PORTFOLIO

CAFÉ MANAGEMENT SYSTEM

Alex Mirrington

TABLE OF CONTENTS

DEFINING AND UNDERSTANDING

PROBLEM STATEMENT

The software solution that I am going to design and implement is a Café management system. My client is my sister, who is planning to start a small café sometime in the near future. She has asked me to make a management system to help her keep track of sales, update menu items and take orders in order to ensure the smooth running of her business. She also needs to keep track of who served which customers, so that she knows how her employees are performing in the workplace. The other main problem that she needs to address is the regular updating of the café menu each month. The program needs to be developed to fit a tablet-sized screen, so employees can carry the software solution around with them. She has looked for existing software solutions that fulfil these requirements, but many are beyond her budget range and/or don't fulfil all of her requirements. Hence she has contracted me to develop a solution for her.
The main requirements that the solution must fulfil (as defined by my sister) are:

- The café logo on each screen, with a consistent user interface.
- A login screen, with separate users for each of the employees at the café. The username of the employee that is logged in will be recorded upon the completion of an order, along with the order details. The manager will also have a separate login that will give them access to functions such as changing menus, viewing all previous sales, etc.
- Ability to create new users as new employees begin work.
- A Main Screen for navigation
- Taking orders – this includes;
    - Selection of a table
    - Selection of beverages/food
    - Review of an order (for repeating the order to the customer to confirm the order details are correct)

  Note that when the order is confirmed, the appropriate order data is saved to file for later access
- Paying for orders – This will be a separate screen that will keep track of current orders made at different tables. Note that it should only ever display the records that have NOT been paid, and are made on that day.
- Saving the details of all completed orders to file for access at a later date. Each order record will contain:
    - Unique order ID
    - Date that the order was made
    - Table Number
    - Order Subtotal
    - Whether the order is paid or not
    - Employee who served the customers
    - Additional comments from the customer
- Viewing, searching and sorting the details of past orders for calculations of tax returns, seeing how employees are performing and viewing customer feedback.
- Adding/Deleting menu items as the menu changes each month. Note that details for menu items are also stored in a database. Each menu Item record will contain:
    - Unique menu item ID
    - Name of the menu item
    - Price

The aim of this project is to create a well-designed, functional café management system for my client. The main limitation of the project is time. Hence it will be necessary to work efficiently in order to complete the project in the required time frame, to the required level of detail. Another limitation is a lack of my technical expertise. This will restrict my ability to learn new concepts within the specified time frame. In addition to this, the budget for my project is zero, and this will limit my ability to simplify the development process by using external APIs and tools. Thus, it is even more essential that I manage my time well and allow time for the development of new skills during the project.

OVERVIEW OF HARDWARE AND SOFTWARE

I will be using various software applications/online services in order to create my software solution. These include:

- Microsoft Visual Studio 2010 – for implementing all of the working code and final solution
- Microsoft Office Suite – Microsoft Word for portfolio work, Microsoft Excel for Gantt chart and Microsoft PowerPoint for GUI mockups.
- Dia – for flow charts, context diagrams, data flow diagrams, system flow charts and structure charts.
- Adobe Illustrator and Photoshop – for creating original graphics to use in my application to add to the visual appeal and overall user experience.
- Google Blogger – for keeping track of learning journal entries and logging new ideas and concepts as I come up with them

The hardware that I will have access to in order to create my application includes:

- A desktop PC running Windows 7 Professional. This computer has 8GB of RAM and has an Intel Core i5 CPU
- Along with the PC I will be using standard accessories such as a keyboard, mouse, headphones, microphone, printer, scanner, USB, etc.

DISCUSSION OF DEVELOPMENT APPROACH

The main objective of this project is to deliver a high quality café management system that fulfils all of the above client requirements. In order to make sure that the project is of the highest quality, time management and proper documentation will be essential. Hence the Structured Approach is an appropriate development process, as it will enable me to manage my time effectively and document my project well, leading to a more consistent, efficient and secure solution.

Proper time management is essential due to the limited time-frame of the project, and the importance of having a finished product to present to my client. Hence the structured approach is suitable, as it will allow me to plan my time adequately and split it amongst all tasks. I can split my time so that I will spend adequate time on defining, planning and testing, whilst minimising the time taken to implement the solution, eventually leading to a higher quality, more efficient, elegant solution that fulfils all user requirements.

The structured approach is suited to medium-large projects that require careful planning and documentation before the actual implementation of the project. This is suited to the specifications of my project, as I need to properly define and plan my solution to make sure that it fulfils all of the requirements of my client before I begin implementing it. This clear definition and planning will shorten the time spent on the implementing stage, allowing more time for testing ad maintaining, leading to a higher quality solution for my client.

The documentation required in the Structured approach also leads to a more efficient and reliable testing and maintaining phase, as less time is spent trawling through code to determine appropriate test data, and potential improvements are more easily identified and implemented. Thorough documentation also minimises the chances of errors and unforseen problems with the solution, again increasing the quality of the final solution and thus fulfilling the aim of the project.

The structured approach is also much more reliable and has a higher chance of success than other development approaches. Approaches such as Rapid Application Development (RAD) often result in failure to meet the initial requirements due to lack of documentation and time management. Although the scope of the projects developed using the RAD approach are usually smaller, they are often of lower quality due to lack of formal documentation and planning. Hence it is much better for me to use the structured approach for my project, as I am developing it for a real-world client, and I cannot afford to fall short of the client's requirements. I need to manage my time well and document the project properly according to the structured approach in order to ensure that I meet the time restrictions and project requirements.

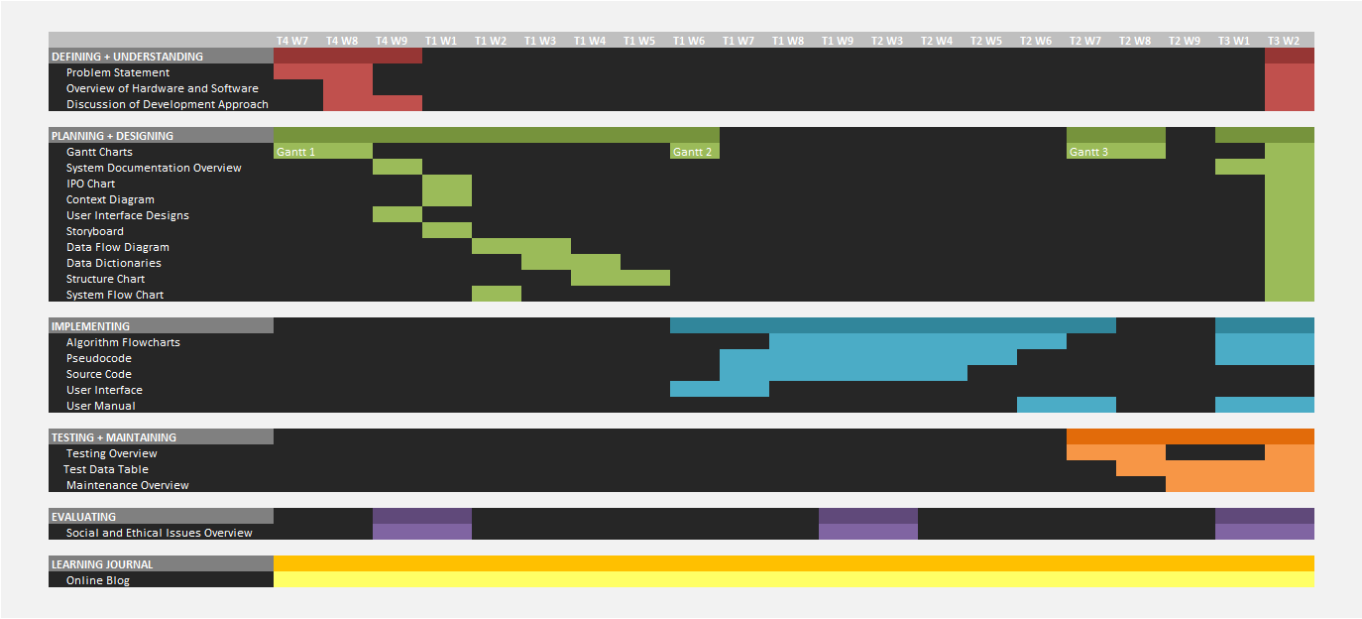
Whilst RAD has the advantage of a short development time, the structured approach will be more suitable for my project, as my aim is not to complete the solution as quickly as possible (a characteristic of the RAD approach). Instead, I aim to create a high quality solution that fulfils all requirements, as I am developing my

solution for a real world client. To do this takes time, but by using time management techniques like Gantt charts, I will be able to manager my time well and achieve my goal.

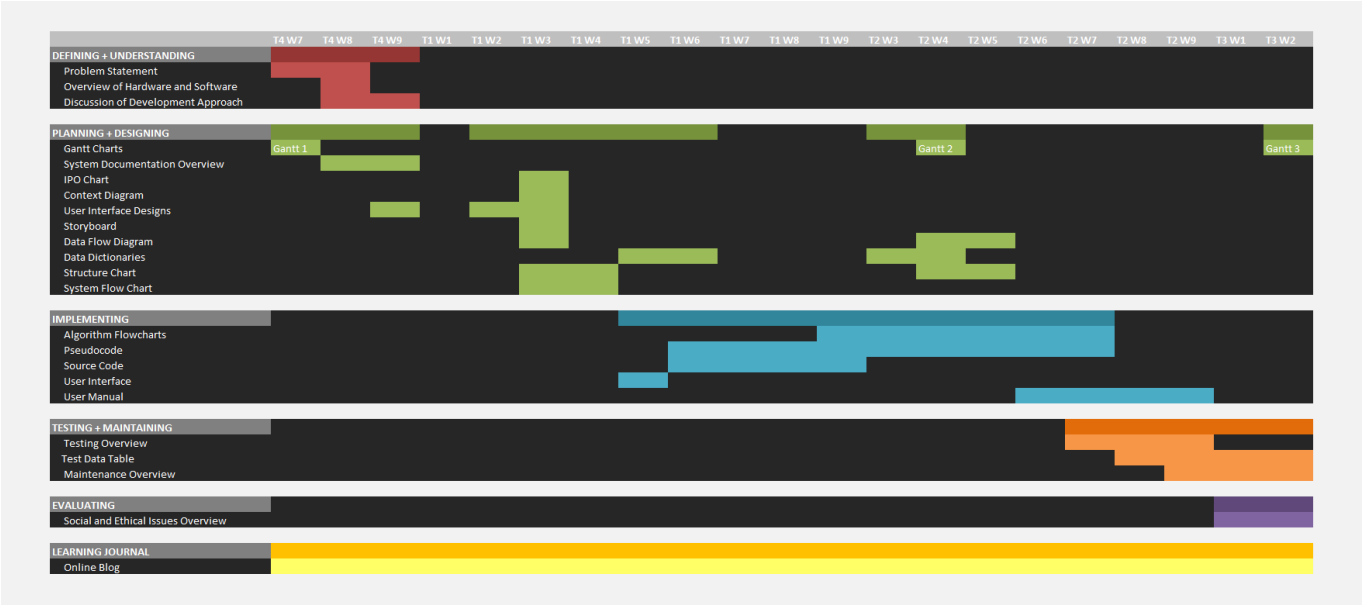
PLANNING AND DESIGNING

GANTT CHARTS

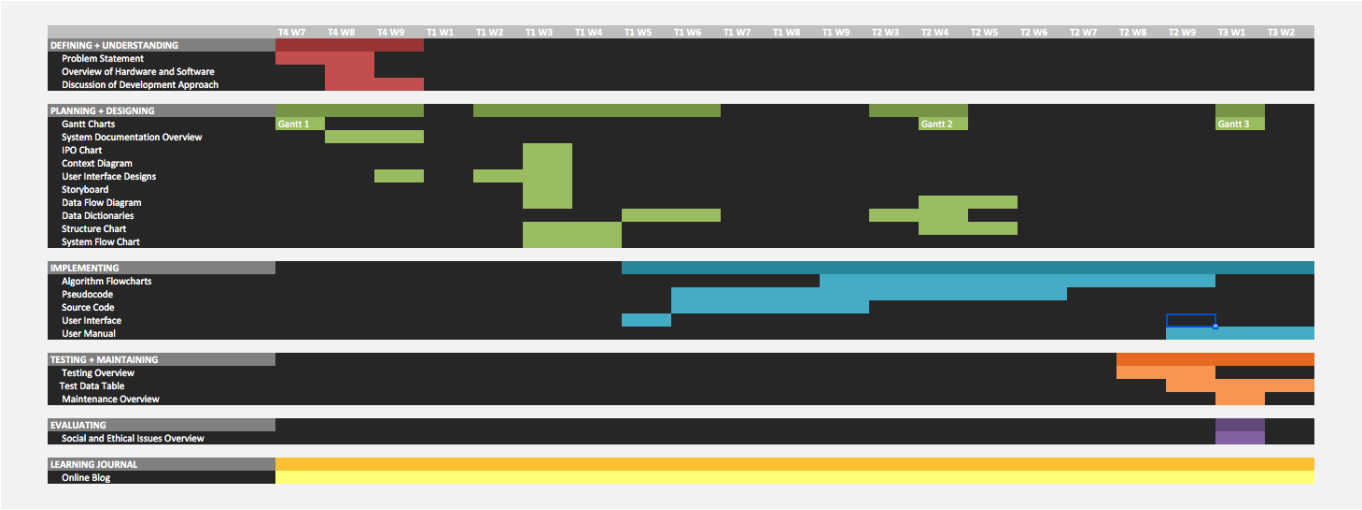
INITIAL



REVISED



FINAL



SYSTEM DOCUMENTATION OVERVIEW AND JUSTIFICATION

It is extremely important to document a software project/system during the development process. Well-planned system documentation has a variety of benefits:

- It improves the quality of the final system – By documenting your project as you complete specific aspects of it, you are reinforcing the aims and goals of the project. This is evident in the structured approach, where heavy documentation is required. In the RAD approach however, there is a lack of formal system documentation, which results in changing focus throughout the project, and ultimately a lower quality project or in some cases a failed project. Tools such the Microsoft suite are useful for defining and planning a software solution, as they allow you to record and edit your ideas as you need to, to keep track of them for a use at a later date, ultimately increasing project focus.
- It shortens the development process – By creating adequate system documentation during the planning/designing stage of a project, the implementation stage of a project can be shortened significantly. Tools such as Dia, Adobe Photoshop and the Microsoft suite are extremely useful for the production of flowcharts, GUI designs and sample algorithms respectively. By beginning the implementing stage of a project with relevant flowcharts, data flow diagrams and sample algorithms, the developer has a much clearer view of how all of the module of the system will work together, and this makes the implementation stage much easier and quicker.
- It makes the system easier to update in the future – Good system documentation allows for new members of a software development team to become familiar with a project more quickly, and allows for ease of expansion on a pre-existing software package. Instead of sifting through code to determine how it works, old and new developers alike can review the system documentation to refresh their memories on how the system functions and thus get to work on bringing out new updates more efficiently.

DESIGN TOOLS

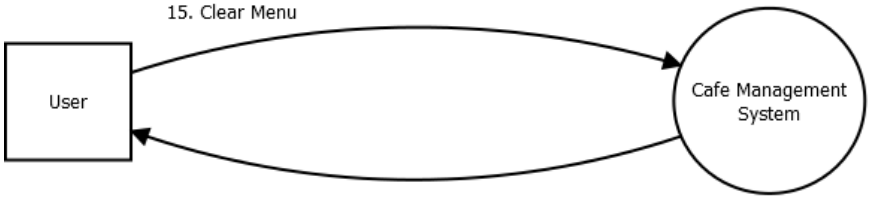
IPO CHART

INPUT	PROCESSING	OUTPUT
Username and Password	Check if the username matches any of the usernames in the database. If so, check if the password matches. If username and password are valid, login is successful.	“Username or Password is incorrect.” “Login Successful”
New user info (includes new username, password and user type e.g. admin or employee)	Check if username already exists in the database. If not, check that the selected password matches the password in the ‘re-type Password’ input box. If so, write the user data to file and give user feedback to indicate that the user has been made successfully.	“Username already exists” “Passwords don’t match” “New user created. Please return to the main menu to login.”
Table Number	Add the selected table number to the record for the current order	Selected table number changes colour to indicate that processing has occurred
Food/Beverages and amount of each	Add the food/beverage items into an array that keeps track of food/beverages for the current order.	Display the order items in a list box.
Reset Order	Remove all food/beverage items from the array	Update order items list box accordingly
Confirm Order	Calculate the order subtotal by accessing the menu database, and write a new order record to file. This order record will include an ID, date, table number, order subtotal, whether the order is paid or not, employee who created order and customer feedback. Note that the orderPaid flag will be set to false and customer feedback will be blank at this stage, as the order has not been paid yet. These parts of the record	“Order Confirmed” Display message box with order data in a user-friendly format (simulates a print-out of an order to take to the kitchen.)

	are updated upon paying for an order.	
Confirm Payment	Set the orderPaid flag to true for the selected order record, and add any customer comments to the record too.	“Order marked as Paid”
Sort by Table Number/Order total	Sort the current order records by table number or by order total (both ascending), depending on which button is pressed.	Display the sorted records in the appropriate list box.
Search Criteria	Find all order records that contain the search criteria in any one of their fields.	Display these filtered records to the user in a list box.
Sort by date. Order total or employee.	Sort all order records by date, employee or by order total, depending on which button is pressed.	Display the sorted records into the appropriate list box.
Inspect Record	Get all data from the currently selected record, and concatenate strings to put all info in a message box for feedback to user.	Display message box with concatenated string.
New menu item data (contains order name and price)	Add the new data to the menu database.	“New item added”
Edited menu Item data (contains overridden order name and price)	Edit the record that was selected to contain the new data and override the database.	“Menu item changed”
Menu item to delete	Delete the selected menu item from the menu items text file	Update the menu item list box accordingly.
Clear Menu	Delete all menu items from the menu items text file	Clear all items from the appropriate list box.

CONTEXT DIAGRAM

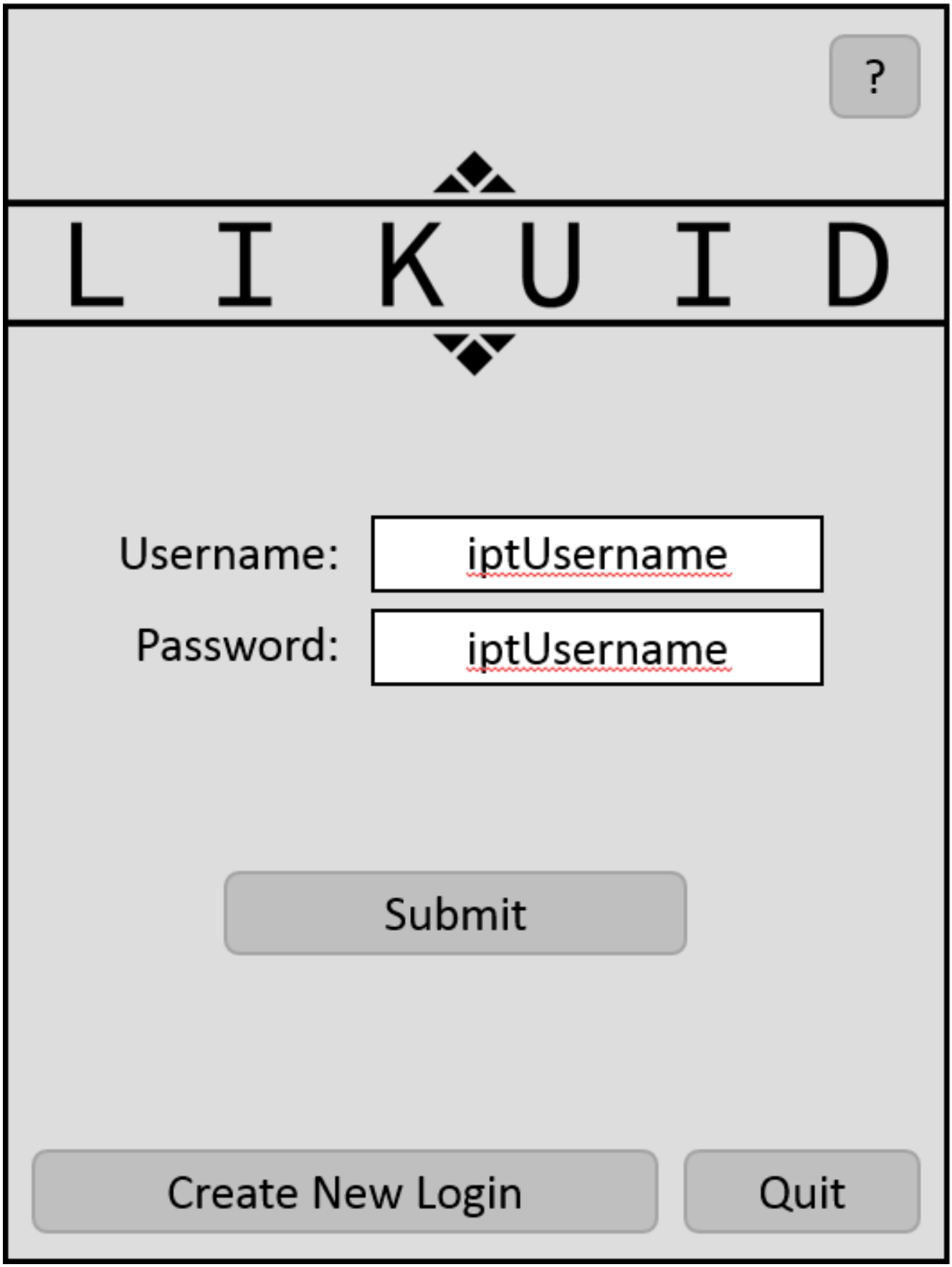
1. Username and Password
2. New user info (includes new username, password and user type e.g. admin or employee)
3. Table Number
4. Food/Beverages and amount of each
5. Reset Order
6. Confirm Order
7. Confirm Payment
8. Sort by Table Number/Order total
9. Sort by date/order total/employee
10. Search Criteria
11. Inspect Record
12. New menu item data (contains order name and price)
13. Edited menu Item data (contains overridden order name and price)
14. Menu item to delete
15. Clear Menu



1. "Username or Password is incorrect." or "Login Successful"
2. "Username already exists" or "Passwords don't match" or "New user created. Please return to the main menu to login."
3. Selected table number changes colour to indicate that processing has occurred
4. Display the order items (food/beverages) in a list box.
5. Reset order items list box accordingly
6. "Order Confirmed", Display message box with order data in a user-friendly format (simulates a print-out of an order to take to the kitchen.
7. "Order marked as Paid"
8. 9. Display the sorted records in the appropriate list box.
10. Display the filtered records to the user in a list box.
11. Display message box with concatenated string that contains record data.
12. "New item added"
13. "Menu item changed"
14. Delete the selected menu item from the list box.
15. Clear all items from the menu items list box.

USER INTERFACE DESIGNS

LOGIN



CREATE NEW LOGIN

?

L I K U I D

Username: iptUsername

Password: iptPassword

Confirm Password: iptConfirm

- (●) Manager
- ( ) Employee

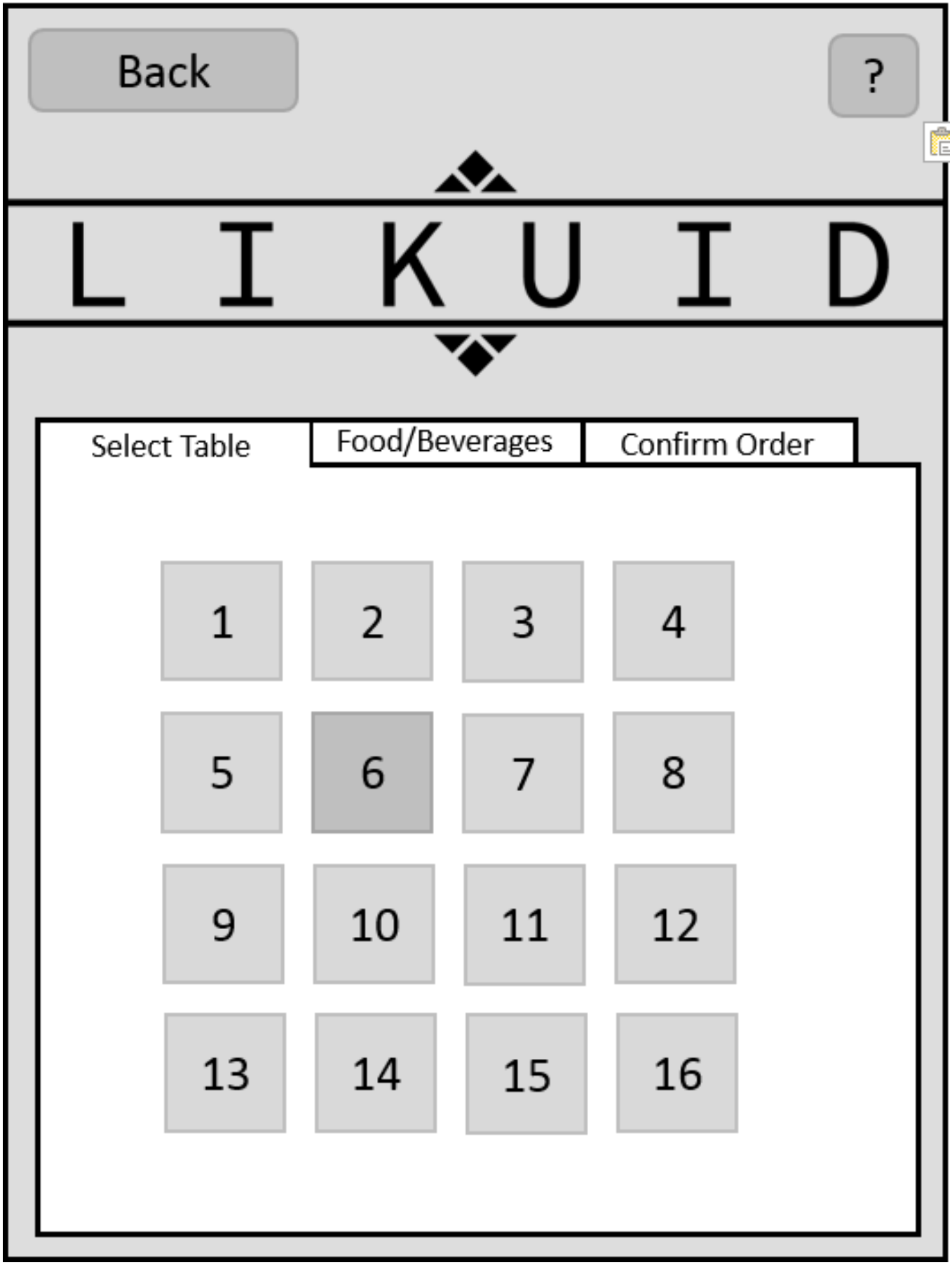
Create User

Back

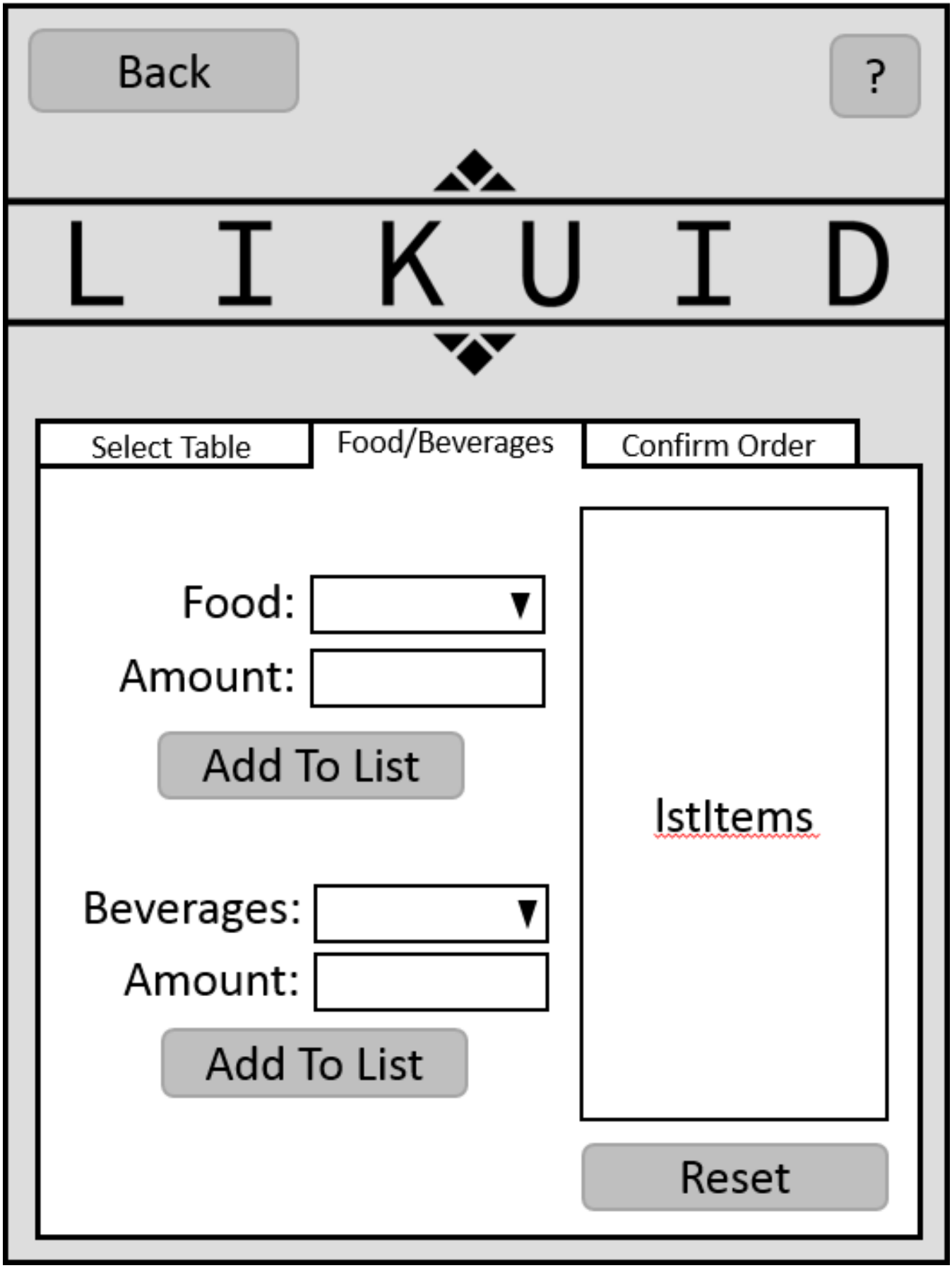
MAIN MENU


?
LIKUID
Welcome, (username)
Take Orders
Pay for Order
Change Menu
View Sales
Switch User
Quit

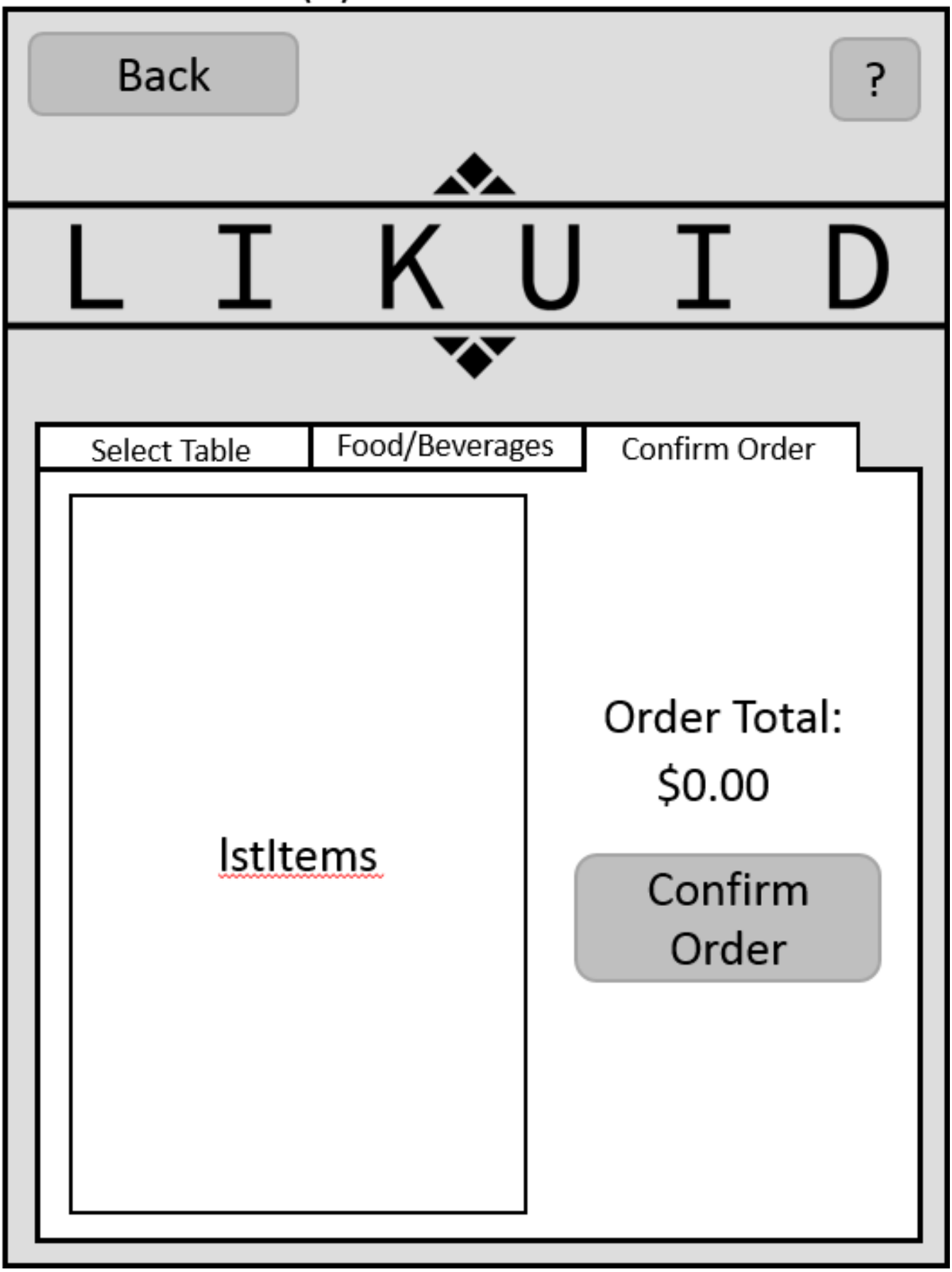
TAKE ORDERS (1)



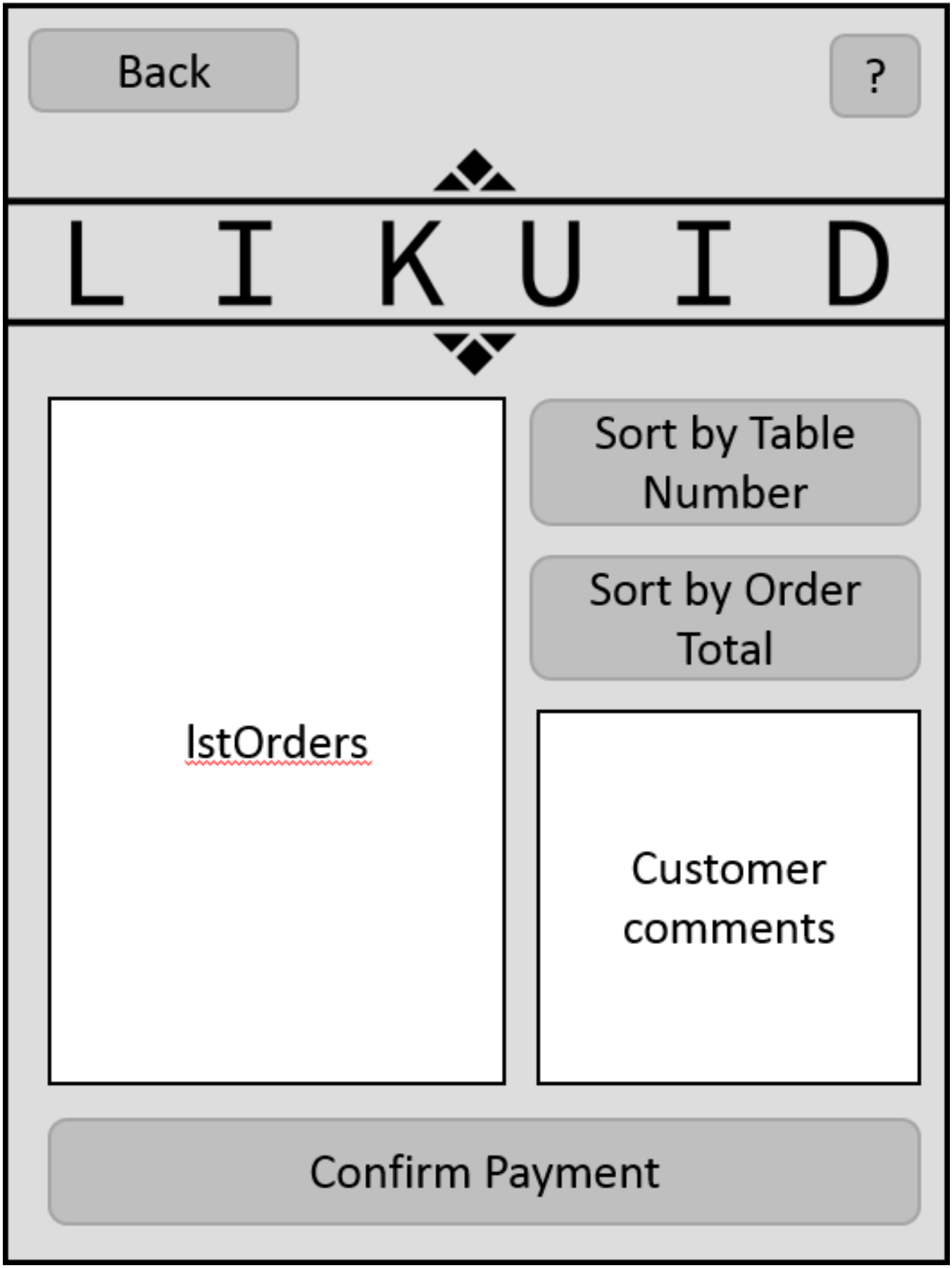
TAKE ORDERS (2)



TAKE ORDERS (3)



PAY FOR ORDERS



VIEW SALES

Back

?

L I K U I D

Search: iptSearch

From: 9/2/16 To: 16/2/16

lstSales

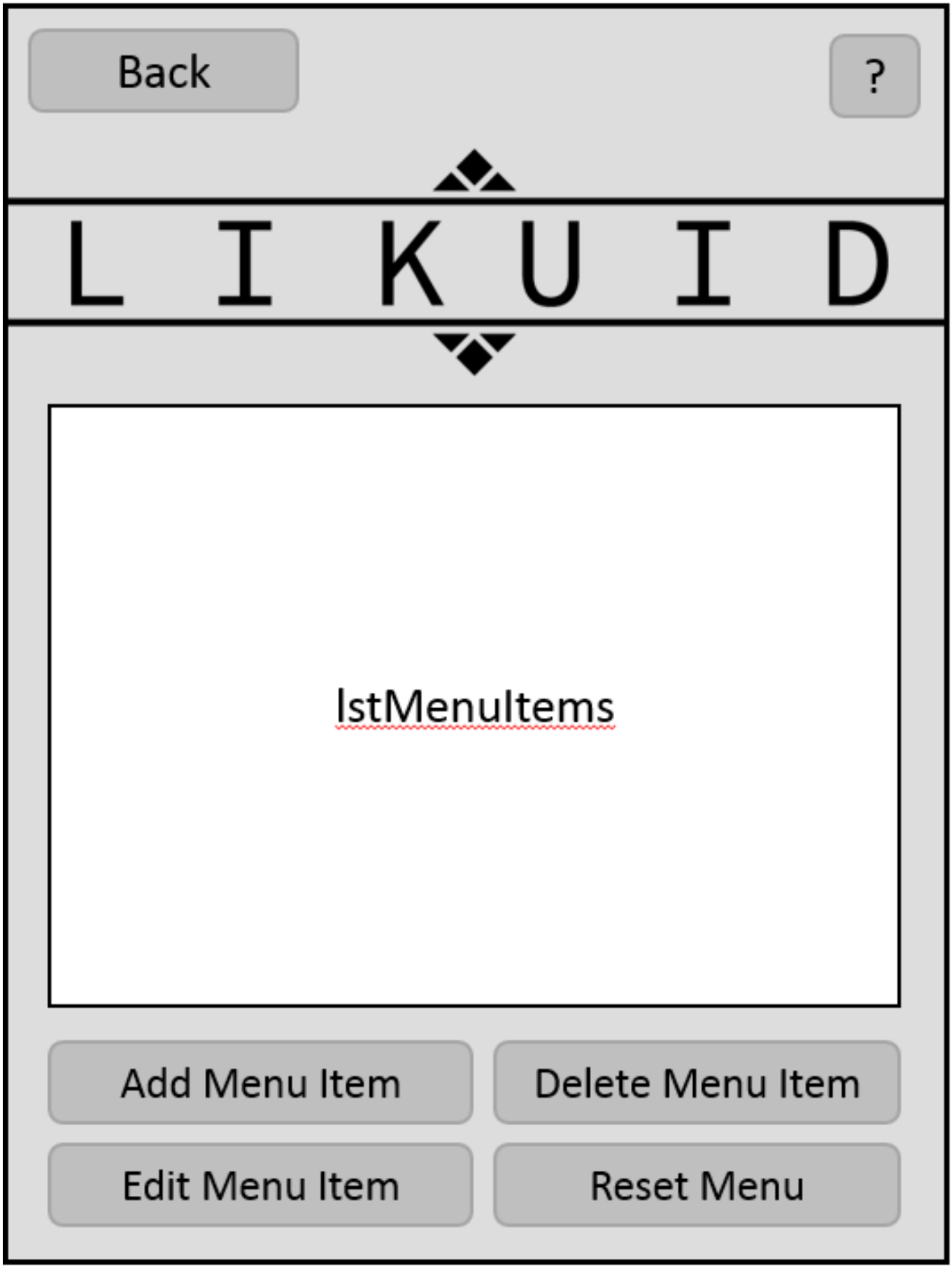
Sort by Date

Sort by Order Total

Sort by Employee

Inspect Record

EDIT MENU


Back
?
LIKUID
lstMenuItems
Add Menu Item
Delete Menu Item
Edit Menu Item
Reset Menu

ADD MENU ITEM

Back

?

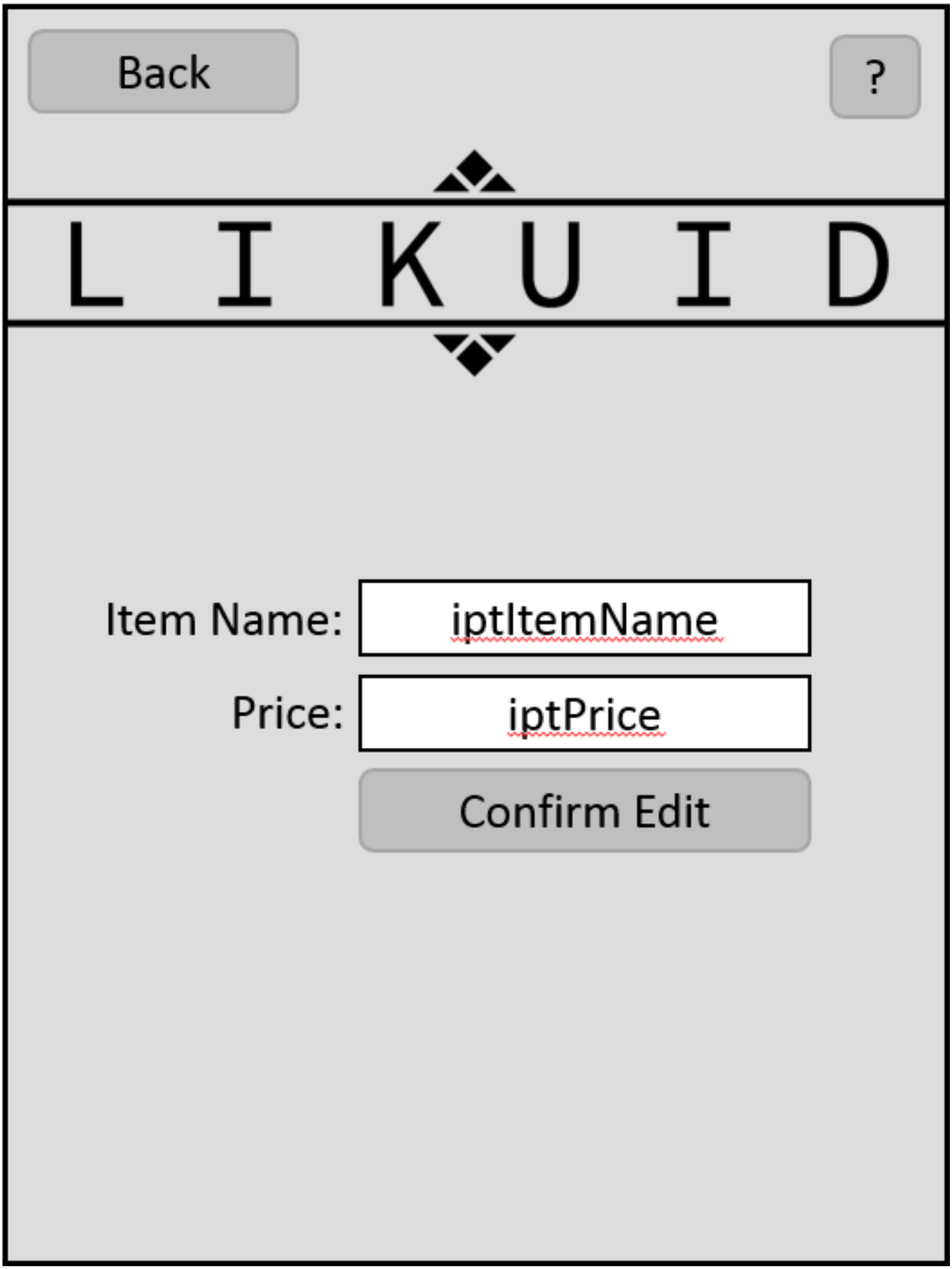
L I K U I D

Item Name: iptItemName

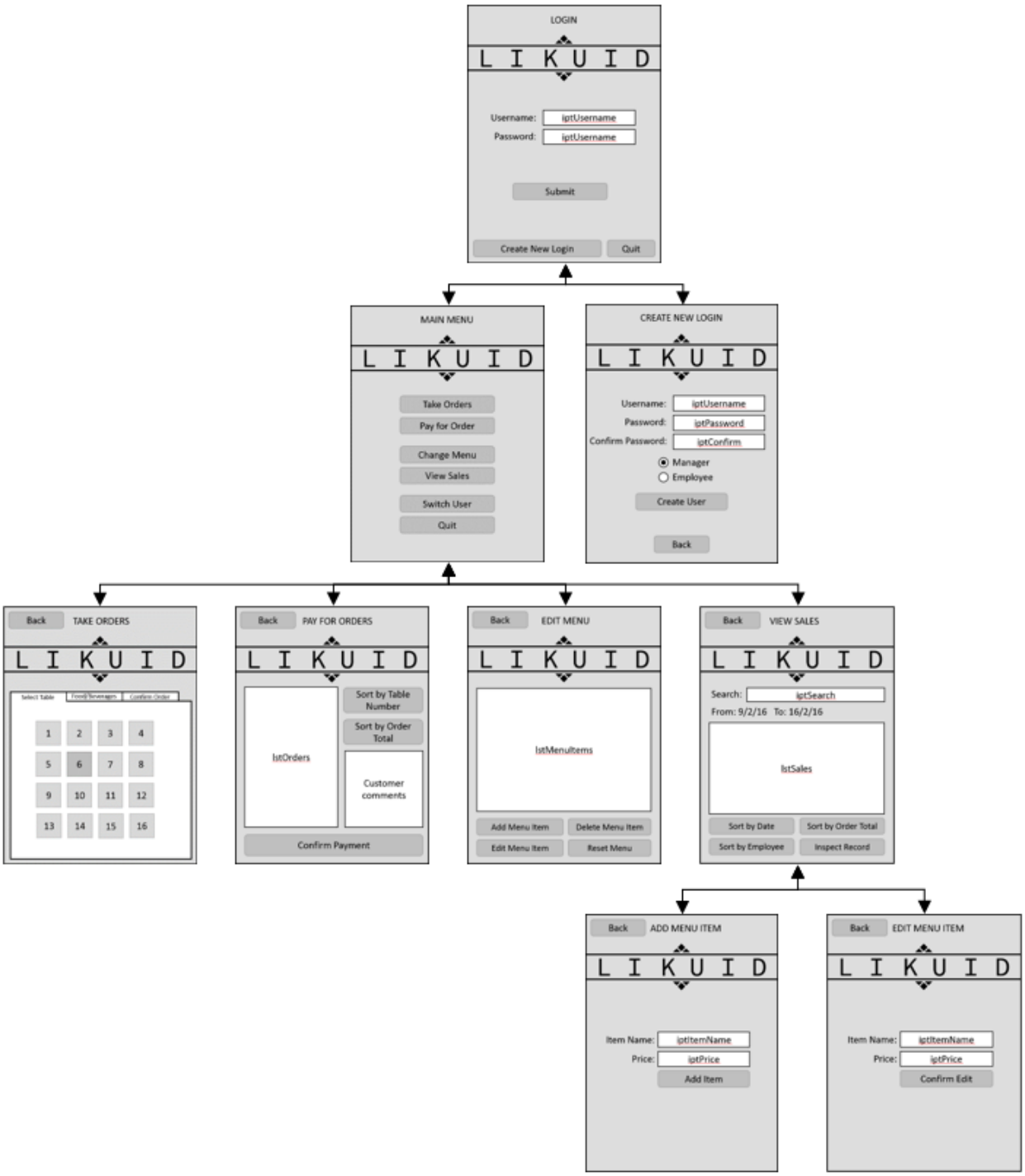
Price: iptPrice

Add Item

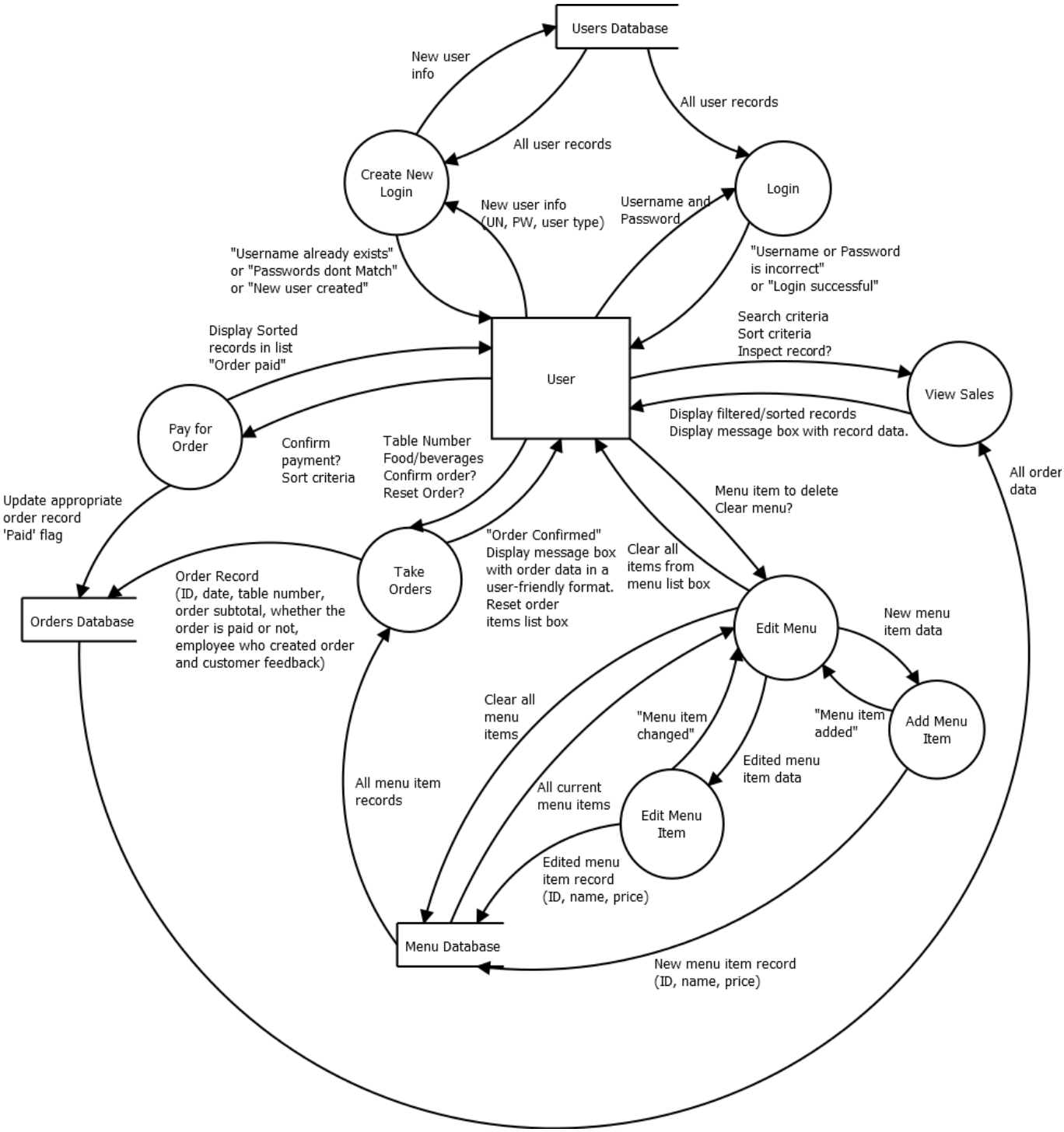
EDIT MENU ITEM



STORYBOARD


LOGIN
L I K U I D
Username:
iptUsernamе
Password:
iptUsernamе
Submit
Create New Login
Quit
MAIN MENU
L I K U I D
Take Orders
Pay for Order
Change Menu
View Sales
Switch User
Quit
CREATE NEW LOGIN
L I K U I D
Username:
iptUsernamе
Password:
iptPassword
Confirm Password:
iptConfirm
Manager
Employee
Create User
Back
Back
TAKE ORDERS
L I K U I D
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Back
PAY FOR ORDERS
L I K U I D
lstOrders
Sort by Table Number
Sort by Order Total
Customer comments
Confirm Payment
Back
EDIT MENU
L I K U I D
lstMenuItems
Add Menu Item
Delete Menu Item
Edit Menu Item
Reset Menu
Back
VIEW SALES
L I K U I D
Search:
iptSearch
From: 9/2/16 To: 16/2/16
lstSales
Sort by Date
Sort by Order Total
Sort by Employee
Inspect Record
Back
ADD MENU ITEM
L I K U I D
Item Name:
iptItemName
Price:
iptPrice
Add Item
Back
EDIT MENU ITEM
L I K U I D
Item Name:
iptItemName
Price:
iptPrice
Confirm Edit

DATA FLOW DIAGRAM



DATA DICTIONARY

NAME	DATA TYPE	SCOPE	DESCRIPTION	EXAMPLE	VALIDATION	FORMAT
GENERAL (Used multiple times)						
users	Array of UserRecords	Local	Local variable for holding data of all users.	0 “Alex” “myPassword” True 1 “Tom” “hisPassword” False	N/A	N/A
menuItems	Array of MenuItem Records	Local	Local variable for holding data of all menu items	0 “Pie” 3.5 True 1 “Coke” 3 False	N/A	N/A
allOrders	Array of Order Records	Local	Local variable for holding data of all orders ever made.	0 12/05/16 12 13.5 True “Alex” “Great nachos” 1 13/05/16 5 40 False “Tom” “None”	N/A	N/A
nextID	Integer	Local	Local variable for keeping track of the next ID that we want to open up for use in an array.	4	N/A	N/A
arrToReturn	Generic	Local	The array of items that we want to return to the user.	0	N/A	N/A
sender	Object	Local	Variable used as a parameter for subroutines that are called by user interactions, indicates the UI object that called the subroutine.	togTable2	N/A	N/A

e	Event	Local	Variable used as a parameter for subroutines that are called by user interactions, indicates the exact event that took place when calling the subroutine.	togTable2. Checked Changed	N/A	N/A
LOGIN						
currentUser	String	Public	Public variable for keeping track of the user that is currently logged in.	“Alex”	N/A	N/A
inputUsername	String	Local	Parameter passed into CheckLogin ( ) function, the username we want to check against the users database.	“Alex”	Length < maxUserLength	N/A
inputPassword	String	Local	Parameter passed into CheckLogin ( ) function, the password we want to check against the users database.	“myPassword”	N/A	N/A
NEW LOGIN						
maxUserLength	Integer	Private	Constant for deciding the maximum username length when creating new users	16	N/A	N/A
userToSearch	String	Local	Parameter passed into UserAlreadyExists ( ) function, the username we want to check if it exists or not.	“Tom”	N/A	N/A

usernames	Array of Strings	Local	Array of strings for holding the usernames of all users.	“Alex” “Tom”	Same length as Users ( )	N/A
TAKE ORDERS						
selectedTable Number	Integer	Private	Variable for keeping track of the currently selected table number	2	Will always be a number 1-16	N/A
orderItem Indexes	Array of Integers	Private	Array for keeping track of the MenuItemIDs of all items that have been ordered.	0, 1, 1, 3	N/A	N/A
menuItemNam e	String	Local	Parameter passed into GetMenuItem Index ( ) function, the name of the menu item that we want to retrieve the corresponding itemID of.	“Coke”	N/A	N/A
orderItem Indexes	Array of integers	Local	Parameter passed into UpdateOrder List ( ) subroutine and Calculate OrderSubtotal ( ) function, an array of the MenuItemIDs of all items that have been ordered.	0, 1, 1, 3	N/A	N/A
subtotal	Single	Local	A floating point variable that keeps track of the order total.	13.5	N/A	N/A
printString	String	Local	Variable that	“Order Confirmed.”	N/A	N/A

			contains the string that we want to print in the receipt message box.			
PAY FOR ORDERS						
currentUnpaid Orders	Array of Order Records	Private	Array of all orders that are currently being displayed in the orders list box.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 5 40 False “Tom” “None”	Note the array will only ever contain OrderRecords with ‘False’ for the orderPaid field and were made on the current day.	N/A
ordersToReturn	Array of Order Records	Local	Local variable for keeping track of all the current, unpaid orders that we want to return from the subroutine GetCurrent UnpaidOrders ( )	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 5 40 False “Tom” “None”	Note the array will only ever contain OrderRecords with ‘False’ for the orderPaid field and were made on the current day.	N/A
arrToDisplay	Array of Order Records	Local	Parameter passed into DisplayArray InListBox ( ) subroutine, storing the data of all the orders that we want to display.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 5 40 False “Tom” “None”	Note the array will only ever contain OrderRecords with ‘False’ for the orderPaid field and were made on the current day.	N/A
orderIDToEdit	Integer	Local	The orderID of the order we want to mark as ‘Paid’	0	N/A	N/A
ADD MENU ITEM						
itemPrice	Single	Local	Price of the item we want to add, to 2 decimal places.	3.50	2 decimal places	N/A

EDIT MENU ITEM						
itemToEditIndex	Integer	Private	The index of the item that we want to edit.	2	N/A	N/A
Index	Integer	Local	Parameter passed into GetItemToEdit Index ( ) from the formEditMenu class, the index of the item we want to edit.	2	N/A	N/A
newRecord	MenuItem Record	Local	An empty MenuItemRecord that will replace the existing one with when we confirm the edit.	0 “” 0 False	N/A	N/A
VIEW SALES						
currently DisplayedItems	Array of Order Records	Private	Array of all order items that are currently being displayed in the sales list box.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
arrToDisplay	Array of Order Records	Local	Parameter passed into DisplayArray InSalesListBox ( ), the array of orderRecords that we want to display.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
startDate	Date	Local	Parameter passed into FindSales WithinDate Range ( ), the	12/05/16	<= endDate	N/A

			starting date for the filter.			
endDate	Date	Local	Parameter passed into FindSales WithinDate Range ( ), the end date for the filter.	13/05/16	>= startDate	N/A
salesToReturn	Array of Order Records	Local	Array of all sales we want to return from FindSalesWithin DateRange ( ) function.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
orderIDTo Display	Integer	Local	The ID of the order that we want to inspect	1	N/A	N/A
itemToDisplay	OrderRecor d	Local	Parameter passed into DisplayItem Info ( ) subroutine, the item we want to inspect	1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
ARRAY UTILITIES						
arrToAddTo	Array of Generic elements	Local	Parameter passed into AddElement ToArray ( ) subroutine, the array of items that we want to add to.	0, 1, 1, 3	N/A	N/A
elementToAdd	Generic	Local	Parameter passed into AddElement ToArray ( ) subroutine, the element that we	4	N/A	N/A

			want to add to the end of arrToAddTo.			
arrToRemove From	Array of Generic elements	Local	The array of items that we want to remove from, a parameter passed into Remove ElementFrom Array ( ) subroutine.	0, 1, 1, 3	N/A	N/A
indexOfElemen t ToRemove	Integer	Local	The index of the element that we want to remove from arrToRemove From, a parameter of RemoveElement FromArray ( )	2	N/A	N/A
arrToPurge	Array of Generic elements	Local	Parameter passed into PurgeArray ( ) subroutine, the array that we want to purge	0, 1, 1, 3	N/A	N/A
DATA UTILITIES						
UserRecord	Structure	Public	Structure for storing information about users on the system. Stores userID (Integer), username (String), password (String), manager (Boolean).	0 “Alex” “myPassword” True	N/A	N/A
MenuItem Record	Structure	Public	Structure for storing info on menu items.	0 “Coke” 3 False	N/A	N/A

