INV2: Inventory Management System 2

Fundamental Concepts: *separate compilation, elementary operator overloading, dynamic linked-list objects*

This assignment will modify and extend INV1. Separate compilation is required for this assignment. Each user-defined class must have its own source code and header files. Other compilation units should be created that reflect the modular decomposition design of the system. Related system sub-sections should be grouped into separate compilation units.

Most of the previous functionality of INV0 and INV1 will be retained, unless explicitly removed or modified in this specification. There are two major extensions.

First, the INV1 inventory data structure will be changed. INV2 will use a **single linked-list** of dynamically allocated nodes to store multiple inventory objects. In order to receive full credit, the list must be fully encapsulated using a C++ class, and the list nodes must themselves be implemented using a node class.

Additionally in order to make the program slightly more useful, you will add the ability to save the in-memory inventory database to a file on disk, and to load an inventory database that was previously saved by your program; see the section on the input file descriptions below for details.

INV2 will normally be invoked from the command-line, and the names of the input files will be specified on the command-line, as:

```
inv2 <InitialInventoryFileName> <InventoryActionsFileName>
```

INV2 will first load the initial inventory data file, creating an in-memory database structure, and then read and process actions from the database actions file. As with INV0 and INV1, when all the specified actions have been processed, INV2 will exit.

Input File Descriptions:

The initial inventory data file and the script actions file will have the same syntax as per INV0/INV1, aside from two new actions. Note that use of hard-coded names for these files will annoy the person evaluating your program, and you **will** be charged points for that annoyance.

The format of the inventory database file, created by the `save` command, is not specified. Part of your assignment is to design a sensible layout for this file. Note that this means that programs from two different students may certainly have incompatible database file formats, and so will not be able to load inventory database files created by another INV2 program. The only restrictions imposed on your inventory database file design are that you should not waste too much space and that the file must be an ASCII text file.

Each line of the actions file will contain one of the commands described in the INV0/INV1 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.

```
save <DatabaseFileName>
```

This causes the creation of an inventory database file on disk, in the current directory. The default extension for the file, "`.inv`" should be automatically appended to the name. As stated above, the format of this file is up to you, subject to light restrictions. Saving does not clear the current in-memory database.

```
load <DatabaseFileName>
```

This causes the reading of the named, (previously saved) inventory database file, and the creation of a new in-memory database holding the information from that file. The default extension for the file, "`.inv`" should be automatically appended to the name for opening. Any previous in-memory database should be properly deallocated, (but not automatically saved to disk), before the file is read. If the named file does not exist, the following error message must be written to the dump file, “`dbase.txt`” out:

```
***Fatal Error***: <DatabaseFileName> does not exist!
```

The database filename specified in the load command must be substituted for `<DatabaseFileName>` in the above message. At this point, your program should **gracefully** shutdown. If the named file does exist, any previous in-memory database should be properly deallocated, (but not automatically saved to disk), before the file is read.

`find <SKU>`

The `Find` command will change slightly. `Find`, as previously, results in a search being performed to locate the inventory object with the corresponding SKU number. If located, the command will display the `<position>` of the object in the linked-list followed by the `<Item>` member field of the object. The head node of the list is considered to be at position zero. The command output must be in the following format made to the output file, `dbase.txt`:

```
Find:     <position>  <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***
```

The `sort` command will no longer be a valid command. Instead of sorting you are required to always maintain the linked-list of inventory objects in ascending, (smallest – largest) sorted order by the SKU numbers. Only ordered inserts into the inventory database linked-list will be performed. Do not assume that the initial `Inventory.txt` file will be ordered.

Note that if you do not properly implement the `save` command, there will be absolutely no way to test your implementation of the `load` command. 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 INV2 will be posted on the course website soon. When they are available, an announcement will be posted on the course Web site.

Input File Samples:

Initial Inventory File

The format of this file is unchanged from INV0/INV1. 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
```

Database Actions File

As previously, there is no 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.

```
save   init
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
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
load   init
find   V0011
find   F0011
find   V0043
find   F0010
stock  F0023
stock  V0010
stock  V0031
dump
```

Output description and sample:

As before, output data resulting from a `dump` command must be written to a file named `dbase.txt` — use of any other output file name **will** result in a point deduction. The format of `dump` output should be the same as previous programs, except that the `xx> MAX INVENTORY STORAGE` output at the end of database dump will no longer be included. Sample output files will also be posted on the course website shortly. Here is the output file corresponding to the given sample input files:

