Fundamental Concepts: array of structures, string objects, searching and sorting

The point of this assignment is to validate your understanding of the basic concepts presented in CS 1044. If you have trouble implementing the following specification correctly, that may be evidence that either you will have difficulty in this course or that you are not ready to attempt CS 1704. Feel free to consult the online notes for CS 1044 as a reference. Follow the given specification exactly — this assignment will be scored using the Curator automated grading system and deviations will generally be penalized heavily.

SCDB: Simple Computers DataBase

With the slow down in PC sales computer manufacturers need to keep close tabs on their stock. Programs are needed to manage, tabulate and help analyze a simple database of computer system information. The SCDB program will read an input file that will specify computer system information (see file description below). The computer system data file will contain comment lines which must be ignored. The comment lines begin with the pound symbol '#' in column one. The remainder of the file will consist of lines of data, with tab-delimited fields that conform to the format specification:

```
<model name><tab><manufacturer name><tab><price><tab>
<processor><tab><speed><tab><ram><tab><drive size><tab>
<video ram><tab><cache size><tab><cd speed><tab><sound
card><tab><case type><newline>
```

The number of data lines is unknown (although it will never exceed 25), so your program must read until input failure at the end of the file. A sample computer system data file is given later in this specification. You must define an appropriate structured type to store all the information about each computer system.

The program will first read and store the computer system data, into an in-memory database structure, and then process a second input file specifying actions to take on the computer system data. Supported actions include adding a computer system record to the database, deleting a computer system record from the database, sorting the database on various fields (keys), and dumping the database to file. When all the specified actions have been processed, the program will exit.

Input file descriptions and samples:

This program requires the use of two input files. The first file contains the computer system data and will be named `CompSys.txt`. The second file contains a list of actions to be performed on the computer system record database, and will be named `Actions.txt`. Note that, due to the automated testing process, use of incorrect input (or output) file names will result in a score of zero.

A newline character will terminate each input line, including the last. You may assume that all of the input values will be syntactically and type correct, and that they will be given in the specified order.

Initial Computer System File

The second line of the computer system input file will usually be a comment label line that specifies column labels. Each of the remaining non-comment lines will contain twelve data fields, separated by tab characters. The order of the data fields on the line and the type of value in the field are given in the table at the right. When the `CompSys.txt` file is initially input the records must be inserted in ascending order by the Model name.

A sample `CompSys.txt` input file follows.

Computer system data Field Name	Field Contents
Model Name	character string
Manufacturer Name	character string
Price	integer
Processor	character string
Speed (MHz)	integer
RAM (MB)	integer
Drive Size (MB)	integer
Video RAM (MB)	integer
Cache Size (K)	character string
CD Speed	integerX
Sound Card	character string
Case Type	character string

```
#4explorermicro.com systems
#Model     Manufacturer        Price  Processor    Speed  RAM (MB)  Drive (MB)  Video (MB)  Cache (K)  CD   Sound         Case
MT-359AD   4explorermicro.com  717    Pentium III  1000   64        20000       8           512K       50X  3D Soundpro   ATX
MT-359AF   4explorermicro.com  567    Pentium III  1000   128       20000       32          None       50X  3D Soundpro   ATX
MT-359AL   4explorermicro.com  777    Pentium III  1000   64        40000       32          None       50X  3D Soundpro   ATX
MT-359AT   4explorermicro.com  1469   Pentium III  1000   64        60000       32          None       50X  3D Soundpro   ATX
MT-359AN   4explorermicro.com  1699   Pentium III  1000   128       40000       32          None       50X  3D Soundpro   ATX
MT-359AV   4explorermicro.com  1899   Pentium III  1000   128       60000       32          None       50X  3D Soundpro   ATX
```

There will never be two different area records with the same Model in the database at the same time. Each of the other fields may be duplicated within the database. There will be no more than 25 items in the computer system database at any time. Be aware that this limit will be increased in future versions of the system, plan accordingly.

You are **required** to use a statically-allocated array of structures to store the computer system information. Use of pointers, classes, STL templates and/or dynamic memory allocation is expressly forbidden in this assignment.

Note that the alignment of the computer system information in the initial file may not be perfect, because the combination of tabs may not align the numbers/fields correctly. This is a good example of why it is suggested that you use spaces (not tabs) to align your output. The tab character is used for input field delimiting because it makes it somewhat easier to parse the field data.

