

Introduction

CS 1044, the introductory programming in C/C++ course, is taught from course notes which are available in PDF format at <http://ei.cs.vt.edu/~cs1044/Notes.html>. The text *Programming and Problem Solving with C++* by Dale, Weems and Headington is also used and should be readily available at the academic bookstores in Blacksburg if you desire an additional reference. The examination for credit will be based on the material covered in the first fourteen chapters of the course notes. Please note that there are many subtle issues of syntax and semantics in the C/C++ language. You are strongly advised to review these notes carefully.

The examination will be given in McBryde 209 between 6:30 and 8:00 on Monday, August 23. In order to take the exam, you must:

- Complete the programming assignment given below, following all specifications.
- Notify mcquain@cs.vt.edu that you intend to attempt credit by examination for CS 1044. Notification must be received no later than 8:00 am Monday, August 23.
- Fill out and submit a credit-by-examination form prior to taking the examination. Forms will be available at your Orientation session, or you may pick one up at the Department of Computer Science office (McB 660) when you return in August. The form must be signed by a departmental representative: Bill McQuain in McB 631. Note that there is a fee of \$30 to attempt the examination.
- No one will be allowed to take the examination before submitting the fee payment receipt and their completed programming assignment.

In order to receive credit-by-exam you must achieve passing scores (70% or better) on **each** of the programming assignment and the examination. **Note: averaging 70% or better will not earn credit.**

Credit by Exam Project for CS 1044:
--

Simple Inventory

The Mall-Wart Mercantile Co needs to track its current inventory. The program will read an input file which will specify the initial inventory (see inventory file description below). The initial inventory will consist of a number of different items; for each item, a product number, a text description, the initial number of units in stock, the cost of one unit and the price of one unit will be specified. You should typedef an appropriate structure variable to store all the information about a particular item.

The program will first read and store the initial inventory data and then proceed the inventory transactions. There are two types of transaction with which the inventory program must deal: sell and buy.

A sell transaction occurs when a customer wishes to purchase a certain number of units of a particular item from the store (e.g., 7 units of item number 1731). The program must look up item 1731 in the inventory list and determine if there are enough units of 1731 in stock to make the sale. If there are, then the number of units in stock should be adjusted to reflect the sale. If there are not, then the sell transaction fails and the number of units in stock does not change. (No partial sales are made; if the customer's full request cannot be met then she/he receives no items. No error message is printed when a sell transaction fails.)

A buy transaction occurs when the Mall-Wart Mercantile management wishes to purchase a certain number of units of a particular item to increase the store's supply (e.g., 8 units of item number 1996). The program must look up item 1996 in the inventory list and update the number of units in stock to reflect the purchase. (Buy transactions always succeed.)

There are some other calculations that must be performed; those are described in the calculations section.

When all the transactions have been processed, the program must print the final inventory list, as described in the output section below.

Input file description and sample:

This program requires the use of one input file with two sections. The first section contains the initial inventory list and the second section contains the transactions. Your program **must** read from a file named `inventory.data` — use of another input file name will generally result in a score of zero.

A newline character will terminate each input line. You may assume that all of the input values will be logically correct (no negative or missing values, for instance), that they will be in the specified ranges, and that they will be given in the specified order.

Initial Inventory Section

The first line of the input file specifies the number of inventory items, a positive integer no greater than 50.

Each remaining line of this section of the input file will contain the following data:

- An item number (SKU), a positive integer in the range 1000 to 9999, followed by a single tab character.
- A description of the item, a character string no more than 25 characters long, followed by a single tab character and zero or more spaces.
- The number of units of this item in stock, a nonnegative integer, followed by a tab and zero or more spaces.
- The price of one unit of this item, a positive decimal value, followed by a tab and zero or more spaces.
- The cost of one unit of this item, a positive decimal value, followed by a newline.

Note that the alignment of the unit numbers in the inventory list is not perfect, because the combination of tabs and spaces may not align the numbers correctly. This is a good example of why we suggest that you use spaces (not tabs) to align your output. Here, we use the tab character because it makes it somewhat easier for you to read the item descriptions.

Use an array of structures to store this table.

The initial inventory section of the input file is followed immediately by a list of transactions, as described below.

