

Managing Parallel Arrays

For this program, you will set up a very simple database of computer system information and then respond to queries and update commands on that database. The data will consist of model names, manufacturer names, and prices. Your program will maintain an array of model names, a parallel array of manufacturer names, and a parallel array of prices.

There will be two input files this time, one containing the initial computer system data and one containing a script of commands to be processed. Your program will write all its output to a log file.

Here are some guarantees. The number of computer systems in the database will never exceed 100. The data will never include two computer systems with the same model name (not even from different manufacturers). For formatting purposes, you may assume that model and manufacturer names will never be longer than 30 characters.

Note: the use of struct or user-defined class variables is strictly prohibited for this assignment. The point of this project is the management of data in parallel arrays. Students who use an array of struct or user-defined class variables instead will be assigned a score of zero.

The computer data file:

The first input file, named "CompSys.txt", will consist of lines of data that conform to the format specification:

```
<model name><tab><manufacturer name><tab><price><newline>
```

The number of data lines is unknown (although it will never exceed 100), 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.

The commands file:

The commands file, named "Script.txt", will consist of single-line commands. Each of the command lines will begin with a command word, followed immediately by a single tab character, and then by one or two tab-separated values, depending upon the particular command word. The commands will be syntactically correct.

Here is a description of each of the commands your program must recognize and process:

`model` <model name>

If the given model name is found in the model names array, print the data for that model (see output sample for formatting). If not, print an error message indicating the model was not found. Note that the specified model name cannot occur more than once, so your search should stop if a match is found.

`update` <model name> <price>

If the given model name is found in the model names array, reset the price value for that model to the value given in the command and print the updated model data. If the model name is not found, print a message indicating that.

`range` <lower bound> <upper bound>

For each model whose price lies between the given bounds (inclusive), print the data for that model. If no matches are found, print a message indicating that.

`exit`

Stop processing the commands file immediately. This produces only a confirming message.

The number of command lines is unknown; your program must read until an `exit` command is found or an input failure occurs. A sample commands file is given later in this specification.

The suggestions that were made for handling script-driven input in project 7 still apply here. Since this is a more complicated program, it is essential that you design your handling of the commands carefully.

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.

As usual, the output begins with a header identifying the programmer and the assignment. For each command, print a comment describing the command to the log file. Immediately after that, print the specified output from the processing of that command. Delimiting lines, as shown, must be printed to separate successive commands.

Each of the delimiting lines and the command echo lines will be assigned zero points when your output is graded, so it is not absolutely necessary that you follow the sample log file exactly. However, if you omit the delimiting lines and/or the command echo lines, the Curator will become confused and compare the wrong lines when grading. It is important that your phrasing for all the other output lines matches that given in the sample output, both here and on the website.

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 five user-defined functions, not counting `main()`. For reference, my solution uses eight 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:

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 arrays 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 be very similar to the way you handled the script file in project 7. 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. Check your results for each command thoroughly before proceeding to the next.

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:

```

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 9 Spring 2001
-----
Looking for Computer named: Precision Workstation 620
Precision Workstation 620 Dell 2848
-----
Looking for Computer named: Deskpro
Deskpro not found
-----
Looking for Computer named: PCV-RX370DS
PCV-RX370DS Sony 1699
-----
Looking for Computer named: Professional AP500
Professional AP500 Compaq 2337
-----
Updating price for: PCV-RX370DS
PCV-RX370DS Sony 1700
-----
Updating price for: Deskpro
Deskpro not found
-----
Looking for computers between 1700 and 2337:
Professional AP500 Compaq 2337
Dimension 8100 Dell 1808
Profile 3cx Gateway 1999
PCV-RX370DS Sony 1700
-----
Exit command found
-----

```