MANZANO, Ninna Robyn M. March 29, 2018

S16-B Prof. Nathalie Lim-Cheng

**Function Specifications**

|  |  |  |  |
| --- | --- | --- | --- |
| Function  Name | Description | Input Parameter | Return Data |
| loadUsers | reads and loads all user information from a file "Users.txt" | aUsers – array of users  \*pUsers - pointer counter for loaded existing users | \*pUsers – updated value of number of users |
| saveUsers | saves all added user information to a file "Users.txt" | aUsers – array of users  nUsers – number of existing users | updated directory of users with the newly added users |
| loadItems | reads and loads all item information from a file "Items.txt" | aItems – array of items  \*pItems – pointer counter for loaded existing items | \*pItems – updated value of number of items |
| saveItems | saves all added item information in a file "Items.txt" | aItems – array of items  nItems – number of existing items | updated directory of items with the newly added items |
| checkUser | checks if userID is unique by searching inputted user ID in the array of user IDs | aUsers – array of users  tempID – user inputted user ID  \*pUsers - pointer counter for existing users | return 1 – user ID is unique  return 0 – an existing user has the same user ID |
| registerUser | permits the user to register if the checked inputted user ID is unique then stores inputted user information to the array of Users. | aUsers – array of users  \*pUsers - pointer counter for existing users | \*pUsers – updated value of number of users |
| userLogin | checks if inputted userID and password matches any registered users then gets matched user ID and its index for accessing in the latter functions. If not, the user is redirected back to the main menu. | aUsers – array of users  \*pUsers - pointer counter for existing users  \*userID – pointer for acquired user ID | return 0 – no existing user  return 1 – there is an existing user |
| checkProductID | checks if product ID is unique (regardless of the seller) by searching inputted product ID in the array of Items | aItems – array of items  \*pItems – pointer counter for existing items  tempProdID – product ID to be checked | return 0 – not unique product ID  return 1 – unique product ID |
| findProductID | searches for the inputted product ID in the existing array of product IDs | aItems – array of items  \*pItems – pointer counter for existing items  tempProdID – product ID to be searched for  \*prodIndex – index of matched product ID | return 0 – no matching product ID  return 1 – there is a matching product ID  and gets its index now stored in \*prodIndex |
| sortProductID | sorts product information in increasing order according to product ID. | aItems – array of items  \*pItems – pointer counter for existing items | sorted array of items in increasing order based on product ID |
| checkSellerItemLimit | checks if the seller has reached the maximum 20 number of items | aItems – array of items  \*pItems – pointer counter for existing items  sellID – seller ID of the user | return 1 – seller exceeded the item limit  return 0 – seller did not exceed the item limit |
| addNewItem | lets the user add a new item to sell. Information for the item (except the seller ID) is asked from the user namely the product ID, item name, category, item description, and price | aItems – array of items  \*pItems – pointer counter for existing items  sellerID – seller ID of the user | \*pItems – updated value of number of items |
| showMyProducts | sorts array of Items in increasing order based on the productID. It then shows all information of the products in the following table format:  productID, item name, category, unit price, quantity | aItems – array of items  \*pItems – pointer counter for existing items  \*sellerID – pointer for seller ID of the user | N/A |
| editStock | shows all of seller's products in table format by calling showMyProducts(). Then if productID is valid, asks for product ID whose information is to be edited and offers the submenus (Replenish, Change Price...) until the user chooses to finish editing. If not, it redirects user back to Sell Menu | aItems – array of items  \*pItems – pointer counter for existing items  \*sellerID – pointer for seller ID of the user | updated product information depending on chosen submenus |
| showLowStocks | shows all the information (including product description but excluding seller's userID) about each product whose quantity is below 5. It displays information one at a time per product. wherein it allows the user to press N to see the next product with low stocks and press X to exit the view. | aItems – array of items  \*pItems – pointer counter for existing items | N/A |
