Function Name	#	Test Description	Sample Input	Expected Result	Actual Result	P/F
swap()	1	swaps the content of two structs	p1 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}, p2 = {productID: 2, availQuantity: 3, sellerID: 102, unitPrice: 7.25, itemName: "Orange", category: "Fruit", itemDescription: "Fresh oranges from local farms."}	p1 = {productID: 2, availQuantity: 3, sellerID: 102, unitPrice: 7.25, itemName: "Orange", category: "Fruit", itemDescription: "Fresh oranges from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	p1 = {productID: 2, availQuantity: 3, sellerID: 102, unitPrice: 7.25, itemName: "Orange", category: "Fruit", itemDescription: "Fresh oranges from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh	P
	2	swaps two structs with identical contents	p1 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	p1 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	p1 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	P
	3	swaps content of two structs (the other struct contains a null item)	p1 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: NULL, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	p1 = {productID: 2, availQuantity: 3, sellerID: NULL, unitPrice: 7.25, itemName: "Orange", category: "Fruit", itemDescription: "Fresh oranges from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	p1 = {productID: 2, availQuantity: 3, sellerID: NULL, unitPrice: 7.25, itemName: "Orange", category: "Fruit", itemDescription: "Fresh oranges from local farms."}, p2 = {productID: 1, availQuantity: 5, sellerID: 101, unitPrice: 10.5, itemName: "Apple", category: "Fruit", itemDescription: "Fresh apples from local farms."}	P
swapTrans()	1	swaps the content of two structs	p1 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a} p2 = {date: 4 1 2023, totalAmount: 10.00, buyerUserID: 127, buyerUserName: qwerty, sellerID: 123 sellerUserName: a}	p1 = {date: 4 1 2023, totalAmount: 10.00, buyerUserID: 127, buyerUserName: qwerty, sellerID: 123 sellerUserName: a} p2 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a}	p1 = {date: 4 1 2023, totalAmount: 10.00, buyerUserID: 127, buyerUserName: qwerty, sellerID: 123 sellerUserName: a} p2 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a}	P
	2	swaps two structs with identical contents	p1 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a} p2 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a}	p1 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a} p2 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a}	p1 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a} p2 = {date: 4 11 2023, totalAmount: 5.00, buyerUserID: 125, buyerUserName: guest, sellerID: 123 sellerUserName: a}	P
	3					
deleteContents()	1	deletes all the value in the array	aData: 1, 2, 3, 4, 5	aData will have zero value	aData will have zero value	P
	2	deletes one value in the middle of the array	aData: 1, 2, 3, 4, 5	aData: 1, 2, 4, 5	aData: 1, 2, 4, 5	P
	3	deletes value of the same number	aData: 1, 2, 3, 3, 3	aData: 1, 2	aData: 1, 2	P
getString()	1	the string contains spaces	Shopping App	Shopping App	Shopping App	P
	2	the string has special characters	Shopping_App	Shopping_App	Shopping_App	P
	3	the string is empty	""	""	""	P

getUserInfo()	1	gets password, name, address, and contact number	Password: mark Name: mark doe Address: qc Contact Number: 1234567890	Password: mark Name: mark doe Address: qc Contact Number: 1234567890	Password: mark Name: mark doe Address: qc Contact Number: 1234567890	P
	2	gets password, name, address, and contact number (password contains special characters)	Password: mark_123 Name: mark doe Address: qc Contact Number: 1234567890	Password: mark_123 Name: mark doe Address: qc Contact Number: 1234567890	Password: mark_123 Name: mark doe Address: qc Contact Number: 1234567890	P
	3	gets password, name, address, and contact number (user inputs long name and address)	Password: mark_123 Name: mark daniels john amiel aquino Address: 123 taft avenue long street name Contact Number: 1234567890	Password: mark_123 Name: mark daniels john amiel aquino Address: 123 taft avenue long street name Contact Number: 1234567890	Password: mark_123 Name: mark daniels john amiel aquino Address: 123 taft avenue long street name Contact Number: 1234567890	P
	4					
