Programming Assignment 2: Gaming Library & Doubly Linked Lists

*~ Part One ~*

In this assignment, you will be creating your own text-based game library manager; let’s call it Sastruga. Each game (data entry) in Sastruga will contain the game title, game developer/publisher, genre, hours played, number of achievements, and rating.

***Bonus***: Instead of supporting just a single genre per game, allow each game to contain one or more genres. For the bonus, you may NOT simply append multiple genres onto one string, and you must be able to support an arbitrarily large number of genres **without** using an arbitrarily large amount of *extra* memory for each set of genres. Read through all of Part One first before coming back here and revisiting the bonus.

Being a text-based game library manager, Sastruga will, naturally, have a text-based interface allowing the player to begin from a menu and to select from the following options:

(\*) - unit tests not required

* Read the new ancient scriptures (load your game library)

Use test file, make sure linked list that is generated is in the correct configuration and has the correct data

* Preserve the sacred teachings (store your game library) (\*)
* Unveil the forgotten records (display the games in your library) (\*)
* Collect a new artifact (add a new game)
* Banish the exile (remove a game)
* Rewrite the stone tablets (modify a game)
* Organize the troops (sort your games)
* Decree divine judgement (rate a game)
* Embark on your next quest (play a game) (\*)
* Open the gates of chaos (shuffle your games)
* Retire to your chambers (exit the game library manager) (\*)

In *Part One*, you will begin with the reading, preserving, unveiling, rewriting, decreeing, embarking, and retiring. Once those are completed, move on to the second part to implement the collecting, banishing, organizing, and opening.

Additionally, you shall utilize a test-driven approach for this assignment. You must implement two methods for tests to be **optionally** easily run when desired: 1) changing the value of a single flag in the code (at the top of main below the preprocessor directives is a good place for this for now), and 2) supplying the command line argument --test. Confer with your instructor for further discussion and details.

As aforementioned, the player begins with the menu. After each time the player selects an option and that task is completed, the menu should be displayed again and the player allowed to select a new option, until the player chooses to retire.

*§* In order to read the new ancient scriptures (load your game library), Sastruga must read each game from a file called *hereMightBeDragons.csv* (a starter file will be provided) into a dynamic doubly linked list. This doubly linked list will be your game library. As each game is read from the file, it must be inserted at the front of the list; the details of each game data entry are as follows:

* Game title – a string
* Developer/publisher – a string
* Genre – a string
* Hours played – a struct Playtime type consisting of hours and minutes, both integers (you must create this struct yourself)
* Number of achievements – an integer
* Rating – an integer in the range [1,6] inclusive scored on a 1 through 5 scale (yes, those ranges do not match; no, this is not trickery)

Each attribute for a single game will be separated by a comma in the csv (comma separated values) file, and there will be one game data entry per line. It is up to you to design an algorithm to extract these attributes for each game. Each field in every game will have a value, and you do not need to check for null/empty values (this time).

For your doubly linked list, you will create a Node struct that contains a game entry, a pointer to the next node, and a pointer to the previous node.

*§*When it is time to preserve the sacred teachings (store your game library), Sastruga must write the current games in your game library (your dynamic doubly linked list) to the same *hereMightBeDragons.csv* file. This means the store command will completely overwrite the previous contents of the file. (Pro Tip: Keep a master/backup copy of the original starter file, so that in the very likely event something goes wrong (which is okay!) you can easily make a new, fresh copy of hereMightBeDragons.csv from the master/backup copy.)

*§* If the player decides (or even more likely, if you decide) to unveil the forgotten records (display the games in your library), Sastruga will display a sub-menu asking to either 1) Unveil all the forgotten records, or 2) Unveil all forgotten records with a rating of 6, and then display the corresponding games (and **all** of their information) to the screen. For the sanity of all parties involved, please make this output legible and make it easy to distinguish one game from another! Prettiness counts! (We are talking games after all.)

