Inventory Inquisitor (InvInq)

Professor Caleb Fowler

January 17, 2021

Problem.

Your significant other has had a flash of inspiration - the world needs another inventory management program! You agreed to write it for them because love is indeed blind. Table 1 has the minimum fields you will need to output as well as the validation rules for Specification B4.

Table 1: Data dictionary for Inventory data structure.

Field Name	**Validation Rules**
Item Id Number	5 digits only
Quantity at Hand	Cannot be negative
Wholesale Cost	Cannot be negative
Retail Cost	Automatically calculate 100% markup
Date Added to Inventory	Use 9/9/99 for all instances

Requirements.

- Do not optimize for computer performance.
- Global variables in any form are forbidden. This includes `#define` statement's, too. You can, and should, put classes and structs right under your include statements and using namespace std; statement.
- Put a Source File Header at the top of your source file.
- Include a `void ProgramGreeting()` function. This will run automatically once when the program first starts. This function should display (on individual lines):
  - A welcome message.
  - Your name (as author).
  - The date this assignment is due. Format this as MonthName day, year. Example: June 30, 1988.
- Do not your c (.h) style libraries. Use c++ libraries instead.
- You cannot use any data structures from the standard template library. No <vectors> or <arrays> or anything like that.
  You can use C-Style arrays, however. You can also build your own dynamic memory structures.
- You are in an advanced programming class. I expect you will fix all compiler warnings before you submit your code.

Check out Chapter 16 if you are unsure what the Standard Template Library is.

Specifications.

☐ // Specification C1 - Alpha Menu
Create an alphabetic menu similar to figure 1. Enter a letter to indicate your selection.
- Add Inventory - Add new elements to the end of your inventory array. You will need to resize your dynamic array for this to occur.
- Delete Inventory - Remove the last element of your inventory array and shrink it.
- Edit Inventory - Enter a record index number and allow the client to change the values. You do not need to edit the data values as those are fixed for this assignment.
- Display Inventory - Display all the records in your inventory array.
- Quit Program - exit this program.

Main Menu.
<A>dd Inventory
<D>elete Inventory
<E>dit Inventory
<D>isplay Inventory
<Q>uit Program
Figure 1: Allowable menu options for this assignment.

☐ // Specification C2 - Dynamic Array
Create an array on the heap. Initialize it to whatever size you feel you will need to make this program run. This is your main data structure. You can either use one array with structs or classes or multiple parallel dynamic arrays.

☐ // Specification C3 - Resize Array - add to end
Allows you to add records to the end of the inventory data structure you created. This is activated with the first menu option above.

☐ // Specification C4 - Resize Array - subtract from end
Remove elements from the end of your inventory structure when you select the second menu option. Shrink your array accordingly.

☐ // Specification B1 - `Date` class
Create a date class to handle all date related tasks. Figure 2 indicates the minimum elements you'll need.

Date
- Month: integer
- Day: integer
- Year: integer
+ Date(): —Uses system date.
+ Date(mm, dd, yy): —User specified date
+ Display() or Operator<<: string or ostream —cout mm/dd/yy format
+ ClassTest(): void —Activates class test code.

Figure 2: UML Class Diagram for the Date class.

☐ //Specification B2 - `ComponentTest` method in `Date`
Add a method in your `Date` class which allows the class to perform auto diagnostics. In an ideal world, you would use this method to test all the critical components of your class. That way, you can prove you class will perform as advertised and you will have an early warning if some changes you made introduced errors. Run this method after you call your `ProgramGreeting()` because I will be looking for the test results. You need a minimum of 2 tests for the entire `Date` class.

☐ // Specification B3 - Menu Input Validation
Only allow the first letter of each line on the main menu as valid

inputs. Obviously, you will need to make sure you don't have two lines which start with the same letter.

- ☐ `// Specification B4 - Inventory Entry Input Validation`
  Apply one of the validation rules from Table 1 to either Add new item or Edit item (Spec A1) menu selections. Reprompt if incorrect input. You do not need to use a string to handle this input.

