Programming Assignment #5

Small Store Inventory Redux

CS 2308.001 and 003, Fall 2013
Instructor: Jill Seaman

Due:
section 001: in class **Monday, 11/4/2013** (upload electronic copy by 3:00pm)
section 003: in class **Tuesday, 11/5/2013** (upload electronic copy by 10:30am)

Write C++ **classes** that manage the inventory of a small store.

Product: For each product, you should store the following info:

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

Note: For a given product, the product name AND locator together must be unique. So you may have two entries for "iPhone 5c" if one has "box3" for the locator and the other has "shelf12".

ProductInventory:
When a product inventory object is created, it should dynamically allocate an array of Product, using a **constructor** parameter to determine the size. (You should also implement a **destructor**).

You should implement the following operations over the electronics store inventory:

addProduct: takes a product and adds it to the inventory. If the inventory is full, it should call the resize function first (see below). If a product with the same name and locator are already in the inventory, the add should fail and return false. If the quantity or price are invalid, it should fail and return false.

removeProduct: takes a product name and locator and removes any matching entry for that product from the inventory. Returns true if a product was removed.

showInventory: displays a listing of the store inventory to the screen, one product entry per line. Output locator, then quantity, then price, then product name.

sortInventory: reorders the products in the list, using the < (or >) operator over the products (see instructions below) (does not display them).

getTotalQuantity: returns the total number of units of all of the products in the inventory.

resize: internal function that doubles the size of the inventory array (recall duplicateArray). Allocates a new array, and copies all the existing products to it. Be sure to clean up.

Classes:

- Create classes for **Product** and **ProductInventory** with appropriate header files.
- Implement methods in the **ProductInventory** class to complete the operations described above.
  - You may add other private, helper, functions if you want.
- Implement functions in the **Product** class to:
  - set and get all instance variables (make instance variables private)
  - overload **==**, **<**,and **>** operators
    - for < and > use product name and when the product names are the same, use the locator.
    - for ==, two products are equal if they have the same product name and locator values.
- You should implement **two constructors** for the Product class: one that takes no arguments (quantity and price are 0, product name and locator are empty strings), and one that takes a value for each of the four member variables.

Input/Output:

The **main function** should be a driver program that tests the functionality of the Product and ProductInventory classes. See the website for a driver program that MUST work with your code (without changing the driver program). I recommend expanding the driver to do more complete testing of your code. Even if your program works correctly with the driver it may still have bugs not exposed by the driver.

Do not add extra I/O to the class functions. All the testing should happen in the driver.

NOTES:

- This program DOES need to be done in a Linux/Unix environment.

- Create and use a makefile to compile the executable program. There will be four goals in this makefile, because you will have three .cpp files. Use the following names for your files:

  Product.h
  Product.cpp
  ProductInventory.h
  ProductInventory.cpp
  ProductDriver.cpp

- Put a header comment at the top of each file.

- DO NOT change the names of the classes, functions or files.

- You do NOT need to use binary search for this assignment.

- You do NOT need to keep the array sorted for this assignment. If someone wants the inventory to be sorted, they will need to call sortInventory.

- Use == on Products whenever you need to search for a Product.

Logistics:

Since there are multiple files for this assignment, you need to combine them into one file before submitting them. **Do NOT submit your ProductDriver.cpp file.** You can use the zip utility from the Linux/Unix command line:

```
[...]$zip assign5_xxxxxx.zip ProductInventory.cpp
ProductInventory.h Product.cpp Product.h makefile
```

This combines the 5 files into one zip file, assign5_xxxxx.zip (where xxxxx is your NetID). Then you should submit only assign5_xxxxx.zip.

There are two steps to the turn-in process:

1. Submit an electronic copy using the Assignments tool in TRACS no later than one half hour before class the day the assignment is due (see top of page 1).

2. Submit a printout of the 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.

   If you are unable to turn a printout in during class, you have until 5pm on the day the assignment is due to turn it in to the computer science department office (Nueces 247). They will stamp it and put it in my mailbox. DO NOT slide it under my office door.