CSS 343 Data Structures, Algorithms, And Discrete Mathematics II

Spring 2019

Assignment 4, 100 possible points (12%)

Design

- Work in group of 2-4 students – setup your group by May 15 (Wednesday) on Canvas. Email me by May 15 noon if you'd like me to assign you to a group
- **Due by:** May 22 (Wednesday) 8:00 pm, submit to Canvas and bring one hardcopy to turn in class (30 points) and be peer evaluated.

Implementation

- Work individually or in pair. If in pair, 2 teammates' CSS 342 grades should be within 10 points (out of 100) difference
- **Due by:** June 11 (Tuesday) 11:59 pm, submit to canvas (70 points)

Description:

A local movie rental store wishes to automate their inventory tracking system. Currently there are three types of movies/videos (in DVD media) to be tracked:

- **Comedy** (denoted as '**F**' for funny)
- **Drama** (denoted as '**D**')
- **Classics** (denoted as '**C**')

Borrows and returns of items by customers are also to be tracked. Four types of actions are desired in the system:

- **Borrow** (denoted as '**B**'): (stock – 1) for each item borrowed
- **Return** (denoted as '**R**'): (stock + 1) for each item returned
- **Inventory** (denoted as '**I**'): outputs the inventory of all the items in the store
- **History** (denoted as '**H**'): outputs all the transactions of a customer

You will design and implement a program that will initialize the contents of the inventory from a file (**data4movies.txt**), the customer list from another file (**data4customers.txt**), and then process an arbitrary sequence of commands from a third file (**data4commands.txt**).

Details

In **data4movies.txt**, the information about each movie is listed as follows:

- For comedy movies: **F**, Stock, Director, Title, Year it released
- For drama movies: **D**, Stock, Director, Title, Year it released
- For classics movies: **C**, Stock, Director, Title, **Major actor Release date**

For example,

```
F, 10, Nora Ephron, You've Got Mail, 1998
D, 10, Steven Spielberg, Schindler's List, 1993
C, 10, George Cukor, Holiday, Katherine Hepburn 9 1938
C, 10, George Cukor, Holiday, Cary Grant 9 1938
Z, 10, Hal Ashby, Harold and Maude, Ruth Gordon 2 1971
D, 10, Phillippe De Broca, King of Hearts, 1967
```

Important notes about **data4movies.txt:**

- You may assume correct formatting, but codes may be invalid; e.g., in this data, the 'Z' code is an invalid entry so this line has to be discarded and users should be notified about this issue.
- While the stock for each line is 10, do not assume that is the case in your design and implementation.

- The **classical movie** type has a different format than the other two. It contains **Major actor** which is always formatted as two strings, first name and last name, separated by a space. Then the **Release date** contains month and year information, and no comma (but a space) between **Major actor** and **Release date.** In addition, for classical movies, one movie (e.g., Holiday) may have multiple lines so the total stock of this movie will be the sum of all the stocks in the lines about this movie (e.g., 20 for Holiday in the above example).

To store the data in the system, the items should be sorted as follows:
- comedy movies (**'F'**) sorted by Title, then Year it released
- dramas ('**D**') are sorted by Director, then Title
- classics ('**C**') are sorted by Release date, then Major actor

You can assume that each item is uniquely identified by its **complete set of sorting attributes**.

data4customers.txt contains customer information, one line per customer. Customers have a 4-digit ID number that uniquely identifies them, followed by last name, first name, all separated by a space. For example,

```
1111 Mouse Mickey
1000 Mouse Minnie
```

You can also assume that this data is formatted correctly.

data4commands.txt is used to test your code. The first field is action type (**B**, **R**, **I**, or **H**). Then the rest of fields are as follows:
- Action types **I:** no more fields
- Action type **H:** one more field **customer ID**. Fields are separated by a space
- Action types **B** and **R**: **customer ID** followed by **type of media (**currently only 'D' for DVD) then **movie type** and **movie data** (with values of the two sorting attributes, using comma or space to separate them as in the movie data file). Fields are separated by a space.

