**CSS 343 Homework #4     Due: March 11 (design due February 25)    Winter 2015**

**Record store simulation**

A local used record store wishes to automate their inventory tracking system. Three types of items are to be tracked:

–        Rock CDs

–        Classical CDs

–        DVDs

Purchases and trades of used items by customers are also to be tracked. Five types of actions are desired in the system:

–        Purchase: removes an item from the inventory

–        Trade-in: adds an item to the inventory

–        Display: outputs the entire inventory of the store, including the number of each item in inventory

–        History: outputs all of the transactions for a customer including the item (in chronological order)

–        HistoryAll: outputs the history for every customer, with the customers in alphabetical order.

You will design and implement a program that will initialize the contents of the inventory from a file, the customer list from another file, and, then, process an arbitrary sequence of commands from a third file.

**Details**

All CDs have an artist name, a CD title, and year of issue. Classical CDs also have a composer. DVDs have a director, a title, and a year of issue. The items in the inventory are sorted as follows:

–        Rock CDs are sorted first by artist name, then by year, then by CD title

–        Classical CDs are sorted first by composer, then by artist name, then by year, then by title

–        DVDs are sorted first by title, then by year, then by director

You can assume that each item is uniquely identified by the complete set of sorting criteria. To facilitate processing of the files, Rock CDs are marked ‘R’, Classical CDs are marked ‘C’, and DVDs are marked ‘D’.

The data file for initialization of the inventory lists each item on a separate line with a character denoting the type of item, number in inventory, artist name/director, title, year and composer (if necessary). Fields are separated by commas as follows:

R, 3, Metallica, Master of Puppets, 1986

R, 1, Queensryche, Operation: Mindcrime, 1988

C, 2, Sir Neville Marriner, Mozart: Requiem, 1991, Wolfgang Amadeus Mozart

C, 1, Janos Ferencsik, Beethoven Collection, 1990, Ludwig van Beethoven

Z, 1, Iron Butterfly, In-A-Gadda-Da-Vida, 1968

D, 3, Bryan Singer, The Usual Suspects, 1999

D, 1, Rob Reiner, This Is Spinal Tap, 1984

You may assume correct formatting, but the data could be invalid. In this data, the ‘Z’ code is an invalid entry.

Customer information is stored in a similar file. Customers have a 3-digit ID number that uniquely identifies them:

      123, Soze, Keyser

      456, Tufnel, Nigel

You can assume that this data is formatted correctly.

Your code will be tested using a third file containing the commands (P for Purchase, T for Trade-in, I for Inventory, H for History, and A for HistoryAll). Except for inventory commands, the second field is customer ID. For purchase and trade-in commands, the third field is the item type and the remaining fields are the item.

      P, 123, D, Bryan Singer, The Usual Suspects, 1999

      T, 456, C, Sir Neville Marriner, Mozart: Requiem, 1991, Wolfgang Amadeus Mozart

      H, 123

      I

      X, 999, R, Iron Butterfly, In-A-Gadda-Da-Vida, 1968

      P, 000, H, Anonymous, Some Title, 3000

      A

Again, the data will be formatted correctly, but may include errors. You should check to make sure that the number of each item in the inventory doesn’t go below zero. Do not print output for successful purchase or trade-in commands, but do print error messages for incorrect data. Output for your History and Inventory commands should be neatly formatted with one line per item/transaction. The inventory should output all rock CDs, then all classical CDs, then all DVDS. Each set should be ordered according to the criteria above. Note that all test files I use will include a blank line at the end of the file (like in previous assignments). Make sure your code handles this correctly.

**Design**

For your design, you may work in pairs. (However, the implementation will be done independently.) 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. You can use Visio or **neatly** draw this by hand.

–        Memory diagram:  This is a diagram showing the use of memory (with respect to data structures) in the program. Show, for example, any linked lists, trees, graphs, and/or hash tables and any relationships between the memory used for them. (Do they share data? Is there a pointer from one to another?)

–        Class descriptions:  For each class in the design, describe the data and methods as part of a documented C++ header file. The task that each function performs and the purpose of each data member should be clearly described. High-level pseudocode 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. (For these put the most important classes first, including manager classes and top-level base classes.)

The design for this lab is weighted as much as the implementation for other labs. You should put considerable effort into the design (comparable to a lab implementation) or you should not expect to receive a good score. Note that you should not use templates in this assignment. Use inheritance instead.. Any collection classes (for example, BSTree) should store a pointer to a base class object. Pointers to descendent classes can also be stored in the collection.

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 types of music or different media or books?

–        Could you easily add new types of items to the store (including very different items, such as clothing)?

–        Could you handle age restrictions on purchasing certain items?

–        Could you easily add other operations, such as rentals?

–        Could you easily add a way for customers/employees to review items? Items could have several reviews.

–        Could you handle another branch of the store or 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.

***Bring a hardcopy of your design to class on February 25.*** We will have in-class design reviews and then your hardcopy will be turned in. No softcopy will be necessary for the design stage.

**Hash tables**

You are required to use at least one hash table in this lab. There are actually multiple places where they could be used. ~~We will be covering hash tables immediately after the midterm.~~ We just covered hash tables. 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. It is generally better to use a hash table than a switch statement (or if-then-else-if) when there are several cases that must be tested to determine which function to call.