```
Programmer:  Dwight Barnette
Grocery Inventory Management System
___________________________________________________________________________________________________
Find:           11      Cut Green Bean 14.5 oz
Find:           10      Grapefruit, large pink
Find:           15      Chili Hot Beans 15 oz
Find:            9      Blueberries, vine ripened 1/2 pt
Stock:       F0023      ***MISSING***
Stock:       V0010      ***MISSING***
Stock:       V0031      ***MISSING***
________________________________________________________________________________________
 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.   F0011  Grapefruit, large pink                       0.79      0.29      88        212
 11.   V0011  Cut Green Bean 14.5 oz                       1.05      0.49      73        227
 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.   V0043  Chili Hot Beans 15 oz                        0.65      0.29      36         64
___________________________________________________________________________________________________
```

Linked List:

You are **required** to use a linked list to store the Inventory information. Your implementation of this list will be examined. In order to receive full credit, you must implement the nodes using a well-designed node class and the linked list itself must be encapsulated using a well-designed list class. In addition, the Inventory must be encapsulated so that the data they store is logically separated from the node pointers.

Note this mimics the behavior of a C++ STL (Standard Template Library) list object. You are specifically forbidden to use any C++ STL list objects in this program, or any other sort of predefined dynamic list type. You may base your implementation on the linked list implementations shown in the textbook, or those shown in the notes and lectures. (The lecture notes implementation is recommended, as the Carrano implementation has some serious shortcomings.) You may not use linked list code from any other source. Advanced C++ OOP constructs and concepts, (e.g., inheritance, virtual functions, templates, etc.), not covered in CS 1704 are also **not** to be used in this program. Violating these restrictions would remove major concepts of this assignment and will result in a large deduction.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the requirements for documentation and coding given in the INV0/INV1 specifications are still in effect. In addition:

Documentation:

- You must describe the purpose of each of your classes in a header comment that precedes the class declaration.
- You must document each data member and function member of your classes, both in the header file containing the class declaration and in the corresponding source file containing the implementation. The header file does not have to contain full documentation for each function, but the purpose of each function should be described there.

Coding:

- You must separate the interface of each of your classes from its implementation by placing each class declaration (interface) in its own header file and the implementation of that class in a corresponding source file. The name of the class should be used as the name of the header and source files.
- You must protect access to your data by making **all** data members of your classes private.
- When a node is removed from your linked list, you must dispose of it properly by using `delete`.
- When you discard your Inventory database list to load an existing one from a file, you must `delete` every node in the old list so that your program does not waste memory.
- Memory leaks that might affect the execution or functionality of your program will be penalized. In fact, you should attempt to avoid memory leaks altogether.

Interim Design:

You will produce an interim design for INV2, and represent that design in a modular structure chart. The structure chart must indicate your design plans for INV2. 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 INV2. You must submit this interim design, to the Curator System, no later than midnight Tuesday, July 31. 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.

Deliverables:

Your final project submission must include the following (and absolutely no other files):

- all source code (`*.cpp` and `*.h` files) comprising your project
- The MS Visual C++ project files (.dsp and .dsw). Do **NOT** submit the debug/release project subdirectories or an executable.exe file.
- a revised design document reflecting the final design of your project, at the time of submission; this must either be in a format that can be viewed in MS Word or be a PDF file.
- one set of input files, named `Inventory.txt` and `Actions.txt`, and the corresponding final dump `dbase.txt`, and the corresponding saved database file, named `INV2.inv`.
- a brief ASCII text readme file, named `readme.txt`, with any special execution instructions

Submissions will be archived, but not scored, by the Curator System. You will submit your project as an archive file in one of two formats. For Windows users, submit a zipped archive containing the items listed above. (The shareware program WinZip is very easy to use and is available from the Computing Services website: `http://www.ucs.vt.edu/`).

Note that omitting files from your archive is a classic error. Once you've created your project archive, copy the file to a new location, unzip it, attempt a build and then test the resulting executable. Submitting an incomplete copy of your project may delay its evaluation and **will** result in a substantial loss of points. In particular, if you omit a source file necessary to compile your program, you **will** be allowed to supply that file; however, we will then apply a **late penalty** corresponding to the date that you have provided a complete copy for evaluation. There will be **no exceptions** to this penalty. **Also note** that including unnecessary files is also a classic error. Visual C++ users: do not zip up the **debug** subdirectory!

Submitting your project archive:

You will submit your project archive to the Curator System, as described above. All submissions for `INV2` must be made by midnight Monday August 6th. There will be **NO late submissions** accepted for `INV2`. `INV2` will be subjected to runtime testing by the TAs, who will also score your implementation for adherence to the specified programming standards. Demonstration time slot signup forms will be made available in the CS lab. An announcement will be posted when they are available. Students will only be allowed to schedule and perform only one demonstration. TA/student demonstration assignments will be posted on the course Web site. You will be allowed to make up to five submissions of `INV2` to the Curator. Note well: **your last submission will be graded.** There are no exceptions to this policy! Changes made to code during a demonstration will be heavily penalized.

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

Each of your project submissions to the Curator System 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 of the `cpp` file containing `main()`:

```
//    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.
```