For example,

```
B 1234 D C 9 1938 Katherine Hepburn
B 1234 D F Pirates of the Caribbean, 2003
R 1234 D C 9 1938 Katherine Hepburn
B 1234 D D Steven Spielberg, Schindler's List,
S
H 1234
X 1234 Z C 9 1938 Katherine Hepburn
B 1234 D Y 2 1971 Ruth Gordon
B 9999 D F Pirates of the Caribbean, 2003
B 1234 D C 2 1975 Blah Blah
```

Again, the data will be formatted correctly, but may include errors.
- You must handle an **invalid action code** (e.g., 'X' in the above example), **invalid video code** (e.g., 'Y'), **incorrect customer ID** (i.e., not found. For example, 9999), and **invalid movie** (i.e., not found. For example, classic movie in month 2 of 1975 with a "Blah Blah" title). For bad data, discard the line and notify users.
- You must also handle incorrect commands. For example, a command that makes the number of an item in the inventory goes below zero, a return command when a movie was not borrowed, etc.

Overall Requirements:

- Do not print output for successful '**B**' or '**R**' commands, but print error messages for incorrect data and/or incorrect command. Output for '**H**' and '**I**' commands should be neatly formatted with one line per item/transaction. '**I**' should output all Comedy movies, then all Dramas, then all Classics. Each category of movies should be ordered according to the **sorting criteria** discussed above. '**H**' should show a list of DVD transactions of a customer in chronological order (latest to earliest) and specify whether the movie was borrowed or returned.
- You are required to use at least one hash table in this assignment. There are actually multiple places where they could be used. We will be covering hash tables soon in the class. If you want to get started, the important aspect of hash tables is that they are used for fast lookup of items. For example, if each item can be mapped into a unique number, you can use an array to store the items according to their unique number and look them up in O(1) time. Hash tables usually waste some memory, since not all of the array will be filled. However, the waste is not too bad, if you store pointers to items, rather than the items themselves.
- You are required to use inheritance. If you find you're using templates a lot, run it by me, as this assignment is designed for you to practice using inheritance.
- There are no other specific requirements for this assignment, but as always it should be well designed (not violate the basic design principles – will be discussed in class), easily extensible, efficiently coded, well documented, etc.

Design Requirements

Your design should document the work that needs to be done to complete the assignment. It should be a complete and clear description of how the program is organized. The more time you spend on your design, the less you will spend coding, debugging, and modifying.

Your design should include (at least) the following components **in this order**:

- Overview: This is a short description of the design and how the pieces fit together (the interaction between the classes). Include a description of main (List the objects that you have. Main should be short.)
- Class diagram: This is a UML diagram showing class relationships, including inheritance and composition. Each class should minimally list class name and public functions. You can use Visio or **neatly** draw this by hand.
- Class descriptions: For each class in the design, describe the data and methods as part of a documented C++ header file (**Exception**: you do not need to include h files of classes you will not implement, of the extensions beyond the assignment specifications, but you must include a description of those classes). The task that each function performs and the purpose of each data member should be clearly described. High-level pseudo code should be included for the most important methods (for example, those that control the flow of the program). Not all parameters need to be included for methods. Please order the files properly, i.e., put the most important classes first, put parent classes before children classes.

You should put considerable effort into the design (comparable to a lab implementation) or you should not expect to receive a good score. A useful approach is to imagine that you are writing a design that someone else will need to implement and extend. Document your design as you would want someone else to document for you. Some questions you might want to ask yourself:

- Can your design be extended beyond the specifications given here?
- Could you easily add new videos or DVDs to your design?
- Can you easily add other categories of videos or DVDs?

- Could you easily add new categories of media to your design, for example, music?
- Could you expand to check out other kinds of items, for example VCRs or DVD players?
- Could you easily add new operations to your design?
- Could you incorporate time, for example, a due date for borrowed items?
- Could you easily add an additional store, or handle a chain of stores?

Your design can go beyond the scope of these specifications (and you won't need to implement extensions). Thinking of possible extensions in advance often improves the design.

Submit softcopy to Canvas (one per each group) by May 22 8:00 pm. Also ***bring a hardcopy of your design to class on May 22.*** We will have in-class design reviews and then your hardcopy will be turned in.

Implementation Requirements:
All the rules, submission requirements, and evaluation criteria are the same as the assignment 2. One exception is that you may use STL libraries. BUT hash needs to be implemented by yourself.