Assignment 8 – Read, Sort, and Search

Due: November 25, 2014 at 8:00 AM

Late penalties will be as described in the syllabus.

General Objectives:

Learn how to apply basic sorting and searching algorithms to a set of data
Learn how to read data from a text file

First Things First

In Linux in your …/Documents/Programs folder
Create a folder named ***A08***
This is necessary as you will compress the A08 folder and its contents for submission.

All source code files should have comments with the file name, your name, the last modified date, and a description of the file contents.

Each function should also be appropriately commented in the header and implementation files. The body of each function may need comments for the less than obvious parts of the code (if any).

While unlikely, details of this assignment may be changed based on student feedback while working on it. Typically these details will be syntax/typing error corrections. But always check the posted assignment page and News items on D2L prior to submitting your final work.

SSclass

Create a class called: SSclass
The declaration of the class should be done in a file named: SSclass.h
The definition/implementation of the class should be done in a file named: SSclass.cpp
Save these files in the folder: A08
For this assignment do NOT use std::vector or any std sorting or searching algorithms.

The class description of SSclass can be found in the following UML diagram:

```
SSclass
-------------------------------------------------------------
- m_sourceFileName : string
- mp_data : float*
- m_dataSize : int
- m_dataIsSorted : bool
-------------------------------------------------------------
+ SSclass(string sourceFile)
+ ~SSclass()
+ DataSize() :int
+ SortData() :void
+ FindValue(float findMe) :bool
- LinearFind(float findMe) :bool
- BinaryFind(float findMe) :bool
-------------------------------------------------------------
Friend Functions
  + operator<<(ostream& ostr, const SSclass & ssc) : ostream&
```

The **constructor** should allocate enough memory to hold all the elements in the specified source file. You will need to devise a way to determine the correct size to allocate. Assume there could be from 0 to infinity. Suggest reading the file twice. It should then store the values in mp_data, and set m_dataSize accordingly. The source file should be CLOSED before the constructor returns. The variable m_dataIsSorted should be initialized to false. Other details may need addressed.

The **destructor** should deallocate the dynamically allocated memory and set mp_data to NULL. Other details may need addressed.

The **DataSize** function should return the number of elements currently stored in mp_data (which should equal the number of numbers read from the source file). If the source file was unable to be opened DataSize() should return negative one. If the source file was able to be opened but contained no data then DataSize() should return zero. If the memory needed to store the data was unable to be allocated DataSize() should return negative 2. Other details may need addressed.

The **SortData** function should sort the data from least to greatest numeric value. Do NOT use any built in STL functions. Write your own sort routine, this should be Insertion or Selection Sort, as presented in class and other homework. When complete, the member variable m_dataIsSorted should be set to true. Other details may need addressed.

The **FindValue** function should call LinearFind if m_dataIsSorted is false, otherwise it should call BinaryFind. It should return whether or not the value specified is found (true means it was found). Other details may need addressed.

The **LinearFind** function should implement a linear (aka sequential) search to locate the value specified. It should return true if the value is in mp_data, and false if it is not. Other details may need addressed.

The **BinaryFind** function should implement a Binary Search to locate the value specified. It should return true if the value is in mp_data, and false if it is not. The function should also print out to the screen each index that is examined as it searches for the value. Other details may need addressed.
Example:

Say that mp_data is sorted and is: 11 12 13 14 15 16 17 18 19 20
And the value to find is 12
The indices printed should be: 4, 2, 1 (or something near that depending on truncation/rounding)

The **operator<<** should output:

the source data's filename followed by a colon followed by an endl
followed by the mp_data[0] followed by a space
followed by the mp_data[1] followed by a space
…
followed by the mp_data[*n*] followed by a space

There should be an endl after every 5th number, and after the last number.
Duplicated numbers should be printed as many times as they occur. The last line of output should be: EOD
Example Output:

```
DataFile.txt:
1 2 3 4 5
6 7 7 7 9
11 12 13 14 15
16 17 18
EOD
```

SSclassTester.cpp File

- The main() function used to test your class SSclass should be placed in a file named: SSclassTester.cpp
- This is your place to test your code.
- Create a main() function that uses your SSclass and a test data file.
- The better you test your code, the better your score is likely to be

It is suggested you compile and run the files:

SSclass.cpp, SSclassTester.cpp

To compile the files create a makefile
or compile from the command line:

```
g++  SSclassTester SSclass.cpp SSclassTester.cpp
```

And run the resulting executable.

```
./ SSclassTester
```

Grading

100 points possible

Grading will first be done by an “automatic grading” program, which will award a score to your program(s). They will then be examined by hand and possibly subjected to further automated grading. The better you write your own testing program(s) the more likely you will pass the instructor’s.

Turn-In Directions

Correctly submitting your work is worth 0 points,
but if not done correctly will likely result in nothing to grade.

Preparation

In Ubuntu Linux browse to your A08 folder
Make sure your source code files are in the folder in the appropriate sub-folders
Right click on the A08 folder,
Select: compress…
Set the file name to be A08_*yourlastname*.tar.gz
where *yourlastname* is your last name
Example: if your last name is Gollygee
then the filename would be A08_Gollygee.tar.gz
Press the create button
This should create the file named A08_ *yourlastname*.tar.gz

Submit

Submit the A08_ *yourlastname*.tar.gz file
to the correct course drop box in D2L