Ethan Dunham

dunhamet@oregonstate.edu

2/04/17

Project 2 design/reflection

Problem:

The problem is to create a grocery shopping list. There will be an Item Class that consists of items, their weight/size/type, number to buy, and the price per unit. I need to use a dynamic array to add a List object for each Item object created. My program must be able to create a list, add items, remove items, and display the shopping list. It will also display the name, unit/weight/size/type, quantity, unit price, and full price. Finally, the program will test if an item is already in the list before adding a new one using an overload of ==.

Design:

The main function will consist of a menu. First, it will create a list. That menu will give options to add items, remove items, display the list, and exit.

When “create a list” is picked, the menu will create a new list using new. The constructor for list will create an array of size 4 that is empty.

When “add items” is picked, the user will be sent to the Item constructor.

At the item constructor, it will ask for the information about the item.

Back in the “add items” function, it will pass a pointer to a pointer to the newly created Item to the addToList function in List.

In addToList(**item), the item will be added to the array. If the array is now over 4 items, it will initialize a new array 1 size larger and increase the number of items variable in the class. It will also then make a 2nd array and initialize those values to reference when it needs to advance in size again. That way the array can remain the same name. The new array will be initialized, then a new array of the original name will be created to then retransfer the items back to the original array, only 1 size larger.

/*Add a way to not readd items of the same name to the list using overridden ==.*/

“Remove items” will prompt you to enter the item name to delete and how many of that item you want to remove. It will then call the ListDelete function.

ListDelete(string, int) will be a friend of Item. This function will compare the names of each items in the list. Once it finds the item, it will have the array location of that Item object. It will then use its friendship to access the item quantity. If the number to delete is equal to or more than the quantity, it will delete the object from the list and reallocate the array to be 1 smaller without that object. If the quantity to delete is less than the quantity on the list, it will subtract the amount to delete from the variable, while keeping the Item in the array.

Display the list will call the Display() function of List. It will also be a friend to Item. It will display the information for each variable as it loops through the array. It will also add up the total by multiplying price by quantity while it loops and display it as a fixed floating point number.

Class Item
Private:
double Itemprice;
string ItemName;
string ItemType;
int quantity;

Public:
Item(string, string, string, int, double, List&);
Friend void List::Display();
Friend void List::ListDelete(string, int);

Class List
Private:
Item listArray[4];
Item oldListArray[4];
int currentArraySize = 4;
public:
Void addToList(item&)
Void ListDelete(string, int)
Void Display()

Input	Function	Expected results	Results
Item name with spaces	Add item	Program will save the entire string.	Program saves the string up to 50 characters long, including space.
Asks to delete more than quantity on list	listDelete	Deletes item from array in list and deletes the item	Deletes the item from the list
Asks to delete less than quantity on list	listDelete	Decreases quantity in item by the desired amount	Decreases quantity on the list
Puts negative number for cost.	Item()	Loops back to ask for price.	Asks for another input.

Reflection:
I ran into some problems with adding the objects to an array. I realized that passing pointers to the function was not the way to solve this. It would just update every item in the cart when I added a new item, showing duplicates. Instead, I passed an object and let the copy constructor do its magic. This solved that problem. Other than that issue, the program went together pretty smoothly. When I ran Valgrind, I realized that passing an object caused a memory leak. I was under the impression that the

copy constructor also had a default copy deconstructor. I changed the addToList function to take all the variables for an Item object and then create the object within the function. This solved the memory leak.