Assignment DS3

Due Date: October 10

Purpose

Now that you have linked lists in your toolbox, data can be stored in a specified order. In this project, you will create a linked list which can be searched in several ways.

Problem

SMAX, a spinoff of HBO MAX, has hired you to write a database that stores movies and their ratings. Viewers will access your database and be able to search for a movie to see the IMDb rating as well as the Rotten Tomatoes score. Alternatively, viewers can search for the top scoring movie(s), or use other metrics.
You will write such a program that uses linked lists such that the data can be retrieved in two different orders. This will necessitate **two** links at each node.

Input

The program should prompt for the name of a file that contains the movie database. The file name can be any length, but is assumed to not include any spaces. The data will be in the form of a CSV (comma separated values) file, except that instead of commas, a single space will delimit each of the fields (data items). The file will contain any number of rows, each row having the data in exactly the following format:

Year Audience Score Rotten Tomatoes Movie Name

For example, a line in the file would look like:

```
2023 8.6 93 Oppenheimer
```

The year is an integer, the audience score is a float and represents the number of viewers/10, the Rotten Tomatoes score is a int representing a percentage, and the remainder of the line is the name of the movie, which can be any number of words.

After prompting for the movie database input file, the program should give the user the following menu (similar to DS2):

```
Menu ~~~~
m - Search by (m)ovie name
a - List all the movies in (a)lphabetical order by movie name
r - List top 15 movies by (R)otten Tomatoes score
w - Display (w)orst movie by Rotten Tomatoes score
i - (I)nsert a new movie into the database
u - (U)pdate a movie
q - (Q)uit
```

Elaboration:

- m - prompt for the movie name. Note that the input name may not exist in the list.
- i - prompt for all of the items needed for a movie to be added to the database. Inserting an item means putting it into the proper order for **both** scores.

- u - first prompt for a movie name. Then display the current data for that movie. Lastly, prompt for all of the items so that it can be updated. Note that you do not have to prompt for one specific item to update; just have the user update/redo all of the information (except for the movie name). Because a particular score may be changed, an updated movie has to be deleted and then re-inserted into the list so as to keep the proper ordering.

Output

The output depends on the choice made by the user. Whenever a movie is to be displayed, it should be in rank, name, year, audience score, and Rotten Tomatoes score order. For example, below is a sample of a few movies shown in Rotten Tomatoes score order (highest score goes first; ties can be displayed in any order):

```
Rank  Movie                    Year   Audience Score   Rotten Tomatoes
-------------------------------------------------------------------
  1.  Oppenheimer              2023          8.6/10              93%
  2.  She's Out of My League   2010          6.0/10              57%
  3.  Twilight                 2008          8.2/10              49%
```

Note that the table is aligned. All movie names should be left-justified. All numeric values should be right-justified. The audience score should have 1 decimal place. The Rotten Tomatoes score should be rounded to nearest whole percentage.

The output for the options 'm' and 'u' should include all of the information except rank. The output for 'w' should include all of the information, **including** rank. In any case, the information should be properly labeled and neatly displayed with the same number of decimal places as required above.

Specifics

- In order to display the top 15 in alphabetical or Rotten Tomatoes score, the list needs to be sorted by **both** values. To achieve this, your program must use a struct as follows:

```
struct dbNode {
   string movieName;
   int year;
   float audienceScore;
   float rottenScore;
   dbNode * nextName;
   dbNode * nextRot;
}
```

  Thus, the `nextName` pointer will point to the next movie in alphabetical order in the list, while `nextRot` will point to the next highest Rotten Tomatoes score.

- Your program should be written with OOP principles. That is, you should create a class such that the constructor creates the initial database from the given file. Public methods should allow the program to accomplish the various menu options. Additional methods, if needed, should be private.

- Methods should be no more than about 15 lines of code.
- You should have only those data members needed by the entire class in the public section. All other variables should be declared locally as needed.
- All data must be stored in a linked list using the struct given above. The struct should be declared **within** the class so as to enhance encapsulation.
- The program should test if the file name given by the user is valid. An error message should be displayed if not, and the program should stop. No other output should be displayed if there is an error.
- An incorrect menu option should produce an error message and then allow the user to type in another option.
- You may assume that the data file is correct; that is, it will not be missing any data or be incorrectly formatted. You may also assume that movie data typed in for the insert and update options will be correct.

Notes

- Use our sample linked list program and/or lab examples liberally.
- As before, write your program in stages. Pointers (links) can be particularly difficult to debug. Begin by creating a linked list of movies in order of just one of the metrics. Only after that works should you then add in the additional link to keep the other ordering. Continue to add methods to handle menu options one at a time.
- Pointers are notorious for generating our favorite *seg fault* (run-time) errors. Work with very small files, and test all possibilities; i.e., adding an item to the front of the list, to the end of the list, and to somewhere in the middle of the list.
- Don't forget comments. The usual rules apply.
- As before, write a declaration (.h) file, and implementation file, and an application file.
- As usual, the name of your source code or zip file should be your first initial, last name, and project number; e.g., `mgousieDS3.zip`. Submit your project before the dealine via Canvas. Submit hard copy of your source code in class the next day. Write/print and **sign** the Wheaton Honor Code Pledge on what you turn in: "I have abided by the Wheaton College Honor Code in this work."

Understanding [lists in C++] is not easy,
so once you have understood the material in this and the remaining sections,
you can consider yourself well beyond the novice stage as a C++ programmer.
– Patrick Henry Winston, in *On To C++*