Database Actions File

Each line of the actions file will contain one of the commands described below. Commands are case-sensitive and take a fixed number of arguments. The command names will be valid and each command will include the correct number of syntactically and type correct arguments. The actions file may also contain comment lines which must be ignored. The comment lines begin with the pound symbol '#' in column one. Command arguments will be tab-delimited.

```
add   <tab><model name><tab><manufacturer name><tab><price><tab>
<processor><tab><speed><tab><ram><tab><drive size><tab> <video ram><tab><cache
size><tab><cd speed><tab><sound card><tab><case type><newline>
```

This causes the insertion of a new computer system record into the database list. Insertion should place the new record in the proper location with respect to the current sort ordering of the list. The initial computer system list will be given in arbitrary order, and you must initially order it by the Model before any processing. If an `add` instruction specifies the <`model`> number of an item that already exists in the list, the list will not be modified. Note that due to page limitations, (not file line size), some of the arguments of the add command description above have wrapped onto the next line. In the actual `Actions.txt` input file they will all be tab separated on the same line. Note: appending the new record to the end of the array and re-sorting is not an insertion operation and will be penalized. If the number of computer system records stored is at the maximum, 25, and an add operation is encountered then the list must not be modified and the add operation will be skipped and never processed.

```
del <tab><model>
```

This causes the deletion of the computer system record for the indicated <`model`> from the database list. If a `del` instruction specifies the number of an item that's not in the list, the list will not be modified.

```
sort <tab><FieldSpecifier>
```

This causes the area list to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `model`, or `price`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You must use the selection sort algorithm.

`dump <tab>`

This causes the computer system information to be printed to an output file named `dbase.txt`. Printing should be in the physical order of the list resulting from the last sort. More than one dump command may exist in the actions file in which case the area listings for each dump command must be appended to the last listing.

There is no guaranteed limit on the number of actions. A small sample `Actions.txt` input file is shown below. Due to page limitations, (not file line size), some of the arguments of the add commands below have wrapped onto the next line. In the actual `Actions.txt` input file they will all be tab separated on the same line.

```
#4explorermicro.com systems: Actions.txt
#COM    ARGS
sort    model
#non-existent delete
del     MT-359AE
#delete first record
del     MT-359AD
#add record back
add     MT-359AD       4explorermicro.com     727     Pentium III    1000    64      20000   8       512K
        50X     3D Soundpro    ATX
#delete last record
del     MT-359AV
#add record back
add     MT-359AV       4explorermicro.com     1999    Pentium III    1000    128     60000   32      None
        50X     3D Soundpro    ATX
sort    price
#delete middle record
del     MT-359AL
#add record back
add     MT-359AL       4explorermicro.com     797     Pentium III    1000    64      40000   32      None
        50X     3D Soundpro    ATX
#attempt to add existing record
add     MT-359AL       4explorermicro.com     999     Pentium III    1000    64      40000   32      None
        50X     3D Soundpro    ATX
sort    model
dump
```

Output files, descriptions and samples:

Your program will produce two output files. The first will be a log file to record the processing of the actions file. The log file must be named `scdblog.txt`. The first line of your log file must include your name only. The second line must include the title "`Simple Computers Database Log`" only. The third line must be a line of underscore characters; the fourth line must display the column labels shown below. The following lines will contain for each command in the `Actions.txt` input file, an integer index numbering of the command, the name of the command and the first argument following the command, followed by a colon and an indication as to the success or failure of the command processing. For failed commands the log must contain a brief comment explaining the failure. (Obviously for dump commands no argument can be output.) For the above `Actions.txt` input file the corresponding `scdblog.txt` would be:

```
Programmer:  Dwight Barnette
Simple Computers Database Log
___________________________________________________________________________________________________
###     COMMAND        ARG            ACTIVITY       COMMENT
001     sort           model:         SUCCESS
002     del            MT-359AE:      FAILURE        Not Found
003     del            MT-359AD:      SUCCESS
004     add            MT-359AD:      SUCCESS
005     del            MT-359AV:      SUCCESS
006     add            MT-359AV:      SUCCESS
007     sort           price:         SUCCESS
008     del            MT-359AL:      SUCCESS
009     add            MT-359AL:      SUCCESS
010     add            MT-359AL:      FAILURE        Present
011     sort           model:         SUCCESS
012     dump                          SUCCESS
```

The second output file will contain the results of the dump command and must be named `dbase.txt` — use of any other output file name **will** result in a runtime testing score of zero. Here is a `dbase.txt` output file corresponding to the given sample input files:

```
Programmer:  Dwight Barnette
Simple Computers Database
____________________________________________________________________________________________________________
Model      Manufacturer        Price    Processor    Speed  RAM (MB)  Drive (MB)  Video (MB)  Cache (K)  CD   Sound         Case
MT-359AD   4explorermicro.com    727    Pentium III  1000      64     20000             8     512K       50X  3D Soundpro   ATX
MT-359AF   4explorermicro.com    567    Pentium III  1000     128     20000            32     None       50X  3D Soundpro   ATX
MT-359AL   4explorermicro.com    797    Pentium III  1000      64     40000            32     None       50X  3D Soundpro   ATX
MT-359AN   4explorermicro.com   1699    Pentium III  1000     128     40000            32     None       50X  3D Soundpro   ATX
MT-359AT   4explorermicro.com   1469    Pentium III  1000      64     60000            32     None       50X  3D Soundpro   ATX
MT-359AV   4explorermicro.com   1999    Pentium III  1000     128     60000            32     None       50X  3D Soundpro   ATX
____________________________________________________________________________________________________________
```

The first line of your output must include your name only. The second line must include the title "`Simple Computers Database`" only. The third line must be a line of underscore characters; the fourth line must display the column labels shown above. The fifth line will contain the computer system data echoed from the current database, aligned under the appropriate headers. The last line of each dump must be a line of underscore characters. The column field headings should be repeated for each display listing resulting from a dump. However, the first three lines (programmer, program title and underscore lines) are not to be repeated.