- ☐ `// Specification A1 - Edit Inventory`
  Add the capability to edit the contents of any inventory item. Edit is different from delete. You may apply input validation from Specification B4 if you wish.
- ☐ `// Specification A2 - Overload operator«`
  Overload the stream extraction operator somewhere in your program. I'm guessing the Inventory struct is the easiest place to do this.
- ☐ `// Specification A3 - Overload operator»`
  Overload the stream insertion operator somewhere in your program. I'm also guessing the Inventory struct is the easiest place to do this too.
- ☐ `// Specification A4 - UnitTest() method in main()`
  This is a method where you call all the `ComponentTest()` methods from all the classes you use. You also put tests in here to verify your classes do indeed, work together. In this assignment, use this method to call your various `ComponentTest()` methods. You should have a `ComponentTest()` method in every class you create.

Rubric.

Element	**Points**
Meets all elements in requirements section above	25
Each of the 4 'C' Specifications	5 (20)
Bonus for all 'C' Specifications	5
Each of the 4 'B' Specifications	5 (20)
Bonus for all 'B' Specifications	5
Each of the 4 'A' Specifications	5 (20)
Bonus for all 'A' Specifications	5
Total Points	100

Due Date.

This assignment is due by 11:59 PM on Sunday on the date on the calendar in Canvas. All the assignments are open the first day of class and you can begin working on them immediately. I encourage you to start sooner rather than later with your homework, these always seem to take longer than you think.

Late Work.

If you miss the due date and time specified above, your work is late. Canvas records the date and time your homework upload COMPLETES. Late work is still acceptable, but it suffers a 1 letter grade penalty. You may turn late work in up until MONDAY 11:59 PM AFTER THE ASSIGNMENT WAS DUE. That is, you have 1 day to turn your work in - after that the Canvas drop box closes. Once Canvas closes I will not accept an assignment.

Pro-Tip: Get a bare bones copy of your code running and turn it in. Then go ahead and modify it, fix it and whatnot. Upload it with the same name when you finish. That way, if something unexpected happens, you have some working code turned in. Risk management, class, risk management. [1]

[1] If you really want to go pro, get some sort of version control system running (like Git).

How to Turn in your Homework.

Follow these guidelines for turning in your work:

1. Each assignment has a submit button to turn in your work.
2. Upload your .cpp file, but make sure you change the file extension to .txt. The plagiarism checker doesn't read .cpp files and I've disabled uploading them. I'll rename your homework to .cpp when I download them.
3. Only submit your homework through Canvas. Do not email them to me or attach them in the comments.
4. Submit a source file I can compile. Do not submit a link to an online editor like repl.it. Do not zip a file. Submit self contained programs - they only have one file to run (no data files, config files, object files, etc.).
5. It is OK to turn in multiple versions of the same file (that you've updated. Canvas will automatically download the latest version.
6. It is your responsibility to make sure your file is readable. If your file is 'corrupted' I will not accept it and you will have failed to turn in that assignment. I expect to be able to see your code in Canvas as well as compile and run it on my computer using Ubuntu.
7. Let me know if you are having any trouble uploading your homework immediately.
8. I download your files, compile and execute them as well as look over your source code.

Style Guide

All programs you write MUST conform to the following style specifications.

Comments.

Use white space and comments to make your code more readable. I run a program called cloc (count lines of code) which actually looks for this stuff.

End of line comments are only permitted with variable declarations. Full line comments are used everywhere else.

Comment Rate.

I use a program called cloc to calculate the total number of blank lines, total number of comments and the total number of lines in your program. When you divide the number of comments by the number of lines you get the comment rate. This number gives me a rough approximation of how well commented your code is. When I see a rate of 10% (for example) I have a good idea of what I will find when I look at the program - pretty much no comments at all. Ditto, when I see a rate of 45% (for example). This code will have comments for many tricky or complex sections as well as functions.

Specification Comments.

