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 good 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 an automated grading system and deviations will generally be penalized heavily.

INV0: Grocery Inventory Management System

This program is an extension of the last program in CS 1044, Summer I, 2001. The INV0 program will read an input file that will specify grocery inventory information (see file description below). The inventory data file will begin with two header lines that should be ignored. The remainder of the file will consist of lines of data that conform to the following format specification:

```
<SKU><tab><description><tab><spaces><retail unit
price><tab><cost unit price><tab><floor
stock><tab><warehouse stock><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 inventory data file is given later in this specification. You must define an appropriate structured type to store all the information about each inventory item.

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

Some advice on handling monetary amounts:

The prices are positive decimal values which you are advised to store these monetary amounts as integers, not as doubles. To achieve precisely correct answers, you should read the dollar amount and the cents amount separately and then store the total price in cents. All internal calculations involving money should operate on cents and produce results that are stored as integers. When the time comes to print a monetary amount, you should compute the correct dollar amount and cents amount, as integers, and print those separated by a decimal point. If you do not do this, some of your answers may differ from the correct results in the last digit.

`struct` variables:

Your design and implementation must treat an inventory item as a "thing", not just as a bunch of separate variables. That means you must design and use a `struct` type. You should have an "item" type that aggregates the components of an item. Your declarations of the `struct` types should be global. Remember that you're declaring a type, not a variable here. Since these types will be used throughout the entire program, including in function parameter lists, the type declarations must be global.

Input file descriptions and samples:

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

Inventory Data Fields	Contents
SKU	character string
Description	character string
Retail	decimal
Cost	decimal
Floor	integer
Warehouse	integer

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

Initial Grocery Inventory File

The first line of the inventory input file is a label line that specifies column labels. The second line is a delimiter line of dashes. On each of the remaining lines will be six 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.

A sample `Inventory.txt` input file is shown below.

```
SKU              Item                              Retail   Cost    Floor   Ware
--------------------------------------------------Price----Price---Stock---Stock
V0011    Cut Green Bean 14.5 oz                    1.05      0.49      73      227
V0013    Cut Green Beans 14.5 oz                   0.75      0.51      68      332
V0017    Lima Beans 15 oz                          0.97      0.49      92      108
V0024    Cut Wax Beans 14.5 oz                     0.69      0.34      47       53
V0043    Chili Hot Beans 15 oz                     0.65      0.29      36       64
F0001    Braeburn apples, 2 pack                   1.79      0.89      16       14
F0002    Braeburn apples, 6 pack                   4.49      2.12      33       27
F0003    Gala apples, 2 pack                       1.89      0.75      26       24
F0004    Gala apples, 6 pack                       4.49      1.99      20       25
F0005    Bananas, green 16 oz                      0.69      0.25      78      112
F0006    Lite Peaches 15 oz                        1.09      0.49      44      134
F0007    Bananas, ripe 16 oz                       0.69      0.20     166      234
F0008    Strawberries 16 oz                        1.49      0.69      72      118
F0009    Blackberries, vine ripened 1/2 pt         4.49      1.99      53      136
F0010    Blueberries, vine ripened 1/2 pt          5.99      2.49      32       68
F0011    Grapefruit, large pink                    0.79      0.29      88      212
```

There will never be two different inventory records with the same SKU, (Stock Keeping Unit), number in the database at the same time. Each of the other fields may be duplicated within the database. There will be no more than 100 items in the grocery inventory database at any time.

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

Note that the alignment of the inventory 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. Here, the tab character is used for input separation because it makes it somewhat easier to parse the item descriptions. The SKU will always be five characters long. Descriptions will be no more than 40 characters long. The prices will be monetary decimal values. The prices are guaranteed to be positive, and the stock values are guaranteed to be non-negative so you don't need to check these for errors.

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 arguments. Command arguments will be tab-delimited.

```
add   <SKU><tab><description><tab><spaces><retail unit price><tab><cost unit
price><tab><floor stock><tab><warehouse stock><newline>
```

This causes the insertion of a new inventory 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 inventory list will be given in arbitrary order, and you must initially sort it by the SKU number before further processing. If an `add` instruction specifies the <`SKU`> number of an item that's already 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 inventory records stored is at the maximum, 100, and an add operation is encountered then the list must not be modified and the add operation will be skipped and never processed.

