Project 2 – Linked Lists

Computer Science 2413 – Data Structures – Fall 2016

Due by 11:00 pm CST on Tuesday, 1 November 2016

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

User Request:

"Create a simple system to read, merge, purge, sort, search, and write merchant vessel data with error checking."

Objectives:

1	Use C++ file IO to read and write files, while using C++ standard I/O (`cin` and `cout`) for user interaction, using appropriate exception handling.	*5 points*
2	Encapsulate primitive arrays inside a templated class that provides controlled access to the array data, retains information on array capacity and use, and can be used to store data of any class or primitive type.	*5 points*
3	Integrate appropriate exception handling into the templated array class.	*5 points*
4	Efficiently sort the data based on the field specified by the user.	*5 points*
5	Efficiently search the data based on the field specified by the user.	*5 points*
6	Integrate appropriate exception handling into classes that implement searching and sorting.	*5 points*
7	Store merchant vessel data in a linked list maintained in order, sorted by country name.	*15 points*
8	Remove entries from the merchant vessel linked list while maintaining its order.	*10 points*
9	Encapsulate linked lists inside a templated class that provides controlled access to the linked-list data, retains information on linked list size, and can be used to store data of any class or primitive type.	*15 points*
10	Integrate appropriate exception handling into classes that implement linked lists.	*5 points*
11	Add user options to add 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 MVP 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 MVP1.0, your new MVP2.0 will allow users to merge and purge merchant vessel data from multiple files in its working memory throughout its use. MVP2.0 will also have a major change "under the hood" – rather than storing the list of merchant vessels using an array that automatically resizes as entries are added, MVP2.0 will store the list of merchant vessels by country using a linked list data structure when countries are being added to and removed from MVP2.0's working memory. (Note that the list of

merchant vessels by country will still use a resizable array when the data is to be sorted and searched using binary search.)

Operational Issues:

MVP2.0 will behave as described for MVP1.0, except that rather than have separate data input and data manipulation loops, there will be a single user option loop that will allow the user to merge new data or purge existing data at any time by selecting ‘M’ for merge and ‘P’ for purge alongside the options present for output, sort, find, and quit. Also, there will be an additional data printing/display option ‘A’ and a change to the data display option ‘O’.

The user will be prompted for a data file name when MVP2.0 is first started, so that it has at least some data present before it enters the user option loop. As MVP2.0 reads that first data file, it will ensure that the list is maintained in order based on name. (If MVP2.0 has trouble accessing the data file named by the user, it should provide the user with an appropriate error message.)

If the user selects merge, MVP2.0 will prompt the user for the name of an merchant vessel data file at the terminal prompt. MVP2.0 will then attempt to read that file and merge its entries into the existing database, keeping the systems sorted in order by name. If, in the course of reading this file, MVP2.0 encounters an entry name equivalent to one already present in the file, the old entry will be replaced by the new entry. (If MVP2.0 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, MVP2.0 will likewise prompt the user for the name of a merchant vessel data file at the terminal prompt. MVP2.0 will then attempt to read that file but in this case it will purge (delete) from the database the entry with the same country name as that of the entry just read from the file. If no such entry is found at the appropriate place in the list, MVP2.0 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, MVP2.0 is performing a set difference operation between the database and the entries in the purge file. (If MVP2.0 has trouble accessing the purge file named by the user, it should provide the user with an appropriate error message.)

During each merge and purge, MVP2.0 must update its internal totals to reflect the new totals from the merged and purged data. These totals are the totals MVP2.0 will display/print for either option ‘A’ or ‘O’ (as described below).

At the end of each merge and purge operation for the merchant vessel list, MVP2.0 will replace the sorted array of merchant vessel data with a new array of merchant vessel data sorted by country name by discarding the old merchant vessel array and copying the data from the merchant vessel linked list to a new merchant vessel data array.

If the user selects ‘A’ for alphabetical order, MVP2.0 will go through the linked list of merchant vessel data, printing each country data line in the order it is found in the linked list, which should be in alphabetical order. As with option ‘O’, this data should be preceded by the two data header lines read from the data files and followed by a line showing accurate column totals.

For both option ‘A’ and ‘O’, MVP2.0 will prompt the user for a file name to which to write the output. If the user hits enter without entering a filename, MVP2.0 will send the output to standard out. Otherwise, it will send the data to the specified filename. If the file already exists, MVP2.0 will ask the user to confirm overwriting the file before it does so. If the user opts not to overwrite the file, MVP2.0 will again ask for a filename.

Implementation Issues:

In most areas, MVP2.0 will be implemented just as was MVP1.0. This includes how MVP reads files (for the most part, see below) and outputs data, 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 merchant vessel entries by country. For MVP2.0, you are no longer allowed to store the merchant vessel list in arrays while data is being added from a file or removed based on entries in a file – instead, MVP2.0 must use a linked list during those operations. The only time MVP2.0 will use arrays is for sorting and searching.

Note that MVP2.0 must keep the list of merchant vessel data sorted by country name as each entry is read in. This task must be accomplished by performing linear searches to find the appropriate places in the linked list to insert each entry and changing pointers as necessary to accommodate each entry. This is true for both the initial file and any merge files specified by the user.

Similarly, MVP2.0, purge files will be processed by linearly searching each list for each entry to delete and deleting it while retaining the order of each list.

Note that an alternative design to keeping these lists sorted during merging and purging would be to allow the lists to become unordered during these operations, then to sort them afterward. Give an analysis comparing the run times of these alternative designs and include this document in your project submission. Please make this a PDF file and name it "**design.pdf**" in your submission.

The only libraries you may use for this assignment are `iostream`, `iomanip`, `string`, and `fstream` (`#include <iostream>`, `#include <iomanip>`, `#include <string>`, `#include <fstream>`).

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.

Due Date:

You must submit an electronic copy of your source code and design document to the appropriate dropbox in D2L by **11:00 pm CST on Tuesday, 1 November 2016**.