Specifications are bundled into groups: "A", "B", "C", "D". You must meet the specifications of the lowest group before I will count the specifications for the highest group. For example, you must meet the "D" specifications before I will count the "C" specifications. If you miss one element of a specification bundle, that is the grade you will get for the assignment - regardless of how much extra work you do.

Use whole line comments for Specifications. Put the comment on the line above the start of the code implementing the Specification. If the same Specification code appears in more than 1 place, only comment the first place that Specification code appears. Number your Specifications according to the specification bundle and the specific specification you are using, also provide a very short description. DO NOT BUNCH ALL YOUR SPECIFICATIONS AT THE TOP OF THE SOURCE FILE. Example specification comment:

```
// Specification A2 - Display variables
Your code to do this starts here;
```

It's very important to get the specifications down correctly. If your specification code isn't commented, it doesn't count. I use the grep

trick to find your specification code. Proper documentation is part of the solution, just like actually coding the solution is.

Compiler Warnings.

Compiler warnings are a potential problem. They are not tolerated in the production environment. In CISP 360 you can have them. I will deduct a small number of points. CISP 400 - I will deduct lots of points if compiler warnings appear. Make sure you compile with -Wall option. This is how you spot them.

C++ Libraries.

We are coding in C++, not C. Therefore, you must use the C++ libraries. The only time you can use the C libraries is if they haven't been ported to C++ (very, very rare).

Functions.

Functions are used to segment your code into easier to work with chunks. You want your functions to do only one thing - one activity. Functions are coded BELOW main(), not above it. Use arguments and parameters to pass information to your functions - global variables are discouraged with prejudice . Whenever possible, try to have a brief comment below the function signature describing what the function does.

```
  void foo()
  // a meaningless function to show a function comment in
the style guide.
```

Function Prototypes.

Use function prototypes and comment them. This is a constraint in this class even if not expressly stated in the homework. I use the grep trick for Function Prototype to look for this. You only need to comment it once, at the top of your source file - above main(). Example:

```
  // Function Prototype
  void ProgramGreeting();
```

Non-Standard Language Extensions.

Some compilers support unapproved extensions to the C++ syntax. These extensions are **unacceptable.** Unsupported extensions are compiler specific and non-portable. Do not use them in your programs.

Program Greeting.

Display a program greeting as soon as the program runs. This is a description of what the program does. Example:

```
// Function Greeting
cout « "Simple program description text here." « endl;
```

You can make this much more elaborate - usually used as cover so clients don't realize we are loading huge data files. If your assignment calls for a menu, DO NOT put the menu in here - that goes in it's own section.

Source File Header.

Start your source file with a program header. This includes the program name, your name, date and this class. I use the grep trick for .cpp (see below) to look for this. I focus on that homework name and display the next 3 lines. Example:

```
// homeworkname.cpp
// Pat Jones, CISP 413
// 12/34/56
```

Space Rate.

I use a program called cloc to calculate the total number of blank lines, total number of comments and the total number of lines in your program. When you divide the number of blank lines by the total number of lines you get the space rate. This number gives me a rough approximation of how well laid out your code is. When I see a rate of 10% (for example) I have a good idea of what I will find when I look at the program - wall of words. Ditto, when I see a rate of 45% (for example). This code will have good spacing between major blocks of code and functions. Note: when this number get's too high (like 70% of so) the blank space becomes a distraction.

Variables.

Document constants with ALLCAPS variables and the const keyword. Magic numbers are generally frowned upon.

Grep Trick.

Always run your code immediately before your turn it in. I can't tell you how many times students make 'one small change' and turn in broken code. It kills me whenever I see this. Don't kill me.

You can check to see if I will find your specification and feature comments by executing the following command from the command line. If you see your comments on the terminal, then I will see them.

