Project 1 – Searching and Sorting

Computer Science 2413 – Data Structures – Fall 2016

Due by 11:00 pm CST on Tuesday, 11 October 2016

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

User Request:

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

Objectives:

1	Use C++ file I/O to read and write files while using C++ standard I/O (`cin`, `cout`) for user interaction, using appropriate exception handling.	*10 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. This class must provide all of the constructor, destructor, accessor, and mutator methods described in Labs 3 and 4.	*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.	*35 points*
5	Efficiently search the data based on the field specified by the user.	*15 points*
6	Integrate appropriate exception handling into classes that implement searching and sorting.	*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 MVP from your labs in several ways. You are encouraged to reuse and build on your code from your labs. In addition to the functionality provided by MVP0.4, your new MVP1.0 will take user input that allows users to specify the files in which data is stored and to sort and search for entries based on various data fields.

Operational Issues:

MVP1.0 will read merchant vessel data files (text files in csv format) via C++ file I/O. The names of the data files will be specified by the user using standard input. When MVP1.0 starts, it will enter a data input loop, prompting the user for the name of a data file. If the user enters the name of an available data file, MVP1.0 will open the file using C++ file I/O and read the data. If the user enters the name of a file that is not accessible, MVP1.0 will report the error to the user and continue in the loop. If the user hits enter without entering anything else, MVP1.0 will exit the data input loop. If no data has been read in when MVP1.0 exits the data input loop, MVP1.0 will exit. Otherwise, MVP1.0 will move on to a data manipulation loop (see below).

The data files will be organized as they were in the labs. However, for this project, the country data may be spread across multiple files. Each data file will be structured the same way as the data file provided for Lab 1, with two header rows at the beginning, one row for totals at the end, and one or more lines of country data in between. When MVP reads in each file, it must make sure the data in that file is internally consistent and alert the user to any errors encountered (as described in Lab 1 and 2). If a country name appears in more than one data file, MVP will retain the data from the file read most recently. As each file is read in, MVP must update its internal totals to reflect the new totals from the additional data.

After reading in and storing all the merchant vessel data from all of the specified files, MVP will enter a data manipulation loop (as mentioned above). In this loop, MVP1.0 will prompt the user with four options: ‘O’ for output, ‘S’ for sort, ‘F’ for find, and ‘Q’ for quit.

If the user selects output, MVP1.0 will send all of its data to standard out, starting with the two header rows, followed by all country data in whichever order it is presently ordered (whether sorted or unsorted), followed by the row showing worldwide totals.

If the user selects sort, MVP1.0 will prompt the user for the data field on which to sort. The user may select any numeric value from 1 to 25 (inclusive), which corresponds to the column number on which to sort. If the user enters an invalid value for the data field on which to sort, MVP1.0 will return to the data manipulation loop. If the user selects to sort by any valid data field, MVP1.0 will sort the data based on that field.

If the user selects find from the data manipulation loop, MVP1.0 will prompt the user for the data field on which to search. As with sort, the user may select any numeric value from 1 to 25 (inclusive) and if the user selects any other value, MVP1.0 will return to the data manipulation loop.

If the user chooses to find by column 1, MVP1.0 will prompt the user for the country name on which to search. If the user hits enter without entering a name, MVP1.0 will return to the data manipulation loop. Otherwise, MVP1.0 will search for the name given.

If the user chooses to find by any other valid field, MVP1.0 will prompt the user for the value on which to search. If the user hits enter without entering a value, MVP1.0 will return to the data manipulation loop. Otherwise, MVP1.0 will ensure that the value entered is valid (can be converted to an integer) and, if it is, search for the given value.

If MVP1.0 finds one or more fields matching a user request, MVP1.0 should display for the user all of the corresponding rows, in the same format as when printing the data on all countries (including the header rows at the top but excluding the worldwide total row at the end).

If the user selects quit from the data manipulation loop, MVP1.0 should display a friendly parting message and exit.

Implementation Issues:

Given that the data is read in from multiple files and country names may be repeated across files but only the last read data for a given country should be retained, MVP1.0 will need to search for each country name before the country data is inserted into the array of data. Consider how to most efficiently maintain the array and search for each country name as its data is read in. **You will create simple design document that explains and justifies your design choices with respect to these issues.** Please make this a PDF file and name it “**design.pdf**” in your submission.

If the data is unsorted or is sorted on a different field from the one on which the user is searching, MVP1.0 will not be able to conduct a binary search for the data. In this case, MVP1.0 should conduct a linear search instead. On the contrary, if the data is sorted on the field on which the user is searching, MVP1.0 should conduct a binary search. Note that you will have to modify the binary search in an important way if the data in a given field is not guaranteed to be unique. **Your design document will**

explain and justify how you modified binary search to account for potential duplicates. Further note that the only data field for which data is guaranteed to be unique is country name.

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

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 Visual Studio project **and your design document** to the appropriate dropbox in D2L by **11:00 pm CST on Tuesday, 11 October 2016**.