Project 1 – Recursion, Arrays, and Vectors

Computer Science 2413 – Data Structures

Fall 2014

This project is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple system to read, <u>search</u>, and write bibliographic data."

Objectives:

1.	Use C++ file IO to read and write files, while using `cin` and `cout` for user interaction.	*10 points*
2.	Encapsulate all primitive arrays inside classes that provide controlled access to the array data and retain information on array capacity and use.	*20 points*
3.	Integrate appropriate exception handling into classes that encapsulate arrays.	*10 points*
4.	Store the bibliographic entries in a data structure that varies in size as elements are inserted and removed.	*15 points*
5.	Use recursive binary search to insert bibliographic entries into and remove bibliographic entries from an initially empty data structure.	*15 points*
6.	Implement a program to manipulate bibliographic data as described below.	*10 points*
►	Develop and use an appropriate design.	*10 points*
►	Use proper documentation and formatting.	*10 points*

Description:

For this project, you will revise and improve PublicScholar from Project 0 in several ways. You are encouraged to reuse and build on your code from Project 0. In addition to the functionality provided by PublicScholar0.0, your new PublicScholar1.0 will allow users to search entries by name and delete entries by name. In addition, PublicScholar1.0 will have several changes "under the hood" including encapsulating primitive array data in classes, storing bibliographic entries in a data structure that varies in size as data is added and removed, and providing exception handling.

Operational Issues:

PublicScholar1.0 will read the bibliographic data file (a text file) in which each bibliographic entry will be organized as it was in Project 0. PublicScholar1.0 will prompt the user for the name of this file at the terminal prompt.

After reading in and storing all entries in the file, PublicScholar1.0 will enter a loop where it will prompt the user with four options: 'S' for search, 'D' for delete, 'P' for print, and 'E' for exit.

If the user selects search, PublicScholar1.0 will prompt the user for the name of the bibliographic entry on which to search. Once the user enters a name, PublicScholar1.0 will perform a binary search looking for a bibliographic entry with that name. If an entry with that name is found, it will be displayed to the screen in the same format as used by PublicScholar0.0. If an entry with that name is not found, a message will be presented to the user to that effect.

If the user selects delete, PublicScholar1.0 will prompt the user for the name of the bibliographic entry to delete. Once the user enters a name, PublicScholar1.0 will perform a binary search looking for a bibliographic entry with that name. If an entry with that name is found, the entry will be displayed to the screen in the same format as used by PublicScholar0.0 and the user will be asked to confirm whether that entry should be deleted. If the user confirms deletion, the entry should be deleted from the data structure PublicScholar1.0 uses to hold bibliographic entries. (Note that it will *not* be deleted from the file.) If an entry with that name is not found, a message will be presented to the user to that effect.

If the user selects print, PublicScholar1.0 will prompt the user for the name of the file to which to print. Once the user enters a file name, PublicScholar1.0 will write to that file. However, if the user hits return without specifying a file name, PublicScholar1.0 will write to standard out as with PublicScholar0.0.

If the user selects exit, PublicScholar1.0 will exit gracefully.

Implementation Issues:

PublicScholar1.0 will read and write files using C++ file IO rather than via redirected standard IO as with PublicScholar0.0. PublicScholar1.0 will use standard IO for user interaction (prompting the user for input, getting user input, and displaying information to the user).

Within PublicScholar1.0, you should encapsulate C primitive arrays inside classes that provide the full functionality of array, string, and/or vector classes as demonstrated in Chapter 3 of your textbook. Bibliographic information, such as field delimiters (such as "title"), field values (such as "Hybrid simulation models {\textendash} When, Why, How?"), and the entire bibliographic database, then, should be stored within appropriate objects of these classes. For example, you might use a class very similar to the String class of Chapter 3 for both field delimiters and field values, since both are essentially strings. Alternately, you might decide that field delimiters and field values are different enough that you will create separate classes for them, perhaps with each being a subclass of the more general String class. On the other hand, it wouldn't make sense to try to keep the entire bibliographic database as one giant String object but would make sense to keep it as something along the lines of a Vector of bibliographic entries.

For each of these classes that encapsulates C primitive arrays (such as String and Vector), you must include appropriate exceptions and exception handling, as with the examples in Chapter 3. You must make it clear in your code documentation what exceptions you are providing and why they are used as they are.

For PublicScholar1.0, you are no longer allowed to store the entire bibliographic database in a single large data structure of fixed size. Rather, PublicScholar1.0 must initialize the database of bibliographic entries to some relatively modest initial capacity (say, 100 entries), then double its size as necessary when adding entries.

PublicScholar1.0 must keep the database of bibliographic entries sorted by entry name as each new entry is read in. This must be accomplished by performing a recursive binary search to determine the appropriate place in the database to insert each new entry, shifting entries out of the way to perform the insertion as necessary to accommodate the new entry.

Similarly, when a user asks to search for or delete an entry by name, PublicScholar1.0 must accomplish this by performing a recursive binary search for the entry. If a deletion is confirmed, PublicScholar1.0 must remove the deleted entry from the database and shift the following entries as necessary to prevent "holes" in the database. If the number of entries in the database falls below half of its current capacity, the data structure should shrink itself by ½ and return the unused space to the free store.

Similar to PublicScholar0.0 – *but note the differences*, for each class you implement, you must implement appropriate (and appropriately named) constructor, destructor, accessor, mutator, display, and (if the class is an array, string, vector, or similar class) *allocation and capacity* methods. The allocation methods will indicate how many items are stored in a given "array" and the capacity methods will indicate how many items it is possible to store in a given "array." For example, if the example entry above is stored in the variable `exEntry`, then `exEntry.getAuthors().allocation()` would return the value 3 because the example above shows three authors while `exEntry.getName().allocation()` would return the value 20 because the name of the entry contains 20 characters (including the final '`\0`').

Due Date:

You must submit an electronic copy of your source code through the dropbox in D2L by **Wednesday, October 1st by 2:45pm**.

Notes:

In this project, the only libraries you will use are iostream, fstream, and cstring.

Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, accessibility modifiers, etc.

Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.