Programming Project 8

EE312 Fall 2015

Due November 9th, 2015 @ 11:59PM

FIVE POINTS

General: In this project you'll use your newly acquired C++ programming skills to create an improved version of the CRM project. For this project, we'll write two C++ classes, the Customer and the CustomerDB. We'll also use the String class that I wrote. We'll use essentially the same data structures and algorithms that we used before, but hopefully this time around you'll discover that (a) the program isn't nearly so tedious to create (no more destroyString problems!) and (b) the code you produce is a whole lot shorter and easier on the eyes. Good luck!

Your Mission: You'll need to modify two files for this project. You'll need to edit CustomerDB.cpp to complete all the member functions for that class, and you'll need to edit Project8.cpp to include your implementations of processInventory, processPurchase, and processSummarize. You'll note that both of these files already have some code provided for you. You are expected to incorporate this code in your solution. While you're not explicitly required to use the code, please bear in mind that part of your score for this project will be based on the style and implementation quality of your code (i.e., how easy it is to read and understand). It was my impression that the solution was a lot shorter, cleaner and simpler when I used those functions, you might want to do the same.

Most of the functions you need to write are fairly straightforward. For the CustomerDB you must write two "search" functions. The first of these functions CustomerDB::isMember searches through the current set of Customers and returns true if it finds a Customer with the matching name (and returns false otherwise) – pretty easy. The second function, CustomerDB::operator[](String) is another story.

You'll note that we've overloaded the [] operator for CustomerDB. We have a normal [] operator that has an int parameter. This operator simply returns the k^{th} Customer in the DB (returning that Customer by reference). We also specified a [] operator that has a String argument. This operator has several requirements that make it interesting and unique.

- If a Customer in the CustomerDB has a name that matches the argument to op[], then your function must return that Customer (returning by reference).
- If there is no Customer in the CustomerDB with that name, then your function must ADD A NEW Customer to the database and then return a reference to that newly added Customer.
- If you add a new Customer to the database, you must ensure that there is capacity for the customer in the array, and you must use amortized doubling to resize the array if there is insufficient capacity.

In other words, rather than providing an explicit “push_back” function, CustomerDB uses the [] operator to add new Customers, and the memory management strategy for CustomerDB uses a technique like amortized doubling. This is the same convention used in C++ for the standard Map data structure (and CustomerDB is actually, in fact, a primitive Map).

Once you’ve finished your CustomerDB implementation, write short, simple and clean implementations of processPurchase, processSummarize, processInventory and processSKU. Part of your grade will be determined by how well these functions utilize the CustomerDB. As a general hint, if you can make those functions shorter, you’re (probably) making them better. You are encouraged to use the same strategy you used in the first CRM project to handle custom SKUs. However, you can change the Customer.h file and the Customer struct in any way that you wish. You must not, however, change the CustomerDB.h file (or the String.h/String.cpp files either, for that matter).

Recall that the SKU command adds a new SKU type to the inventory system. We will test your program with at most two custom SKUs. Once a custom SKU is introduced into the system, the application must report all operations (inventory, summarize and purchase) correctly using that SKU. When a new custom SKU is first added to the application, there are zero of that item in inventory.

Testing: The same three test files from last time are provided. You might want to add some additional tests of your own. You are, of course, responsible for ensuring that your program is correct (i.e., not just able to execute the three test inputs and produce the correct output through some coincidence).