CPSC-131 Data Structures
Project 4: GroceryCheckout

This project is an extension of Project 1 that adds the use of the C++ Standard Library map class. It changes the GroceryInventory class to use a map instead of a vector, adds a Receipt class, and adds a ReceiptItem struct.

GroceryCheckout's input file records contain an item name, a quantity, a price, and a taxable indicator, the same as Project 1's file. Item names do not contain any spaces.

The GroceryCheckout class should have a map where the key is the item name and the data is the item's price and taxable indicator.

GroceryInventory has seven functions:

1. AddItem(), whose input is one item record. It checks for a duplicate before adding the item to the map. It returns a success/failure status; failure means the item was already present and couldn't be added.
2. CreateFromFile(), which reads an inventory file and calls AddItem to add each item to the map. *This is already completed*.
3. CreateReceipt(), which reads a file containing the item names of the customer's purchases and creates a receipt for the customer's transaction. It also reduces the inventory quantities of the items to reflect the purchases.
4. FindItem(), which finds the item with a given name and returns a pointer to the item's data. If the item wasn't found, it returns a null pointer.
5. RemoveItem(), which removes the item with a given name. It returns a success/failure status; failure means the item wasn't found.
6. SetTaxRate(), which does what its name implies. The value is a percentage, not a fraction.
7. Size(), which returns the number of items in the inventory.

In the past, projects that had to produce output files have failed because the actual and expected formats didn't match exactly, often merely differences in spacing or punctuation. This project's alternative is to have the actual results stored in Receipt's data members so that format isn't relevant. Those data members are:

- a vector of ReceiptItem structs (item name/price pairs)
- a subtotal
- a tax amount
- a total.

The Receipt data members will be compared with a set of expected results.

Source Code Files

You are given "skeleton" code files with declarations that may be incomplete and without any implementation. Implement the code and ensure that all the tests in `main.cpp` pass successfully.

- GroceryCheckout`.h`: This is to be completed with private member variables for class GroceryInventory.
- GroceryCheckout`.cpp`: This is to be completed with implementations of the GroceryCheckout functions.
- `Main.cpp`: The main function tests the output of your functions. It has several functions whose names begin with Test…; you can comment out the calls to tests of functions that you haven't implemented yet. This is already complete and should not change unless you wish to add helpful functions. When your project is graded, it will be tested using a different version of Main.cpp, not the one you were given.
- `README.md`: You must edit this file to include the name and CSUF email of each student in your group. Do this even if you are working by yourself. This information will be used so that we can enter your grades into Titanium.

Test Data Files

There are two data files that the GroceryCheckout project must read:

- Shipment.txt. This is the same file that Project 1 used to create the grocery inventory
- Checkout.txt. This file is used to create a receipt. It contains names of items purchased by a customer.

When your project is graded, it will be tested using different versions of these files, not the ones you were given.

Hints

You will find it helpful to implement GroceryItem's functions in small steps, building up from simple functions to more complex ones, testing each one as you go. You can edit your copy of main.cpp to enable only the tests you need to run. Don't wait until the very end to test your code.

Here's a suggested order:

1. AddItem and Size
2. FindItem
3. RemoveItem
4. SetTaxRate
5. CreateReceipt

Important Guidelines

Here are some guidelines for submitting projects. Failing to follow these guidelines will affect project grades.

Repository Name

There is a standard name format that identifies the project and the section whose instructor should grade it. All names begin with "projectN" *that is already entered in github*. This should be followed by the class section number in the form "-0n". The team name should follow the section number, separated by a dash (-). The team name can be whatever you want. Here's an example of a name:

04-someteamname

This will create the repository URL:
https://github.com/CSUF-CPSC-131-Fall2018/project4-04-someteamname

Team Members

Put the names of the team members (or the single person doing the project) into the README.md file so they will get the grade for the project. When this file doesn't contain student names, there's no way to know who did the work.

Ada Lovelace died in 1852 and Charles Babbage died in 1871, so they can't possibly have contributed to any project.

Two-person Teams

Make only one submittal for a project done by two people, even if they are in different sections. The instructor whose student is listed first will grade the project and post the grades for both members. If you have created another repository, make sure to remove your name and email address from the README file before the deadline.

Obtaining and Submitting Code

Click the assignment link to fork your own copy of the skeleton code to your PC. One student from a group clicks on the link below and forms a new team. This student then invites his/her project partner as an "Outside collaborator" in the settings menu.

https://classroom.github.com/g/Q9AE3yUX

Development environment

The test platform is Linux with the g++ -std=c++14 compiler. For this reason, the recommended development platform is Linux.

Linux environment

To attempt to compile the test program, use the following command:

```
$ clang++ -g -std=c++14 *.cpp -o test
```

To attempt to run the compiled test program, use the following command:

```
$ ./test
```

Grading rubric

Your grade will be comprised of two parts, *Form* and *Function*.
Function refers to whether your code works properly as tested by the main function (80%).
Form refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is November 11th at 11:55pm.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline.

Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.