**Requirements**

The goal of the assignment is the creation of a grocery shopping list which allows the user to add and remove items from the list and which also prints to console the extended price of the items on the list along with the total price of all items on the list.

Each item on the list is to be identified by:

* Name
* Unit type, i.e. whether item is a box, can, weight
* Quantity
* Unit price

For the display to the console option, the items added to the list will display all the item identifiers, as well as the extended price, i.e. unit price \* quantity and also the total price of all the items on the list.

**Design decisions**

These are the final design decisions, updated after the results of the test plan showed incorrect output or non­optimal results which are elaborated in a separate section.

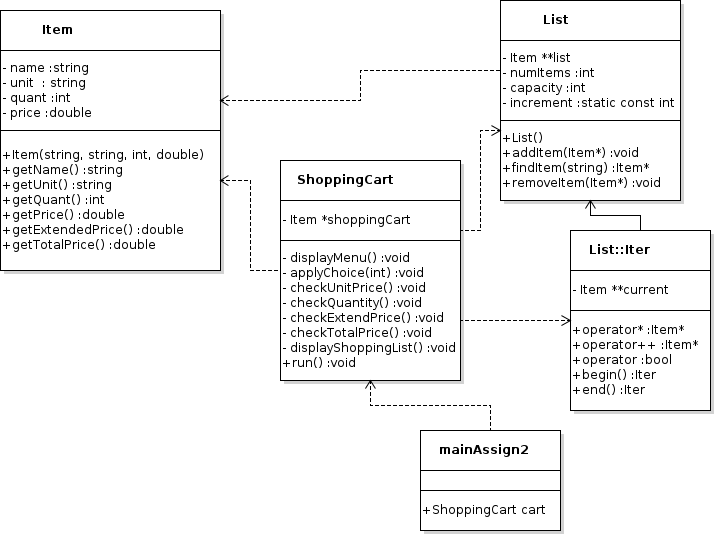
**Design overview**

To meet the program requirements, I decided to create separate classes for Item and List and designed them such that they are encapsulated from each other, while another class, the class ShoppingCart, was used for interactions between Item and List, interact with the user and to drive the program. mainAssign2 was used to print to console the welcome message, instantiate the ShoppingCart object and run it. At no point, did the ShoppingCart class lose control to the mainAssign2 function. The **program was mainly driven by menu selection from the user**.

Input validation of user input is done via functions in the “utils” namespace that is declared in the header file, “utils.hpp”. I created this utilities function, as generic functions that could be used for many programs. Since these “utils” functions are generic and not intrinsic to this program, I have not shown the utils function in my model design.

**utils Function:** was used to create generic functions that could be used elsewhere. int readInteger (string prompt, int min, int max) → allowed me make it a very generic function which can be used to get input from user and validate it based on criteria that can be passed in. **CHANGES BECAUSE OF TEST PLAN RESULTS: because of bad output on the unit price, I had to write an additional function for readDouble,** which validated the input for a double type variable for unit Price.

**Model Design**

****

**The functionality of each class is as follows:**

**class Item:** represents each item in the list and gives the item name, units of item, unit price & quantity to buy. I also used inheritance to **create overload the ==operator function** in this class. **The overloaded== operator was used for the getName() function in the Item class.** As was indicated in the assignment requirements documentation, I assumed that the user would input the information correctly and ONLY compared the items by the item names.

**class List:** was used to store Item objects in the list object. I used dynamic allocation of array, since at the time of compilation, there was no knowledge of number of items that a user would input. Associated with this class, is the class Iter, which was defined to iterate through the array, when additional items are added and also to move items up the array, when any item is removed, so that there are no empty indices within the array. While defined within the List class, the functionality of the member functions of class Iter, were used in the ShoppingCart class when array items were added and removed from the list.

**class ShoppingCart**: So that I could encapsulate the Item and List class, I created this class, which was used to drive the program, with the following functions:

* displayMenu() function - wherein I got the user selection for either adding items, removing them, displaying the cart or exiting the program
* applyChoice() function – in which I used the switch operation to cycle through the different choices that the user could make
* displayCart() function to print the contents of the cart, namely, name of item, units to buy, unit price per item, extended price and total price of all contents of the cart.

**MainAssign2:** creates the ShoppingCart object and calls the run function. The ShoppingCart retains control and all actions are via the menu functions, which only give the option to continue with the options in the displayMenu() function.

**Test Plan**

Compared to the previous assignment, there were fewer changes to be made to the

original design, mainly because I spent a lot more time on the design part of the assignment, which paid off in less time needed for debugging. My original plan included, creation of the Iter class, which allowed me to iterate through the array, to make the changes necessary, when the user removed items from the cart.



**Changes made to program based on problems highlighted by test plan**

* Formatting of Shopping Cart display, showing the items in the cart along with the extended price and total price
* Exiting the program caused a segmentation core dump, which turned out to be a difficult task for me, since I could not find what was causing the problem. Turned out, it was a misplaced [ ] in the Destructor of the List class, where I had used dynamic allocation of array to get user input. By tracing through the debugger, I spent several hours, tracking each addition and removal of items from the Shopping Cart, to find the error.
* For the removeItem() functionality, I realized that I could optimize it by creating another function, called findItem() function in the ShoppingCart class, which would search through the list, and for this, I used the Iter class I had written. Additionally, I was able to use the overloaded== operator to find matches.
* The Iter class came in very handy for several cases:
  + to iterate through the array for the findItem()
  + to iterate through the array to calculate the sum of all extended price of each of the items in the list of items
* Quantity to buy was changed from int to double, after realizing that a user could buy ½ a pound of something and the output need not be truncated.