Lab 3 - Creating Larger Programs

Due Date: 5:00 p.m., February 24, 2015
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.

The goal of Lab 3 is to extend your previous lab with new features and more capabilities. In particular: you will create an additional class to hold many different users, extend the user profile, add password authentication, and use command line arguments.

(Parts A must be completed in lab)

Part A: Extending the Movie List Program

Extend your C++ program to allow multiple users that can 'log in' to their account to make changes and keep track of their favorite movies. You should have the following classes and attributes:

- User
  - Instance Variables:
    - Last Name (C++ string - public)
    - First Name (C++ string - public)
    - Age (int - public)
    - Favorite Movie List (pointer to a C++ string - private)
    - Username (C++ string - private)
    - Password (C++ string - private)
  - Methods:
    - User(string fname, string lname, int age, string username, string password)
    - User()
    - ~User() //Don't implement a destructor
    - bool authenticate(string password)
    - string getUsername()
    - void setPassword(string new_password)
    - string * movieList()

Show your TA your code, and that you can compile your code with a makefile.
--END OF IN LAB REQUIRED WORK--

Part B: Container Classes

- MovieChart
  - Instance Variables
    - User users (static array of 255 Users - private)
    - int num_users (integer containing the number of users - private)
    - User * current_user (reference to a User - private)
  - Methods
    - MovieChart()
    - bool login()
      - should prompt the user for a username and password, then set current_user to the user and return true if there is a match. If there is no match, return false.
    - logout()
      - should set the current_user to NULL
    - create()
      - should prompt for first name, last name, age, username, and password
    - update()
      - should let the currently authenticated user select one of the following attributes to update:
        - Password
        - Favorites
      - If no user is authenticated, should throw a message, “Please login before continuing.”
    - view()
      - Should print out information regarding the current user:
        - First Name
        - Last Name
        - Age
        - Username
        - favorite movies

The program should initially prompt the user to make a selection by typing one of the following commands, with the following effects:

- Type “Create” to add all new information (name, age) and to null out the movie list.
- Type “Login” to log into the system. The user should be prompted for a username, then a password. If the user name or password does not exist, the user should should just be given a single message: “Invalid username/password combination”;
- Type “Update”, then select the attribute you would like to update.
- Type “View” to print out all the user’s current info EXCEPT the password.

- Type “Logout” etc...
- Type “Quit” to end the program.

Your program should echo (print out) any input it receives (including in the login, create, update methods) and should continue to accept commands until the user enters “Quit”.

Part C : Code Organization and Submission

- Required code organization:
    - lab3.cpp
    - User.h
    - User.cpp (if needed)
    - MovieChart.h
    - MovieChart.cpp
    - makefile
        - executable should be called: lab3
            - *do not add a .exe extension*
    - readme
- Create a readme file following the format provided, and add it to your project folder
- While inside your lab 3 folder, create a zip archive with the following command
    - zip -r lab3 *
        - This creates an archive of all file and folders in the current directory called lab3.zip
        - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under ‘Lab 3’

Expected Interface and Test Output

Test Commands

- cat input.txt | ./lab3 > 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: 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 B**
  - Passes Authenticated Test (1 point)
  - Passes Favorites Test (1 point)
  - Passes Invalid Command and Login Test (2 points)
  - Passes Login Test (2 points)
  - Passes Multiple Users Test (1 point)
  - Passes Password Change Test (2 points)
- **Part C:**
  - 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