Sean Mulholland ID No: 932652783 mulholls@onid.oregonstate.edu

Assignment 2 Design Document-

My basic plan is fairly straight forward. I imagine 2 classes are necessary for my program, one to represent Items that the user picks, and a List to store the items in. The items will be fairly basic in that they will be composed of 2 private strings, 2 private doubles, and get and set functions for both. I also will be adding an extra get function to get the subtotal of an item object. A diagram for my Item class is shown in Figure 1. For my List class, it will also be fairly basic. It will contain a single private vector of Items, a single private string of the list name, and functions to add an item, get the vector, get an item address in the vector, set and get the list name, and get the list size. A diagram is shown in Figure 2.

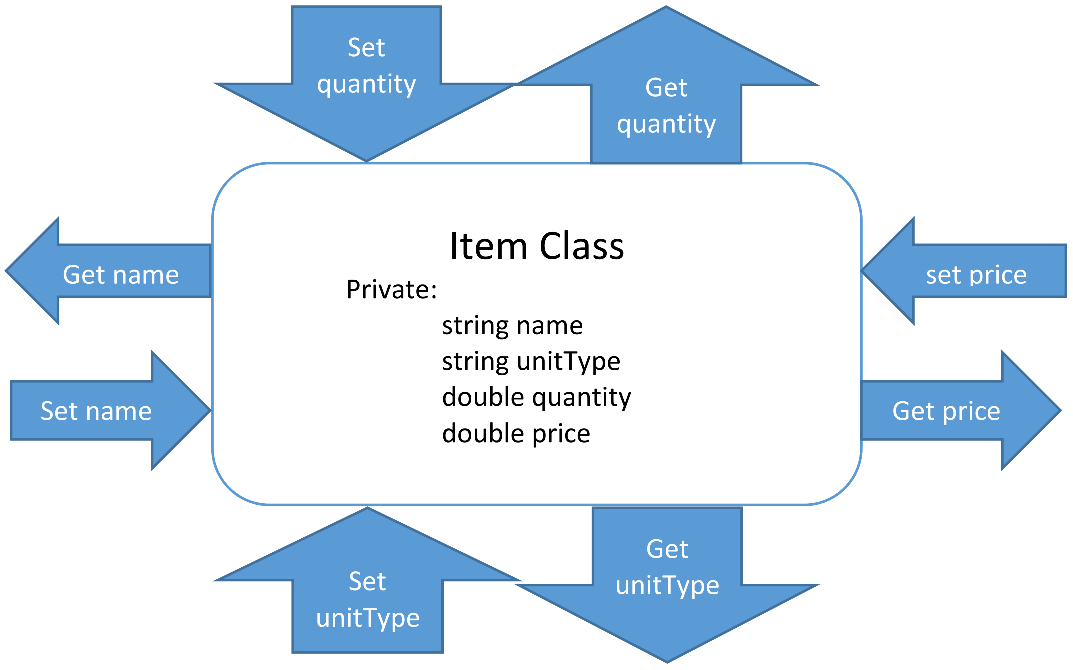

Figure 1. Item Class Layout

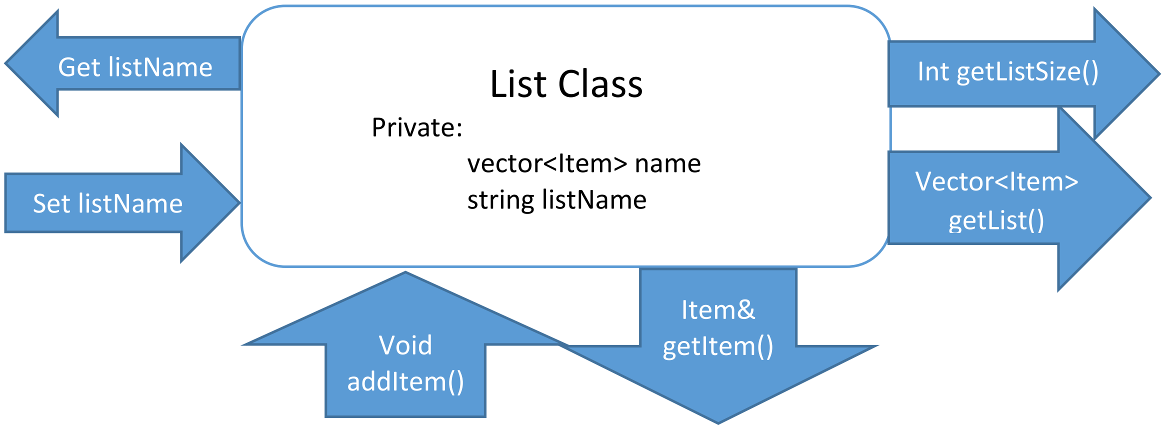

Figure 2. List Class Layout

The rest of my program should also follow a fairly basic information flow. I plan on doing several looped menus, and from each you can select the option and then return to the main menu again. Figure 3 gives a basic idea of the layout I'm trying to achieve. It's not the best diagram, but there will be room to use functions in multiple places if implement it properly.

