**Design of classes and how they are used**

List class

Private: (data members)

Item* itemList

- an array of pointers to Item objects

Int listNumber

- an integer that keeps track of the number of Item objects in the Itemlist array of pointers

Public: (member functions)

List() (constructor)

- creates an array of 4 pointers to Item objects (the starting Item objects are set to be 'empty')

Void addItem()

- adds an Item object to be pointed to by the itemList array

Void removeItem()

- removes a specified Item object from itemList and points the Previously used pointer to an 'empty' Item object

Void displayList()

- displays the contents of the Item objects pointed to by the itemList array in a readable format; also
- calculates and displays the total price of the Item objects

Void extendList()

- deletes the current itemList
- creates a new itemList array (with the same name) with 1 more pointer added to the array

void match(Item,int)

- Takes an Item object (the object which is to be matched with an existing object) and an int (number of non-empty Item objects pointed to by Itemlist) as parameters
- Used by the addItem function

- Compares newly pointed to Item object name with each Item object name already pointed to by the itemList array

Void deleteList()

- Deletes the itemList array of pointers to Item objects

Item class

Private: (data members)

string name

- A string that saves the Item object name, as set by the user

string unit

- A string that saves the Item object unit name, as set by the user

Int quantity

- An integer that saves the number of items of this Item object that the user would like to purchase

Int unitPrice

- An integer that saves the price per unit of the Item object

Int extPrice

- An integer that saves the extended price of the Item object

Public: (member functions)

Bool operator==(Item &right)

- Overloads the '==' operator to be used with 2 Item objects
- Compares the two Item objects using only the name and returns true if the names are the same, and false if not

Item() (default constructor)

- Creates an 'empty' Item object by setting all data members to '0' or 'none'

Item(string, string, int, int) (constructor)

- Takes a string (item name), another string (item unit name), an int (quantity of item to purchase), and another int (unit price of item) as parameters
- Creates a non-empty Item object by saving user input as the data members

String getName()

- Returns the name (as a string) of an Item object

string getUnit()

- Returns the unit name (as a string) of an Item object

int getQuantity()

- Returns the quantity of the item the user would like to purchase (as an int) of an Item object

Int getUnitPrice()

- Returns the unit price (as an int) of an Item object

Int getExtPrice()

- Returns the extended price (as an int) of an Item object

**Testing plan**

| Test | Input Values | Expected Outcome | Observed Outcome |
|---|---|---|---|
| Input for main menu out of range | Input > 2 or input <1 | Output prompt for invalid input and loop back to variable input | Output prompt for invalid input and loop back to variable input |
| Input for main menu of different data type | 'm' or '5.3' | Output prompt for invalid input and loop back to variable input | Output prompt for invalid input and loop back to variable input |
| Input for program menu out of range | Input > 4 or input < 1 | Output prompt for invalid input and loop back to variable input | Output prompt for invalid input and loop back to variable input |
| Input for program menu of different data type | 'm' or '5.3' | Output prompt for invalid input and loop back to variable input | Output prompt for invalid input and loop back to variable input |
| Input for any prompted user int input in the addItem function is of different data type | 'm' or '5.3' | Output prompt for invalid input and loop back to variable input | Output prompt for invalid input and loop back to variable input |
| Program correctly displays list of Item objects, uses the addItem function properly, and uses the removeItem function properly | Add 6 items to itemList, remove 2 items, add 3 items | The displayList function should display 4 of the 6 original items and the 3 added items of itemList | The displayList function displayed 4 of the 6 original items and the 3 added items of itemList |
| Program switches between the main menu and the program menu correctly | Go to the program menu, add 2 items, return to the main menu, return to the program menu, display itemList contents | The display should show the 2 Item objects added to itemList | The display showed the 2 Item objects added to item list |
| Program deletes all dynamically allocated memory before exiting | Execute program with valgrind, Add 8 Item objects to itemList, exit | Valgrind should show no memory leaks | Valgrind showed no memory leaks |
| Program correctly removes an item from itemList and moves every following item up 1 in the list | Add 5 items to itemList, remove the 2nd item, display itemList | The display should show that the 2nd item entered into itemList is removed and the following items moved up (totaling 4 items) | The display showed that the 2nd item entered into itemList is removed and the following items moved up (totaling 4 items) |
| Program correctly matches a new Item object with an existing Item object using the overloaded '==' operator | Add 2 items to itemList, add a third item with the same name as the 2nd item, choose to update the item information, display itemList | The display should show 2 items with the 2nd item having the most recently entered data | The display showed 2 items with the 2nd item having the most recently entered data |

**Reflection Statement**

I learned many useful aspects of the c++ language with this creating this program. Namely, overloading operators. One challenge I came across was that I needed an organized and efficient way to display the different options of this program. I resolved this issue by reworking my menu function to have to different menus to display, a main menu for starting and closing the program, and a program menu for executing any of the functions the program offers. This made my main function very organized as well. Another challenge was creating more dynamically allocated space in the array when needed. I resolved this by creating a temporary list in the addItem function and then calling the member function extendList which simply deleted the old itemList and created a new one with 1 more pointer added. Yet another challenge was that I need to make sure the displayList function and the removeItem function did not attempt to retrieve the value of itemList that was out of bounds, resulting in a segmentation fault. This was resolved by creating an int data member of the List class named listNumber that starts at 0, is incremented whenever a new Item object if pointed to by itemList, and is decremented when an Item object is no longer pointed to by itemList.