Chermaine Cheang
Assignment 2 Design

Implement a grocery shopping program in which user can add item into the list by asking them for the item's name, item's unit of sale, number needed and unit price, and user can remove item from the list. The program then can display each item in the list with their extended price and also the total cost of all the item in the list.

In order to implement this program, I need a class for item object which will hold the information of the item (item's name, item's unit of sale, number needed, unit price and extended price). I also need a class for list which will hold an array of pointers to Item object.

Functions that I need in my Item class are setter and getter for each data member, and a function to calculate extended price of each item.

```
void extendedPrice(){
        this->extendedPrice = this->numberNeeded * this->unitPrice
}
```

Since I need a dynamic array to hold Item objects and the initial size of the array is 4, every time a List object is created, the default constructor will dynamically allocate memory in the heap to hold 4 Item objects. Hence, my default constructor will look like this.

```
List(){
        Size = 4;
        array = new Item[4];
        totalList = 0;
}
```

I will need a function to initialize every cell of the array to a NULL pointer after it is created. I need to keep track of whether my array is full. If my array is already full, user will not be able to add another Item into the list. Therefore, to allow user to continue adding item into the list even if the list is full, I will need to dynamically allocated an array with bigger size, copy Items already in my array to this new array and then delete the old array to free the allocated space.

To check if my array is full, I will have a function called checkArray in my List class.

```
bool checkArray(){
        for (int i=0; i<array_size; i++){
                if (array[i]==NULL)
                        return false;
                else
                        return true;
        }
}
```

This function will loop through the entire array and determine if any particular cell in my array is NULL pointer (not holding any address of Item object). If there is a cell who is a NULL pointer, then the array is not full and we can proceed with adding item into the array. If there is no cell in the array that is a NULL pointer, then all the cell in the array is pointing to an Item object, hence, I need to expand my array first before I can add item into the array.

To expand my array, I will have a function called expandArray() in my List class.

```
void expandArray(){
        Item* temp = new Item[size+1];
        for (int i=0; i<size+1; i++){
                if (array[i] != NULL){
                        temp[i] = array[i];
                }
                else
                        temp[i] = NULL;
        }
        delete [] array;
        array = temp;
        setSize(size + 1);
}
```

After making sure that I have space in my list, I can prompt user for the Item's information and add that item into the list.

```
void addItem(string name, string unit, int num, double price){
        for (int i=0; i<size; i++){
                if (array[i] == NULL) {
                        array[i]->setName(name);
                        array[i]->setUnit(unit);
                        array[i]->setNumber(num);
                        array[i]->setPrice(price);
                }
        }
}
```

I will have a menu option display on my screen at the beginning of the program to allow user to choose what they wish to do. My menu will have option for adding item to the list, removing item from the list, displaying the list and also an option for quitting the program.

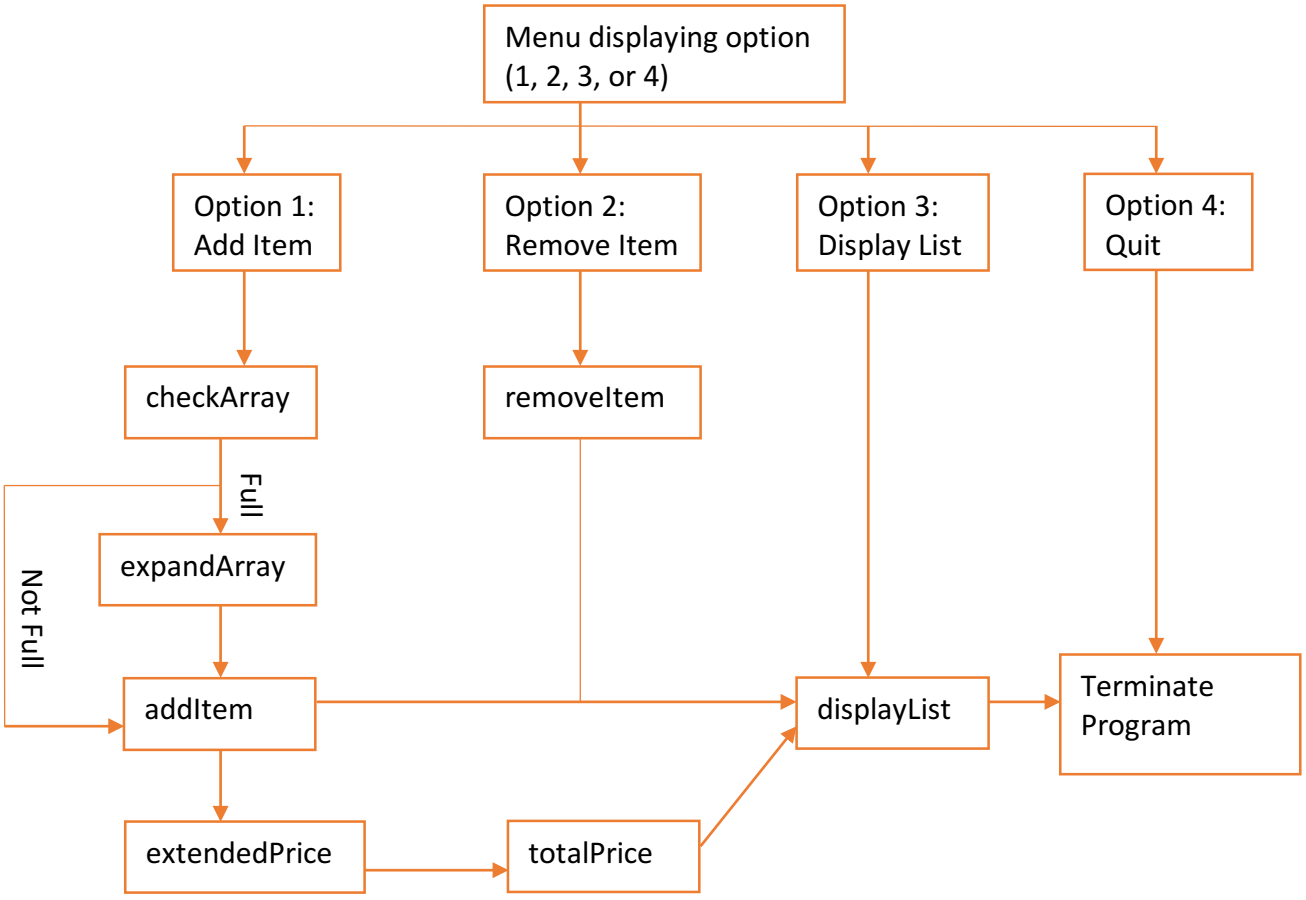
Starting Menu
1. Add item to shopping cart
2. Remove item from shopping cart
3. Display item in shopping cart
4. Quit