| sortSellerID | sorts array of items in increasing order based on seller ID | aItems – array of items  nItems – counter for existing items | sorted array of items in increasing order based on seller ID |
| viewAllProducts | displays all items from 1 seller at a time, sorted in increasing order based on seller ID. Seller ID is shown first, then followed by a table of the products of that seller. When the user presses N, the next seller ID is shown and under it, the table of products of that seller, and so on. The user can press X to exit this view and go back to the Buy Menu. | aUsers – array of users  nUsers – number of existing users  aItems – array of items  nItems – counter for existing items | N/A |
| showProductsBySeller | searches if inputted seller ID matches with an existing sellerID. Then, the program displays the seller’s ID and below it the table of that seller’s products | aItems – array of items  \*pItems – pointer counter for existing items  sellerID – seller ID of the user | N/A |
| searchProductsByCategory | asks the user to input the category to search for. It then displays the number of items found in the search then displays those products matching the given category in table format similar to Show Low Stocks. It also allows the user type N and X to navigate through the display. | aItems – array of items  \*pItems – pointer counter for existing items | N/A |
| searchProductsByName | asks the user to input keywords of the product (substring) to search for. It then  displays number of items found in the search followed by a table format that is similar with Show Low Stocks. It also allows the user type N and X to navigate through the display. | aItems – array of items  \*pItems – pointer counter for existing items | N/A |
| loadPreviousCart | This function loads previous cart information from a binary file "<user’s ID>.bag"  if there is a previous cart. If not, it creates a new cart by writing into a new binary file. | aCart – array of cart  \*pCart – pointer counter for items in cart  \*userID – pointer for user ID | \*pCart - updated value of number of loaded items in the cart |
| saveCart | saves remaining item information to a binary file of Items with the filename <user's ID>.bag | aCart – array of cart  \*pCart – pointer counter for items in cart  \*userID – pointer for user ID | remaining item information in the cart |
| displayCart | displays all product information contained in Cart | aCart – array of cart  \*pCart – pointer counter for items in cart | displays all product information contained in Cart |
| checkBuyer | finds the product ID of the item that the buyer wishes to buy then checks if the currently logged-in buyer is in the directory of seller IDs in the stored array of Items | aItems – array of items  \*pItems – pointer counter for existing items  userID – userID to be checked  product | return 0 – the buyer is a seller  return 1 – the buyer is not a seller |
| addToCart | asks for the product ID and first checks if the buyer is trying to buy his own product. It then searches  for the product in the array of Items. When found, it checks if the item quantity is enough for the user input quantity. If the cart is full already, no additional items can be added to the cart suggests that the user proceed to Edit Cart or Check Out first before adding more items. | aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  userID – ID of the user | Updated cart array |
| editCart | displays all items in the Cart then provides the submenus (Remove all items from Seller, Remove Specific Item...) for editing | aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  userID – ID of the user | \*pCart - updated value of number of loaded items in the cart |
| checkIfAvailable | checks if items to be checked out are available and if seller has updated the information (..quantity and price). If there is any change, the buyer is notified by displaying the old and the new quantity and price; changes are also updated in the cart. The buyer is given notice (message) that he can still go to Edit Cart | aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart | Outputs corresponding display notifications for the updates |
| saveTransac | saves the transactions made by reading the end of "Transactions.dat" then writing into the file | transact – structure for Transaction data | Saves all transactions made in “Transactions.dat” |