			Contains itemID (Integer), itemName (string), itemPrice (Single), isFood (Boolean).			
OrderRecord	Structure	Public	Structure for storing data on all orders. Contains orderID (Integer), orderDate (Date), tableNumber (Integer), orderSubtotal (Single), orderPaid (Boolean), employee (String), customer Comments (String).	1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
filePath	String	Public	The file path of the project file.	Y:\Mirrington\ Major Project 2016\Program \Major Project 2016\Major Project 2016\bin\Deb ug	N/A	N/A
arrToLoadInto	UserRecord or MenuItem Record or OrderRecor d	Local	The array that we want to load the data from appropriate text files into. (Note type is dependent on which database we want to access)	0 “” 0 False	N/A	N/A
nullElement Removed	Boolean	Local	Flag for keeping track of whether the initial null element of the declared array has been removed or not.	False	N/A	N/A

oRead	Stream Reader	Local	Variable for sequentially accessing the contents of text files.	N/A	N/A	N/A
lineIn	String	Local	The line from the text file that is about to be processed.	0|13/05/16|7|35|True|Tom| Great nachos	N/A	N/A
tmp	Object	Local	Temporary variable for storing data while it is processed and placed into arrToLoadInto	N/A	N/A	N/A
currentRecord	UserRecord or MenuItem Record or OrderRecord	Local	The current record that we are analysing when loading data from text files.	1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
recordToAdd	UserRecord or MenuItem Record or OrderRecord	Local	The record we want to add to the relational database when adding a new record.	1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
arrRecords	Array of UserRecords or MenuItem Records or Order Records	Local	The array of records we want to add new records to.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
username	String	Local	Parameter passed into AddUser Record ( ) subroutine, the username component of the UserRecord we want to add to the relative file.	“Alex”	N/A	N/A
password	String	Local		“myPassword”	N/A	N/A

			Parameter passed into AddUser Record ( ) subroutine, the password component of the UserRecord we want to add to the relative file.			
manager	Boolean	Local	Parameter passed into AddUser Record ( ) subroutine, the manager component of the UserRecord we want to add to the relative file.	True	N/A	N/A
menuItemNam e	String	Local	Parameter passed into AddMenuItem Record ( ) subroutine, the name component of the MenuItemRecord we want to add to the relative file.	“Coke”	N/A	N/A
menuItemPrice	Single	Local	Parameter passed into AddMenuItem Record ( ) subroutine, the price component of the MenuItemRecord we want to add to the relative file.	3.5	N/A	N/A
isFood	Boolean	Local	Parameter passed into AddMenuItem Record ( ) subroutine, the isFood component of the MenuItemRecord	False	N/A	N/A

			we want to add to the relative file.			
orderDate	Date	Local	Parameter passed into AddOrder Record ( ) subroutine, the date component of the OrderRecord we want to add to tthe relative file.	17/05/16	N/A	N/A
tableNumber	Integer	Local	Parameter passed into AddOrder Record ( ) subroutine, the table number component of the OrderRecord we want to add to the relative file.	4	N/A	N/A
orderSubtotal	Single	Local	Parameter passed into AddOrder Record ( ) subroutine, the subtotal component of the OrderRecord we want to add to the relative file.	34.5	N/A	N/A
orderPaid	Boolean	Local	Parameter passed into AddOrder Record ( ) subroutine, the orderPaid component of the OrderRecord we want to add to the relative file.	True	N/A	N/A
employee	String	Local	Parameter passed into AddOrder Record ( ) subroutine, the	“Alex”	N/A	N/A

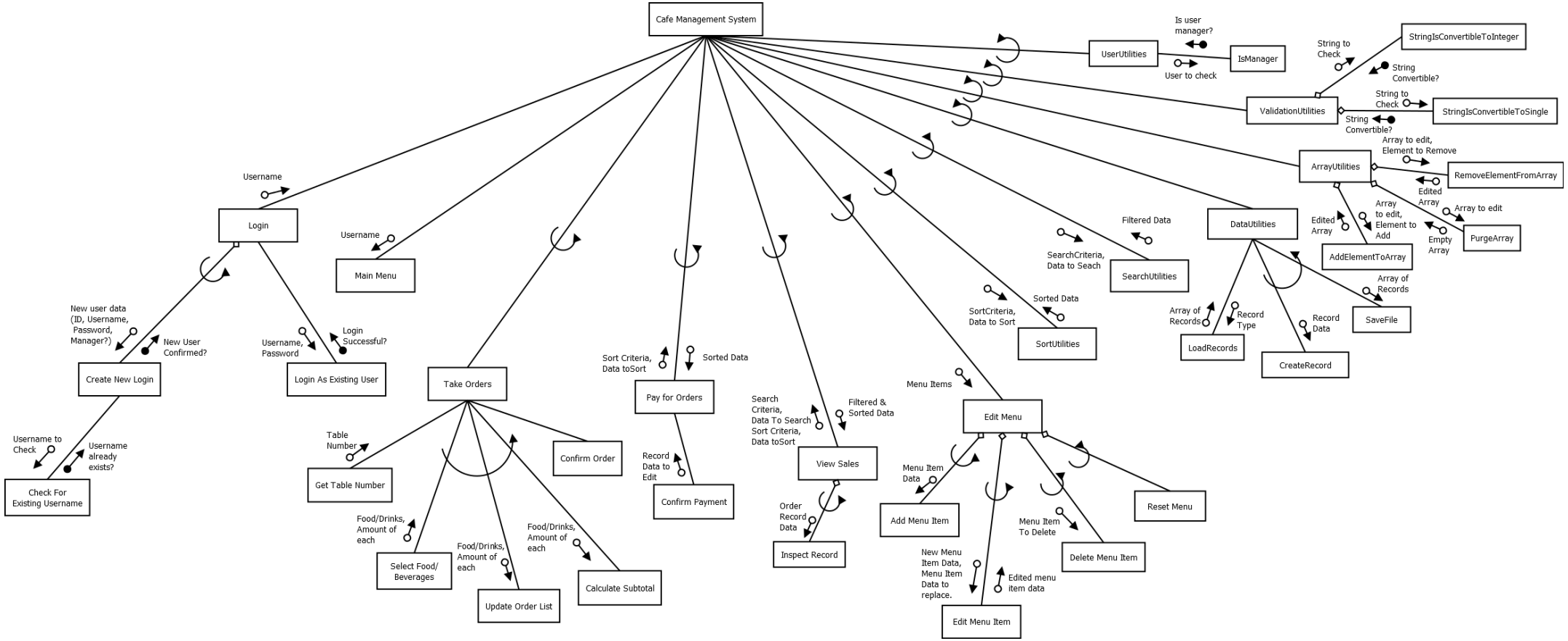
			employee component of the OrderRecord we want to add to the relative file.			
customer Comments	String	Local	Parameter passed into AddOrder Record ( ) subroutine, the comments component of the OrderRecord we want to add to the relative file.	“Great Nachos”	N/A	N/A
arrToSave	Array of UserRecord s or MenuItem Records or Order Records	Local	Parameter passed into SaveUserArray ToFile ( ), SaveMenuTo File ( ) or SaveOrderTo File ( ) subroutines, the array of items that we want to save to the file.			
sep	Char	Local	The separator character for separating individual fields when writing temporary data to text files.	“|”	N/A	N/A
writeLine	String	Local	The string that is to be written to the text file.	0|13/05/16|7|35|True|Tom| Great nachos	N/A	N/A
fileName	String	Local	Parameter passed into TextFileIs Empty ( ) function, the name of the text file we want to check for data.	“sales”	N/A	N/A
SEARCH UTILITIES						
arrToSearch	Array of Generic elements	Local	Parameter passed into	0, 1, 1, 3	N/A	N/A

			ItemExistsIn Array ( ) function, an array of all the data items we want to search.			
elementTo SearchFor	Generic	Local	Parameter passed into ItemExistsIn Array ( ) function, the element we want to check exists in arrToSearch	3	N/A	N/A
arrToSearch	Array of Order Records	Local	Parameter passed into FindRecords WithSearch Criteria ( ), an array of all the data items we want to search.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
searchCriteria	String	Local	Parameter passed into FindRecords WithSearch Criteria ( ), the string that we want to locate in specific fields of elements in arrToSearch	“Nachos”	N/A	N/A
SORT UTILITIES						
arrToSort	Array of Order Records	Local	Parameter passed into various insertion sort algorithms, the array that we want to sort.	0 12/05/16 12 13.5 False “Alex” “None” 1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
firstIndex	Integer	Local	The index of arrToSort where we want to start sorting.	0	N/A	N/A
lastIndex	Integer	Local	The last index of arrToSort that we want to sort.	2	N/A	N/A

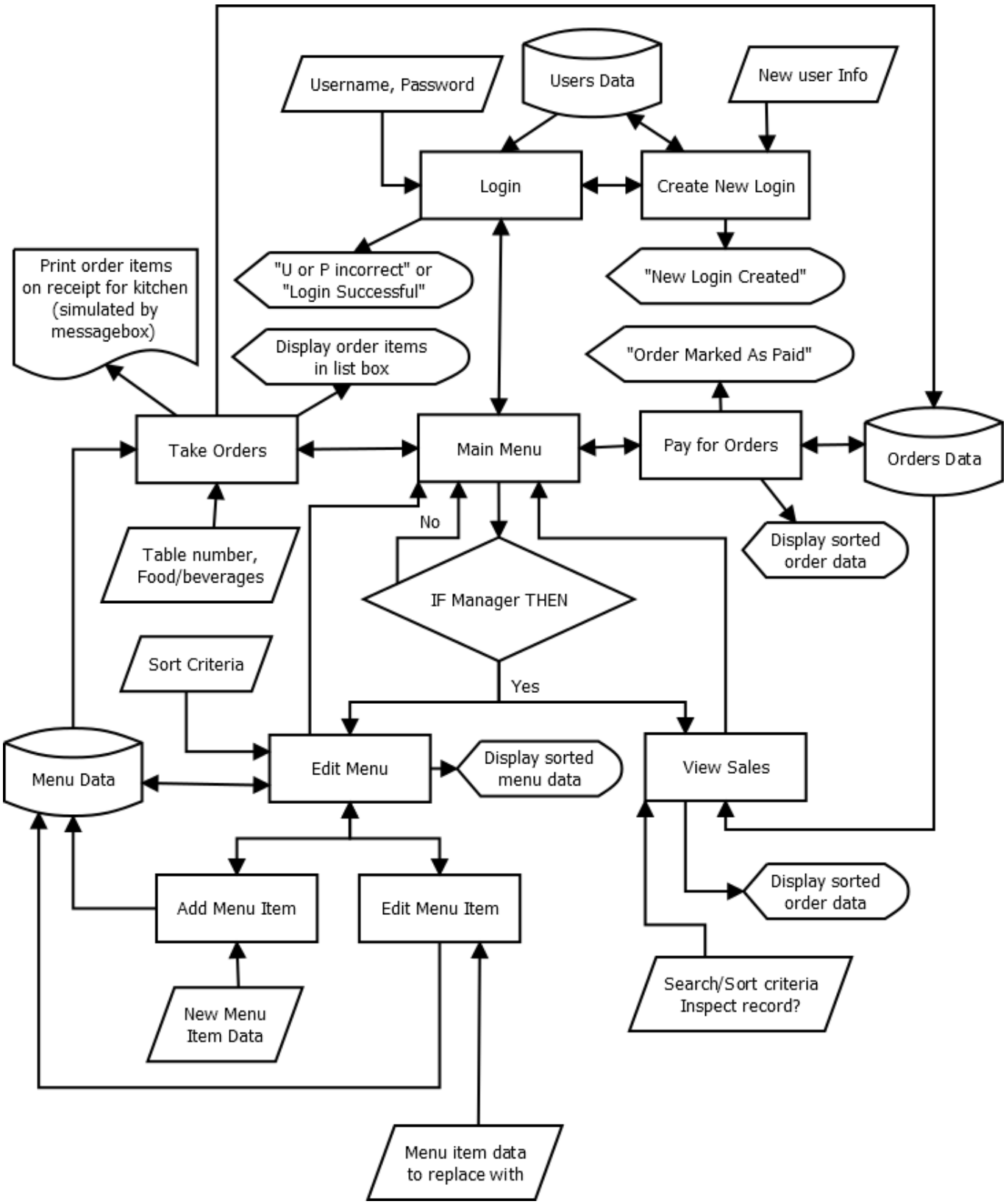
indexToSort	Integer	Local	The index of the array that we are trying to insert into the correct position.	1	N/A	N/A
currentIndex OfLoop	Integer	Local	The current index of the inner loop, for keeping track of which item we are currently comparing to the item we are sorting.	1	N/A	N/A
temp	OrderRecor d	Local	Temporary variable for storing the element to be inserted while he rest of the elements are shifted upwards.	1 13/05/16 7 35 True “Tom” “Great nachos”	N/A	N/A
VALIDATION UTILITIES						
str	String	Local	Parameter passed into StringIs ConvertibleTo Integer ( ) and StringIs ConvertibleTo Single ( ) subroutines, the string we want to check.	“2”	N/A	N/A
USER UTILITIES						
userToCheck	String	Local	Parameter passed into IsManager ( ) function, the name of the user	“Alex”	N/A	N/A

			we want to check is a manager or not.			

STRUCTURE CHART



SYSTEM FLOWCHART

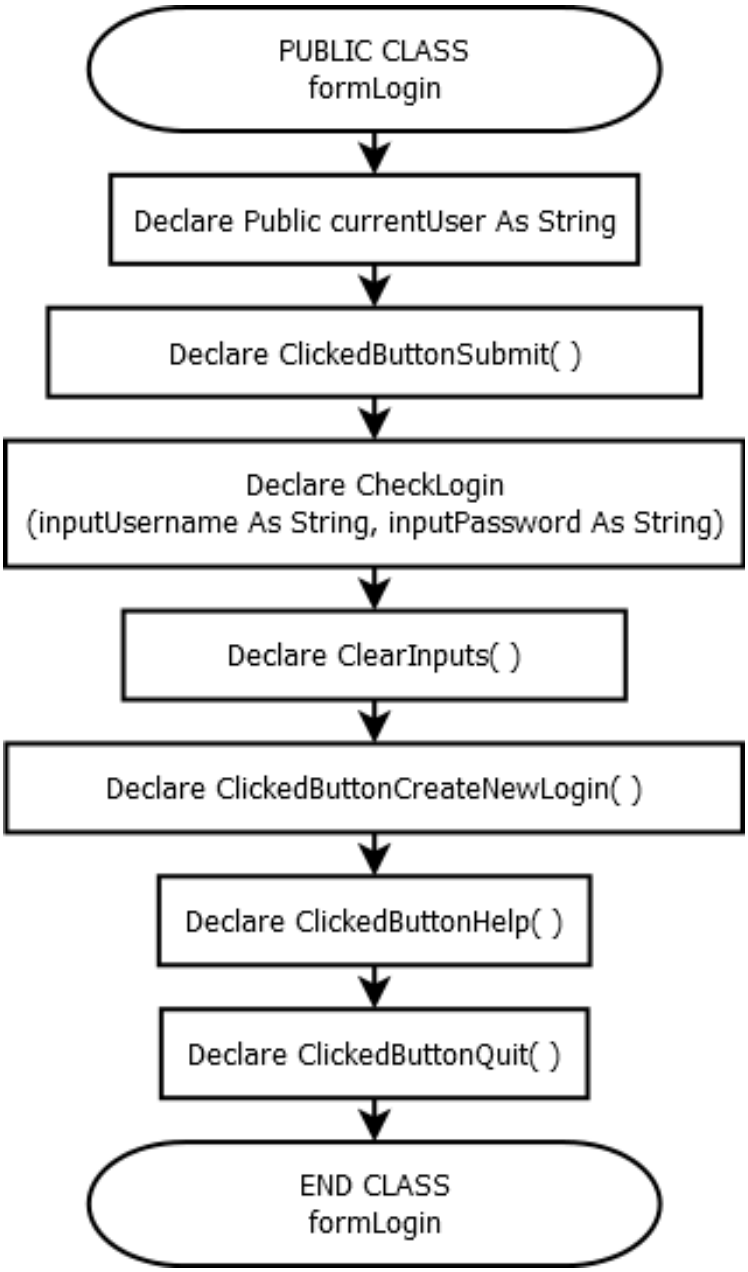


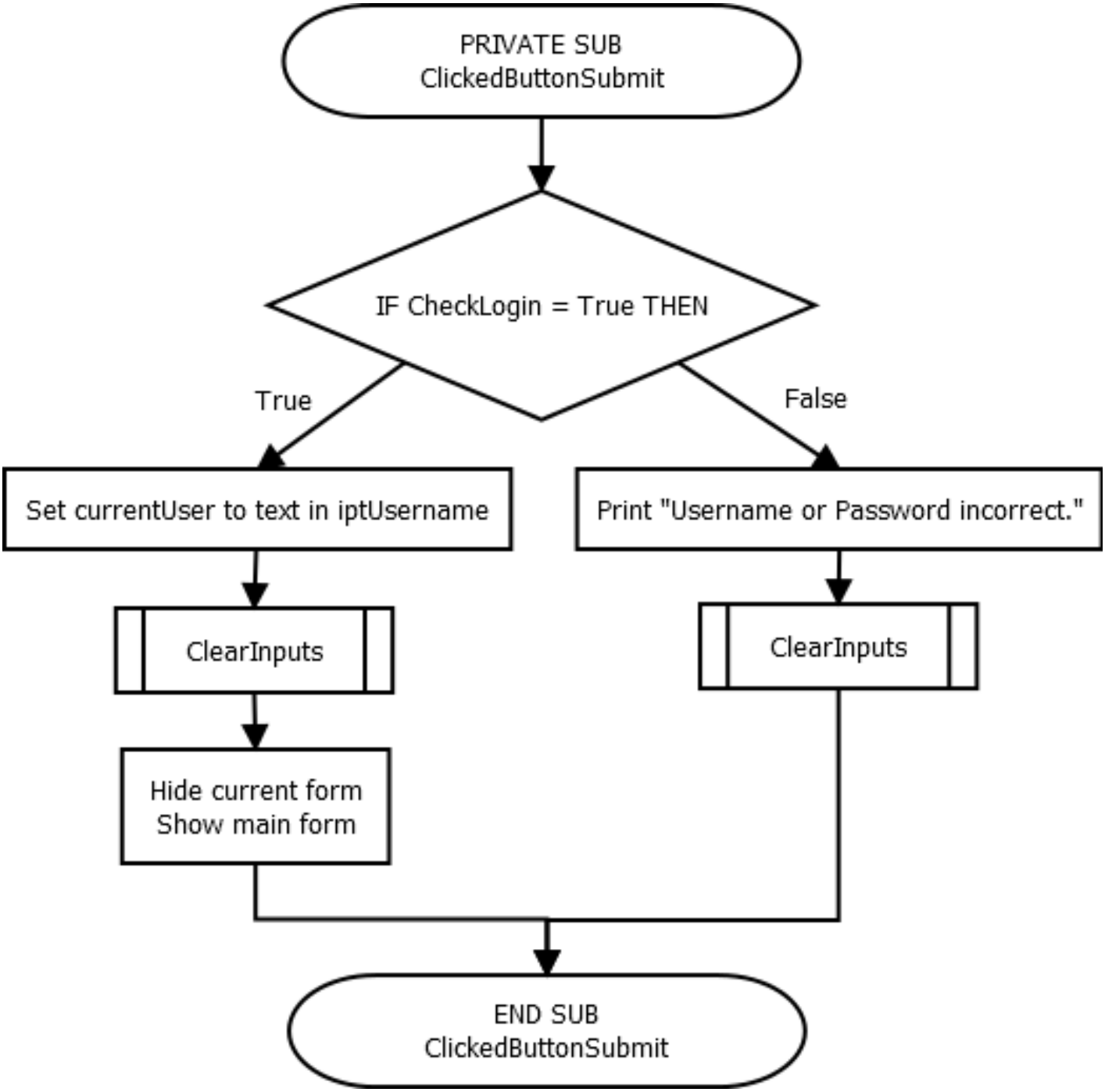
IMPLEMENTING

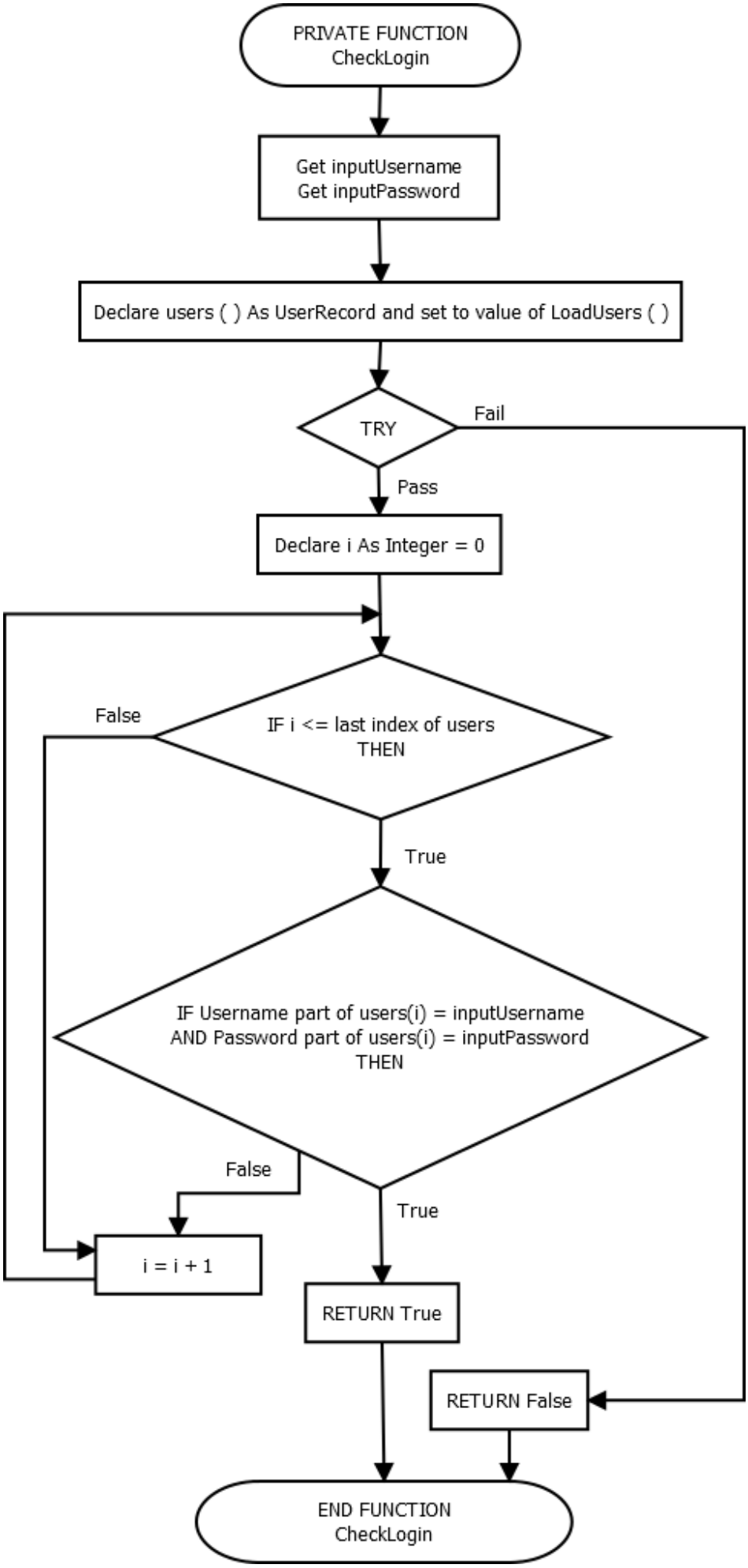
ALGORITHMS

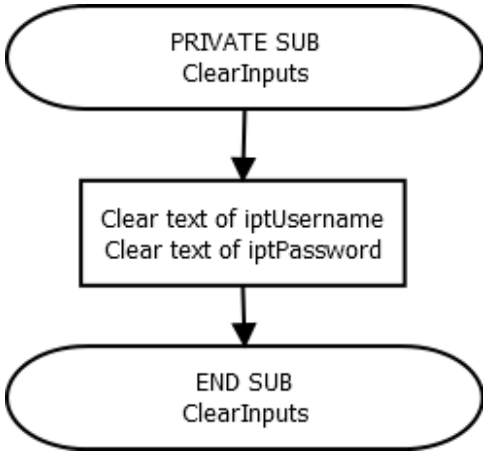
FLOWCHARTS

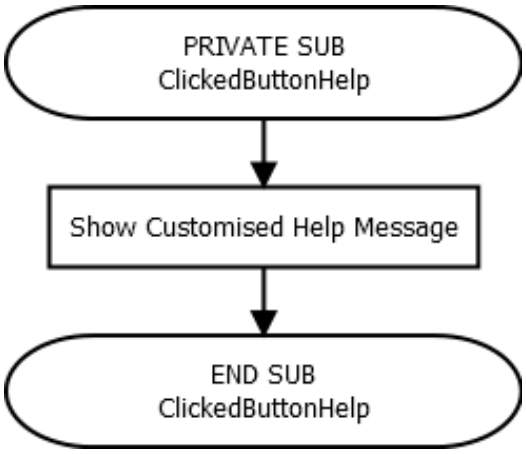
LOGIN

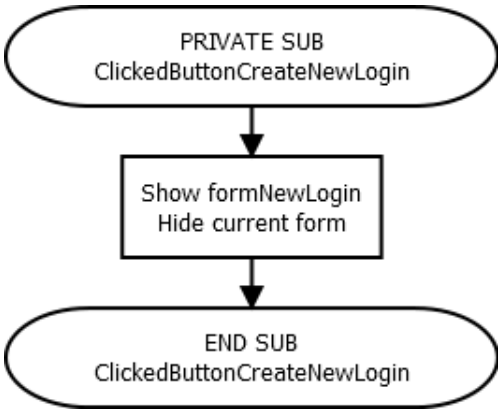


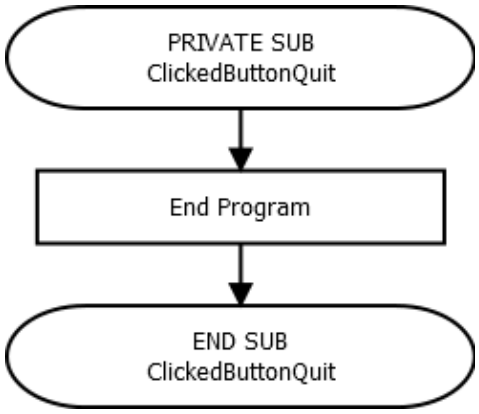

PRIVATE SUB
ClickedButtonSubmit
IF CheckLogin = True THEN
True
False
Set currentUser to text in iptUsername
Print "Username or Password incorrect."
ClearInputs
ClearInputs
Hide current form
Show main form
END SUB
ClickedButtonSubmit


PRIVATE FUNCTION
CheckLogin
Get inputUsername
Get inputPassword
Declare users ( ) As UserRecord and set to value of LoadUsers ( )
TRY
Fail
Pass
Declare i As Integer = 0
False
IF i <= last index of users
THEN
True
IF Username part of users(i) = inputUsername
AND Password part of users(i) = inputPassword
THEN
False
True
i = i + 1
RETURN True
RETURN False
END FUNCTION
CheckLogin

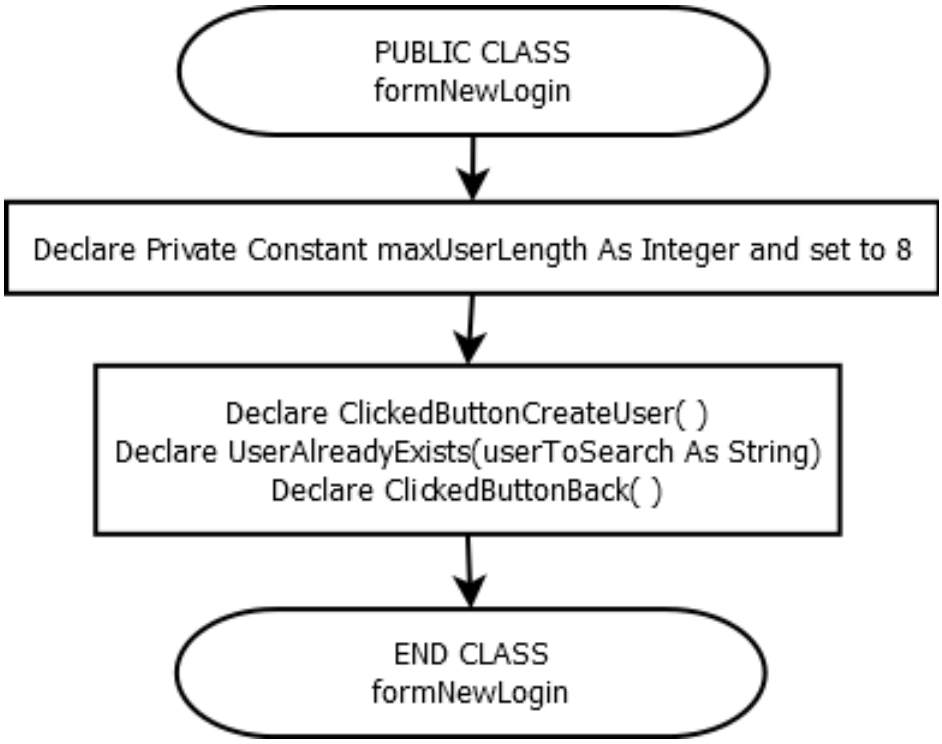

PRIVATE SUB
ClearInputs
Clear text of iptUsername
Clear text of iptPassword
END SUB
ClearInputs

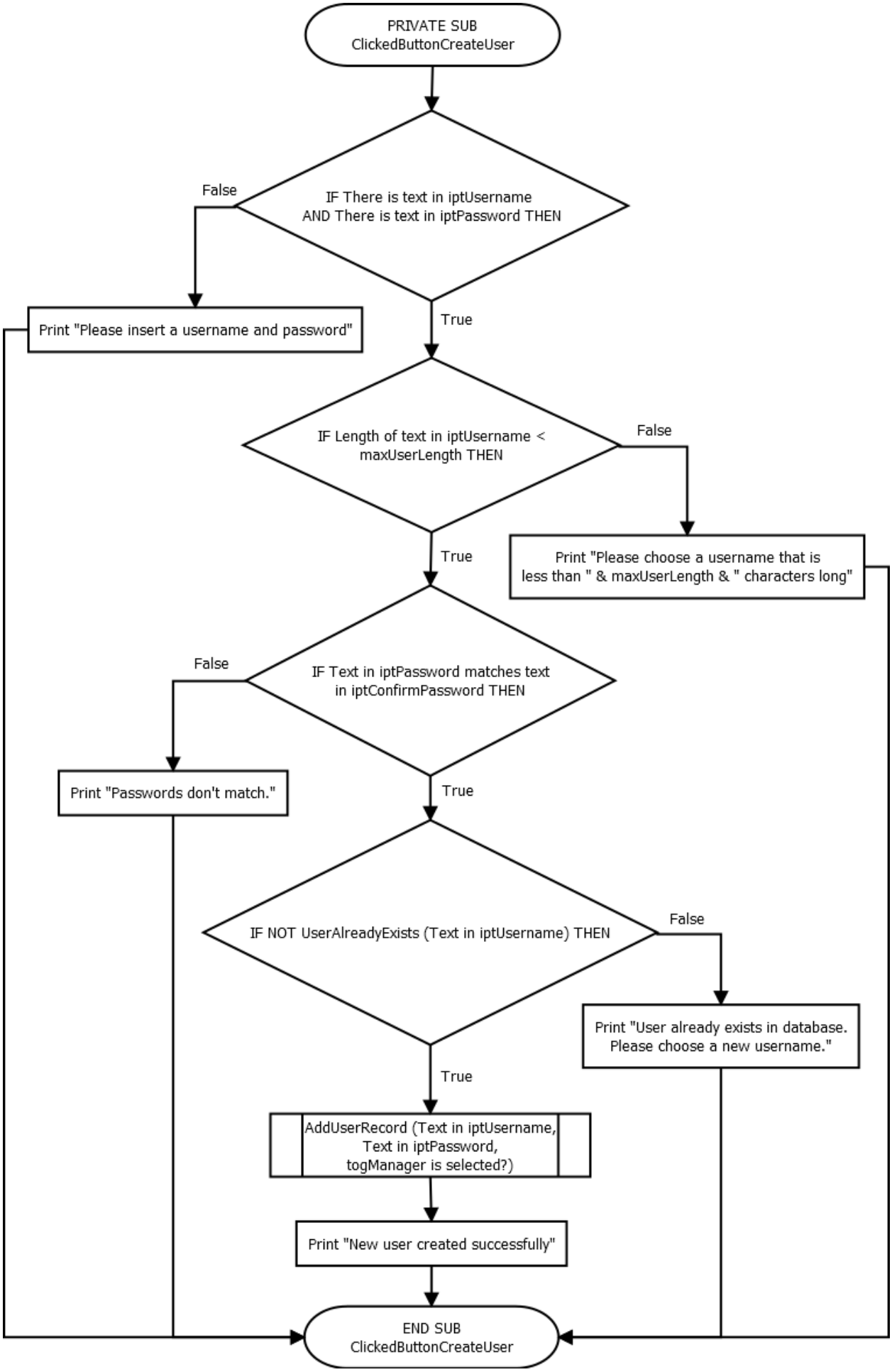

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

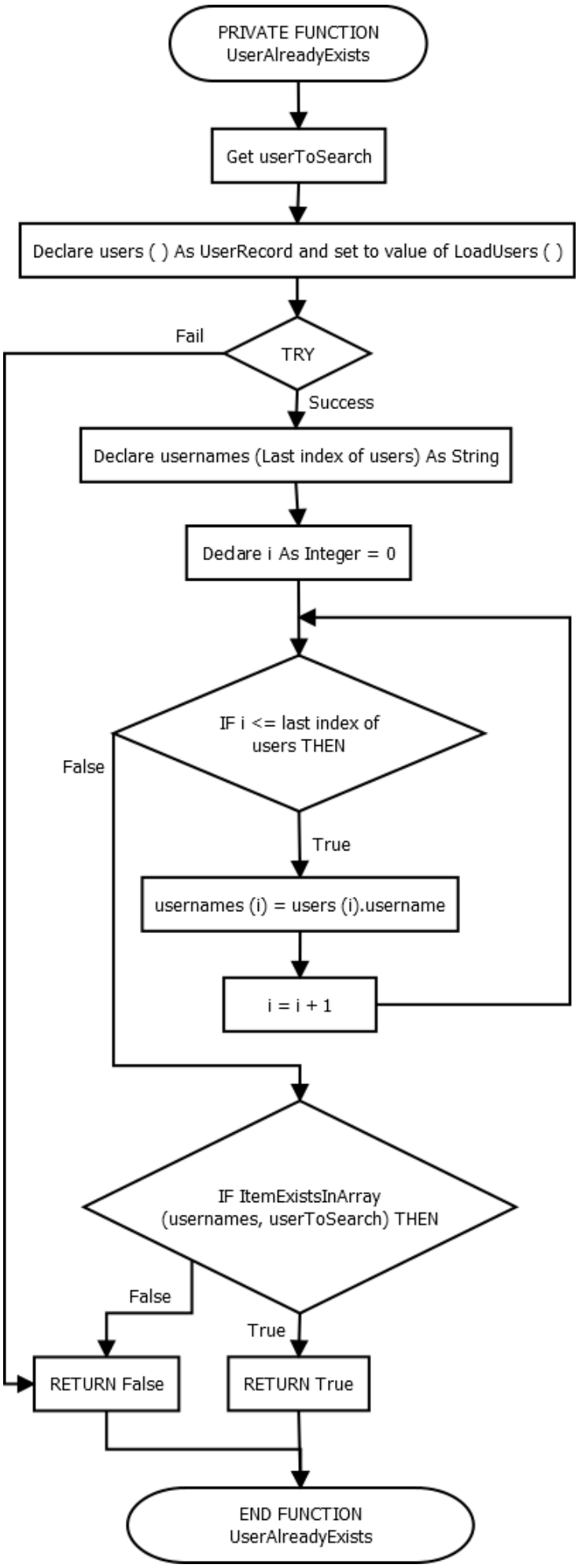

PRIVATE SUB
ClickedButtonCreateNewLogin
Show formNewLogin
Hide current form
END SUB
ClickedButtonCreateNewLogin

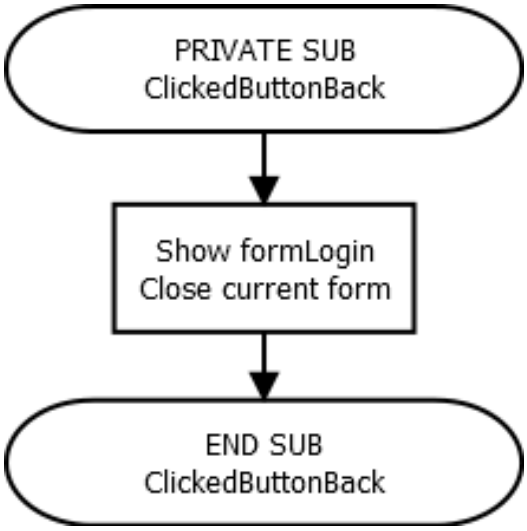

PRIVATE SUB
ClickedButtonQuit
End Program
END SUB
ClickedButtonQuit

NEW LOGIN

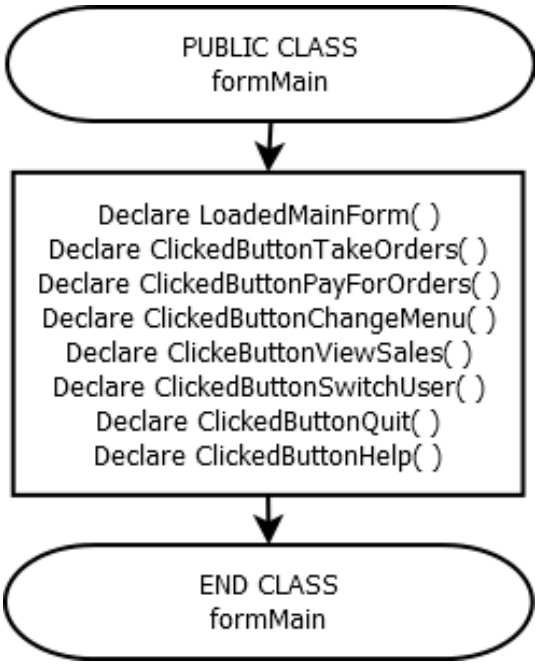

PUBLIC CLASS
formNewLogin
Declare Private Constant maxUserLength As Integer and set to 8
Declare ClickedButtonCreateUser( )
Declare UserAlreadyExists(userToSearch As String)
Declare ClickedButtonBack( )
END CLASS
formNewLogin

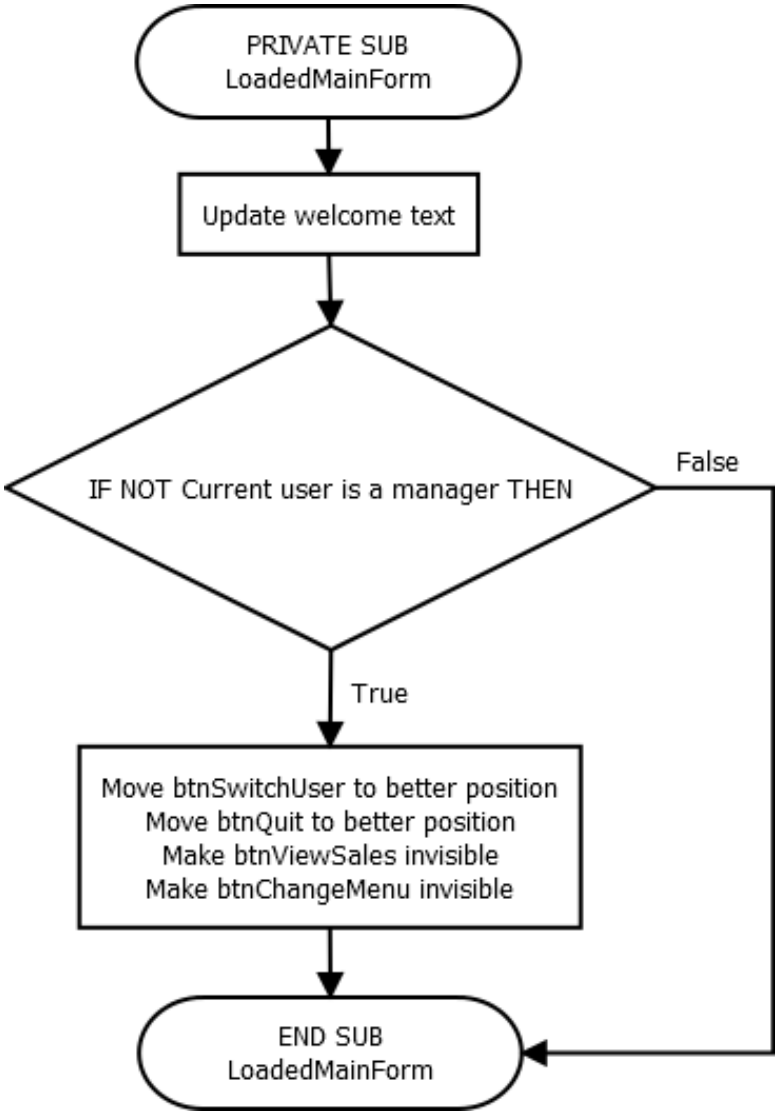

PRIVATE SUB
ClickedButtonCreateUser
IF There is text in iptUsername
AND There is text in iptPassword THEN
False
True
Print "Please insert a username and password"
IF Length of text in iptUsername <
maxUserLength THEN
False
True
Print "Please choose a username that is
less than " & maxUserLength & " characters long"
IF Text in iptPassword matches text
in iptConfirmPassword THEN
False
True
Print "Passwords don't match."
IF NOT UserAlreadyExists (Text in iptUsername) THEN
False
True
Print "User already exists in database.
Please choose a new username."
AddUserRecord (Text in iptUsername,
Text in iptPassword,
togManager is selected?)
Print "New user created successfully"
END SUB
ClickedButtonCreateUser

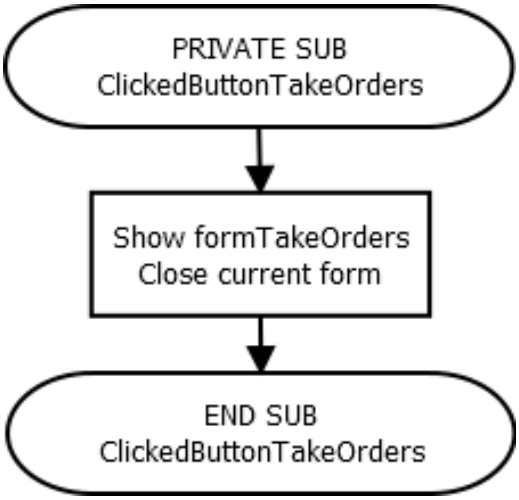

PRIVATE FUNCTION
UserAlreadyExists
Get userToSearch
Declare users ( ) As UserRecord and set to value of LoadUsers ( )
Fail
TRY
Success
Declare usernames (Last index of users) As String
Declare i As Integer = 0
IF i <= last index of
users THEN
False
True
usernames (i) = users (i).username
i = i + 1
IF ItemExistsInArray
(usernames, userToSearch) THEN
False
True
RETURN False
RETURN True
END FUNCTION
UserAlreadyExists

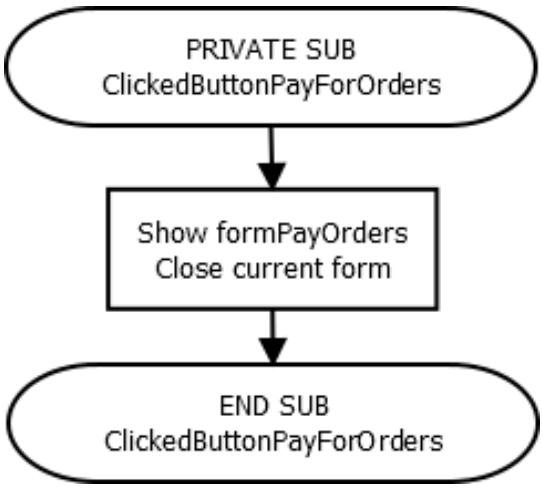

PRIVATE SUB
ClickedButtonBack
Show formLogin
Close current form
END SUB
ClickedButtonBack

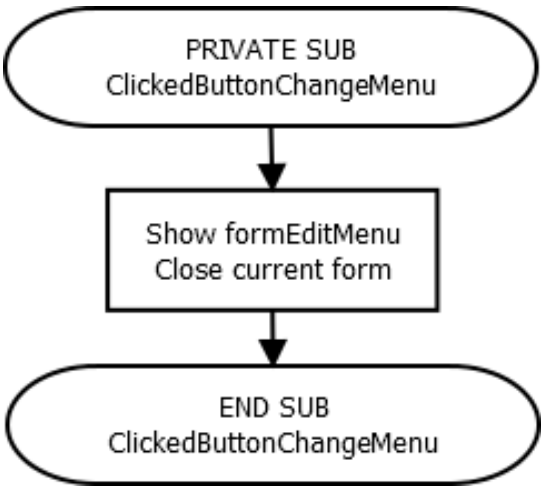
MAIN FORM

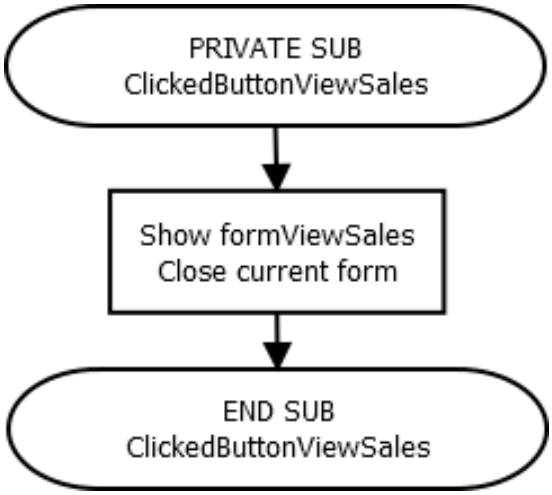

PUBLIC CLASS
formMain
Declare LoadedMainForm( )
Declare ClickedButtonTakeOrders( )
Declare ClickedButtonPayForOrders( )
Declare ClickedButtonChangeMenu( )
Declare ClickeButtonViewSales( )
Declare ClickedButtonSwitchUser( )
Declare ClickedButtonQuit( )
Declare ClickedButtonHelp( )
END CLASS
formMain

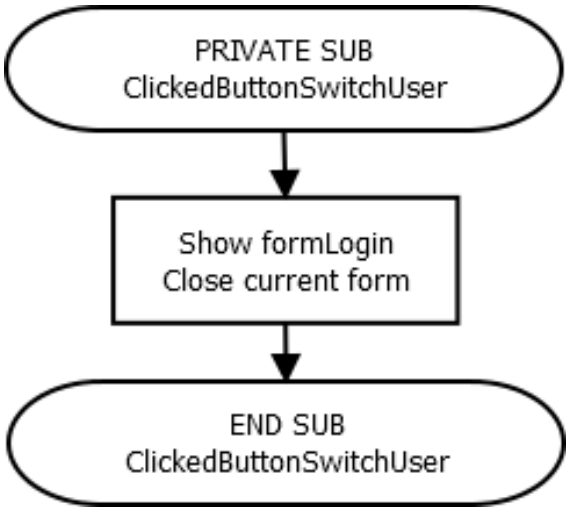

PRIVATE SUB
LoadedMainForm
Update welcome text
IF NOT Current user is a manager THEN
False
True
Move btnSwitchUser to better position
Move btnQuit to better position
Make btnViewSales invisible
Make btnChangeMenu invisible
END SUB
LoadedMainForm

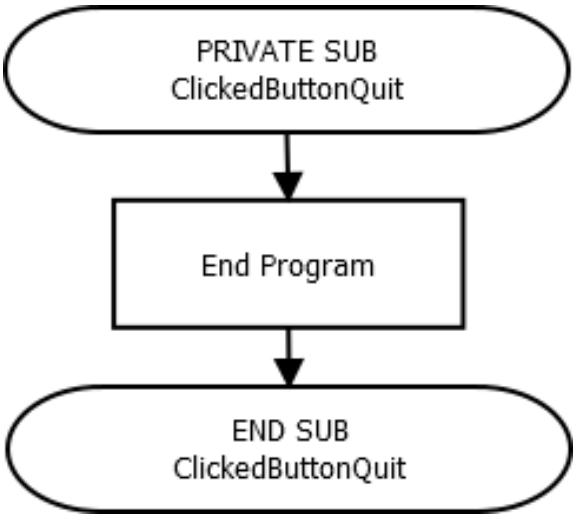

PRIVATE SUB
ClickedButtonTakeOrders
Show formTakeOrders
Close current form
END SUB
ClickedButtonTakeOrders

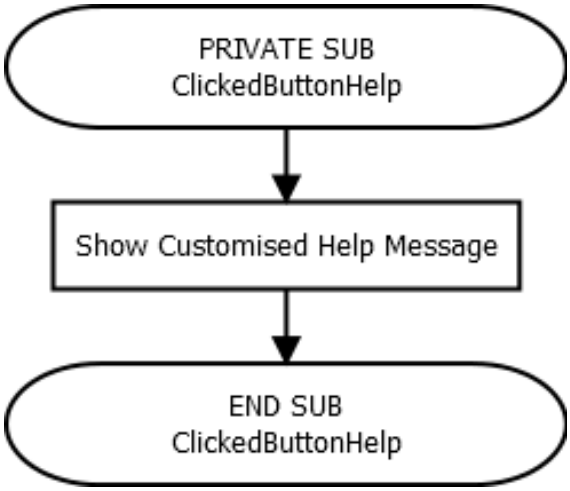

PRIVATE SUB
ClickedButtonPayForOrders
Show formPayOrders
Close current form
END SUB
ClickedButtonPayForOrders


PRIVATE SUB
ClickedButtonChangeMenu
Show formEditMenu
Close current form
END SUB
ClickedButtonChangeMenu

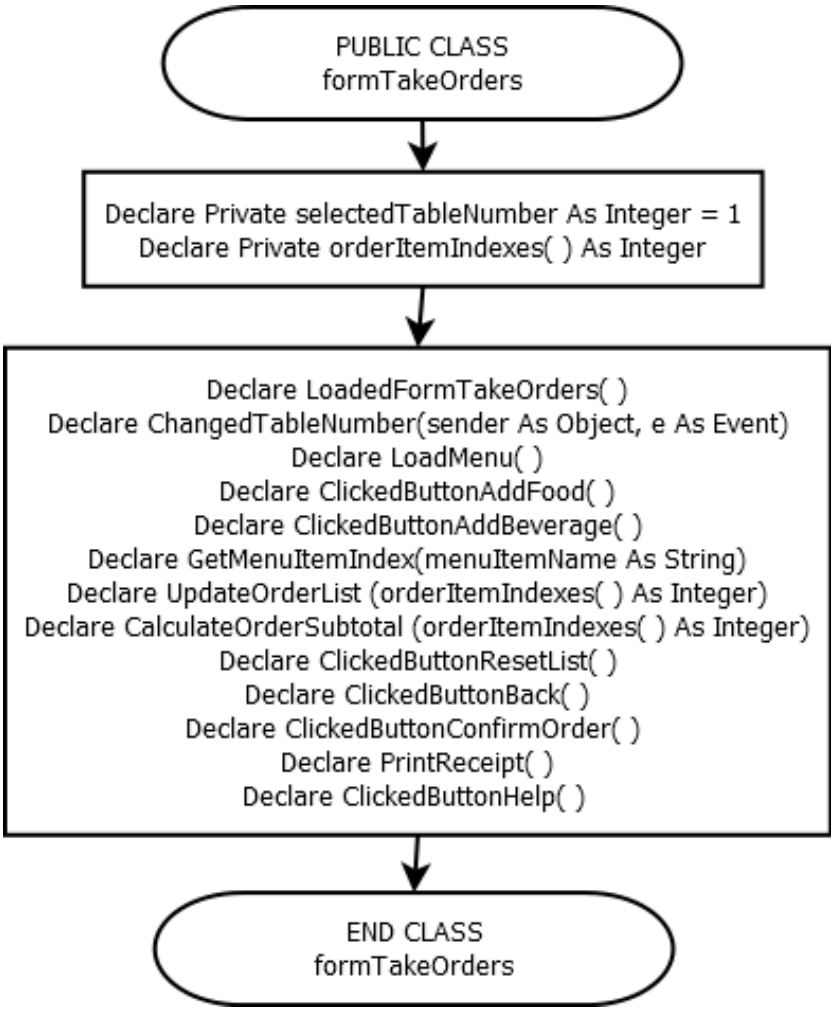

PRIVATE SUB
ClickedButtonViewSales
Show formViewSales
Close current form
END SUB
ClickedButtonViewSales

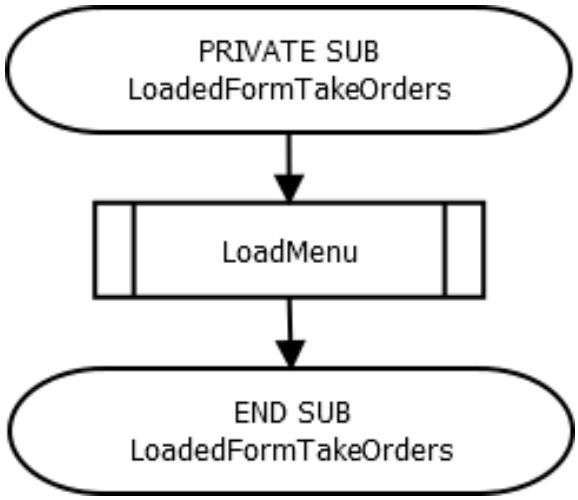

PRIVATE SUB
ClickedButtonSwitchUser
Show formLogin
Close current form
END SUB
ClickedButtonSwitchUser

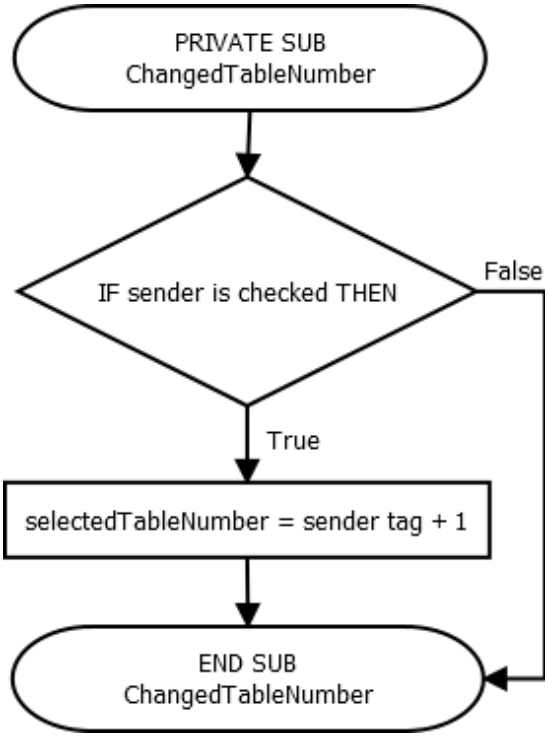

PRIVATE SUB
ClickedButtonQuit
End Program
END SUB
ClickedButtonQuit

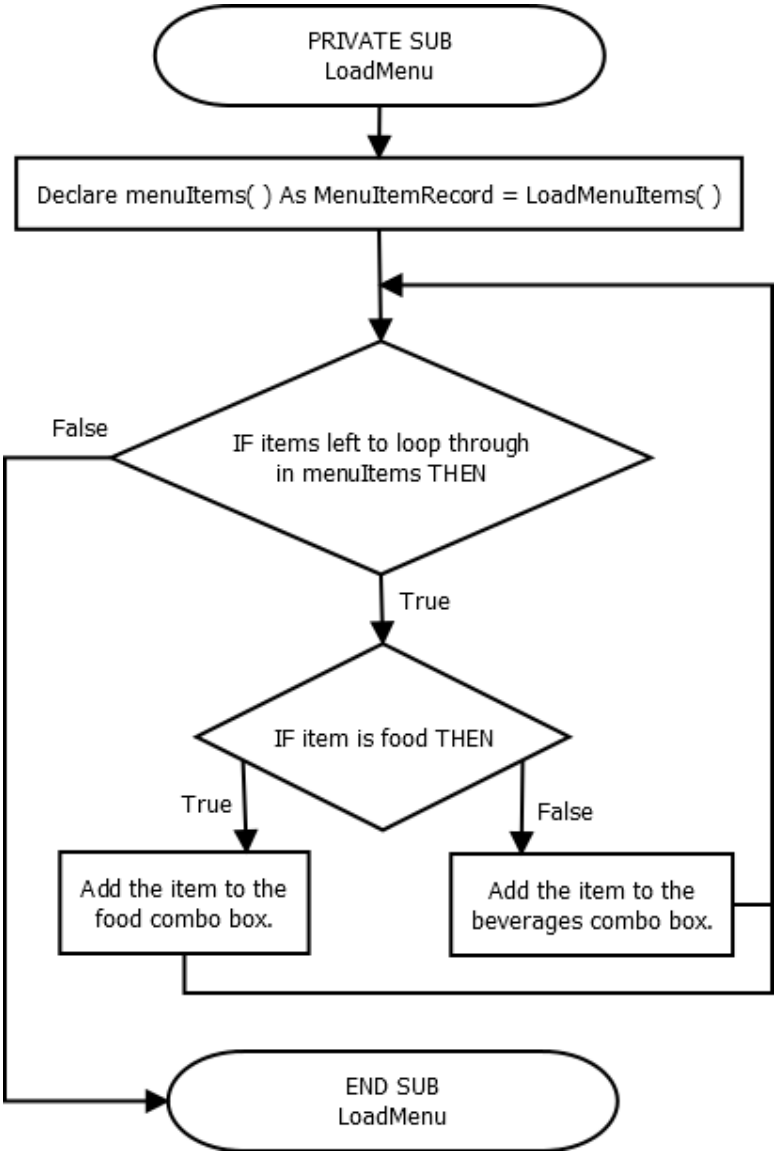

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

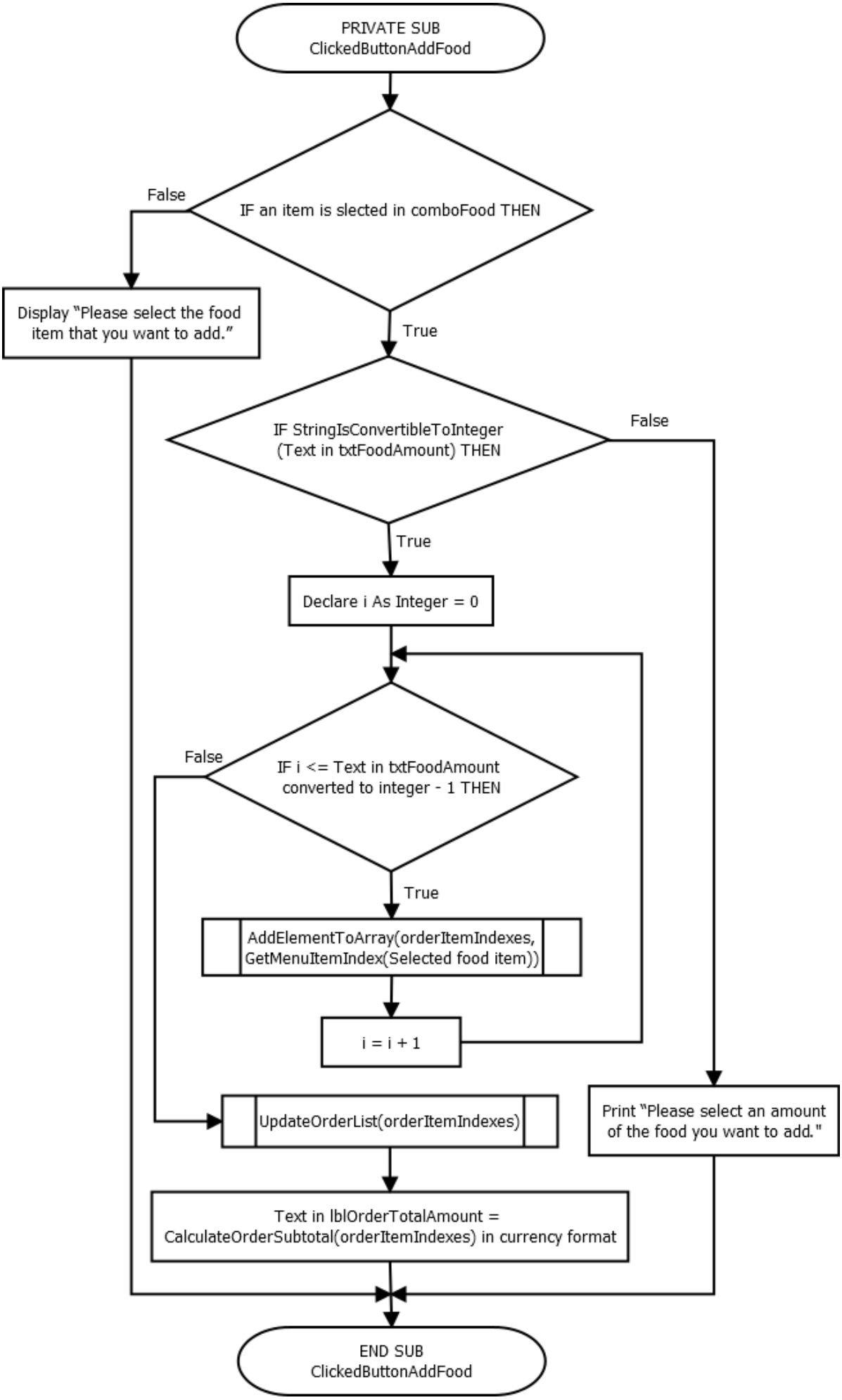
TAKE ORDERS

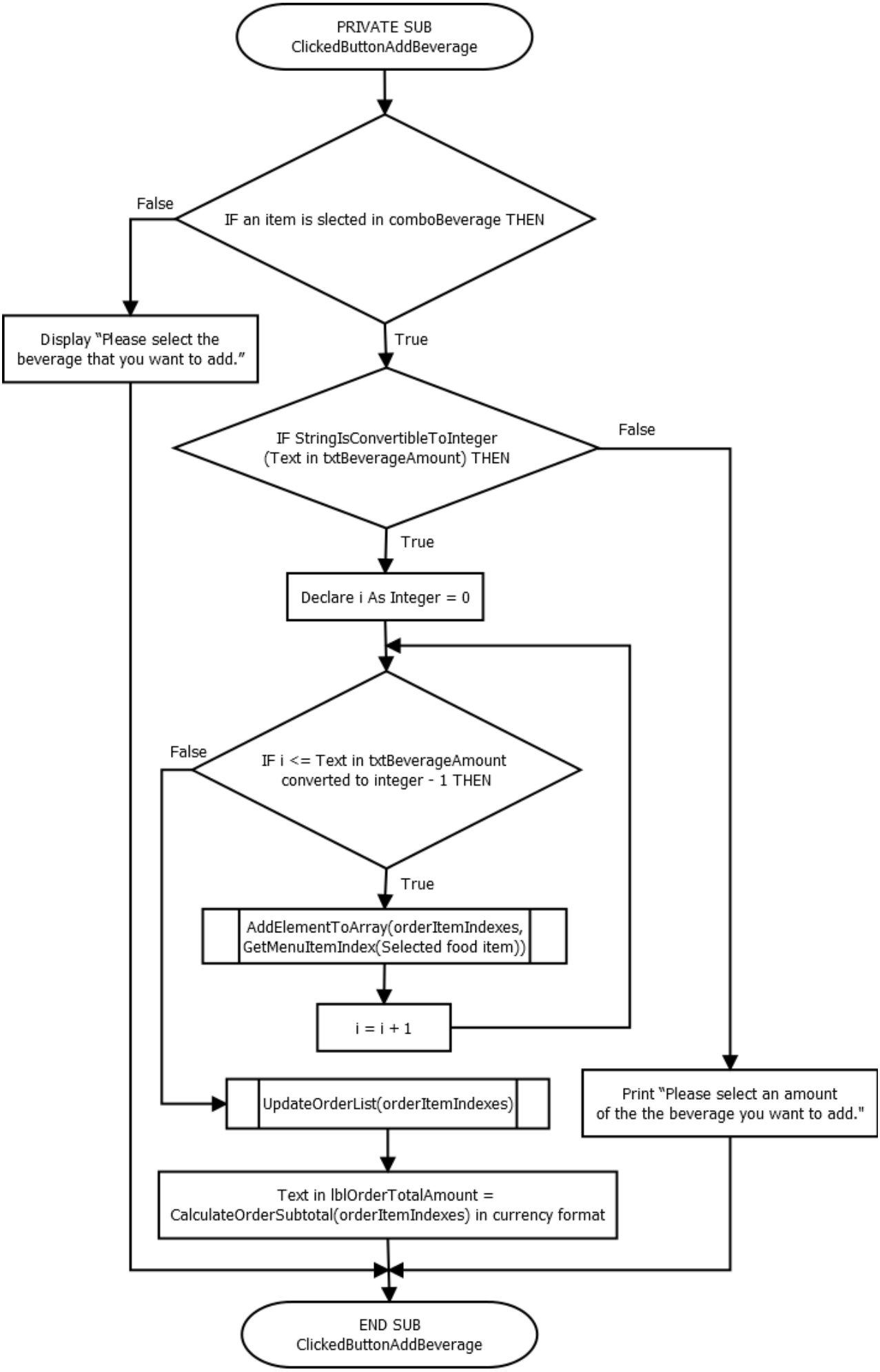

PUBLIC CLASS
formTakeOrders
Declare Private selectedTableNumber As Integer = 1
Declare Private orderItemIndexes( ) As Integer
Declare LoadedFormTakeOrders( )
Declare ChangedTableNumber(sender As Object, e As Event)
Declare LoadMenu( )
Declare ClickedButtonAddFood( )
Declare ClickedButtonAddBeverage( )
Declare GetMenuItemIndex(menuItemName As String)
Declare UpdateOrderList (orderItemIndexes( ) As Integer)
Declare CalculateOrderSubtotal (orderItemIndexes( ) As Integer)
Declare ClickedButtonResetList( )
Declare ClickedButtonBack( )
Declare ClickedButtonConfirmOrder( )
Declare PrintReceipt( )
Declare ClickedButtonHelp( )
END CLASS
formTakeOrders

