**FUNCTION SPECS**

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **DESCRIPTION** | **INPUT PARAMETER** | **RETURN DATA** |
| removeEnter | Removes \n from a string | word[] – word where enter is removed from | word[] - new word without \n |
| checkID | Checks if ID is in the array | ID – ID to be found  array[] – array where the ID is to be found  userCount – number of users registered in the program | marker – returns 1 if the ID is in the array. Returns 0 if the ID wasn’t found. |
| checkitemID | Checks if ID is in the item array and returns the index | ID – ID to be found  array[] – item array where the ID is to be found  itemCount – number of items registered in the program | ctr – returns the index of the ID if found in the array. Returns -1 if not found |
| registerUser | Allows the user to register his/her information in the program | aUser[] – user array where the information will be stored  userCounter – number of users registered in the program | \*User - information of the user |
| findID | Finds the ID of the user in the user array and returns the index | ID – ID to found  aUser[] – array where the ID is to be found.  userCnt – number of users registered in the program | index – index of the ID |
| checkPassword | Checks if the password inputted by the user is correct | index – location of the user information in the user array  password[] – password to be checked if correct  aUser[] – array where the information of all the users is stored | 0 if the password is correct  1 if the password is wrong |
| addItem | Allows the user to add items | aItems[] – array where all the item information is stored  ID – ID of the current user | \*item – item information  \*itemCount – updated item count |
| swap | swaps the location of two items | \*item1 – information of item1  \*item2 – information of item2 |  |
| sortByID | sorts the item array based on their product ID | items[] – array of items where all the item information is stored  numItems – number of items stored in the program |  |
| removeItem | removes all the information of an item | \*item – current information stored in the item | \*item – “empty” information of the item |
| display | displays all the items of the user | aItem[] – array where all the item information is stored  ID – the ID of the user  itemCount – number of items stored in the program |  |
| editStock | allows the user to edit the information of his/her item | \*item – current information stored in the item | \*item – updated information |
| SellMenu | prints the sell menu | aItems[] – array where all the information of the items is stored  ID – ID of the user  \*itemCount – number of items stored in the program | \*itemCount – updated number of items |
| swapUser | swaps the information of two users | \*user1 – the information of the first user  \*user2 – the information of the second user |  |
| sortuserID | sorts the user array based on the user ID from lowest to highest | aUser[] – array where all the user information is stored  userCount – number of users registered in the program |  |
| printReceipt | prints the receipt of a transaction | transCtr – number of items in the transaction  transaction – the transaction |  |
| removeItemID | removes all the information of the items with the same seller IDs | item[] – array where all the item information is stored  ID – the ID of the seller  cartCounter – number of items in the cart |  |
| checkout | prints the checkout menu and allows the user to check out his/her items | cart – information stored in the cart  cartCounter – number of items stored in the cart  aItems[] – array where all the item information is stored  userID – ID of the user |  |
| sortCart | sorts the cart based on the sellerID from lowest to highest | aItems[] – array where all the item information is stored  cartCount – number of items stored in the cart |  |
| BuyMenu | prints the buy menu and allows the user to buy items | userCount – number of users registered in the program  aUser[] – array where all the user information is stored  aItems[] – array where all the item information is stored  itemCount – number of items registered in the program  userID – ID of the user |  |
| UserMenu | prints the user menu | aUser[] – array where all the user information is stored  userCnt – number of users registered in the program  aItems[] – array where all the item information is stored  \*itemCount – number of items stored in the program |  |
| saveItems | saves all the items in the item array in a text file | aItems[] array where all the item information is stored  itemCount – number of items stored in the program |  |
| loadItems | loads all the items stored in Items.txt | aItems[] – array of items where all the item information is stored | aItems[] – updated array with all the items from Items.txt |
| saveUsers | saves all the user information in Users.txt | aUser[] – array of users where all the user information is stored  userCnt – number of users registered in the program |  |