If not, I will NOT see them and you will NOT get credit for them. The following will check to see if you have commented your specifications:

```
grep -i 'specification' homework.cpp
```

This will generate the following output. Notice the specifications are numbered to match the specification number in the assignment. This is what I would expect to see for a 'C' Drake assignment. Note the cd Desktop changes the file location to the desktop - which is where the source file is located.

```
calebfowler@ubuntu:~$ cd Desktop
calebfowler@ubuntu:~/Desktop$ grep -i 'specification' cDrake.cpp
    // Specification C2 - Declare Variables
    // Specification C3 - Separate calculation
    // Specification C1 - Program Output
calebfowler@ubuntu:~/Desktop$
```

This is what I would expect to see for an 'A' level Drake assignment.

```
calebfowler@ubuntu:~/Desktop$ grep -i 'specification' aDrake.cpp
{   // Specification C2 - Declare Variables
    // Specification C3 - Separate calculation
    // Specification B1 - Calculation
    // Specification C1 - Program Output
    // Specification B 2 - double and half
    // Specification A1 - Output Headers
    // Specification A2 - Display variables
calebfowler@ubuntu:~/Desktop$
```

We can also look at the line(s) after the grep statement. I do this to pay attention to code segments.

```
grep -i -C 1 'specification' aDrake.cpp
```

```
calebfowler@ubuntu:~/Desktop$ grep -i -C 1 'specification' aDrake.cpp
int main()
{   // Specification C2 - Declare Variables
    int r_starcreation = 7;              // rate of star creation
--

    // Specification C3 - Separate calculation
    float drake = 0;                     // initialize to 0
    // Specification B1 - Calculation
    drake =  r_starcreation * perc_starswithplanets * ave_numberofplanetslife *
perc_devlife * perc_devintlife * perc_comm *  exp_lifetime;

    // Specification C1 - Program Output
    cout << "The estimated number of potential alien civilizations in the univer
se is ";
--

    // Specification B 2 - double and half
    cout << "Half this value: " << drake * .5 << endl;
--

    // Specification A1 - Output Headers
    cout << endl;
--

    // Specification A2 - Display variables
    cout << "Variables:" << endl;
calebfowler@ubuntu:~/Desktop$
```

We can also use this to look for other sections of your code. The

grep command searches for anything withing the single quotes ”, and the -i option makes it case insensitive. This is how I will look for your program greeting:

```
calebfowler@ubuntu:~/Desktop$ grep -i -C 1 'greeting' aDrake.cpp

    // Program Greeting
    cout << "This program calculates and displays the number of potential";
calebfowler@ubuntu:~/Desktop$
```

The grep trick is extremely powerful. Use it often, especially right before you turn in your code. This is the best way I can think of for you to be sure you met all the requirements of the assignment.

Client System.

Your code must compile and run on the client's system. That will be Ubuntu Desktop Linux, version 18.04. Remember, sourcefile.cpp is YOUR program's name. I will type the following command to compile your code:

```
g++ -std=c++14 -g -Wall sourcefile.cpp
```

If you do not follow this standard it is likely I will detect errors you miss - and grade accordingly. If you choose to develop on another system there is a high likelihood your program will fail to compile. You have been warned.

Using the Work of Others.

This is an individual assignment, you may use the Internet and your text to research it, but I expect you to work alone. You **may** discuss code and the assignment. Copying code from someone else and turning it in as your own is plagiarism. I also consider isomorphic homework to be plagiarism. You are ultimately responsible for your homework, regardless of who may have helped you on it.

Canvas has a built in plagiarism detector. You should strive to generate a green color box. If you submit it and the score is too high, delete it, change your code and resubmit. You are still subject to the due date, however. This does not apply if I have already graded your homework.

Often, you will not be able to change the code to lower the score. In this case, include as a comment with your homework, what you did and why you thought it was ineffective in lowering your score. This shows me something very important - you are paying attention to what you are doing and you are mindful of your plagiarism score.

ProTip: Get a bare bones copy of your code running and turn it in. Then go ahead and modify it with bonuses and whatnot. Upload it with the same name so it replaces your previous homework. This way, if something comes up or you can't finish your homework for some reason, you still have something turned in. A "C" is better than a zero. Risk management class, risk management.