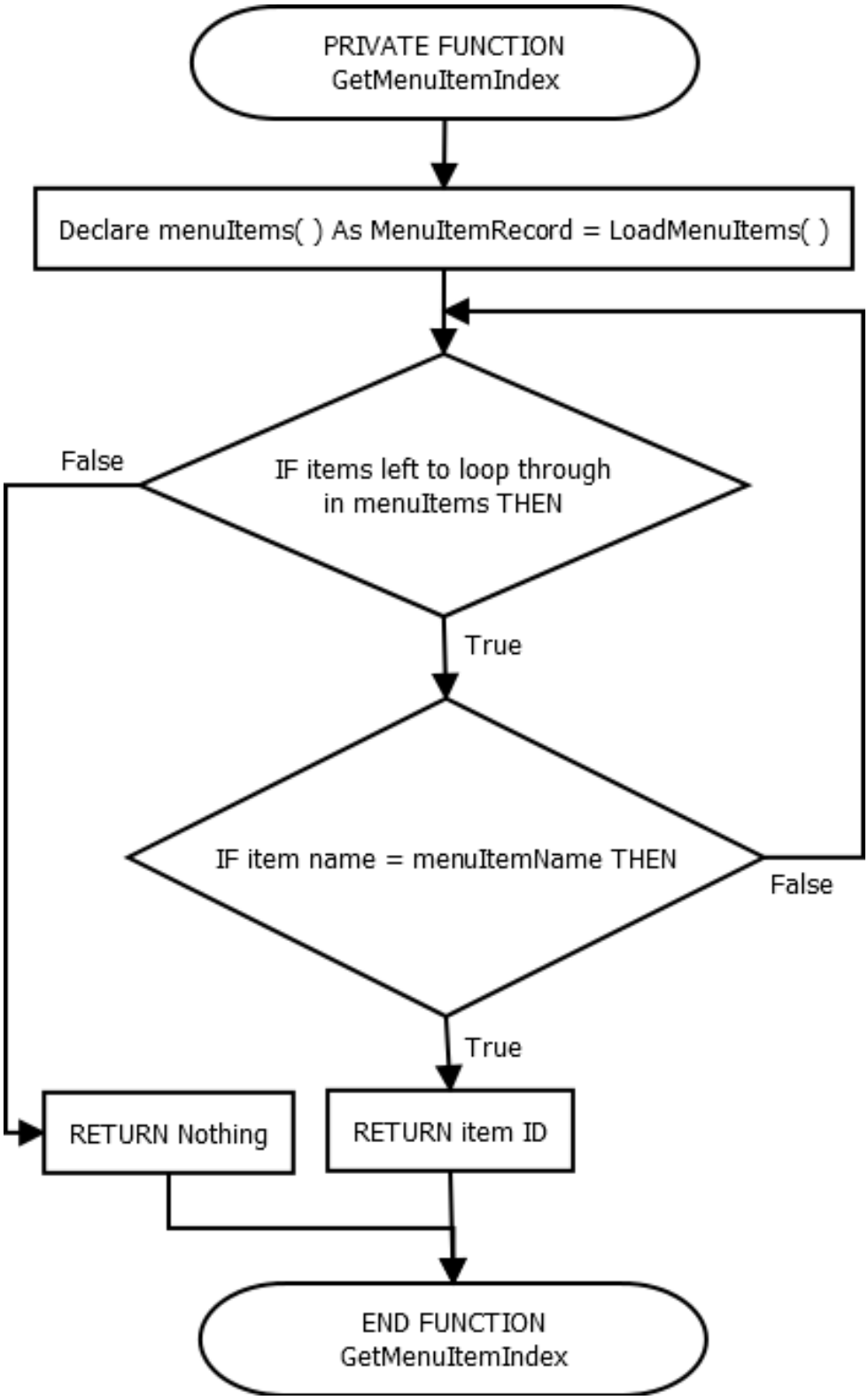

PRIVATE SUB
LoadedFormTakeOrders
LoadMenu
END SUB
LoadedFormTakeOrders

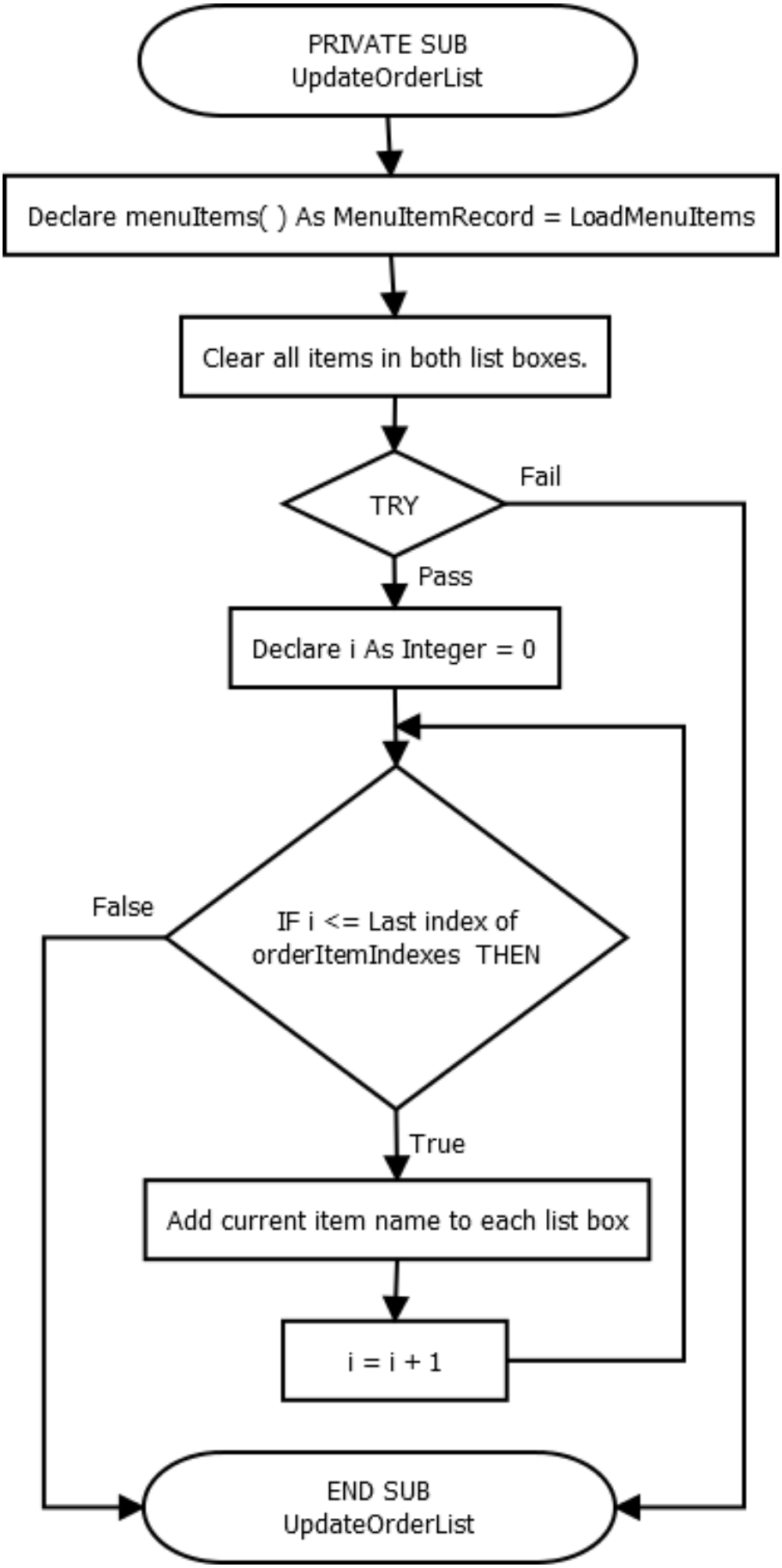

PRIVATE SUB
ChangedTableNumber
IF sender is checked THEN
False
True
selectedTableNumber = sender tag + 1
END SUB
ChangedTableNumber

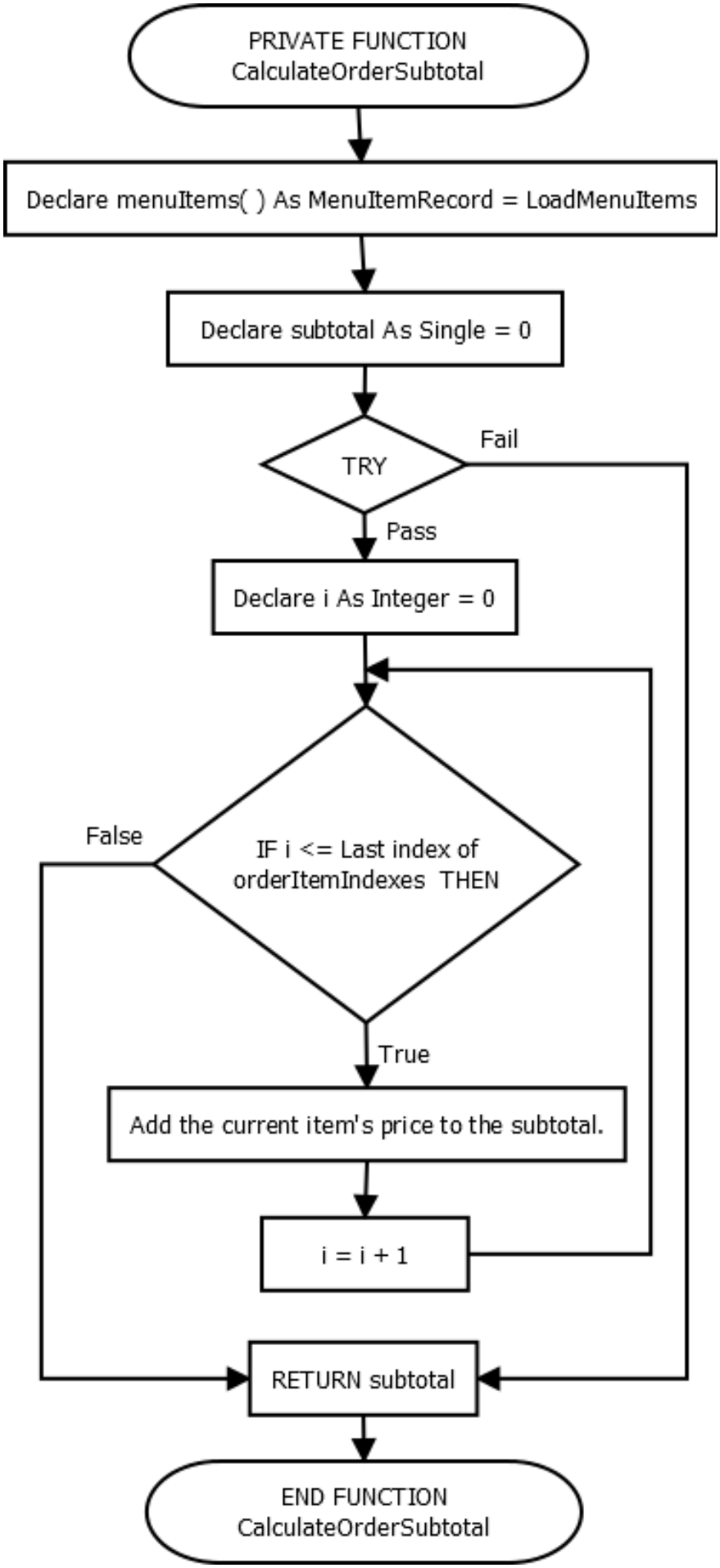

PRIVATE SUB
LoadMenu
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
False
IF items left to loop through
in menuItems THEN
True
IF item is food THEN
True
False
Add the item to the
food combo box.
Add the item to the
beverages combo box.
END SUB
LoadMenu

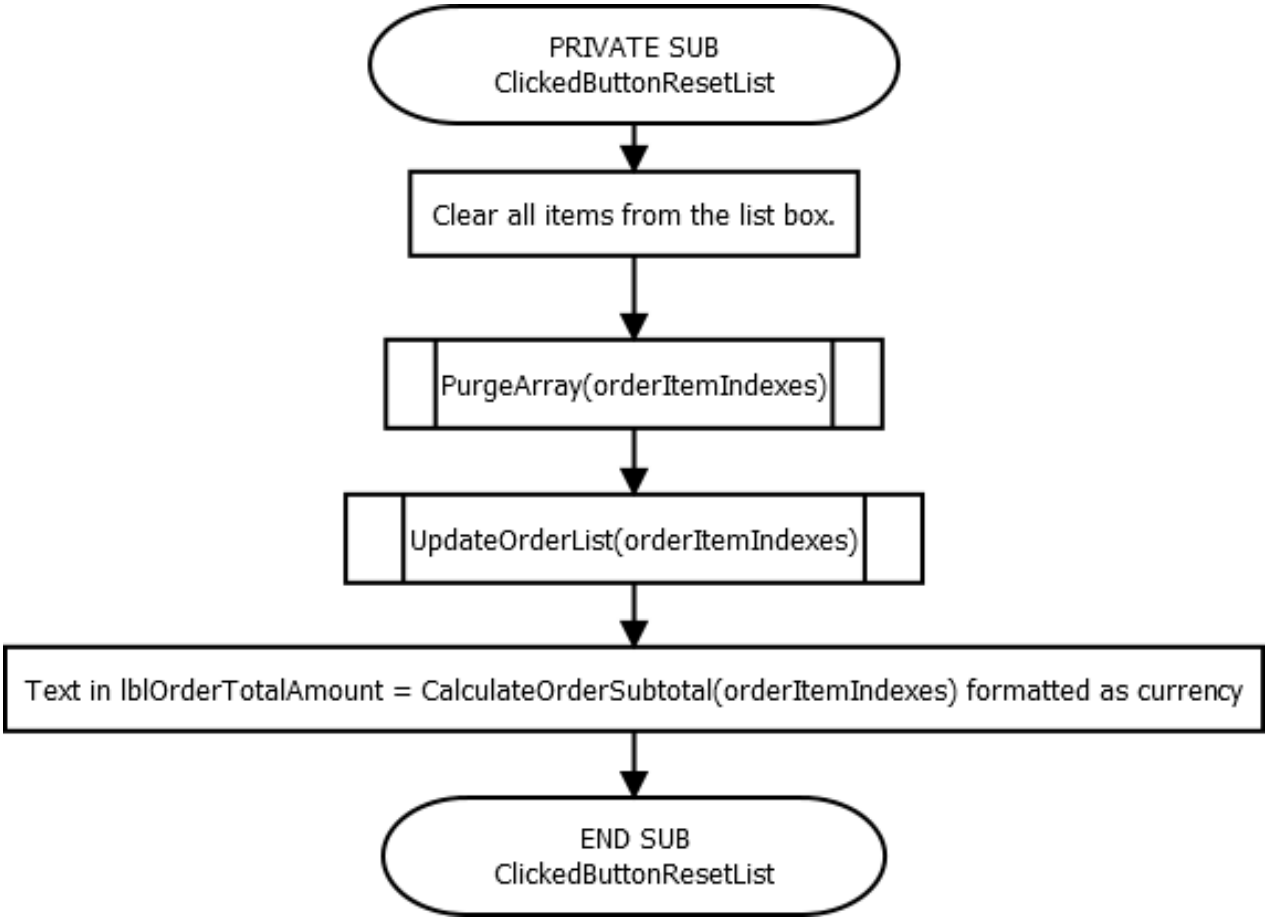

PRIVATE SUB
ClickedButtonAddFood
IF an item is slected in comboFood THEN
False
Display "Please select the food item that you want to add."
True
IF StringIsConvertibleToInteger (Text in txtFoodAmount) THEN
False
True
Declare i As Integer = 0
IF i <= Text in txtFoodAmount converted to integer - 1 THEN
False
True
AddElementToArray(orderItemIndexes, GetMenuItemIndex(Selected food item))
i = i + 1
UpdateOrderList(orderItemIndexes)
Print "Please select an amount of the food you want to add."
Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) in currency format
END SUB
ClickedButtonAddFood

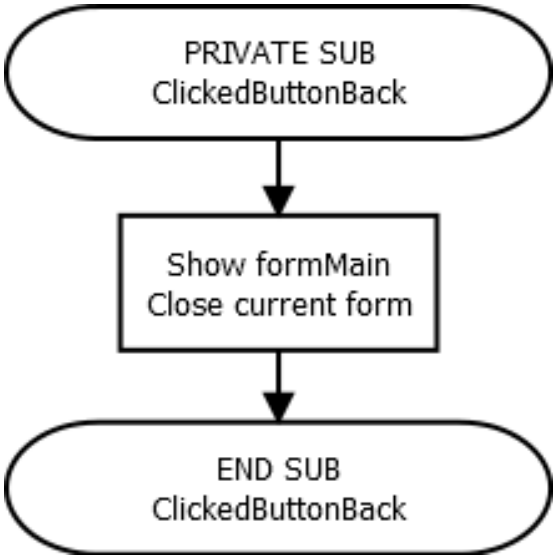

PRIVATE SUB
ClickedButtonAddBeverage
IF an item is slected in comboBeverage THEN
False
True
Display "Please select the beverage that you want to add."
IF StringIsConvertibleToInteger (Text in txtBeverageAmount) THEN
False
True
Declare i As Integer = 0
IF i <= Text in txtBeverageAmount converted to integer - 1 THEN
False
True
AddElementToArray(orderItemIndexes, GetMenuItemIndex(Selected food item))
i = i + 1
UpdateOrderList(orderItemIndexes)
Print "Please select an amount of the the beverage you want to add."
Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) in currency format
END SUB
ClickedButtonAddBeverage

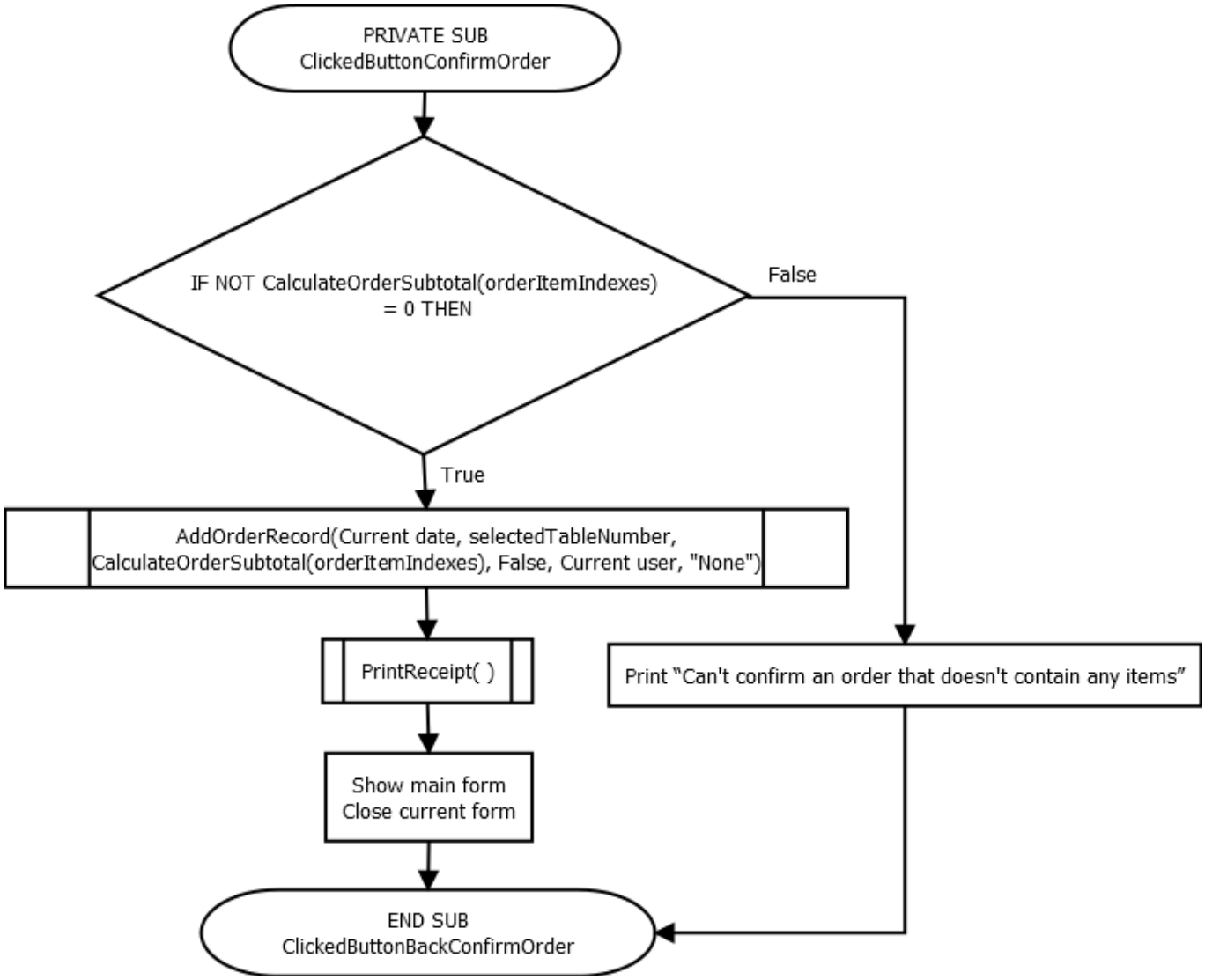

PRIVATE FUNCTION
GetMenuItemIndex
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
IF items left to loop through
in menuItems THEN
False
True
IF item name = menuItemName THEN
False
True
RETURN Nothing
RETURN item ID
END FUNCTION
GetMenuItemIndex

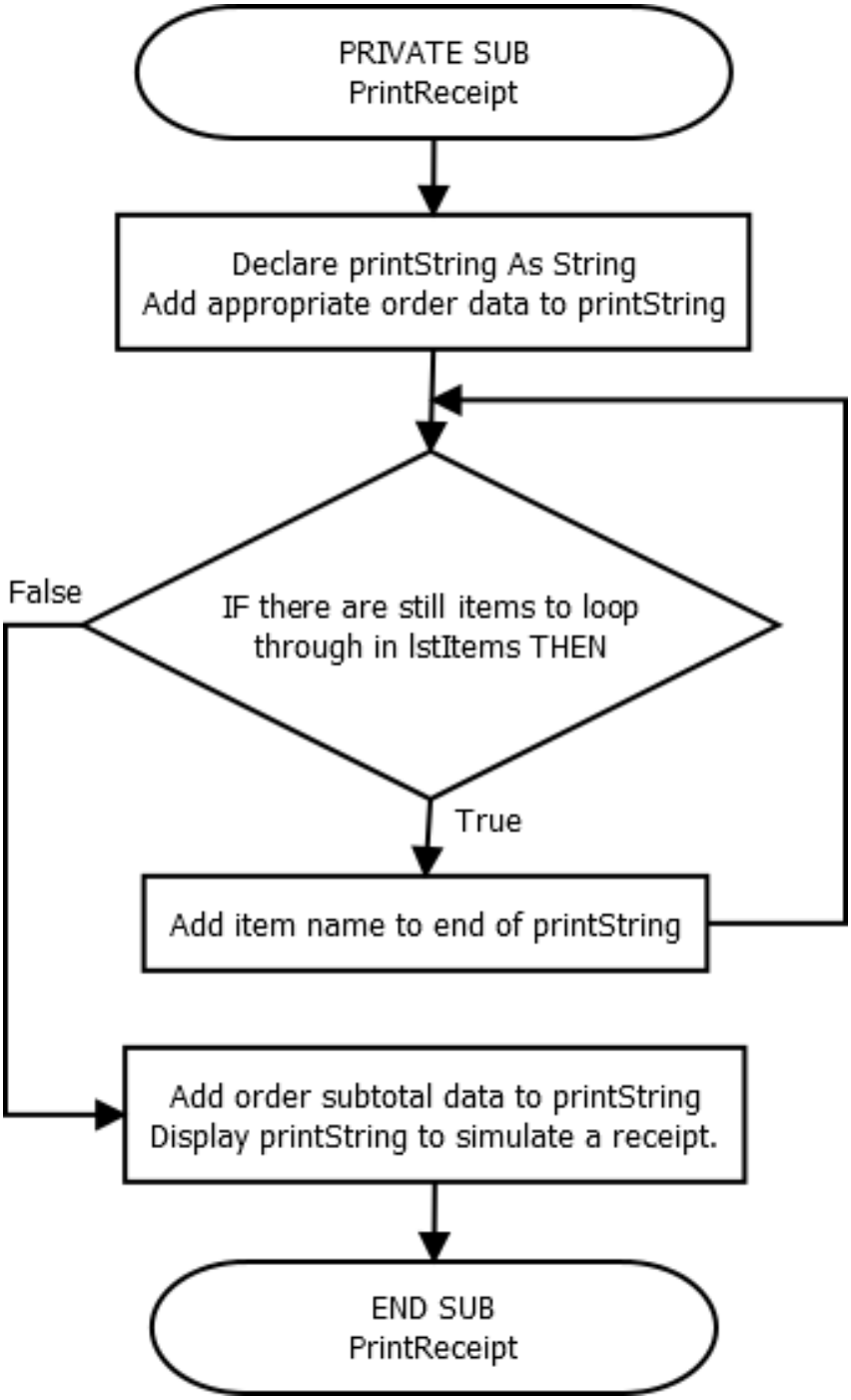

PRIVATE SUB
UpdateOrderList
Declare menuItems( ) As MenuItemRecord = LoadMenuItems
Clear all items in both list boxes.
TRY
Fail
Pass
Declare i As Integer = 0
IF i <= Last index of
orderItemIndexes THEN
False
True
Add current item name to each list box
i = i + 1
END SUB
UpdateOrderList

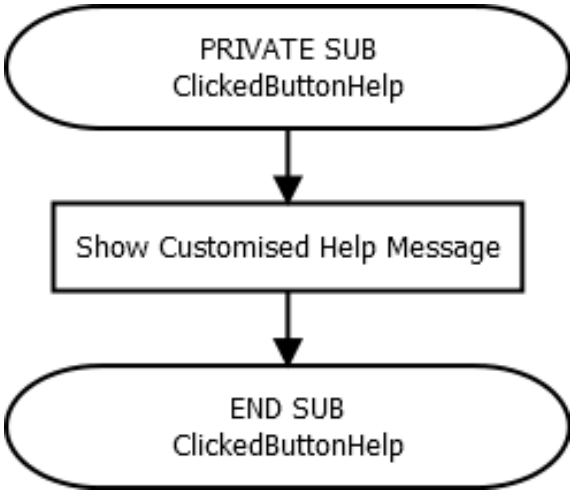

PRIVATE FUNCTION
CalculateOrderSubtotal
Declare menuItems( ) As MenuItemRecord = LoadMenuItems
Declare subtotal As Single = 0
TRY
Fail
Pass
Declare i As Integer = 0
IF i <= Last index of
orderItemIndexes THEN
False
True
Add the current item's price to the subtotal.
i = i + 1
RETURN subtotal
END FUNCTION
CalculateOrderSubtotal


PRIVATE SUB
ClickedButtonResetList
Clear all items from the list box.
PurgeArray(orderItemIndexes)
UpdateOrderList(orderItemIndexes)
Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) formatted as currency
END SUB
ClickedButtonResetList

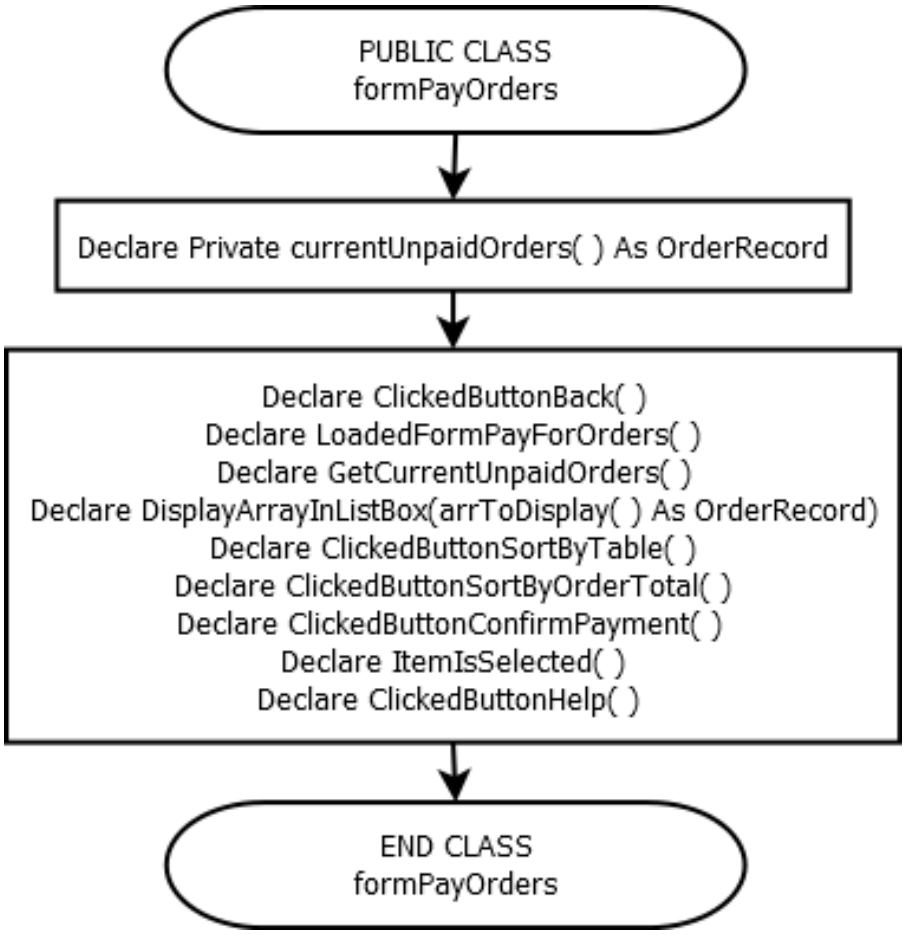

PRIVATE SUB
ClickedButtonBack
Show formMain
Close current form
END SUB
ClickedButtonBack

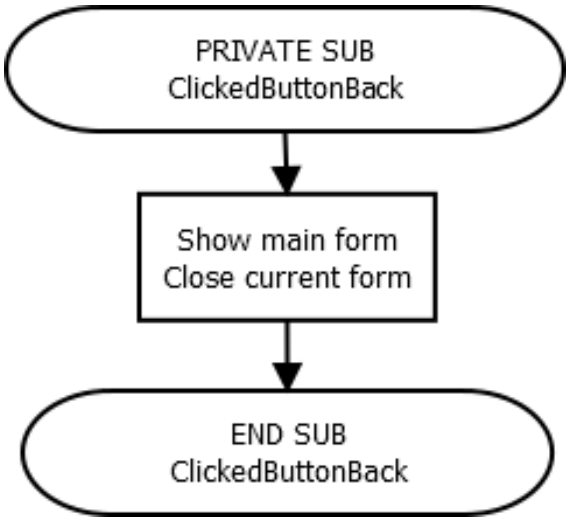

PRIVATE SUB
ClickedButtonConfirmOrder
IF NOT CalculateOrderSubtotal(orderItemIndexes)
= 0 THEN
False
True
AddOrderRecord(Current date, selectedTableNumber,
CalculateOrderSubtotal(orderItemIndexes), False, Current user, "None")
PrintReceipt( )
Print "Can't confirm an order that doesn't contain any items"
Show main form
Close current form
END SUB
ClickedButtonBackConfirmOrder

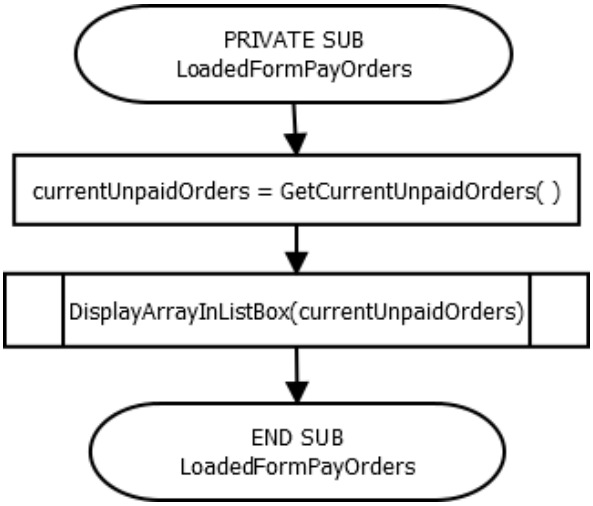

PRIVATE SUB
PrintReceipt
Declare printString As String
Add appropriate order data to printString
False
IF there are still items to loop
through in lstItems THEN
True
Add item name to end of printString
Add order subtotal data to printString
Display printString to simulate a receipt.
END SUB
PrintReceipt

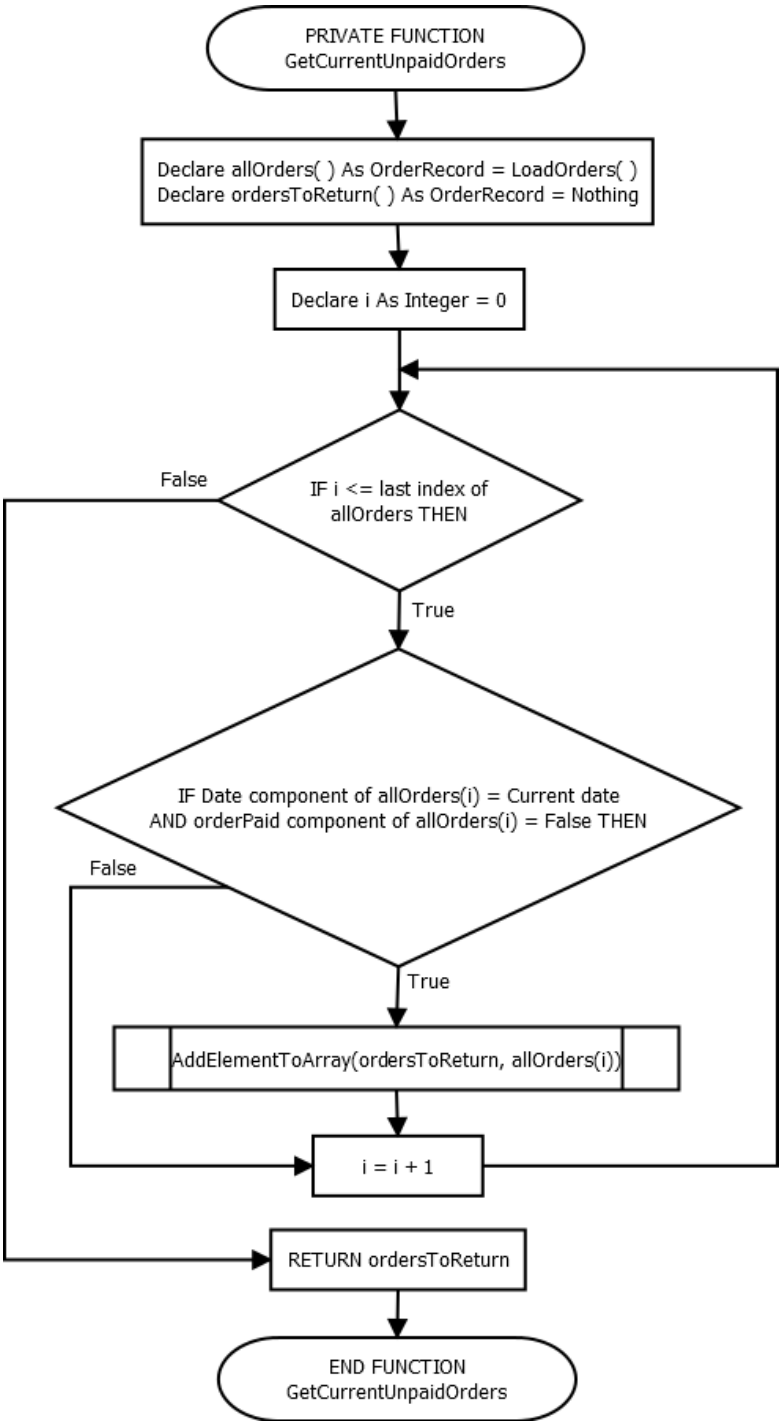

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

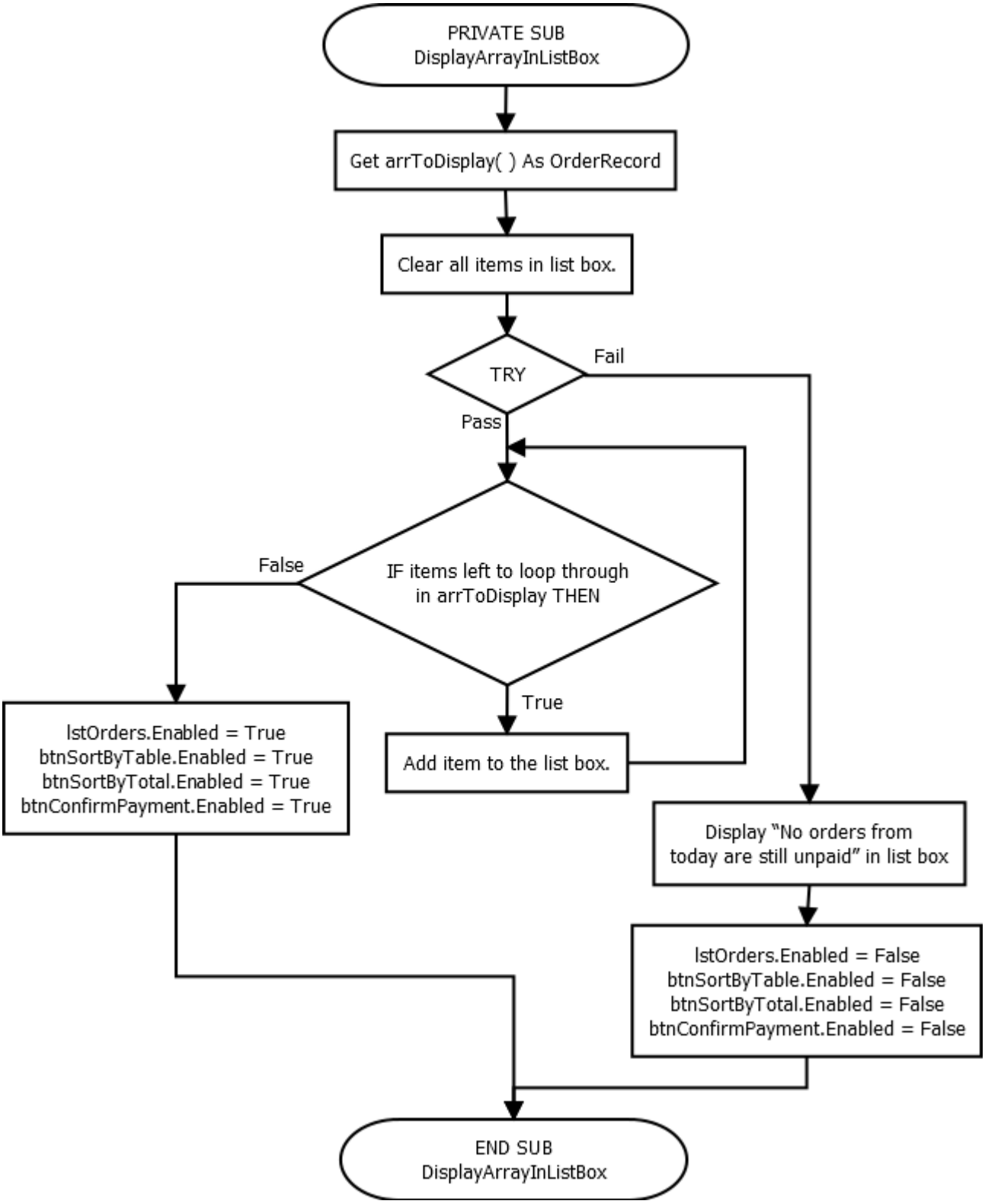
PAY FOR ORDERS

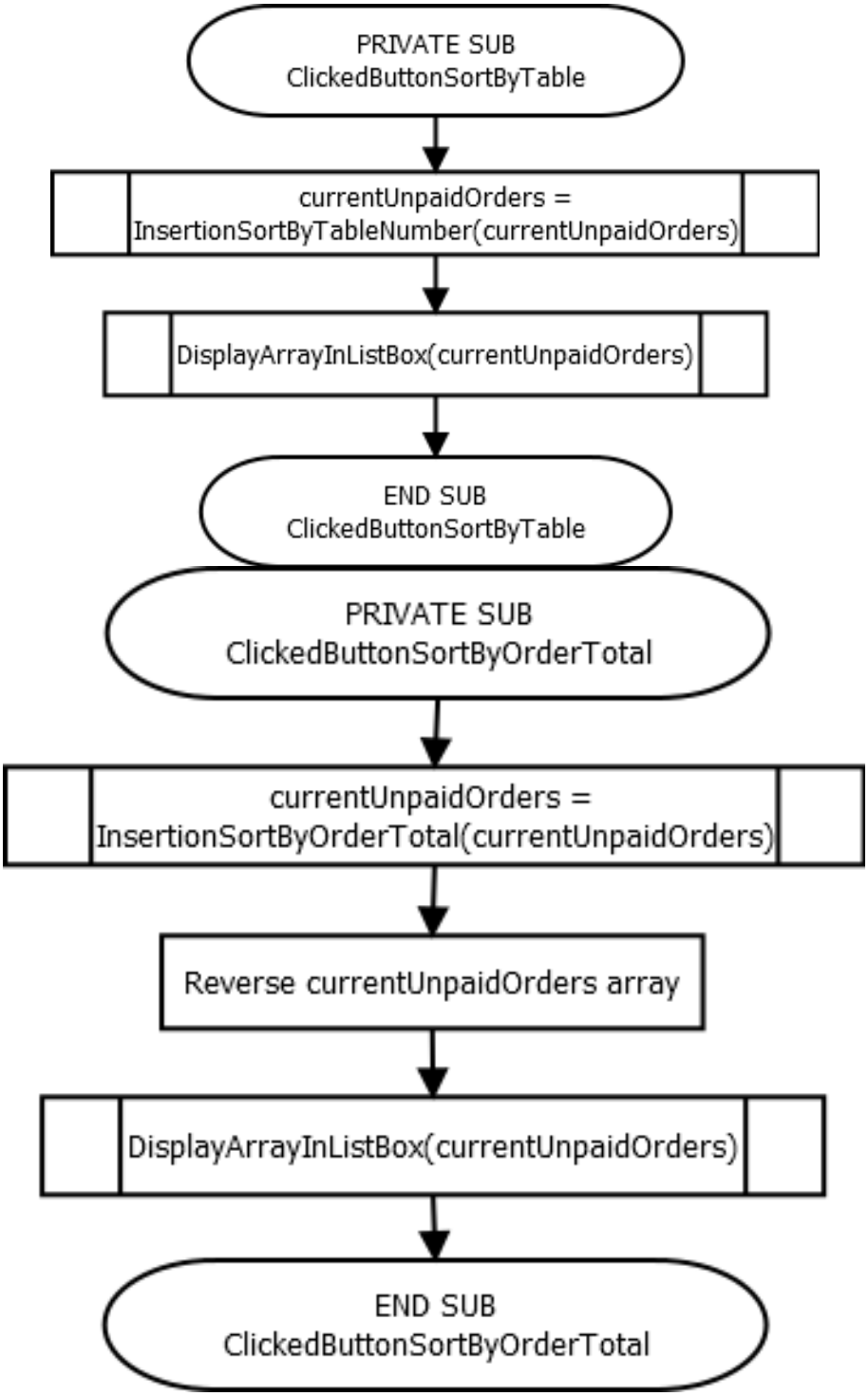


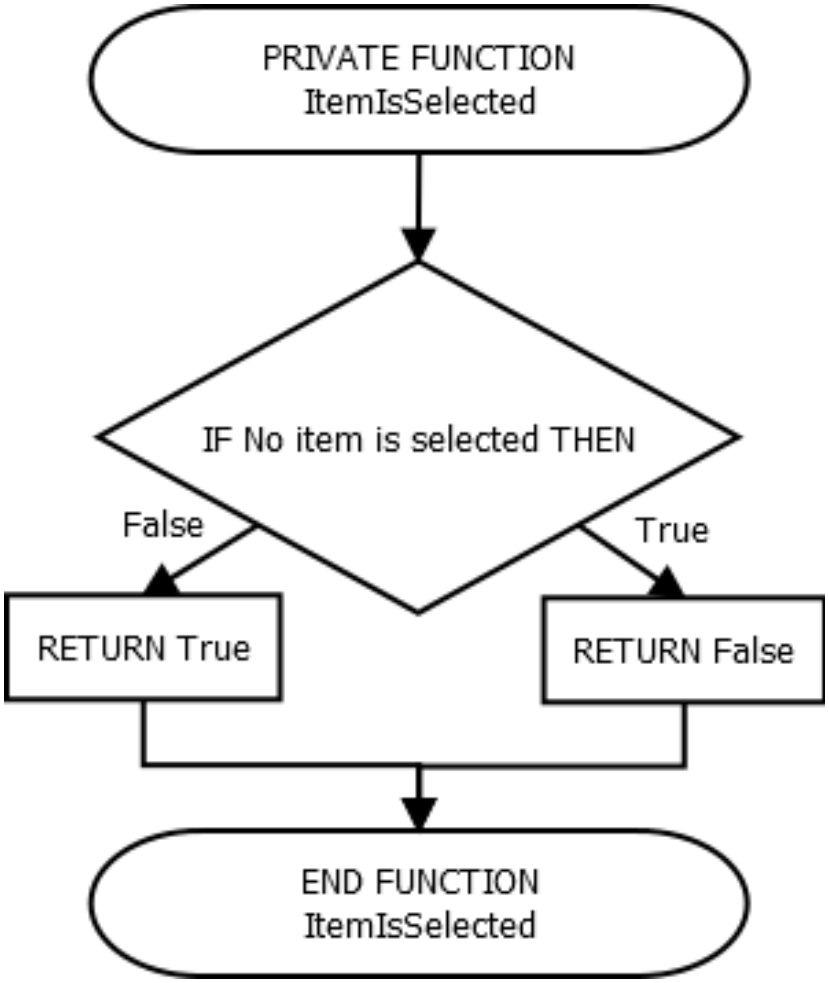


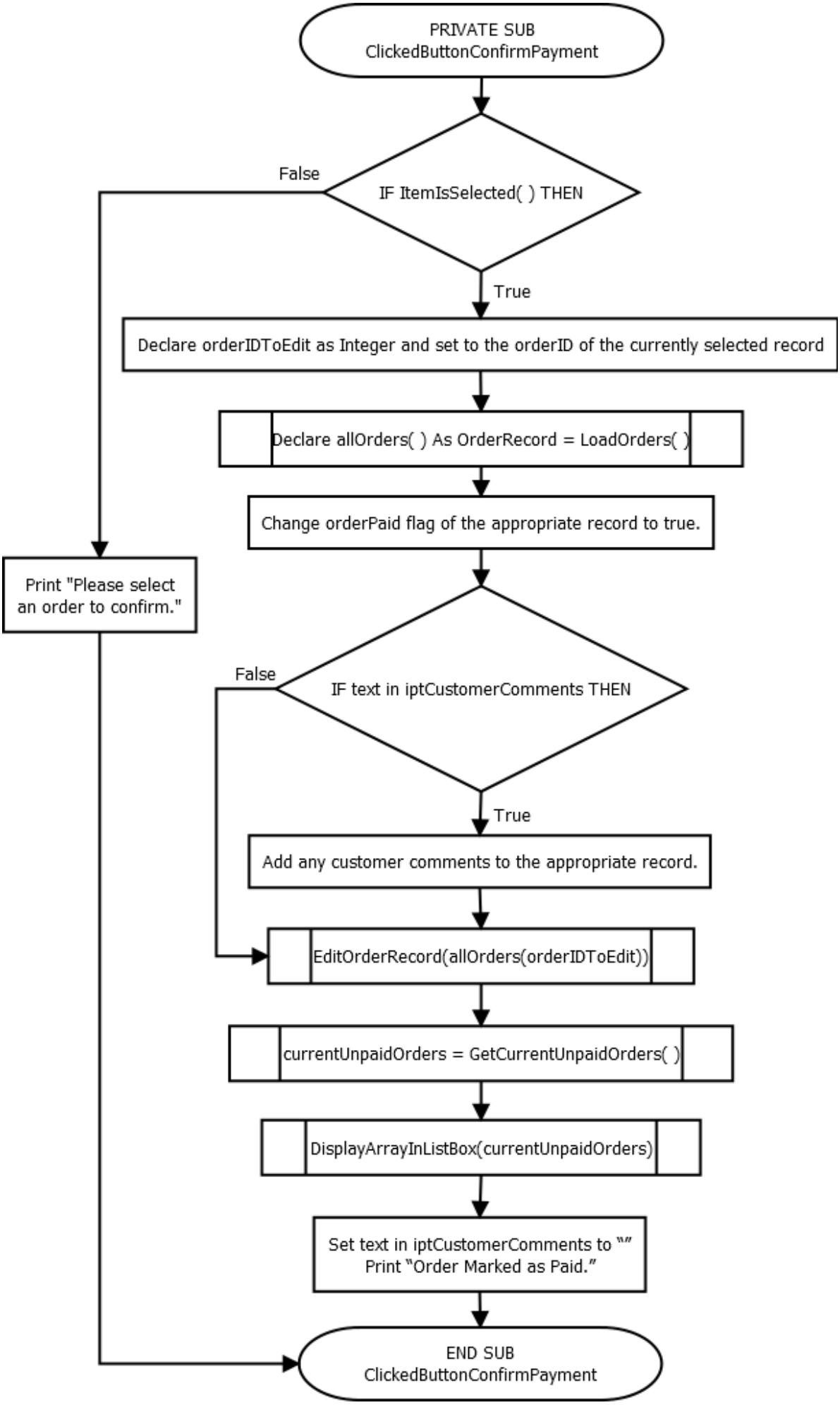

PRIVATE SUB
LoadedFormPayOrders
currentUnpaidOrders = GetCurrentUnpaidOrders( )
DisplayArrayInListBox(currentUnpaidOrders)
END SUB
LoadedFormPayOrders

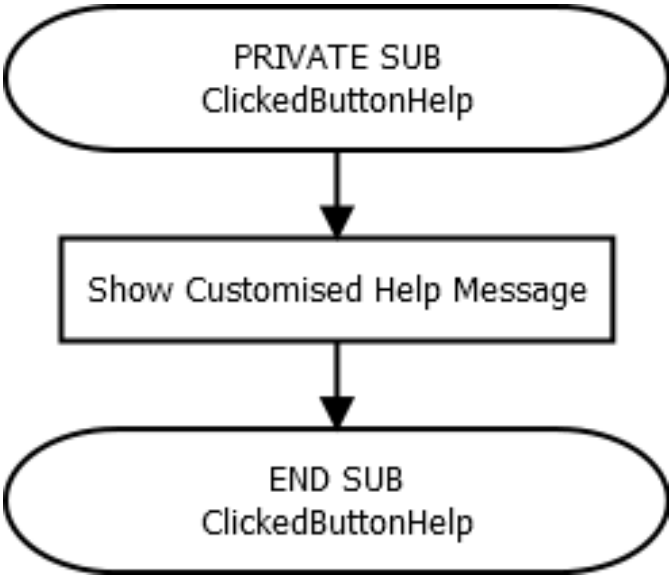

PRIVATE FUNCTION
GetCurrentUnpaidOrders
Declare allOrders( ) As OrderRecord = LoadOrders( )
Declare ordersToReturn( ) As OrderRecord = Nothing
Declare i As Integer = 0
IF i <= last index of
allOrders THEN
False
True
IF Date component of allOrders(i) = Current date
AND orderPaid component of allOrders(i) = False THEN
False
True
AddElementToArray(ordersToReturn, allOrders(i))
i = i + 1
RETURN ordersToReturn
END FUNCTION
GetCurrentUnpaidOrders


PRIVATE SUB
DisplayArrayInListBox
Get arrToDisplay( ) As OrderRecord
Clear all items in list box.
TRY
Fail
Pass
IF items left to loop through
in arrToDisplay THEN
False
True
lstOrders.Enabled = True
btnSortByTable.Enabled = True
btnSortByTotal.Enabled = True
btnConfirmPayment.Enabled = True
Add item to the list box.
Display "No orders from
today are still unpaid" in list box
lstOrders.Enabled = False
btnSortByTable.Enabled = False
btnSortByTotal.Enabled = False
btnConfirmPayment.Enabled = False
END SUB
DisplayArrayInListBox

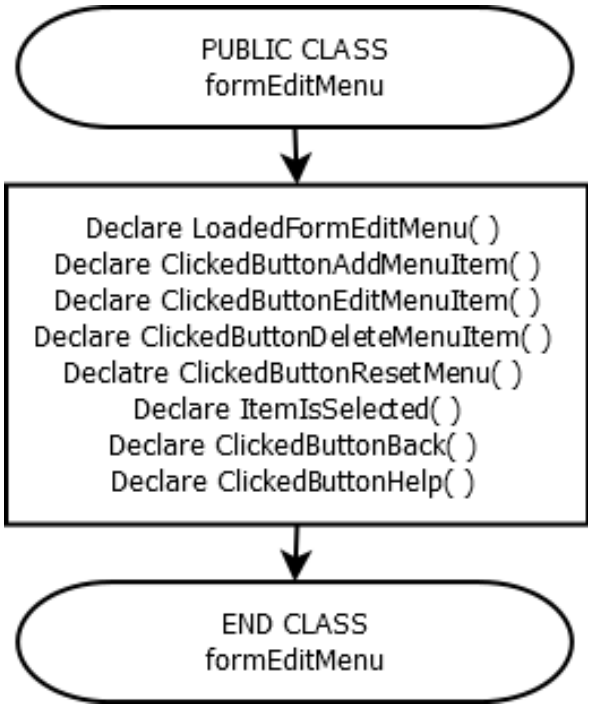

PRIVATE SUB
ClickedButtonSortByTable
currentUnpaidOrders =
InsertionSortByTableNumber(currentUnpaidOrders)
DisplayArrayInListBox(currentUnpaidOrders)
END SUB
ClickedButtonSortByTable
PRIVATE SUB
ClickedButtonSortByOrderTotal
currentUnpaidOrders =
InsertionSortByOrderTotal(currentUnpaidOrders)
Reverse currentUnpaidOrders array
DisplayArrayInListBox(currentUnpaidOrders)
END SUB
ClickedButtonSortByOrderTotal

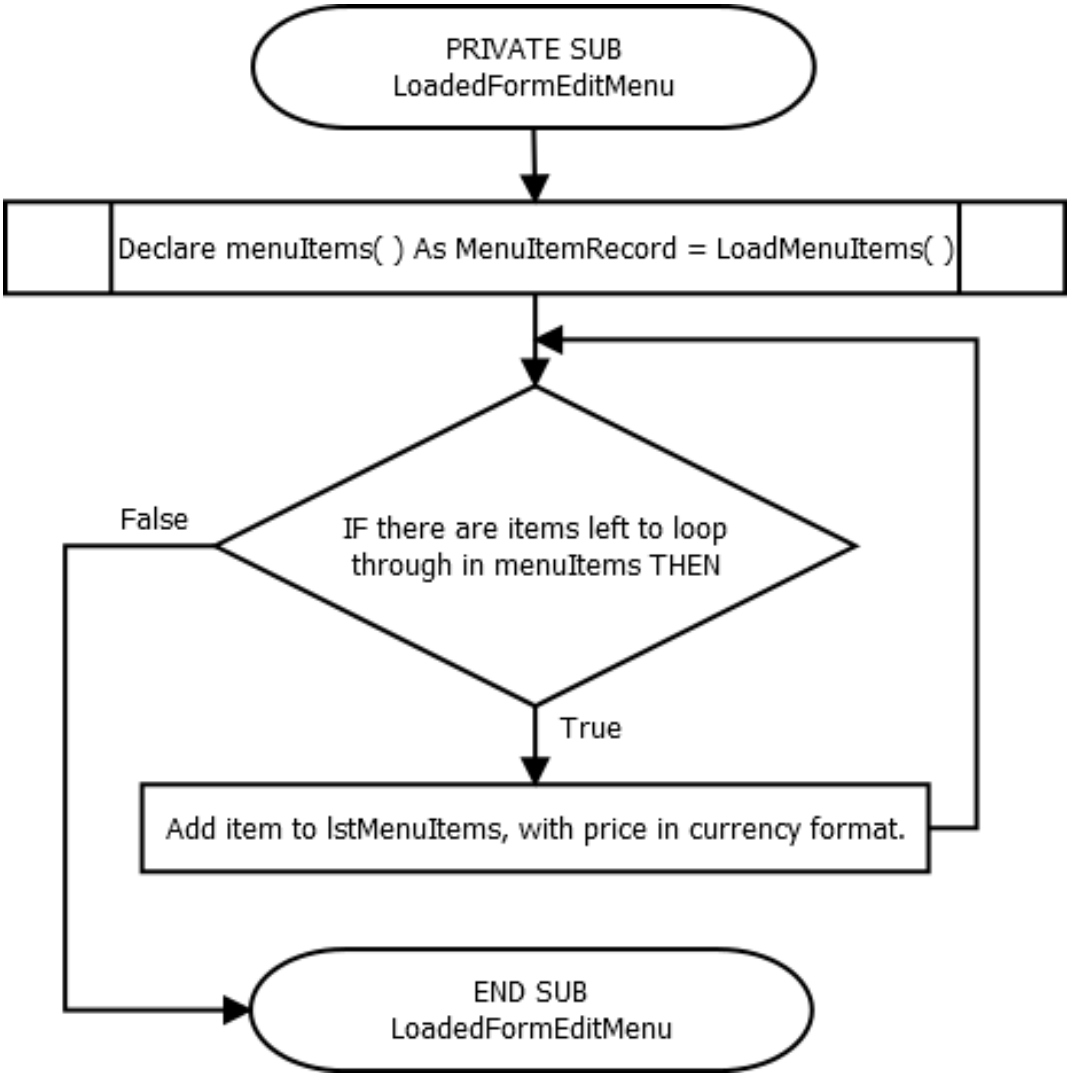

PRIVATE FUNCTION
ItemIsSelected
IF No item is selected THEN
False
True
RETURN True
RETURN False
END FUNCTION
ItemIsSelected

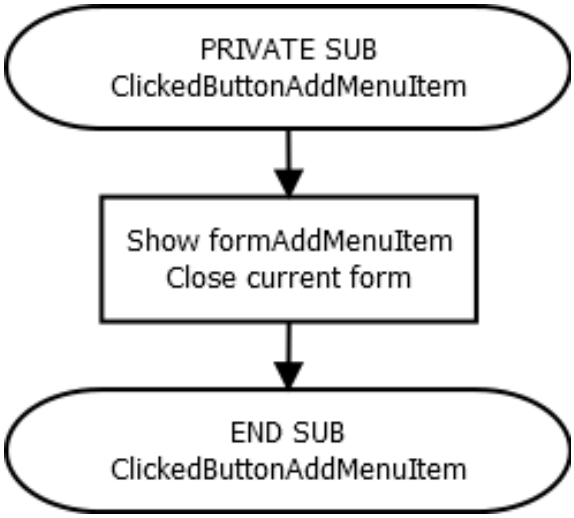

PRIVATE SUB
ClickedButtonConfirmPayment
False
IF ItemIsSelected( ) THEN
True
Declare orderIDToEdit as Integer and set to the orderID of the currently selected record
Declare allOrders( ) As OrderRecord = LoadOrders( )
Change orderPaid flag of the appropriate record to true.
Print "Please select an order to confirm."
False
IF text in iptCustomerComments THEN
True
Add any customer comments to the appropriate record.
EditOrderRecord(allOrders(orderIDToEdit))
currentUnpaidOrders = GetCurrentUnpaidOrders( )
DisplayArrayInListBox(currentUnpaidOrders)
Set text in iptCustomerComments to ""
Print "Order Marked as Paid."
END SUB
ClickedButtonConfirmPayment