Enter integer 1, 2, 3, or 4 as your choice.

If user chose to add item, then I will first check if the array is full. If the array is full, then I will create a new array which is bigger in size before asking user for the item details. If the array is not full, then I will directly ask user for the item details. After getting the details, I will add that item into the list.

If user chose to remove item, then I will prompt user for the name of item to be removed and the number to be removed. After getting the details, I will remove the item from the list based on the number to be removed and number we have in the list. If all quantities of that item were to be removed, I will remove all of it from the list and set that cell where the item is in to NULL. If only some amount of that item were to be removed, I will update the current number of that item.

If user chose to display item in cart, then I will display all item in the list including their extended prices and also the total cost of all items in the cart. If user chose to quit, then I will terminate the program.



After I have my List and Item classes working correctly, as per instruction, I have to overload the == operator. Hence, in my List.hpp I will add a friend member function to overload the == operator, and define the function in List.cpp. However, since this is a friend function, I will not have scope operator in the function definition.

```
friend bool operator==(string a, string b); //function prototype in List.hpp

bool operator==(string a, string b)     //function header in List.cpp
{
        return !(a.compare(b));
}
```

Since I need to compare two strings, I will use the string class built-in compare function. When two strings are equal, the function will return 0. Therefore, I place a "!" in front of the function call so that it will return true when two strings are equal, and false when two strings are not equal.

When user tried to add item already in the list. I will have another menu option to ask user whether he or she wants to add the same item again. If user wants to add the same item again, I will check if the item has the same unit of sale, if the item has the same unit of sale, I will just update the number needed to buy. If the item has a different unit of sale, I will treat the item as a different item and add it to the list.

Testing Plan

Test	Input values	Driver function	Expected outcome
1	Display starting menu	startMenu()	Starting menu displayed on screen.
2	Display starting menu, user chooses 4 (quit)	startMenu() intValidate()	Starting menu displayed on screen. User's choice is valid. Program terminates.
3	Display starting menu, user chooses 5	startMenu() intValidate()	Starting menu displayed on screen. Invalid input. Prompt user to re-enter choices.
4	Display starting menu, user chooses 1	startMenu() intValidate() userAdd() addItem()	Starting menu displayed on screen. Valid input. Prompt user for item's details. Add item to list
5	Display starting menu, user chooses 2	startMenu() intValidate() userRemove() removeItem()	Starting menu displayed on screen. Valid input. Prompt user for item's details. Remove item from list
6	Array is full. User wants to add item to list	checkArray() expandArray()	Return true. Create a new array with bigger size.

		userAdd() addItem()	Prompt user for item's details Add item to list
7	User wants to display list	displayList()	Print out content of List and total price of all item in the list.
8	User added bread to the list. User trys to add bread again.	Operator==()	Print out "Item already in list"

Pseudocode

//item.hpp

class Item
{
    private:
        variable to hold item's name (string)
        variable to hold item's unit of sale (string)
        variable to hold number needed (int)
        variable to hold unit price (double)
        variable to hold extended price (double)
    public:
        default constructor
        constructor
        function to calculate extended price
        setter function for each variables
        getter function for each variables
};