getItemInfo()	1	gets valid inputs	itemName: "Apple" category: "Fruits" itemDescription: "Fresh red apples" availQuantity: 10 unitPrice: 50	itemName: "Apple" category: "Fruits" itemDescription: "Fresh red apples" availQuantity: 10 unitPrice: 50	itemName: "Apple" category: "Fruits" itemDescription: "Fresh red apples" availQuantity: 10 unitPrice: 50	P
	2	gets a negative input	availQuantity = -4	Error message and prompt to enter quantity again	Error message and prompt to enter quantity again	P
	3	gets valid float input	unitPrice = 1.74	unitPrice:1.74	unitPrice:1.74	P
	4					
	5					
getItemString()	1	Test for choice 1 to get item name string	i->itemName = "Product A"	"Product A"	"Product A"	P
	2	Test for choice 2 to get category string	i->category = "Electronics"	"Electronics"	"Electronics"	P
	3	Test for choice 3 to get item description string	i->itemDescription = "New product"	"New product"	"New product"	P
sortProductID()	1	integers in the array are in random order	ProductID: 123, 125, 122, 135, 100, 126	ProductID: 100, 122, 123,125, 126, 135	ProductID: 100, 122, 123,125, 126, 135	P
(sorts the product ID in increasing order)	2	integers in the array are already in increasing order	ProductID: 100, 122, 123, 125, 126, 135	ProductID: 100, 122, 123,125, 126, 135	ProductID: 100, 122, 123,125, 126, 135	P
	3	integers in the array are in decreasing order	ProductID: 135, 126, 125, 123, 122, 100	ProductID: 100, 122, 123,125, 126, 135	ProductID: 100, 122, 123,125, 126, 135	P
sortBuyerIDTransaction()	1	integers in the array are in random order	sellerID: 123, 125, 122, 135, 100, 126	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
	2	integers in the array are already in increasing order	sellerID: 100, 122, 123, 125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
	3	integers in the array are in decreasing order	sellerID: 135, 126, 125, 123, 122, 100	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
sortSellerID()	1	integers in the array are in random order	sellerID: 123, 125, 122, 135, 100, 126	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
(sorts the seller ID in increasing order)	2	integers in the array are already in increasing order	sellerID: 100, 122, 123, 125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
	3	integers in the array are in decreasing order	sellerID: 135, 126, 125, 123, 122, 100	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
sortSellerID1()	1	integers in the array are in random order	sellerID: 123, 125, 122, 135, 100, 126	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P

(sorts the seller ID in increasing order)	2	integers in the array are already in increasing order	sellerID: 100, 122, 123, 125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
	3	integers in the array are in decreasing order	sellerID: 135, 126, 125, 123, 122, 100	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
sortSellerIDTransaction()	1	integers in the array are in random order	sellerID: 123, 125, 122, 135, 100, 126	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
(sorts the seller ID in increasing order)	2	integers in the array are already in increasing order	sellerID: 100, 122, 123, 125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
	3	integers in the array are in decreasing order	sellerID: 135, 126, 125, 123, 122, 100	sellerID: 100, 122, 123,125, 126, 135	sellerID: 100, 122, 123,125, 126, 135	P
showLowStock()	1	testing when an item stock is greater than 5	i[current][j].availQuantity = 8	displays "no low stock item"	displays "no low stock item"	P
	2	testing when an item stock is less than 5	i[current][j].availQuantity = 3	displays the item information	displays the item information	p
	3	testing when an item is equal to 5	i[current][j].availQuantity = 5	displays "no low stock item"	displays "no low stock items"	P
changeQuantity()	1	testing when user checks out all item	item Quantity1 = 10 cart Quantity1 = 4 item Quantity2 = 15 cart Quantity2 = 10	item Quantity1 = 6 item Quantity2 = 5	item Quantity1 = 6 item Quantity2 = 5	P
	2	testing when user checks out specific item	item Quantity3 = 20 cart Quantity3 = 1	item Quantity3 = 19	item Quantity3 = 19	p
	3	testing when user checks out specific seller	item Quantity4 = 2 cart Quantity4 = 1	item Quantity4 = 1	item Quantity4 = 1	p
checkItems()	1	testing when product ID does not exist	-	returns 0	returns 0	P
	2	testing when the quantity is greater than the stock	-	returns 0	returns 0	P
