Brian Palmer
palmebri@oregonstate.edu
October 23, 2016

Assignment 2

Goal:

The objective of this project was to create a shopping cart list that could handle mutations such as adding items, removing items, extending the length of the list, and shortening it. The user also should also be able to display the contents of the list. Since we know the size of the list and items will be mutable during runtime, this project will require a lot of dynamically allocated memory.

PseudoCode

1. Add Item
    a. Get user inputs
        i. Get name of item from user
        ii. Get cost from user
        iii. Get quantity from user
        iv. Get unit from user
    b. Build Item object
    c. Check if item already exists in the list by comparing names *
        i. If an object already exists,
            1. add to that object's quantity
            2. Delete new object
        ii. If it does not exist
            1. Find the next NULL pointer in the list
            2. Assign the list pointer to the object
    d. Calculate the total cost
2. Remove Item
    a. Get user inputs
        i. Get name of item from user
        ii. Get number of items to remove
    b. First try to find object in list by comparing names *
        i. If there is a match
            1. Delete the item
            2. Maybe shorten the list
        ii. If there is no match
            1. Do nothing
    c. Calculate the total cost
3. Print List

a. Iterate through the list and print off each items members
b. Print out the number of items
c. Print out the length of the list

List class

- *Private*
  - Item ** List: This member will hold an array of Item pointers.
  - Int num_items: the number of items that are in the list. Will be updated everytime and item is added or removed.
  - Int size: The size of the array. Used to create a new array twice the size of the original when the array is completely filled, or to create an array half the size of the original if items only fill half the list.
  - Int total_cost: The total cost of all the items in the list
- *Public:*
  - **Setters and getters**
  - **AddItem():** this function collects user's inputs to create an item, checks if that item already exists in the list, and then will either update the quantity of the item in the list if the item does exist, or will add the users item to the end of the list. Before adding the item, the function will determine if the list needs to be extended first or not. Finally, the total cost of all the items will be updated.
    - **itemName()**: This function gets the name of the item that the user entered and validates that it only contains letter characters and spaces.
    - **maybeExtendList():** checks to see if the list is full of items. If so,
      - **extendList():** Creates a new dynamically allocated array in memory that is twice the size of the current list
    - **findMatch():** This function takes an Item-pointer parameter and compares its name member to each item in the list. If there is a match, the item in the list is returned. If there is no match, a NULL Item-pointer is returned.
      - In the *AddItem()* function, the item pointer parameter is initialized with the newly constructed item from the user's inputs
      - In the *RemoveItem()* function, a temporary item is created and is given the name of the item that the user wants to mutate. The temporary item initializes the parameter, and is deleted at the end of the function.
    - **AddToQuantity():** this function takes an Item-pointer and an integer as parameters, and adds that integer to the item pointers quantity member. If the user enters an item that already exists in the list, this function updates that items quantity with the quantity that the user entered.
    - **SubtractFromQuantity():** This function takes an item pointer and an integer as parameters and subtracts the quantity of that item by the integer.

        - **AddToEnd()**: this function takes an item pointer parameter and adds it to the first NULL pointer in the list member. It is used if the user is adding an item to the list that does not already exist there.
        - **CalcTotalCost():** This function updates the total cost member by finding the sum of all the items external price members.
    - **removeItem()**: This function will remove an item from the list, reduce the quantity of an item in the list, or do nothing and display a message if the item the user wants removed does not exist in the list.
    - **sort():** This function goes through the list member looking for NULL pointers that have items one iteration forwards. This is likely to be the case after an item is removed from the list. After this function is finished, all the NULL pointers should be grouped together at the end of the array (from using the swapItems() function).
    - **swapItems():** Switches an item in the list with another item.
    - **MaybeShortenList():** This function checks if the quotient between the number of items in the list and the size of the list equals two. It so, then a new array half the size of the original is created and replaces the original list.
    - **ShortenList():** Replaces the current array with a new array that is half the size of the original.

Item class

- *Private:*
    - String name: the name of the item
    - Unit unit: the items units
    - Int quantity: the number of items
    - Double unit_price: the price of the unit
    - Double ext_price: the total price of that item (quantity * price);
- *Public:*
    - **Setters and getters**
    - **Operator==(const Item &obj):** this function compares two items names. If they are the same, returns true. Else, it returns false.
    - **calcExtPrice():** This function multiplies the quantity of the item with the unit_price member and sets the ext_price member with the product.
    - Item constructor(name = “”, quantity = 0; unit_price = 0.00, Unit = CAN): The default values are used as fillers in the temporary object that is created when the user goes to remove an item.

Tests:

Tests	Input	Expected Output	Pass (y/n)	Comments
Adding an Item				
The user can add an item successfully.	(name = apple, quantity = 1(	Item will be created with mem	Y	
The user can add multiple items to the list	Create multiple items with different names	A list of all the items with their names dispalying	Y	
Test adding and Item that already exists in the list	Add an item with the same name as the first item in the list	The quantity should change to show the sum of the original quanity and the new quantity	Y	If the user enters two items with the same name but with different prices, the calculated extendable price will reflect the price of the original item.
Total cost changes accordingly	Add some items to the list	For each item added, the total cost reflects the sum of all the item extended price	Y	
Removing an Item				
If the user tries to remove an item from the list that does not exist, nothing will happen	Enter the name of an item that doesn't exist in the list	program should let uesr know that the item could not be found	Y	
User can remove an item successfully	name of an item that exists in the list	item should be removed with no segmentation faults	N	There was a segmentation fault. **UPDATE: a new approach for deleting the items was used that was successful. This can be read in the reflections section of the design document.**
User can reduce the quantity of an item	User enters the name of an item that is in the list and is asked how many items to remove. The user enters the highest number.	The quantity member of the item in the list should reflect correct difference	Y	
Print				
Print function display each items name, quantity, units, price, and extended price, as well as the total cost, number of items in the list, and size of the list	None	nber appropriately with zero se	Y	

Reflection:

In my original design, I decided that my add item and remove item functions would both need to traverse the list to see if an item exists in the list. If a match was found, the function would return the item (Item *). The add and remove functions might then mutate the quantity of the item, or the remove function could delete the dynamic memory. The former cases passed their case tests; the latter however caused a memory leak.

The memory leak could be because the “find item” function initialized another pointer to the same item in a different scope, which was used to delete the item if the items quantity reached zero. The list still contained a pointer to that allocated object and caused bugs when that pointer was referenced in other places in the program (i.e. the print function). To resolve, I created a separate function to delete items directly from the list. I made sure that my

segmentation faults were gone by running by executable with valgrind a few times. The lesson here is to be careful when creating multiple pointers to an object to try to avoid leaving dangling pointers.

New Functions:

- **Deletion()**: This function takes an item pointer parameter, finds the item in the list, deletes the item, and sets the item pointer to NULL. This function was created to resolve a segmentation fault as mentioned in my reflection section.
- **DeleteItem()**: This function takes an item pointer as a parameter and compares the value of that pointer to each item in the list. If a match is found, the item in the list is deleted and replaced with a NULL value.