Assignment 2: List and Item objects

Design

My initial approach to creating the Shopping list application would involve separating responsibilities of the two objects. My initial plan is to have the List class only be responsible for adding, removing, and displaying the items that are on the list. The item class will be responsible for holding data and checking if it is the same as another item that might exist by overloading the equality comparison operator.

Each data member of the Item class should have a getter function and setter function so that you can create, update or display any part of the item. The only other part that an Item function would need is a function for calculating the extended price, since this can be done by simply multiplying data members against one another and returning it.

Finally, the main program will do the majority of the control flow and displaying prompts to the user. This will allow the Item and List to be separate from the data input until they are invoked to take inputs from the user.

Test Plan

Options:

1. Add Item
2. Remove Item
3. Display List
4. Quit Program

Test Inputs	Function Tested	Expected Results
1; Item = {name:"Beer", unitType:"Can", quantity: 6, price: 2.99 }	addItem()	Item added to list with correct calculations
1; Item = {name:"Beer", unitType:"Can", quantity: 6, price: 2.99 }, Item = {name:"Beer", unitType:"Can", quantity: 6, price: 2.99 }	addItem()	Second attempt would trigger check and allow user to attempt to update quantity of that item in their list
1; (see above item) && 3	addItem() && printItems()	The newly added item would be displayed on the list with correct totals
1; (see above, with variations on name) multiple times && 3	addItem() && printItems()	Correct totals and list length for multiple items
1; (see above) && 2; Selecting item number corresponding to those on the list	addItem() && removeItem()	The selected item would be removed from the list
1; (see above) then 2 then 3; Selecting item number corresponding to those on the list	addItem() && removeItem() && printItems()	The selected item would be removed from the list and printItems() would show a new correct list
4	N/A	The program would exit with a correct farewell

I had tested this for various inputs and had to tweak along the way when I realized that several parts were not correctly displaying. During testing of adding items, I realized that I was adding the new item into the vector inside of the loop, which caused it to continue prompting me on updating the quantity, so I moved it outside of the loop since we would exit it anyways if we didn't have a match. Ultimately, most things worked as intended and was tested in more of a test driven way since pieces were implemented and tested along the way with a final testing phase the passed all stages.

Reflection

In the process of implementing my functions, I had realized the care that should be taken when trying to invoke the “new” operator along with a constructor. Also that I don’t necessarily need a name for each new object I create since they are all being stored inside a vector and could be referenced from with that vector anyways. This had set me back for quite a while until I realized this, after which most everything else was fairly straightforward.

The overloading of the equal comparison operator was interesting and I ultimately went with a simple comparison instead of a deep comparison of the Item objects, just to prevent any confusion or complication. The only compare on the name string, but I could have also made sure the unit type and price matched so that someone could have the same named items of different units with different prices (like bulk items). I ended up just going with simplicity since I didn’t want to introduce any other complications like I had run into with my trouble creating the new Item objects dynamically.

My main approach throughout development was to piecemeal each part of the program and test it right after implementation to ensure it worked before moving to the next piece. The only main bulk I implemented was the getter and setter function since these were fairly simple. I also (as a design choice) decided to have the addItem() function be responsible for checking for a duplicate prior to adding it, since it made sense for the add function to be the gatekeeper.

Finally, I opted to have a slightly more complex Makefile since it really sped up my testing process by not having to recompile all of my files and only the ones that I had updated since the last compile. It was a significant benefit and I look forward to enhancing my future Makefiles since it was a few second through each test phase that I saved (which adds up fast when you are constantly tweaking things in the program).