INV1: Inventory Management System 1

Fundamental Concepts: data class, dynamic array of objects

The point of this assignment is to give you beginning programming experience with programmer-defined classes and dynamic memory. This project will require you to apply what you have learned about classes and dynamic memory in CS 1704. This assignment will modify and extend INV0. All of the specified functionality of INV0 will be retained. In order to make the program more useful, other action commands will be implemented. The data structure that holds the in-memory inventory database will also be changed; see the section on Dynamic Array Management below for details. INV1 will first load the initial inventory data file as before, creating a dynamic in-memory inventory database array, sort by SKU, and then read and process actions from the `Actions.txt` script file. As with INV0, when all the specified actions have been processed, INV1 will exit.

File Descriptions:

The initial inventory data file and the script actions file will have precisely the same syntax as per INV0, aside from some new actions.

Each line of the actions file will contain one of the commands described in the INV0 specification, or one of the new commands described below. As before, commands are case-sensitive and take a fixed number of tab-delimited arguments. The command names will be valid, and each command will include the correct number of arguments.

`find <SKU>`

This command results in a search being performed to locate the inventory object with the corresponding SKU number. If located, the command will display the `<index>` of the object in the array followed by the `<Item>` member field of the object. The command output must be in the following format made to the output file, `dbase.txt`:

```
Find:    <index>    <Item>
```

If a corresponding SKU object cannot be located, the output of the command must be the following, where `<SKU>` is replaced by the SKU number that was to be found:

```
Find:    <SKU> ***MISSING***
```

`stock <SKU>`

This command also results in a search being performed to locate the inventory object with the corresponding `SKU` number. If located the command will display the sum of the `<floor>` and `<ware>` stock member fields of the object. The command output must be in the following format made to the output file, `dbase.txt`:

```
Stock:   <SKU>       total stock
```

If a corresponding `SKU` object cannot be located the output of the command must be the following, where `<SKU>` is replaced by the `SKU` number that was to be found:

```
Stock:   <SKU> ***MISSING***
```

As previously, for both input files, 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.

Updated sample input files for INV1 will be posted on the course web site soon. When they are available, an announcement will be posted.

Simple Inventory Class & Objects

You are **required** to change your inventory struct type into a simple data class type. This will require the creation of constructor, reporter (get), mutator (set) and summation member function(s). Under no circumstances is the class to contain any dynamic memory allocation. The inventory class will be the only allowed programmer-defined class in the program. Since the class will be a simple data class, containing no dynamic memory, no destructor will be required. In addition, no overloading of the assignment operator will be required. Member functions should be documented the same as non-member functions. See the course notes for a description of how the class member functions are to be depicted on the structure chart design.

Dynamic Array Management

You are **required** to use a dynamically-allocated array of inventory class objects. The dynamic array of inventory objects must **not** be implemented as a class itself. To make the program memory resource efficient, the dynamically allocated array must be dynamically resized during execution as the number of inventory records to be stored grows and shrinks. You will initially allocate an array capable of holding exactly 5 inventory record objects. If the list outgrows the current array size, you will dynamically enlarge the array to hold exactly 5 additional inventory record objects. If the number of unused array locations grows to 10, you will dynamically shrink the array to hold 5 fewer records (allowing some slack space for future growth). Unlike before, if the script actions file contained add commands that would overflow the array capability the commands were ignored. However such a situation will now cause the array to dynamically re-size itself and grow.

This crudely mimics the behavior of a C++ vector object. You are specifically forbidden to use any C++ vector objects or any other any STL templates in this program, or to use a linked list of any type in place of the specified array! Violating that restriction would remove one of the major points of this assignment and will certainly result in a major deduction.

Input File Samples:

Initial Inventory File

The format of this file is unchanged from INV0. A sample `Inventory.txt` input file is shown below.