Transactions Section

Each line of the transactions section will contain a transaction entry, consisting of three fields:

- a character string specifying the type of transaction. The character string will be followed immediately by a tab character, and then zero or more spaces.
- a positive integer representing the item number, followed by one or more spaces.
- a positive integer representing the number of units to be bought or sold, followed by a newline character.

There is no guarantee that the character string will be valid. Legal values are "sell" and "buy" but there may also be lines with other, illegal, values; those lines should be read but otherwise ignored. The character string for the transaction is guaranteed to be no more than 8 characters long.

There is no guarantee that the item number will correspond to any item in the inventory list. In that case, the transaction should be ignored.

There is no guarantee, for a sell transaction, that the specified number of units is available in inventory. As mentioned before, if there are not enough units in inventory to make the full sale, then no units are sold and the transaction is ignored.

There is no guaranteed limit on the number of transactions.

An example input file appears on the next page:

# items:	19		
3809 Complete iMac system	4	18.42	14.77
4071 Widget Snaffler, Green	12	18.38	14.23
4107 #8-32 Knurled Nut	8	7.29	2.48
4297 11th Century Anasazi Pot	17	14.05	10.78
4377 Small Bevel Square	20	20.01	19.92
1613 Microsoft Office 4.0	9	10.89	7.39
1691 WDC31600 Hard Disk	18	9.16	5.48
1730 Scroll Saw Blade 10-Pack	18	20.48	16.52
1996 2 lb Mallet	1	6.52	1.59
2028 5 lb 3.5" AOL Diskettes	13	16.72	15.97
2029 Rubber Mallet	3	3.50	0.77
2258 80386 CPU, 20MHz	21	13.41	11.28
2471 16MB 80ns SIMM Module	3	6.28	5.79
2720 Project Adhesive, tube	18	7.21	5.41
3045 Stainless Steel Foo Bar	1	3.71	2.10
3055 Humphrey (Beanie), mint	12	19.73	17.65
3212 VAX VMS User Manual set	1	8.40	3.44
3381 Skydiving for Dummies	16	13.96	9.09
3600 Myst	21	21.16	18.18
sell	2258	17	
sell	1996	1	
sell	2471	3	
sell	4107	2	
buy	3600		18
sell	1731	7	
sell	4377	12	
sell	1613	3	
sell	1730	42	
sell	4071	6	
sell	2720	2	
sell	4071	2	
buy	4107		3
buy	2720		16
sell	3809	2	
sell	1730	14	
sell	2720	30	
buy	3600		19
sell	1613	3	
sell	3055	26	
buy	3600		33
sell	1691	16	
sell	4297	7	
sell	1730	1	
sell	4107	3	
sell	3381	10	
sell	3381	4	
sell	3055	7	
buy	4297		5
sell	1996	6	
make	4107	2	
buy	1996		8
sell	2258	10	
sell	3809	26	

What to Calculate:

Your program must output a table showing the final inventory, after all the transactions have been read and processed, if possible. The table format is shown in the output section below. In particular, for each inventory item, you must calculate:

- the total activity on this item; that is, the total change in the number of units in stock. For example, consider item # 1996 in the input example on the next page. The initial supply is 1 unit. There is a sell transaction for 1 unit, another attempted sell transaction for 6 units which cannot be made, and finally a buy transaction for 8 units. So the final supply is 8 units and the total activity is 7 units (one sold and eight bought).
- the number of units in stock after all the transactions.
- the total number of units that were sold or bought as a result of all the transactions.
- the total cost and total price of all units that are in stock after all the transactions.

In addition, you must sort the inventory list before printing it, so that the items are listed in alphabetical order of their descriptions. Modify the selection sort code given in chapter 11 of the course notes to do this, using the `strcmp()` function to compare the character strings.

