

Managing an Array of Structures

For this program, you will modify your design and code for Project 9 to organize the computer system data in an array of `struct` variables, rather than a collection of parallel arrays. Note that this will entail some major changes to your implementation. While the design for Project 10 will probably be similar to that of Project 9, you are advised to not simply attempt to edit your Project 9 implementation.

Aside from the addition of two new commands, there will be no changes to the input files from Project 9. As in Project 9, your program will write all its output to a log file. The log file format is changed only in that whenever you print data about a computer you must also print the array index at which its record is stored.

Note: the use of an array of struct variables is required for this assignment. Students who use a collection of parallel arrays, or user-defined class variables instead will be assigned a score of zero.

The computer data file:

The format and name of the computer data file is exactly as specified for Project 9.

The commands file:

The commands, named "Script.txt", file also has the same name and almost the same format as specified for Project 9. For this project, whenever you print the data for a computer you must also print the index at which its record is stored. Two new commands are added:

`sort`

The list of computer system data should be sorted into ascending order on the price field, using the bubble sort algorithm from the course notes.

`manufacturer <manufacturer name>`

For each computer system that is produced by the given manufacturer, print the data for that computer system. If no such computer systems exist in the database, print an appropriate error message indicating that no such computer system were found, (e.g., "Dell systems not found"). Note that the given manufacturer name may occur many times, for different model names, so your search should not stop at the first match.

The log file:

The output file must be named "CompLog.txt". A sample log file is given later in this specification. See the sample for details of formatting.

Aside from changing the project number, and printing confirming messages for the `sort` and `manufacturer` commands, there is only one change to the log file format given in Project 9. Now, whenever you print a computer record you must also print the array index at which it occurs (as shown in the sample log file).

Function requirements for this program:

You must make good use of functions in your implementation. There are many opportunities to do so in this project. For instance, you might use a function to initialize the arrays to hold dummy values, a function to read the initial model data into the arrays, a separate function to carry out each command, a function to determine what the command is, a function to print the data for a model given its index, etc.

Your design must include at least seven user-defined functions, not counting `main()`. For reference, my solution uses ten user-defined functions. It is also important that you choose the appropriate parameter-passing mechanisms, especially when you pass the array to a function. Pass parameters by reference only when it is logically necessary; otherwise, pass parameters by value or by constant reference.

Implementation suggestions:

Before reading the input files, initialize your data arrays to store "No Name Computer" for the model names, "*****" for the manufacturer names, and -1 for the price.

For a program of this size (about 250 lines of C++ code for my solution, not counting any documentation), 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 them to hold dummy values. Print them out to verify your work.
- Read the initial model data into the data arrays. Print out the model data to verify your work.
- Implement reading of the commands file. This should reuse your implementation from project 9, with relatively minor changes to allow for the new commands and the use of an array of structures. 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 exit command.
- Add a function (or two) to handle the `model` command. Verify that you're producing correct results in all the logical cases.
- One by one, add functions to handle each of the other commands from project 9. Check your results for each command thoroughly before proceeding to the next.
- Add in handling of the `manufacturer` command.
- Add in handling of the `sort` command. Be careful that you use a bubble sort, as specified, rather than another sorting algorithm. Also be careful that you pass the sort function the correct parameters.

If you get stuck on handling a command, or just 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 generating the correct number of output lines.

Documentation and other requirements:

You must meet the following requirements (in addition to designing and implementing a program that merely produces correct output):

- Write a header comment with your identification information, the required pledge statement (below), and a brief description of what the program does.
- Write a comment explaining the purpose of every variable and named constant you use.
- Write comments describing what most of the statements in your program do.
- Use descriptive identifiers for variables and for constants.
- Use named constants instead of "magic numbers" whenever it is appropriate. Note: there are some candidates in this category.
- Use `string` variables to store the command strings. The use of `char` arrays and/or `char` pointers is explicitly banned.
- Design and implement functions, as specified above.
- The first function implemented in your source file must be `main()` – so you must provide appropriate prototypes for each of your functions.
- Each function implementation must be accompanied by a header comment that describes what the function does, the logical purpose of each parameter (including whether it is input, output, or input/output), the pre-conditions that a function call assumes and the post-conditions that are guaranteed, the return value (if any), a list of the functions that call this function, and a list of other functions called by this function (if any). An acceptable sample function header is given in the notes.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* can be found at: <http://ei.cs.vt.edu/~eags/Curator.html>

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

Evaluation:

Your submitted program will be assigned a score based upon the runtime testing performed by the Curator System.

The TAs will also evaluate your submission of this program to see whether you followed the documentation and implementation requirements given in this specification. If you do not, your score will be adjusted (downward) accordingly.

Note that this time the TAs will be conducting a careful examination of your submission for internal documentation; you are expected to follow the requirements given in this specification precisely. Your instructor will specify how the TAs scoring of your submission will be counted.

Pledge:

Each of your program submissions 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 System.  
//  
// <Student Name>
```

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

Sample Files

Computer data file:

Professional AP500	Compaq	2337
Dimension 4100	Dell	1609
Dimension 8100	Dell	1808
Precision Workstation 620	Dell	2848
AMD Athlon Thunderbird	Explorer Micro	717
Profile 3cx	Gateway	1999
e-Vectra P2024T	Hewlett-Packard	2619
PowerMate VT300	NEC Computers	1160
PCV-RX270DS	Sony	1478
PCV-RX370DS	Sony	1699

Commands file:

```

sort
manufacturer          Sony
model      Precision Workstation 620
model      Deskpro
model      PCV-RX370DS
model      Professional AP500
update     PCV-RX370DS      1700
update     Deskpro         1260
range      1700            2337
exit

```

Log file:

```

Programmer:  D Barnette
CS 1044 Project 10 Spring 2001
-----
Sorting by price
-----
Looking for computers from:  Sony
  2:  PCV-RX270DS           Sony           1478
  4:  PCV-RX370DS           Sony           1699
-----
Looking for Computer named:  Precision Workstation 620
  9:  Precision Workstation 620   Dell           2848
-----
Looking for Computer named:  Deskpro
Deskpro not found
-----
Looking for Computer named:  PCV-RX370DS
  4:  PCV-RX370DS           Sony           1699
-----
Looking for Computer named:  Professional AP500
  7:  Professional AP500       Compaq        2337
-----
Updating price for:  PCV-RX370DS
  4:  PCV-RX370DS           Sony           1700
-----
Updating price for:  Deskpro
Deskpro not found
-----
Looking for computers between 1700 and 2337:
  4:  PCV-RX370DS           Sony           1700
  5:  Dimension 8100       Dell           1808
  6:  Profile 3cx           Gateway        1999
  7:  Professional AP500   Compaq        2337
-----
Exit command found
-----

```