Project 2

The grocery shopping list program maintains a list of items in a customer's shopping cart. A customer is given an option to add an item to the cart, remove an item from the cart, and display the current items in the cart. If a user adds an item name that already exists, they are given the choice to replace the item or do nothing and return to the menu.

I designed this program by creating an Item class and a List class. My Item class has itemName, unitType, quantity, unitPrice, and extendedPrice as member variables. The class also contains get methods for each member variable. My item class functions to store user input into the appropriate variables to work alongside my List class.

My List class contains the members Item** shoppingList, cartCapacity, and itemsInCart. The methods are printList(), which displays the items in the user's cart, addItem(), which adds an item to the user's cart, removeItem(), which removes an item from the user's cart, increaseCartcapacity(), which increases the cart capacity after its initial capacity of four is exceeded, and the operator== function, which compares a list object to an item object and returns true if the name of the item matches an already existing item in the customer's current shopping list. The input that is received from the user assigns a value to the private variables stored in the Item class. The List class accesses these variables through get methods.

The program also utilizes a menu class and input validation functions. The menu class displays the menu at the start of the program, which is displayed repeatedly until the user chooses to exit the program. The input validation functions include a string input validator, which only accept characters and no spaces, and an integer input validator, which only accepts

integers. These functions are used each time a user is prompted to enter in either string or integer input.

My program starts by displaying a menu with a list of four options to the user. The user is prompted to select an option from the menu. The options are add item, remove item, display list, and exit. The program is designed to continuously display the menu and prompt the user for input until the user chooses to exit out of the program. If the user chooses to add an item, the add item() function will execute, and the user will be prompted to fill out the item name, unit, quantity and unit price for that item. If the user enters an item name already in the list, the user will be asked to either replace the existing item or resume shopping and return to the menu. If the user chooses to remove an item, the user will type in the name of the item they want to remove and the removeItem() function will execute. If the user chooses to display the list, the printList() function will execute and show all items in the user's cart. The exit option ends the program.

While implementing my program, I had trouble figuring out how to increase my dynamically created array. I found this task difficult because I was used to working with vectors, which have a built-in functionality to increase in size. After doing some research, I decided that the best solution would be to create a temporary array with a capacity increased by four and transfer all the items from the original array into the temporary array. I then deleted my old array and renamed my temporary array to have the same name as my original array. Doing so allowed me to increase the size of my array correctly without affecting the items in the original list.

I also had trouble creating the logic for my addItem() function. The number of factors the addItem() function needed to account for made creating this function challenging. I needed to collect the user input for the item name, quantity, unit type, and price. I then needed to create a

new item object from this input. My program needed to assess the newly created item object and determine whether it already existed using the operator function. If it did, I needed to give my user an option to replace the already existing item or resume shopping. To accomplish this, I created an operator== function that compared an implicit list object with the newly created item object. The contents of the function would discern if the name of the item object already existed in the array. If it did, a value of true would be returned. Otherwise, false would be returned. In my addItem function, the user would be prompted to replace an existing item only if the operator= function returned a true value. It was difficult for me to initially decide what the return type my operator function should be. After looking at examples in the textbook, I concluded that making my operator function return type a Boolean was an effective way to accomplish what I wanted to do.

I also needed to account for when my items would exceed the initial array capacity of four in my addItem() function. If the number of items equaled my array capacity, I had to expand my array and then include the item in the list. I initially wanted to make my addItem function responsible for directly increasing the array capacity of my shopping list. However, after much deliberation, I realized my code would be more modular and readable if the array capacity was increased in a separate function and called within the addItem() function. Doing so allowed my addItem() function to look more streamlined. It was thus challenging for me to develop the logic for the addItem() function and arrange the pieces of code in a proper format to execute correctly. I had to carefully consider the sequence that involved an item being introduced to the cart, checking to see if the item already existed, and then checking to see how the current items in the cart compared to the cart's current capacity.

I resolved my problems by writing multiple drafts of pseudocode. First, I started with an initial draft of what I wanted each function to accomplish. Each subsequent draft revision addressed all the factors the functions needed to address in greater detail. By my fifth pseudocode draft, I was able easily translate my pseudocode into a programming language.

Creating the grocery list program presented several challenges that I had to confront to create a successful program. Although I initially ran into many problems in regards to how to create the logical framework for some of my functions and how to accomplish certain tasks using arrays instead of a vector, I accomplished creating a workable program by researching dynamically created array functionality and creating and revising multiple drafts of pseudocode until all the impediments in my design were thought through.

Test Case	Input Value	Driver Functions	Expected Outcome	Observed Outcome
Input characters instead of integers	sdfsdf	integerInputValidation()	Input is invalid. Please enter a valid number.	Input is invalid. Please enter a valid number.
Input special characters and an alphabetic character	$#$%%%j	integerInputValidation()	Input is invalid. Please enter a valid number.	Input is invalid. Please enter a valid number.
Input special characters and an integer	#$#$%1	integerInputValidation()	Input is invalid. Please enter a valid number.	Input is invalid. Please enter a valid number.
Input integers	2342	stringInputValidation()	Input is invalid. Please enter a valid input without any spaces.	Input is invalid. Please enter a valid input without any spaces.
Input special characters and an alphabetic character	#$#%%j	stringInputValidation()	Input is invalid. Please enter a valid input without any spaces.	Input is invalid. Please enter a valid input without any spaces.
Input characters, integers, and special characters	J$$543jjj	stringInputValidation()	Input is invalid. Please enter a valid input without any spaces.	Input is invalid. Please enter a valid input without any spaces.
Input a space	j j	stringInputValidation()	Input is invalid. Please enter a valid input without any spaces.	Input is invalid. Please enter a valid input without any spaces.
Add three items	mangoes, grapes, olives	addItem()	Three items are added.	Three items are added.
Add nine items	mangoes, grapes, olives, seaweed, berries, kiwi, oranges, apple, bananas	addItem()	Array capacity expands to accommodate this.	Array capacity expands to accommodate this.

Add existing item mangoes	mangoes	addItem()	User gets asked if he/she wants to replace the existing item or do nothing.	User gets asked if he/she wants to replace the existing item or do nothing.
Remove item *mango*, then display list		removeItem()	Item mango is removed; list displays without mango	Item mango is removed; list displays without mango
Display list of items		printList()	Displays all items in cart	Displays all items in cart
Add five items	broccoli, cucumber, beets, lettuce, pear	increaseCartCapacity()	All items get stored and list displays all items.	All items get stored and list displays all items.
Add two items, then exit program	broccoli, lettuce	addItem()	Two items are added; program successfully exits.	Two items are added; program successfully exits.