Homework 2 — Testing the Store

Due Wednesday, Friday 20th at 9pm

(50 points)

Objectives:

- Use continuous integration with a project that uses unit testing
- Understand and diagram how an implementation of a program that uses a separate UI class works
- Design and implement unit tests for all parts of an application

Credit:

- `hw2_diagram.pdf`
- a link to your private repository (github or bitbucket)
- `hw2.zip`, the downloaded zip file of your master branch of your repository when you are finished
- Your source code must include newly written comments for the methods (we recommend doing this in conjunction with Part 2)
- Your repo must include your newly written unit tests (Part 3)
- Your repo must include a Makefile updated to build your testing target. It must compile via Travis' build system.
- Your repo must include at least 2 new store files for testing (`test1.txt, test2.txt`)

Instructions:

We have provided most of a program that lets a user electronically shop at a store. The program that you are given is not be complete (there will be no money or goods actually being exchanged, the software only allows one person to shop at the store at any one time), but it does let one person visit the store and make purchases.

You **may** work with a partner for this assignment. If you work with a partner, you should indicate this in your README.md file. In addition, you should list which part(s) each person in your partnership worked on in your README.md.

Part 1: Version Control and Continuous Integration (10 points) - DO THIS NOW

Before beginning on the rest of the project, create a private repository on github or bitbucket and add your partner (if you have one) and Felix Muzny as collaborators in your repo.

Usernames: muzny (github), felixmuzny (bitbucket)

Create a README.md file that has both partner's names and add the provided source code to your repository.

Connect your repository to Travis CI and add a build status icon to your README (see the course resources document for instructions on how to do this).

We will be looking for you to have made at least 8 different commits to your repository and have pushed to remote at least 4 times, triggering at least 4 different builds on Travis. when we are grading this homework. If you are working with a partner, each partner needs to have made at least 4 different commits in the repository and have pushed to remote at least 2 times, triggering a total of at least 4 different builds on Travis. We will leave decisions about what branch to develop on up to you.

The Actual Program

Program Flow:

1) When the program starts, the user implicitly enters the given store.
2) The user then has the choice to do any of the following things:
    a) Display inventory
    b) Add item to cart
    c) Remove item from cart
    d) Checkout
    e) Leave
3) The user can do any number of this task **in any order** until they leave the store, at which point the program exits.

Loading and Saving Data:

Your program should be launched by running the command:
`./shop <STORE_FILE.txt>`
For example: `./shop store.txt`

Each time the user checks out, you should overwrite the previous copy of the given `<STORE_FILE.txt>` with an updated version to account for the changed quantities.

The data should be in the following format:

```
Store name

Inventory:
Kind,price,quantity
Kind,price,quantity
Kind,price,quantity
Kind,price,quantity
```

Name	Type	Purpose
Item	class	Represents each item in the store. There should be no way to put an `Item` in an illegal state.
Shopping Cart	class (provided)	Represents the user's shopping cart. There should be no way to put a `ShopingCart` in an illegal state.
Store	class (provided)	Represents the store. Your `Store` will have a name and inventory. Your `Store` will keep track of how many of each item it has left and let the user place items in their `ShoppingCart`, checkout, and, eventually leave.

		There should be no way to put a `Store` in an illegal state. (The user should not be allowed to steal from the `Store`, for example.)
TextUI	class (provided)	You will notice that the TextUI object calls a number of `Store` object methods in the `RouteChoice` method.

Part 2: Diagram the Application and write a main.cpp (15 points)

1) Your first task is to draw a diagram of all the classes involved with this software and how they communicate with each other. Make sure that your diagram indicates which methods are used by which objects and indicate how they affect any other objects. You don't have to follow a formal diagramming schema to create this picture.

Your diagram must include:

- Every class and every method it contains
- For every method, which other methods they call

Your diagram may be handwritten, but must be legible. Poorly lit photos will not be accepted.

2) Take a look at the `Clone()` method in `Item`. Why does this method exist? Write your answer on the diagram that you created for the first question of this part.

3) Once you have completed the first two steps, write a `main.cpp` file. As a reference, our `main.cpp`, including comments and empty lines is 33 lines long. You will need to instantiate a `Store` and a `TextUI` in your main function.

Part 3: Testing (25 points)

You must write unit tests using the Catch framework (https://github.com/catchorg/Catch2) for each one of the methods and constructors, for all classes except the TextUI. For methods that write to output files, test the content that they would write. For methods that read from input files, create a testing input file or pass in a compatible stream.

You should both write tests that check the basic functionality of your methods, and also tests that check more comprehensive run throughs of your program.

You should use the TEST_CASE and SECTION macros to take advantage of Catch's ability to minimize code copy+pasting.

Each TEST_CASE should only test 1 kind of thing (e.g. adding an item to the cart).

Each SECTION should only test 1 functionality (e.g. adding an item to the cart when it is empty). It's okay if you have to do other things to get your program to the state you need to test, but those other things should also be tested elsewhere.

You should use tags (https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md#test-cases-and-sections) to mark which tests go with which object, as well as tests that correspond to complete run-throughs of your program.

If you write a test that puts your Store/Cart/Item/etc into an illegal state: edit the source code so that this is no longer possible. This may involve parameter checking, throwing errors, or another strategy. **Comment these changes.**

Example Outputs
The example outputs provided on the class website show the basic functionality of the program. Notice what happens to the inventory when the user leaves the program and runs it again.

Evaluation of your tests
We will run your tests against a slightly buggy version of the same software to test how full your testing coverage is. (AKA: **Do not change any method names**, etc, as that will break our grading scripts and you do not want to do that).