| checkOutAll | removes all items bought by the user and updates the cart. Each transaction contains all items from the same seller only then a display of summary of each transaction should be displayed on the screen following the table format: quantity, product ID, item name, unit price, total price for item. Then below the table, a total amount due for the transaction and payable to sellerID and seller name. Only then will  transaction information added to the binary file "Transactions.dat". | aUsers – array of users  \*pUsers - pointer counter for existing users  aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  transact – structure for Transaction data  buyerID – user ID of the currently logged-in user | N/A |
| checkOutBySpecSeller | removes items bought from the cart based on inputted seller ID | aUsers – array of users  \*pUsers - pointer counter for existing users  aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  transact – structure for Transaction data  sellerID – user inputted seller ID to search for the items to be bought in Cart  buyerID – user ID of the currently logged-in user | N/A |
| checkOutBySpecItem | checks out a specific item inputted by the user then displays the summary of transactions. Then it will update the cart. | aUsers – array of users  \*pUsers - pointer counter for existing users  aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  transact – structure for Transaction data  productID – user inputted product ID to search for the items to be bought in Cart  buyerID – user ID of the currently logged-in user | N/A |
| checkOutMenu | gets input date from the user then displays the options for checkout. | aUsers – array of users  \*pUsers - pointer counter for existing users  aItems – array of items  \*pItems – pointer counter for existing items  aCart – array of cart  \*pCart – pointer counter for items in cart  transact – structure for Transaction data  buyerID – user ID of the currently logged-in user | N/A |
| adminLog | checks if the inputted password matches with the required admin password before being able to enter the Admin Menu | placePass – the inputted password by the user | return 1 – inputted password matches with the admin password  return 0 – does not match with the admin password |
| sortUserID | sorts user information in increasing order according to user ID | aUsers – array of users  nUsers – number of existing users | sorted array of users in increasing order based on user ID |
| showAllUsers | shows all users arranged by their user ID in a table following the format: userID, password, name, address, phone number | (arrUsers \*pU, int \*pCount) | N/A |
| showAllSellers | shows all the users who have items to be sold (i.e., if there exists in the Items array a  seller ID matching the user’s ID). The display should be in table format in the following sequence: userID, password, name, address, phone number, number of items for sale [not quantity]. | aUsers – array of users  \*pUsers - pointer counter for existing users  aItems – array of items  \*pItems – pointer counter for loaded existing items | N/A |
| checkTransacDate | asks the user to input 2 dates to serve as the start and end dates. It then checks the contents of Transactions.dat whose dates fall within the duration from start to the end dates, then returns 1 if it is. | transact – structure for Transaction data | Return 1 – date falls in the start and end dates  Return 0 – date does not fall in the start and end dates |
| showTotalSales | displays total amount of all the transactions if input date falls within the start and end dates. | transact – structure for Transaction data | N/A |
| showSellerSales | displays the total sales for each seller in table format in the following sequence: seller ID, seller name, total sales in the duration if the input date falls within the start and end dates. | transact – structure for Transaction data  aUsers – array of users  \*pUsers - pointer counter for existing users | N/A |
| showShopaholics | displays the total amount for each buyer in table format in the following sequence: buyer ID, buyer name, total amount bought in the duration if the input date falls within the start and end dates. | transact – structure for Transaction data  aUsers – array of users  \*pUsers - pointer counter for existing users | N/A |

**TEST SCRIPT**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Function** | **#** | **Description** | **Sample Input Data** | **Expected Output** | **Actual Output** | **P/F** |
| loadUsers  \*\*test for max. 20 users | 1 | The file “Users.txt” has no contents. | User.txt contents: N/A | \*pUsers = 0 | \*pUsers = 0 | P |