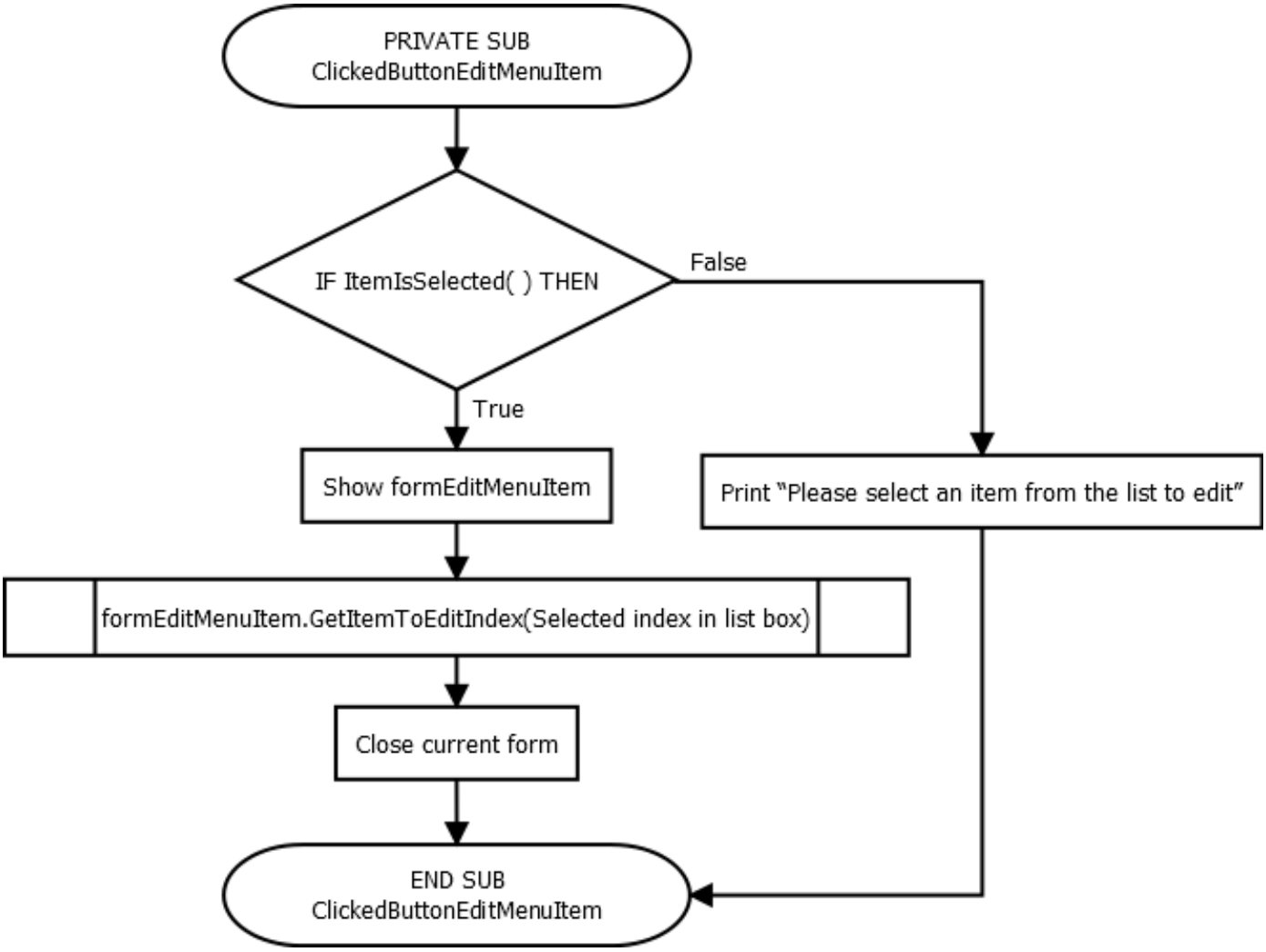

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

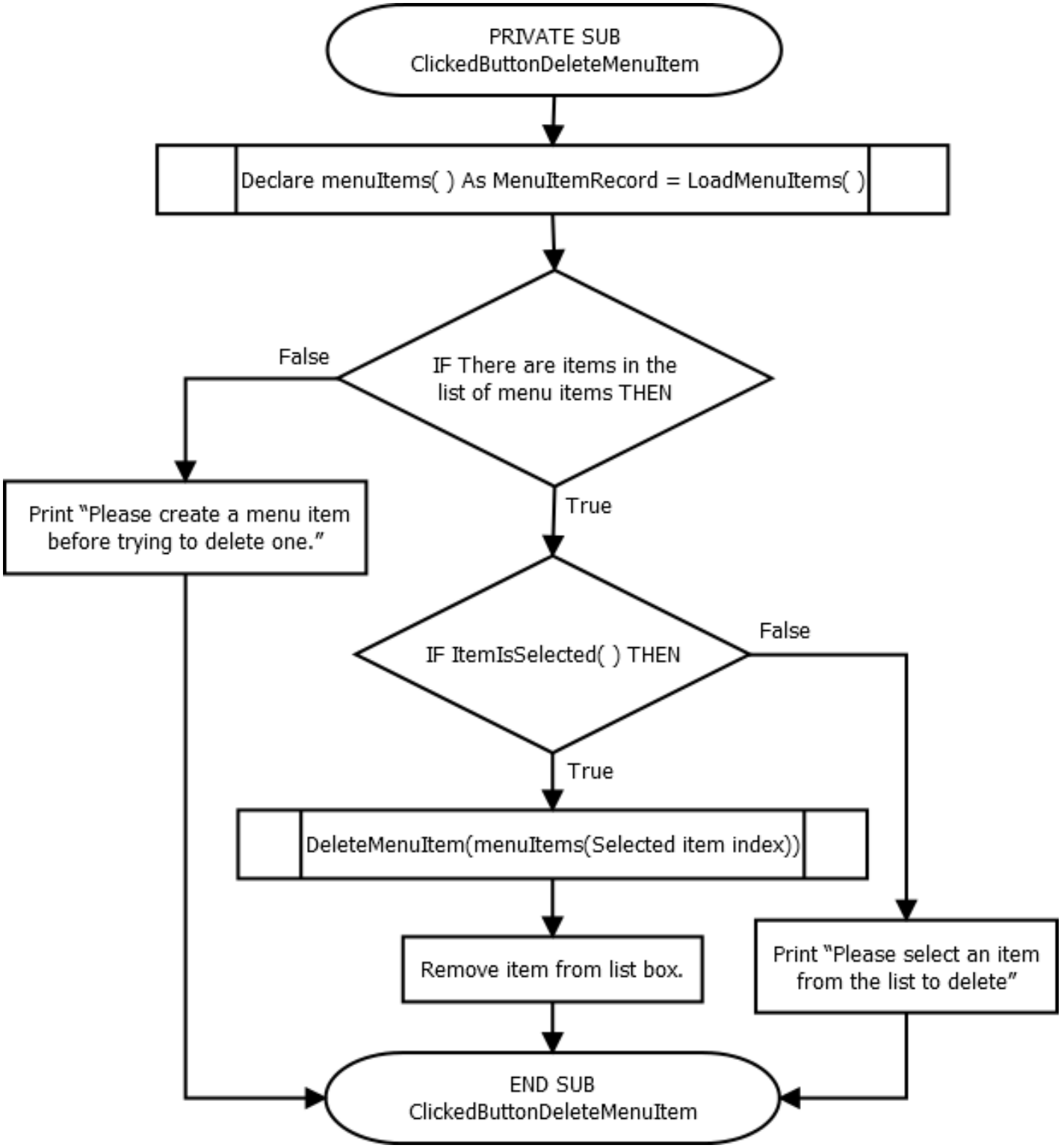
EDIT MENU

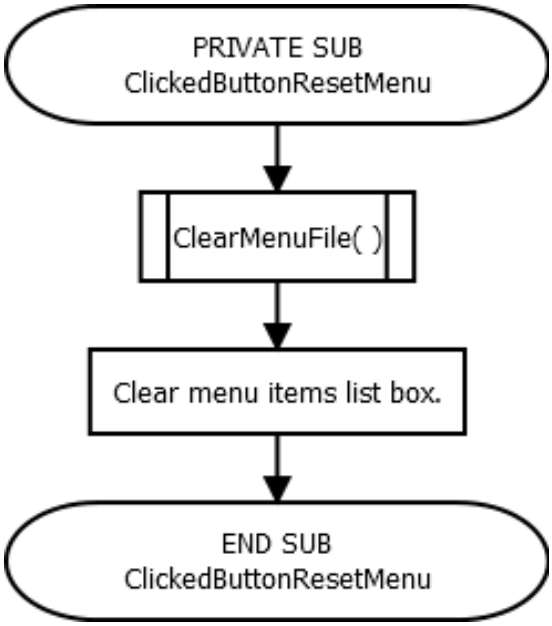

PUBLIC CLASS
formEditMenu
Declare LoadedFormEditMenu( )
Declare ClickedButtonAddMenuItem( )
Declare ClickedButtonEditMenuItem( )
Declare ClickedButtonDeleteMenuItem( )
Declatre ClickedButtonResetMenu( )
Declare ItemIsSelected( )
Declare ClickedButtonBack( )
Declare ClickedButtonHelp( )
END CLASS
formEditMenu

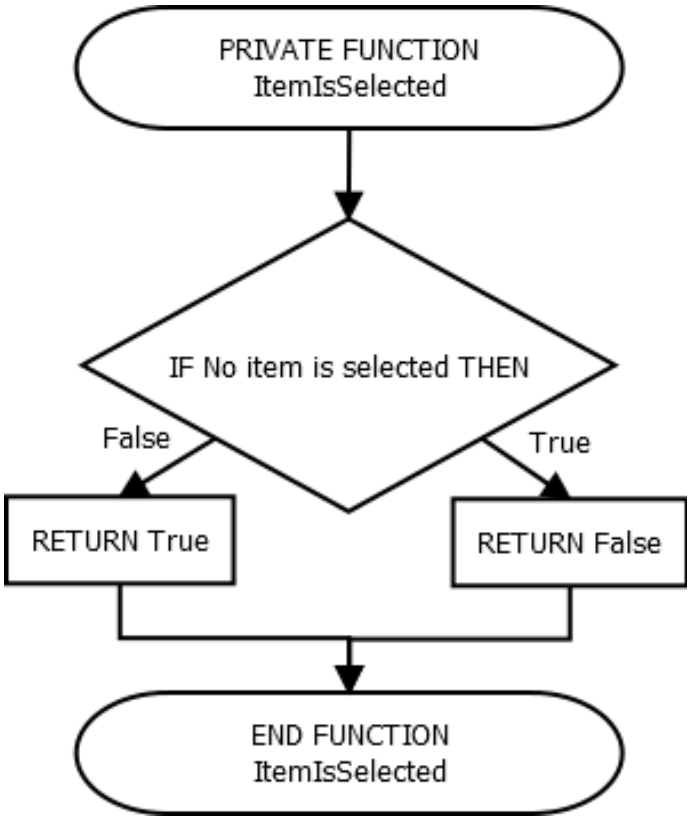

PRIVATE SUB
LoadedFormEditMenu
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
IF there are items left to loop through in menuItems THEN
False
True
Add item to lstMenuItems, with price in currency format.
END SUB
LoadedFormEditMenu

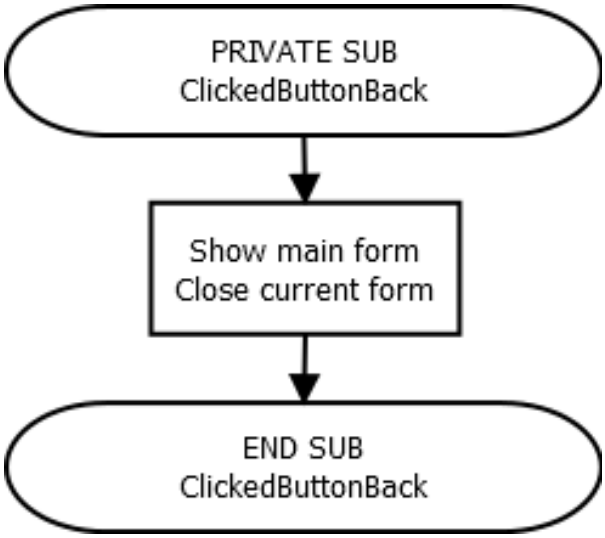

PRIVATE SUB
ClickedButtonAddMenuItem
Show formAddMenuItem
Close current form
END SUB
ClickedButtonAddMenuItem

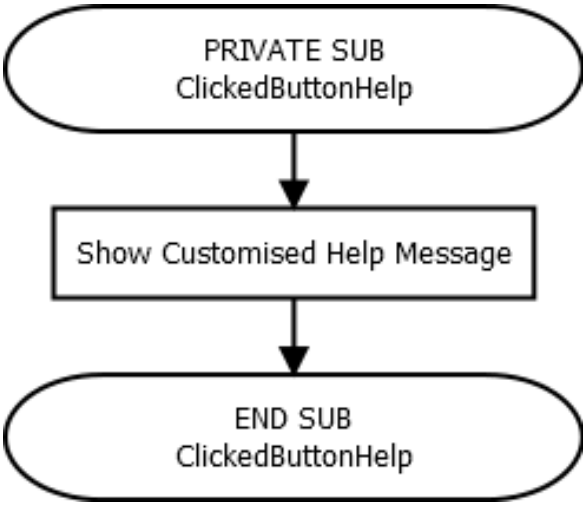

PRIVATE SUB
ClickedButtonEditMenuItem
IF ItemIsSelected( ) THEN
False
True
Show formEditMenuItem
Print "Please select an item from the list to edit"
formEditMenuItem.GetItemToEditIndex(Selected index in list box)
Close current form
END SUB
ClickedButtonEditMenuItem


PRIVATE SUB
ClickedButtonDeleteMenuItem
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
IF There are items in the
list of menu items THEN
False
True
Print "Please create a menu item
before trying to delete one."
IF ItemIsSelected( ) THEN
False
True
DeleteMenuItem(menuItems(Selected item index))
Remove item from list box.
Print "Please select an item
from the list to delete"
END SUB
ClickedButtonDeleteMenuItem

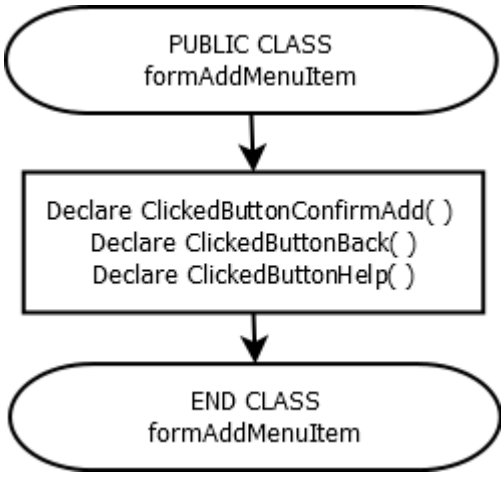

PRIVATE SUB
ClickedButtonResetMenu
ClearMenuFile( )
Clear menu items list box.
END SUB
ClickedButtonResetMenu

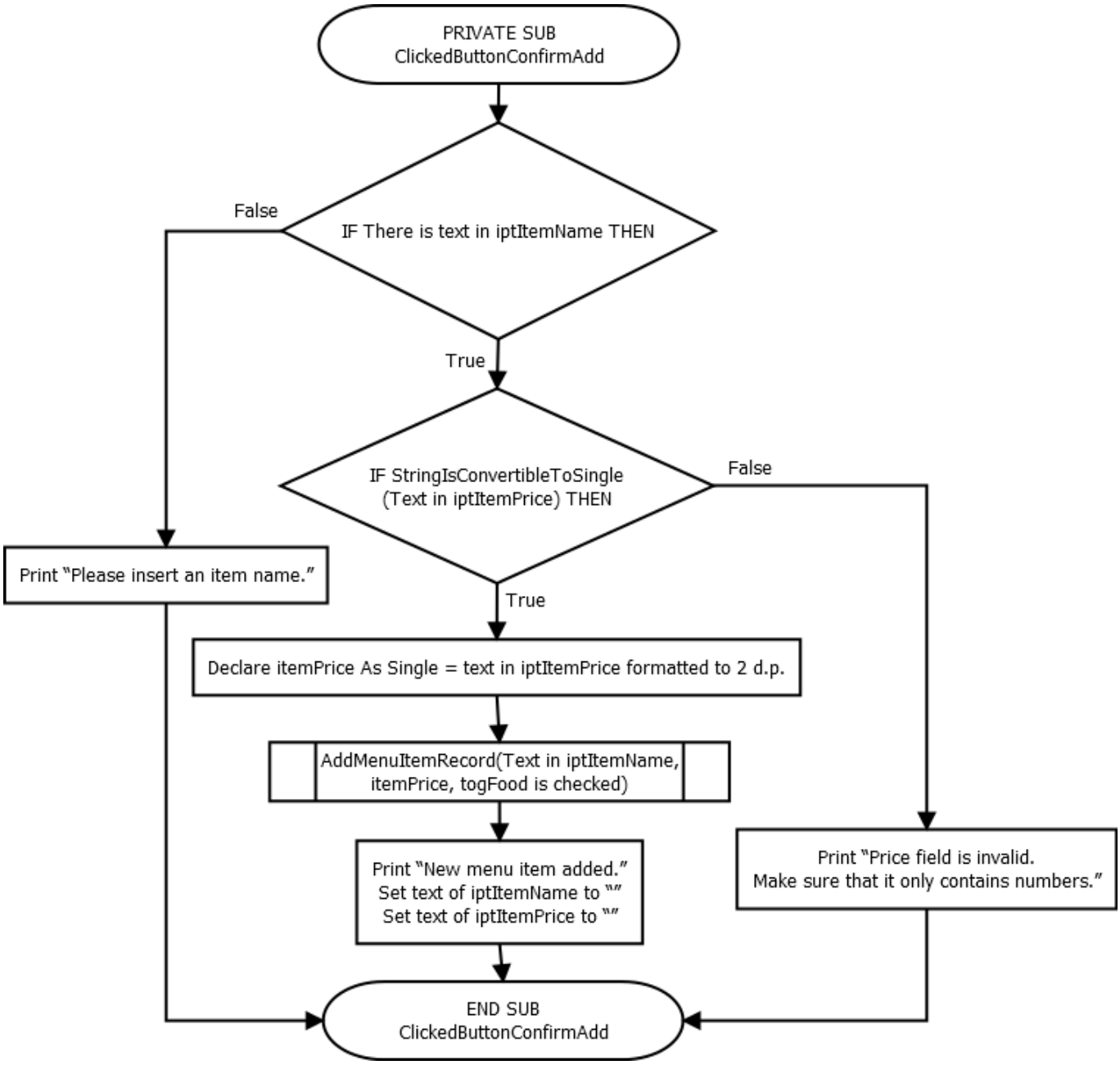

PRIVATE FUNCTION
ItemIsSelected
IF No item is selected THEN
False
True
RETURN True
RETURN False
END FUNCTION
ItemIsSelected

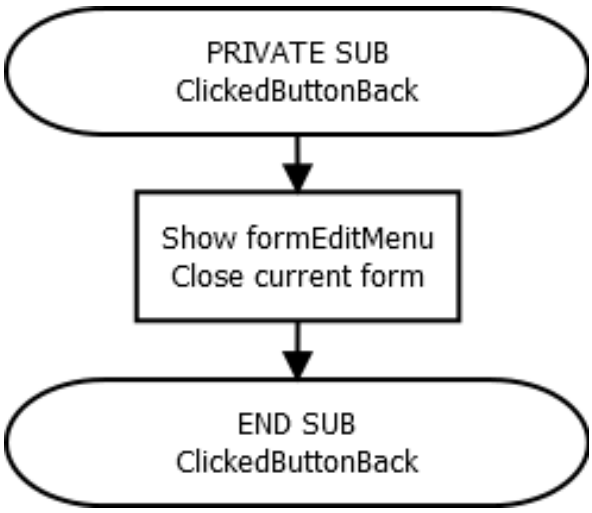

PRIVATE SUB
ClickedButtonBack
Show main form
Close current form
END SUB
ClickedButtonBack

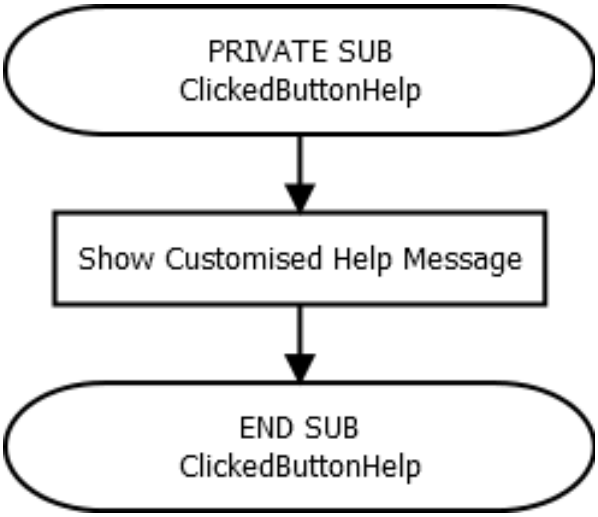

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

ADD MENU ITEM

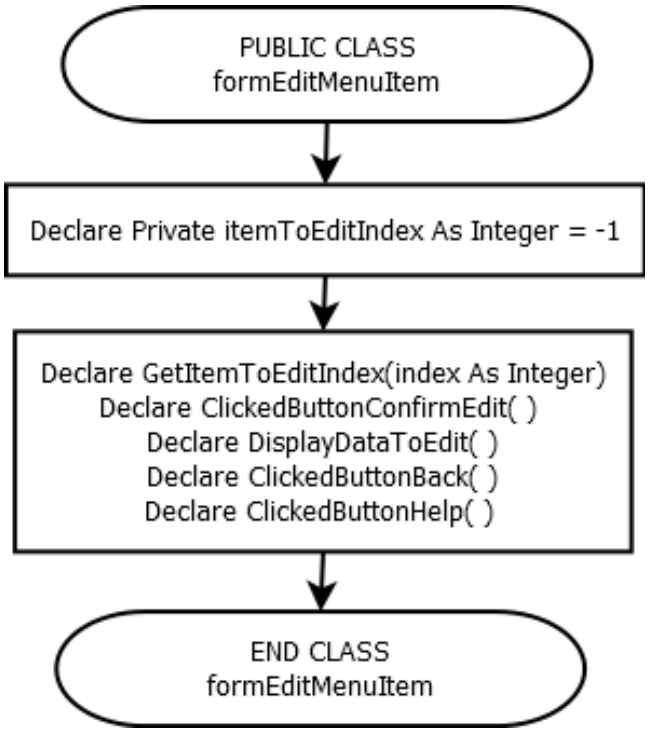


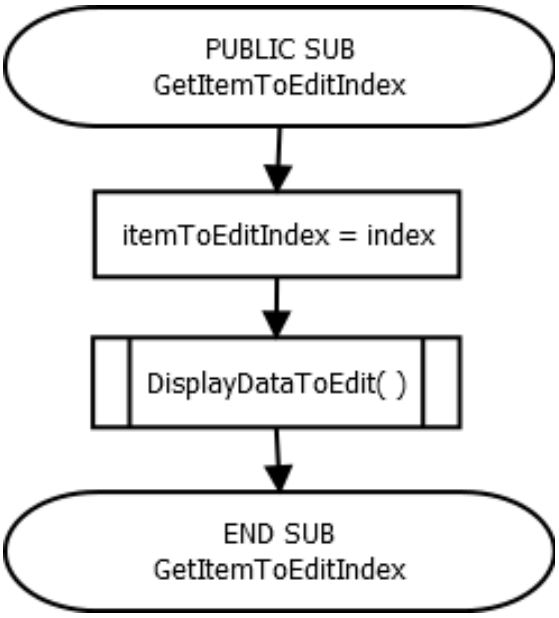


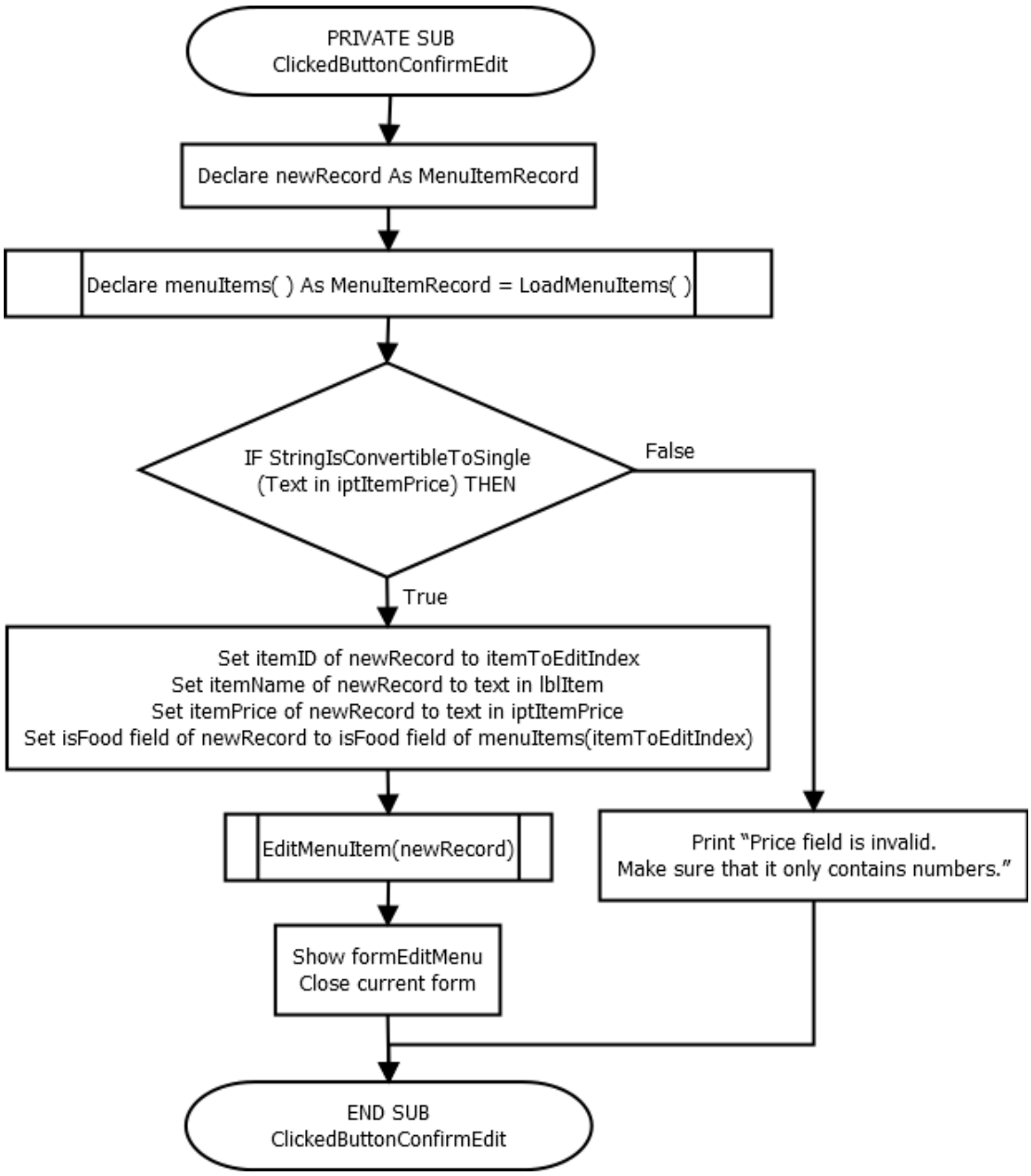

PRIVATE SUB
ClickedButtonBack
Show formEditMenu
Close current form
END SUB
ClickedButtonBack

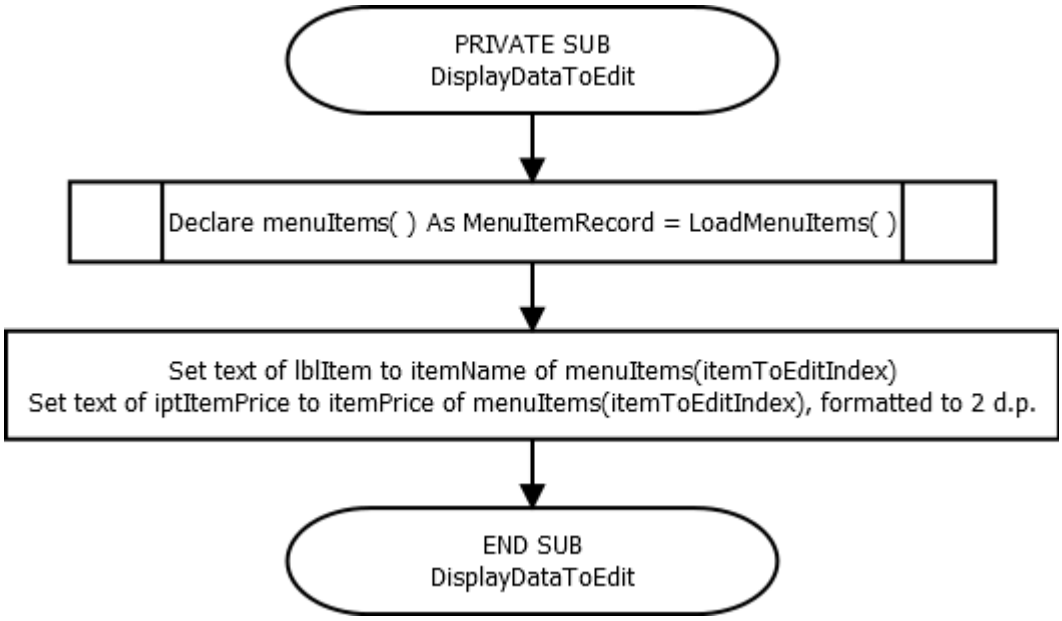

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

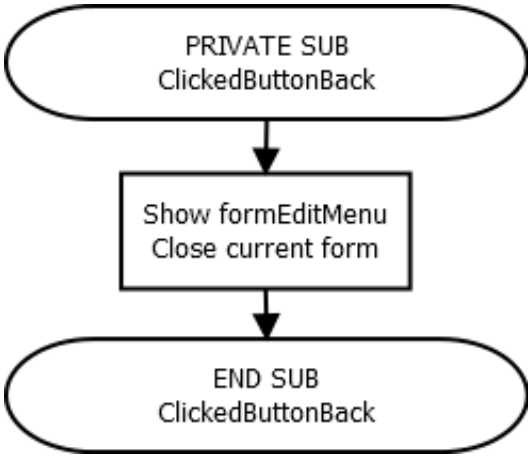
EDIT MENU ITEM

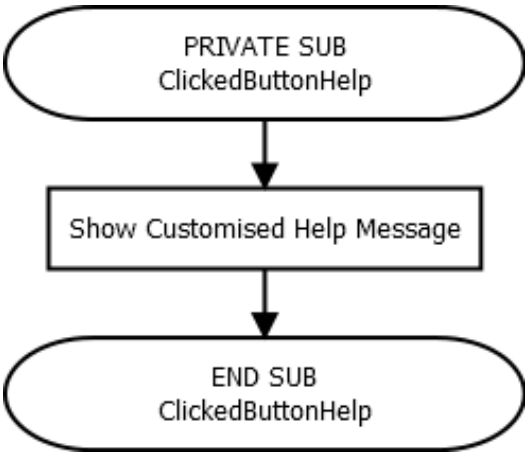




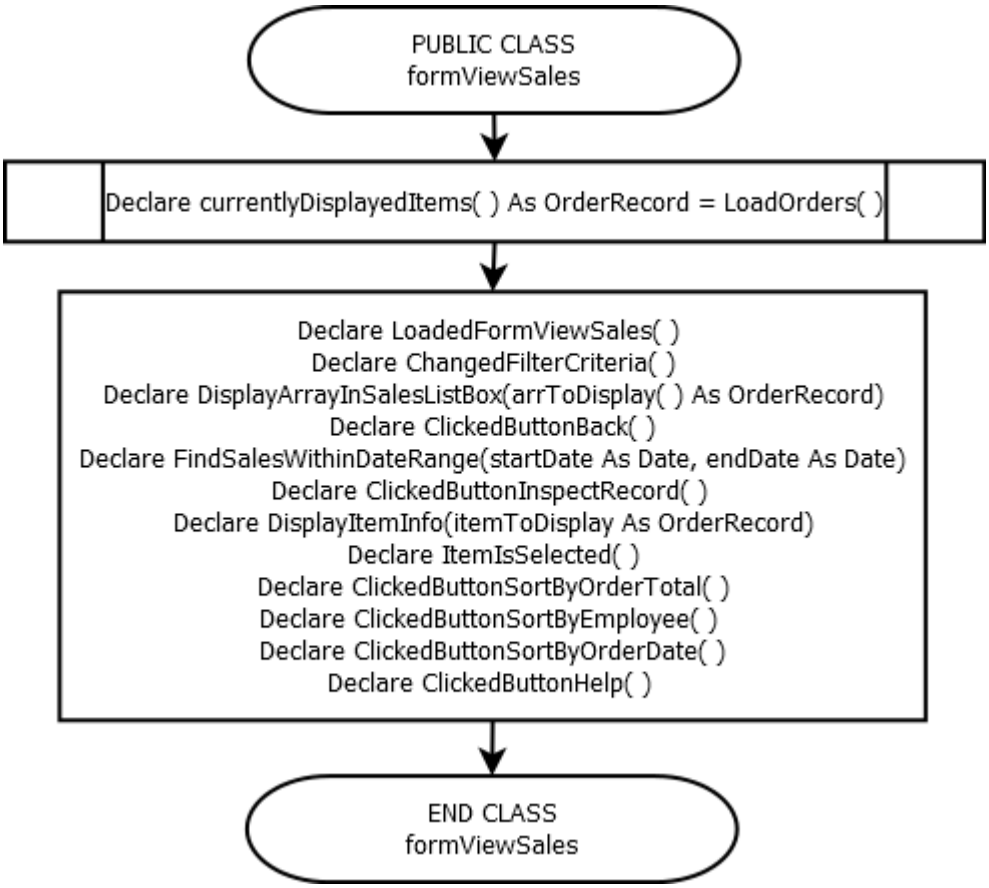

PRIVATE SUB
ClickedButtonConfirmEdit
Declare newRecord As MenuItemRecord
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
IF StringIsConvertibleToSingle
(Text in iptItemPrice) THEN
False
True
Set itemID of newRecord to itemToEditIndex
Set itemName of newRecord to text in lblItem
Set itemPrice of newRecord to text in iptItemPrice
Set isFood field of newRecord to isFood field of menuItems(itemToEditIndex)
EditMenuItem(newRecord)
Print "Price field is invalid.
Make sure that it only contains numbers."
Show formEditMenu
Close current form
END SUB
ClickedButtonConfirmEdit

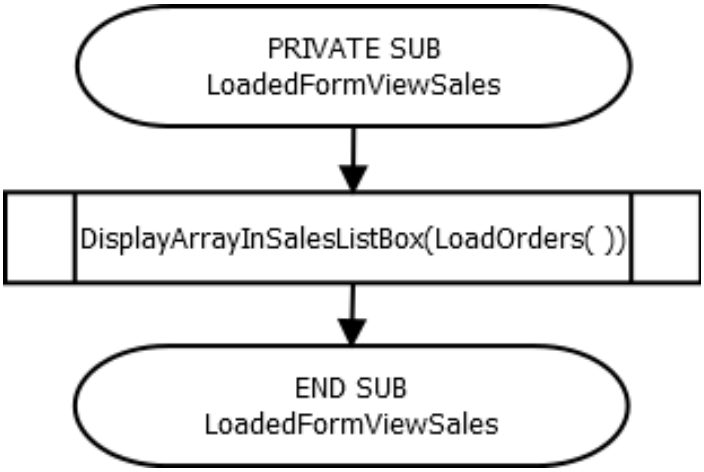

PRIVATE SUB
DisplayDataToEdit
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
Set text of lblItem to itemName of menuItems(itemToEditIndex)
Set text of iptItemPrice to itemPrice of menuItems(itemToEditIndex), formatted to 2 d.p.
END SUB
DisplayDataToEdit

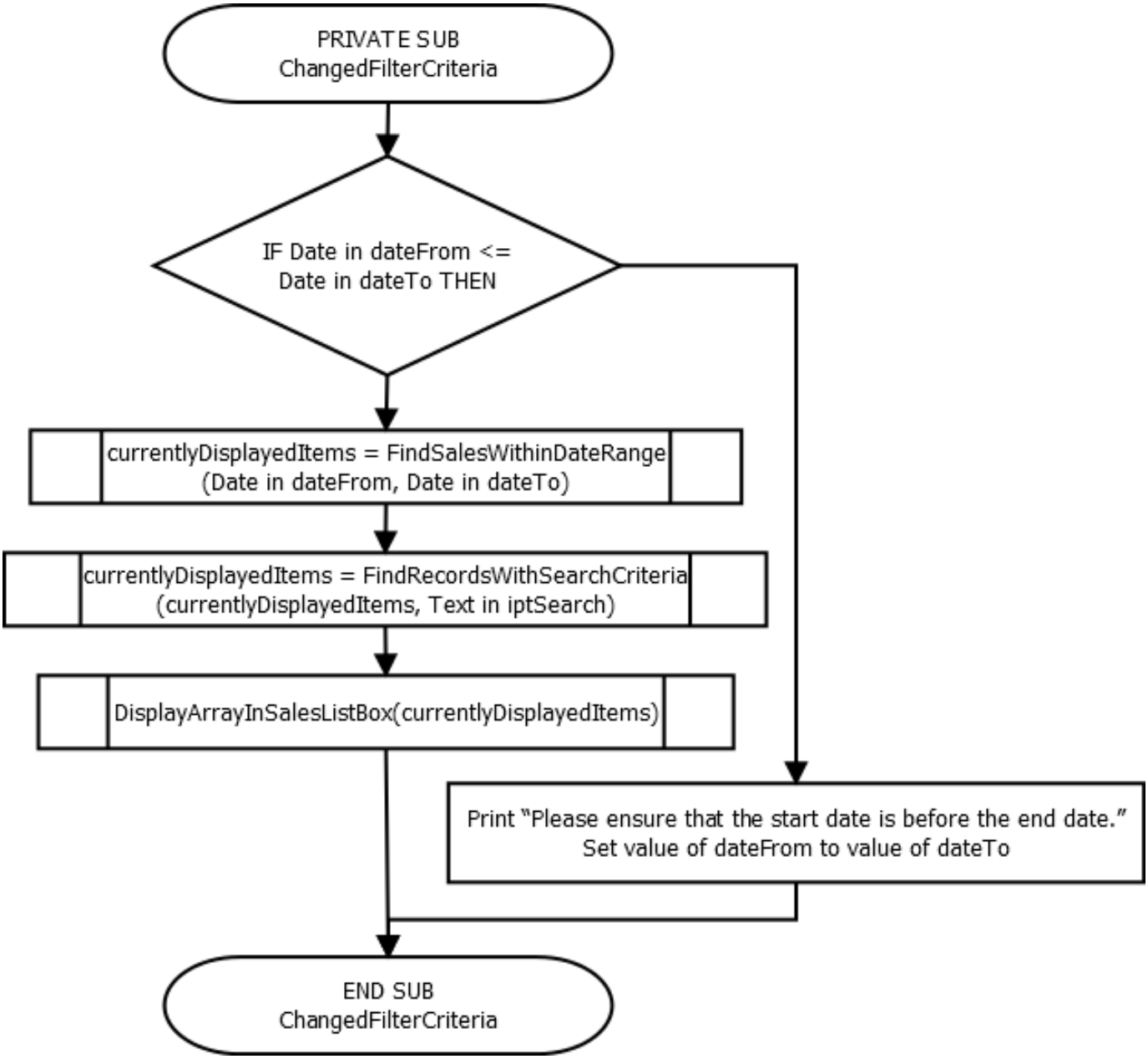

PRIVATE SUB
ClickedButtonBack
Show formEditMenu
Close current form
END SUB
ClickedButtonBack

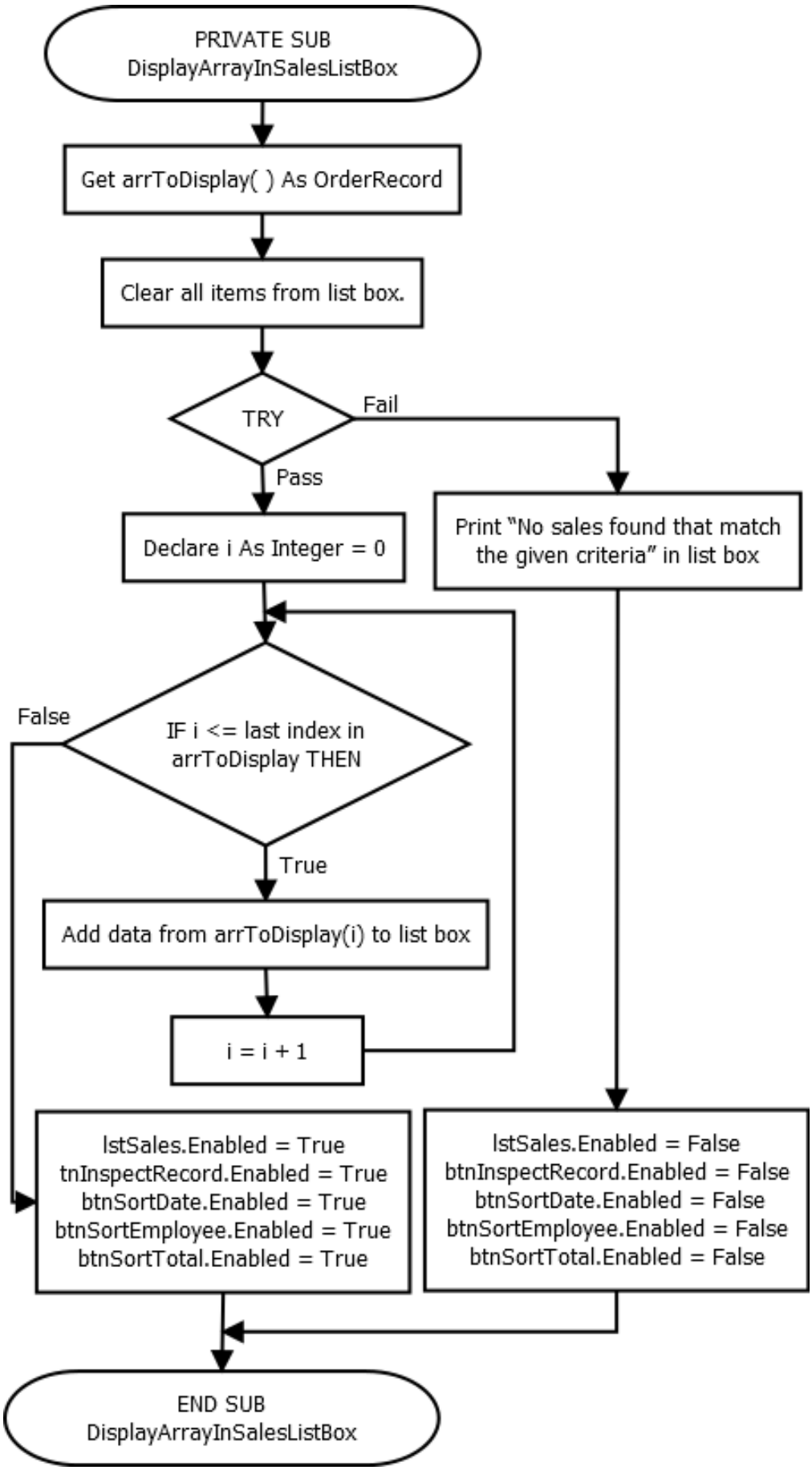

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

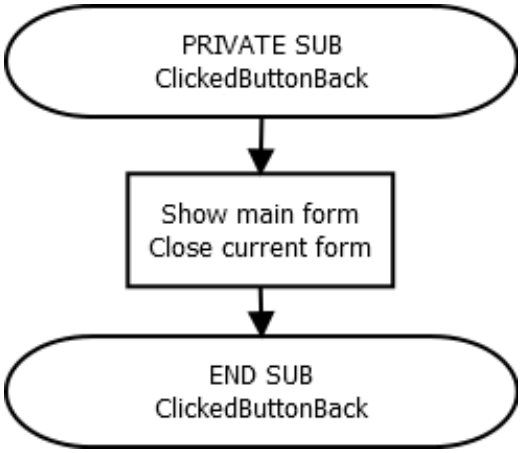
VIEW SALES

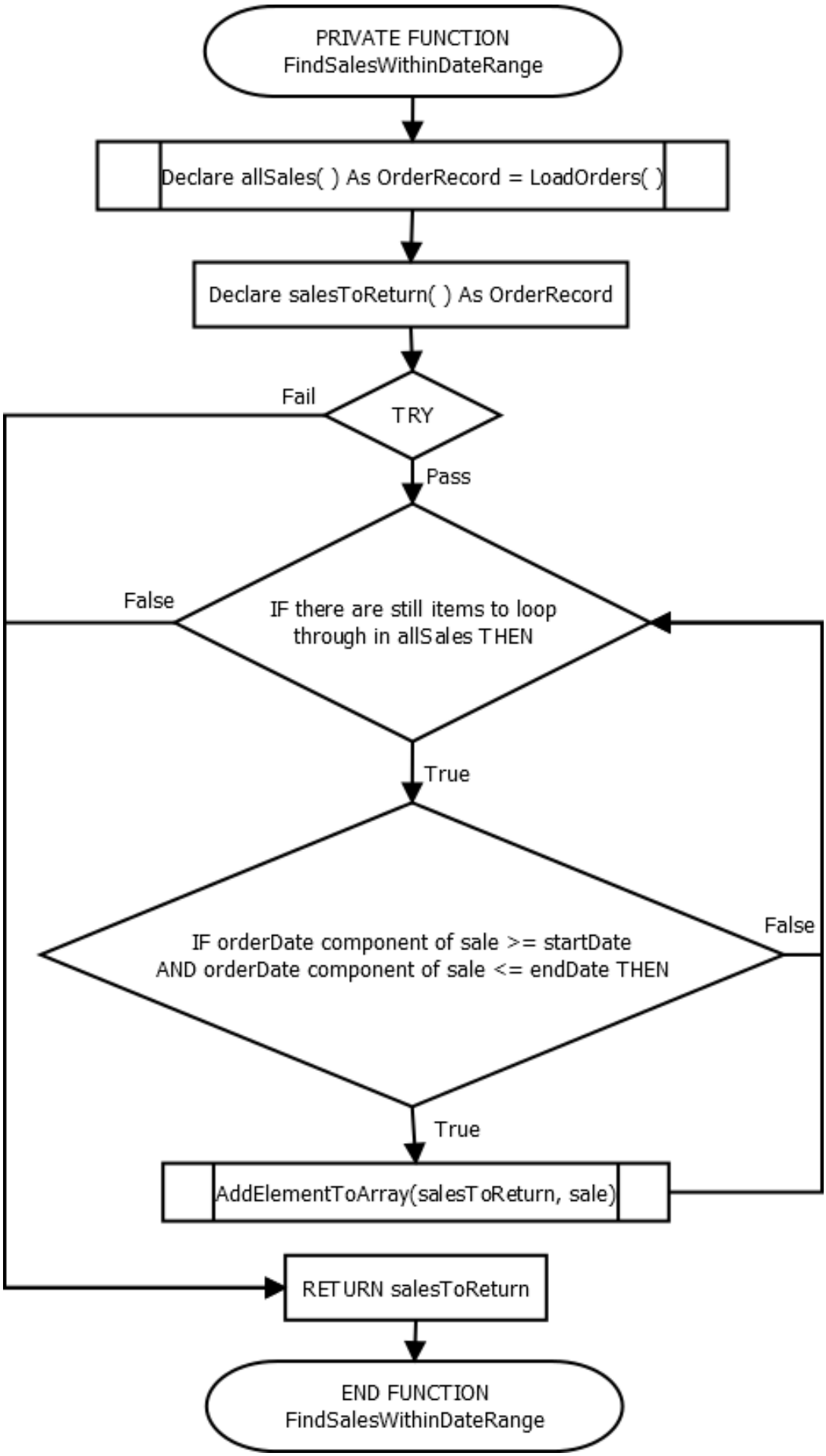

PUBLIC CLASS
formViewSales
Declare currentlyDisplayedItems( ) As OrderRecord = LoadOrders( )
Declare LoadedFormViewSales( )
Declare ChangedFilterCriteria( )
Declare DisplayArrayInSalesListBox(arrToDisplay( ) As OrderRecord)
Declare ClickedButtonBack( )
Declare FindSalesWithinDateRange(startDate As Date, endDate As Date)
Declare ClickedButtonInspectRecord( )
Declare DisplayItemInfo(itemToDisplay As OrderRecord)
Declare ItemIsSelected( )
Declare ClickedButtonSortByOrderTotal( )
Declare ClickedButtonSortByEmployee( )
Declare ClickedButtonSortByOrderDate( )
Declare ClickedButtonHelp( )
END CLASS
formViewSales

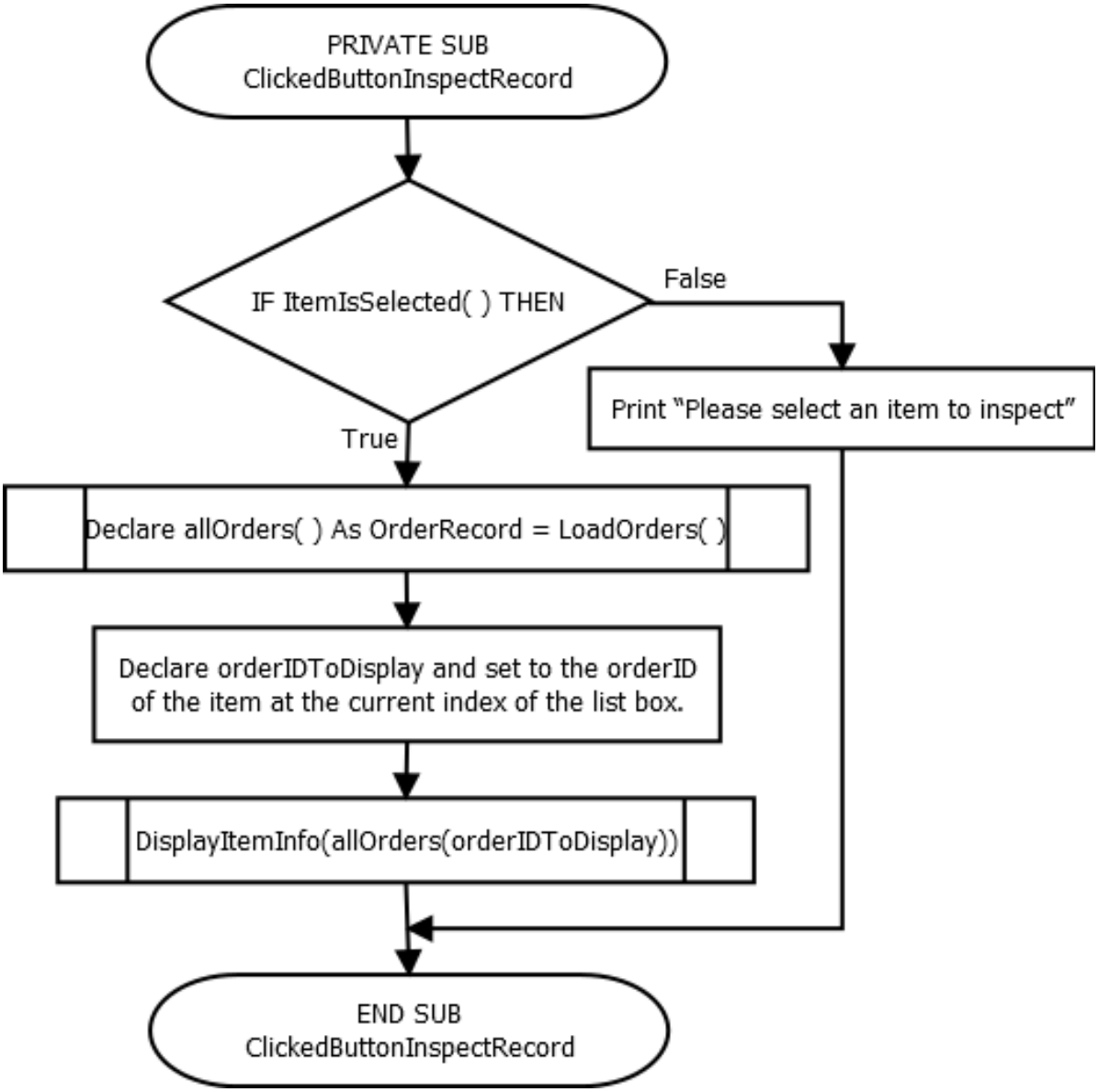

PRIVATE SUB
LoadedFormViewSales
DisplayArrayInSalesListBox(LoadOrders( ))
END SUB
LoadedFormViewSales

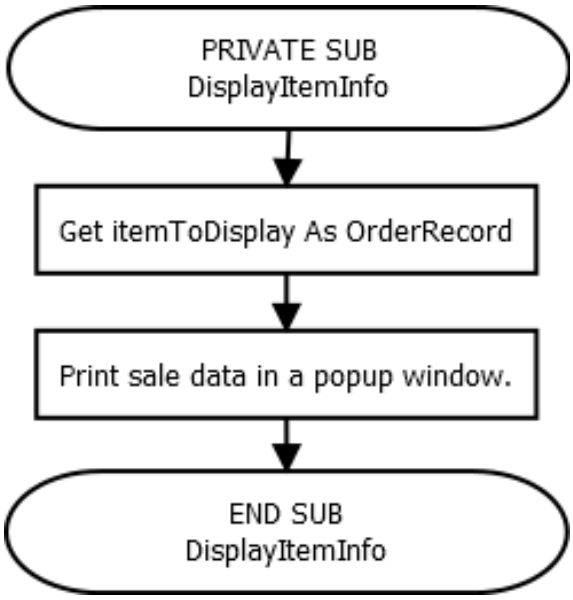

PRIVATE SUB
ChangedFilterCriteria
IF Date in dateFrom <=
Date in dateTo THEN
currentlyDisplayedItems = FindSalesWithinDateRange
(Date in dateFrom, Date in dateTo)
currentlyDisplayedItems = FindRecordsWithSearchCriteria
(currentlyDisplayedItems, Text in iptSearch)
DisplayArrayInSalesListBox(currentlyDisplayedItems)
Print "Please ensure that the start date is before the end date."
Set value of dateFrom to value of dateTo
END SUB
ChangedFilterCriteria

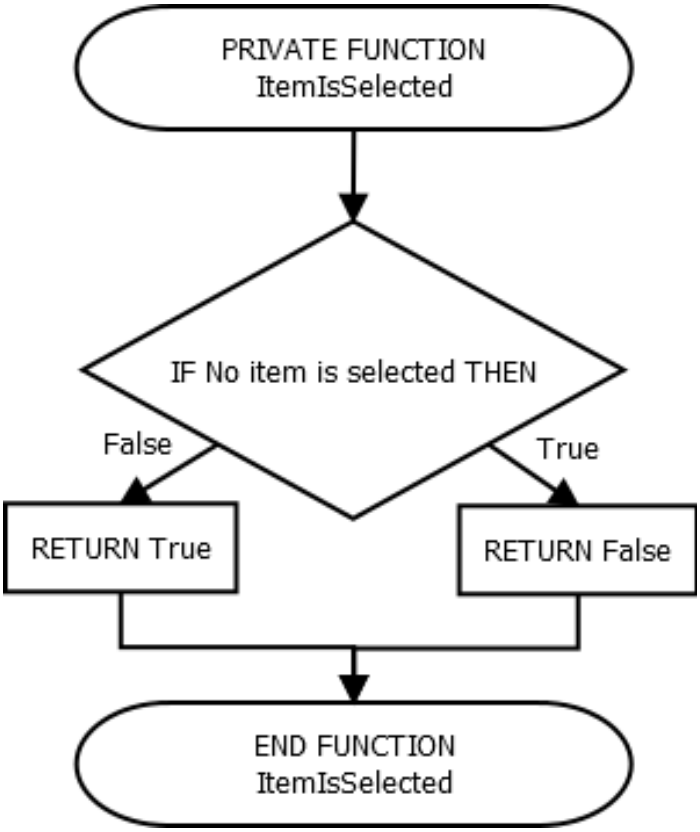

PRIVATE SUB
DisplayArrayInSalesListBox
Get arrToDisplay( ) As OrderRecord
Clear all items from list box.
TRY
Fail
Pass
Print "No sales found that match the given criteria" in list box
Declare i As Integer = 0
False
IF i <= last index in arrToDisplay THEN
True
Add data from arrToDisplay(i) to list box
i = i + 1
lstSales.Enabled = True
tnInspectRecord.Enabled = True
btnSortDate.Enabled = True
btnSortEmployee.Enabled = True
btnSortTotal.Enabled = True
lstSales.Enabled = False
btnInspectRecord.Enabled = False
btnSortDate.Enabled = False
btnSortEmployee.Enabled = False
btnSortTotal.Enabled = False
END SUB
DisplayArrayInSalesListBox

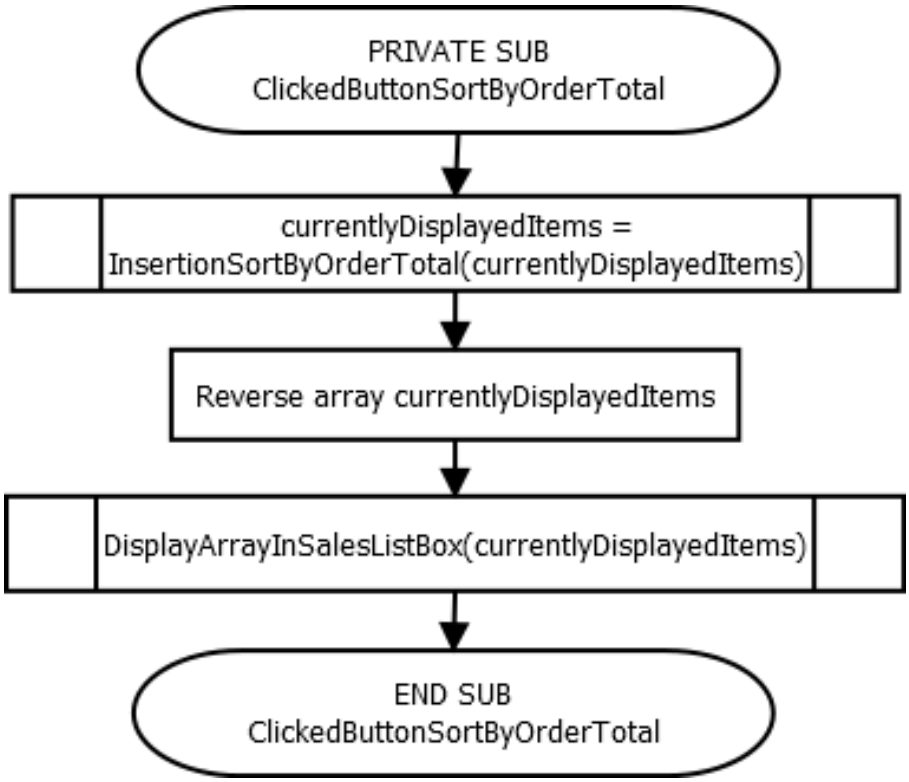

PRIVATE SUB
ClickedButtonBack
Show main form
Close current form
END SUB
ClickedButtonBack

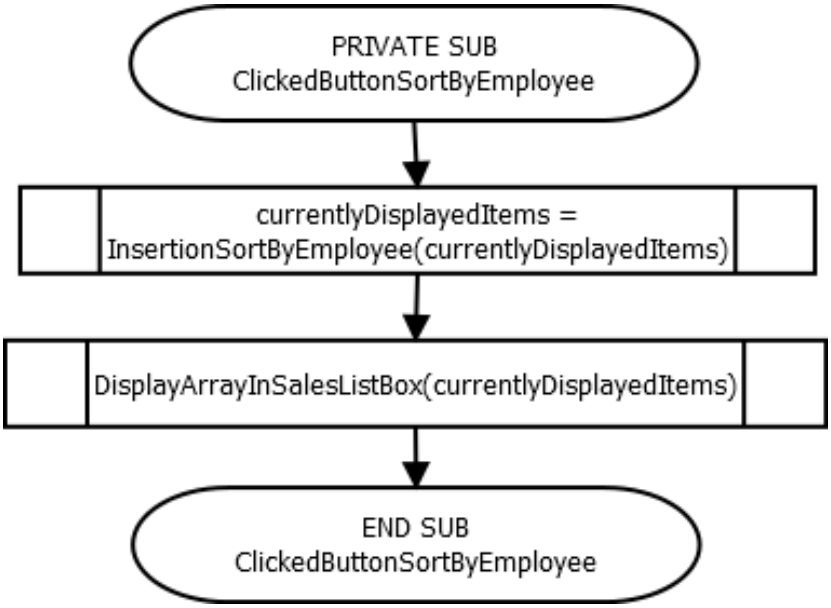

PRIVATE FUNCTION
FindSalesWithinDateRange
Declare allSales( ) As OrderRecord = LoadOrders( )
Declare salesToReturn( ) As OrderRecord
Fail
TRY
Pass
False
IF there are still items to loop
through in allSales THEN
True
False
IF orderDate component of sale >= startDate
AND orderDate component of sale <= endDate THEN
True
AddElementToArray(salesToReturn, sale)
RETURN salesToReturn
END FUNCTION
FindSalesWithinDateRange

