Program 1 - Movie Chart App

Due Date: 5:00 p.m., March 9, 2016
All function interfaces are suggested naming and parameter guidelines. If you feel there is a better way, you are free to alter names, functions interfaces, etc, as long as you follow the lab and style guidelines. Output should match exactly unless otherwise stated.

In Program 1, you will continue with C++ I/O and text processing, and you will begin to utilize slightly more complex data structures to store information. In particular, you will implement a singly linked list container to store, manipulate, and save stories found in text files. Program 1 will provide you experience with:

- a simple singly linked list container that requires pointers
- C++ heap memory management (strategic and extensive use of new and delete)
- deep copies of stored data, including in copy constructors and assignment operators
- operator implementation and overloading (including assignment operators, unary operators, binary operators, and friend functions)
- some simple C++ class and object design.

This program will be difficult for some of you so please begin early and work on it consistently. You can do it and we will help! When you finish successfully, you will really understand linked lists, operator overloading, deep vs. shallow copy, and more, guaranteed!

Part A: Linked List

For part A you are going to write a singly linked list library. This means you will need a LinkedList.h and a LinkedList.cpp. The design of your linked list is largely up to you, but there are a few restrictions you must adhere to:

- All instance variables must be private and encapsulated
    - This means that everything must be accomplished through methods and you cannot return a pointer to anything inside the class (head, current, etc)
- It must be a singly linked list
    - No pointer to the tail of the list or pointers to previous nodes
- The Node class must be in LinkedList.h, not a separate file
- You list class must contain the following:
    - An internal iterator
    - A default constructor
    - A copy constructor
    - A destructor
    - All CRUD operations
        - How you implement these CRUD operations is up to you, but again, you must not expose the internals of the list. **This means you never return a pointer to any Node from any of your public methods.**

            - You may return a pointer to the data in a Node.
        - You also cannot ‘index’ into the list with the method. No indexInsert(), indexRead(), etc..
        - However, you will need to implement a find and delete for the ‘close’ command (described in section C).
    - The following overloaded operators
        - a << b
            - Should take data b and add it to the linked list a
        - a++
            - advances the internal iterator 1 node if not null

I strongly suggest you test your Linked List rigorously before moving on to the next part. Write a short main that runs several tests to ensure all methods are working.

Part B: Making Your Linked List a Template

Once you know your Linked List class is working, you will need to convert it to a Template class. Here is a review of template classes if you need it (We will/did cover it in class though). Your new list class should be able to store any type, which means you will have to write your methods to match.

Part C: Update and Expand Your Movie Chart Program

You will need to make the following changes to your existing code:

- Changes to your MovieChart class
    - Replace your Users array with your Linked List.
        - This means that any number of users can join MovieChart
    - Your constructor in Movie Chart class should read in from a file titled ‘users.txt’ which will have the following format:
        - Username1
First
Last
Age
Password
movie1
movie2
…

Username2

First
Last
etc…

        - *Be sure you check if the file exists before trying to read from it.*
    - Your destructor should clean up all memory, and write the entire user base information back out to the user.txt file in the same format.
- Changes to your User Class
    - You will need to add a copy constructor and destructor to your User class.
    - Use your Singly Linked List to store your Favorites rather than an array
    - Instead of returning a pointer to the favorites movies array, you should return copy (pass by value) of the favorite movies list.
        - Your copy constructor should automatically be called
    - To alter your favorite movies you will need an addMovie() and removeMovie() method in your User class
    - You will need to overload the == operator for the user class
    - You will need to add the MovieChart class as a friend class so you can access the private information in its destructor
- Changes to your Driver Code
    - Instead of just a “Favorites” that overwrites the entire list, alter your Update code to have an “Add” and “Remove” command that adds a new movie or removes a movie from the list, respectively.
    - Add a command, “Close”, that closes a user's account if they are logged in, removing the user from the list.
    - Add the command “List” which lists all user’s usernames and favorite movies
        - do not need to be logged in for this command

Part D : Code Organization and Submission

- Required code organization:
    - program1.cpp
    - User.h
    - User.cpp (if needed)
    - MovieChart.h
    - MovieChart.cpp
    - LinkedList.h
    - makefile
    - readme
- Create a readme file following the format provided, and add it to your project folder
- While inside your program1 folder, create a zip archive with the following command
    - zip -r program1 *

        - This creates an archive of all file and folders in the current directory called program1.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under 'Program 1'

Expected Interface and Test Output

Test Commands

- The following is an example of the test user.txt
- cat input.txt | ./program1 > output.txt
  diff expected.txt output.txt
    - *Expected Test Output (with our test case):*
        - Test case: favorites.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: invalidcommand.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: close.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: login.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: multiuser.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: passwordchange.txt
        - output.txt
    - *Expected Test Output (with our test case):*
        - Test case: userloggedin.txt
        - output.txt

Grading Guidelines

- **Part A**
    - Linked List contains an internal iterator (2 points)
    - Linked List does not return a pointer to a node (3 points)
    - The << and ++ operators are overloaded (2 points)
- **Part B**
    - Linked List must be templated to work with any type(4 points)

    - Destructor writes out a list of user's info for the MovieChart Class (2 points)

- **Part C:**
    - Passes all tests (each worth 3 points)

- **Part D:**
    - Follows formatting guidelines, requested project structure and naming conventions, contains written Readme, and submission does not include .o files or binary (1 point)

Formatting Guidelines

- Stores all values in a named variable.
    - No Magic Numbers.
- Uses indentation to identify code blocks.
    - Every Code block should be indented from it's parent block to identify scope.
- No single letter or non-descriptive variable names
    - The only exception to this rule is 'i' in a for loop
- Separates code blocks and logical sections with whitespace
    - Optimize your code for the reader, not the writer
- Output is formatted with an explanation of the output values
    - Format your output so that someone who does not know what the program is supposed to do would know what the output meant
- Each method is preceded by a comment explaining what the method does
- Each significant code block is preceded by a comment explaining what the code block does.
    - A significant code block is more than 3 lines performing a single logical operation
- CONSTANTS are in all caps
- Only data types start with a capital letter
    - Classes, Enums, Structs, etc.
- Do not use the 'using namespace' declaration in a header (.h) file
- In general we will follow the Google C++ style guidelines. If you want more info, you can view them here: https://google.github.io/styleguide/cppguide.html