*§* To correct a mistake (or change one’s mind) and rewrite the stone tablets (modify a game), Sastruga must first allow the player to find a game in the list by the developer/publisher. If there are multiple games with the same developer/publisher in the list, Sastruga must prompt the player for which one to edit. The player may edit any/all of the information for the game. Aren’t you glad you don’t have to rewrite the stone tablets by hand?

Can mess with any fields, create a menu

If removing last genre from list, warn user and abort

Check that rating is between 1-6

*§* As decisions are made and one wishes to decree divine judgement (rate a game), the player must be allowed to assign a value of 1 through 6 inclusive to a game based on a 5-point scale, where 1 is terrible, 5 is excellent, and 6 is absolutely amazing. The new rating will replace the previous rating. All 5-star rating systems should include a 6th star.

*§* It is time! Embark on your next quest (play a game) by prompting the player to select a game to begin with. Sastruga will display that game’s information to the screen, and then continue to display the next game in the list’s information to the screen, and the next game, etc., until the end of the list is reached. The player may choose to begin “playing” from either the beginning of the list, or from a specific game in the list based on the game title. (Yes, I know, this isn’t as fun as it originally sounds, but hey, at least you’ll always live.)

*§* Rest is important, and to allow you to retire to your chambers (exit the game library manager), Sastruga must save the most recent list to the *hereMightBeDragons.csv* file, completely overwriting the previous contents of the file, before exiting the application.

Recall, that in a double linked list, not only does every node have a pointer to the next node and the previous node in the list, the first node’s previous pointer is always NULL and the last node’s next pointer is always NULL.

~ The Second Scroll ~

*§* If one’s wish is to collect a new artifact (add a new game), Sastruga must prompt the player for all of the attributes of a game. The new game data entry must be inserted at the front of the list.

*§* If one decides to banish the exile (remove a game), Sastruga must prompt the player for the title of the game to remove. If the game title does not exist in the list, the list must remain unchanged. And of course if a match is found Sastruga must delete the entry promptly without hesitation and with no questions asked. Clearly this is always the best approach to take forever and always and absolutely nothing can go wrong.

*§* In the case of imminent battle, Sastruga will organize the troops (sort your games) by first asking how the player would like their games sorted; the player may select to sort by (ASCII) ascending game title, ascending achievements, descending achievements, descending playtime, or (ASCII) descending developer/publisher. Then, Sastruga must perform the corresponding sort on the games in the doubly linked list **without** moving, copying, creating, or deleting any of the information for the games themselves (i.e. only the links between the nodes may change).

*§* Should one choose to open the gates of chaos (shuffle your games), Sastruga must provide a random order in which the songs are played. This command must **not** modify the links in the list. It must only specify the order in which songs are played, based on the position of the song in the list. For example, let’s say we have a list with 5 songs at positions 1 – 5 in the list. Shuffle must generate an order 1 – 5 in which the songs are played. An order 2, 5, 3, 1, 4 would require that the second song in the list is played first, the fifth song in the list is played second, the third song in the list is played third, the first song in the list is played fourth, and the fourth song in the list is played fifth. The songs are accessed by traversing the list both forwards and backwards to satisfy the order; you may **not** start from the head of the list more than once. Hence, the need for a doubly linked list!

~ Revisiting the Archives ~

Consider the bonus (supporting multiple genres per game). What if the task of displaying games to the screen had the option to display all games matching a genre instead of a developer/publisher? How would this affect your current implementation; would it be a big or small change, and why?

Say you wanted to implement this new change. How can you do so without writing an entire set of new logic/algorithms just for this feature? Does the data structure chosen to support multiple genres per game matter? (Hint: Think modular programming! Is there a functionality commonly used throughout this project that could be generalized?)

**Extra Bonus**: Update Sastruga to support a circular doubly linked list; this means that rather than the first node’s previous and last node’s next pointers pointing to NULL, the first node’s previous pointer will point to the last node, and the last node’s next pointer will point to the first node.

Important considerations: How do you determine when you’re at the end of the list? Where is the end of the list? Will operations that continued to the end of the linear linked list still stop at the same location in a circular linked list/could any operations stop at a different location? Do you need to update/replace any of your tests?

Food for thought: How does a circular linked list compare to a circular array?