Output description and sample:

Your program must write its output data to a file named `inventory.out` — use of any other output file name **will** result in a score of zero. Here is the output file corresponding to the given sample input:

```

Programmer:  Bill McQuain
Simple Inventory

Number  Name                      Activity    In Stock   Price      Cost
-----
4107    #8-32 Knurled Nut              -2          6          7.29       2.48
4297    11th Century Anasazi Pot       -2         15         14.05      10.78
2471    16MB 80ns SIMM Module         -3          0          6.28       5.79
1996    2 lb Mallet                    7           8          6.52       1.59
2028    5 lb 3.5" AOL Diskettes        0         13         16.72      15.97
2258    80386 CPU, 20MHz              -17          4          13.41      11.28
3809    Complete iMac system           -2           2          18.42      14.77
3055    Humphrey (Beanie), mint        -7           5          19.73      17.65
1613    Microsoft Office 4.0           -6           3          10.89       7.39
3600    Myst                           70          91         21.16      18.18
2720    Project Adhesive, tube         -16           2           7.21       5.41
2029    Rubber Mallet                   0           3           3.50       0.77
1730    Scroll Saw Blade 10-Pack       -15           3          20.48      16.52
3381    Skydiving for Dummies          -14           2          13.96       9.09
4377    Small Bevel Square            -12           8          20.01      19.92
3045    Stainless Steel Foo Bar         0            1           3.71       2.10
3212    VAX VMS User Manual set         0            1           8.40       3.44
1691    WDC31600 Hard Disk            -16           2           9.16       5.48
4071    Widget Snaffler, Green         -8           4          18.38      14.23
-----
                        197          173       3049.68    2550.02

```

The first line of your output should identify you by name, as shown. The second line should include the title “Simple Inventory” only. The third line should be blank. The fourth and fifth lines should contain the specified column labels and a row of delimiters to mark the top of the table.

Next your output file will contain a table, with a line of output for each inventory item. Each line should contain the item

number, and description, the total activity on that item, the number of units of that item in stock, and the price and cost of that item.

After the last line of the table, print a line of delimiters and then display the total number of units sold or bought, and the total cost and total price of all units of all items that are in stock.

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 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.
- You must use the same ordering of the columns as shown here, and print the dollar amounts with precision two.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in the course notes about formatting, structure, and commenting your code should be followed. See the Elements of Programming Style for additional guidelines.

Documentation:

- You must include the honor pledge specified below in your program header comment.
- 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 that suggest the meaning or 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 explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and state reasonable pre- and post-conditions.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Implement your solution without any user-defined classes — CS 1044 emphasizes strictly procedural design.
- 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 dollar amounts.
- Use an array of structure variables to store the inventory data.
- 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 eight functions, besides `main()`. For reference, my solution uses twelve.
- You should write a function to read the inventory list file and store that data in your array of structures, a function to search the inventory list for an item with a given item number, a function to update the inventory list (which would probably use the previous function), and a function to sort the inventory list before printing it. If you also write two or three output functions, you'll have the required number of functions already.

- 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 `typedef` statement 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.

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. 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 given along with this specification. 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.

What to Turn In:

You are required to submit your solution in a 9x12 envelope, with your name and "CS 1044 Credit by Exam" written on the outside. The envelope must contain the following:

- A printout of your C++ source code.
- A printout of one input file and the corresponding output file that your program generated. You may feel free to use the sample input given above. Additional input/output examples will not be welcome.
- A 3.5" Windows-compatible diskette, labeled with your name, containing your source code file and at least one sample input/output pair.
- A signed hardcopy of the Honor Code Pledge given below.

Honor Code Issues:

The examination on August 23 will be closed book and notes. You will be permitted only the use of pencil, paper, and your mastery of the C/C++ language. In completing the programming assignment, you **may** consult any published C/C++ language references and/or programming guides you like. You **may not** consult other programmers, or program code produced by others, aside from published examples. Consultation of koofers and/or programs written for C/C++ classes here or elsewhere is expressly forbidden. See the CS department statement on koofers in the Survival Guide.

Feel free to e-mail questions regarding the program specification to mcquain@cs.vt.edu. Understand that I am quite busy and cannot guarantee an immediate response.

The Honor Code will be strictly enforced in this credit by examination. Submitted programs shall be considered pledged graded work. All aspects of your work will be covered by the Honor System. Honesty in your academic work will develop into professional integrity. The faculty and students of Virginia Tech will not tolerate any form of academic dishonesty.

Honor Code Pledge:

Your submitted program must be pledged to conform to the Honor Code requirements stated above. 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 the CS 1044 Credit by Examination administrator  
//        (mcquain@cs.vt.edu).  
//  
//      - 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.
```

Programs submitted without this pledge will not be evaluated.