| 2 | The file “Users.txt” has 3 sets user information. | User.txt contents:  1121 hi  Miriam Santiago  2030 Pineview Drive  9237895646  <new line>  …  <user3 id> | \*pUsers = 3 | \*pUsers = 3 | P |
| 3 | The file “Users.txt” to be read is non-existent. | User.txt contents: N/A | “File is not available for reading.” | “File is not available for reading.” | P |
| saveUsers | 1 | There is a “Users.txt” file in the directory | “Users.txt” file in the directory | User information is saved in “Users.txt” | User information is saved in “Users.txt” | P |
| 2 | There is no “Users.txt” file in the directory | Different file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| 3 | The file to be opened has a different name in the directory. | “users.txt” file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| loadItems | 1 | The file “Items.txt” has no contents. | Items.txt contents:  N/A | \*pItems = 0 | \*pItems = 0 | P |
| 2 | The file “Items.txt” 10 sets of item information. | Items.txt contents:  10 sets of items | \*pItems = 10 | \*pItems = 10 | P |
| 3 | The file “Users.txt” to be read is non-existent. | Items.txt contents:  N/A | \*pItems = 0 | \*pItems = 0 | P |
| saveItems | 1 | There is a “Items.txt” file in the directory | “Items.txt” file in the directory | Items information  are saved in “Items.txt” | Items information  are saved in “Items.txt” | P |
| 2 | There is no “Items.txt” file in the directory | Different file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| 3 | The file to be opened has a different name in the directory. | “items.txt” file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| checkUser | 1 | User ID is unique. | User ID: 118902732 (not a part of the registered users) | returns 1 | returns 1 | P |
| 2 | User ID is not unique. | User ID: 1121  (a part of the registered users) | “UserID already exists!” then returns 0 | “UserID already exists!” then returns 0 | P |
| 3 | User ID input is negative or 0. | User ID: -12345 | “Invalid user ID input!” then returns 0 | “Invalid user ID input!” then returns 0 | P |
| registerUser | 1 | User ID is unique. | User ID: 118902732 (not a part of the registered users) | //allows the user to enter the remaining user information | //allows the user to enter the remaining user information | P |
| 2 | User ID is not unique. | User ID: 1121  (a part of the registered users) | //does not allow the user to enter the remaining user information | //does not allow the user to enter the remaining user information | P |
| 3 | User ID input is negative or 0. | User ID: -12345 | //does not allow the user to enter the remaining user information | //does not allow the user to enter the remaining user information | P |
| userLogin | 1 | User ID and password matches a registered user | User ID: 1121  Password: hi | //proceeds to enter to the User Menu options | //proceeds to enter to the User Menu options | P |
| 2 | User ID and password does not match a registered user | User ID: 10001  Password: thisisit5454  (non-existent in registered users) | “Error: Input does not match any users.” | “Error: Input does not match any users.” | P |
| 3 | User ID is negative. | User ID: -10001  Password: thisisit5454  (non-existent in registered users) | “Error: Input does not match any users.” | “Error: Input does not match any users.” | P |
| checkProductID | 1 | Product ID is not unique. | Product ID: 5555 | returns 0 | returns 0 | P |
| 2 | Product ID is unique. | Product ID: 23278 | returns 1 | returns 1 | P |
| 3 | Product ID is negative or 0. | Product ID: -3242 | returns 0 | returns 0 | P |
| findProductID | 1 | Product ID is existing in array of  product IDs. | Product ID: 5555 | returns 1 | returns 1 | P |
| 2 | Product ID is not existing in array of  product IDs. | Product ID: 23278 | returns 0 | returns 0 | P |
| 3 | Product ID input is negative or 0. | Product ID: -3242 | returns 1 | returns 1 | P |
| sortProductID | 1 | Product IDs in the array of items is already in increasing order. | aItems contains:  309 4464 5678 6345 9011 | aItems contains:  309 4464 5678 6345 9011 | aItems contains:  309 4464 5678 6345 9011 | P |
| 2 | Product IDs in the array of items is in mixed order | aItems contains:  6345 9011 5678 309 7777 4464 | aItems contains:  309 4464 5678 6345 9011 | aItems contains:  309 4464 5678 6345 9011 | P |
| 3 | Product IDs in the array of items is already in decreasing order. | aItems contains:  9011 6345 5678 4464 309 | aItems contains:  309 4464 5678 6345 9011 | aItems contains:  309 4464 5678 6345 9011 | P |
