Project 2 Reflection

Byron Kooima - 932707491

CS/162

July 23, 2017

Prof. Luyao Zhang

Project 2 Reflection

Lab Description

The main focus of this program is to implement a grocery shopping list and maintain and display a list of items. The Items in the List must contain the following:

1. Item name
2. Unit (i.e. can, box, pound, or ounce)
3. Quantity to Buy (int)
4. Unit Price

The following options will be made available to the user:

1. Add items to the Shopping List.
2. Remove items from the Shopping List.
3. Print the Shopping List.

Requirements

- Make a List class
- The List is made up of Items
- Items are made up of the following information:
    - Item name
    - Unit
    - Quantity to buy
    - Unit price ($/unit)
- List has the following information:
    - Store Item objects to the List object
    - Option to remove Item objects from the List
    - Option to display the List (item name, item unit, quantity of units, unit price, extended price for each item, & total price of all items)
    - Dynamic array initialized to 4 to store the Items

- Overloaded “==” operator to perform Item test
    - Test if item is already on the List by comparing Item name
    - Allow user to decide to update the Item
        - If they want to update, replace the Item information
        - If they don’t update, return to main menu

The main nouns in the program design (and whether they are classes) are as follows:

Nouns:

- Item Objects
- User
- Shopping List
- Item name
- Item unit
- Item quantity
- Item price
- Item extended price
- Total price
- Menu

Classes:

- Item Objects -> Item
- Shopping List -> List
- Menu -> userMenu

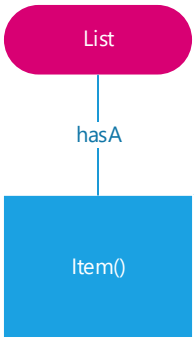
Class Design

Class Name	List (hasA Item())
Data Members	protected member variables: (none) private member variables: **Item* groceryList** **Item* aPtr** **int itemCount** **int arraySize** public member variables: (none)
Member Functions	private member functions: **double total_price()** – Calculate the total price for all of the List Items protected member functions: (none) public member functions: **List()** – Constructor: Set default values **~List() –** Destructor **Item get_Item** – Get the Item parameters from the user and return an Item object **void add_Items** – Adds Item to the groceryList array. Checks to ensure the Item name does not exist in the array. If it does, the user can update/not update the Item in the List. The function also increases the size of the List array and populates the List array with Items **void remove_Items** – User selects Item from groceryList array to delete. The array is re-sequenced if Item is removed. **void print** – Prints the entire Grocery List array for all of the Items. Also provides the Total for all Items in the List array.

Class Name	Item
Data Members	protected member variables: **(none)** private member variables: **string itemName** **string unitName** **int quantityOfUnit** **double unitPrice** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Item()** - Constructor: Set default values **Item(string, string, int, double) –** Constructor: user defined Item values **updateItem(const Item&)** – Updates an Item object with tempItem variable values **print()** – Print the current Item object parameters **get_itemName() –** Returns the itemName (string) **get_unitName() –** Returns the unitName (string) **get_Quantity() –** Returns the quantityOfUnit (int) **get_unitPrice()-** Returns the unitPrice (double) **get_total() –** Returns the Quantity * unitPrice (double) **bool Item::operator==(const Item& rhs) const –** Overloads the == operator to compare two Item names

Class Name	userMenu
Data Members	private member variables: **int selectedChoice** **vector<string> choice**
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **void add_choice(string) –** Add menu options to vector **void printMenu() –** Print the menu options **int makeChoice() –** Returns int for the user selection

Class Interactions



Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
Input too low	**Input < 0**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input not an integer	**Input = 0d**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input below range	**Input = -1**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input not a double	**Input = 34d**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
User chooses option #4 to quit the program	**Input = 4**	**userMenu -> makeChoice()**	**Exit the program normally**	**Fail: Could not exit the program. The input verification was set up incorrectly and would not allow a value of 4. Fixed to allow correct menu options.**
Add items to the Shopping List	**Item name = Dog food Units of measure = can Qty of units = 44 Item price per unit = 22**	**List->add_Items() List->get_Items()**	**Should add the Item to the List array[]**	**Fail: List was storing a pointer to the Item but the Item pointer was deleted and memory was lost Fixed the ~List() destructor to**

Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
				handle removal of dynamic memory instead of the ~Item() destructor
Add five items to the Shopping List (array out of bounds)	**Input = 4 Items with various values**	**List->add_Items() List->get_Items()**	**Should resize List array and store all five Items to the List array**	**Fail: The array went out of bounds when fifth item was added. Fixed the checker to make sure the array was resized BEFORE the Item was added.**
Add Item with the same name to the List (override ==). User selects not to override the Item.	**Input = 2 Items with the same name but different values User chooses not to update (return to main menu)**	**List->add_Items() List->get_Items() Item->updateItem()**	**The program should prompt the User to override the existing Item or leave existing Item in List. If the user selects "N", the main menu is displayed**	**Fail: The updateItem function was overwriting the Item object regardless of the User input. Fixed the Boolean logic in the loop to make sure the == operator was returning the correct state.**
Add Item with the same name to the List (override ==). User selects to override the Item.	**Input = 2 Items with the same name but different values User chooses to update (Item is overridden)**	**List->add_Items() List->get_Items() Item->updateItem()**	**The program should prompt the User to override the existing Item or leave existing Item in List. If the user selects "Y", the Item is updated and the main menu is displayed**	**Pass after the "N" selection was fixed. List Item was updated with new information**
Remove item from the Shopping List	**menuOpt = 2 No Items exist**	**List->remove_Items() Item->get_itemName()**	**The program should prompt the user that no Items exist in the List.**	**The prompt indicates the List is empty**
Remove item from the Shopping List	**menuOpt = 2 Three Items are loaded into List. User selects Item 2 to be removed.**	**List->remove_Items() Item->get_itemName()**	**The program should prompt the user for the Item in the List they would like to remove. When the user selects number 2 to be removed, the array is re-indexed with Item 2 removed and the**	**Fail: The array went out of bounds when trying to shift the values. Fixed the array index to grab the correct List Item to start shifting from.**

Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
			User is informed on which Item was removed.	
Print the Shopping List	**menuOpt = 3 No items exist**	**List->print() Item->print()**	**The program should prompt the user that no Items exist in the List.**	**The prompt indicates the List is empty**
Print the Shopping List	**menuOpt = 3 Three Items are loaded**	**List->print() Item->print()**	**The entire contents of the List array should be printed to the command window**	**Partial pass: The information was printed to the screen, but the formatting made the output difficult to read. Fixed the setw() of each of the output lines to align the text for readability.**

Reflection

Project 2 was fairly straight forward but I had some initial issues with overloading the "==" operator. In passing just one Item name to the operator, it made the function a little more difficult to implement. Through examples online and reading in the book, I found a fairly simple solution and it worked fairly well. Additionally, I was trying to compare the Item name, Item qty, and Item price before checking with the User if they wanted to update the Item in the List array. This wasn't necessary and I found that just comparing the name first allowed me to check with user on just two possible options.

Based on my previous lab, I opted to simplify my Main cpp file so that it just called class functions based on the user choice. This made the code more readable and the Classes were more integrated. I also used a similar Print function in the List and Item class so that it made the loop to print the entire List re-usable.

The formatting of the output caused me the most issues and I spent a significant amount of time trying to make it readable for the end user. User experience is very important when it comes to dealing with customers and I didn't want my List to be unreadable. After playing around with the setw() option, I came up with a fairly robust system and the output looked very respectable. If I had more time, I would have implement additional functions to make check the length of the Item name and Unit name strings. I could have used that information to refactor the table and provide an even more robust printing scheme. I also realized that my menu functions would have been better served inside the Game class.