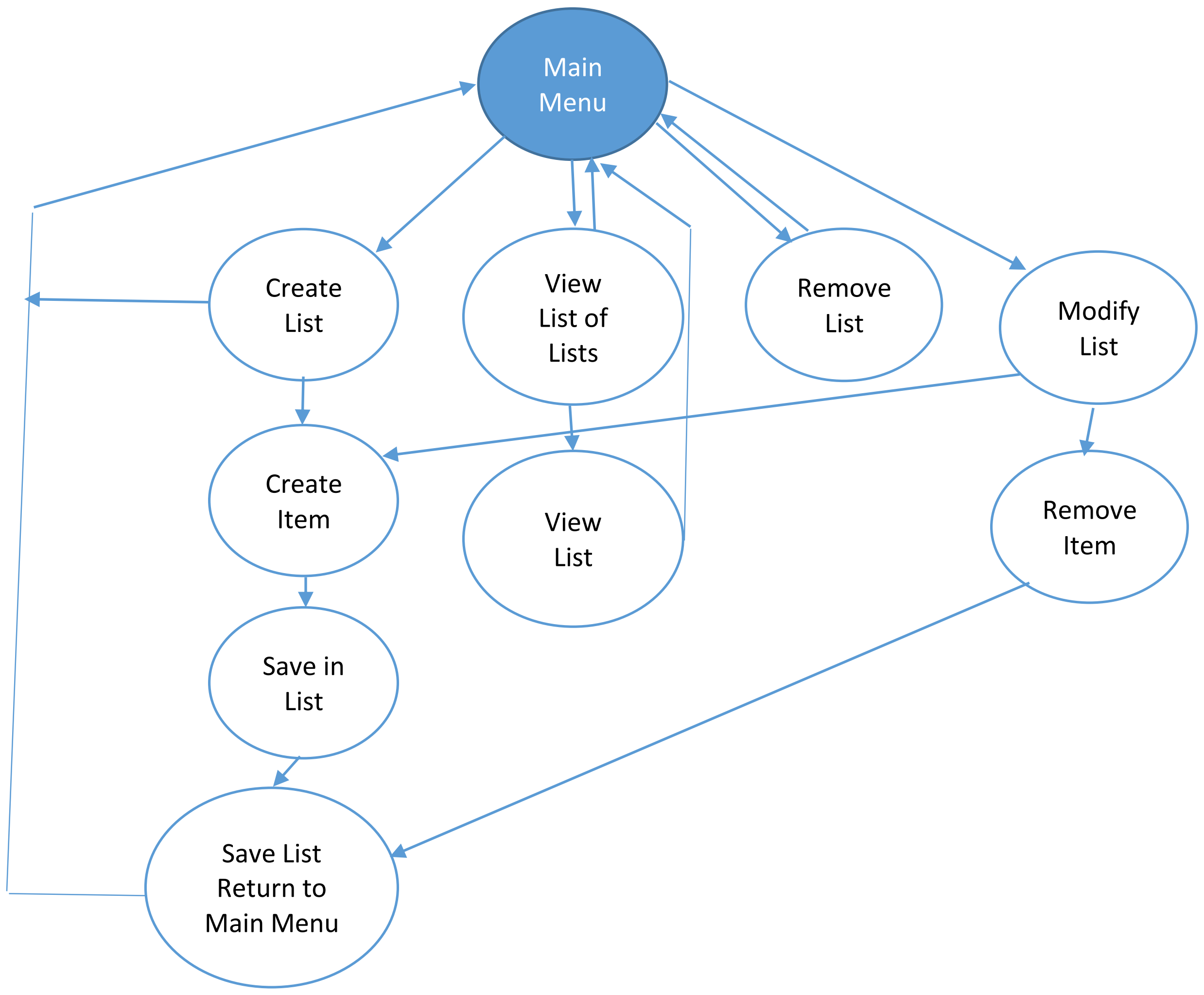

Figure 3. Assignment 2 Function Layout

Test Plan:

I plan on having a good bit of user input, and some information being input from files. My test involves ensuring that I validate input at each of my menus by trying a variety of inputs at each menu stage. I also plan on creating several types of items with a variety of entries in each input. I am working a lot with strings, so by using the '=' operator and by filtering inputs to a common easily comparable form I should be able to process most user input. For the item creation, I plan on validating that the quantity and price only accept integers/doubles and store the information appropriately as a double. I need to ensure my list inputs work so I will try altering the ***.txt files and seeing what happens. I plan on using a delimiting character in my lists (‘|’) to separate information, so I believe that as long as I don’t go too crazy the lists should withstand SOME tampering. I’m not sure how to parse an arbitrarily modified file, but I will do my best to allow large inputs (within reason) from the user.

I will also be testing blank files, and inputs with no entry. Strings seem to handle the burden of blankness fairly well, so I believe that even though it may accept something funky from the user… The program won’t break, and they should be able to modify their list accordingly after the fact.

Reflection:

My implementation went a little bit more messy than planned. The classes I created worked great and covered the important information for the user exceptionally well. Where I had issues was creating a sensible Assignment2.cpp without too many repetitive functions and that did what they were supposed to do. I still think with a couple more revisions the code could be trimmed down further, but it is fully functional as it stands right now. The process