CSC 415 Fall 2013

Assignment 3 – Introduction to C++

Due: Tuesday October 8, 2013 by midnight
Grade: 45 points, penalty 10% per day late

Objective:
The objective of this assignment is for students to develop proficiency in programming constructs, functions, arrays and classes in C++, and becoming more familiar with Unix-based operating systems and IDEs.

Requirements:
You must design and develop the solution for this assignment **individually**. General questions about design and programming concepts, requirements or compile error messages may be asked publicly in class or on Piazza. Students are encouraged to respond to such questions without waiting for my intervention. Specific questions about your code should be addressed to me directly, either through 'private' posts on Piazza, after class, or during my office hours.

Assume that you are a software engineer at GF Software Solutions, Inc. and that your company has been awarded a contract to develop a system for an online movie rental service. You have been assigned to develop a module to recommend movies to users based on ratings from reviewers with similar interests.

Design a class called `Movie` that stores data about a movie: movie ID, title of the movie, the year it was released, ratings by three critics, and the user's rating if available. The ratings range from 0 ("This is AWFUL") to 4 ("Love it").

Design another class called `MovieList` that uses a fixed size array to store objects of type `Movie`. Assume that there will be at most 1900 unique movies.

Write a program that uses the above classes and enables the user to:

- Input data about movies from a text file. Assume that the file has comma-separated values, i.e. the fields in the file are separated by commas.
- Store unique data in an object of type `MovieList`, i.e. duplicate movies are ignored.
- Manually input data to add a new movie. A movie is added only if the it does not already exist in the system.
- Update data for a movie that is already in the system.
- Rate movies he or she has seen.
- Ask for movie recommendations based on the closest match to critics' ratings (see further description of Recommendation List below).
- Write data out to a text file in the same format as the input file.

Recommendation List:
If the user has rated fewer than 5 movies, s/he is asked to rate a few more. The program then identifies the critic whose ratings most closely match the ratings input by the user and predicts the user's interest in the other movies by displaying the ratings by the chosen critic for movies not rated by the user. Only movies with ratings above 3.0 are displayed as recommendations.

You can use "distance", a very simplified version of the nearest neighbor classification algorithm, as a metric to determine how close a critic's ratings are to the user's ratings.

The distance between a user and a critic can be calculated using the formula below:

$$\text{distance} = \sqrt{\sum_{i=1}^{n} (cr_i - ur_i)^2}$$

where `n` is the total number of movies rated by the user
`i` is a movie rated by the user
cr_i is the rating given by the critic for movie `i` and
ur_i is the rating given by the user for movie `i`.

The closest match is the critic at the shortest distance from the user. It is then predicted that the user will give other movies ratings similar to those given by this critic. So the recommendation list will include movies for which that critic has given a rating above 3.

Part 1: Design due October 1

Using correct UML syntax, develop a class diagram for the structural design of this system. Consider how you can appropriately encapsulate data to facilitate information hiding and reuse. Your class diagram must show the relationships between the classes correctly. Be sure to label the diagram with your name and assignment number.

Use LucidChart or a similar UML-aware tool to draw the class diagram. Export the diagram as a `pdf` document or `jpeg` file. No other format will be accepted. You may sign up to join my LucidChart account at https://www.lucidchart.com/e/pulimood.tcnj.edu.

I will review the class diagram and give you feedback. Be sure to address this feedback in your final implementation. If required, submit a revised class diagram with Part 2.

Part 2: Implementation due October 8

Implement the classes and functionality described above, and write a simple driver that will test your implementation. A text file, `movies.csv`, has been provided for testing purposes, but the user should be allowed to specify the names of the input and output files.

Handle errors gracefully. For example, the program should not crash if the user inputs data of an incorrect data type, or the number of movies exceeds the capacity of the array.

Document every source code file submitted:

- Include identification information at the top of every file;
- Document every class definition and method;
- Name identifiers well so that your code is "self-documenting"; and
- Provide adequate comments for complex code fragments or algorithms.

Create a script file that demonstrates that your program compiles without errors or warnings, and executes without runtime errors. The script should also demonstrate all the functionality and error-handling capabilities that have been provided.

CSC 415 Fall 2013

General Instructions:
Before starting to work on this project, review Chapters 1 – 12 in Dale & Weems, the related slides, related class and homework exercises, as well as the guidelines posted on the Canvas wiki.

Your program must:

- Be implemented in C++ and follow best practices for software development,
- Include appropriate documentation, including for identification and maintenance,
- Implement a class (`MovieList`) that has a fixed size array as one of its attributes,
- Manipulate arrays extensively,
- Implement selection and iteration constructs,
- Read data from, and write output to, user-specified files,
- Enable user interaction (from the keyboard),
- Use appropriate formatting commands to display data neatly,
- Handle errors, such as user input and incorrect data types, and
- Be compiled on the Mac or Linux operating system using g++.

Deliverables:

Part 1: The UML class diagram that models the structure of the system, submitted as a `.pdf` or `.jpg` file, under 'Assignment 4' by 11:59 p.m. on October 1.

Part 2: Zipped together and submitted under 'Assignment 4' by 11:59 p.m. on October 8

- Script showing successful compilation and execution of your program.
- Complete source code for your program.
- A "readme" text file with a description of what your program does, and how to use it.
- Revised UML class diagram, if required.

Grade:

As per the rubric provided.