Alex Hoffer
10/18/2015

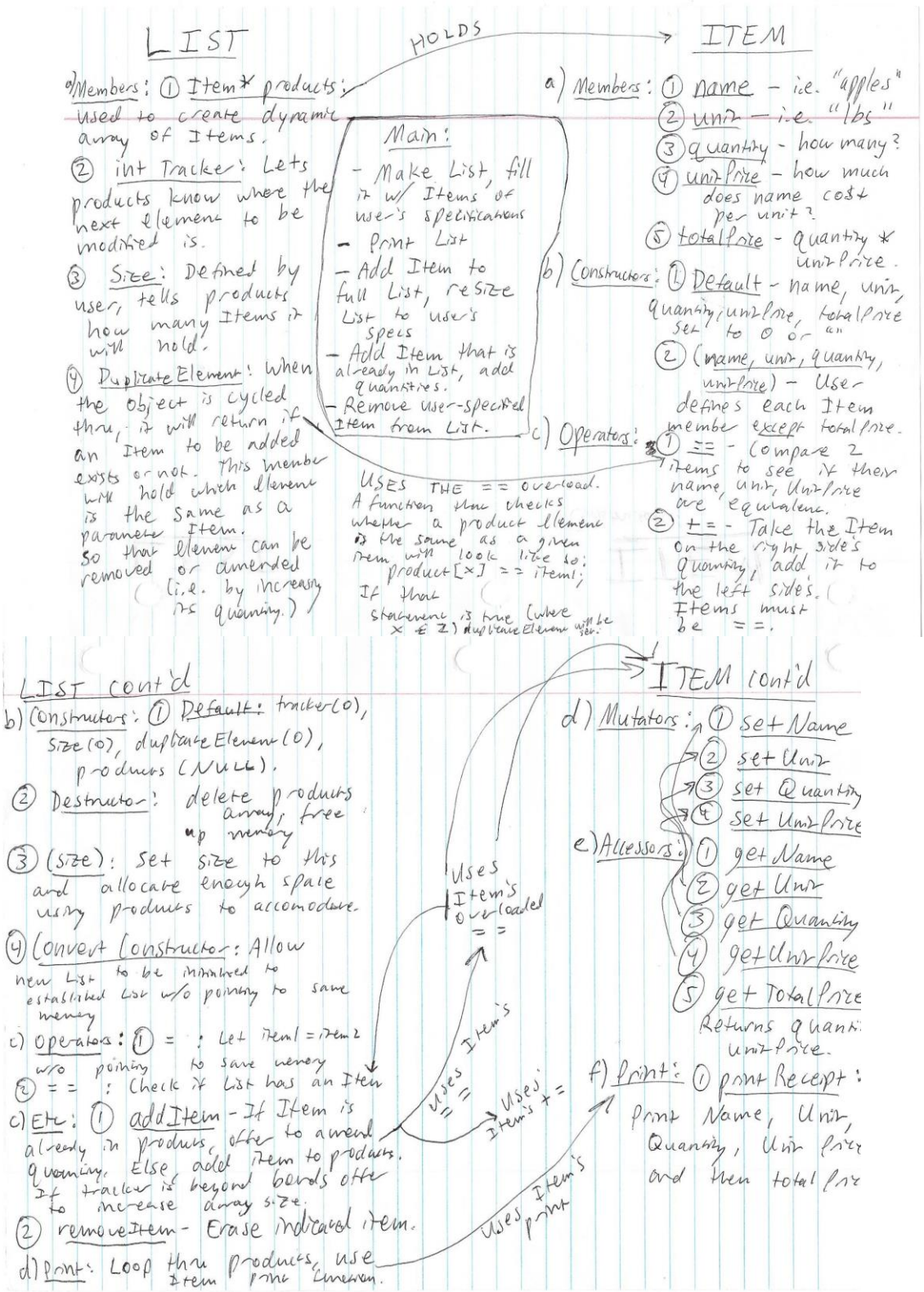
Reflections, Design, and Test Plan

1. Reflections: I am greatly satisfied with the end product's functionality. There are a few examples of places where I made stylistic decisions that I think make this little program perform at a high level. For example, in the case of removing an item from the array, I wrote three separate conditions which encompass each possibility of where the removed element is located and how it should be handled. I ensured that if an element in the middle of an array of Items is erased, each element after it is pushed back so that there isn't some random empty element and so that another Item can be added without any problems. Another decision I made was with regards to resizing the array, which I allowed if an Item was being added to a full array. I made it so the user could resize the array to any value beyond the current size (I don't think it's necessary to reduce the size of the array if an Item is to be added. However, I could've added exceptions to make it so if the user DID put in a smaller number it would've printed an error. I just don't think this is necessary, given the scope of the assignment). I could've just made it so I extend the array by one, but I thought it would be nice to give the user the option to increase the size by 1, 2, or whatever. I like that my program is adaptable, I added a lot of sensible if-else statements so that there are a number of different outcomes given the user's input.

What I kind of don't like (but view as a necessity) about my program is some of the printed out statements that feel kind of rigid. For example, I use a function called getUserItem() that asks the user to put in name, etc. of an Item and then it returns that item. I use it in a number of different contexts. A context that it feels rigid in, though, is when the user adds an Item that already is in the List and they take the opportunity to add the quantities together. I ask the user using that function to VERBATIM (another thing I don't like: I wish the user didn't have to type

the EXACT same thing in to find the right object) type in their Item's information, but it makes it seem like they can't select a different quantity then already established, which isn't true: whichever quantity they add will be added to the pre-existing Item. Finally, another stylistic decision I am mostly ambivalent about is the idea of "What makes two Items equal?" and the contexts in which the answer to this question changes. If I am searching through the array, the Item to be removed should be completely identical. However, if I am intending merely to add quantities, it doesn't even matter what the quantity of the Item I'm looking for is when I'm searching for it. I wish I had a bit more guidance on the assignment about what it means for two Items to be equivalent in these contexts.

