Discovering Requirements

- Implement a simple grocery list program
  - Needs following attributes
    - Item name
    - unit of measure
    - quantity
    - price per unit
  - Needs following actions
    - Add item
    - delete item
    - print out list
      - Show all attributes, plus subtotal, plus total
    - save list to file
    - read list from file

Design

I will make two classes:
- List in List.cpp and defined in List.h
- Item in Item.cpp and defined in Item.h

An item will contain attributes for name, unit, quantity, price per unit, and its subtotal. Its constructor will take everything but the subtotal, and we will create the subtotal while constructing the object. The Item will be immutable after creation and will only have accessor functions.

A list will have a private vector of Items. The list will be created at program start, and overwritten if a file is loaded. There will be two functions that facilitate adding and deleting items (add item by passing a reference to it (it will be copied into the vector), delete item by giving the index of the item to delete). There will be getter functions for the count of items in the list, accessing individual items, and getting the total cost. Furthermore, there will be a print function that uses the above functions to print out the list to the screen.

The list object will also have two methods to read the list from a file and write the list to a file. These will take references to their respective file streams. When creating a file, I will make a two line header so that I can identify files that were created by the program (and not read in garbage data). The file itself will just be each attribute of an item, followed by a newline. Because blank lines will not be valid in my program, the program simply needs to loop through the file and can expect that the data that it needs will be there. If not, though, I will simply fail out of the reading process and clear the list in memory. Reading a list will clear any current items in the memory held list. I didn’t see any requirements that would discourage doing this.

The main loop will just be a switch statement that uses the correct functions in order to present a menu to the user.

Testing

I tested adding various objects, and everything seemed to work correctly. One concern was how the program should handle spaces in names or units. I simply used getline() for all of my inputs and that took care of that issue.

Another issue is handling negative quantities and prices. I chose not to allow that, and re-prompt the user for input.

If a user chooses to delete an item that doesn't exist (index out of range) I tell them so and fall back to the main menu.

Items being added to the list have proper prices, as verified by hand.

Save or Load failures are caught by the program and reported to the user.

Reflection

Once one figured out how they were going to design their objects, this project was pretty easy. As always, I think that most of the difficulty was located in finding all of the edge cases and deciding how to deal with them. Fortunately, there were very few things during testing that I had to go back and account for, probably due to design decisions that I made up front.

I'm getting quite tired of having to write menu systems. It seems that half of what I am doing on these projects is parsing user input and making sure that it is valid. I think I am going to start creating a library of my most common functions for validating user input, that way I don't have to keep writing boilerplate code.

This project was good for re-enforcing how objects work and how to properly encapsulate data. I'm sure the lessons learned here will be useful as we move forward and get into more complex object-oriented concepts such as operator overloading and inheritance.