Computer system Field	Field Output Limit
Model	15
Manufacturer	25
Price	5
Processor	15
Speed	4
RAM	4
Drive Size	6
Video RAM	4
Cache Size	6
CD Speed	5
Sound Card	15
Case Type	10

You are not required to use the exact horizontal spacing shown in the example above, but your output must satisfy the following requirements:

- You must use the specified header and column labels, and print a row of underscore delimiters before and after the table body, as shown.
- You must arrange your output in neatly aligned columns. Use spaces, not tabs to align your output. For output alignment purposes the output field limits in the table at the right may be implemented.
- You must use the same ordering of the columns as shown here.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class, in the course notes and on the course Web site about formatting, structure, and commenting your code should be followed. Some specifics:

Documentation:

- You must include the honor pledge in your program header comment above the `main()` function, (see below).
- You must include a header comment that describes what your program does and specifying any constraints or assumptions of which a user should be aware, (such as preset file names, value ranges, etc.).
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names suggesting the meaning/purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment. This should give the name of the function, explain in one sentence what the function does, describe any special algorithm used/implemented, give a list of the other functions that call this function, give a list of the other functions that this function calls, then describe the logical purpose of each parameter (if any), describe the return value (if any), give your name as the author of the function, and state reasonable pre- and post-conditions and invariants.
- Use the `assert()` function to check for error conditions and verify function pre-conditions, post- conditions and invariant conditions whenever possible.
- You must use indentation and blank lines to make control structures, (loops, if-else statements, etc.) more readable.

You are required to conform to the minimal coding requirements specified below.

Coding:

- Implement your solution without any user-defined classes.
- Implement your solution in a single source file, with no user-defined header files. (This restriction is for ease of testing and evaluation.)
- Use named constants instead of variables where appropriate.
- Use `double` variables for all decimal numbers.
- Use an array of structure variables to store the computer system data.
- Use C++ `string` objects, not C-style `char` arrays to store character strings, (aside from string literals).
- Declare and make appropriate use of an enumerated type in your program.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- You must write at least ten functions, besides `main()`.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes and you may use file-scoped constants. You may also make the `struct` type statement declaration for your structured variable type file-scoped (in fact you must do this).
- You may not use file-scoped, (global), variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed.

Design:

An initial structure chart design of the program is NOT required. However, students may wish to go ahead and produce a structure chart design as all other projects will require structure chart designs and will build off of this project. If a design is produced it is not to be submitted in any manner for evaluation or documentation.

Stepwise Refinement:

For a program of this size it is essential that you practice incremental implementation. That is, don't attempt to write the entire program at once, even though you may have a complete design. Here is a suggested order of implementation:

- Declare the data array and initialize it to hold dummy values. Print it out to verify your work.
- Read the initial model data into the data array. Print out the model data to verify your work.
- Implement reading of the actions file. Initially, don't worry about actually carrying out any of the commands, just find the command word, and any parameters, and echo them to the log file to be sure that you're reading them correctly. Also verify that you're stopping on the end of the file correctly.
- Add a function (or two) to handle the `sort` command. Be careful that you use a selection sort, as specified, rather than another sorting algorithm. Also be careful that you pass the sort function the correct parameters. Verify that you're producing correct results in all the sort cases.
- One by one, add functions to handle each of the other action commands. Check your results for each command thoroughly before proceeding to the next.
    - Add in handling of the `add` command.
    - Add in handling of the `del` command.
    - Add in handling of the `dump` command.

If you get stuck on handling a command, or run out of time, echo the command to the log file and print a message (like: "Command not implemented"). That way you'll at least have a shot at partial credit.

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input files. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

At minimum, you should test your program on **all** the posted input/output examples. The same program that will be used to test your solution generated those input/output examples. You could make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Submitting your solution:

You will submit your source code electronically, as described here. `SCDB` will be subjected to runtime testing by the Curator automated grading system. Your code will also be evaluated by a GTA for adherence to these specification requirements, software engineering principles and good programming practices. Deductions will incurred and applied to your project Curator grade. You will be allowed to make up to five submissions of `SCDB` to the Curator.

Instructions for submitting your program are available in the *Student Guide* at the Curator Homepage:

http://ei.cs.vt.edu/~eags/Curator.html

Read the instructions carefully.

Note well: your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those will be graded and also evaluated by the TAs who will assess a deduction for adherence to the specified programming standards. There will be absolutely no exceptions to this policy! The GTA evaluation deduction will be applied to your highest score from the Curator. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.

Pledge:

Each of your project submissions to the Curator must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator Server.
```

Failure to include this pledge in a submission is a violation of the VT Honor Code.