| checkSellerItemLimit | 1 | Seller has not reached maximum limit of 20 items | Counter for items under the seller is 13 | returns 0 | returns 0 | P |
| 2 | Seller has reached maximum limit of 20 items | Counter for items under the seller is 20 | returns 1 | returns 1 | P |
| 3 | Seller has 0 number of items. | Counter for items under the seller 0 | returns 1 | returns 1 | P |
| addNewItem | 1 | Seller has not reached maximum limit of 20 items | \*pItems = 7 | //lets the user input the remaining item information and adds to the array of items | //lets the user input the remaining item information and adds to the array of items | P |
| 2 | Seller has reached maximum limit of 20 items | \*pItems = 20 | //does not let the user input the remaining item information and adds to the array of items | //does not let the user input the remaining item information and adds to the array of items | P |
| 3 | Seller has 0 number of items. | \*pItems = 0 | //lets the user input the remaining item information and adds to the array of items | //lets the user input the remaining item information and adds to the array of items | P |
| editStock | 1 | Valid product ID is given | Product ID: 5678 | //Repeatedly offers the submenus | //Repeatedly offers the submenus | P |
| 2 | Invalid product ID is given. | Product ID: 1118 | “Invalid product ID.” | “Invalid product ID.” | P |
| 3 | Product ID inputs is negative or 0. | Product ID: 000 | “Invalid product ID.” | “Invalid product ID.” | P |
| sortSellerID | 1 | Seller IDs in the array of items is already in increasing order. | aItems contains:  1121 1318 1362 5555 | aItems contains:  1121 1318 1362 5555 | aItems contains:  1121 1318 1362 5555 | P |
| 2 | Seller IDs in the array of items is in mixed order | aItems contains:  1362 1318 5555 1121 | aItems contains:  1121 1318 1362 5555 | aItems contains:  1121 1318 1362 5555 | P |
| 3 | Seller IDs in the array of items is already in decreasing order. | aItems contains:  5555 1362 1318 1121 | aItems contains:  1121 1318 1362 5555 | aItems contains:  1121 1318 1362 5555 | P |
| searchProductsByCategory | 1 | Category to search for is not a category in items | Category: “medical” | Item/s not found and does not display anything. | Item/s not found and does not display anything. | P |
| 2 | Category to search for is a category in items | Category: “beverages” | Item/s found and display the items. | Item/s found and display the items. | P |
| 3 | Category to search for is mispelled | Category: “beravages” | Item/s found and display the items. | Item/s found and display the items. | P |
| searchProductsByName | 1 | Item name to search for is in items | Item: “creamer” | Item/s found and displays items with this substring | Item/s found and displays items with this substring | P |
| 2 | Item name to search for is not in items | Item name: “pencil” | Item/s not found and does not display anything. | Item/s not found and does not display anything. | P |
| 3 | Item name to search for is mispelled | Item name: “cereamer” | Item/s not found and does not display anything. | “0 items found.” | P |
| loadPreviousCart | 1 | The binary file has contents. | <user’s>.bag file contains 3 instances in cart | \*pCart = 3 | \*pCart = 3 | P |
| 2 | The binary file has no contents. | <user’s>.bag file contains no instances in cart | \*pCart = 0  “Error opening binary file for loading.” | \*pCart = 0  “Error opening binary file for loading.” | P |
| 3 | The binary file is non-existent. | <user’s>.bag is not in the directory | Makes a new binary file | Makes a new binary file | P |
| saveCart | 1 | Binary file in the directory | “Items.txt” file in the directory | Items information  are saved in “Items.txt” | Cart information  are saved in binary | P |
| 2 | Binary file in the directory | Different file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| 3 | Binary file has a different name in the directory | “items.txt” file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| checkBuyer | 1 | The buyer is not a seller. | ProductID: 1318 | Returns 1 | Returns 1 | P |
| 2 | The buyer is also a seller. | productID: 5555 | Returns 0 | Returns 0 | P |
| 3 | Product ID found and buyer is also a seller | productID: 3001 | Returns 0 | Returns 0 | P |
| addToCart | 1 | Product ID exists; sellerID and buyerID are the same | ProductID: 1232  SellerID: 5555  BuyerID: 3424 | Does not add to cart | Does not add to cart | P |
| 2 | Product ID exists; sellerID and buyerID are equal | ProductID: 3090  SellerID: 5555  BuyerID: 5555 | Adds item to cart | Adds item to cart | P |