//Item.cpp

Item::Item(){
    Set item's name to empty string
    Set item's unit to empty string
    Set number to buy to 0
    Set unit price to 0
    Set extended price to 0
}

Item::Item(string name, string unit, int num, double price){
    Set item's name to name
    Set item's unit to unit
    Set number to buy to num
    Set unit price to price
    *Set extended price to num*price*
}

```
 void Item::extendedPrice(){
                this->extendedPrice = this->numberNeeded * this->unitPrice
        }

void Item::setItemName(string name){
        set item's name to name
}

void Item::setItemUnit(string unit){
        set item's unit to unit
}

void Item::setNumberNeeded(int num){
        set number to buy to num
}

void Item::setUnitPrice(double price){
        set unit price to price
}

string Item::getItemName(){
        return item's name
}

string Item::getItemUnit(){
        return item's unit
}

int Item::getNumberNeeded(){
        return number to buy
}

double Item::getUnitPrice(){
        return unit price
}

double Item::getExtendedPrice(){
        return extended price
}

//List.hpp
class List
{
        private:
```

```
        pointer variable to Item object
        variable to hold size of array
        variable to hold total price
    public:
        default constructor
        function to check if array is full
        function to expand array
        function to prompt user for details of item to be added
        function to add item to list
        function to prompt user for details of item to be removed
        function to remove item from list
        function to display content of list
        function to calculate total price of items in list
        function to initialize array
        setter functions for data members
        getter funcitons for data members
};

//List.cpp
List::List(){
    Set size to 4
    Dynamically allocated an array of 4 item
    Set total price to 0
}

bool List::checkArray(){
        for (int i=0; i<array_size; i++){
            if (array[i]==NULL)
                return false;
            else
                return true;
        }
    }

void List:: expandArray(){
        Item* temp = new Item[size+1];
        for (int i=0; i<size+1; i++){
            if (array[i] != NULL){
                temp[i] = array[i];
            }
            else
                temp[i] = NULL;
        }
        delete [] array;
```

```
                array = temp;
                setSize(size + 1);
        }

void List::userAdd(){
        prompt user for item's name
        prompt user for item's unit
        prompt user for number to buy
        prompt user for unit price
        call addItem function with information gathered from user
}

void List::addItem(string name, string unit, int num, double price){
        for (int i=0; i<arraySize; i++)
                if (array[i] is empty)
                        create Item with parameters passed in using item constructor
}

void List::userRemove(){
        prompt user for name of item to be removed
        prompt user for number of item to be removed
        call removeItem function with information gathered from user
}

void List::removeItem(string name, int num){
        int number;
        for (int i=0; i<arraySize; i++) {
                if (array[i].getItemName() == name) {
                        number = array[i].getNumberNeeded() – num;
                        if (num == 0)
                        {
                                delete array[i];
                                array[i]=NULL;
                        }
                        else
                                array[i].setNumberNeeded(number);
                }
        }
}

void List::displayList(){
        for loop to loop through every element in array
                cout item's name
                cout item's unit
```

```
                cout number needed
                cout unit price
                cout extended price
        cout total price
}

void List::totalPrice(){
        for loop to loop through all element in array
                if element is not NULL
                        add this element extended price to total price
}

void List::initialize(){
        for loop to loop through every element in array
                set each element to NULL
}

void List::setArraySize(int size){
        arraySize = size
}

int List::getArraySize(){
        return arraySize
}

double List::getTotalPrice(){
        return totalPrice;
}
```

//validate.cpp

```
#include <iostream>

bool intValidate(int input){
        bool valid;
        if (input <= 0){
                cout << "Invalid input. Number must be greater than 0." << endl;
                valid = false;
        }
        else
                valid = true;
        return valid;
}
```

//menu.cpp

```
#include <iostream>

int startMenu(){
        int choice;
        bool valid;
        cout << "What do you wish to do?" << endl;
        cout << "1 – Add item to shopping cart" << endl;
        cout << "2 – Remove item from shopping cart" << endl;
        cout << "3 – Display items in shopping cart" << endl;
        cout << "4 – Quit" << endl;
        cout << "Enter 1, 2, 3 or 4 as your choice." << endl;
        cin >> choice;
        valid = intValidate(choice);
        while (!valid) {
                ask user to reenter choices
                update valid
        }
        return choice;
}

//main.cpp
int main(){
        create list object
        do {
                display starting menu
                validate user's choice
                get user choices
                if user wants to add item {
                        prompt user for item's details
                        check if array is full
                                if array is full
                                        expand array
                        add item to list
                }
                If user wants to remove item {
                        prompt user for item's details
                        remove item from list
                }
                if user wants to display list {
                        displayList();
                }
        } while user's choice is not quit
        return 0;
}
```