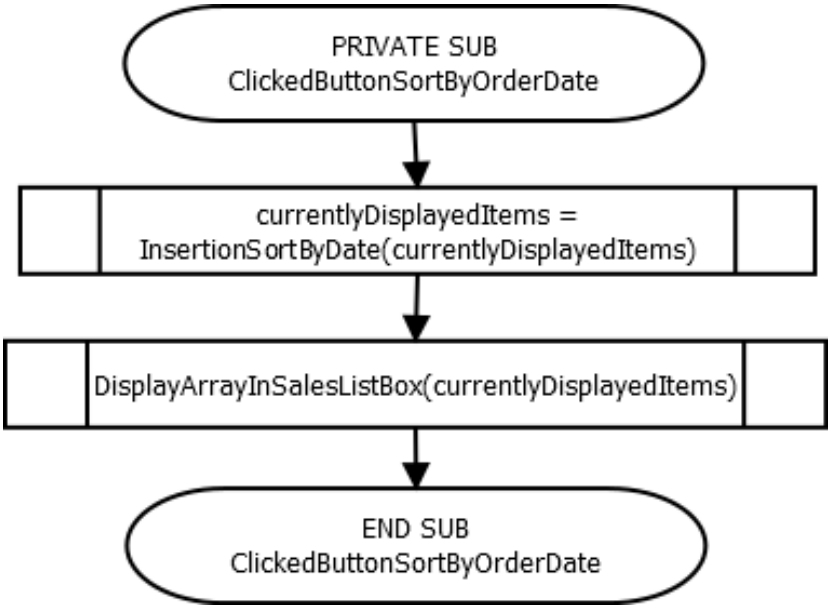

PRIVATE SUB
ClickedButtonInspectRecord
IF ItemIsSelected( ) THEN
False
Print "Please select an item to inspect"
True
Declare allOrders( ) As OrderRecord = LoadOrders( )
Declare orderIDToDisplay and set to the orderID of the item at the current index of the list box.
DisplayItemInfo(allOrders(orderIDToDisplay))
END SUB
ClickedButtonInspectRecord

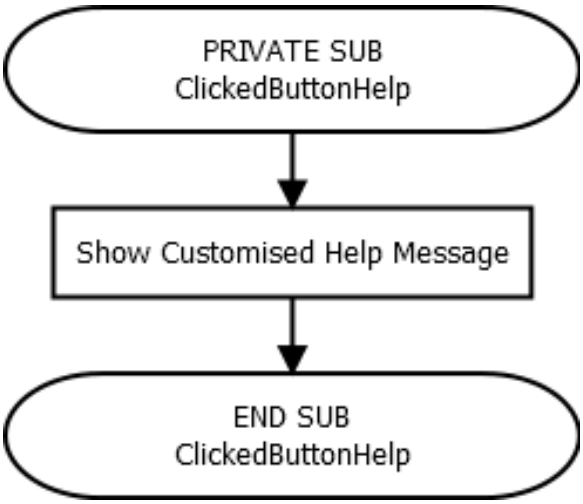

PRIVATE SUB
DisplayItemInfo
Get itemToDisplay As OrderRecord
Print sale data in a popup window.
END SUB
DisplayItemInfo


PRIVATE FUNCTION
ItemIsSelected
IF No item is selected THEN
False
True
RETURN True
RETURN False
END FUNCTION
ItemIsSelected

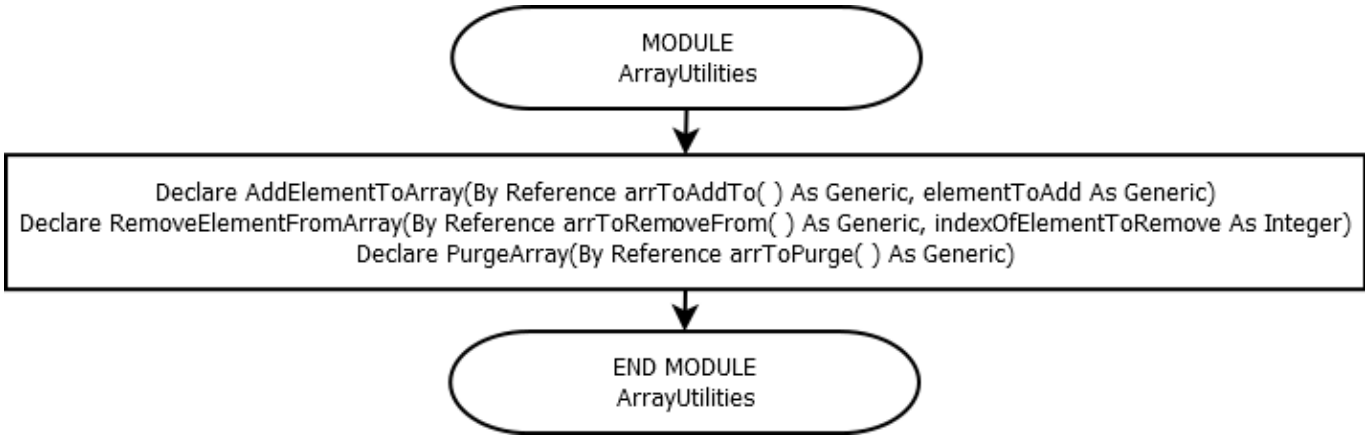

PRIVATE SUB
ClickedButtonSortByOrderTotal
currentlyDisplayedItems =
InsertionSortByOrderTotal(currentlyDisplayedItems)
Reverse array currentlyDisplayedItems
DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB
ClickedButtonSortByOrderTotal

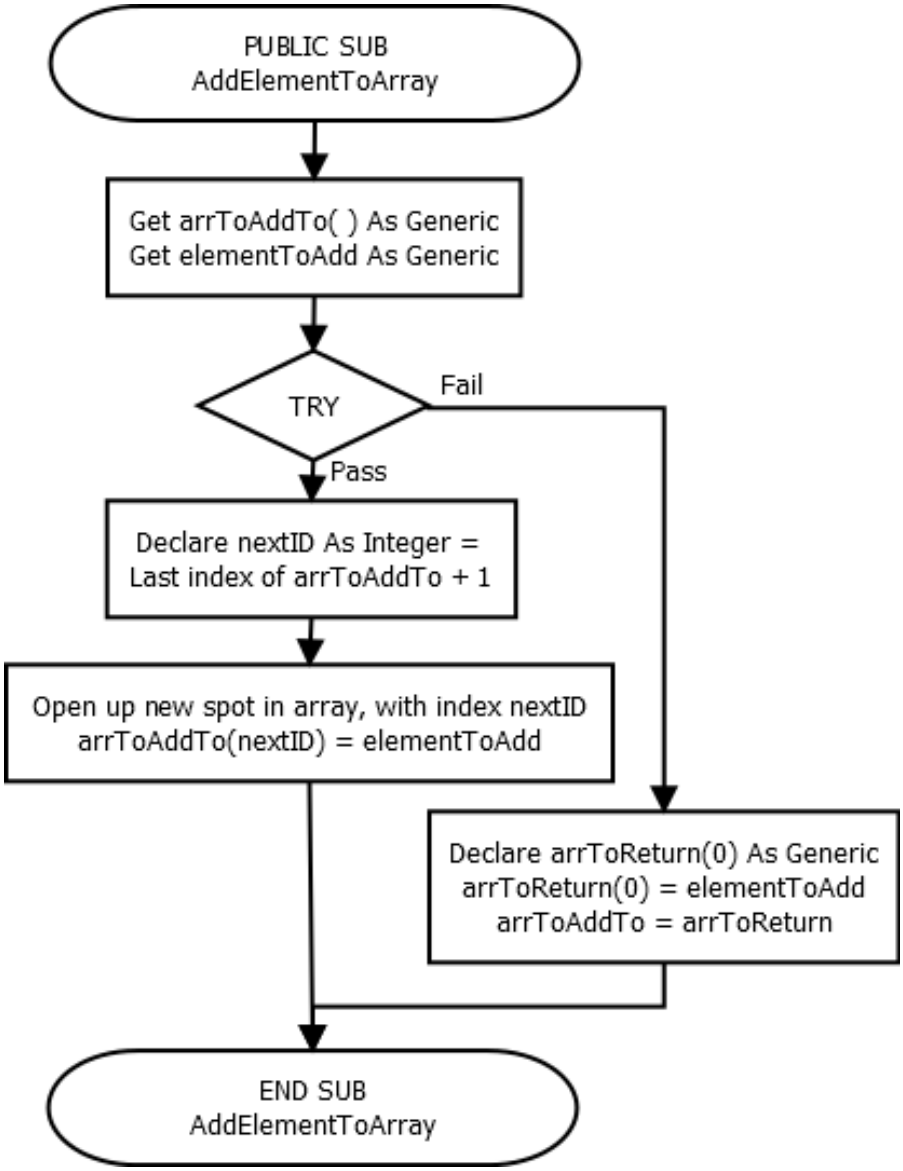

PRIVATE SUB
ClickedButtonSortByEmployee
currentlyDisplayedItems =
InsertionSortByEmployee(currentlyDisplayedItems)
DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB
ClickedButtonSortByEmployee

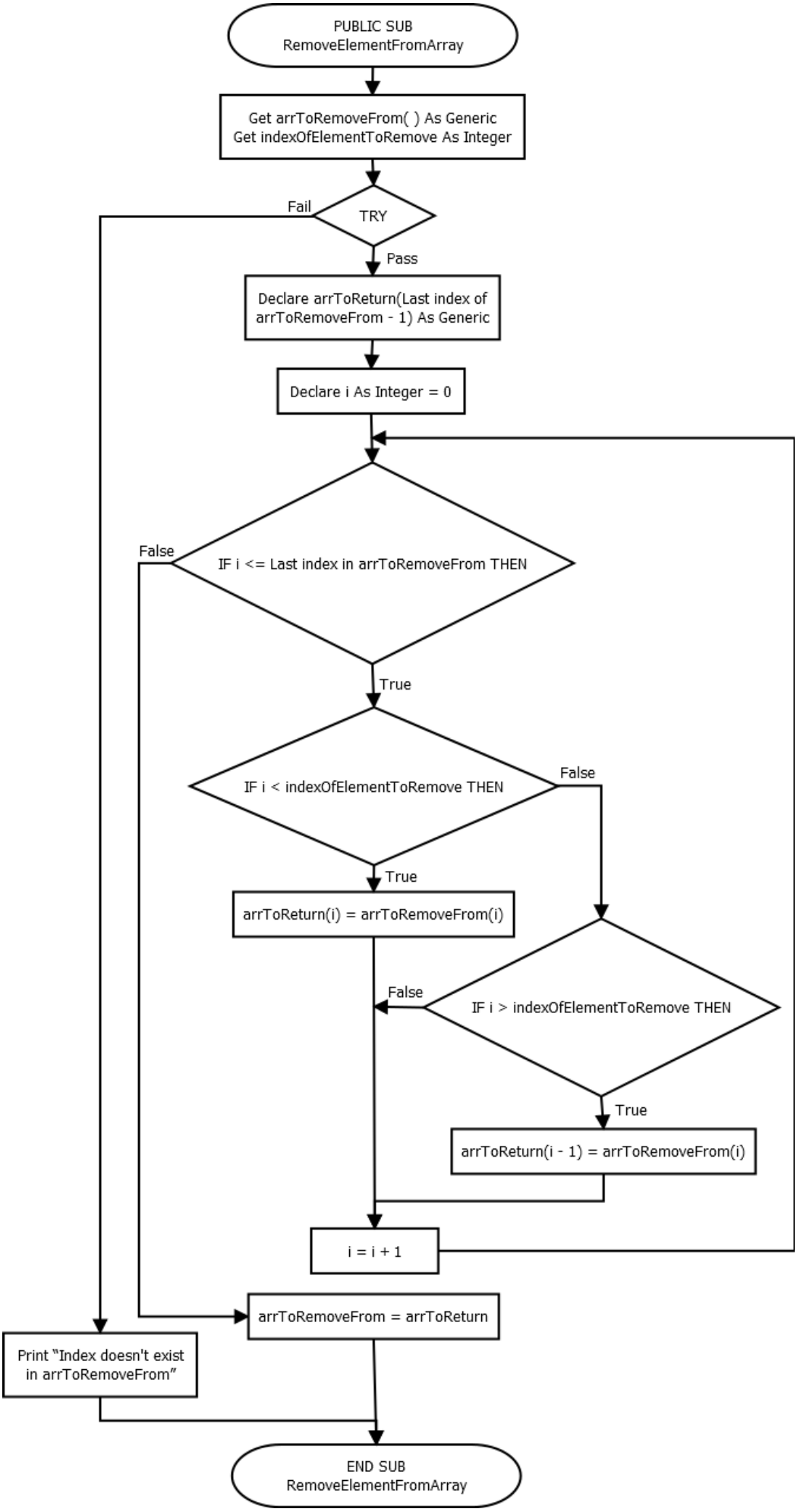

PRIVATE SUB
ClickedButtonSortByOrderDate
currentlyDisplayedItems =
InsertionSortByDate(currentlyDisplayedItems)
DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB
ClickedButtonSortByOrderDate

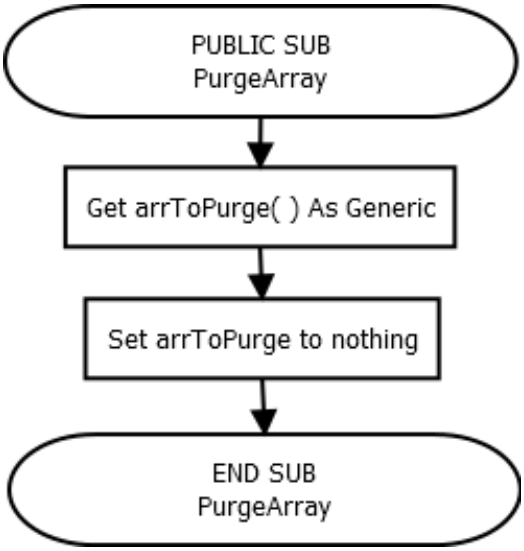

PRIVATE SUB
ClickedButtonHelp
Show Customised Help Message
END SUB
ClickedButtonHelp

ARRAY UTILITIES

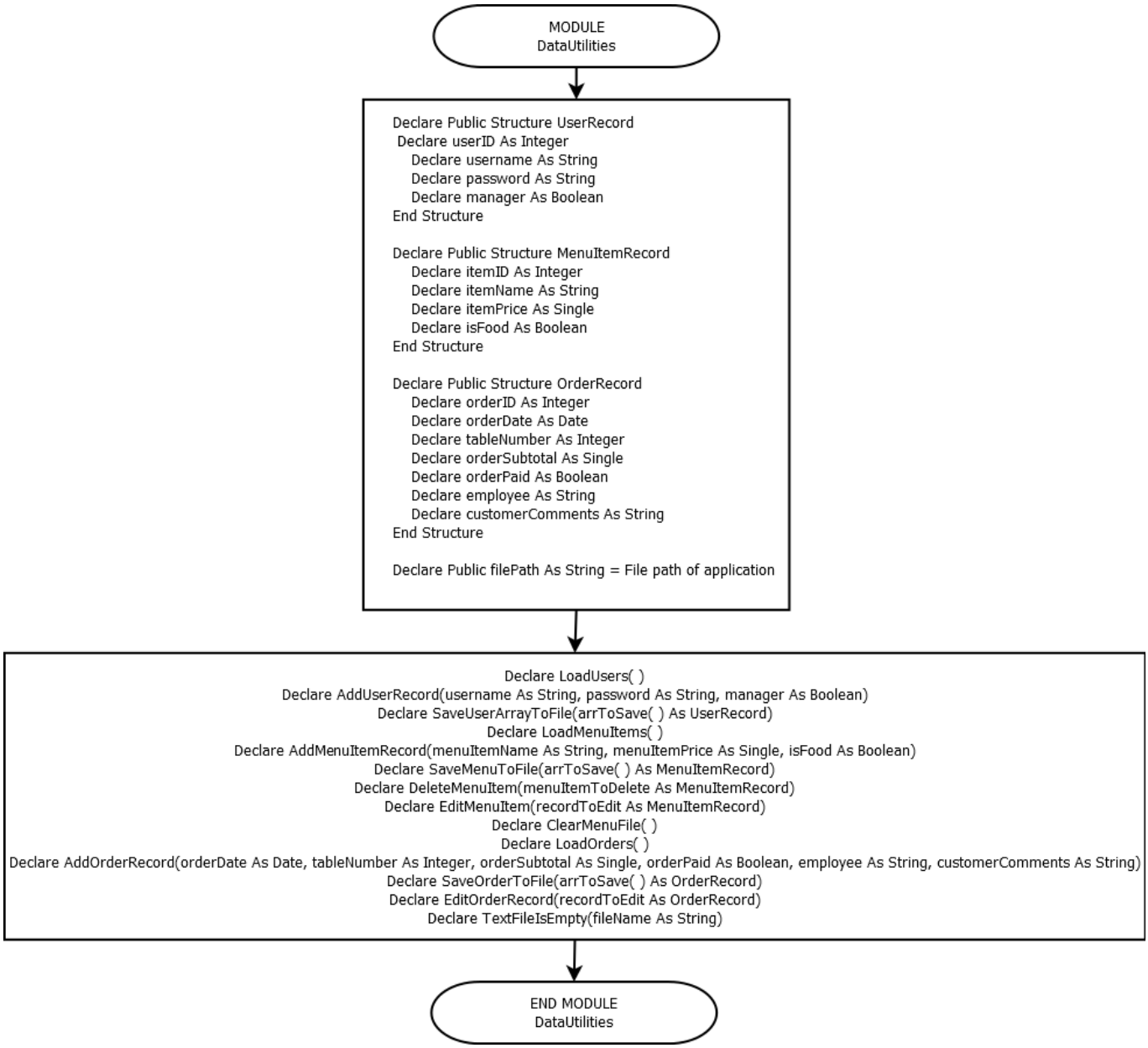


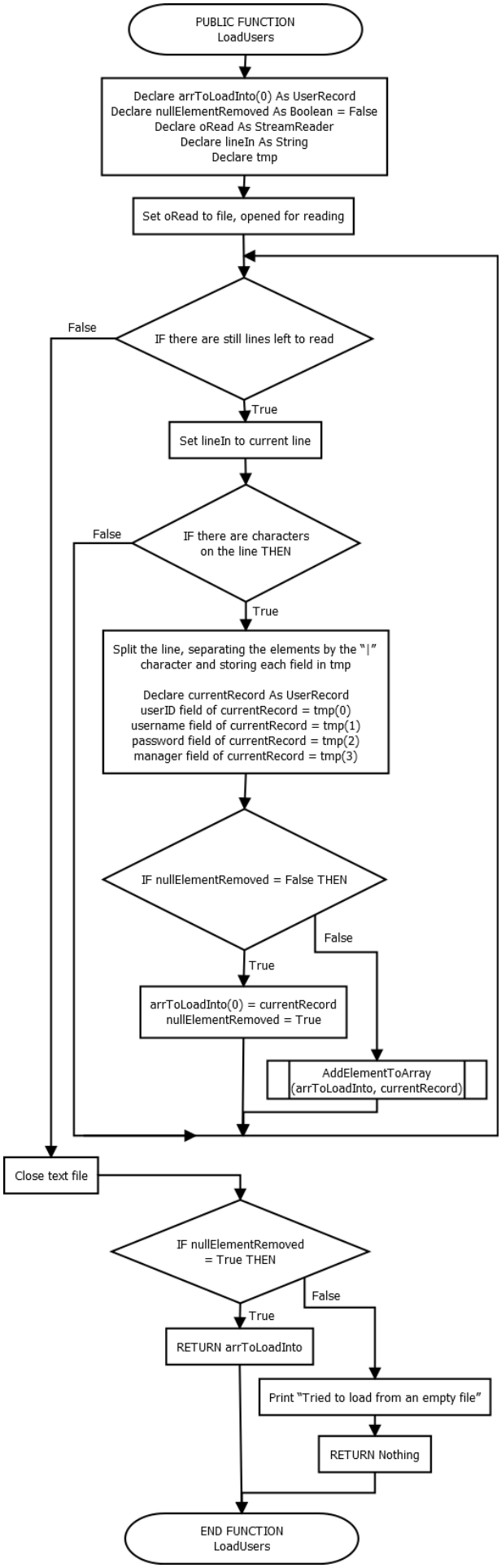


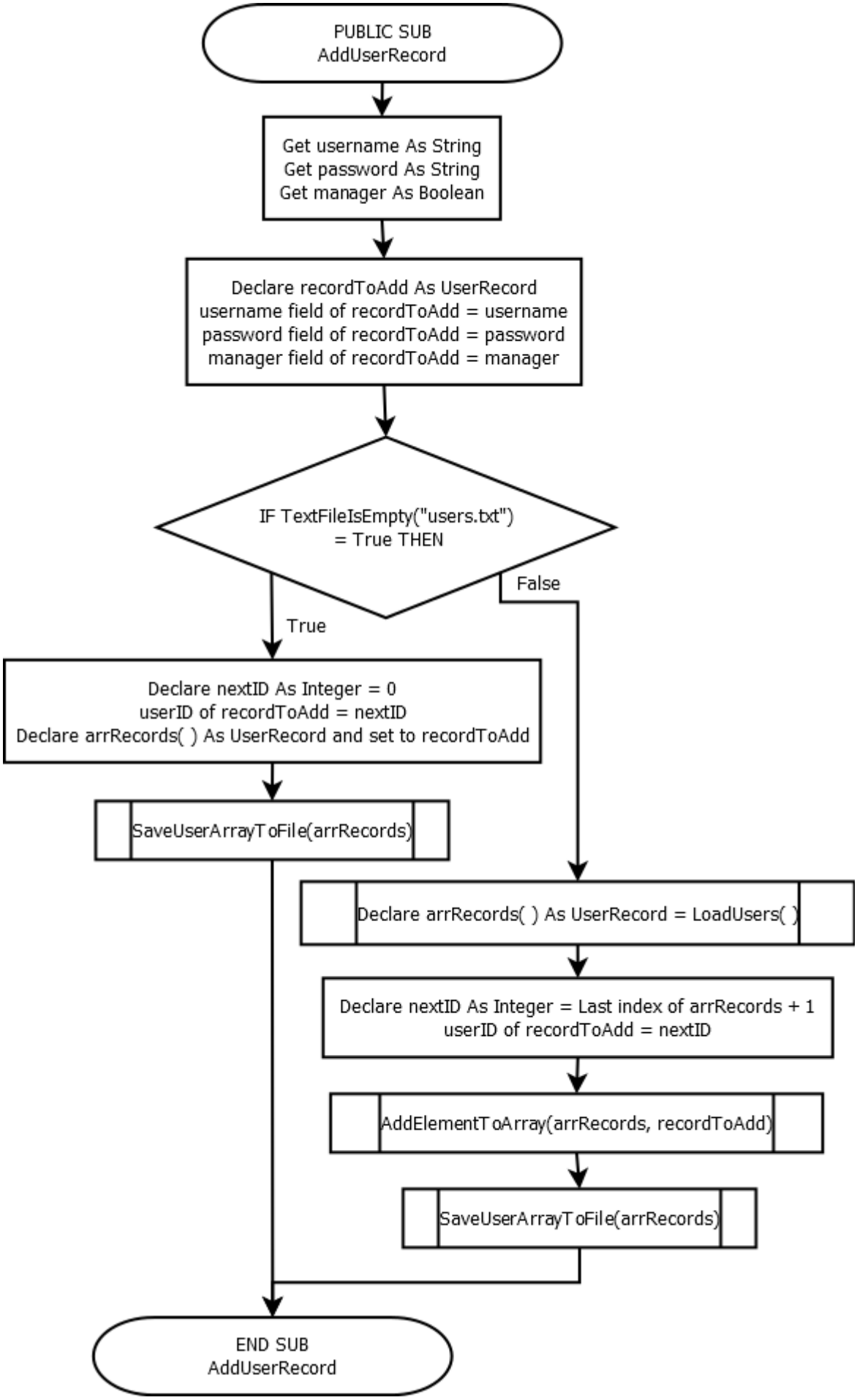

PUBLIC SUB
RemoveElementFromArray
Get arrToRemoveFrom( ) As Generic
Get indexOfElementToRemove As Integer
TRY
Fail
Pass
Declare arrToReturn(Last index of
arrToRemoveFrom - 1) As Generic
Declare i As Integer = 0
IF i <= Last index in arrToRemoveFrom THEN
False
True
IF i < indexOfElementToRemove THEN
False
True
arrToReturn(i) = arrToRemoveFrom(i)
IF i > indexOfElementToRemove THEN
False
True
arrToReturn(i - 1) = arrToRemoveFrom(i)
i = i + 1
arrToRemoveFrom = arrToReturn
Print "Index doesn't exist
in arrToRemoveFrom"
END SUB
RemoveElementFromArray

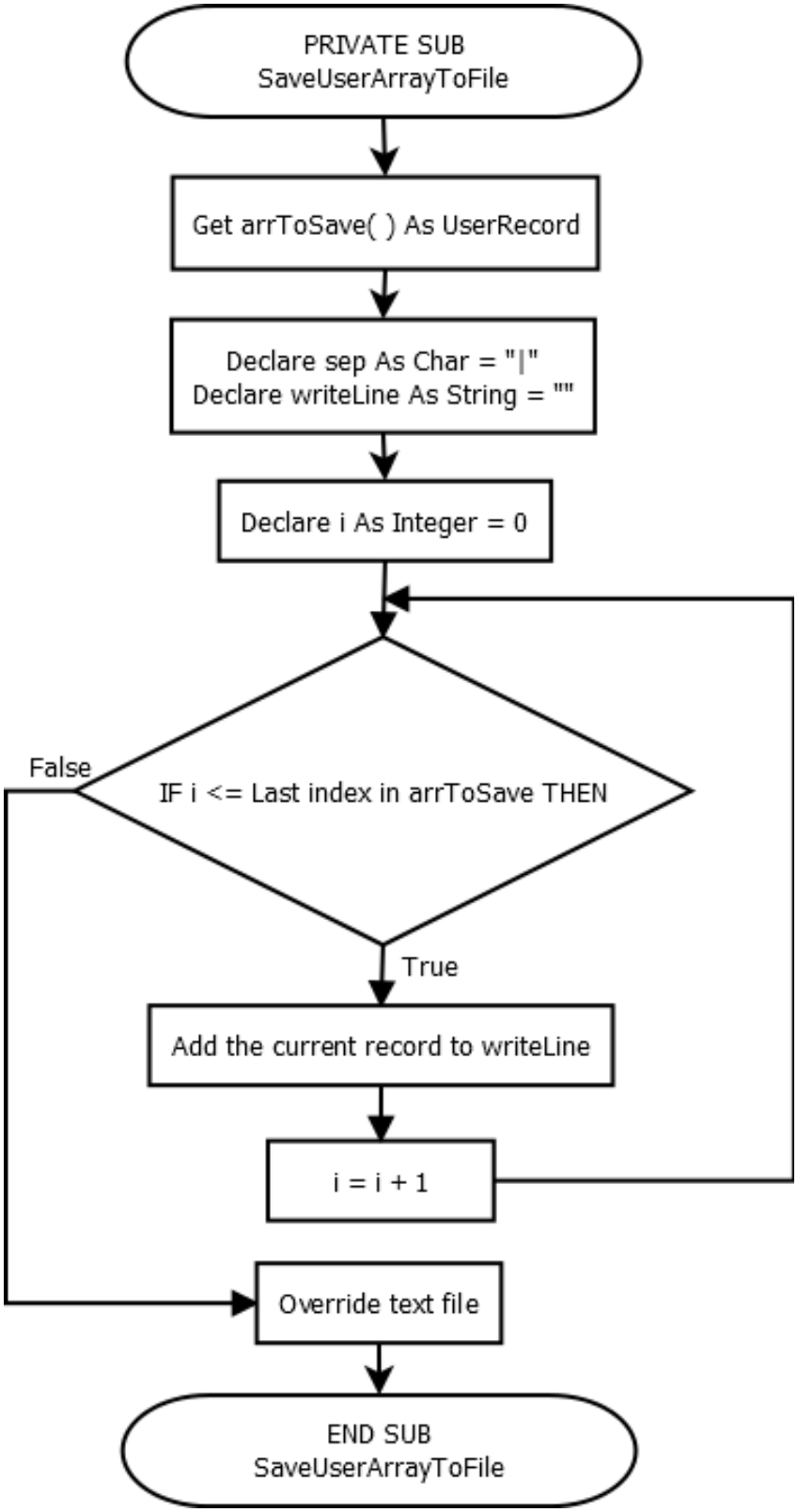

PUBLIC SUB
PurgeArray
Get arrToPurge( ) As Generic
Set arrToPurge to nothing
END SUB
PurgeArray

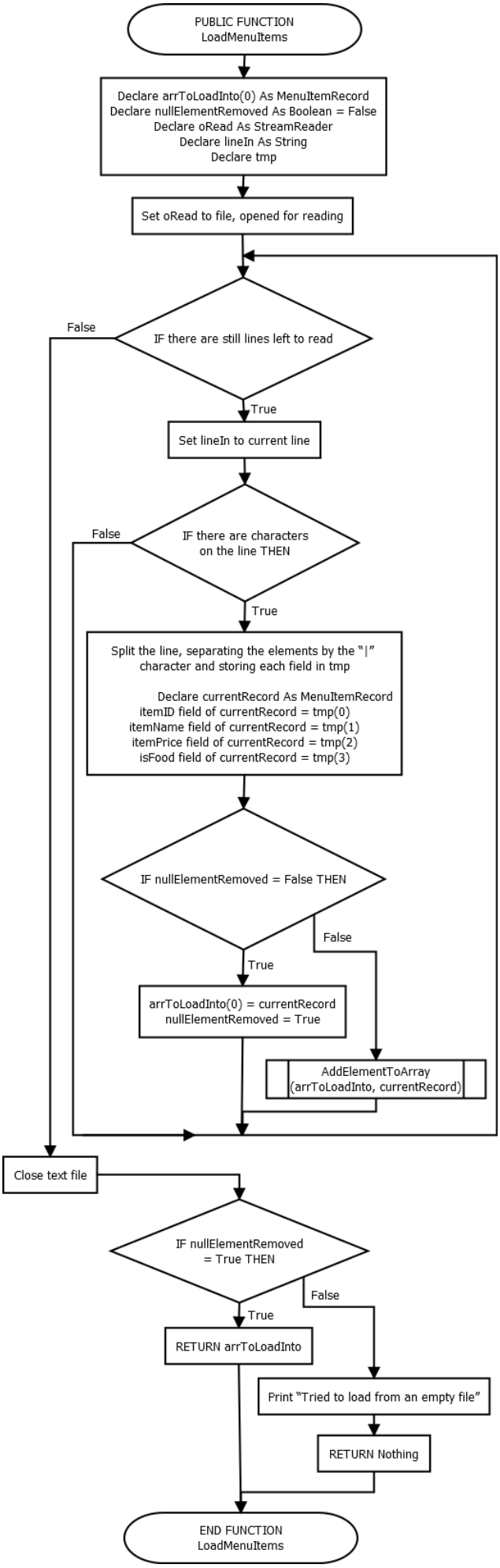
DATA UTILITIES

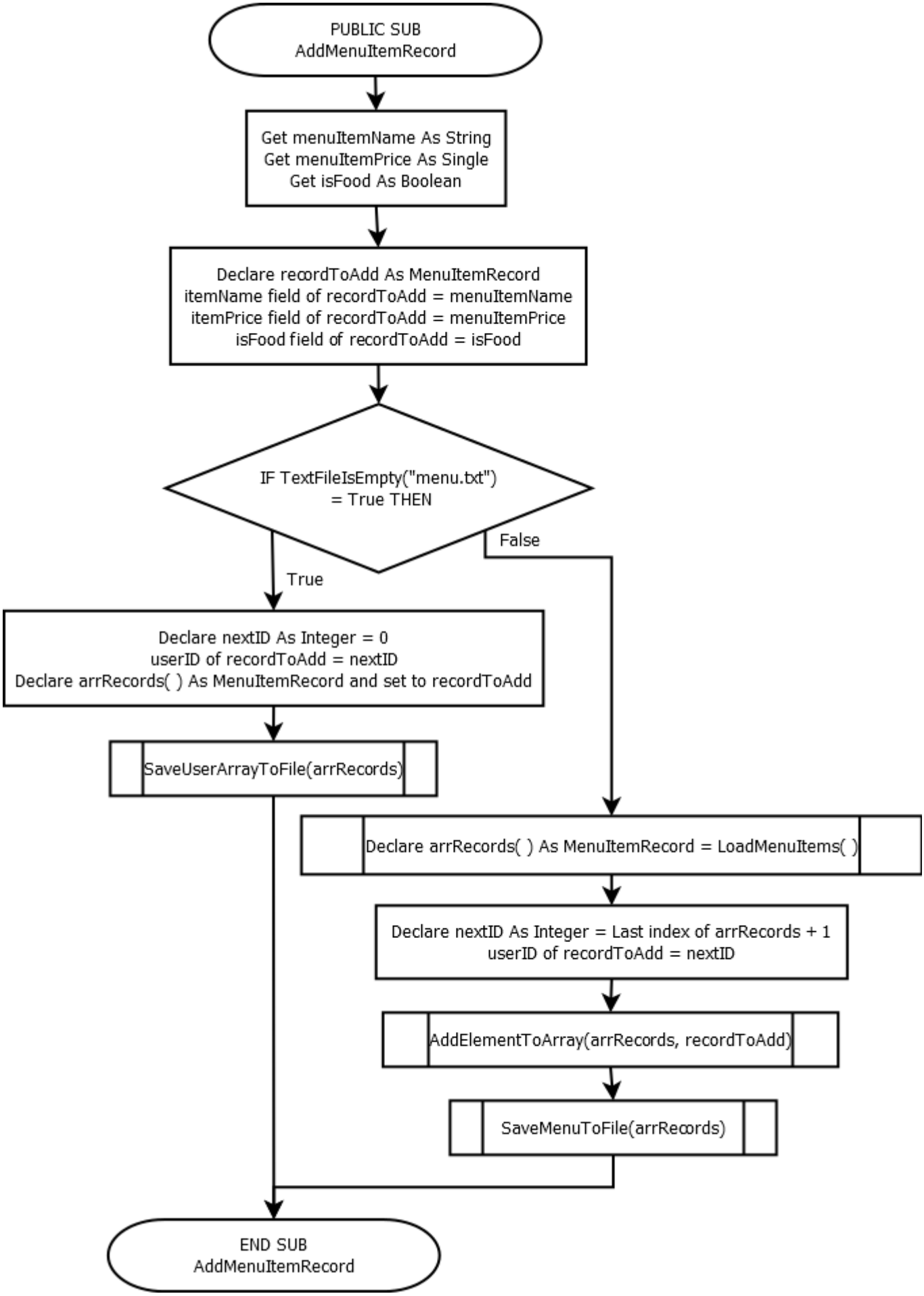


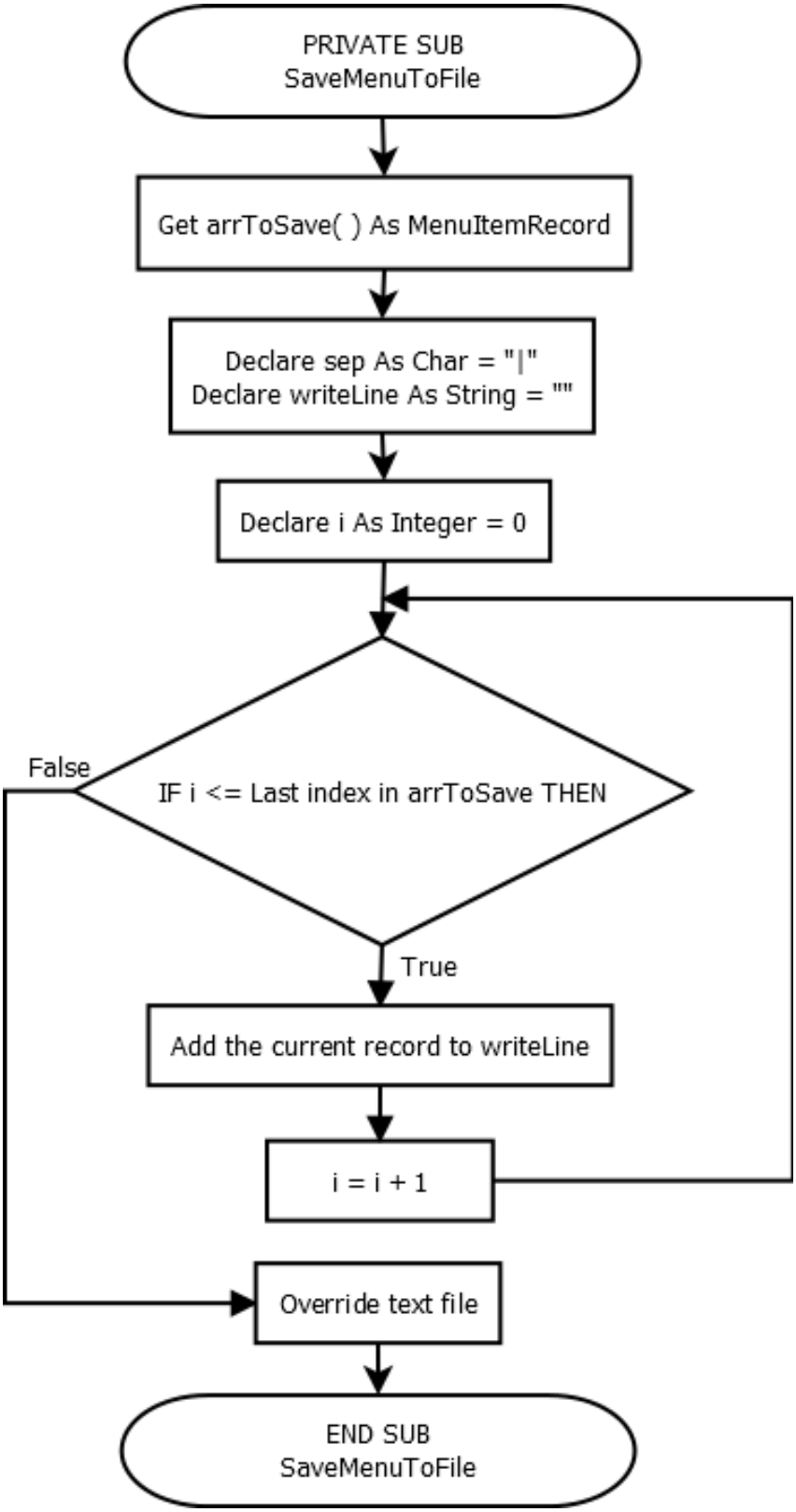

PUBLIC FUNCTION
LoadUsers
Declare arrToLoadInto(0) As UserRecord
Declare nullElementRemoved As Boolean = False
Declare oRead As StreamReader
Declare lineIn As String
Declare tmp
Set oRead to file, opened for reading
IF there are still lines left to read
False
True
Set lineIn to current line
IF there are characters
on the line THEN
False
True
Split the line, separating the elements by the "|"
character and storing each field in tmp
Declare currentRecord As UserRecord
userID field of currentRecord = tmp(0)
username field of currentRecord = tmp(1)
password field of currentRecord = tmp(2)
manager field of currentRecord = tmp(3)
IF nullElementRemoved = False THEN
False
True
arrToLoadInto(0) = currentRecord
nullElementRemoved = True
AddElementToArray
(arrToLoadInto, currentRecord)
Close text file
IF nullElementRemoved
= True THEN
False
True
RETURN arrToLoadInto
Print "Tried to load from an empty file"
RETURN Nothing
END FUNCTION
LoadUsers

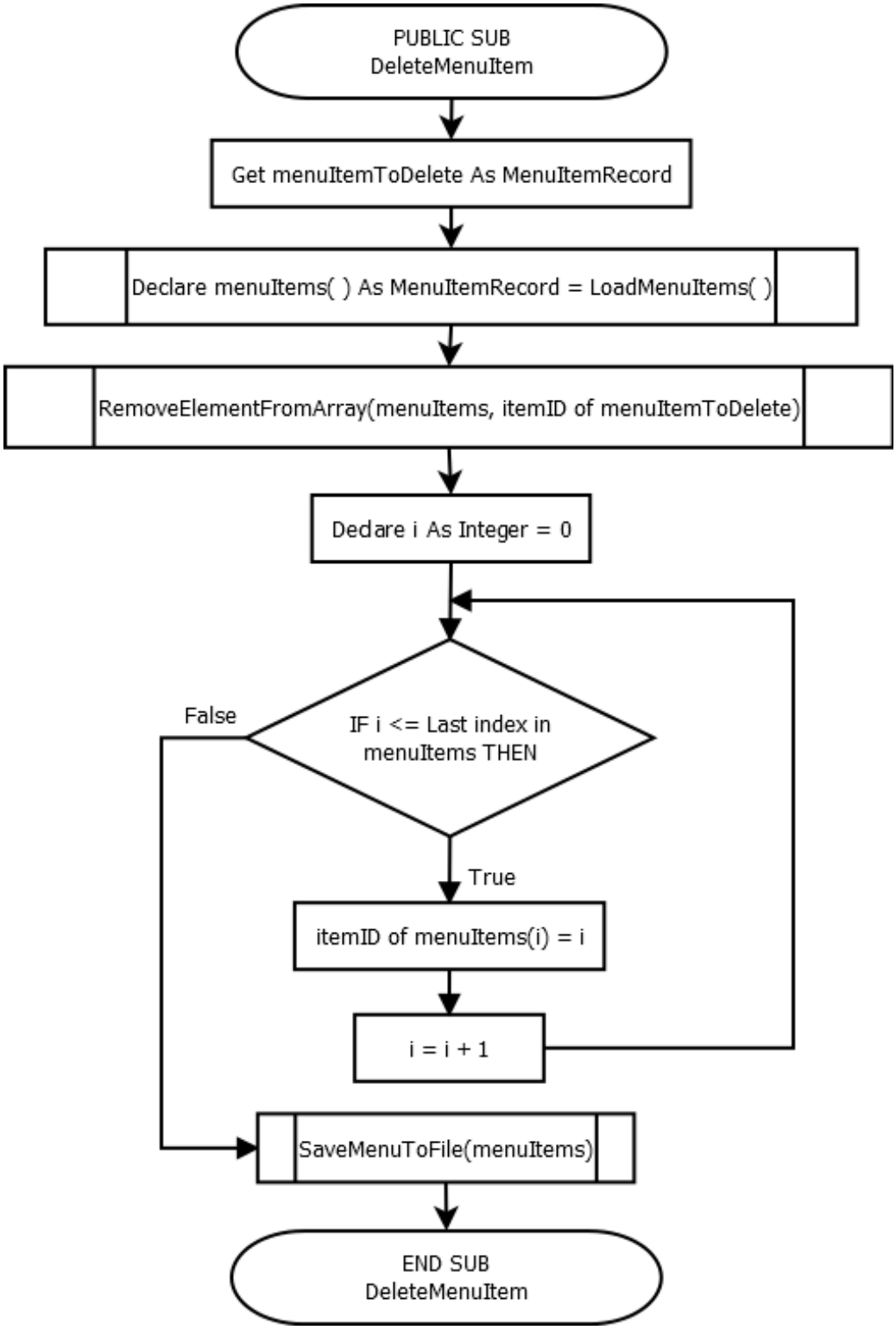

PUBLIC SUB
AddUserRecord
Get username As String
Get password As String
Get manager As Boolean
Declare recordToAdd As UserRecord
username field of recordToAdd = username
password field of recordToAdd = password
manager field of recordToAdd = manager
IF TextFileIsEmpty("users.txt")
= True THEN
True
False
Declare nextID As Integer = 0
userID of recordToAdd = nextID
Declare arrRecords( ) As UserRecord and set to recordToAdd
SaveUserArrayToFile(arrRecords)
Declare arrRecords( ) As UserRecord = LoadUsers( )
Declare nextID As Integer = Last index of arrRecords + 1
userID of recordToAdd = nextID
AddElementToArray(arrRecords, recordToAdd)
SaveUserArrayToFile(arrRecords)
END SUB
AddUserRecord

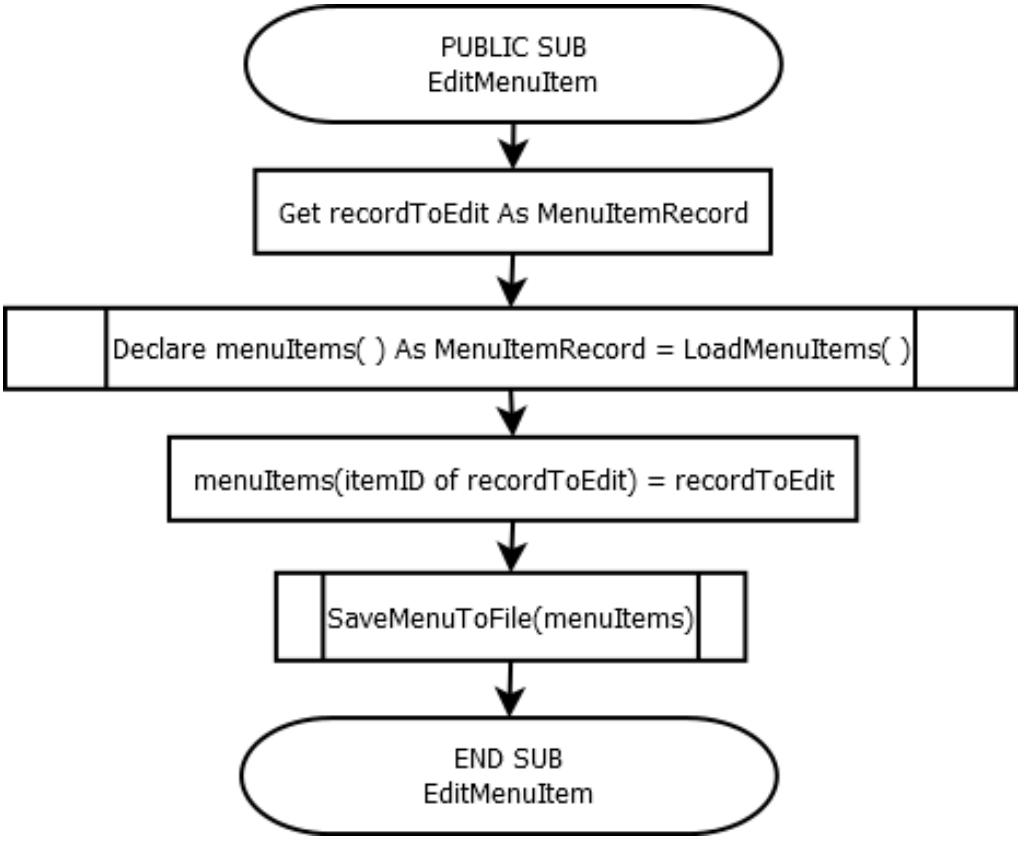

PRIVATE SUB
SaveUserArrayToFile
Get arrToSave( ) As UserRecord
Declare sep As Char = "|"
Declare writeLine As String = ""
Declare i As Integer = 0
False
IF i <= Last index in arrToSave THEN
True
Add the current record to writeLine
i = i + 1
Override text file
END SUB
SaveUserArrayToFile

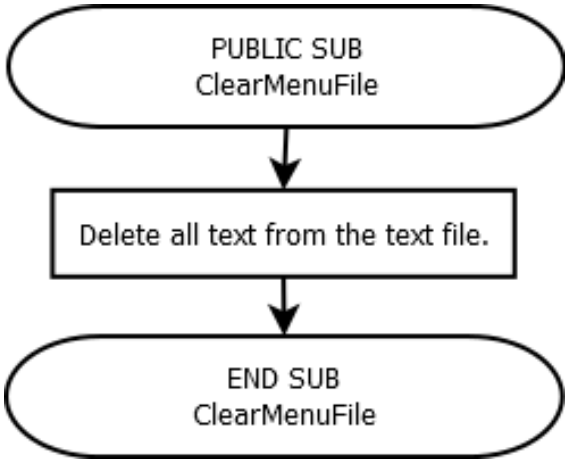

PUBLIC FUNCTION
LoadMenuItems
Declare arrToLoadInto(0) As MenuItemRecord
Declare nullElementRemoved As Boolean = False
Declare oRead As StreamReader
Declare lineIn As String
Declare tmp
Set oRead to file, opened for reading
IF there are still lines left to read
False
True
Set lineIn to current line
IF there are characters
on the line THEN
False
True
Split the line, separating the elements by the "|"
character and storing each field in tmp
Declare currentRecord As MenuItemRecord
itemID field of currentRecord = tmp(0)
itemName field of currentRecord = tmp(1)
itemPrice field of currentRecord = tmp(2)
isFood field of currentRecord = tmp(3)
IF nullElementRemoved = False THEN
False
True
arrToLoadInto(0) = currentRecord
nullElementRemoved = True
AddElementToArray
(arrToLoadInto, currentRecord)
Close text file
IF nullElementRemoved
= True THEN
False
True
RETURN arrToLoadInto
Print "Tried to load from an empty file"
RETURN Nothing
END FUNCTION
LoadMenuItems

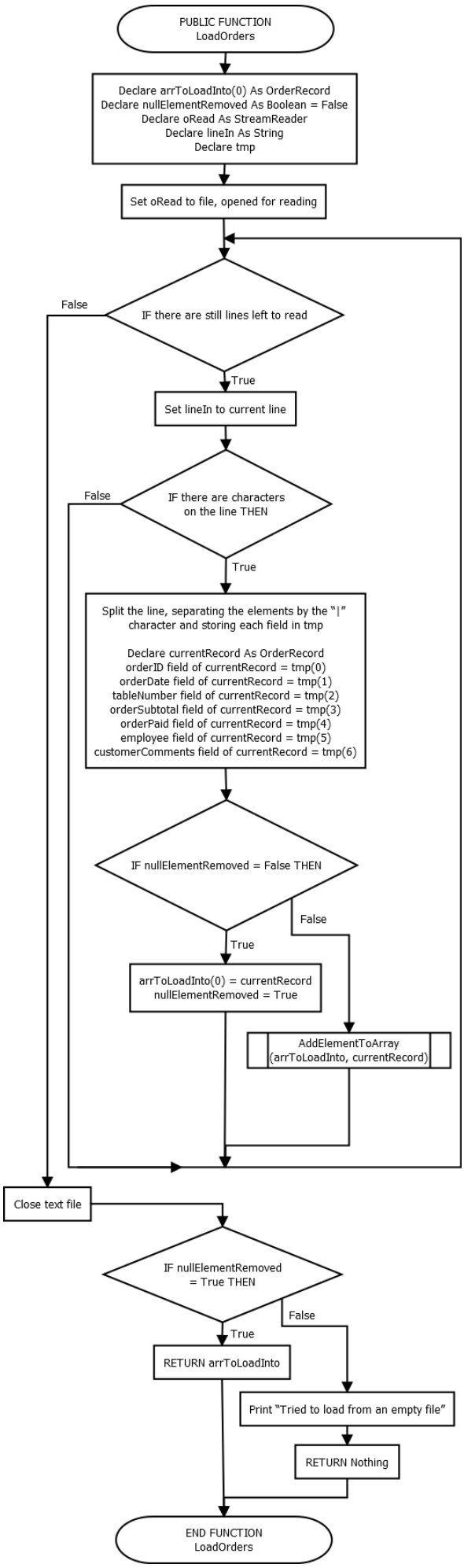

PUBLIC SUB
AddMenuItemRecord
Get menuItemName As String
Get menuItemPrice As Single
Get isFood As Boolean
Declare recordToAdd As MenuItemRecord
itemName field of recordToAdd = menuItemName
itemPrice field of recordToAdd = menuItemPrice
isFood field of recordToAdd = isFood
IF TextFileIsEmpty("menu.txt")
= True THEN
False
True
Declare nextID As Integer = 0
userID of recordToAdd = nextID
Declare arrRecords( ) As MenuItemRecord and set to recordToAdd
SaveUserArrayToFile(arrRecords)
Declare arrRecords( ) As MenuItemRecord = LoadMenuItems( )
Declare nextID As Integer = Last index of arrRecords + 1
userID of recordToAdd = nextID
AddElementToArray(arrRecords, recordToAdd)
SaveMenuToFile(arrRecords)
END SUB
AddMenuItemRecord

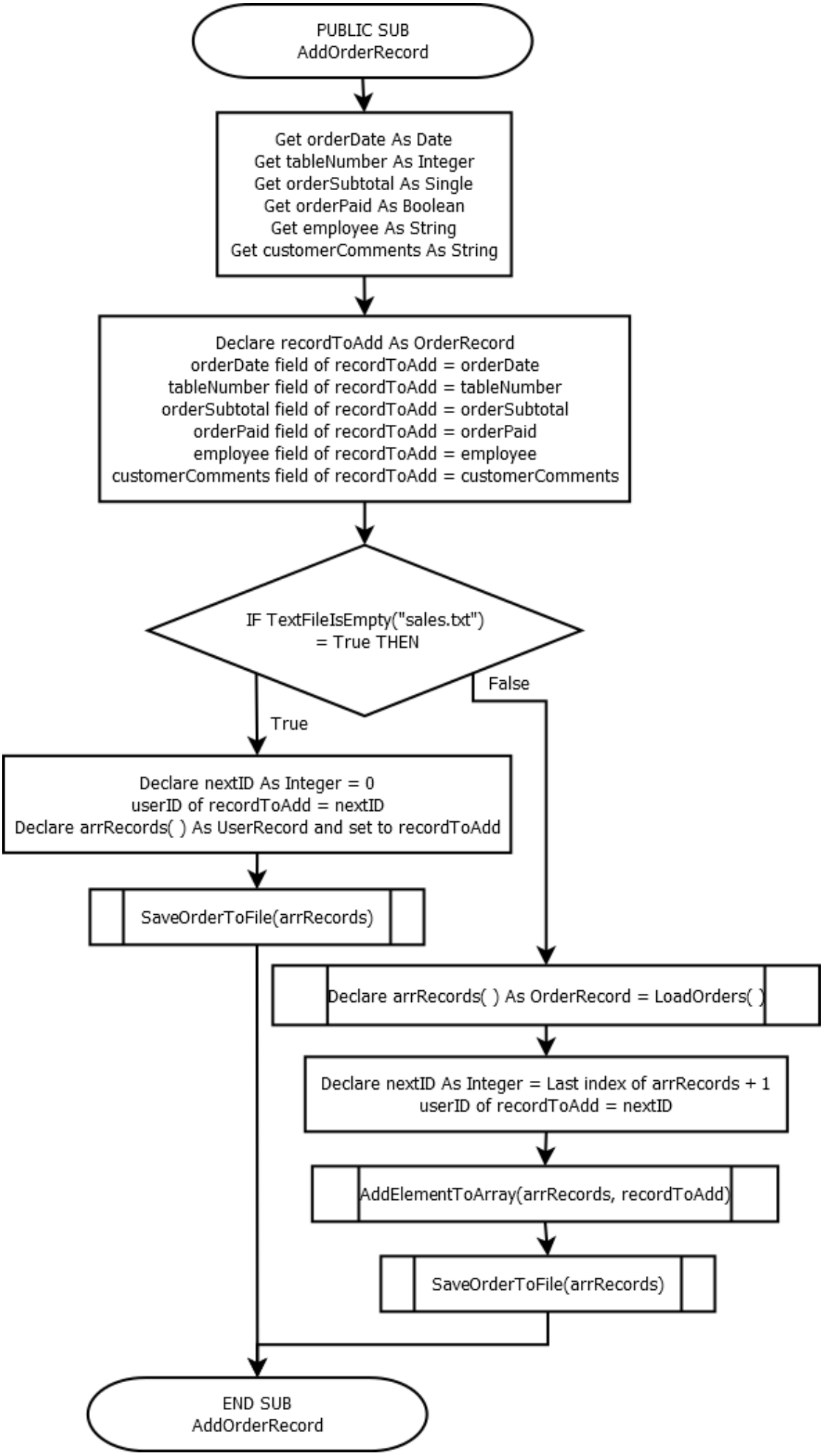

PRIVATE SUB
SaveMenuToFile
Get arrToSave( ) As MenuItemRecord
Declare sep As Char = "|"
Declare writeLine As String = ""
Declare i As Integer = 0
False
IF i <= Last index in arrToSave THEN
True
Add the current record to writeLine
i = i + 1
Override text file
END SUB
SaveMenuToFile

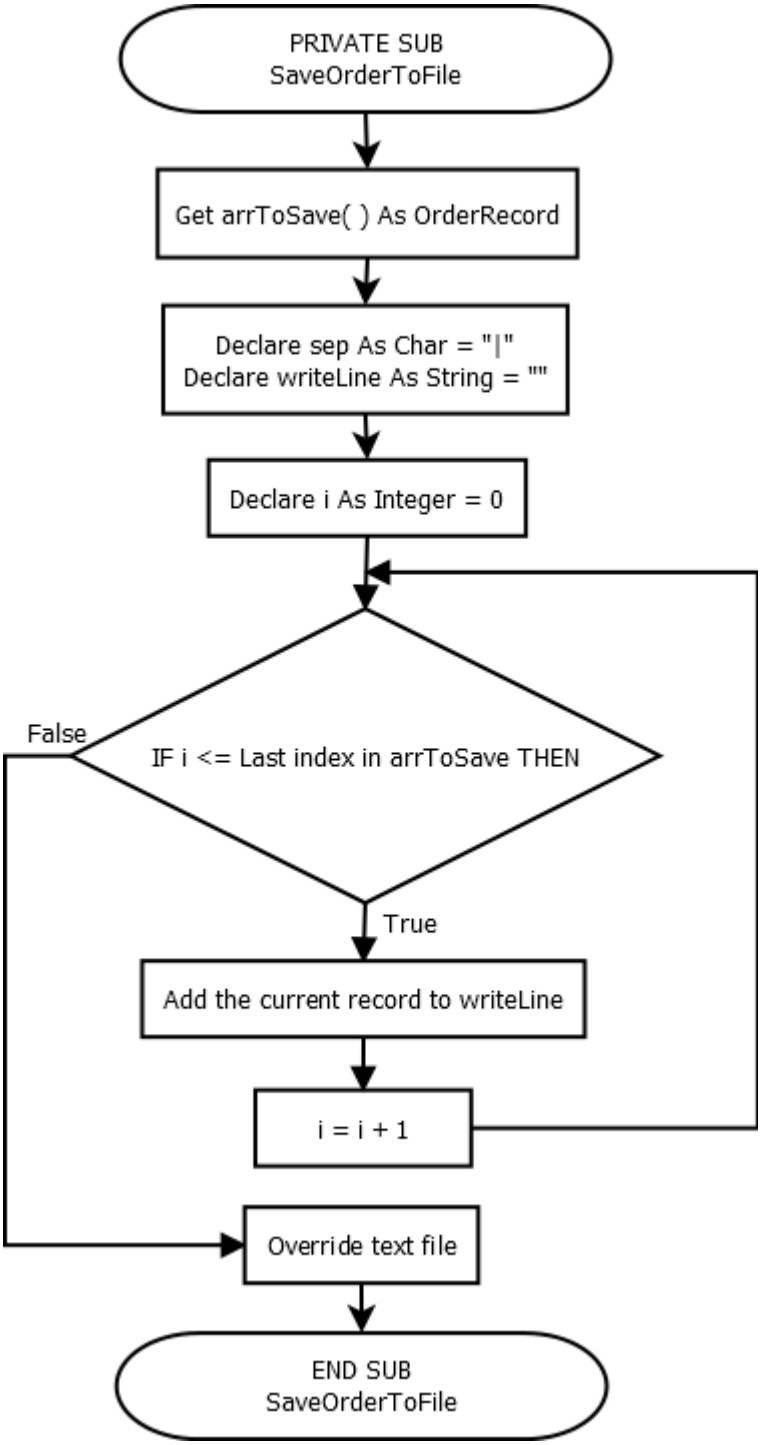

PUBLIC SUB
DeleteMenuItem
Get menuItemToDelete As MenuItemRecord
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
RemoveElementFromArray(menuItems, itemID of menuItemToDelete)
Declare i As Integer = 0
IF i <= Last index in
menuItems THEN
False
True
itemID of menuItems(i) = i
i = i + 1
SaveMenuToFile(menuItems)
END SUB
DeleteMenuItem

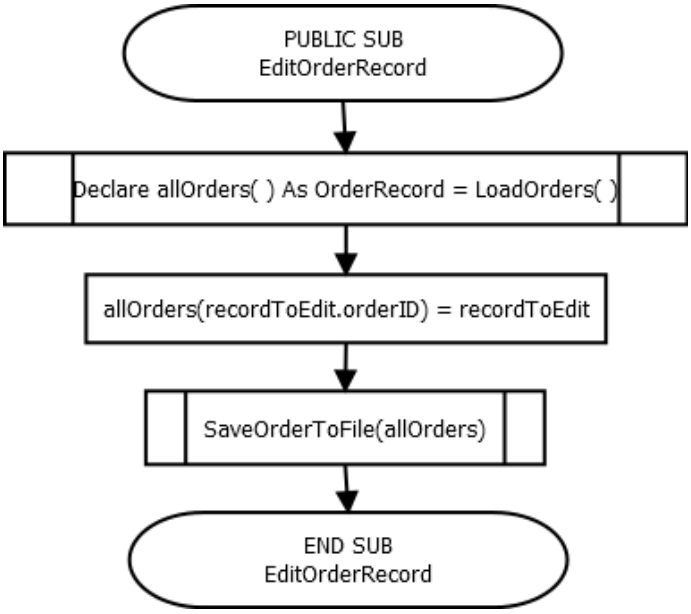

PUBLIC SUB
EditMenuItem
Get recordToEdit As MenuItemRecord
Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
menuItems(itemID of recordToEdit) = recordToEdit
SaveMenuToFile(menuItems)
END SUB
EditMenuItem