| 3 | ProductID does not exist | ProductID: 3424  SellerID: 5555  BuyerID: 5555 | Does not add to cart | Does not add to cart | P |
| editCart | 1 | User input for option is found in the sub-options of editCart | option = 2 | Allows user to enter Edit Cart options | Allows user to enter Edit Cart options | P |
| 2 | User input for option is not found in the sub-options of editCart | option = 2 | Does not allow user to enter Edit Cart options | Does not allow user to enter Edit Cart options | P |
| 3 | User input for option is invalid char | option = a | Does not allow user to enter Edit Cart options | Does not allow user to enter Edit Cart options | P |
| checkIfAvailable | 1 | Old price and new price are the same | Old price: 50.00  New price: 50.00 | Does not notify user for changes | Does not notify user for changes | P |
| 2 | Old price and new price are not the same | Old price: 50.00  New price: 23.00 | Notifies users there are changes | Notifies users there are changes | P |
| 3 | New price is 0.00 | Old price: 50.00  New price: 0.00 | Notifies users the changes | Notifies users the changes | P |
| saveTransac | 1 | Binary file in the directory | “Transactions.dat” file in the directory | Items information  are saved in “Transactions.dat” | Transaction information  are saved in binary | P |
| 2 | Binary file in the directory | Different file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| 3 | Binary file has a different name in the directory | “Transactions.dat” file in the directory | "Error opening file for writing" | "Error opening file for writing" | P |
| checkOutAll | 1 | Product ID input exists in the Cart | Product ID: 1362 | checks out the item/s | Wrong check out | F |
| 2 | Product ID input is not in the Cart | Product ID: 309232 | Does not check out the item/s | Wrong check out | F |
| 3 | Product ID input has a character value | Product ID: 342a | Does not check out the item/s | Wrong check out | F |
| checkOutBySpecSeller | 1 | Product ID input exists in the Cart | Product ID: 1362 | checks out the item/s | Wrong check out | F |
| 2 | Product ID input is not in the Cart | Product ID: 309232 | Does not check out the item/s | Wrong check out | F |
| 3 | Product ID input has a character value | Product ID: 342a | Does not check out the item/s | Wrong check out | F |
| checkOutBySpecItem | 1 | Product ID input exists in the Cart | Product ID: 1362 | checks out the item/s | checks out the item/s | P |
| 2 | Product ID input is not in the Cart | Product ID: 309232 | Does not check out the item/s | Does not check out the item/s | P |
| 3 | Product ID input has a character value | Product ID: 342a | Does not check out the item/s | Does not check out the item/s | P |
| checkOutMenu | 1 | User input for option is found in the sub-options | Option = 3 | Allows user to enter checkOutMenu  options | Allows user to enter checkOutMenu  options | P |
| 2 | User input for option is not found in the sub-options | Option = 7 | Does not allow user to enter checkOutMenu  options | Does not allow user to enter checkOutMenu  options | P |
| 3 | User input for option is invalid char | Option = a | Does not allow user to enter checkOutMenu  options | Does not allow user to enter checkOutMenu  options | P |
| adminLog | 1 | Correct inputted password | Password: “H3LLo?” | Returns 1 | Returns 1 | P |
| 2 | Incorrect Inputted password | Password: “hello” | Returns 0 | Returns 0 | P |
| 3 | Inputted password is greater than the string length | Password: “helloworldandlife123” | Returns 0 | Returns 0 | P |
| sortUserID | 1 | User IDs in the array of items is already in increasing order. | aUsers contains:  1121 1318 1362 5555 | aUsers contains:  1121 1318 1362 5555 | aUsers contains:  1121 1318 1362 5555 | P |
| 2 | User IDs in the array of items is in mixed order | aUsers contains:  1362 1318 5555 1121 | aUsers contains:  1121 1318 1362 5555 | aUsers contains:  1121 1318 1362 5555 | P |
| 3 | User IDs in the array of items is already in decreasing order. | aUsers contains:  5555 1362 1318 1121 | aUsers contains:  1121 1318 1362 5555 | aUsers contains:  1121 1318 1362 5555 | P |
| checkTransacDate | 1 | Input date falls within start and end dates. | Input date: 03 29 2019  Start date: 02 26 2017  End date: 03 29 2020 | Returns 1 | Returns 1 | F |
| 2 | Input date does not fall within start and end dates. | Input date: 03 29 2021  Start date: 02 26 2017  End date: 03 29 2020 | Returns 0 | Returns 0 | F |
| 3 | Invalid input date and not in range | Input date: 04 29 2019  Start date: 02 26 2017  End date: 03 29 2020 | Returns 0 | Returns 0 | F |