Programming Assignment #7

In this project you'll use your newly acquired C++ programming skills to create an improved version of the CRM project. This will involve writing two C++ classes: the `Customer` and the `CustomerDB`. We will also use a provided `UTString` class. The project will involve essentially the same data structures and algorithms from before, but hopefully this time around you'll discover that (a) the program isn't nearly as tedious and (b) the code you produce is a whole lot shorter and easier on the eyes.

Setup

Begin by downloading the starter files provided on Canvas and creating a new project on VS Code.

CRM

You will do the following for this project: edit `CustomerDB.cpp` to complete all member functions for the class, and edit `PA7.cpp` to implement `processInventory()`, `processPurchase()`, `processReturn()`, and `processSummarize()`. First, you should read the provided files. You will note that both files to be edited already have some code. While you're not explicitly required to use the provided code, your solution may be a good deal shorter, cleaner and simpler when using it.

CustomerDB.cpp

You will write the following functions in CustomerDB.cpp:

- `bool CustomerDB::isMember(UTString name)`: Search through the current set of Customers and returns true if it finds a `Customer` with the matching name, and returns false otherwise.
- `Customer& CustomerDB::operator[](String s)`: If a `Customer` in the `CustomerDB` has a name that matches `s`, then your function must return that `Customer` (returning the reference). If there is no `Customer` in the `CustomerDB` with that name, then your function must add the new `Customer` to the database and return a reference to the newly added `Customer`.

PA7.cpp

You will write the following functions in PA7.cpp:

- `processInventory()` – This function should read the item type and quantity from the input file and update the store's inventory of the indicated item type.
- `processPurchase()` – This function should read the customer's name, the item type and the quantity. The function should look up the customer in the customer database (creating a new customer record if this is a 1st-time customer) and increment the number of items purchased by this customer in their customer record. For example, if the customer record indicates that "Frank" had previously purchased 10 dice and the current command indicates that "Frank" is purchasing 20 dice, then the customer record should be set to indicate that 30 dice have been purchased by Frank. Note that each customer should have their own customer record (so that the innkeeper can keep track of who their best customers are and offer incentives like coupons and things).

- `processReturn()` - This command is used to record a return of items from a customer. Like before, each `<name>` will be a one-word string for the customer's name. The `<type>` is the same as the previous functions. Finally, the last part of the command is the number of items the customer is trying to return. After a successful return, the customer should have less `<type>`, and the database should have more `<type>`.
- `processSummarize()` – This command should print out a summary. The summary should first display the number of Books, Dice, Figures, Towers remaining in inventory at the time of the Summarize command. Next, the summary should display how many different customers have come to the store for purchases. Finally, the summary should report which customer purchased the most dice (and how many dice), who purchased the most books (and how many), who purchased the most figures (and how many), and who purchased the most towers (and how many towers). If a certain item has not been purchased by anybody, then the summary should indicate that. You are provided with three input files. At the end of each file (after the Quit command) is a transcript of what the output should be from the **Summary** command. Please format your output exactly as shown in the file.

Bounds and Requirements

When you add a new `Customer` to your `CustomerDB`, you must ensure that there is capacity for the customer in the array. To do so, use amortized doubling to resize the array if there is insufficient capacity (i.e., new array size = 2 * current array size).

For the functions in PA7.cpp, utilize the `CustomerDB` to the best of your ability. As a general hint, if you can make those functions shorter, you're (probably) making them better.

Common Questions

- No customers will attempt to buy negative numbers of anything
- **Do not add a `Customer` to the `CustomerDB` if they try to buy 0, also don't print anything.**
- Do not add customers to `CustomerDB` if they try to buy more than the available amount of inventory
- Only use `cout` to print. Use the `c_str()` function of the `UTString` class to get a string that `cout` can print properly (ex. `std::cout << foo.c_str()`)

Submission

Only edit and submit CustomerDB.cpp and PA7.cpp. Do not change the names of these files. After a short time, a few test case results should be visible to you on Gradescope. These are sample test cases, letting you know your Gradescope submission compiled and ran successfully on those tests. **Please note passing the sample cases DOES NOT MEAN you scored a 100 on the assignment.** Debug, debug, debug, and test, test, test!

Please do your own work on the project and do not share your code with others.

Good luck and have fun!