Project 1: Grocery Store Inventory

CPSC 131 Fall 2018

Introduction

Assume you're writing software for a grocery store to maintain its inventory. You are provided a listing of items from the latest shipment to the store with each items name, quantity, price, and whether it is taxable or not. The store wants you to build a program that is able to store each grocery item in an internal list and ultimately calculate the potential revenue, tax revenue, and total revenue for the entire shipment.

Objective

You are given a partial implementation of two classes. `GroceryItem` is a class that holds the information for each grocery item. `GroceryInventory` is a class that holds an internal listing of `GroceryItems` as well as the tax rate for the store. Your inventory could be 100 items it could be 9000 items. You won't know until your program gets the shipment at runtime. For this you should use the vector class from the C++ standard library.

Complete the implementation of these classes, adding public/private member variables and functions as needed. You should choose an appropriate data structure to maintain this inventory with an unknown size known only at runtime. Your code is tested in the provided `main.cpp.`

Initially the given code will not compile. As you complete the code, the tests should start to pass in `main.cpp.`

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.

- `GroceryItem.h`: This is to be completed
- `GroceryInventory.h`: This is to be completed
- `main.cpp`: The main function tests the output of your functions. This is already complete and should not change. You may wish to add additional tests. During grading your main.cpp file will be replaced with the one you were provided with.

- `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.

Hints

Start implementing the `GroceryItem` class, then the `GroceryInventory` class. As you write each class, test your code. main.cpp provides many of these tests already. Do not wait till the very end to test your code.
Names of items do not contain any spaces.

Obtaining and submitting code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions. This requires you to have an account on github.com. If you are new to GitHub, do the following to get started:

1. Create an account at github.com. You may want to use this account to show a portfolio of your work to prospective employers in the future, so choose something professional.
2. Read Understanding the GitHub Flow and Hello World at GitHub Guides.
3. Read the instructions below for instructions on how to test.

Once you understand the basic operation of git, 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.

Do not fork your repository to your personal github account (instructors have admin access to private repositories under https://github.com/CSUF-CPSC-131-Fall2018/). Your code should have a URL like https://github.com/CSUF-CPSC-131-Fall2018/project1-brians, NOT https://github.com/brian/project1-brians.

https://classroom.github.com/a/Ci2GcHVy

Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission. Don’t wait until the deadline to learn how to push code!

Testing

You can test how well your code is working by running the three test programs. Of course it is possible to compile and run each of these manually by hand, but this can become tedious. For that reason we have provided you with a testing script that automates the process, called critic.

critic is a Python 3 program that will work in the endorsed GNU/Linux environment. You can execute the ./critic command within your repository directory. With no arguments it prints out usage information:

```
$ ./critic
Usage:

     critic <COMMAND>

Commands:

     build      compile and link
     help       print this usage information
     team       print team members
     test       compile, then run unit tests
```

To attempt to compile all three test programs, use the ./critic build command:

```
$ ./critic build
```

To attempt to run all three test programs, use the ./critic test command:

```
$ ./critic test
```

We will use this process to test the functionality of your code while grading. We recommend that you try it out yourself ahead of time to confirm that your code works properly, or to get a feeling for how far from complete your work is.

Automated testing

We are also piloting the use of a continuous integration web service that automatically tests your code and gives real-time feedback. This service is provided courtesy of Travis CI.

The Travis service will wait for you to push code to GitHub. After you do, it will try to compile and build your code. It will then send you an email describing the outcome. The email also has a link to a dashboard web page you can view to see a detailed log of what worked, or didn’t. This way you can get immediate objective feedback about how well your code is working. We recommend pushing your code to GitHub every time you reach a milestone and would like feedback on your progress.

You can check the results of automatic testing on Travis at https://travis-ci.com/ (same login as your github account).

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 September 23rd at 11:50pm.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

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.