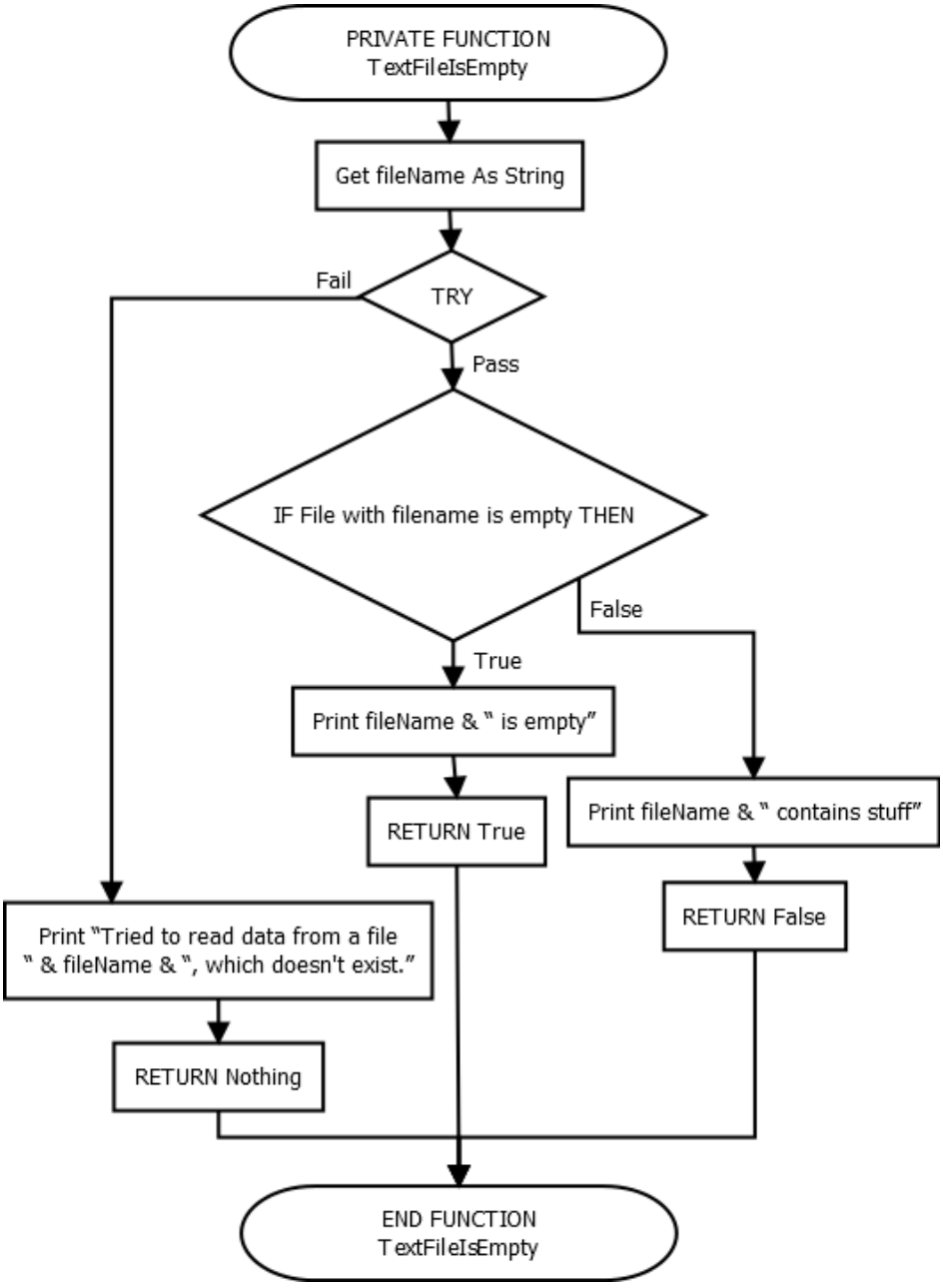

PUBLIC SUB
ClearMenuFile
Delete all text from the text file.
END SUB
ClearMenuFile


PUBLIC FUNCTION
LoadOrders
Declare arrToLoadInto(0) As OrderRecord
Declare nullElementRemoved As Boolean = False
Declare oRead As StreamReader
Declare lineIn As String
Declare tmp
Set oRead to file, opened for reading
IF there are still lines left to read
False
True
Set lineIn to current line
IF there are characters
on the line THEN
False
True
Split the line, separating the elements by the "|"
character and storing each field in tmp
Declare currentRecord As OrderRecord
orderID field of currentRecord = tmp(0)
orderDate field of currentRecord = tmp(1)
tableNumber field of currentRecord = tmp(2)
orderSubtotal field of currentRecord = tmp(3)
orderPaid field of currentRecord = tmp(4)
employee field of currentRecord = tmp(5)
customerComments field of currentRecord = tmp(6)
IF nullElementRemoved = False THEN
False
True
arrToLoadInto(0) = currentRecord
nullElementRemoved = True
AddElementToArray
(arrToLoadInto, currentRecord)
Close text file
IF nullElementRemoved
= True THEN
False
True
RETURN arrToLoadInto
Print "Tried to load from an empty file"
RETURN Nothing
END FUNCTION
LoadOrders

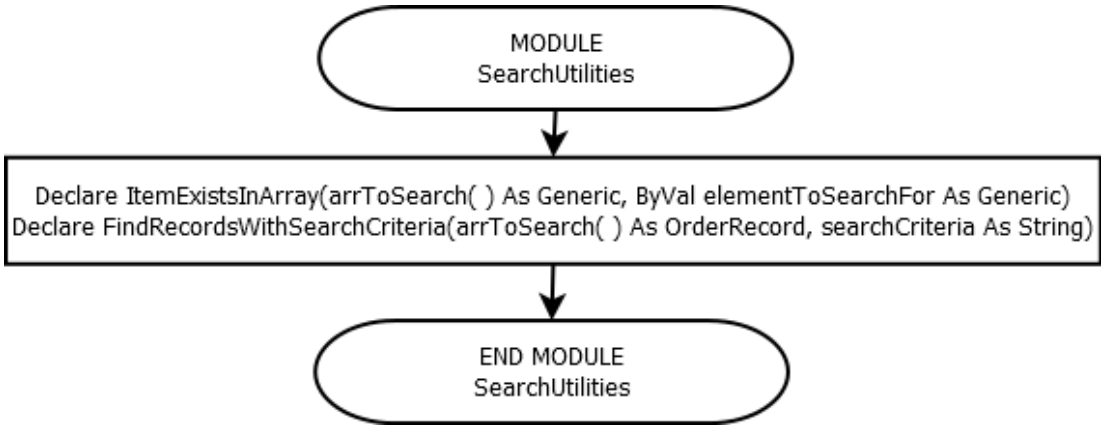

PUBLIC SUB
AddOrderRecord
Get orderDate As Date
Get tableNumber As Integer
Get orderSubtotal As Single
Get orderPaid As Boolean
Get employee As String
Get customerComments As String
Declare recordToAdd As OrderRecord
orderDate field of recordToAdd = orderDate
tableNumber field of recordToAdd = tableNumber
orderSubtotal field of recordToAdd = orderSubtotal
orderPaid field of recordToAdd = orderPaid
employee field of recordToAdd = employee
customerComments field of recordToAdd = customerComments
IF TextFileIsEmpty("sales.txt")
= True THEN
True
False
Declare nextID As Integer = 0
userID of recordToAdd = nextID
Declare arrRecords( ) As UserRecord and set to recordToAdd
SaveOrderToFile(arrRecords)
Declare arrRecords( ) As OrderRecord = LoadOrders( )
Declare nextID As Integer = Last index of arrRecords + 1
userID of recordToAdd = nextID
AddElementToArray(arrRecords, recordToAdd)
SaveOrderToFile(arrRecords)
END SUB
AddOrderRecord

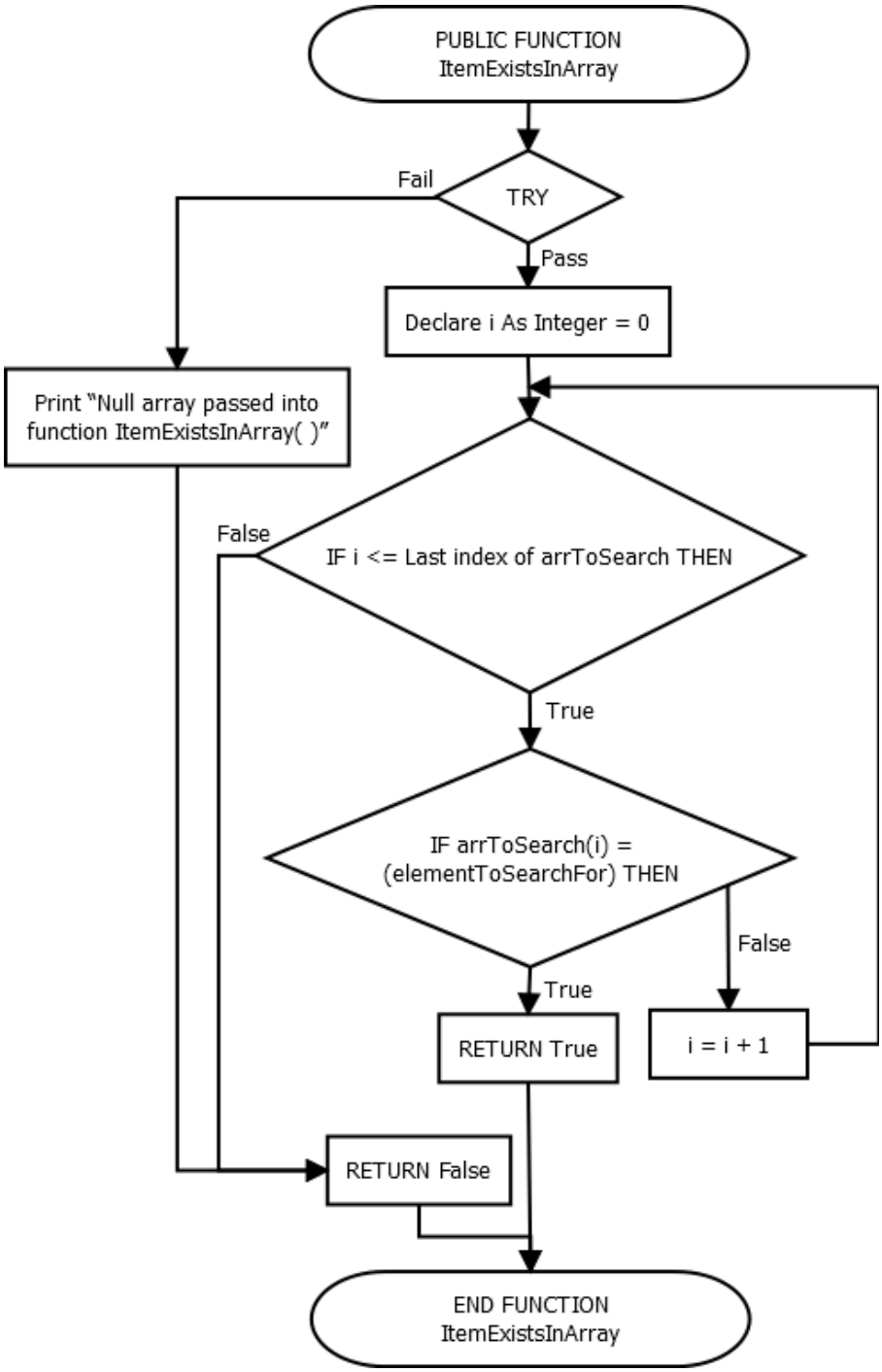

PRIVATE SUB
SaveOrderToFile
Get arrToSave( ) As OrderRecord
Declare sep As Char = "|"
Declare writeLine As String = ""
Declare i As Integer = 0
False
IF i <= Last index in arrToSave THEN
True
Add the current record to writeLine
i = i + 1
Override text file
END SUB
SaveOrderToFile

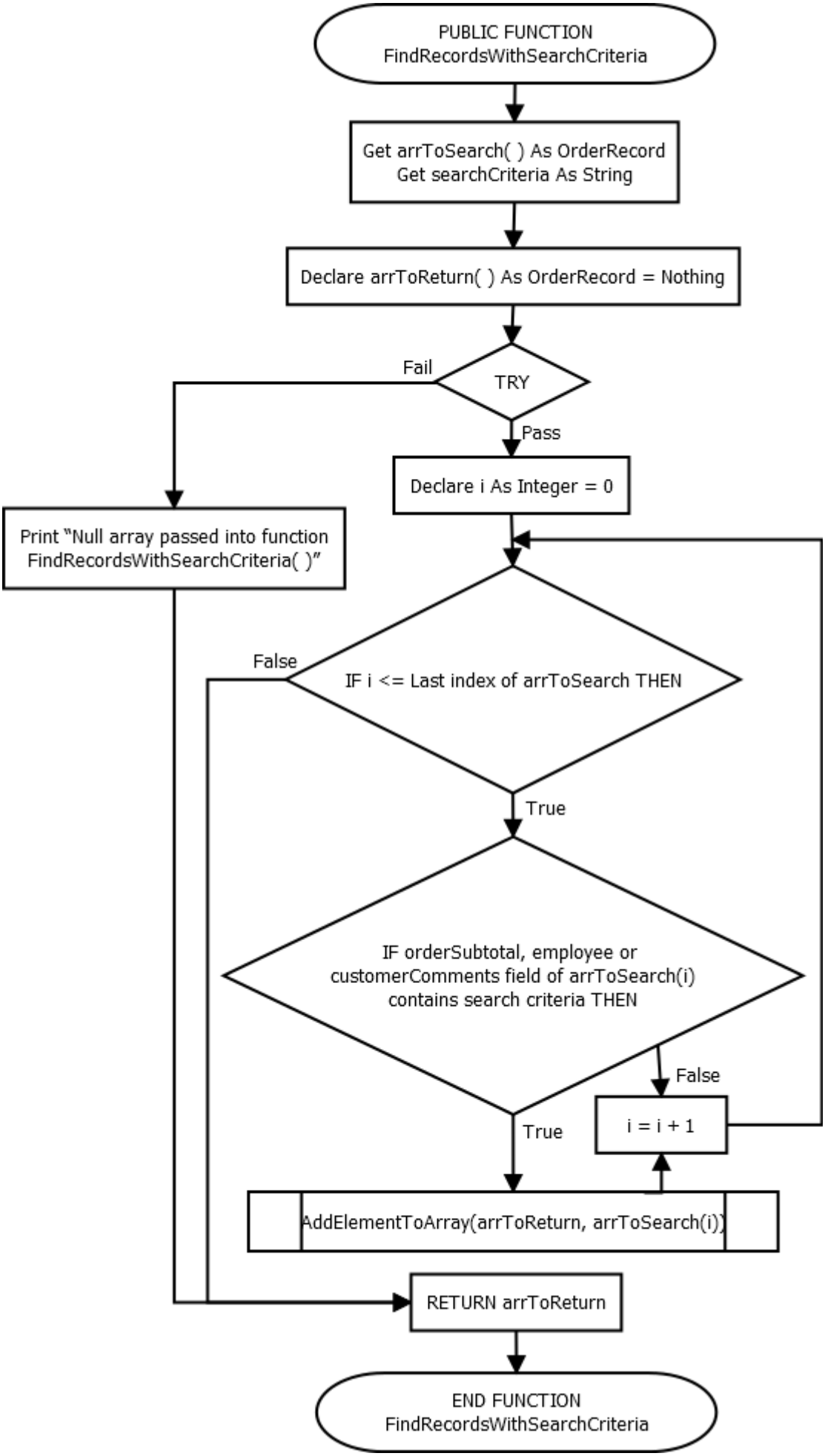

PUBLIC SUB
EditOrderRecord
Declare allOrders( ) As OrderRecord = LoadOrders( )
allOrders(recordToEdit.orderID) = recordToEdit
SaveOrderToFile(allOrders)
END SUB
EditOrderRecord


PRIVATE FUNCTION
TextFileIsEmpty
Get fileName As String
TRY
Fail
Pass
IF File with filename is empty THEN
False
True
Print fileName & " is empty"
RETURN True
Print fileName & " contains stuff"
RETURN False
Print "Tried to read data from a file
" & fileName & ", which doesn't exist."
RETURN Nothing
END FUNCTION
TextFileIsEmpty

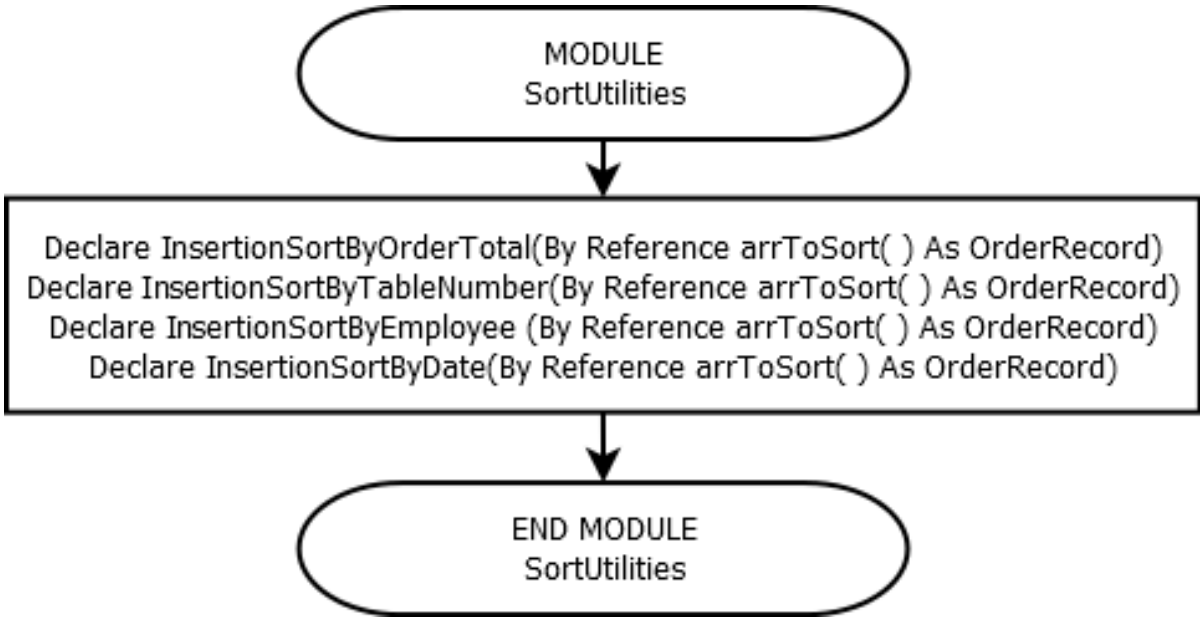
SEARCH UTILITIES

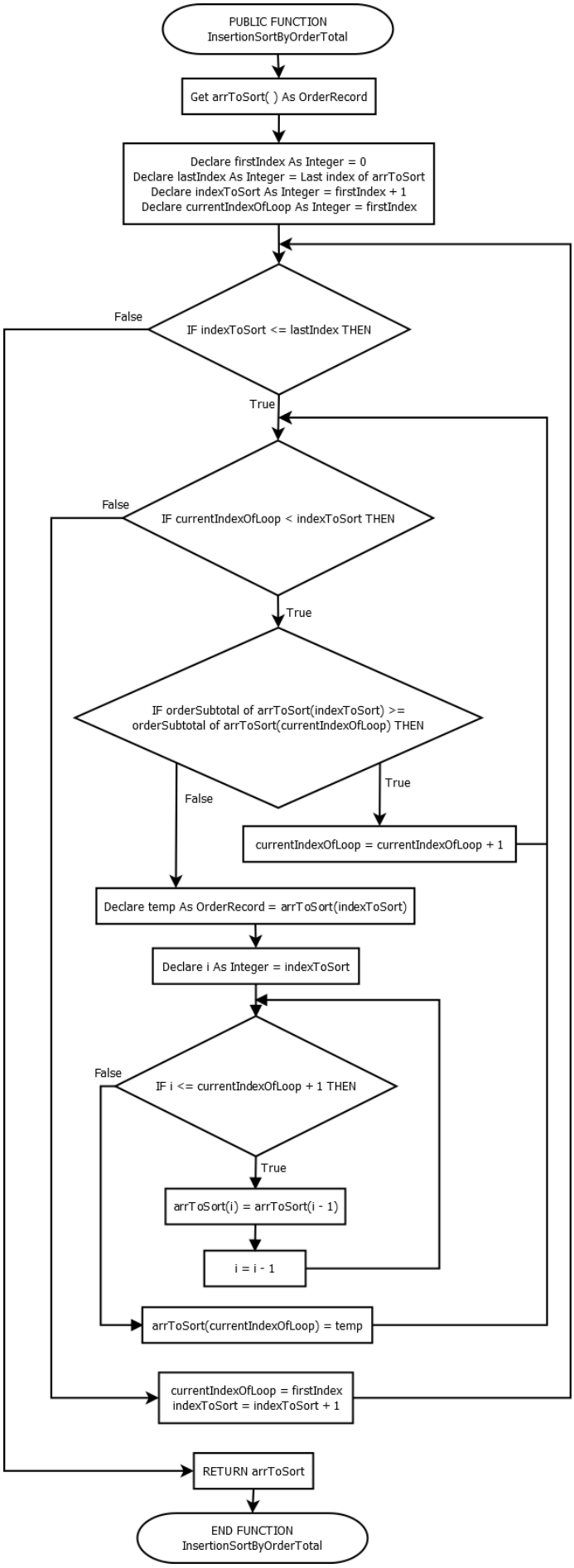


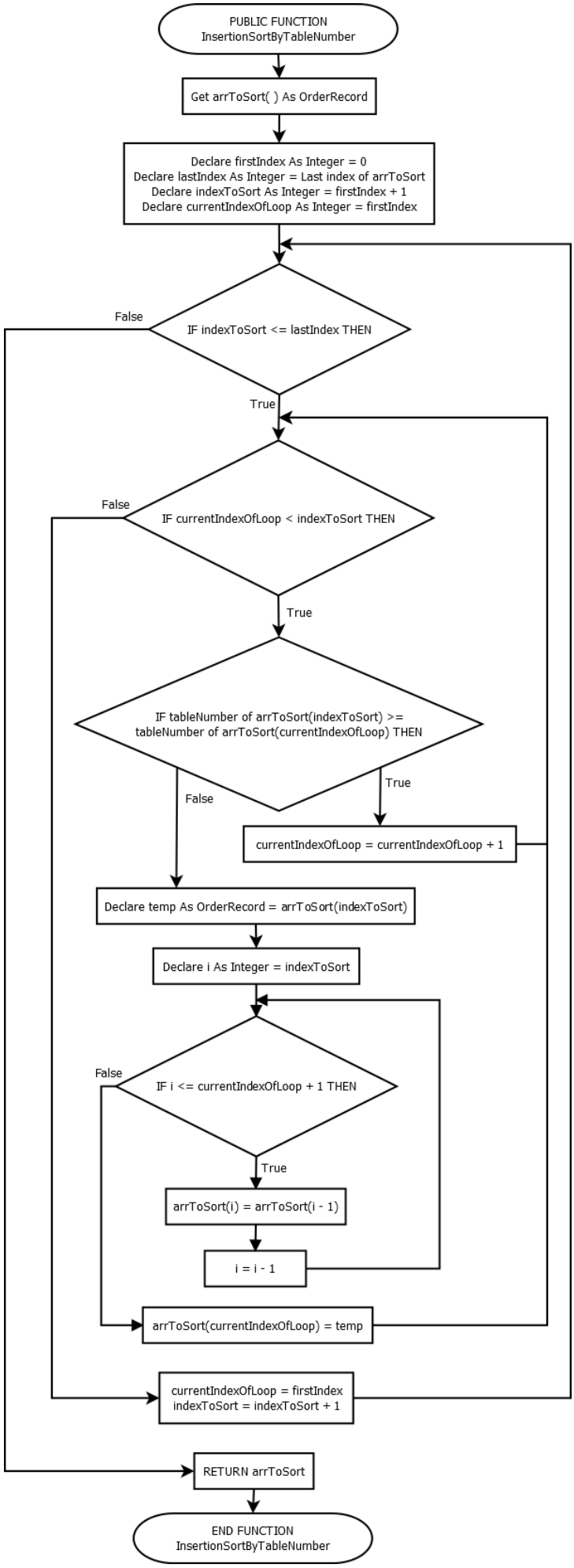


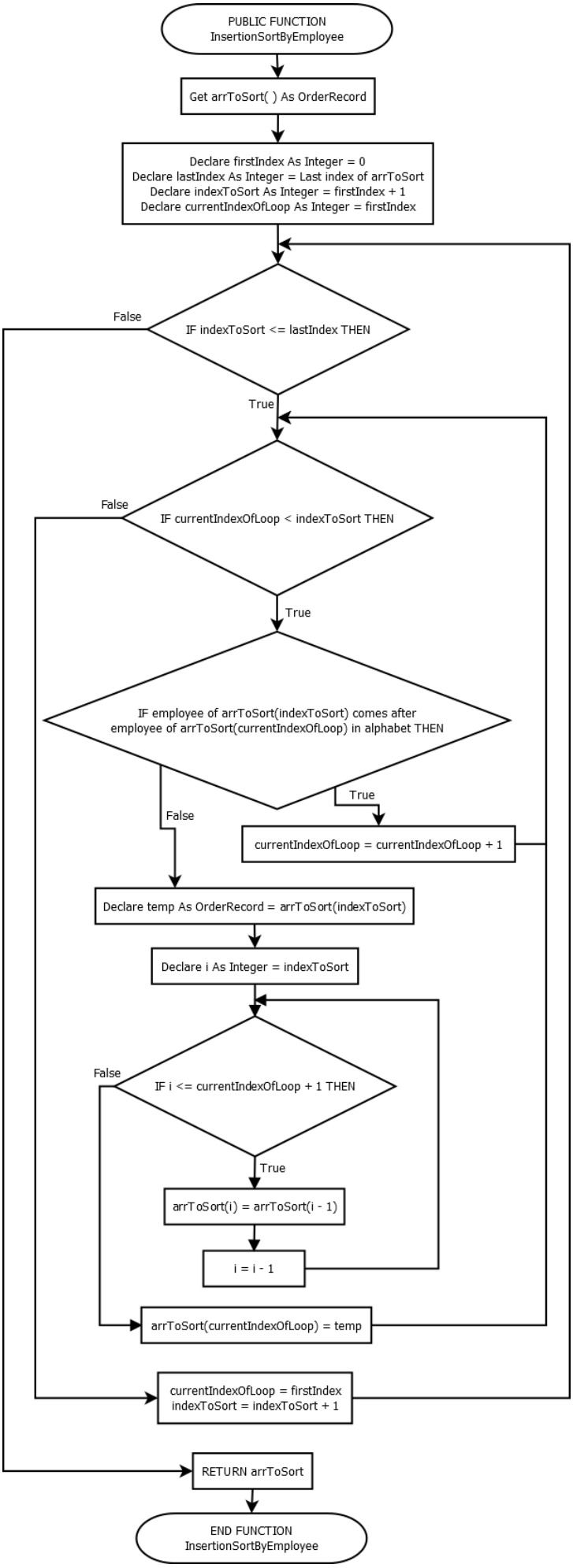

PUBLIC FUNCTION
FindRecordsWithSearchCriteria
Get arrToSearch( ) As OrderRecord
Get searchCriteria As String
Declare arrToReturn( ) As OrderRecord = Nothing
TRY
Fail
Pass
Declare i As Integer = 0
Print "Null array passed into function
FindRecordsWithSearchCriteria( )"
IF i <= Last index of arrToSearch THEN
False
True
IF orderSubtotal, employee or
customerComments field of arrToSearch(i)
contains search criteria THEN
False
True
i = i + 1
AddElementToArray(arrToReturn, arrToSearch(i))
RETURN arrToReturn
END FUNCTION
FindRecordsWithSearchCriteria

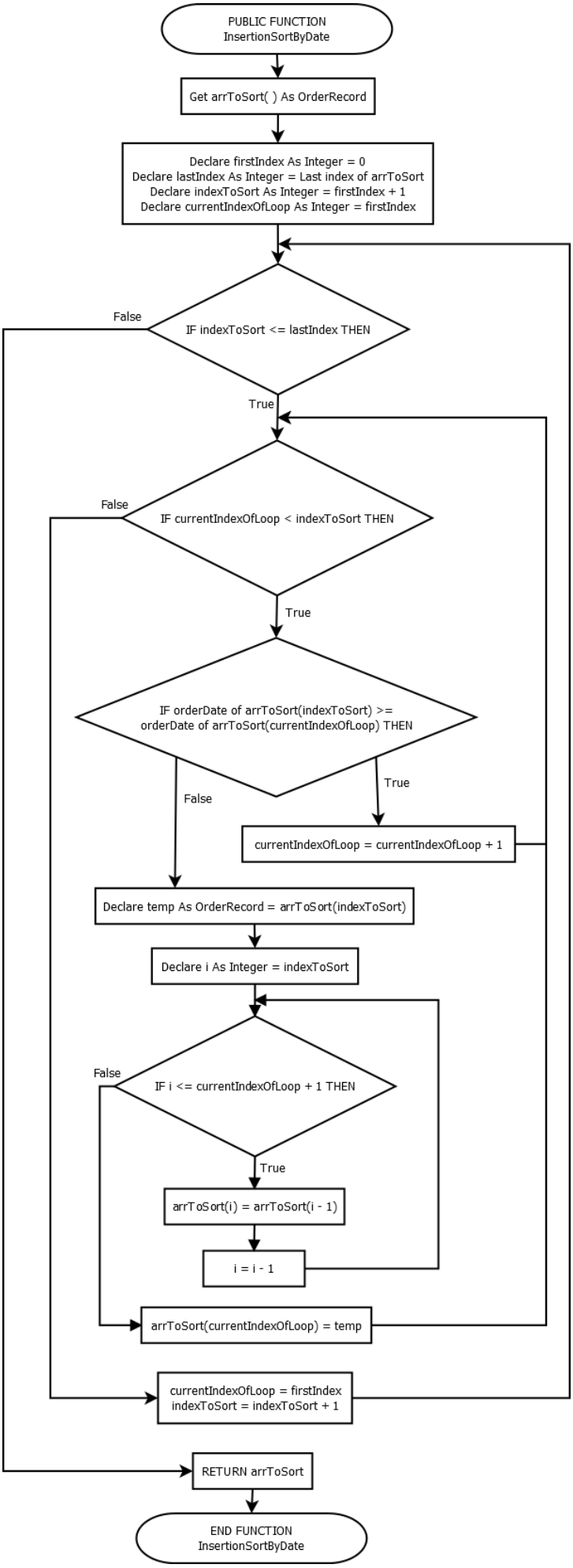
SORT UTILITIES



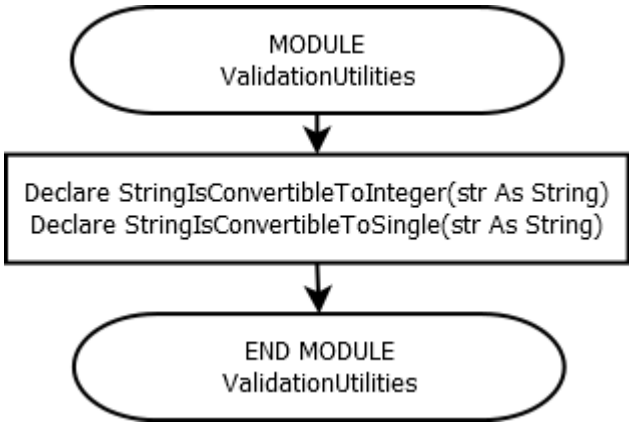

PUBLIC FUNCTION
InsertionSortByOrderTotal
Get arrToSort( ) As OrderRecord
Declare firstIndex As Integer = 0
Declare lastIndex As Integer = Last index of arrToSort
Declare indexToSort As Integer = firstIndex + 1
Declare currentIndexOfLoop As Integer = firstIndex
IF indexToSort <= lastIndex THEN
False
True
IF currentIndexOfLoop < indexToSort THEN
False
True
IF orderSubtotal of arrToSort(indexToSort) >=
orderSubtotal of arrToSort(currentIndexOfLoop) THEN
True
False
currentIndexOfLoop = currentIndexOfLoop + 1
Declare temp As OrderRecord = arrToSort(indexToSort)
Declare i As Integer = indexToSort
IF i <= currentIndexOfLoop + 1 THEN
False
True
arrToSort(i) = arrToSort(i - 1)
i = i - 1
arrToSort(currentIndexOfLoop) = temp
currentIndexOfLoop = firstIndex
indexToSort = indexToSort + 1
RETURN arrToSort
END FUNCTION
InsertionSortByOrderTotal

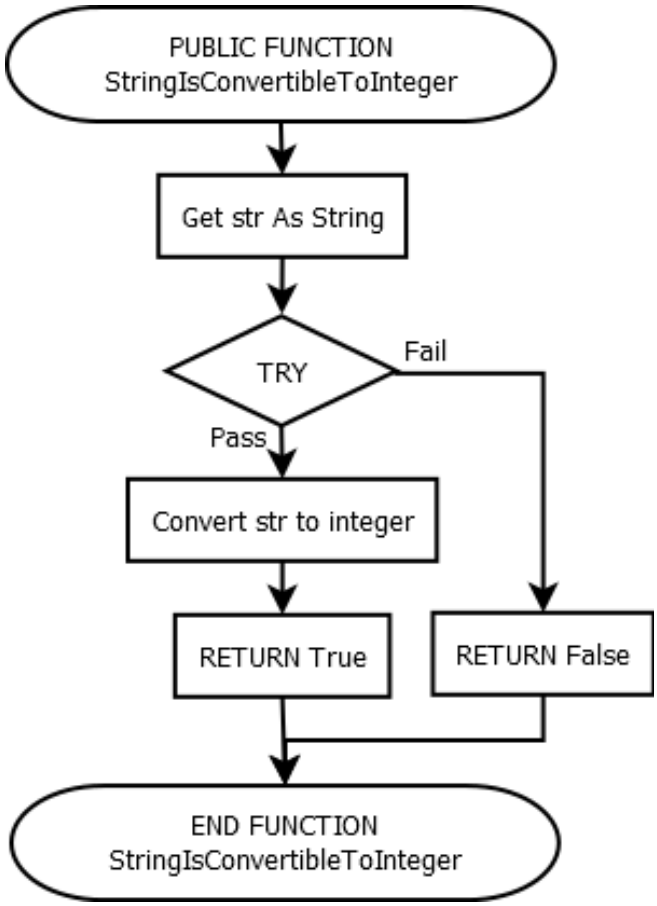

PUBLIC FUNCTION
InsertionSortByTableNumber
Get arrToSort( ) As OrderRecord
Declare firstIndex As Integer = 0
Declare lastIndex As Integer = Last index of arrToSort
Declare indexToSort As Integer = firstIndex + 1
Declare currentIndexOfLoop As Integer = firstIndex
IF indexToSort <= lastIndex THEN
False
True
IF currentIndexOfLoop < indexToSort THEN
False
True
IF tableNumber of arrToSort(indexToSort) >=
tableNumber of arrToSort(currentIndexOfLoop) THEN
True
False
currentIndexOfLoop = currentIndexOfLoop + 1
Declare temp As OrderRecord = arrToSort(indexToSort)
Declare i As Integer = indexToSort
IF i <= currentIndexOfLoop + 1 THEN
False
True
arrToSort(i) = arrToSort(i - 1)
i = i - 1
arrToSort(currentIndexOfLoop) = temp
currentIndexOfLoop = firstIndex
indexToSort = indexToSort + 1
RETURN arrToSort
END FUNCTION
InsertionSortByTableNumber

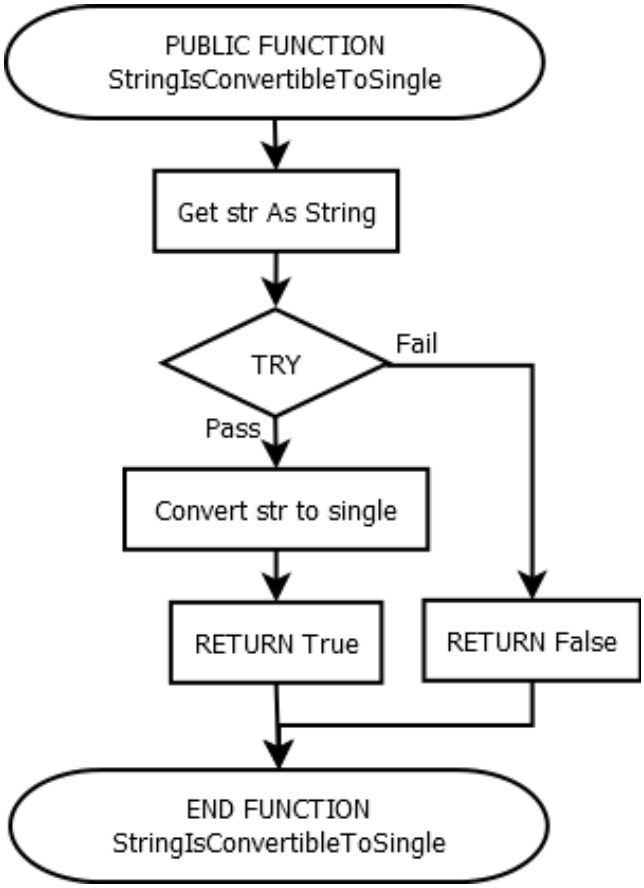

PUBLIC FUNCTION
InsertionSortByEmployee
Get arrToSort( ) As OrderRecord
Declare firstIndex As Integer = 0
Declare lastIndex As Integer = Last index of arrToSort
Declare indexToSort As Integer = firstIndex + 1
Declare currentIndexOfLoop As Integer = firstIndex
IF indexToSort <= lastIndex THEN
False
True
IF currentIndexOfLoop < indexToSort THEN
False
True
IF employee of arrToSort(indexToSort) comes after
employee of arrToSort(currentIndexOfLoop) in alphabet THEN
True
False
currentIndexOfLoop = currentIndexOfLoop + 1
Declare temp As OrderRecord = arrToSort(indexToSort)
Declare i As Integer = indexToSort
IF i <= currentIndexOfLoop + 1 THEN
False
True
arrToSort(i) = arrToSort(i - 1)
i = i - 1
arrToSort(currentIndexOfLoop) = temp
currentIndexOfLoop = firstIndex
indexToSort = indexToSort + 1
RETURN arrToSort
END FUNCTION
InsertionSortByEmployee


PUBLIC FUNCTION
InsertionSortByDate
Get arrToSort( ) As OrderRecord
Declare firstIndex As Integer = 0
Declare lastIndex As Integer = Last index of arrToSort
Declare indexToSort As Integer = firstIndex + 1
Declare currentIndexOfLoop As Integer = firstIndex
IF indexToSort <= lastIndex THEN
False
True
IF currentIndexOfLoop < indexToSort THEN
False
True
IF orderDate of arrToSort(indexToSort) >=
orderDate of arrToSort(currentIndexOfLoop) THEN
True
False
currentIndexOfLoop = currentIndexOfLoop + 1
Declare temp As OrderRecord = arrToSort(indexToSort)
Declare i As Integer = indexToSort
IF i <= currentIndexOfLoop + 1 THEN
False
True
arrToSort(i) = arrToSort(i - 1)
i = i - 1
arrToSort(currentIndexOfLoop) = temp
currentIndexOfLoop = firstIndex
indexToSort = indexToSort + 1
RETURN arrToSort
END FUNCTION
InsertionSortByDate

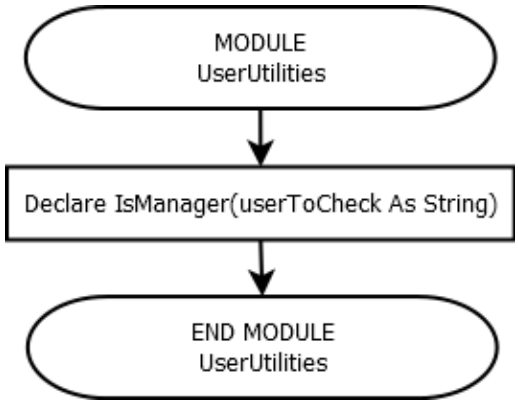
VALIDATION UTILITIES

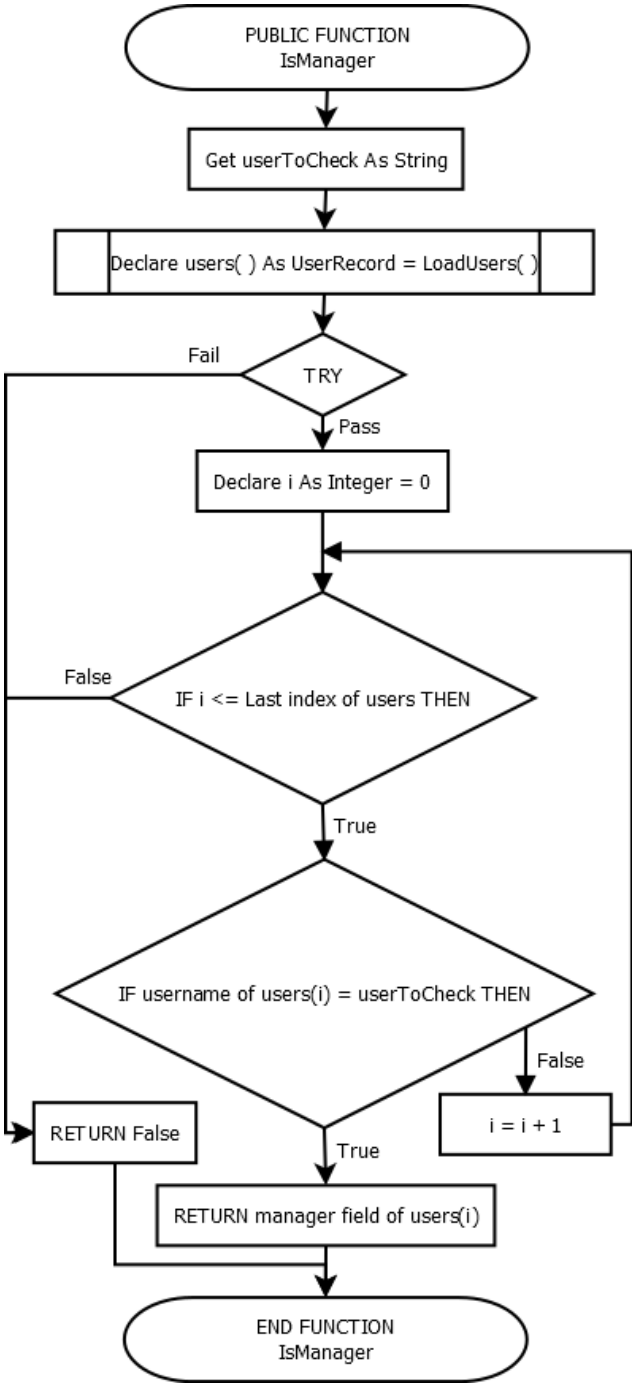





PUBLIC FUNCTION
StringIsConvertibleToSingle
Get str As String
TRY
Fail
Pass
Convert str to single
RETURN True
RETURN False
END FUNCTION
StringIsConvertibleToSingle

USER UTILITIES





PSEUDOCODE

LOGIN

```
Declare Public currentUser As String

PRIVATE SUB ClickedButtonSubmit( ) When submit button is clicked
   IF CheckLogin(Text component of iptUsername, Text component of iptPassword) THEN
      currentUser = iptUsername.Text
      ClearInputs( )
      Hide current form
      Show main form
   ELSE
      Print "Username or Password incorrect."
      ClearInputs( )
   END IF
END SUB

PRIVATE FUNCTION CheckLogin(inputUsername As String, inputPassword As String)
   Declare users( ) As UserRecord and set to value of LoadUsers( )
   TRY
      FOR i = 0 TO last index of users
         IF Username part of users(i) = inputUsername AND Password part of users(i) = inputPassword
THEN
            RETURN True
         END IF
      NEXT
   CATCH
   END TRY
   RETURN False
END FUNCTION

PRIVATE SUB ClearInputs( )
   Clear text of iptUsername
   Clear text of iptPassword
END SUB

PRIVATE SUB ClickedButtonCreateNewLogin( ) When create login button is clicked
   Show formNewLogin
   Hide current form
END SUB

PRIVATE SUB ClickedButtonHelp( ) When help button is clicked
   Show Customised Help Message
END SUB

PRIVATE SUB ClickedButtonQuit( ) When quit button is clicked
   End
END SUB
```

NEW LOGIN

```
Declare Private Constant maxUserLength As Integer and set to 8

PRIVATE SUB ClickedButtonCreateUser( ) When create user button is clicked
   IF There is text in iptUsername AND There is text in iptPassword THEN
      IF Length of text in iptUsername < maxUserLength THEN
         IF Text in iptPassword matches text in iptConfirmPassword THEN
            IF NOT UserAlreadyExists(Text in iptUsername) THEN
               AddUserRecord(Text in iptUsername, Text in iptPassword, togManager is selected?)
               Print "New user created successfully"
            ELSE
               Print "User already exists in database. Please choose a new username."
            END IF
         ELSE
            Print "Passwords don't match."
         END IF
      ELSE
         Print "Please choose a username that is less than " & maxUserLength & " characters long"
      END IF
   ELSE
      Print "Please insert a username and password"
   END IF
END SUB

PRIVATE FUNCTION UserAlreadyExists(userToSearch As String)
   Declare users ( ) As UserRecord and set to value of LoadUsers ( )
   TRY
      Declare usernames (Last index of users) As String
      FOR i = 0 TO Last index of users
         usernames (i) = users (i).username
      NEXT
      IF ItemExistsInArray(usernames, userToSearch) THEN
         RETURN True
      ELSE
         RETURN False
      END IF
   CATCH
      RETURN False
   END TRY
END FUNCTION

PRIVATE SUB ClickedButtonBack( ) When back button is clicked
   Show formLogin
   Close current form
END SUB
```

MAIN FORM

```
PRIVATE SUB LoadedMainForm( ) When formMain is loaded
   Update welcome text
   IF NOT Current user is a manager THEN
      Move btnSwitchUser to better position
      Move btnQuit to better position
      Make btnViewSales invisible
      Make btnChangeMenu invisible
   END IF
END SUB

PRIVATE SUB ClickedButtonTakeOrders( ) When btnTakeOrders is clicked
   Show formTakeOrders
   Close current form
END SUB

PRIVATE SUB ClickedButtonPayForOrders( ) When btnPayForOrders is clicked
   Show formPayOrders
   Close current form
END SUB

PRIVATE SUB ClickedButtonChangeMenu( ) When btnChangeMenu is clicked
   Show formEditMenu
   Close current form
END SUB

PRIVATE SUB ClickeButtonViewSales( ) When btnViewSales is clicked
   Show formViewSales
   Close current form
END SUB

PRIVATE SUB ClickedButtonSwitchUser( ) When btnSwitchUser is clicked
   Show formLogin
   Close current form
END SUB

PRIVATE SUB ClickedButtonQuit( ) When btnQuit is clicked
   End Program
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

TAKE ORDERS

```
Declare Private selectedTableNumber As Integer = 1
Declare Private orderItemIndexes( ) As Integer

PRIVATE SUB LoadedFormTakeOrders( ) When formTakeOrders is loaded
   LoadMenu( )
END SUB

PRIVATE SUB ChangedTableNumber(sender As Object, e As Event) When togTable1- togTable16
changes state
   IF sender is checked THEN
      selectedTableNumber = sender tag + 1
   END IF
END SUB

PRIVATE SUB LoadMenu( )
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   FOR EACH MenuItemRecord item IN menuItems
      IF item is food THEN
         Add the item to the food combo box.
      ELSE
         Add the item to the beverages combo box.
      END IF
   NEXT
END SUB

PRIVATE SUB ClickedButtonAddFood( )When btnAddFood is clicked
   IF an item is slected in comboFood THEN
      IF StringIsConvertibleToInteger(Text in txtFoodAmount) THEN
         FOR i = 0 TO Text in txtFoodAmount converted to integer - 1
            AddElementToArray(orderItemIndexes, GetMenuItemIndex(Selected food item))
         NEXT
         UpdateOrderList(orderItemIndexes)
         Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) in currency format
      ELSE
         Print “Please select an amount of the food you want to add."
      END IF
   ELSE
      Display “Please select the food item that you want to add.”
   END IF
END SUB

PRIVATE SUB ClickedButtonAddBeverage( )When btnAddBeverage is clicked
   IF an item is slected in comboBeverage THEN
      IF StringIsConvertibleToInteger(Text in txtBeverageAmount) THEN
         FOR i = 0 TO Text in txtBeverageAmount converted to integer - 1
            AddElementToArray(orderItemIndexes, GetMenuItemIndex(Selected beverage item))
         NEXT
         UpdateOrderList(orderItemIndexes)
         Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) in currency format
      ELSE
         Print “Please select an amount of the beverage you want to add."
      END IF
   ELSE
      Display “Please select the beverage that you want to add.”
   END IF
END SUB

PRIVATE FUNCTION GetMenuItemIndex(menuItemName As String)
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
```

```
    FOR EACH MenuItemRecord item IN menuItems
      IF item name = menuItemName THEN
        RETURN item ID
      END IF
    NEXT
    RETURN Nothing
  END FUNCTION

  PRIVATE SUB UpdateOrderList (orderItemIndexes( ) As Integer)
    Declare menuItems( ) As MenuItemRecord = LoadMenuItems
    Clear all items in both list boxes.
    TRY
      FOR i = 0 TO Last index of orderItemIndexes
        Add current item name to each list box
      NEXT
    CATCH
    END TRY
  END SUB

  PRIVATE FUNCTION CalculateOrderSubtotal (orderItemIndexes( ) As Integer)
    Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
    Declare subtotal As Single = 0
    TRY
      FOR i = 0 TO Last index of orderItemIndexes
        Add the current item's price to the subtotal.
      NEXT
    CATCH
    END TRY
    RETURN subtotal
  END FUNCTION

  PRIVATE SUB ClickedButtonResetList( ) When btnResetList is clicked
    Clear all items from the list box.
    PurgeArray(orderItemIndexes)
    UpdateOrderList(orderItemIndexes)
    Text in lblOrderTotalAmount = CalculateOrderSubtotal(orderItemIndexes) formatted as currency
  END SUB

  PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
    Show main form
    Close current form
  END SUB

  PRIVATE SUB ClickedButtonConfirmOrder( ) When btnConfirmOrder is clicked
    IF NOT CalculateOrderSubtotal(orderItemIndexes) = 0 THEN
      AddOrderRecord(Current date, selectedTableNumber, CalculateOrderSubtotal(orderItemIndexes),
False, Current user, "None")
      PrintReceipt( )
      Show main form
      Close current form
    ELSE
      Print “Can't confirm an order that doesn't contain any items”
    END IF
  END SUB

  PRIVATE SUB PrintReceipt( )
    Declare printString As String
    Add appropriate order data to printString
    FOR EACH item IN lstItems
      Add item name to end of printString
    NEXT
```

```
   Add order subtotal data to printString
   Display printString to simulate a receipt.
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

PAY FOR ORDERS

```
Declare Private currentUnpaidOrders( ) As OrderRecord

PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
   Show main form
   Close current form
END SUB

PRIVATE SUB LoadedFormPayForOrders( ) When formPayOrders is loaded
   currentUnpaidOrders = GetCurrentUnpaidOrders( )
   DisplayArrayInListBox(currentUnpaidOrders)
END SUB

PRIVATE FUNCTION GetCurrentUnpaidOrders( )
   Declare allOrders( ) As OrderRecord = LoadOrders( )
   Declare ordersToReturn( ) As OrderRecord = Nothing
   FOR i = 0 TO Last index of allOrders
      IF Date component of allOrders(i) = Current date AND orderPaid component of allOrders(i) = False
THEN
         AddElementToArray(ordersToReturn, allOrders(i))
      END IF
   NEXT
   RETURN ordersToReturn
END FUNCTION

PRIVATE SUB DisplayArrayInListBox(arrToDisplay( ) As OrderRecord)
   Clear all items in list box.
   TRY
      FOR EACH item IN arrToDisplay
         Add item to the list box.
      NEXT
      lstOrders.Enabled = True
      btnSortByTable.Enabled = True
      btnSortByTotal.Enabled = True
      btnConfirmPayment.Enabled = True
   CATCH
      Display “No orders from today are still unpaid” in list box
      lstOrders.Enabled = False
      btnSortByTable.Enabled = False
      btnSortByTotal.Enabled = False
      btnConfirmPayment.Enabled = False
   END TRY
END SUB

PRIVATE SUB ClickedButtonSortByTable( ) When btnSortByTable is clicked
   currentUnpaidOrders = InsertionSortByTableNumber(currentUnpaidOrders)
   DisplayArrayInListBox(currentUnpaidOrders)
END SUB

PRIVATE SUB ClickedButtonSortByOrderTotal( ) When btnSortByTotal is clicked
   currentUnpaidOrders = InsertionSortByOrderTotal(currentUnpaidOrders)
   Reverse currentUnpaidOrders array
   DisplayArrayInListBox(currentUnpaidOrders)
END SUB

PRIVATE SUB ClickedButtonConfirmPayment( ) When btnConfirmPayment is clicked
   IF ItemIsSelected( ) THEN
      Declare orderIDToEdit as Integer and set to the orderID of the currently selected record.
      Declare allOrders( ) As OrderRecord = LoadOrders( )
      Change orderPaid flag of the appropriate record to true.
```

```
      IF text in iptCustomerComments THEN
         Add any customer comments to the appropriate record.
      END IF

      EditOrderRecord(allOrders(orderIDToEdit))
      currentUnpaidOrders = GetCurrentUnpaidOrders( )
      DisplayArrayInListBox(currentUnpaidOrders)
      Set text in iptCustomerComments to “”
      Print “Order Marked as Paid.”
   ELSE
      Print “Please select an order to confirm.”
   END IF
END SUB

PRIVATE FUNCTION ItemIsSelected( )
   IF No item is selected THEN
      RETURN False
   ELSE
      RETURN True
   END IF
END FUNCTION

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message.
END SUB
```

EDIT MENU

```
PRIVATE SUB LoadedFormEditMenu( ) When formEditMenu is loaded
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   FOR EACH MenuItemRecord item IN menuItems
      Add item to lstMenuItems, with price in currency format.
   NEXT
END SUB

PRIVATE SUB ClickedButtonAddMenuItem( ) When btnAddMenuItem is clicked
   Show formAddMenuItem
   Close current form
END SUB

PRIVATE SUB ClickedButtonEditMenuItem( ) When btnEditMenuItem is clicked
   IF ItemIsSelected( ) THEN
      Show formEditMenuItem
      formEditMenuItem.GetItemToEditIndex(Selected index in list box)
      Close current form
   ELSE
      Print “Please select an item from the list to edit”
   END IF
END SUB

PRIVATE SUB ClickedButtonDeleteMenuItem( ) When btnDeleteMenuItem is clicked
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   IF There are items in the list of menu items THEN
      IF ItemIsSelected( ) THEN
         DeleteMenuItem(menuItems(Selected item index))
         Remove item from list box.
      ELSE
         Print “Please select an item from the list to delete”
      END IF
   ELSE
      Print “Please create a menu item before trying to delete one.”
   END IF
END SUB

PRIVATE SUB ClickedButtonResetMenu( ) When btnResetMenu is clicked
   ClearMenuFile( )
   Clear menu items list box.
END SUB

PRIVATE FUNCTION ItemIsSelected( )
   IF No item is selected THEN
      RETURN False
   ELSE
      RETURN True
   END IF
END FUNCTION

PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
   Show main form
   Close current form
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

ADD MENU ITEM

```
PRIVATE SUB ClickedButtonConfirmAdd( ) When btnAddItem is clicked
   IF There is text in iptItemName THEN
      IF StringIsConvertibleToSingle(Text in iptItemPrice) THEN
         Declare itemPrice As Single = text in iptItemPrice formatted to 2 d.p.
         AddMenuItemRecord(Text in iptItemName, itemPrice, togFood is checked)
         Print “New menu item added.”
         Set text of iptItemName to “”
         Set text of iptItemPrice to “”
      ELSE
         Print “Price field is invalid. Make sure that it only contains numbers.”
      END IF
   ELSE
      Print “Please insert an item name.”
   END IF
END SUB

PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
   Show formEditMenu
   Close current form
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

EDIT MENU ITEM

```
Declare Private itemToEditIndex As Integer = -1

PUBLIC SUB GetItemToEditIndex(index As Integer)
   itemToEditIndex = index
   DisplayDataToEdit( )
END SUB

PRIVATE SUB ClickedButtonConfirmEdit( ) When btnEditItem is clicked
   Declare newRecord As MenuItemRecord
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   IF StringIsConvertibleToSingle(Text in iptItemPrice) THEN
      Set itemID of newRecord to itemToEditIndex
      Set itemName of newRecord to text in lblItem
      Set itemPrice of newRecord to text in iptItemPrice
      Set isFood field of newRecord to isFood field of menuItems(itemToEditIndex)
      EditMenuItem(newRecord)
      Show formEditMenu
      Close current form
   ELSE
      Print "Price field is invalid. Make sure that it only contains numbers."
   END IF
END SUB

PRIVATE SUB DisplayDataToEdit( )
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   Set text of lblItem to itemName of menuItems(itemToEditIndex)
   Set text of iptItemPrice to itemPrice of menuItems(itemToEditIndex), formatted to 2 d.p.
END SUB

PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
   Show formEditMenu
   Close current form
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

VIEW SALES

```
Declare currentlyDisplayedItems( ) As OrderRecord = LoadOrders( )

PRIVATE SUB LoadedFormViewSales( ) When formViewSales is loaded
   DisplayArrayInSalesListBox(LoadOrders( ))
END SUB

PRIVATE SUB ChangedFilterCriteria( ) When iptSearch is changed, dateFrom is changed, dateTo is
changed
   IF Date in dateFrom <= Date in dateTo THEN
      currentlyDisplayedItems = FindSalesWithinDateRange(Date in dateFrom, Date in dateTo)
      currentlyDisplayedItems = FindRecordsWithSearchCriteria(currentlyDisplayedItems, Text in
iptSearch)
      DisplayArrayInSalesListBox(currentlyDisplayedItems)
   ELSE
      Print "Please ensure that the start date is before the end date."
      Set value of dateFrom to value of dateTo
   END IF
END SUB

PRIVATE SUB DisplayArrayInSalesListBox(arrToDisplay( ) As OrderRecord)
   Clear all items from list box.
   TRY
      FOR i = 0 TO Last index in arrToDisplay
      Add data from arrToDisplay(i) to list box
      NEXT
      lstSales.Enabled = True
      btnInspectRecord.Enabled = True
      btnSortDate.Enabled = True
      btnSortEmployee.Enabled = True
      btnSortTotal.Enabled = True
   CATCH
      Print "No sales found that match the given criteria" in list box
      lstSales.Enabled = False
      btnInspectRecord.Enabled = False
      btnSortDate.Enabled = False
      btnSortEmployee.Enabled = False
      btnSortTotal.Enabled = False
   END TRY
END SUB

PRIVATE SUB ClickedButtonBack( ) When btnBack is clicked
   Show main form
   Close current form
END SUB

PRIVATE FUNCTION FindSalesWithinDateRange(startDate As Date, endDate As Date)
   Declare allSales( ) As OrderRecord = LoadOrders( )
   Declare salesToReturn( ) As OrderRecord
   TRY
      FOR EACH sale IN allSales
         IF orderDate component of sale >= startDate AND orderDate component of sale <= endDate THEN
            AddElementToArray(salesToReturn, sale)
         END IF
      NEXT
   CATCH
   END TRY
   RETURN salesToReturn
END FUNCTION
```

```
PRIVATE SUB ClickedButtonInspectRecord( ) When btnInspectRecord is clicked
   IF ItemIsSelected( ) THEN
      Declare allOrders( ) As OrderRecord = LoadOrders( )
      Declare orderIDToDisplay and set to the orderID of the item at the current index of the list box.
      DisplayItemInfo(allOrders(orderIDToDisplay))
   ELSE
      Print "Please select an item to inspect"
   END IF
END SUB

PRIVATE SUB DisplayItemInfo(itemToDisplay As OrderRecord)
   Print sale data in a popup window.
END SUB

PRIVATE FUNCTION ItemIsSelected( )
   IF Item isn't selected THEN
      RETURN False
   ELSE
      RETURN True
   END IF
END FUNCTION

PRIVATE SUB ClickedButtonSortByOrderTotal( ) When btnSortTotal is clicked
   currentlyDisplayedItems = InsertionSortByOrderTotal(currentlyDisplayedItems)
   Reverse array currentlyDisplayedItems
   DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB

PRIVATE SUB ClickedButtonSortByEmployee( ) When btnSortEmployee is clicked
   currentlyDisplayedItems = InsertionSortByEmployee(currentlyDisplayedItems)
   DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB

PRIVATE SUB ClickedButtonSortByOrderDate( ) When btnSortDate is clicked
   currentlyDisplayedItems = InsertionSortByDate(currentlyDisplayedItems)
   DisplayArrayInSalesListBox(currentlyDisplayedItems)
END SUB

PRIVATE SUB ClickedButtonHelp( ) When btnHelp is clicked
   Display help message
END SUB
```

ARRAY UTILITIES

```
PUBLIC SUB AddElementToArray(By Reference arrToAddTo( ) As Generic, elementToAdd As Generic)
   TRY
      Declare nextID As Integer = Last index of arrToAddTo + 1
      Open up new spot in array, with index nextID
      arrToAddTo(nextID) = elementToAdd
   CATCH
      Declare arrToReturn(0) As Generic
      arrToReturn(0) = elementToAdd
      arrToAddTo = arrToReturn
   END TRY
END SUB

PUBLIC SUB RemoveElementFromArray(By Reference arrToRemoveFrom( ) As Generic,
indexOfElementToRemove As Integer)
   TRY
      Declare arrToReturn(Last index of arrToRemoveFrom - 1) As Generic
      FOR i = 0 TO Last index in arrToRemoveFrom
         IF i < indexOfElementToRemove THEN
            arrToReturn(i) = arrToRemoveFrom(i)
         ELSE IF i > indexOfElementToRemove THEN
            arrToReturn(i - 1) = arrToRemoveFrom(i)
         END IF
      NEXT
      arrToRemoveFrom = arrToReturn
   CATCH
      Print “Index doesn't exist in arrToRemoveFrom”
   END TRY
END SUB

PUBLIC SUB PurgeArray(By Reference arrToPurge( ) As Generic)
   Set arrToPurge to nothing
END SUB
```

DATA UTILITIES

```
Declare Public Structure UserRecord
   Declare userID As Integer
   Declare username As String
   Declare password As String
   Declare manager As Boolean
End Structure

Declare Public Structure MenuItemRecord
   Declare itemID As Integer
   Declare itemName As String
   Declare itemPrice As Single
   Declare isFood As Boolean
End Structure

Declare Public Structure OrderRecord
   Declare orderID As Integer
   Declare orderDate As Date
   Declare tableNumber As Integer
   Declare orderSubtotal As Single
   Declare orderPaid As Boolean
   Declare employee As String
   Declare customerComments As String
End Structure

Declare Public filePath As String = File path of application

PUBLIC FUNCTION LoadUsers( )
   Declare arrToLoadInto(0) As UserRecord
   Declare nullElementRemoved As Boolean = False
   Declare oRead As StreamReader
   Declare lineIn As String
   Declare tmp
   Set oRead to file, opened for reading
   WHILE there are still lines left to read
      Set lineIn to current line
      IF there are characters on the line THEN
         Split the line, separating the elements by the "|" character and storing each field in tmp
         Declare currentRecord As UserRecord
         userID field of currentRecord = tmp(0)
         username field of currentRecord = tmp(1)
         password field of currentRecord = tmp(2)
         manager field of currentRecord = tmp(3)
         IF nullElementRemoved = False THEN
            arrToLoadInto(0) = currentRecord
            nullElementRemoved = True
         ELSE
            AddElementToArray(arrToLoadInto, currentRecord)
         END IF
      END IF
   END WHILE
   Close text file
   IF nullElementRemoved = True THEN
      RETURN arrToLoadInto
   ELSE
      Print "Tried to load from an empty file"
      RETURN Nothing
   END IF
END FUNCTION
```

