Programming Assignment #2

Manage a Small Store Inventory

CS 2308.251, 252, and 257 Spring 2016
Instructor: Jill Seaman

Due: Wednesday, 2/17/2016: upload electronic copy by 9:00am

Problem:

Write a C++ program that will allow a user to manage the inventory of a small store.

The inventory for the small store will contain the following information for each product in the inventory:

product name	(i.e. "Apple iPhone 3GS 8GB", may contain spaces in it, must be unique)
locator	(string with no spaces, used to physically locate product, **not** unique)
quantity	(how many of this product in stock, greater than or equal to 0)
price	(in dollars and cents, greater than 0)

Note: Your program should be able to store up to 100 different products.

The program should offer the user a menu with the following options:

1. Add a new product to the inventory (prompt user for input values).
2. Remove a product from the inventory (by product name).
3. Adjust the quantity of a product (given the product name and change amount).
4. Display the information for a product (given the product name).
5. Display the inventory sorted by product name.
6. Quit

The program should perform the selected operation and then re-display the menu.

Do not change the menu numbers associated with the operations.

Note for options 1 and 2, you are not changing the quantity of a product. You are adding (or removing) the information about a product from the inventory (you are adding or removing an element from an array).

For the **Add** operation, a complete solution will ensure that the inventory is not full (100 products) before adding a new product, and it should perform input validation (see constraints above). If it fails to add a product, it will output a message indicating why.

For the **Remove** operation, the program should indicate whether the operation was successful or not (if the product was not found).

For **option 3, Adjust the quantity,** the program should ask for the desired change to the quantity (a positive value increases the quantity, a negative value decreases the quantity by that amount). If the resulting quantity would be less than 0, do not change the quantity and display an appropriate message.

For **option 4**, label the output values (i.e. Name:, Locator:, etc.) If the product is not found, display an appropriate message.

For **option 5**, display the information for each product on a separate line. The values should line up in columns (headers for the table are optional). If the inventory is empty, you may output an empty table (no need to display an error message).

NOTES:

- This is a long program! Start soon! Read all the instructions carefully!
- Do not use global variables (global constants are encouraged, especially for the maximum capacity of the inventory).
- Use an array of structures to store the inventory in the program. The structure definition should be global, but the array of structures should NOT be global.
- You MUST use a **partially filled array:** there should be no empty slots or gaps in the inventory. Keep all the products at the front of the array, in locations 0..count-1 where count is the number of products. If you have empty slots in your array, the provided search and sort code will NOT work!
- You MUST use **binary search** for all searches! No linear searches anywhere!
- The program must be modular, with significant work done by functions. Each function should perform a single, well-defined task (Hint: each menu choice). Also note that some arguments will need to be passed by reference!
- To input the product name (which may contain spaces) use this each time:

```
cin >> ws;       // skips whitespace (newline) after prev input
getline(cin, productName); // where productName is a string var
```

- OR you may require that the productName has no spaces and use cin >> productName; (for a small point deduction).
- You may use the following code from the book. See the Resources tool in TRACS:
  - Program 5-8: use this as a pattern for the menu portion of the program.
  - Program 8-2: this contains the binary search code.
  - Program 8-4: this contains the bubble sort code.

- I recommend implementing the features in this order (do one per day):
  - The menu (output a sentence for each menu option).
  - Add a product (without input validation first, then add it later).
  - Display the inventory (first unsorted, then (when that works) sorted).
  - Display the information for ONE product (by product name, requires search).
  - Adjust quantity (requires search).
  - Remove a product (requires search).

- I will put sample output on the class website in a separate file (output2.txt)

- Your program **must compile** and run, otherwise you will receive a score of 0.

- Your program must pass **Test Case 0** or you will receive a score of 30 or less with no credit for the other grading categories (correctness/constraints/style). The input values and expected output are in a file called **TC0.txt** on the class website. This test case Adds 3 products, with valid data and no spaces in the product name. It then selects option 5 to output the inventory in a table (it does **not** need to be sorted to pass TC0).

Logistics:

Name your file **assign2_xxxxx.cpp** where xxxxx is your TX State NetID (your txstate.edu email id). The file name should look something like this: assign2_js236.cpp

There are two steps to the turn-in process:

1. Submit an electronic copy using the Assignments tool on the TRACS website for this class.

2. Submit a printout of the source file at the beginning of class, the day the assignment is due. Please print your name on the front page, staple if there is more than one page.

See the assignment turn-in policy on the course website (cs.txstate.edu/~js236/cs2308) for more details.