2. Design:



3. Test Plan:

Value	Expected	Actual	Comments
Add item to list	When printed, item should appear	All member variables of item appear in proper format.	
Remove item from list	Print list after item is added. Remove item. Then print list again. Item should be gone.	Upon printing, the item is gone and if the removed element isn't at the end, each element after it is moved back.	Ensures that removing an element doesn't cause issues within the array.
Allocate a list of size 5. Attempt to put 6 items in list.	Offer to resize.	Offers to resize array to whatever size user wants. Adds the item to next free element.	Tests proper dynamic allocation.
Add item to list of size 5. Remove one item at element 4. Attempt to put an item at element 4.	Print list: item should be located at element 4.	Prints entire list and the proper item exists at the final location. Tells me that there is no more space in the array.	Tests proper removal.
List of size 5: remove item at element n, where n < 4.	Print list: the item should be located at the final element not marked by 0. Each element should be shifted backwards 1, making the last free element available.	Elements have been shifted back by one. Element 4 is free for use.	Tests the shifting of elements backwards after deletion.

Est. list of size 5. Initialize list to this list. Change new list's first value. Print both lists.	The two lists should have different first values but identical other values.	Same as expected.	Tests List's overloaded = operator.
Est. two lists of size 5. Set one list (a) to the other (b). Change a's first value. Print both lists.	Two lists should have different first values but identical other values.	Same as expected.	Tests List's copy constructor.
Add an item to a list that already has an item of that name, unit, and unit price.	Ask the user if he would like to add that quantity to the previous quantity.	Asks the user same as expected. Adds the quantity if yes and prints out the quantity change.	Tests the add quantity functionality.
Est. list of size 5, print out total price.	The total price should correspond to the summation of each item's unit price * quantity.	Same as expected.	Tests getTotalPrice(), function in Item.
Est. two Items of the same name, unit, and unit price. Compare them using the overloaded == operator.	== should return TRUE, corresponding to yes, they are identical.	The overloaded operator determines the two Items are the same based on each criteria but not quantity (as specified).	Tests Item's == operator.
Attempt to remove an item in a list that	Throw an error.	Prints out an error.	Tests If-Else comprehensiveness.

doesn't contain any items.			