```
PUBLIC SUB AddUserRecord(username As String, password As String, manager As Boolean)
   Declare recordToAdd As UserRecord
   username field of recordToAdd = username
   password field of recordToAdd = password
   manager field of recordToAdd = manager
   IF TextFileIsEmpty("users.txt") = True THEN
      Declare nextID As Integer = 0
      userID of recordToAdd = nextID
      Declare arrRecords( ) As UserRecord and set to recordToAdd
      SaveUserArrayToFile(arrRecords)
   ELSE
      Declare arrRecords( ) As UserRecord = LoadUsers( )
      Declare nextID As Integer = Last index of arrRecords + 1
      userID of recordToAdd = nextID
      AddElementToArray(arrRecords, recordToAdd)
      SaveUserArrayToFile(arrRecords)
   END IF
END SUB

PRIVATE SUB SaveUserArrayToFile(arrToSave( ) As UserRecord)
   Declare sep As Char = "|"
   Declare writeLine As String = ""
   FOR i = 0 TO Last index in arrToSave
      Add the current record to writeLine
   NEXT
   Override text file
END SUB

PUBLIC FUNCTION LoadMenuItems( )
   Declare arrToLoadInto(0) As MenuItemRecord
   Declare nullElementRemoved As Boolean = False
   Declare oRead As StreamReader
   Declare lineIn As String
   Declare tmp
   Set oRead to file, opened for reading
   WHILE there are still lines left to read
      Set lineIn to current line
      IF there are characters on the line THEN
         Split the line, separating the elements by the “|” character and storing each field in tmp
         Declare currentRecord As MenuItemRecord
         itemID field of currentRecord = tmp(0)
         itemName field of currentRecord = tmp(1)
         itemPrice field of currentRecord = tmp(2)
         isFood field of currentRecord = tmp(3)
         IF nullElementRemoved = False THEN
            arrToLoadInto(0) = currentRecord
            nullElementRemoved = True
         ELSE
            AddElementToArray(arrToLoadInto, currentRecord)
         END IF
      END IF
   END WHILE
   Close text file
   IF nullElementRemoved = True THEN
      RETURN arrToLoadInto
   ELSE
      Print “Tried to load from an empty file”
      RETURN Nothing
   END IF
END FUNCTION
```

```
PUBLIC SUB AddMenuItemRecord(menuItemName As String, menuItemPrice As Single, isFood As
Boolean)
    Declare recordToAdd As MenuItemRecord
    itemName field of recordToAdd = menuItemName
    itemPrice field of recordToAdd = menuItemPrice
    isFood field of recordToAdd = isFood
    IF TextFileIsEmpty("menu.txt") = True THEN
        Declare nextID As Integer = 0
        itemID field of recordToAdd = nextID
        Declare arrRecords( ) As MenuItemRecord and set to recordToAdd
        SaveMenuToFile(arrRecords)
    ELSE
        Declare arrRecords( ) As MenuItemRecord = LoadMenuItems ( )
        Declare nextID As Integer = Last index of arrRecords + 1
        itemID field of recordToAdd = nextID
        AddElementToArray(arrRecords, recordToAdd)
        SaveMenuToFile(arrRecords)
    END IF
END SUB

PRIVATE SUB SaveMenuToFile(arrToSave( ) As MenuItemRecord)
    Declare sep As Char = "|"
    Declare writeLine As String = ""
    FOR i = 0 TO Last index of arrToSave
        Add the current record to writeLine
    NEXT
    Override the text file
END SUB

PUBLIC SUB DeleteMenuItem(menuItemToDelete As MenuItemRecord)
    Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
    RemoveElementFromArray(menuItems, itemID of menuItemToDelete)
    FOR i = 0 To Last index in menuItems
        itemID of menuItems(i) = i
    NEXT
    SaveMenuToFile(menuItems)
END SUB

PUBLIC SUB EditMenuItem(recordToEdit As MenuItemRecord)
    Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
    menuItems(itemID of recordToEdit) = recordToEdit
    SaveMenuToFile(menuItems)
END SUB

PUBLIC SUB ClearMenuFile( )
    Delete all text from the text file.
END SUB

PUBLIC FUNCTION LoadOrders( )
    Declare arrToLoadInto(0) As OrderRecord
    Declare nullElementRemoved As Boolean = False
    Declare oRead As StreamReader
    Declare lineIn As String
    Declare tmp
    Set oRead to file, opened for reading
    WHILE there are still lines left to read
        Set lineIn to current line
        IF there are characters on the line THEN
            Split the line, separating the elements by the “|” character and storing each field in tmp
            Declare currentRecord As OrderRecord
            orderID field of currentRecord = tmp(0)
```

```
            orderDate field of currentRecord = tmp(1)
            tableNumber field of currentRecord = tmp(2)
            orderSubtotal field of currentRecord = tmp(3)
            orderPaid field of currentRecord = tmp(4)
            employee field of currentRecord = tmp(5)
            customerComments field of currentRecord = tmp(6)
            IF nullElementRemoved = False THEN
               arrToLoadInto(0) = currentRecord
               nullElementRemoved = True
            ELSE
               AddElementToArray(arrToLoadInto, currentRecord)
            END IF
         END IF
      END WHILE
      Close text file
      IF nullElementRemoved = True THEN
         RETURN arrToLoadInto
      ELSE
         Print “Tried to load from an empty file”
         RETURN Nothing
      END IF
   END FUNCTION

   PUBLIC SUB AddOrderRecord(orderDate As Date, tableNumber As Integer, orderSubtotal As Single,
orderPaid As Boolean, employee As String, customerComments As String)
      Declare recordToAdd As OrderRecord
      orderDate field of recordToAdd = orderDate
      tableNumber field of recordToAdd = tableNumber
      orderSubtotal field of recordToAdd = orderSubtotal
      orderPaid field of recordToAdd = orderPaid
      employee field of recordToAdd = employee
      customerComments field of recordToAdd = customerComments
      IF TextFileIsEmpty("sales.txt") = True THEN
         Declare nextID As Integer = 0
         orderID of recordToAdd = nextID
         Declare arrRecords( ) As OrderRecord and set to recordToAdd
         SaveOrderToFile(arrRecords)
      ELSE
         Declare arrRecords( ) As OrderRecord = LoadOrders( )
         Declare nextID As Integer = Last index of arrRecords + 1
         orderID of recordToAdd = nextID
         AddElementToArray(arrRecords, recordToAdd)
         SaveOrderToFile(arrRecords)
      END IF
   END SUB

   PRIVATE SUB SaveOrderToFile(arrToSave( ) As OrderRecord)
      Declare sep As Char = "|"
      Declare writeLine As String = ""
      FOR i = 0 TO Last index of arrToSave
         Add the current record to writeLine
      NEXT
      Override the text file with writeLine
   END SUB

   PUBLIC SUB EditOrderRecord(recordToEdit As OrderRecord)
      Declare allOrders( ) As OrderRecord = LoadOrders( )
      allOrders(recordToEdit.orderID) = recordToEdit
      SaveOrderToFile(allOrders)
   END SUB
```

```
PRIVATE FUNCTION TextFileIsEmpty(fileName As String)
   TRY
      IF File with filename is empty THEN
         Print fileName & “ is empty”
         RETURN True
      ELSE
         Print fileName & “ contains stuff”
         RETURN False
      END IF
   CATCH
      Print “Tried to read data from a file “ & fileName & “, which doesn't exist.”
      RETURN Nothing
   END TRY
END FUNCTION
```

SEARCH UTILITIES

```
PUBLIC FUNCTION ItemExistsInArray(arrToSearch( ) As Generic, ByVal elementToSearchFor As Generic)
    TRY
        FOR i = 0 TO Last index of arrToSearch
            IF arrToSearch(i) = (elementToSearchFor) THEN
                RETURN True
            END IF
        NEXT
    CATCH
        Print "Null array passed into function ItemExistsInArray( )"
    END TRY
    RETURN False
END FUNCTION

PUBLIC FUNCTION FindRecordsWithSearchCriteria(arrToSearch( ) As OrderRecord, searchCriteria As String)
    Declare arrToReturn( ) As OrderRecord = Nothing
    TRY
        FOR i = 0 TO Last index of arrToSearch
            IF orderSubtotal, employee or customerComments field of arrToSearch(i) contains search criteria THEN
                AddElementToArray(arrToReturn, arrToSearch(i))
            END IF
        NEXT
    CATCH
        Print "Null array passed into function FindRecordsWithSearchCriteria( )"
    END TRY
    RETURN arrToReturn
END FUNCTION
```

SORT UTILITIES

```
PUBLIC FUNCTION InsertionSortByOrderTotal(By Reference arrToSort( ) As OrderRecord)
   Declare firstIndex As Integer = 0
   Declare lastIndex As Integer = Last index of arrToSort
   Declare indexToSort As Integer = firstIndex + 1
   Declare currentIndexOfLoop As Integer = firstIndex
   WHILE indexToSort <= lastIndex
      WHILE currentIndexOfLoop < indexToSort
         IF orderSubtotal of arrToSort(indexToSort) >= orderSubtotal of arrToSort(currentIndexOfLoop)
THEN
            currentIndexOfLoop = currentIndexOfLoop + 1
         ELSE
            Declare temp As OrderRecord = arrToSort(indexToSort)
            FOR i = indexToSort TO currentIndexOfLoop + 1 STEP -1
               arrToSort(i) = arrToSort(i - 1)
            NEXT
            arrToSort(currentIndexOfLoop) = temp
         END IF
      END WHILE
      currentIndexOfLoop = firstIndex
      indexToSort = indexToSort + 1
   END WHILE
   RETURN arrToSort
END FUNCTION

PUBLIC FUNCTION InsertionSortByTableNumber(By Reference arrToSort( ) As OrderRecord)
   Declare firstIndex As Integer = 0
   Declare lastIndex As Integer = Last index of arrToSort
   Declare indexToSort As Integer = firstIndex + 1
   Declare currentIndexOfLoop As Integer = firstIndex
   WHILE indexToSort <= lastIndex
      WHILE currentIndexOfLoop < indexToSort
         IF tableNumber of arrToSort(indexToSort) >= tableNumber of arrToSort(currentIndexOfLoop)
THEN
            currentIndexOfLoop = currentIndexOfLoop + 1
         ELSE
            Declare temp As OrderRecord = arrToSort(indexToSort)
            FOR i = indexToSort TO currentIndexOfLoop + 1 STEP -1
               arrToSort(i) = arrToSort(i - 1)
            NEXT
            arrToSort(currentIndexOfLoop) = temp
         END IF
      END WHILE
      currentIndexOfLoop = firstIndex
      indexToSort = indexToSort + 1
   END WHILE
   RETURN arrToSort
END FUNCTION

PUBLIC FUNCTION InsertionSortByEmployee (By Reference arrToSort( ) As OrderRecord)
   Declare firstIndex As Integer = 0
   Declare lastIndex As Integer = Last index of arrToSort
   Declare indexToSort As Integer = firstIndex + 1
   Declare currentIndexOfLoop As Integer = firstIndex
   WHILE indexToSort <= lastIndex
      WHILE currentIndexOfLoop < indexToSort
         IF employee of arrToSort(indexToSort) comes after employee of arrToSort(currentIndexOfLoop) in
alphabet THEN
            currentIndexOfLoop = currentIndexOfLoop + 1
         ELSE
```

```
            Declare temp As OrderRecord = arrToSort(indexToSort)
            FOR i = indexToSort TO currentIndexOfLoop + 1 STEP -1
               arrToSort(i) = arrToSort(i - 1)
            NEXT
            arrToSort(currentIndexOfLoop) = temp
         END IF
      END WHILE
      currentIndexOfLoop = firstIndex
      indexToSort = indexToSort + 1
   END WHILE
   RETURN arrToSort
END FUNCTION

PUBLIC FUNCTION InsertionSortByDate(By Reference arrToSort( ) As OrderRecord)
   Declare firstIndex As Integer = 0
   Declare lastIndex As Integer = Last index of arrToSort
   Declare indexToSort As Integer = firstIndex + 1
   Declare currentIndexOfLoop As Integer = firstIndex
   WHILE indexToSort <= lastIndex
      WHILE currentIndexOfLoop < indexToSort
         IF orderDate of arrToSort(indexToSort) >= orderDate of arrToSort(currentIndexOfLoop) THEN
            currentIndexOfLoop = currentIndexOfLoop + 1
         ELSE
            Declare temp As OrderRecord = arrToSort(indexToSort)
            FOR i = indexToSort TO currentIndexOfLoop + 1 STEP -1
               arrToSort(i) = arrToSort(i - 1)
            NEXT
            arrToSort(currentIndexOfLoop) = temp
         END IF
      END WHILE
      currentIndexOfLoop = firstIndex
      indexToSort = indexToSort + 1
   END WHILE
   RETURN arrToSort
END FUNCTION
```

VALIDATION UTILITIES

```
PUBLIC FUNCTION StringIsConvertibleToInteger(str As String)
   TRY
      Convert str to integer
      RETURN True
   CATCH
      RETURN False
   END TRY
END FUNCTION

PUBLIC FUNCTION StringIsConvertibleToSingle(str As String)
   TRY
      Convert str to single
      RETURN True
   CATCH
      RETURN False
   END TRY
END FUNCTION
```

USER UTILITIES

```
PUBLIC FUNCTION IsManager(userToCheck As String)
   Declare users( ) As UserRecord = LoadUsers( )
   TRY
      FOR i = 0 TO Last index of users
         IF username of users(i) = userToCheck THEN
            RETURN manager field of users(i)
         END IF
      NEXT
   CATCH
   END TRY
   RETURN FALSE
END FUNCTION
```

SOURCE CODE

LOGIN

```
  Public Class formLogin

    Public currentUser As String 'Public variable for keeping track of the user that is currently
logged in.

    Private Sub ClickedButtonSubmit() Handles btnSubmit.Click
        'Check if the username and password fields satisfy all criteria then
        If CheckLogin(iptUsername.Text, iptPassword.Text) = True Then
            currentUser = iptUsername.Text 'Update value of current user
            ClearInputs() 'Clear all inputs
            Me.Hide() 'Hide current form
            formMain.Show() 'Show main form
        Else
            MsgBox("Username or Password incorrect.") 'Display error message
            ClearInputs() 'Clear all inputs
        End If
    End Sub

    Private Function CheckLogin(ByVal inputUsername As String, ByVal inputPassword As String)
        Dim users() As UserRecord = LoadUsers() 'Array of all users
        Try
            'Loop through users and see if username and password match
            For i = 0 To UBound(users)
                If users(i).username = inputUsername And users(i).password = inputPassword Then
                    'Login successful
                    Return True
                End If
            Next
        Catch ex As Exception
            'i.e. There is nothing in the users array
        End Try
        Return False
    End Function

    Private Sub ClearInputs()
        'Reset text boxes
        iptPassword.Text = ""
        iptUsername.Text = ""
    End Sub

    Private Sub ClickedButtonCreateNewLogin() Handles btnCreateLogin.Click
        formNewLogin.Show() 'Show main form
        Me.Hide()   'Hide current form
    End Sub

    Private Sub ClickedButtonQuit() Handles btnQuit.Click
        End 'End program
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        MsgBox("Login as an existing user or create a new user.") 'Display help message
    End Sub
End Class
```

NEW LOGIN

```
Public Class formNewLogin
    Const maxUserLength As Integer = 8

    Private Sub ClickedButtonCreateUser() Handles btnCreateUser.Click
        'Check if username and password have been given yet
        If Not iptUsername.Text = "" And Not iptPassword.Text = "" Then
            'Make sure username length is less than maxUserLength
            If iptUsername.Text.Length < maxUserLength Then
                'Check the given passwords match
                If iptPassword.Text = iptConfirmPassword.Text Then
                    If Not UserAlreadyExists(iptUsername.Text) Then
                        'New user creation successful
                        AddUserRecord(iptUsername.Text, iptPassword.Text, togManager.Checked)
'Add a new user
                        MsgBox("New user created successfully")
                        formLogin.currentUser = iptUsername.Text 'Update value of currentUser
                        formMain.Show() 'Show main form
                        Me.Close() 'Close current form
                    Else
                        MsgBox("User already exists in database. Please choose a new username.")
                    End If
                Else
                    MsgBox("Passwords don't match.")
                End If
            Else
                MsgBox("Please choose a username that is less than " & maxUserLength & "
characters long")
            End If
        Else
            MsgBox("Please insert a username and password")
        End If

    End Sub

    Private Function UserAlreadyExists(ByVal userToSearch As String)
        Dim users() As UserRecord = LoadUsers() 'Array of all user records
        Try
            'Extract usernames from the userRecord array
            Dim usernames(UBound(users)) As String
            For i = 0 To UBound(users)
                usernames(i) = users(i).username
            Next

            'Check if the username already exists
            If ItemExistsInArray(usernames, userToSearch) Then
                Return True
            Else
                Return False
            End If

        Catch ex As Exception
            'i.e. There is nothing in the users array
            Return False
        End Try
    End Function

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formLogin.Show() 'Show login form
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
```

```
        MsgBox("Select a username and a password for your account. Repeat your password to make
sure it is correct, and then select either an employee or a manager account. You will be taken to
the main screen after your account is successfully created.")
    End Sub
End Class
```

MAIN FORM

```
Public Class formMain

    Private Sub LoadedMainForm() Handles MyBase.Load
        lblWelcome.Text = "Welcome, " & formLogin.currentUser 'Update welcome text
        'Check if the user is a manager or not
        If Not IsManager(formLogin.currentUser) Then
            'Move buttons around and make some buttons invisible to remove functionality if the
user isn't a manager
            btnSwitchUser.Location = New Point(btnChangeMenu.Location.X,
btnChangeMenu.Location.Y)
            btnQuit.Location = New Point(btnViewSales.Location.X, btnViewSales.Location.Y)
            btnViewSales.Visible = False
            btnChangeMenu.Visible = False
        End If
    End Sub

#Region "MainMenuButtons"

    Private Sub ClickedButtonTakeOrders() Handles btnTakeOrders.Click
        formTakeOrders.Show() 'Show formTakeOrders
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonPayForOrders() Handles btnPayForOrders.Click
        formPayOrders.Show() 'Show formPayOrders
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonChangeMenu() Handles btnChangeMenu.Click
        formEditMenu.Show() 'Show formEditMenu
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickeButtonViewSales() Handles btnViewSales.Click
        formViewSales.Show() 'Show formViewSales
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonSwitchUser() Handles btnSwitchUser.Click
        formLogin.Show() 'Show login form
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonQuit() Handles btnQuit.Click
        End 'End Program
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
        MsgBox("Click:" & vbCrLf & "'Take Orders' to wait on customers." & vbCrLf & "'Pay for
Orders' to confirm payment as the customer leaves" & vbCrLf & "'Switch User' to log in as someone
else" & vbCrLf & "'Quit' to stop using the application" & vbCrLf & vbCrLf & "If you are a
manager, you are also able to click:" & vbCrLf & "'Change Menu' to add, edit and delete menu
items" & vbCrLf & "'View Sales' to view and search all previous sales")
    End Sub

#End Region

End Class
```

TAKE ORDERS

```
Public Class formTakeOrders

    Private selectedTableNumber As Integer = 1 'Variable for keeping track of the currently
selected table number
    Private orderItemIndexes() As Integer 'Array for keeping track of the MenuItemIDs of all
items that have been ordered.

    Private Sub LoadedFormTakeOrders() Handles MyBase.Load
        LoadMenu() 'Load the menu items from the database
    End Sub

    Private Sub ChangedTableNumber(sender As System.Object, e As System.EventArgs) Handles
togTable1.CheckedChanged, togTable2.CheckedChanged, togTable3.CheckedChanged,
togTable4.CheckedChanged, togTable5.CheckedChanged, togTable6.CheckedChanged,
togTable7.CheckedChanged, togTable8.CheckedChanged, togTable9.CheckedChanged,
togTable10.CheckedChanged, togTable11.CheckedChanged, togTable12.CheckedChanged,
togTable13.CheckedChanged, togTable14.CheckedChanged, togTable15.CheckedChanged,
togTable16.CheckedChanged
        'Check if the toggle that was updated is now checked
        If sender.checked = True Then
            selectedTableNumber = sender.tag + 1 'Set selectedTableNumber to the tag of the radio
button + 1 (Tags start from 0)
        End If
    End Sub

    Private Sub LoadMenu()
        'Load data from the menu database into the combo boxes.
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all the menu items in the
database
        For Each item As MenuItemRecord In menuItems
            If item.isFood Then
                comboFood.Items.Add(item.itemName) 'Add the item to the food combo box.
            Else
                comboBeverages.Items.Add(item.itemName) 'Add the item to the beverages combo box.
            End If
        Next
    End Sub

    Private Sub ClickedButtonAddFood() Handles btnAddFood.Click
        If Not comboFood.SelectedIndex = -1 Then 'Check that an item has been selected.
            If StringIsConvertibleToInteger(txtFoodAmount.Text) Then 'Check if the amount given
in the text box is an integer. (Avoids errors trying to parse a string that isn't convertible to
an integer)
                For i = 0 To Integer.Parse(txtFoodAmount.Text) - 1
                    AddElementToArray(orderItemIndexes, GetMenuItemIndex(comboFood.SelectedItem))
'Add the selected item to the orderItemIndexes array.
                Next
                UpdateOrderList(orderItemIndexes) 'Update the order list box.
                lblOrderTotalAmount.Text = Format(CalculateOrderSubtotal(orderItemIndexes),
"Currency") 'Calculate new order total and update orderTotal label.
            Else
                MsgBox("Please select an amount of the food you want to add.")
            End If
        Else
            MsgBox("Please select the food item that you want to add.")
        End If
    End Sub

    Private Sub ClickedButtonAddBeverage() Handles btnAddBeverage.Click
        If Not comboBeverages.SelectedIndex = -1 Then 'Check that an item has been selected.
            If StringIsConvertibleToInteger(txtBeverageAmount.Text) Then 'Check if the amount
given in the text box is an integer. (Avoids errors trying to parse a string that isn't
convertible to an integer)
                For i = 0 To Integer.Parse(txtBeverageAmount.Text) - 1
```

```
                    AddElementToArray(orderItemIndexes,
GetMenuItemIndex(comboBeverages.SelectedItem)) 'Add the selected item to the orderItemIndexes
array.
                Next
                UpdateOrderList(orderItemIndexes) 'Update the order list box.
                lblOrderTotalAmount.Text = Format(CalculateOrderSubtotal(orderItemIndexes),
"Currency") 'Calculate new order total and update orderTotal label.
            Else
                MsgBox("Please select an amount of the beverage you want to add.")
            End If
        Else
            MsgBox("Please select the beverage that you want to add.")
        End If
    End Sub

    Private Function GetMenuItemIndex(ByVal menuItemName As String) 'Function for traversing all
menu items. Takes in the menu item name and returns its unique itemID for reference.
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items, loaded from
the menu database.
        For Each item As MenuItemRecord In menuItems
            If item.itemName = menuItemName Then 'If input itemName matches the itemName of this
specific data item
                Return item.itemID 'Return the itemID
            End If
        Next
        Debug.Print("Couldn't find menu item " & menuItemName & " in database.") 'Menu item
doesn't exist.
        Return Nothing
    End Function

    Private Sub UpdateOrderList(ByVal orderItemIndexes() As Integer)
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items, loaded from
the menu item database
        'Clear all items in both list boxes.
        lstItems.Items.Clear()
        lstItemsConfirm.Items.Clear()
        Try
            For i = 0 To UBound(orderItemIndexes)
                'Add current item to each list box
                lstItems.Items.Add(menuItems(orderItemIndexes(i)).itemName)
                lstItemsConfirm.Items.Add(menuItems(orderItemIndexes(i)).itemName)
            Next
        Catch ex As Exception
            'No indexes in orderItemIndexes
        End Try
    End Sub

    Private Function CalculateOrderSubtotal(ByVal orderItemIndexes() As Integer)
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items, loaded from
the menu database.
        Dim subtotal As Single = 0 'Running total of the order.
        Try
            For i = 0 To UBound(orderItemIndexes)
                subtotal = subtotal + menuItems(orderItemIndexes(i)).itemPrice 'Add the current
item's price to the subtotal.
            Next
        Catch ex As Exception
            'nothing in the array
        End Try
        Return subtotal 'Return the subtotal
    End Function

    Private Sub ClickedButtonResetList() Handles btnResetList.Click
        lstItems.Items.Clear() 'Clear all items from the list box.
        PurgeArray(orderItemIndexes) 'Remove all elements from the array orderItemIndexes.
        UpdateOrderList(orderItemIndexes) 'Update list boxes.
```

```
        lblOrderTotalAmount.Text = Format(CalculateOrderSubtotal(orderItemIndexes), "Currency")
'Update order subtotal
    End Sub

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formMain.Show() 'Show main form
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonConfirmOrder() Handles btnConfirmOrder.Click
        'If the order subtotal is not $0.00
        If Not CalculateOrderSubtotal(orderItemIndexes) = 0 Then
            AddOrderRecord(Today().Date, selectedTableNumber,
CalculateOrderSubtotal(orderItemIndexes), False, formLogin.currentUser, "None") 'Add the order
record to the database.
            PrintReceipt() 'Print the receipt with all order details.
            formMain.Show() 'Show main form
            Me.Close() 'Close current form
        Else
            MsgBox("Can't confirm an order that doesn't contain any items")
        End If

    End Sub

    Private Sub PrintReceipt()
        Dim printString As String 'Variable that keeps track of the string to print.
        'Add order data to the printString
        printString = "Order Confirmed." & vbCrLf & vbCrLf & "Table Number: " &
selectedTableNumber & vbCrLf & vbCrLf & "Order details:" & vbCrLf
        For Each item In lstItems.Items
            printString += item & vbCrLf
        Next
        printString += vbCrLf & "Order Subtotal:" & vbCrLf &
Format(CalculateOrderSubtotal(orderItemIndexes), "Currency")
        MsgBox(printString) 'Display the printString in a messagebox to simulate a receipt.
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
        MsgBox("Select a table number in the 'Select Table' tab." & vbCrLf & "Add food and
beverages to the order in the 'Food/Beverages' tab." & vbCrLf & "Review and confirm the order in
the 'Confirm Order' tab.")
    End Sub
End Class
```

PAY FOR ORDERS

```
Public Class formPayOrders

    Private currentUnpaidOrders() As OrderRecord 'Array of all items that are currently being
displayed in the list box. Note that it mirrors the exact items being displayed.

    Private Sub CLickedButtonBack() Handles btnBack.Click
        formMain.Show() 'Show main form
        Me.Close() 'Close current form
    End Sub

    Private Sub LoadedFormPayForOrders() Handles MyBase.Load
        currentUnpaidOrders = GetCurrentUnpaidOrders() 'Load currentUnpaid orders into the
control array
        DisplayArrayInListBox(currentUnpaidOrders) 'Display current unpaid orders in the list box
    End Sub

    Private Function GetCurrentUnpaidOrders()
        Dim allOrders() As OrderRecord = LoadOrders() 'Array of all orders ever made.
        Dim ordersToReturn() As OrderRecord = Nothing 'Array of orders to return
        'Find orders that have been made today and have not been paid for yet
        For i = 0 To UBound(allOrders)
            If allOrders(i).orderDate = Today.Date And allOrders(i).orderPaid = False Then
                AddElementToArray(ordersToReturn, allOrders(i)) 'Add this order to ordersToReturn
            End If
        Next
        Return ordersToReturn 'Return ordersToReturn
    End Function

    Private Sub DisplayArrayInListBox(ByVal arrToDisplay() As OrderRecord)
        lstOrders.Items.Clear() 'Clear all items in list box.
        Try
            For Each item In arrToDisplay
                'Add each order to the list box. Note the order id is added on the end so we know
which record to edit later.
                lstOrders.Items.Add(item.tableNumber & vbTab & item.employee & vbTab &
Format(item.orderSubtotal, "Currency"))
            Next
        Catch ex As Exception
            'Nothing to display
            lstOrders.Items.Add("")
            lstOrders.Items.Add("No orders from today")
            lstOrders.Items.Add("are still unpaid")
        End Try
    End Sub

    Private Sub ClickedButtonSortByTable() Handles btnSortByTable.Click
        currentUnpaidOrders = InsertionSortByTableNumber(currentUnpaidOrders) 'Sorted array of
all  unpaid current orders.
        DisplayArrayInListBox(currentUnpaidOrders) 'Display this array in list box.
    End Sub

    Private Sub ClickedButtonSortByOrderTotal() Handles btnSortByTotal.Click
        currentUnpaidOrders = InsertionSortByOrderTotal(currentUnpaidOrders) 'Sorted array of all
unpaid current orders.
        Array.Reverse(currentUnpaidOrders)
        DisplayArrayInListBox(currentUnpaidOrders) 'Display this array in list box.
    End Sub


    Private Sub ClickedButtonConfirmPayment() Handles btnConfirmPayment.Click
        If ItemIsSelected() Then
            'Add any customer comments to the order and mark it as paid.
            Dim orderIDToEdit As Integer = currentUnpaidOrders(lstOrders.SelectedIndex).orderID
'Declare orderIDToEdit and set to the orderID of the currently selected record.
```

```
            Dim allOrders() As OrderRecord = LoadOrders() 'Array of all orders ever made.
            allOrders(orderIDToEdit).orderPaid = True 'Change orderPaid flag of the appropriate
record to true.

            If Not iptCustomerComments.Text = "" Then
                allOrders(orderIDToEdit).customerComments = iptCustomerComments.Text 'Add any
customer comments to the appropriate record.
            End If

            EditOrderRecord(allOrders(orderIDToEdit)) 'Make edits to the order at orderIDToEdit
            currentUnpaidOrders = GetCurrentUnpaidOrders()
            DisplayArrayInListBox(currentUnpaidOrders) 'Display all current unpaid orders in the
list box. (Note this removes the order we just edited, as it is now paid for)
            iptCustomerComments.Text = "" 'Reset customer comments
            MsgBox("Order Marked as Paid.")
        Else
            MsgBox("Please select an order to confirm.")
        End If
    End Sub

    Private Function ItemIsSelected() 'Check if an item is selected in the list box.
        If lstOrders.SelectedIndex = -1 Then
            Return False
        Else
            Return True
        End If
    End Function

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message.
        MsgBox("All of the orders displayed are unpaid orders from today. Sort by table number or
by order total to find the correct order, add any customer comments and mark the order as paid
once payment is confirmed.")
    End Sub
End Class
```

EDIT MENU

```
Public Class formEditMenu

    Private Sub LoadedFormEditMenu() Handles MyBase.Load
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items
        'Add all menu items to the list box.
        For Each item As MenuItemRecord In menuItems
            lstMenuItems.Items.Add(item.itemName & vbTab & Format(item.itemPrice, "Currency"))
        Next
    End Sub

    Private Sub ClickedButtonAddMenuItem() Handles btnAddMenuItem.Click
        formAddMenuItem.Show() 'Show formAddMenuItem
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonEditMenuItem() Handles btnEditMenuItem.Click
        'Check if an item is currently selected
        If ItemIsSelected() Then
            formEditMenuItem.Show() 'Show formEditMenuItem
            formEditMenuItem.GetItemToEditIndex(lstMenuItems.SelectedIndex) 'Pass in the selected
index to formEditMenuItem
            Me.Close() 'Close current form
        Else
            MsgBox("Please select an item from the list to edit")
        End If
    End Sub

    Private Sub ClickedButtonDeleteMenuItem() Handles btnDeleteMenuItem.Click
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items
        'Check if there are items in the list
        If Not lstMenuItems.Items.Count = 0 Then
            'Check if an item is currently selected
            If ItemIsSelected() Then
                DeleteMenuItem(menuItems(lstMenuItems.SelectedIndex)) 'Delete selected menu item
                lstMenuItems.Items.RemoveAt(lstMenuItems.SelectedIndex) 'Remove item from list
box.
            Else
                MsgBox("Please select an item from the list to delete")
            End If
        Else
            MsgBox("Please create a menu item before trying to delete one.")
        End If
    End Sub

    Private Sub ClickedButtonResetMenu() Handles btnResetMenu.Click
        ClearMenuFile() 'Clear menu file
        lstMenuItems.Items.Clear() 'Clear menu items list box.
    End Sub

    Private Function ItemIsSelected() 'Checks if an item is selected in a list box or not.
        If lstMenuItems.SelectedIndex = -1 Then
            Return False
        Else
            Return True
        End If
    End Function

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formMain.Show() 'Show main form
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
```

```
        MsgBox("Click:" & vbCrLf & "'Add Menu Item' to add an item to the menu." & vbCrLf &
"'Edit Menu Item' to edit the price of an existing item" & vbCrLf & "'Delete Menu Item' after
selecting an item from the list to remove it from the menu" & vbCrLf & "'Reset Menu' to reset the
entire menu")
    End Sub

End Class
```

ADD MENU ITEM

```
Public Class formAddMenuItem

    Private Sub ClickedButtonConfirmAdd() Handles btnAddItem.Click
        'Check if there is an item name in the input box.
        If Not iptItemName.Text = "" Then
            'Check if the price value is convertible to a single data type.
            If StringIsConvertibleToSingle(iptItemPrice.Text) Then
                Dim itemPrice As Single = FormatNumber(iptItemPrice.Text, 2) 'Price of the item,
to 2 d.p.
                AddMenuItemRecord(iptItemName.Text, itemPrice, togFood.Checked) 'Add menu item to
database.
                MsgBox("New menu item added.")
                'Reset input boxes
                iptItemName.Text = ""
                iptItemPrice.Text = ""
            Else
                MsgBox("Price field is invalid. Make sure that it only contains numbers.")
            End If
        Else
            MsgBox("Please insert an item name.")
        End If
    End Sub

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formEditMenu.Show() 'Show formEditMenu
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
        MsgBox("Choose an item name (A short but recognisable name is best) and a price, and
select whether it is a food or a beverage. Press 'Add Item' when finished.")
    End Sub
End Class
```

EDIT MENU ITEM

```
Public Class formEditMenuItem

    Private itemToEditIndex As Integer = -1 'The index of the item that we want to edit. Set to -
1 initially, as no item is selected.

    Public Sub GetItemToEditIndex(ByVal index As Integer)
        itemToEditIndex = index 'Set itemToEdit index to the item index passed in from
formEditMenu
        DisplayDataToEdit() 'Display the data that we want to edit.
    End Sub

    Private Sub ClickedButtonConfirmEdit() Handles btnEditItem.Click
        Dim newRecord As MenuItemRecord 'The new record that we will replace the existing one
with.
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items
        'Check if the item price field is convertible to Single data type.
        If StringIsConvertibleToSingle(iptItemPrice.Text) Then
            'Populate fields of newRecord
            newRecord.itemID = itemToEditIndex
            newRecord.itemName = lblItem.Text
            newRecord.itemPrice = iptItemPrice.Text
            newRecord.isFood = menuItems(itemToEditIndex).isFood
            EditMenuItem(newRecord) 'Override the existing menu item
            formEditMenu.Show() 'Show formEditMenu
            Me.Close() 'Close current form
        Else
            MsgBox("Price field is invalid. Make sure that it only contains numbers.")
        End If
    End Sub


    Private Sub DisplayDataToEdit()
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items
        'Display the data that we want to edit
        lblItem.Text = menuItems(itemToEditIndex).itemName
        iptItemPrice.Text = FormatNumber(menuItems(itemToEditIndex).itemPrice, 2)
    End Sub

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formEditMenu.Show() 'Show formEditMenu
        Me.Close() 'Close current form
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
        MsgBox("Change the price of an existing item. Press 'Confirm Edit Item' when you have
finished editing the menu item.")
    End Sub

End Class
```

VIEW SALES

```
Public Class formViewSales

    Dim currentlyDisplayedItems() As OrderRecord = LoadOrders() 'Array of all items that are
currently being displayed. Note that it mirrors the exact items being displayed.

    Private Sub LoadedFormViewSales() Handles MyBase.Load
        DisplayArrayInSalesListBox(LoadOrders()) 'Display ALL sales in list box.
    End Sub

    Private Sub ChangedFilterCriteria() Handles iptSearch.TextChanged, dateFrom.ValueChanged,
dateTo.ValueChanged
        'Check if the from-date is earlier than the to-date
        If dateFrom.Value.Date <= dateTo.Value.Date Then
            currentlyDisplayedItems = FindSalesWithinDateRange(dateFrom.Value.Date,
dateTo.Value.Date) 'Filter currentlyDisplayedItems, returning only the sales that are within the
specified date range.
            currentlyDisplayedItems = FindRecordsWithSearchCriteria(currentlyDisplayedItems,
iptSearch.Text) 'Filter currentlyDisplayedItems, returning the sales that match the previous
filter AND contain the search criteria.
            DisplayArrayInSalesListBox(currentlyDisplayedItems) 'Display all filtered sales in a
list box.
        Else
            MsgBox("Please ensure that the start date is before the end date.")
            dateFrom.Value = dateTo.Value.Date 'Set from-date to the to-date.
        End If
    End Sub

    Private Sub DisplayArrayInSalesListBox(ByVal arrToDisplay() As OrderRecord)
        lstSales.Items.Clear() 'Clear all items from list box.
        Try
            'Loop through all the elements of the array we want to display.
            For i = 0 To UBound(arrToDisplay)
                'Add item to list box, starting from first item
                lstSales.Items.Add(arrToDisplay(i).orderDate & vbTab &
Format(arrToDisplay(i).orderSubtotal, "Currency") & vbTab & arrToDisplay(i).employee)
            Next
        Catch ex As Exception
            'Nothing to display
            lstSales.Items.Add("")
            lstSales.Items.Add("No sales found that match the given criteria") 'Display message
in list box for feedback.
        End Try

    End Sub

    Private Sub ClickedButtonBack() Handles btnBack.Click
        formMain.Show() 'Show main form
        Me.Close() 'Close current form
    End Sub

    Private Function FindSalesWithinDateRange(ByVal startDate As Date, ByVal endDate As Date)
        Dim allSales() As OrderRecord = LoadOrders() 'Array of all sales that have ever been
made.
        Dim salesToReturn() As OrderRecord = Nothing 'Array of all sales we want to return
        Try
            'Loop through array of all sales.
            For Each sale In allSales
                'Check if the sale date is within the specified date range.
                If sale.orderDate >= startDate And sale.orderDate <= endDate Then
                    AddElementToArray(salesToReturn, sale) 'Add the sale to the salesToReturn
array
                End If
            Next
        Catch ex As Exception
```

```
            'No sales to loop through
        End Try
        Return salesToReturn 'Return all sales within date range.
    End Function

    Private Sub ClickedButtonInspectRecord() Handles btnInspectRecord.Click
        'Check if an item is selected in the list box.
        If ItemIsSelected() Then
            Dim allOrders() As OrderRecord = LoadOrders() 'Array of all orders
            Dim orderIDToDisplay As Integer =
currentlyDisplayedItems(lstSales.SelectedIndex).orderID 'Declare orderIDToDisplay and set to the
orderID of the item at the current index of the list box.
            DisplayItemInfo(allOrders(orderIDToDisplay)) 'Display item info in a popup window.
        Else
            MsgBox("Please select an item to inspect")
        End If
    End Sub

    Private Sub DisplayItemInfo(ByVal itemToDisplay As OrderRecord)
        'Print sale data in a popup window.
        MsgBox("Sale info:" & vbCrLf & vbCrLf & "Order ID: " & vbTab & itemToDisplay.orderID &
vbCrLf & "Date: " & vbTab & itemToDisplay.orderDate & vbCrLf & "Waiter: " & vbTab &
itemToDisplay.employee & vbCrLf & "Subtotal: " & vbTab & Format(itemToDisplay.orderSubtotal,
"Currency") & vbCrLf & "Tips:" & vbTab & itemToDisplay.customerComments)
    End Sub

    Private Function ItemIsSelected() 'Check if an item is selected in a listbox.
        If lstSales.SelectedIndex = -1 Then
            Return False
        Else
            Return True
        End If
    End Function

    Private Sub ClickedButtonSortByOrderTotal() Handles btnSortTotal.Click
        currentlyDisplayedItems = InsertionSortByOrderTotal(currentlyDisplayedItems) 'Sort
current items by order total
        Array.Reverse(currentlyDisplayedItems)
        DisplayArrayInSalesListBox(currentlyDisplayedItems) 'Update list box display
    End Sub

    Private Sub ClickedButtonSortByEmployee() Handles btnSortEmployee.Click
        currentlyDisplayedItems = InsertionSortByEmployee(currentlyDisplayedItems) 'Sort current
items by employee
        DisplayArrayInSalesListBox(currentlyDisplayedItems) 'Update list box display
    End Sub

    Private Sub ClickedButtonSortByOrderDate() Handles btnSortDate.Click
        currentlyDisplayedItems = InsertionSortByDate(currentlyDisplayedItems) 'Sort current
items by date.
        DisplayArrayInSalesListBox(currentlyDisplayedItems) 'Update list box display
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        'Display help message
        MsgBox("Enter search criteria in the input box at the top of the screen." & vbCrLf &
"Choose a range of dates that you want to inspect, followed by any sorting you want to apply to
the data." & vbCrLf & "Click 'Inspect Record' after selecting an item from the list to view more
information about it.")
    End Sub
End Class
```

ARRAY UTILITIES

```
Module ArrayUtilities
    Public Sub AddElementToArray(Of T)(ByRef arrToAddTo() As T, ByVal elementToAdd As T)
        Try
            Dim nextID As Integer = UBound(arrToAddTo) + 1 'The next ID of the array that we want
to create.
            ReDim Preserve arrToAddTo(nextID) 'Open up new spot in array
            arrToAddTo(nextID) = elementToAdd 'Set value of new spot to the element we want to
add
        Catch ex As Exception 'Empty array parameter.
            Dim arrToReturn(0) As T 'Create new array with 1 element
            arrToReturn(0) = elementToAdd 'Set this element to the element we want to add.
            arrToAddTo = arrToReturn 'Set arrToReturn to this new array.
        End Try
    End Sub

    Public Sub RemoveElementFromArray(Of T)(ByRef arrToRemoveFrom() As T, ByVal
indexOfElementToRemove As Integer)
        Try
            Dim arrToReturn(UBound(arrToRemoveFrom) - 1) As T 'Create new array with 1 less
element than the parameter array.
            'Loop through the array that we want to remove from
            For i = 0 To UBound(arrToRemoveFrom)
                If i < indexOfElementToRemove Then 'If the current index is less than the index
of the element we want to remove then
                    arrToReturn(i) = arrToRemoveFrom(i) 'Copy the data from the parameter array
to the array that we want to return, keeping the same indices.
                ElseIf i > indexOfElementToRemove Then 'If the current index is greater than the
index of the element we want to remove then
                    arrToReturn(i - 1) = arrToRemoveFrom(i) 'Copy the data from the parameter
array to the array that we want to return, decreasing the index by 1
                End If
            Next
            arrToRemoveFrom = arrToReturn 'Return the final array
        Catch ex As Exception
            Debug.Print("Index doesn't exist in arrToRemoveFrom")
        End Try
    End Sub

    Public Sub PurgeArray(Of T)(ByRef arrToPurge() As T)
        arrToPurge = Nothing 'Set array to nothing
    End Sub

End Module
```

DATA UTILITIES

```
Imports System.IO

Module DataUtilities

    Public Structure UserRecord
        Dim userID As Integer 'ID for reference
        Dim username As String 'Username
        Dim password As String 'Password
        Dim manager As Boolean 'Is the user a manager?
    End Structure

    Public Structure MenuItemRecord
        Dim itemID As Integer 'ID for reference
        Dim itemName As String 'Name of the item
        Dim itemPrice As Single 'Price of the item
        Dim isFood As Boolean 'Is the item a food item?
    End Structure

    Public Structure OrderRecord
        Dim orderID As Integer 'ID for reference
        Dim orderDate As Date 'Date that the order was made
        Dim tableNumber As Integer 'Table number for a specific order
        Dim orderSubtotal As Single 'Order subtotal
        Dim orderPaid As Boolean 'Has the order been paid for?
        Dim employee As String 'The employee that entered the order
        Dim customerComments As String 'Additional customer comments.
    End Structure

    Public filePath As String = Application.StartupPath 'The file path of the project

#Region "Users"

    Public Function LoadUsers()
        Dim arrToLoadInto(0) As UserRecord 'The array that we want to load the data into
        Dim nullElementRemoved As Boolean = False 'Has the intial null element been removed yet?
        Dim oRead As System.IO.StreamReader 'For sequentially accessing the contents of text
files.
        Dim lineIn As String 'The line read by oRead that is then processed.
        Dim tmp 'Temporary variable for storing data while it is processed and placed into
arrToLoadInto.

        oRead = File.OpenText(filePath & "\users.txt") 'Locate text file
        Debug.Print(filePath)

        While oRead.Peek <> -1 'while there are still lines left to read
            lineIn = oRead.ReadLine() 'Set lineIn to current line
            If Mid(lineIn, 1, 1) <> "" Then 'if there are characters on the line
                tmp = Split(lineIn, "|") 'Split the line, separating the elements by the “|”
character
                Dim currentRecord As UserRecord 'The current userRecord that we are analysing
                'Populate fields in currentUser variable
                currentRecord.userID = tmp(0)
                currentRecord.username = tmp(1)
                currentRecord.password = tmp(2)
                currentRecord.manager = tmp(3)
                'Check if the null element has been removed.
                If nullElementRemoved = False Then
                    'i.e. The array still contains its initial null element and this element
needs to be overrided
                    arrToLoadInto(0) = currentRecord 'override
                    nullElementRemoved = True
                Else
                    'i.e. The array contains only useful data, add the next useful record on the
end
```

```
                    AddElementToArray(arrToLoadInto, currentRecord)
                End If
            End If
        End While
        oRead.Close() 'Close text file
        'Check if the null element has been removed
        If nullElementRemoved = True Then
            Return arrToLoadInto
        Else
            'We dont want to return anything
            Debug.Print("Tried to load users from an empty file")
            Return Nothing
        End If
    End Function

    Public Sub AddUserRecord(ByVal username As String, password As String, ByVal manager As
Boolean)
        Dim recordToAdd As UserRecord 'The record to add to arrRecords

        'Load data into recordToAdd from parameters. Note that the ID isn't decided on yet, we
determine that in the next IF statements.
        recordToAdd.username = username
        recordToAdd.password = password
        recordToAdd.manager = manager

        'Checkif the text file is empty
        If TextFileIsEmpty("users.txt") = True Then
            Dim nextID As Integer = 0 'The next ID of the record we want to add
            recordToAdd.userID = nextID 'Set the ID of the record we want to add to nextID
            Dim arrRecords() As UserRecord = {recordToAdd} 'Array containing the record we want
to add
            SaveUserArrayToFile(arrRecords) 'Save new array to file
        Else
            Dim arrRecords() As UserRecord = LoadUsers() 'Create an array of all users, and load
data from file into it
            Dim nextID As Integer = UBound(arrRecords) + 1 'The next ID of the record we want to
add
            recordToAdd.userID = nextID 'Set the ID of the record we want to add to nextID
            AddElementToArray(arrRecords, recordToAdd) 'Add recordToAdd to the end of array
arrRecords
            SaveUserArrayToFile(arrRecords) 'Save new array to file
        End If
    End Sub

    Private Sub SaveUserArrayToFile(ByVal arrToSave() As UserRecord)
        Dim sep As Char = "|" 'The separator character between elements in the text file.
        Dim writeLine As String = "" 'The string to be written to file
        'Loop through the array that we want to save
        For i = 0 To UBound(arrToSave)
            writeLine &= arrToSave(i).userID & sep & arrToSave(i).username & sep &
arrToSave(i).password & sep & arrToSave(i).manager & vbCrLf 'Add the current record to writeLine
        Next
        File.WriteAllText(filePath & "\users.txt", writeLine) 'Override text file
    End Sub
#End Region

#Region "Menu"

    Public Function LoadMenuItems()
        Dim arrToLoadInto(0) As MenuItemRecord 'The array we want to load the data into
        Dim nullElementRemoved As Boolean = False 'Has the intial null element been removed yet?
        Dim oRead As System.IO.StreamReader 'For sequentially accessing the contents of text
files.
        Dim lineIn As String 'The line read by oRead that is then processed.
        Dim tmp 'Temporary variable for storing data while it is processed and placed into
arrToLoadInto.
```

```
        oRead = File.OpenText(filePath & "\menu.txt") 'Locate text file

        While oRead.Peek <> -1 'While there are still lines left to read
            lineIn = oRead.ReadLine() 'Set lineIn to current line
            If Mid(lineIn, 1, 1) <> "" Then 'if there are characters on the line
                tmp = Split(lineIn, "|") 'Split the line, separating the elements by the “|”
character
                Dim currentRecord As MenuItemRecord 'The current menu item we are examining
                'Populate fields in currentMenuItem
                currentRecord.itemID = tmp(0)
                currentRecord.itemName = tmp(1)
                currentRecord.itemPrice = tmp(2)
                currentRecord.isFood = tmp(3)
                'Check if the null element has been removed yet
                If nullElementRemoved = False Then
                    'i.e. The array still contains its initial null element and this element
needs to be overrided
                    arrToLoadInto(0) = currentRecord 'Override the first item in the array
                    nullElementRemoved = True 'Set nullElementRemoved to True
                Else
                    'i.e. The array contains only useful data, add the next useful record on the
end
                    AddElementToArray(arrToLoadInto, currentRecord) 'Add the next useful record
to the end of the array.
                End If
            End If
        End While
        oRead.Close() 'Close text file
        'Check if the null element has been romoved.
        If nullElementRemoved = True Then
            Return arrToLoadInto 'Return array of all menu records.
        Else
            'We dont want to return anything
            Debug.Print("Tried to load menu items from an empty file")
            Return Nothing 'Return nothing.
        End If
    End Function

    Public Sub AddMenuItemRecord(ByVal menuItemName As String, ByVal menuItemPrice As Single,
ByVal isFood As Boolean)
        Dim recordToAdd As MenuItemRecord 'The record to add to arrRecords

        'Load data into recordToAdd from parameters. Note that the ID isn't decided on yet, we
determine that in the next IF statements.
        recordToAdd.itemName = menuItemName
        recordToAdd.itemPrice = menuItemPrice
        recordToAdd.isFood = isFood

        'Check if the text file is empty.
        If TextFileIsEmpty("menu.txt") = True Then
            Dim nextID As Integer = 0 'The ID of the next record we want to create.
            recordToAdd.itemID = nextID 'Set ID of record we want to add to nextID
            Dim arrRecords() As MenuItemRecord = {recordToAdd} 'Declare array of menu item
records and set to the value of the record we want to add.
            SaveMenuToFile(arrRecords) 'Save new array to file
        Else
            Dim arrRecords() As MenuItemRecord = LoadMenuItems() 'Create an array of all users,
and load data from file into it
            Dim nextID As Integer = UBound(arrRecords) + 1 'The ID of the new record (1 more than
the previous ID)
            recordToAdd.itemID = nextID 'Set ID of record we want to add to nextID
            AddElementToArray(arrRecords, recordToAdd) 'Add recordToAdd to the end of array
arrRecords
            SaveMenuToFile(arrRecords) 'Save new array to file
        End If
```

```
    End Sub

    Private Sub SaveMenuToFile(ByVal arrToSave() As MenuItemRecord)
        Dim sep As Char = "|" 'The separator character between elements in the text file.
        Dim writeLine As String = "" 'The string to be written to file
        'Loop through the array that we want to save
        For i = 0 To UBound(arrToSave)
            writeLine &= arrToSave(i).itemID & sep & arrToSave(i).itemName & sep &
arrToSave(i).itemPrice & sep & arrToSave(i).isFood & vbCrLf 'Add the current record to writeLine
        Next
        File.WriteAllText(filePath & "\menu.txt", writeLine) 'Override the text file
    End Sub

    Public Sub DeleteMenuItem(ByVal menuItemToDelete As MenuItemRecord)
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menu items
        RemoveElementFromArray(menuItems, menuItemToDelete.itemID) 'Remove menu item at a
specific index.
        For i = 0 To UBound(menuItems) 'Loop through arrMenuItems
            menuItems(i).itemID = i 'Change the itemID field of the current menu item to its
current index in the array.
        Next
        SaveMenuToFile(menuItems) 'Save array to file.
    End Sub

    Public Sub EditMenuItem(recordToEdit As MenuItemRecord)
        Dim menuItems() As MenuItemRecord = LoadMenuItems() 'Array of all menuItem records.
        menuItems(recordToEdit.itemID) = recordToEdit 'Override existing record with new record.
        SaveMenuToFile(menuItems) 'Save new array to file.
    End Sub

    Public Sub ClearMenuFile()
        File.WriteAllText(filePath & "\menu.txt", "") 'Delete all text from the text file.
    End Sub

#End Region

#Region "Orders"
    Public Function LoadOrders()
        Dim arrToLoadInto(0) As OrderRecord 'The array we want to load the data into
        Dim nullElementRemoved As Boolean = False 'Has the first null element been removed yet?
        Dim oRead As System.IO.StreamReader 'For sequentially accessing the contents of text
files.
        Dim lineIn As String 'The line read by oRead that is then processed.
        Dim tmp 'Temporary variable for storing data while it is processed and placed into
arrToLoadInto.

        oRead = File.OpenText(filePath & "\sales.txt") 'Locate text file

        While oRead.Peek <> -1 'while there are still lines left to read
            lineIn = oRead.ReadLine() 'Set lineIn to current line
            If Mid(lineIn, 1, 1) <> "" Then 'if there are characters on the line
                tmp = Split(lineIn, "|") 'Split the line, separating the elements by the “|”
character
                Dim currentRecord As OrderRecord 'The current orderRecord that we are examining
                'Populate fields in currentOrder
                currentRecord.orderID = tmp(0)
                currentRecord.orderDate = tmp(1)
                currentRecord.tableNumber = tmp(2)
                currentRecord.orderSubtotal = tmp(3)
                currentRecord.orderPaid = tmp(4)
                currentRecord.employee = tmp(5)
                currentRecord.customerComments = tmp(6)
                'Check if the null element has been removed or not.
                If nullElementRemoved = False Then
                    'i.e. The array still contains its initial null element and this element
needs to be overrided
```

```
                    arrToLoadInto(0) = currentRecord 'Override the null element with the
currentOrder we just loaded
                    nullElementRemoved = True 'Set nullElementRemoved to true
                Else
                    'i.e. The array contains only useful data, add the next useful record on the
end
                    AddElementToArray(arrToLoadInto, currentRecord) 'Add currentOrder to the end
of arrToLoadInto
                End If
            End If
        End While
        oRead.Close() 'Close text file
        'Check if null element has been removed
        If nullElementRemoved = True Then
            Return arrToLoadInto 'Return array of all orders
        Else
            Debug.Print("Tried to load order records from an empty file") 'File was empty
            Return Nothing 'Return Nothing
        End If
    End Function

    Public Sub AddOrderRecord(ByVal orderDate As Date, ByVal tableNumber As Integer, ByVal
orderSubtotal As Single, ByVal orderPaid As Boolean, ByVal employee As String, ByVal
customerComments As String)
        Dim recordToAdd As OrderRecord 'The record to add to arrRecords

        'Load data into recordToAdd from parameters. Note that the ID isn't decided on yet, we
determine that in the following IF statements.
        recordToAdd.orderDate = orderDate
        recordToAdd.tableNumber = tableNumber
        recordToAdd.orderSubtotal = orderSubtotal
        recordToAdd.orderPaid = orderPaid
        recordToAdd.employee = employee
        recordToAdd.customerComments = customerComments

        'Check if text file is empty
        If TextFileIsEmpty("sales.txt") = True Then
            Dim nextID As Integer = 0 'The ID of the next record we are going to add.
            recordToAdd.orderID = nextID 'Set the orderID of recordToAdd to nextID.
            Dim arrRecords() As OrderRecord = {recordToAdd} 'Declare array of orderRecords and
set to the value of the record we want to add.
            SaveOrderToFile(arrRecords) 'Save new array to file
        Else
            Dim arrRecords() As OrderRecord = LoadOrders() 'Create an array of all orders, and
load data from file into it
            Dim nextID As Integer = UBound(arrRecords) + 1 'The ID of the new record (1 more than
the previous ID)
            recordToAdd.orderID = nextID 'Set the orderID of recordToAdd to nextID.
            AddElementToArray(arrRecords, recordToAdd) 'Add recordToAdd to the end of array
arrRecords
            SaveOrderToFile(arrRecords) 'Save new array to file
        End If
    End Sub

    Private Sub SaveOrderToFile(ByVal arrToSave() As OrderRecord)
        Dim sep As Char = "|" 'The separator character between elements in the text file.
        Dim writeLine As String = "" 'The string to be written to file
        'Loop through the array that we want to save.
        For i = 0 To UBound(arrToSave)
            writeLine &= arrToSave(i).orderID & sep & arrToSave(i).orderDate & sep &
arrToSave(i).tableNumber & sep & arrToSave(i).orderSubtotal & sep & arrToSave(i).orderPaid & sep
& arrToSave(i).employee & sep & arrToSave(i).customerComments & vbCrLf 'Add the current record to
writeLine
        Next
        File.WriteAllText(filePath & "\sales.txt", writeLine) 'Override the text file
    End Sub
```

```
    Public Sub EditOrderRecord(recordToEdit As OrderRecord)
        Dim allOrders() As OrderRecord = LoadOrders() 'Array of all order records.
        allOrders(recordToEdit.orderID) = recordToEdit 'Override existing record with new record.
        SaveOrderToFile(allOrders) 'Save new array to file
    End Sub
#End Region

#Region "General"
    Private Function TextFileIsEmpty(ByVal fileName As String)
        Try
            'Check if there is no text in the text file
            If File.ReadAllText(filePath & "\" & fileName) = "" Then
                Debug.Print(fileName & " is empty")
                Return True 'Return true
            Else
                Debug.Print(fileName & " contains stuff")
                Return False 'Return False
            End If
        Catch ex As Exception
            Debug.Fail("Tried to read data from a file " & fileName & ", which doesn't exist.")
            Return Nothing 'Return Nothing, as the text file doesn't exist.
        End Try
    End Function
#End Region

End Module
```

SEARCH UTILITIES

```
Module SearchUtilities

#Region "Basic Searches"
    Public Function ItemExistsInArray(Of T)(ByVal arrToSearch() As T, ByVal elementToSearchFor As
T)
        'Algorithm based on linear search

        'Note the try/catch statement is here in case a null array is passed in as a parameter.
        Try
            'Loop through the array that we want to search
            For i = 0 To UBound(arrToSearch)
                If arrToSearch(i).Equals(elementToSearchFor) Then 'Check if the element at the
current loop index matches the element we are searching for
                    Return True 'Return true
                End If
            Next
        Catch ex As Exception
            Debug.Print("Null array passed into function ItemExistsInArray()")
        End Try
        Return False 'Return false
    End Function
#End Region

#Region "Order Searches"
    Public Function FindRecordsWithSearchCriteria(ByVal arrToSearch() As OrderRecord, ByVal
searchCriteria As String)
        'Based on linear search

        Dim arrToReturn() As OrderRecord = Nothing 'Array for storing records that we want to
return

        'Note the try/catch statement is here in case a null array is passed in as a parameter.
        Try
            'Loop through array of items we are searching
            For i = 0 To UBound(arrToSearch)
                'Check if the current record contains the search criteria in either its subtotal,
employee, or customerComments fields
                If arrToSearch(i).orderSubtotal.ToString.ToUpper.Contains(searchCriteria.ToUpper)
Or arrToSearch(i).employee.ToString.ToUpper.Contains(searchCriteria.ToUpper) Or
arrToSearch(i).customerComments.ToString.ToUpper.Contains(searchCriteria.ToUpper) Then
                    AddElementToArray(arrToReturn, arrToSearch(i)) 'Add the current record to the
array of records that we want to return.
                End If
            Next
        Catch ex As Exception
            Debug.Print("Null array passed into function FindRecordsWithSearchCriteria()")
        End Try
        Return arrToReturn 'Return recordsToReturn.
    End Function
#End Region
End Module
```

SORT UTILITIES

```
Module SortUtilities

#Region "OrderRecordSorts"
    Public Function InsertionSortByOrderTotal(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the orderSubtotal at the index that we want to sort is greater than or
equal to the orderSubtotal at currentIndexOfLoop
                If arrToSort(indexToSort).orderSubtotal >=
arrToSort(currentIndexOfLoop).orderSubtotal Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort 'Return sorted array
    End Function

    Public Function InsertionSortByTableNumber(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the tableNumber at the index that we want to sort is greater than or
equal to the tableNumber at currentIndexOfLoop
                If arrToSort(indexToSort).tableNumber >=
arrToSort(currentIndexOfLoop).tableNumber Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
```

```
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function

    Public Function InsertionSortByEmployee(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the employee at the index that we want to sort is further in the
alphabet than the employee at currentIndexOfLoop
                If arrToSort(indexToSort).employee >= arrToSort(currentIndexOfLoop).employee Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function

    Public Function InsertionSortByDate(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
```

```
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the orderDate at the index that we want to sort is greater than or
equal to the orderDate at currentIndexOfLoop
                If arrToSort(indexToSort).orderDate >= arrToSort(currentIndexOfLoop).orderDate
Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function
#End Region

