David Mednikov

CS 162 W17

onid: mednikod

Project 2 Reflection

Class Design

2 classes: Item and List

Item class

Member variables:

- string itemName
- enum unit type
- int quantity
- double price

Member functions:

- Default constructor
- Constructor with inputs
- int getQuantity()
- double getPrice()
- string getName()
- string getUnitString()
- overloaded == operator

List class

Member variables:

- Item* itemList (array of Items)
- int listLength

Member functions:

- Default constructor
- Destructor
- void addItem()
- void removeItem()
- void displayList()

Test Plan

Test Case	Input Values	Expected Outcomes	Observed Outcomes
Spaces in name	String itemName = "ice cream"	Program accepts input Item name is stored as "ice cream"	Program accepts input Item name is stored as "ice cream"
Calculate extended price	Quantity = 12 Price = $2.99	Extended price = 35.88	Extended price = 35.88
Calculate total	Item1 extPrice = 35.88 Item2 extPrice = 4.09 Item3 extPrice = 10	Total price = 49.97	Total price = 49.97
Remove matching item	itemName = cereal	Remove Item called "Cereal" from list	Remove Item called "Cereal" from list
Catch duplicate item	itemName = steak	Tell user that item already exists, loop back to main menu	Tell user that item already exists, loop back to main menu
Make sure unit input is valid	Input = "pond"	Tell user that item type is invalid and ask again	Tell user that item type is invalid and ask again
Print or remove item before initializing list	Menu choice = 2 or 3	Tell user that list has not been initialized yet	Tell user that list has not been initialized yet
Increase array size by 1	listLength >= 4 Item = cheese	Create new array of listLength + 1 elements and add cheese to the end	Create new array of listLength + 1 elements and add cheese to the end

Test Results

All of my tests passed. Using getline(cin, name), I am able to capture spaces in the item name. I use accessor variables to calculate the extended price and keep a running total of the extended prices to calculate the total. I match strings to remove items from the list, and use an overloaded == operator to test for duplicate items. I use input validation to make sure that the user enters a proper unit type. I also designed the program so that it doesn't print the list or try to remove any items if no items have been added. Last (and this was the hard part) if the length of the list is >= 4 and I want to add an element, I increase the size of the array by one element. Likewise, if the length of the list is >4 and I want to remove an element, I decrease the size of the array by one element. This requires dynamic allocation and copying arrays using a temp variable.

Comments

I had a good idea how to design this program. The basis was pretty simple: I needed an Item class that had all of the item variables and accessor methods to get those variables from the class. I also needed to overload the equality operator (==) by comparing the itemName strings within two Item objects.

Creating the List class was more difficult because it wasn't as easy as creating variables and their accessors. I had to fully design functions that would add an item, change the array size, check for duplicates, remove an item, and make the array smaller. I was getting segmentation fault errors early on and I realized that it was because I was dynamically allocating the same variable twice. It took me a while to figure that out. I also had to do a lot of debugging when increasing or decreasing the size of the error because I often found myself going out of range, usually because I had a loop trying to access an array element past the limit of the array. Adding items was pretty easy once I got rid of the segmentation faults. However removing items gave me some trouble as well because the program would often crash when I would try to replace an item in an index with the item in index + 1.

I also tweaked my displayList() function a good amount to make it readable with nice table-like formatting. I learned a lot about the <iomanip> library and was able to make it look better as I got more done.

The last issue I had was memory deallocation. I spent 2 hours finding the 1 allocated byte I wasn't freeing. But I was able to debug the program step by step and figure out which item I wasn't deleting.