I ended up using was by finding repetitive bits of code and seeing how I could generalize it. For instance, my create list function was standalone and all it’s sub-functions were standalone. After I started implementing my modify list function, it became clear that I could consolidate code into a single modify list function. This happened in several other places, but the revision process was the same.

The inputs are all validated, so the user will either be able to make a proper selection or be told their input is invalid, or be reprompted. The file access points all function well, as well as I wrote a function to validate the filename prior to use. It gets a user input, and checks to see if a file opens with that name, and if not it reprompts the user.

It took me a few iterations to get the UI properly lined up with all the appropriate information but I think it is readable and updates fairly well. It removes most clutter from the screen so the user isn’t overwhelmed. It took a lot of fine tuning and retesting of the setw() values, but once everything lined up it looks pretty nice.

My favorite part of my final implementation is that the program can update save lists. If I had more time I would rearrange some functions so the user could modify a list more than once, but I didn’t have time to add that functionality. As it stands, they can open a list which is turned into a list object populated with the appropriate items, and from there they can add an Item or remove an item from the saved list. For creating a brand new list, the user loops through the “modifyList” function I wrote so they continually get the choices. My original idea is that a saved list worth keeping wouldn’t need much modification (like weekly groceries that you always purchase). But, hindsight for the user it’s just easier to be able to modify repeatedly if you need to.