SKU	Item	Retail Price	Cost Price	Floor Stock	Ware 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

Database Actions File

There is no guaranteed limit on the number of actions. The changes to this file have been discussed previously, (see the File Descriptions section above). A small sample `Actions.txt` input file is shown below.

```
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
find    V0011
find    F0011
find    V0043
find    F0010
stock   F0023
stock   V0010
stock   V0031
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 the output file corresponding to the given sample input files:

```
Programmer:  Dwight Barnette
Grocery Inventory Management System
_____________________________________________________________________________________
Find:        V0011      ***MISSING***
Find:        F0011      ***MISSING***
Find:           17      Chili Hot Beans 15 oz
Find:            9      Blueberries, vine ripened 1/2 pt
Stock:       F0023         46
Stock:       V0010      ***MISSING***
Stock:       V0031        410
_____________________________________________________________________________________
 Inv    SKU     Item                                  Retail      Cost    Floor   Warehouse
  0.    F0001   Braeburn apples, 2 pack                1.79        0.89       16      14
  1.    F0002   Braeburn apples, 6 pack                4.49        2.12       33      27
  2.    F0003   Gala apples, 2 pack                    1.89        0.75       26      24
  3.    F0004   Gala apples, 6 pack                    4.49        1.99       20      25
  4.    F0005   Bananas, green 16 oz                   0.69        0.25       78     112
  5.    F0006   Lite Peaches 15 oz                     1.09        0.49       44     134
  6.    F0007   Bananas, ripe 16 oz                    0.69        0.20      166     234
  7.    F0008   Strawberries 16 oz                     1.49        0.69       72     118
  8.    F0009   Blackberries, vine ripened 1/2 pt      4.49        1.99       53     136
  9.    F0010   Blueberries, vine ripened 1/2 pt       5.99        2.49       32      68
 10.    F0023   Avocado                                2.24        1.09       23      23
 11.    V0007   Whole Green Bean 14.5 oz               0.75        0.45      177     123
 12.    V0013   Cut Green Beans 14.5 oz                0.75        0.51       68     332
 13.    V0017   Lima Beans 15 oz                       0.97        0.49       92     108
 14.    V0024   Cut Wax Beans 14.5 oz                  0.69        0.34       47      53
 15.    V0031   Great Northern Bean 12.5 oz            1.25        0.59       86     324
 16.    V0033   Red Kidney Bean 12.5 oz                1.29        0.55       82     320
 17.    V0043   Chili Hot Beans 15 oz                  0.65        0.29       36      64
 20>   MAX INVENTORY STORAGE
_____________________________________________________________________________________
```

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 first three lines (programmer, program title and underscore lines) are not to be repeated.

The `dump` command will be modified slightly. Each inventory record output will be numbered starting at zero. At the end of the inventory record table listing, the current size of the dynamic array will be output. Note that this is not the same as the number of objects stored in the array. Note well that the `Actions.txt` file is not required to end in a dump or save command and multiple dump commands may exist in the file. The first and last line of each dump must be a line of underscores. The second dump line must display the column labels shown above. The third dump line will contain the inventory data echoed from the current database, aligned under the appropriate headers. The column field headings should be repeated for each display listing resulting from a dump.

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.

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 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 an array of objects 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 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.
- 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 `class` declaration statement for your inventory class type file-scoped (in fact you must do this).
- You may not use file-scoped 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. Pointers should be passed by reference, `const` pointer and/or `const` target as appropriate.

Interim Design:

You will produce an interim design for INV1, and represent that design in a modular structure chart. The structure chart must indicate your design plans for INV1. It is expected that your final design will differ from the interim design. Nevertheless, your interim design should be relatively complete. If the interim design is incomplete or if the differences between your interim design and your final code are excessive, you will be penalized. That means that you should take the production of the interim design seriously, but **not** that you should avoid changes that would improve your final implementation of INV1. You must submit this interim design, to the Curator System, no later than midnight Monday, July 16, (i.e. prior to July 17). Submit a MS Word.doc file. Do **NOT** compress, (zip), the interim design submission!

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. 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. `INV1` will be subjected to runtime testing by the Curator automated grading system. You will be allowed to make up to five submissions of `INV1` 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.