Project 2 – Linked Lists

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, merge, purge, search, and write bibliographic data."

Objectives:

1.	Use C++ file IO to read and write files, while using `cin` and `cout` for user interaction.	*5 points*
2.	Encapsulate all primitive arrays inside classes that provide controlled access to the array data and retain information on array capacity and use.	*10 points*
3.	Integrate appropriate exception handling into classes that encapsulate arrays.	*5 points*
4.	Store the bibliographic entries in a linked list maintained in order, sorted by name.	*25 points*
5.	Remove entries from the linked list while maintaining its order.	*20 points*
6.	Integrate appropriate exception handling into classes that implement linked lists.	*10 points*
7.	Add user options to select additional functionality (merge and purge), prompt the user for additional file names, and provide error messages in case of file access errors.	*5 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 1 in several ways. You are encouraged to reuse and build on your code from Project 1. In addition to the functionality provided by PublicScholar1.0, your new PublicScholar2.0 will allow users to merge multiple bibliographic data files into one database in memory, and purge all bibliographic entries named in selected files from the database in memory. PublicScholar2.0 will also have a major change "under the hood" – rather than storing the list of bibliographic entries using a "vector" (an array that automatically resizes as entries are added or removed), PublicScholar2.0 will store the database in memory using a linked list data structure.

Operational Issues:

PublicScholar2.0 will behave as described for PublicScholar1.0 with the exception that the user will have two new options: 'M' for merge and 'P' for purge. Also, since 'P' was previously used for print, that option will be remapped to 'W' for write.

If the user selects merge, PublicScholar2.0 will prompt the user for the name of a bibliography file at the terminal prompt. PublicScholar2.0 will then attempt to read that file and merge its entries into the existing database, keeping the database sorted in order by entry name. If, in the course of reading this file, PublicScholar2.0 encounters an entry name equivalent to one already present in the file, the old entry will be replace by the new entry. (If PublicScholar2.0 has trouble accessing the merge file named by the user, it should provide the user with an appropriate error message.)

If the user selects purge, PublicScholar2.0 will likewise prompt the user for the name of a bibliography file at the terminal prompt. PublicScholar2.0 will then attempt to read that file but in this case it will purge (delete) from the database the entry with the same name as that of the entry just read from the file. If no such entry is found at the appropriate place in the list, PublicScholar2.0 will display to the user a message that an entry with that name was not found and move on to the next entry in the file. Essentially, PublicScholar2.0 is performing a set difference operation between the database and the entries in the purge file. (If PublicScholar2.0 has trouble accessing the purge file named by the user, it should provide the user with an appropriate error message.)

Implementation Issues:

In most areas, PublicScholar2.0 will be implemented just as was PublicScholar1.0. This includes how PublicScholar reads and writes files, carries out user interaction via standard in and standard out, encapsulates C primitive arrays, and how exception handling is implemented for arrays and similar classes. The big implementation change will be the data structure used in the code to hold the list of bibliographic entries. For PublicScholar2.0, you are no longer allowed to store the bibliographic database in a vector – instead, PublicScholar2.0 must store the database in a linked list.

As with PublicScholar1.0, PublicScholar2.0 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 linear search to determine the appropriate place in the database to insert each new entry and changing pointers as necessary to accommodate the new entry. This is true for both the initial file read in and any merge files specified by the user.

Similarly, as with PublicScholar1.0, when a user asks to search for or delete an entry by name, PublicScholar2.0 must accomplish this by performing a linear search for the entry. If a deletion is confirmed, PublicScholar2.0 must remove the deleted entry from the database and adjust pointers as necessary. With PublicScholar2.0, purge files will likewise be processed by linearly searching the database for each entry to delete and deleting it while retaining the order of the list.

Due Date:

You must submit an electronic copy of your source code through the dropbox in D2L by **Wednesday, October 22nd 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.