```
del <SKU>
```

This causes the deletion of the inventory record for the indicated <`SKU`> number 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 <FieldSpecifier>
```

This causes the inventory list to be sorted into ascending order by the specified field. The FieldSpecifier must be one of: `SKU`, or `Item`. If a `sort` instruction specifies an invalid FieldSpecifier, the list will not be modified. You must use the selection sort algorithm, ordering the alphabetically by their description when an `Item` FieldSpecifier is indicated.

```
dump
```

This causes the inventory 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 inventory listings for each dump command must be appended to the last listing.

There is no limit on the number of actions. A small sample `Actions.txt` input file is shown below. Note that due to page limitations, (not file line size). In the actual `Actions.txt` input file they will all be tab separated on the same line.

```
sort    SKU
del     V0010
add     V0011   Cut Green Bean 14.5 oz                  1.05        0.49       73    227
del     F0011
del     V0011
add     V0031   Great Northern Bean 14.5 oz             1.25        0.59       86    324
del     V0031
add     V0031   Great Northern Bean 12.5 oz             1.25        0.59       86    324
sort    Item
add     V0033   Red Kidney Bean 12.5 oz                 1.29        0.55       82    320
add     V0007   Whole Green Bean 14.5 oz                0.75        0.45      177    123
add     F0023   Avocado                                 2.29        1.09       24     23
del     F0023
add     F0023   Avocado                                 2.24        1.09       23     23
sort    SKU
dump
```

Output description and sample:

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

```
Programmer:  Dwight Barnette
Grocery Inventory Management System
____________________________________________________________________________________________________
SKU     Item                                     Retail      Cost    Floor   Warehouse
F0001   Braeburn apples, 2 pack                  1.79        0.89       16      14
F0002   Braeburn apples, 6 pack                  4.49        2.12       33      27
F0003   Gala apples, 2 pack                      1.89        0.75       26      24
F0004   Gala apples, 6 pack                      4.49        1.99       20      25
F0005   Bananas, green 16 oz                     0.69        0.25       78     112
F0006   Lite Peaches 15 oz                       1.09        0.49       44     134
F0007   Bananas, ripe 16 oz                      0.69        0.20      166     234
F0008   Strawberries 16 oz                       1.49        0.69       72     118
F0009   Blackberries, vine ripened 1/2 pt        4.49        1.99       53     136
F0010   Blueberries, vine ripened 1/2 pt         5.99        2.49       32      68
F0023   Avocado                                  2.24        1.09       23      23
V0007   Whole Green Bean 14.5 oz                 0.75        0.45      177     123
V0013   Cut Green Beans 14.5 oz                  0.75        0.51       68     332
V0017   Lima Beans 15 oz                         0.97        0.49       92     108
V0024   Cut Wax Beans 14.5 oz                    0.69        0.34       47      53
V0031   Great Northern Bean 12.5 oz              1.25        0.59       86     324
V0033   Red Kidney Bean 12.5 oz                  1.29        0.55       82     320
V0043   Chili Hot Beans 15 oz                    0.65        0.29       36      64
____________________________________________________________________________________________________
```

The first line of your output must include your name only. The second line must include the title "`Grocery Inventory Management System`" 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 inventory 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.

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.
- You must use the same ordering of the columns as shown here, and print the Price fields to two decimal places.

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, (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 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 and invariants.
- Use the assert function to check for error conditions and verify function pre- and post-conditions whenever possible.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

You are also required to conform to the 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 `integer` variables for all monetary values.
- Use an array of structure variables to store the inventory data.
- Use C++ `string` objects, not C-style `char` arrays to store character strings, (aside from string literals).
- Declare and make appropriate use of enumerated type(s) 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 not use file-scoped variables of any kind, (file-scoped types are allowed).
- 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.

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. `INV0` will be subjected to runtime testing by the Curator automated grading system. You will be allowed to make up to five submissions of `INV0` 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. There will be absolutely no exceptions to this policy! 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. The 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 Honor Code.