs early in a standalone module, then it will save you a lot of time that you would have spent debugging the entire project later on.

Below are the steps that we suggest for implementing this project:

1. Start by reading in the given `movie_library.txt` file. In your code, you will write a loop, in which, you will first read 4 lines from the file (information for 1 movie), and then print it on the console in a nice format.
2. Write the **createmovieNode()** function which takes the 4 values representing movie information read from the file and creates (allocates) a movieNode structure. Then it returns a pointer to it.
3. Write a function **void printmovieInfo(movieNode *movie)** which takes a pointer to the structure and prints the movie information on the console in a nice format. Move some of the code (for printing on console) you wrote in step 1 in to this function.

4. Next, we will first create one singly linked list which will hold all the movies in the movie library. Write a function **movieNode *createEmptyList()**, which creates an empty linked list and returns a pointer to its head. Based on your design choice, the head pointer may point to a dummy node, or to NULL.
5. Write a function which can append a movieNode to the end of the linked list, call it **appendmovie(movieNode *head, movieNode *newNode)**.
6. Write the function, **printList(movieNode *head)**, which traverses the entire linked list, and for each movieNode, it calls the **printmovieInfo()** function for printing movie details.
7. Write a similar function, **computeDuration(movieNode *head)**, which traverses the entire linked list, and for each movieNode, it sums up the duration of the movie. Then it returns the summation.
8. At this point, we need to focus on testing. Test the above functions thoroughly. Use the input file (`movie_library.txt`) for testing. Every time you modify the list using **appendmovie()** function, print the list to confirm that the changes are correctly applied. Also, use some incorrect values in the input file and make sure your program responds to incorrect input in a proper (graceful) way. Once you are confident about your implementation, take a backup of all your files in the project, and save it somewhere else.
9. In the next step, change the **appendmovie()** function such that it can also take an additional integer *position as* input. Instead of appending a movie to the end of the list, it will insert the movie at a specified *position.* This is the **insertmovie()** function.
10. Test the **insertmovie()** function thoroughly. Use incorrect inputs and show proper error messages.
11. At this point, you may want to implement the 3 menus (specified in "flow of operation") and test that you can navigate your menus properly.
12. Write the function **searchByTitle()** and test it thoroughly.
13. Implement the **removemovie()** function and test it thoroughly.
14. Now you have implemented your basic set of arsenal which will allow you to build various functionalities of this project. You can use the **createEmptyList()** function from step 4 to create the user watchlist (empty). You can implement the functionality "Add a movie to watchlist" by creating another function. This function prompts the user for the movie title, then uses **searchByTitle()**, **removemovie()** and **insertmovie()** to fetch a movie node from the movie library to the user watchlist. Test your code thoroughly.
15. Implement the **getNodePosition()** function.
16. Using **getNodePosition()**, **removemovie()** and **insertmovie()** functions, you can now implement the menu options "move up" and "move down" in the user watchlist. Once you complete that, conduct extensive testing of the functionalities.
17. The last two functions, **savewatchlist()** and **loadwatchlist()** can be implemented by traversing each movieNode in the watchlist and writing it to file or reading it from file, respectively.
    -->
18. Test your integrated program by navigating the menu options.
19. You will also need to implement a function which deletes the entire linked list, so that all dynamically allocated memory can be deallocated. You will need to call this function before your program terminates, in order to avoid memory leaks. You can call it deletemovieList(), or anything you prefer, and it is your choice whether you would want to make use of the deletemovie() function for implementing it.

Important Notes

- Your program should not have any memory leaks. Before your program exits, be sure to deallocate any memory that your program allocates at runtime.
- Your program should be as *robust* as possible. It should not crash abruptly even if the user inputs incorrect values, or if the input file is misformatted. It should diswatch proper error messages, and allow the user to redo the operation. If the error is unrecoverable, your program should terminate gracefully (e.g, free up any dynamically allocated memory, close any open FILE pointers, etc.)
- You cannot use any global variables.
- Be sure to follow the *program style guide* when writing your source code.

Testing and Submitting

Use valgrind to test your code for memory leaks and dangling pointers.

On zeus, create a directory named Project3_<username>_<labsection>. Copy your source files and Makefile to this directory. Change to this directory and create a typescript file showing that you are on zeus, listing your program to the screen, compiling it using the Makefile, and demonstrating a sample interactive run. Change to the parent directory and create a tarfile of your project directory. Name this tarfile Project3_<username>_<labsection>.tar

Submit this tarfile to Blackboard no later than 11:59 p.m. on Monday, May 2, 2021.

Update

- 4/15/2021 - **save watchlist** and **load watchlist** menu options are added