	3	testing when the product ID exist	-	returns 1	returns 1	P
	4	testing when the quantity is less than the stock	-	returns 1	returns 1	P
writeToFileUsers()	1	write the contents of array to the user file	-	writes the contents to the file	writes the contents to the file	
writeToFileItems()	1	write the contents of array to the items file	-	writes the contents to the file	writes the contents to the file	
readItemsFile()	1	reads the content of items file	-	stores the contents in the items array	stores the contents in the items array	p
readCartFile()	1	reads the content of cart file	-	stores the contents in the items array	stores the contents in the items array	P
writingCartFile()	1	write the contents of array to the user's cart file	-	writes the contents to the file	writes the contents to the file	p

writeTransactionFile()	1	write the contents of array to transaction file when checking out specific seller	-	writes the contents to the file	writes the contents to the file	p
	2	write the contents of array to transaction file when checking out specific item	-	writes the contents to the file	writes the contents to the file	p
writeTransactionFile1()	1	write the contents of array to transaction file when checking out all items	-	writes the contents to the file	writes the contents to the file	p
readTransactionFile()	1	reads a single transaction	-	reads the content of transaction file	reads the content of transaction file	p
	2	reads multiple transactions				
	3	reads an empty file				
checkDateRange()	1	the given date is before the start date	given date: 01/15/2023 start date: 01/16/2023 end date: 01/20/2023	returns 0	returns 0	P
	2	the given date is within the start and end date	given date: 01/17/2023 start date: 01/16/2023 end date: 01/20/2023	returns 1	returns 1	P
	3	the given date exceeds the end date	given date: 01/21/2023 start date: 01/16/2023 end date: 01/20/2023	returns 0	returns 0	P
isValidDate()	1	testing when the month is greater than the range	13	it will ask the user for input again	ask the user for input	P
	2	testing when the day is greater than the range	32	it will ask the user for input again	ask the user for input	P
	3	given date is within the range	month: 12 day: 31	returns 1	returns 1	P
getDate()	1	Test with a valid date	month = 3, day = 15, year = 2023	function returns without errors	function returns without errors	P
	2	Test with an invalid month (month > 12)	month = 13, day = 3, year = 2023	Error message and prompt to enter date again	Error message and prompt to enter date again	P
	3	Test with an invalid day (day > 31)	month = 5, day = 32, year = 2023	Error message and prompt to enter date again	Error message and prompt to enter date again	P
changeQuantityCO()	1	Updates product quantity for given product ID	i[1][1].productID = 123, c[0].ItemsCart[0].productID = 123, c[0].ItemsCart[0].availQuantity = 2, num[1] = 2, tempVar = 123, choice = 1	i[1][1].availQuantity = 8	i[1][1].availQuantity = 8	P
	2	Updates product quantity for given seller ID	i[0][0].sellerID = 100, c[0].ItemsCart[0].sellerID = 100, c[0].ItemsCart[0].productID = 234, c[0].ItemsCart[0].availQuantity = 3, num[0] = 1, tempVar = 100, choice = 2	i[0][0].availQuantity = 2	i[0][0].availQuantity = 2	P
	3	Updates product quantity for all products in cart	i[2][2].productID = 456, i[2][2].availQuantity = 10, c[0].ItemsCart[0].productID = 456, c[0].ItemsCart[0].availQuantity = 1, c[0].ItemsCart[1].productID = 789, c[0].ItemsCart[1].availQuantity = 2, num[2] = 3, tempVar = -1, choice = 3	i[2][2].availQuantity = 9	i[2][2].availQuantity = 9	P
checkAvailability()	1	testing when there's change in price	-	displays the new and old values of price	displays the new and old values of price	P

	2	testing when there's change in quantity	-	displays the new and old values of quantity	displays the new and old values of quantity	P
	3	testing when there's no change in price and quantity	-	displays nothing	displays nothing	P
isValidQuantity()	1	when the quantity is negative	i->quantity = -1	asks the user for input again	asks the user for input again	P
	2	when quantity is positive	i->quantity = 10	i->quantity = 10	i->quantity = 10	P
isValidPrice()	1	when unit price is negative	i->unitPrice = -100	asks the user for input	asks the user for input	P
	2	when unit price is positive	i->unitPrice = 10000	i->unitPrice = 10000	i->unitPrice = 10000	P