| loadUsers | loads all the user information stored in Users.txt | aUser[] – array of users where all the user information is stored | aUsers[] – updated array with the items from Users.txt |
| loadTransaction | loads all the transactions from Transactions.dat | aTransaction[] – array of transactions where all the transactions are stored | aTransaction[] – updated array with the transactions from Transactions.dat |
| findsellerID | checks if the seller is already in the seller array | seller[] – array of sellers where all the sellers are stored  sellercount – number of sellers stored in the seller array  ID – ID to be checked if inside the seller array | ctr – if the ID is found  -1 – if the ID wasn’t found |
| adminmenu | prints the admin menu | aUser[] – array where all the user information is stored  aItems[] – array where all the item information is stored  userCnt – number of users registered in the program  itemCount – number of items registered in the program |  |
|  |  |  |  |

**TEST CASES**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **FUNCTION** | **#** | **DESCRIPTION** | **SAMPLE INPUT DATA** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** | **P/F** |
| sortByID | 1 | Integers in array have not been sorted yet | 1,2,3,4,5,6,7,8,9,10 | 1,2,3,4,5,6,7,8,9,10 | 1,2,3,4,5,6,7,8,9,10 | P |
|  | 2 | Integers in array are in decreasing order | 10,9,8,7,6,5,4,3,2,1 | 1,2,3,4,5,6,7,8,9,10 | 1,2,3,4,5,6,7,8,9,10 | P |
|  | 3 | Integers in array contain positive and negative values | -10,9,-8,7,-6,5,-4,3,-2,1 | -10,-8,-6,-4,-2,1,3,5,7,9 | -10,-8,-6,-4,-2,1,3,5,7,9 | P |
| findID checkitemID  sortCart  findsellerID | 1 | ID is not in the array | 1,2,3,4,5,6,7,8,9,10 ID = 22 | -1 | -1 | P |
|  | 2 | ID is in the array | 1,2,3,4,5,6,7,8,9,10  ID = 5 | 4 | 4 | P |
|  | 3 | ID is a negative value | 1,2,3,4,5,6,7,8,9,10  ID = -23 | -1 | -1 | P |
| checkID | 1 | ID is not in the array | 1,2,3,4,5,6,7,8,9,10 ID = 22 | -1 | -1 | P |
|  | 2 | ID is in the array | 1,2,3,4,5,6,7,8,9,10  ID = 5 | 1 | 1 | P |
|  | 3 | ID is a negative value | 1,2,3,4,5,6,7,8,9,10  ID = -23 | -1 | -1 | P |
| checkPassword | 1 | Password is correct | Password = Machine  Input = Machine | 0 | 0 | P |
|  | 2 | Password is incorrect | Password = Machine  Input = Robot | 1 | 1 | P |
|  | 3 | Password is correct but wrong capitalization of letters | Password = Machine  Input = MACHINE | 1 | 1 | P |
| editStock | 1 | User wants to replenish his/her stock | Current Stock: 10 Quantity to be added: 2 | Stock = 12 | Stock = 12 | P |
|  | 2 | User wants to change the price of his product | Current Price: 20.00 Changed Price: 30.00 | Price = 30.00 | Price = 30.00 | P |
|  | 3 | User wants to change the name of his product | Current Name: Hotdog  Changed Name: Itlog | Name: Itlog | Name: Itlog | P |
|  | 4 | User wants to change the description of his product | Current Description: Tender Juicy Changed Description: Nilaga | Description: NIlaga | Description: NIlaga | P |
|  | 5 | User wants to change the category of his product | Current Category: Processed Food Changed Category: Chicken | Category: Chicken | Category: Chicken | P |
| saveItems | 1 | No items to be saved |  | Contents of Items.txt:  None | Contents of Items.txt:  None | P |
|  | 2 | There are items to be saved | Item1, Item2, Item3 | Contents of Items.txt:  Item1, Item2, Item3 | Contents of Items.txt:  Item1, Item2, Item3 | P |
|  | 3 | Items.txt has items inside | Contents of Items.txt:  Item4, Item5  Input:  Item1, Item2, Item3 | Contents of Items.txt:  Item1, Item2, Item3 | Contents of Items.txt:  Item1, Item2, Item3 | P |
| loadItems | 1 | Items.txt has no content |  | Item Array: Empty | Item Array: Empty | P |
|  | 2 | Items.txt does not exist |  | Item Array: Empty | Item Array: Empty | P |
|  | 3 | Items.txt has content | item1, item2, item3 | Item Array:  item1, item2, item3 | Item Array:  item1, item2, item3 | P |
| saveUsers | 1 | No user info to be saved |  | Contents of Users.txt:  None | Contents of Users.txt:  None | P |
|  | 2 | There are user info that needs to be saved | user1, user2, user3 | Contents of Users.txt:  user1, user2, user3 | Contents of Users.txt:  user1, user2, user3 | P |
|  | 3 | User.txt does not exist | user1, user2, user3 | Users.txt gets created  Contents of Users.txt:  user1, user2, user3 | Users.txt gets created  Contents of Users.txt:  user1, user2, user3 | P |
| loadUsers | 1 | Users.txt has no content |  | User Array: Empty | User Array: Empty | P |
|  | 2 | Users.txt does not exist |  | User Array: Empty | User Array: Empty | P |
|  | 3 | Users.txt has content | user1. user2, user3 | User Array:  user1, user2, user3 | User Array:  user1, user2, user3 | P |
| loadTransaction | 1 | Transactions.dat has no content |  | Transaction Array:  Empty | Transaction Array:  Empty | P |
|  | 2 | Transactions.dat does not exist |  | Transaction Array:  Empty | Transaction Array:  Empty | P |
|  | 3 | Transactions.dat has content | trans1, trans2, trans3 | Transaction Array:  trans1, trans2, trans3 | Transaction Array:  trans1, trans2, trans3 | P |