End Module
```

VALIDATION UTILITIES

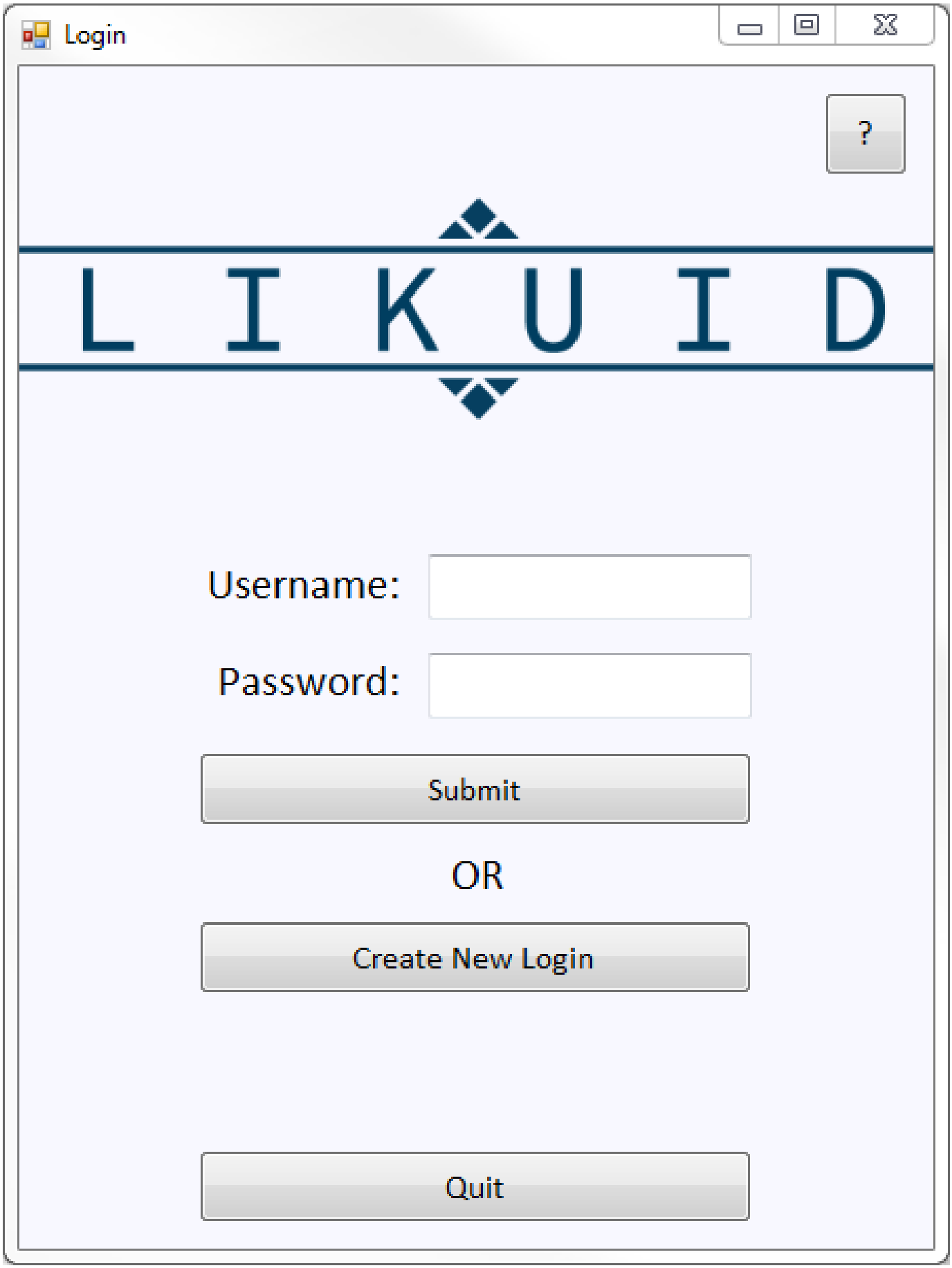
```
Module ValidationUtilities
    Public Function StringIsConvertibleToInteger(ByVal str As String)
        Try
            Integer.Parse(str)
            Return True
        Catch ex As Exception
            Return False
        End Try
    End Function

    Public Function StringIsConvertibleToSingle(ByVal str As String)
        Try
            Single.Parse(str)
            Return True
        Catch ex As Exception
            Return False
        End Try
    End Function
End Module
```

USER UTILITIES

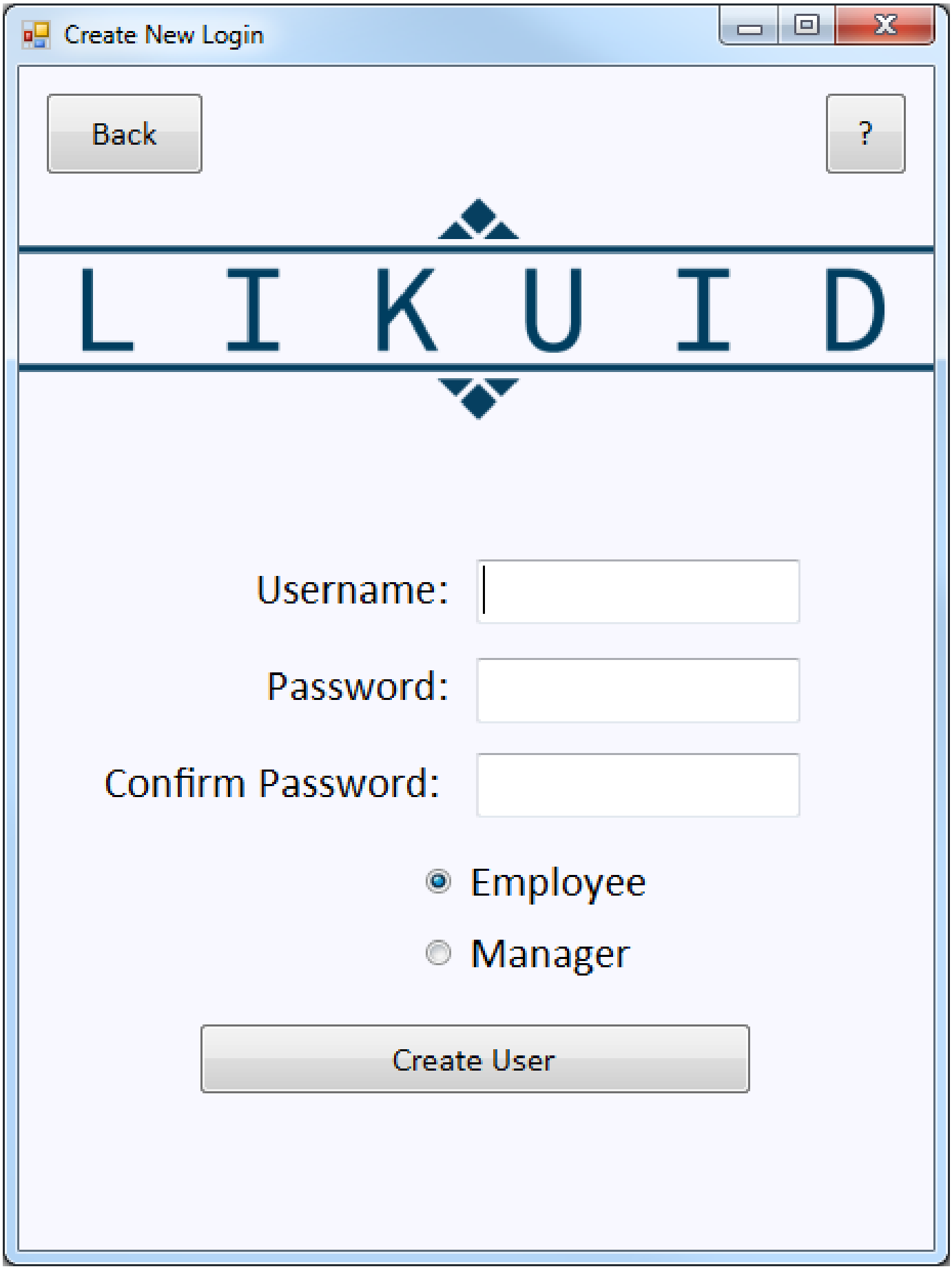
```
Module UserUtilities
    Public Function IsManager(ByVal userToCheck As String)
        Dim users() As UserRecord = LoadUsers()
        Try
            'Loop through users until we find the correct one.
            For i = 0 To UBound(users)
                If users(i).username = userToCheck Then
                    'Found the user
                    Return users(i).manager
                End If
            Next
        Catch ex As Exception
            'i.e. There is nothing in the users array
        End Try
        Return False
    End Function
End Module
```

USER INTERFACE

LOGIN



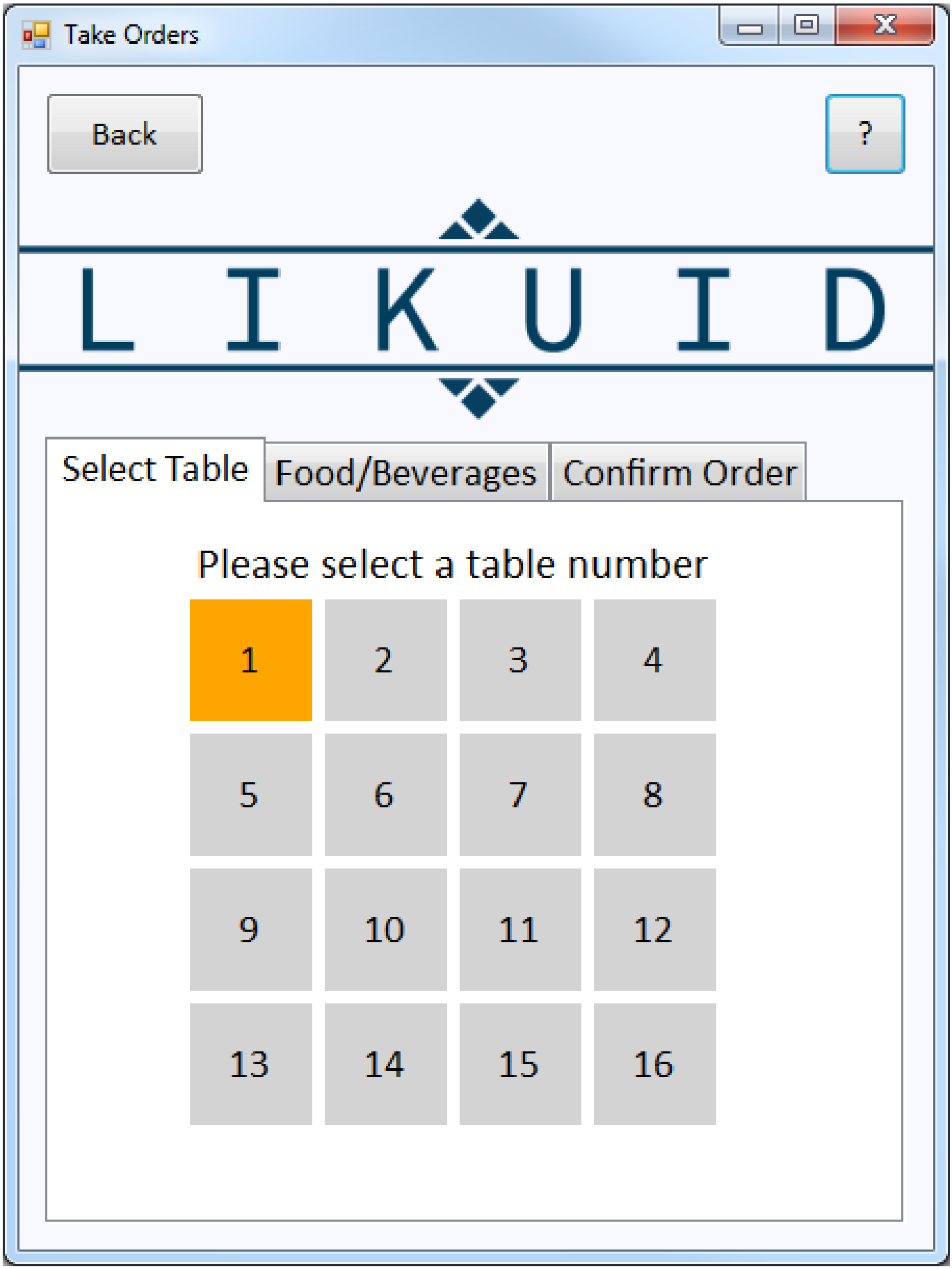
NEW LOGIN


Create New Login
Back
?
LIKUID
Username:
Password:
Confirm Password:
Employee
Manager
Create User

MAIN FORM


Main
?
LIKUID
Welcome, Alex
Take Orders
Pay for Orders
Change Menu
View Sales
Switch User
Quit

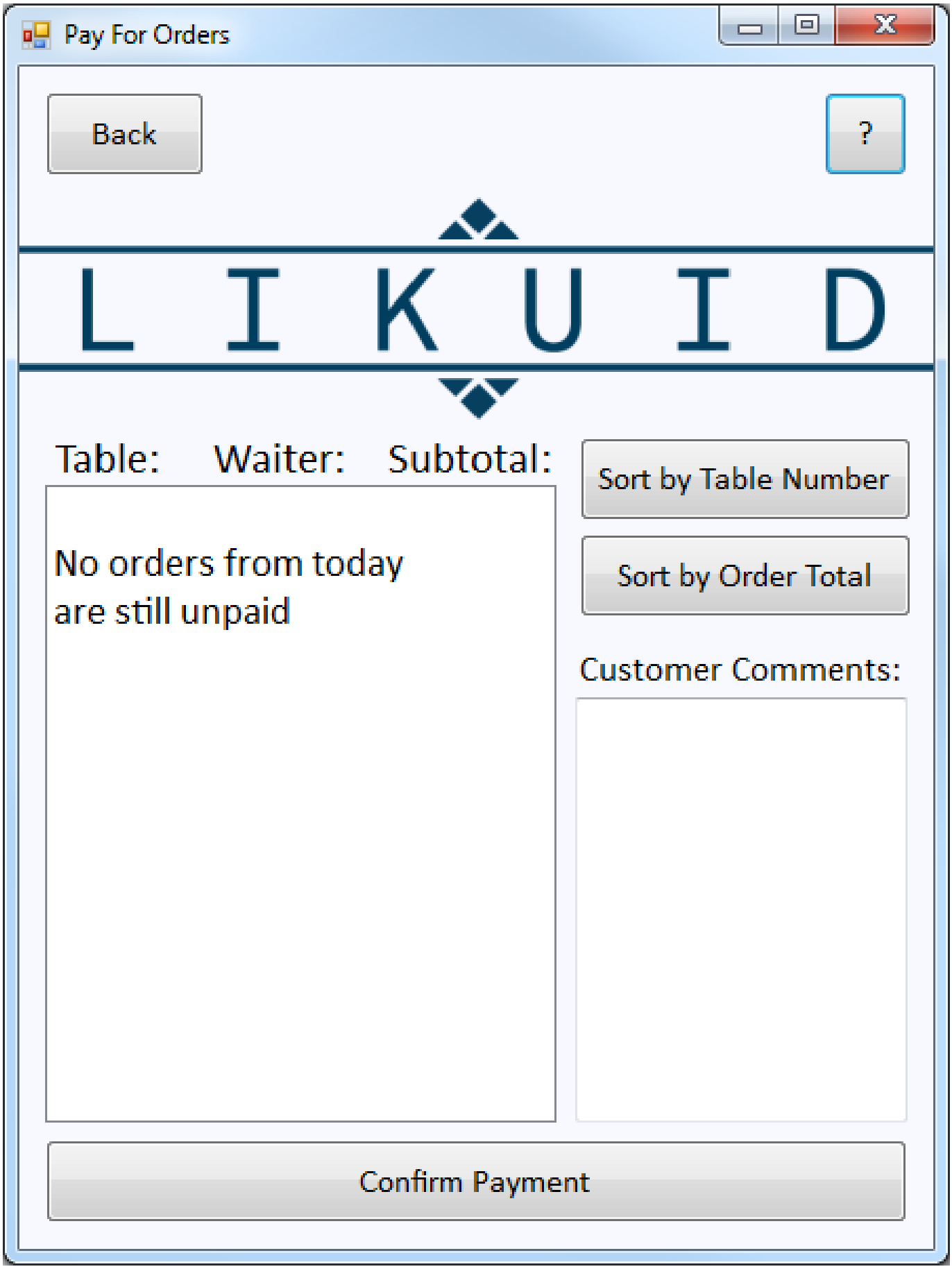
TAKE ORDERS


Take Orders
Back
?
LIKUID
Select Table
Food/Beverages
Confirm Order
Please select a table number
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


Take Orders
Back
?
LIKUID
Select Table
Food/Beverages
Confirm Order
Order Summary
Food:
Amount:
Add to List
Beverage:
Amount:
Add to List
Reset List


Take Orders
Back
?
LIKUID
Select Table
Food/Beverages
Confirm Order
Order Summary
Order Total:
$00.00
Confirm Order

PAY FOR ORDERS

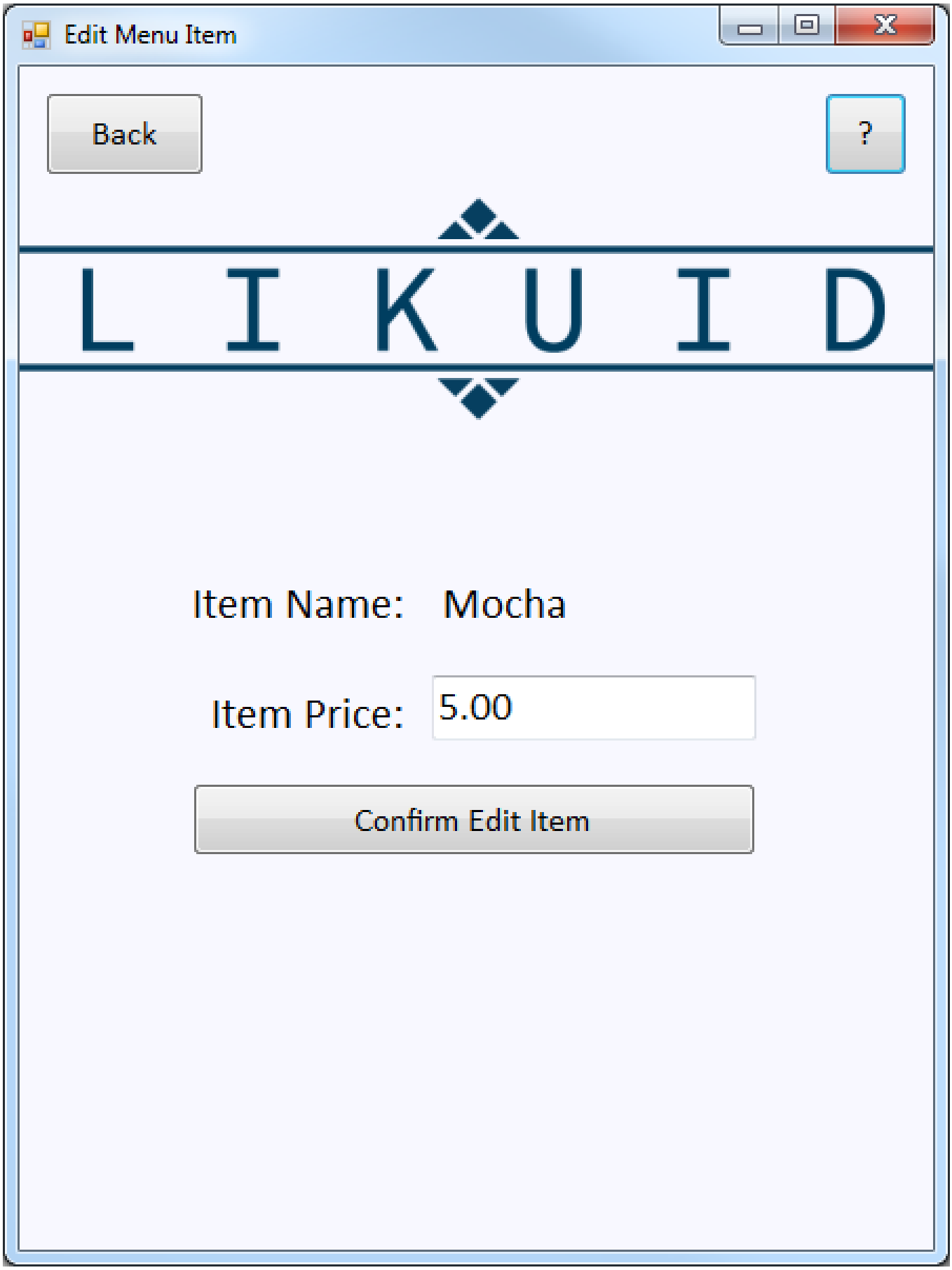


CHANGE MENU


Edit Menu
Back
?
LIKUID
Pancakes $11.00
Beer $6.00
Coke $3.50
Bacon $4.00
Nachos $14.00
Wedges $7.00
Cider $6.00
Coffee $4.50
Mocha $5.00
Add Menu Item
Delete Menu Item
Edit Menu Item
Reset Menu

ADD MENU ITEM

EDIT MENU ITEM



VIEW SALES

View Sales

Back

?

LIKUID

Search:

From: Thursday , 19 May 2016

To: Thursday , 19 May 2016

Order Date:	Total:	Waiter:
11/03/2016	$32.50	Alex
14/02/2016	$75.50	Canada
16/02/2016	$13.00	Alex
10/03/2016	$32.50	Canada
11/02/2016	$38.50	Canada
25/01/2016	$4.00	Alex
24/02/2016	$39.50	Alex

Sort by Date

Sort by Order Total

Sort by Employee

Inspect Record

TESTING AND MAINTAINING

TESTING OVERVIEW

Testing is essential to the software development process, as it enables the production of a high-quality final solution that meets all initial requirements and functions properly in a variety of environments.

There are various types of testing that are used to achieve different goals. Black box testing is the testing of a software product whereby the user is aware of the inputs and expected outputs, but is unsure of the explicit logic behind the program. This is useful for determining a user's reaction to the software package. White box testing is the opposite, and is usually carried out by the developer with the intent of removing logic, syntax and runtime errors; The exact processing that occurs is known to the user in this case.

Similarities can be drawn between white box testing and alpha testing, a method of testing conducted by the developer prior to the release of a product to make sure it performs as intended. Beta testing is similar to black box testing, as it is a way of gauging user experience and interaction through the release to the public.

Both alpha/white box and beta/black box testing are essential to the testing phase of software development, as together they provide a well-rounded view of the overall quality of the solution. However, not all of these methods of testing are applicable to the scope of my project. Alpha and white box testing are completed both informally during the implementation phase, as well as formally through a test data table. Examples of test data used include:

- Clicking button "Confirm Payment" before selecting an order to confirm – To test the correct sequencing of events.
- Set search criteria to "cAnaDa" – To test case sensitivity when performing searches
- Click "Confirm Edit Item", with "10" as the item price field – Makes sure that the expected result occurs.
- Username input "Mirrington", Password input "myPassword", Confirm password input "myPassword" – Test that the username is < 8 characters long.

More test data can be found in the test data table (see next section), however these few examples of test data illustrate the depth of testing required to ensure the program both works as intended, and is able to deal with unexpected data inputs.

Beta testing is not within the scope of my project, as I am not planning to release the software solution to real-world clients. It is however necessary for me to gauge the effectiveness of elements such as the user interface, help sections and inclusivity of the software solution using relevant black box testing techniques. Whilst these techniques were inflormal, often just observations of a user's experience, they were extremely valuable in the testing process, providing feedback on the elements of my project that needed improving.

In addition to the different types of testing, there are also multiple levels of testing that need to be completed. There are three main levels, namely module level testing, program level, and system level testing. System level testing involves all aspects of the final system, including hardware, software, data, personnel and procedures. It is therefore obsolete in the scope of my project, as my solution is only really a prototype for a café management system, and does not run on mobile devices like it would in a proper system. System tests involving hardware, software, data, personnel and procedures are therefore difficult to perform.

Module level testing is however highly relevant, as it is the testing of individual modules of code to ensure it is free of syntax, logic and runtime errors. Program level testing is also highly relevant, as it makes sure each module passes parameters to other modules correctly and the whole program works as intended. These types of tests are recorded formally using a test data table, as seen on the next page.

TEST DATA TABLE

TEST DATA	REASON FOR INCLUSION	EXPECTED RESULT	ACTUAL RESULT
LOGIN			
Username input “” Password input “”	Make sure no errors occur when there is no text to check against database	Display *“Username or password Incorrect”*	Pass
Username input “Alex” Password input “myPassword”	Make sure manager user can log in correctly	Main form loads, with all features available.	Pass
Username input “Canada” Password input “cupcakes”	Make sure employee can log in correctly	Main form loads, but access is restricted to taking and paying for orders, switching users and exiting the program.	Pass
Username input “canada” Password input “Cupcakes”	Make sure login details are case-sensitive	Display *“Username or password Incorrect”*	Pass
Username input “Fred” Password input “pw”	Make sure false login details don’t grant access to system.	Display *“Username or password Incorrect”*	Pass
NEW LOGIN			
Username input “” Password input “” Confirm password input “”	Make sure a new user is not created if no username and password are provided.	Display *“Please insert a username and password”*	Pass
Username input “Jarrie” Password input “myPassword” Confirm password input “myPass”	Make sure a new user is not created if the password and confirm password fields don’t match	Display *“Passwords don’t match”*	Pass
Username input “Mirrington” Password input “myPassword” Confirm password input “myPassword”	Make sure a new use is not created if the username has 8 or more characters in it.	Display *“Please insert a username that is less than 8 characters long”*	Pass
Username input “Alex” Password input “pw” Confirm password input “pw”	Make sure a new user is not created if the user already exists in the database.	Display *“User already exists in database. Please choose a new username.”*	Pass
Username input “Cameron” Password input “pw” Confirm password input “pw”	Make sure a new user is created if all data provided is valid.	Display *“New user created successfully”*, write user record to file and load main form.	Pass
TAKE ORDERS			
Click button “Add to List” before selecting a food or beverage	Make sure no items are added to the list box.	Display *“Please select the food/beverage you want to add”*	Pass

Click button “Add to List” before selecting a food or beverage or specifying an amount	Make sure no items are added to the list box.	Display *“Please select the food/beverage you want to add”*	Pass
Click button “Add to List” before specifying how many of the item you want to add.	Make sure no items are added to the list box.	Display *“Please select the amount of the food/beverage you want to add”*	Pass
Click button “Add to List”, giving number 1.5 as amount of food or beverage to add.	Make sure no items are added to the list box.	Display *“Please select the amount of the food/beverage you want to add”*	Pass
Click button “Add to List”, giving string “awd” as amount of food or beverage to add.	Make sure no items are added to the list box.	Display *“Please select the amount of the food/beverage you want to add”*	Pass
Click button “Reset List”	Make sure the current order list is reset and the display box is updated accordingly.	Reset the current order list and update the display box accordingly.	Pass
Click button ‘Confirm Order’ before adding any items to the order	Make sure no new order record is created.	Display *“Can’t confirm an order that doesn’t contain any items”*	Pass
Click button “Confirm Order” after items have been added to the order	Make sure an order record is created, appropriate feedback is given and the user is returned to the main menu.	Write new order Record to file, print receipt for the kitchen and return to main menu	Pass
PAY FOR ORDERS			
Load form Pay For Orders	Make sure all of the records displayed are unpaid orders from the current day.	Display all unpaid orders from the current day.	Pass
Click button “Sort by Table Number”	Make sure records are sorted by table number	Sort records by table number	Pass
Click button “Sort by Order Total”	Make sure records are sorted by order total	Sort records by order total	Pass
Click button “Confirm Payment” before selecting an order to confirm	Make sure no orders are confirmed and appropriate feedback is given.	Print “Please select an order to confirm”	Pass
Click button “Confirm Payment” after selecting an order to confirm	Ensure that the selected order is marked as paid and appropriate feedback is given.	Mark the selected order as ‘paid’ in the database and print *“Order marked as Paid”* to give feedback to user.	Pass
EDIT MENU			
Load form Edit Menu	Make sure all menu items are displayed along with prices.	Display all menu items along with prices.	Pass
Click “Add Menu Item”	Make sure user is taken to form for adding new menu items	Take user to formAddMenuItem	Pass

Click “Edit Menu Item” before selecting a menu item to edit.	Make sure appropriate instructions are given to inform the user on how to edit a menu item	Display *“Please select an item from the list to edit”*	Pass
Click “Edit Menu Item” after selecting a menu item to edit.	Make sure user is taken to form for editing menu items and correct data is passed between forms	Take user to formEditMenuItem, passing through the data that is to be edited.	Pass
Click “Delete Menu Item” before selecting an item to delete	Make sure that no menu items are deleted and appropriate feedback is given.	Display *“Please select an item from the list to delete”*	Pass
Click “Delete Menu Item” after selecting an item to delete	Make sure that the appropriate menu item is removed from the menu database and the list box is updated accordingly.	Remove the appropriate menu item from the database and update the list box accordingly.	Pass
Click “Delete Menu Item” when there are no menu items to delete	Make sure that appropriate user feedback is given and no runtime errors occur	Display *“Please create a menu item before trying to delete one”*	Pass
Click “Reset Menu” button at any time	Make sure that all menu items are removed from the menu database and the list box is updated accordingly.	Remove all menu items from the database and update the list box accordingly.	Pass
ADD MENU ITEM			
Click “Add Item” without putting any text in the Item Name field or Item Price field.	Make sure no new menuItemRecord is created and appropriate user feedback is given.	Display *“Please insert an item name.”*	Pass
Click “Add Item” without putting any text in the Item Name field and setting the Item Price field to 10	Make sure no new menuItemRecord is created and appropriate user feedback is given.	Display *“Please insert an item name.”*	Pass
Click “Add Item” without putting any text in the Item Price field, and setting the Item Name as “Nachos”	Make sure no new menuItemRecord is created and appropriate user feedback is given.	Display *“Price field is invalid. Make sure it only contains numbers”*	Pass
Click “Add Item”, setting Item Name to “Nachos”, item price to “14” and selected ‘food’ radio button	Make sure menu database is updated, all fields are cleared ready for next entry and appropriate user feedback is given.	Display *“New menu item added”*, update menu database and clear all fields ready for next entry.	Pass
Click “Add Item”, setting Item Name to “Cider”, item price to “4.5” and selected ‘Beverage’ radio button	Make sure menu database is updated, all fields are cleared ready for next entry and appropriate user feedback is given.	Display *“New menu item added”*, update menu database and clear all fields ready for next entry.	Pass
EDIT MENU ITEM			
Click “Confirm Edit Item”, with “” as the item price	Catch to make sure empty data is not fed into	Display *“Price field is invalid. Make sure it only*	Pass

field	the system. Make sure the record is not edited, and give appropriate user feedback.	*contains numbers”*	
Click “Confirm Edit Item”, with “a>b” as the item price field	Catch to make sure letters or symbols not fed into the system. Make sure the record is not edited, and give appropriate user feedback.	Display *“Price field is invalid. Make sure it only contains numbers”*	Pass
Click “Confirm Edit Item”, with “10” as the item price field	Make sure the existing record is edited and the user is returned to the previous form, as all the data is valid.	Edit existing record and return user to previous form.	Pass
VIEW SALES			
Click “Sort by Date” at any time	Make sure all currently displayed records are sorted by date as intended	Sort all currently displayed records by date.	Pass
Click “Sort by Order Total” at any time	Make sure all currently displayed records are sorted by order total as intended	Sort all currently displayed records by order total.	Pass
Click “Sort by Employee” at any time	Make sure all currently displayed records are sorted by employee as intended	Sort all currently displayed records by employee.	Pass
Click “Inspect Record” without first selecting a record to inspect.	Make sure processes are completed in correct order and appropriate instructions are provided	Display *“Please select an item to inspect”*	Pass
Click “Inspect Record” after selecting a record to inspect.	Make sure details of the record are disclosed as required.	Display details of the record that has been selected, including ID, date, waiter, subtotal and feedback.	Pass
“From” date picker set to day after “To” date picker	Make sure no searching occurs (to avoid runtime errors) and appropriate feedback is given.	Display *“Please ensure that the start date is before the end date”* and set the date in the “From” picker to that in the “To” picker.	Pass
“To” date picker set to day before “From” date picker	Make sure no searching occurs (to avoid runtime errors) and appropriate feedback is given.	Display *“Please ensure that the start date is before the end date”* and set the date in the “From” picker to that in the “To” picker.	Pass
“From” date picker set to 01/03/2016 and “To” date picker set to 01/04/2016	Make sure all records within that date range and that match the sort criteria are returned and displayed.	Display all records within that date range and that match the sort criteria	Pass
Set search criteria to “cAnaDa”	Make sure records are returned with the employee field “Canada”, regardless of upper or	Display all records within the current date range and that have employee field “Canada”	Pass

	lower case input.		
Set search criteria to “c”	Testing that strings are being checked properly. Make sure any records that contain a “c” in the employee, price or comments field are returned.	Display all records within the current date range and that contain a “c” in the employee, price or comments field.	Pass
Set search criteria to “2”	Testing that numbers are being checked properly. Make sure any records that contain a “2” in the employee, price or comments field are returned.	Display all records within the current date range and that contain a “2” in the employee, price or comments field.	Pass
Set search criteria to “*”	Make sure no records are returned, as no records contain the “*” character.	Display *“No sales found that match the given criteria”* and disable sorting and inspection buttons.	Pass

MAINTENANCE OVERVIEW

There are always updates or improvements that can be made to a software solution, due to the evolving nature of the software industry and need to keep up with competing products. There are many updates that could be undertaken in order to maintain and/or improve the functionality and quality of my project including:

- Additional metadata for special requests from customers.
    - For example, a customer might order a burger with no cheese – it is important that this extra message is passed through to the kitchen.
    - This could be implemented by adding an extra field to the orderRecord structure for keeping track of any special requests. These special requests would be printed on the receipt that is passed through to the kitchen.
- Include the ability to complete multiple orders for a table and automatically merge the data from the orders to calculate a grand total for the table.
    - This would be useful in the case that a customer wants to order drinks, then meals, then desserts. Instead of putting in three separate orders for the same table that need to be marked as paid, adding items to the original order would make it easier to complete the transaction process.
    - This is a major change that would require a major redesign of the system for taking and paying for orders, but could be completed by doing the following:
        - When a table is selected during the order taking process, the system checks if there is an existing unpaid order from today for that table.
        - If there isn't, then the order process continues as normal.
        - If there is, then when the order process continues as normal until the order is confirmed (in order to reduce the learning curve for the waiters). Wen the order is confirmed, a receipt is generated as usual (displaying only the items ordered this time around, i.e. not the items from the previous order that has already been served), but instead of generating a new orderRecord and writing this data to file, the order subtotal for the second order is added to the order subtotal of the first, and the updated orderRecord is saved to fie. This way when it is time for the customers to pay for their order, there will be only one order to be marked as paid, and the subtotal of multiple orders will not need to be added up.
- Adding the ability to keep track of currently used tables, in order to avoid confusion.
    - This could be achieved by implementing an additional form for viewing and updating the state of tables.
    - Each table would have one of two states: 'Empty' or 'Taken' (represented by an array of Boolean variables stored in a text file.)
    - As customers arrive, the waiters can easily see which tables are taken from this new form and seat the customers at an empty table. They then mark the table as being 'Taken'.
    - Similarly, as customers leave, waiters mark the table as 'Empty', ready for the next customer. If this change was implemented alongside the previous one (Merging data from multiple orders), then tables could automatically be marked as 'Empty' once the orderRecord for that table is marked as 'Paid', increasing efficiency.

If I were to implement any of these changes in the future, it would be vital to ensure that all documentation from the previous version is kept as a backup, and that a revised set of internal and external documentation is created. This ensures that ALL aspects of the solution remain up to date, not just the functional parts. This can be done by following the steps of the structured approach again: Re-evaulate the problem to be solved, plan and document changes, implement the changes, test them thoroughly and of course return to the maintenance stage. This constant revising and updating of software systems is absolutely essential in maintaining a high quality, functional and successful solution in today's constantly evolving software market.

EVALUATING

SOCIAL AND ETHICAL ISSUES OVERVIEW

Identify and describe the relevant social and ethical issues associated with your project, including IP, Quality, Inclusivity, Privacy and Ergonomics.

The development of any software product requires considerations of relevant social and ethical issues including Intellectual Property (IP) rights and copyright laws, quality, inclusivity, privacy and ergonomics. These considerations are necessary in order to produce a solution that is legally sound and is accepted by users.

Intellectual Property (IP) refers to any material that results from mental labour. As soon as this material is created, it belongs to the creator. Intellectual Property and copyright laws aim to ensure that creators are rewarded for their efforts. Developers are required to credit creators for their work instead of claiming it as their own. It is thus vital that any code or ideas developed by someone else are only used in a project if the creator gives their consent, and used only in the manner stated by the creator. These concepts are relevant to my project as I am creating original work whilst also learning from the work of others. Hence I need to make sure I am protecting my own intellectual property as well as acknowledging the work of others.

Another idea to be considered during development is the overall quality of the poduct. Developers are responsible for providing customers with high-quality products that work as intended. A high quality software product is:

- Able to perform as intended and advertised
- Consistent in its operation
- Efficient
- Able to keep data secure
- Easy to use from an end-user's perspective
- Easy to maintain, reuse, test and modify
- Able to function with other hardware and software.

Failure to meet these quality constraints results in an unsuccessful product that is socially and ethically unsound. Whilst my project is not going to be released for public use, it is still essential to consider the aspects that improve the quality of the product. For example: it is still important that my solution performs as intended and is consistent in its operation, however efficiency and data security aren't of utmost importance as I am not launching my system in a real-world system.

Inclusivity is another element that needs to be addressed carefully when developing a software solution. A non-inclusive software product is not only regarded as socially and ethically incorrect, but also limits the user base for the application. It is hence in the interests of the developer to make the solution as inclusive as possible. This can be done by considering:

- Cultural background e.g. American dates are presented differently to ours.
- Economic background e.g. making the product accessible to all, regardless of financial situation.
- Gender e.g. using neutral colour schemes
- Disabilities e.g. including support for colour blindness and hearing disabilities.

Not all of these are relevant to the scope of my project, however they still require consideration; Neutral colour schemes and support for colour blindness are much more relevant to my project than economic or cultural background due to the fact that the product will not be released for public use.

As a developer, it is essential to make sure that software solutions maintain the privacy of users and their information. Under the Privacy Act of 1988, and organisations that hold personal data must be able to:

- Explain why the data is being collected and for what purposes it will be used.
- Provide clients with access to their own records
- Maintain accurate information
- Disclose any other organisations that could be provided with the information
- Describe how the data is being stored.

Although not crucial to my project, privacy and ways of maintaining it need to be considered. The passwords used in my software solution are not encrypted during storage and retrieval, allowing access to user profiles. Whilst this is not ideal, it is not crucial to my project as no other personal information is stored with the

password like email addresses or credit cards. If I were to store such information, higher levels of security would be required.
An ergonomically sound software solution is critical to its success. Ergonomics is described as *"The study of the relationship between humans and their work environment"*. Consequently, an ergonomic software solution aims to maximise productivity and minimise confusion. This can be done by:

- Minimising response times
- Maintaining a consistent, easy to read user interface
- Using appropriate colours and fonts
- Maximising navigation efficiency and minimising confusion.
- Utilising keyboard shortcuts

These elements are of great importance to my project, as they determine how well the user interacts with the solution and ultimately whether they will be happy to continue using it.

Outline and justify your strategies for addressing relevant social and ethical issues in your project.

I applied a variety of strategies in order to address relevant social and ethical issues in my project.

In order to make sure I respected the intellectual property of others, I:

- Checked the policies of various online websites such as StackExchange and StackOverflow to see whether any answers that are shared are freed of copyright laws.
- Made sure to give credit to the original creator of any code that I used in my project after asking their permission. This credit was given with URLs embedded within the source code comments. If I were to release the application to the public, more visible recognition would be required.

My main reasons for doing this were to make sure that the original creators were rewarded for their efforts, and to make sure that I was not plagiarising their work. It should however be noted that the majority of the code is in fact my own intellectual property.

Strategies employed to increase product quality include:

- Constant revisiting of the problem statement and re-evaluation of whether the product was fulfilling the initial requirements. This leads to a higher quality product that is more likely to be accepted by the end user.
- Use of multiple levels of testing (white box, black box, module and project level testing) to ensure consistency of operation.
- Use of efficient algorithms where possible to avoid unnecessary processing times. This reduces user frustration as well as increases productivity.
- Use of passwords to add a level of security. Whilst these passwords are not encrypted, they do provide a basic level of security necessary for a small business.
- Detailed planning of the user interface layout and form navigation through the use of storyboards and UI mock-ups. This improves navigation and productivity through minimisation of confusion.
- Making code easy to maintain, reuse, test and modify by using modules for public subroutines and functions, avoiding global variables, using parameters where possible and maintaining adequate external, internal and intrinsic documentation. Although this has no direct impact on the end user, it makes implementation and maintenance easier from the developer's perspective, meaning future updates and features can be released more often.

I made my solution more inclusive by:

- Using a neutral colour scheme so that it was not gender biased. This shifts the focus to the function of the product instead of the visual aspects, a shift that is necessary for an ergonomic solution.
- Using colours that are not commonly confused, especially by people who have vision impairments such as colourblindness. I used a bright orange to indicate a checked radio button, making is easily recognisable. I also used high contrast colours such as black/navy and a white-blue in order to minimise confusion.

There was no need to worry about pricing or date management, as the solution was only developed for a single Australian-based client. If I were to develop a similar solution in the future for public release, these issues would need to addressed.

Privacy was maintained by:

- Ensuring passwords were required for login, in order to restrict access to sales data.
- Granting access to specific features such as viewing sales and editing the menu only to the manager of the café. This not only maintains privacy but improves efficiency and learning time for the other employees.
- Avoiding the collection of unnecessary data that would need protection such as email addresses or phone numbers. This avoids the problems associated with maintaining the privacy of information altogether.

However the privacy of the data used could have been much improved, through the use of:

- Encryption for storage of passwords
- A system that requires manager approval before a new manager user is made. This way multiple manager users could exist, but could not be created by anyone.

I made my solution more ergonomically sound by:

- Minimising response times through the use of efficient algorithms
- Maintaining a consistent, easy to read user interface
- Using appropriate colours and fonts

- Maximising navigation efficiency and minimising confusion.

Addressing these elements properly ultimately improves user experience and efficiency, a main goal of the software solution and highly relevant in the hospitality industry.

LEARNING JOURNAL

ONLINE BLOG

Blog URL:
http://alexmirrington12sdd.blogspot.com.au/

Term 4 Week 7 Update
Finally, the start of a new project. This week I have been thinking about what kind of software solution I want to create as well as setting up the initial components of my project, such as my portfolio and Gantt chart. This is the initial Gantt chart for my project:
(Gantt chart)
Throughout the project, I will be revising my Gantt chart in order to make sure that I am managing my time appropriately. This will help me to maximise the functionality and quality of my final solution and documentation.
So far I have a few ideas as to what kind of software solution I want to implement. The first is a management system for an online store, which would keep track of customer details, dates of purchases, and manage the inventory of the company. The second idea (which I am leaning towards at the moment) is a café management system that will perform functions like keeping track of reservations, orders, subtotals and previous purchases. I am thinking about working on this second option, as my sister is planning to start a café sometime in the near future, and it would be helpful to have a real-world client rather than establishing the requirements of the project myself. The next step is to send a few emails and establish the exact requirements for my project, as part of the Defining and Understanding stage of the project. I am aiming to have that completed by the end of next week.

Term 4 Week 8 Update
I have spent this week establishing the requirements for my software project with my client, as part of the defining stage of the development process.
The software solution that I am going to design and implement is a Café management system. My client is my sister, who is planning to start a small café sometime in the near future. She has asked me to make a management system to help her keep track of sales, update menu items and take orders in order to ensure the smooth running of her business. She also needs to keep track of who served which customers, so that she knows how her employees are performing in the workplace. The other main problem that she needs to address is the regular updating of the café menu each month. She has looked for existing software solutions that fulfil these requirements, but many are beyond her budget range and/or don't fulfil all of her requirements. Hence she has contracted me to develop a solution for her.
The main requirements that the solution must fulfil (as defined by my sister) are:

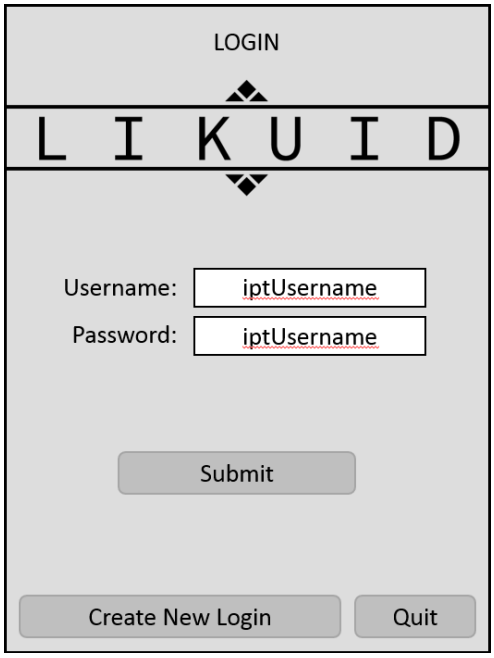
- A login screen, with separate users for each of the employees at the café. The username of the employee that is logged in will be recorded upon the completion of an order, along with the order details. The manager will also have a separate login that will give them access to functions such as changing menus, viewing all previous sales, etc.
- Ability to create new users as new employees begin work.
- Taking orders – this includes;
    - Selection of a table – This must be compatible with the reservation tracker
    - Selection of beverages/food
    - Review of an order (for repeating the order to the customer to confirm the order details are correct)
    - Paying for the order
- Saving the details of all completed orders to file for access at a later date. Each order record will contain:
    - Date that the order was made
    - Employee who served the customers
    - Order Subtotal
    - Additional comments from the customer
- Viewing, searching and sorting the details of past orders for calculations of tax returns, seeing how employees are performing and viewing customer feedback.
- Adding/Deleting menu items as the menu changes each month.

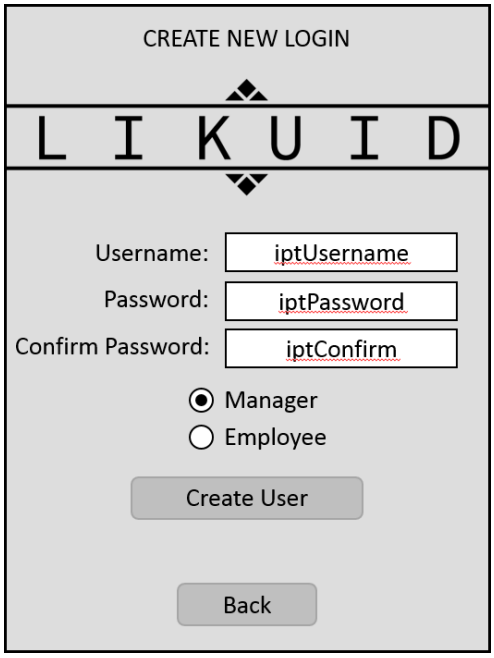
The aim of this project is to create a well-designed, functional café management system for my client. The main limitation of the project is time. Hence it will be necessary to work efficiently in order to complete the project in the required time frame, to the required level of detail. Another limitation is a lack of my technical

expertise. This will restrict my ability to learn new concepts within the specified time frame. In addition to this, the budget for my project is zero, and this will limit my ability to simplify the development process by using external APIs and tools. Thus, it is even more essential that I manage my time well and allow time for the development of new skills during the project.
Next week, I plan to begin work on some basic user interface designs as well as begin my discussion of system documentation. I am pleased with the progress I have made so far, and I have been keeping to the schedule of my Gantt chart reasonably well.

Term 4 Week 8 Update 2
I have successfully completed the defining and understanding phase of my project as of Friday this week. This involved the defining of a problem statement (See Term 4 Week 8 Update 1) as well as a discussion of my choice of the structured approach as my development approach. This can be found below:
The main objective of this project is to deliver a high quality café management system that fulfils all of the above client requirements. In order to make sure that the project is of the highest quality, time management and proper documentation will be essential. Hence the Structured Approach is an appropriate development process, as it will enable to manage my time effectively and document my project well, essentially increasing the quality of my final solution.
The structured approach is suited to medium-large projects that require careful planning and documentation before the actual implementation of the project. This is suited to the specifications of my project, as I need to properly define and plan my solution to make sure that it fulfils all of the requirements of my client before I begin implementing it.
The structured approach is also much more reliable and has a higher chance of success than other development approaches. Approaches such as Rapid Application Development (RAD) often result in failure to meet the initial requirements due to lack of documentation and time management. Although the scope of the projects developed using the RAD approach are usually smaller, they are often of lower quality due to lack of formal documentation and planning. Hence it is much better for me to use the structured approach for my project, as I am developing it for a real-world client, and I cannot afford to fall short of the client's requirements. I need to manage my time well and document the project properly according to the structured approach in order to ensure that I meet the time restrictions and project requirements.
Whilst RAD has the advantage of a short development time, the structured approach will be more suitable for my project, as my aim is not to complete the solution as quickly as possible (a characteristic of the RAD approach). Instead, I aim to create a high quality solution that fulfils all requirements, as I am developing my solution for a real world client. To do this takes time, but by using time management techniques like Gantt charts, I will be able to manager my time well and achieve my goal.
I am currently slightly ahead of my Gantt chart schedule, and hope to begin UI mock-ups at the beginning of next week.

Term 4 Week 9 Update
I began work on my User Interface mock-ups this week. I decided to do the user interface drafts before any of the other planning tools, as I am a visual planner and like to know how everything fits together before I plan anything in great detail. Hence doing the UI first helps me to translate my written requirements into a visual form, which I can use as a reference whilst creating my other planning documentation (Flowcharts, Data Flow Diagrams, Data Dictionaries etc.) I will not have a large amount of time this week, as it is the last week of school and end-of-year preparations will begin to take up the majority of my time. The main thing I've learned this week is the process behind turning text-based requirements into a visual interface that not only fulfils all of the requirements but is easy to use and aesthetically pleasing. Here is my progress so far:


LOGIN
L I K U I D
Username:
iptUsername
Password:
iptUsername
Submit
Create New Login
Quit


CREATE NEW LOGIN
L I K U I D
Username:
iptUsername
Password:
iptPassword
Confirm Password:
iptConfirm
Manager
Employee
Create User
Back

Term 1 Week 1-2 Update
I did not make any progress on my project during week 1 this term, due to the fact that it was only three days long and we were focusing more on the theory aspect of software design during class. Again, week 2 was similar, however I did manage to find some time to continue work on my UI designs. The theory that we have been covering is related to algorithm development, specifically standard algorithms such as linear search, binary search, bubble sort, selection sort and insertion sort. Throughout week 2, we completed a few practical activities related to these standard algorithms.
Here is the linear search algorithm that I produced:

```
    Public Function ItemExistsInArray(Of T)(ByVal arrToSearch() As T, ByVal elementToSearchFor As
T)
        'Perform Linear Search
        For i = 0 To UBound(arrToSearch)
            Debug.Print(i)
            If arrToSearch(i).Equals(elementToSearchFor) Then
                Return True
            End If
        Next
        Return False
    End Function
```

And here is the insertion sort algorithm that I came up with:

```
Public Function InsertionSortAscending(ByRef arrToSort() As Integer)
        Debug.Print("Insertion Sort")
        Dim firstIndex As Integer = 0 'the starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'the last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex

        While indexToSort <= lastIndex
            'i.e. while the array isnt sorted yet
            While currentIndexOfLoop < indexToSort
                If arrToSort(indexToSort) > arrToSort(currentIndexOfLoop) Then
                    'increase currentIndexOfLoop
                    currentIndexOfLoop = currentIndexOfLoop + 1
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As Integer = arrToSort(indexToSort) 'Bring the element to be sorted
into a temporary spot while we shift all previous items up into the spot
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp
                End If
            End While
            currentIndexOfLoop = firstIndex
            indexToSort = indexToSort + 1
            PrintItemsInArray(arrToSort)
        End While

        Return arrToSort
    End Function
```

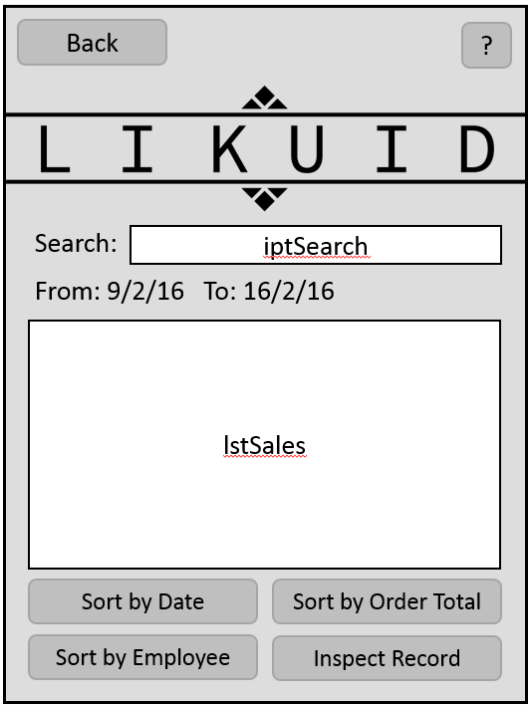
In the implementation of the search and sort algorithms, my aim was to make them as independent as possible, so I would not have to perform any modifications if I wanted to use them in my major project. All of the searches and sorts that I developed are functions that accept adequate data types and perform the correct processes, regardless of the information that is passed as a parameter.

Term 1 Week 3 Update
I am about two weeks behind schedule at the moment (according to my Gantt chart) due to the large amount of theory work that I have had to complete in the last few weeks. This week I plan to catch up to where I am supposed to be according to my Gantt chart so as not to run out of time. This will involve completing my IPO chart, UI designs, Storyboard, Context diagram, Data flow diagram and system flow chart. In essence, this means that the main skill I will be learning this week is time management.
Anyway, time complaints aside, here is what I achieved this week:

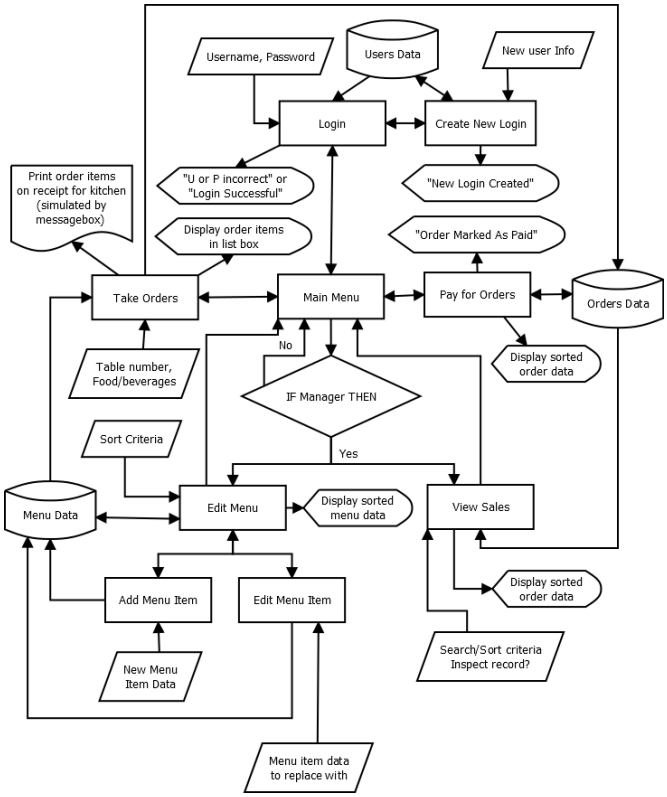
- Context Diagram
- UI designs
- Storyboard
- IPO chart
- Structure chart
- System flowchart
- Data flow diagram

Term 1 Week 4 Update
My focus this week was to polish up any discrepancies in my planning tools. Small details such as the inclusion of a help button in the corner of each UI screen needed to be changed on my UI mock-ups:

VIEW SALES

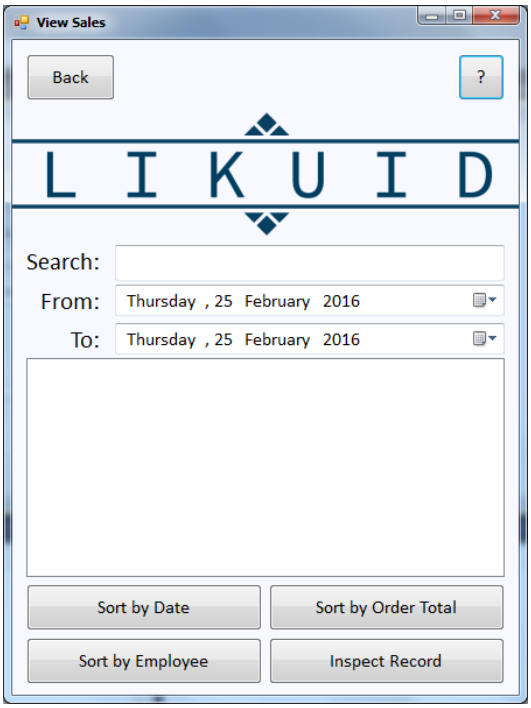


Other small things such as the omission of online displays on my system flowchart needed to be fixed in order to accurately represent my system. I also decided to change all of my diagrams to allow for the simulation of a paper document; The paper document seen in the system flowchart represents a printed receipt that would be sent to the kitchens at the café, with details on what food to prepare and which table to deliver to. (Note that I will be simulating this paper document with a message box in order to save time and paper) Here is the newest version of my system flow chart:



I also had various discussions with my teacher about the correct direction of parameter arrow in a structure chart. We came to the conclusion that because each box represents an individual subroutine/function, the parameter arrows must represent the actual parameters that are to be passed into the routine for processing. e.g. The 'CreateNewLogin' function accepts a username, password, and Boolean variable as input parameters, and returns whether the new login creation was successful or not.
See previous week's post for my structure chart.
Next week I plan to create my final UI designs in Visual Studio, and begin coding. I plan to create my pseudocode and flowchart algorithms as I code each module of my problem, as this will require less refinement on all my algorithms at the end of the project and essentially save time. I also plan to complete my data dictionary as I introduce new variables through code.

Term 1 Week 5 Update
My main task this week was to create all of my UI designs within Visual Studio. Whilst most of this was simply transferring my mock-up designs into Visual Studio, there were some other considerations that I had to make. Subtleties such as the exact size of forms and background colours had to be confirmed, so as to make sure each form was similar to the last (consistency of layout).
Similarly, fonts had to be decided upon as well as how large to make the fonts in order to balance fitting all of the controls on the screen, text readability and white space.
I also ended up changing my logo colour in order to make the UI seem a little more interesting and appealing, as my mock-up designs were all in black and white.
Here are some of the final UI designs for my café management system:



Next week I plan to start coding my solution, and developing my data dictionary, pseudocode and flowchart algorithms in parallel. This is to make sure that I am still planning out my project properly, but I can include adequate detail in my algorithms and data dictionaries; it is often difficult to know the exact nature of the processing that each function needs to perform until you code it, hence why I am planning and implementing in parallel.

Term 1 Week 6 Update
I completed coding my login module this week, which involves logging in as an existing user and creating a new user. The main thing that I learnt is how to process arrays effectively, including adding elements to arrays, removing elements from arrays and editing elements in arrays without causing errors. The main error that I ran into was a 'Null reference exception' error, which occurred when trying to add elements to empty arrays or removing items from empty arrays. I worked around this by using Try/Catch statements to check if the array is empty or not before doing any processing. Here is the code and pseudocode that I came up with for my login module:

And here is my module for adding, deleting and editing elements in arrays: (Note the try/catch statements in the event that an empty array is passed in as a parameter)

Next week I plan to continue implementing the rest of my solution and develop pseudocode and flowcharts to match.

Term 1 Week 7 Update
My main focus this week was on implementing, getting the functionality of the program happening. I began work on the form for adding, editing and deleting menu items first, as all of the other forms relied on the menu information being present before they would work.
The main thing that I learned this week was how to manage arrays and list-boxes. Up until now I had come to the conclusion that a list-box is simply a way of displaying the data held in an array. The main complications came when I had to get the index of a currently selected item in the list box and use this information to edit the array data appropriately. Here is an example of the code I came up with to do this:

```
Public Class formEditMenu

    Private Sub LoadedFormEditMenu(sender As System.Object, e As System.EventArgs) Handles
MyBase.Load
        Dim arrMenuItems() As MenuItemRecord = LoadMenuItems()
        For Each item As MenuItemRecord In arrMenuItems
            lstMenuItems.Items.Add(item.itemName & vbTab & Format(item.itemPrice, "Currency"))
        Next
    End Sub

    Private Sub ClickedButtonAddMenuItem(sender As System.Object, e As System.EventArgs) Handles
btnAddMenuItem.Click
        formAddMenuItem.Show()
        Me.Close()
    End Sub

    Private Sub ClickedButtonEditMenuItem(sender As System.Object, e As System.EventArgs) Handles
btnEditMenuItem.Click
        If ItemIsSelected() Then
            formEditMenuItem.Show()
            formEditMenuItem.GetItemToEditIndex(lstMenuItems.SelectedIndex)
            Me.Close()
        Else
            MsgBox("Please select an item from the list to edit")
        End If
    End Sub

    Private Sub ClickedButtonDeleteMenuItem(sender As System.Object, e As System.EventArgs)
Handles btnDeleteMenuItem.Click
        Dim arrMenuItems() As MenuItemRecord = LoadMenuItems()
        If Not lstMenuItems.Items.Count = 0 Then
            If ItemIsSelected() Then
                DeleteMenuItem(arrMenuItems(lstMenuItems.SelectedIndex))
                lstMenuItems.Items.RemoveAt(lstMenuItems.SelectedIndex)
            Else
                MsgBox("Please select an item from the list to delete")
            End If
        Else
            MsgBox("Please create a menu item before trying to delete one.")
```

```
        End If
    End Sub

    Private Sub ClickedButtonResetMenu(sender As System.Object, e As System.EventArgs) Handles
btnResetMenu.Click
        ClearMenuFile()
        lstMenuItems.Items.Clear()
    End Sub

    Private Function ItemIsSelected()
        If lstMenuItems.SelectedIndex = -1 Then
            Return False
        Else
            Return True
        End If
    End Function

    Private Sub ClickedButtonBack(sender As System.Object, e As System.EventArgs) Handles
btnBack.Click
        formMain.Show()
        Me.Close()
    End Sub

    Private Sub ClickedButtonHelp() Handles btnHelp.Click
        MsgBox("Click:" & vbCrLf & "'Add Menu Item' to add an item to the menu." & vbCrLf &
"'Edit Menu Item' to edit the price of an existing item" & vbCrLf & "'Delete Menu Item' after
selecting an item from the list to remove it from the menu" & vbCrLf & "'Reset Menu' to reset the
entire menu")
    End Sub

End Class
```

It should be noted that the position of each of the items in the list box corresponds to the position of the item in the array. Hence to edit or delete an element from the array, all I needed to do was access the index of the currently selected item in the list-box.
I also managed to complete all of my other forms except the 'ViewSales' form this week, though the only problems I ran into there were similar to the one described above, hence why the problem described above is the main thing that I learned this week.
Next week, I will be making sure that all of my pseudocode and flowcharts match my source code, and making sure that my data dictionary matched my source code also. I may also need to update my structure chart, as I ended up implementing a few more subroutines and functions than I originally anticipated.

Term 1 Week 8 Update
My main task this week is going through my source code and adding comments where necessary. This is essentially a part of making sure my source code and pseudocode match up, as the comments slightly resemble pseudocode; they explain the logic of what is occurring at each step within the program. Here is the commented source code from my ArrayUtilities module:

```
Module ArrayUtilities
    Public Sub AddElementToArray(Of T)(ByRef arrToAddTo() As T, ByVal elementToAdd As T)
        Try
            Dim nextID As Integer = UBound(arrToAddTo) + 1 'The next ID of the array that we want
to create.
            ReDim Preserve arrToAddTo(nextID) 'Open up new spot in array
            arrToAddTo(nextID) = elementToAdd 'Set value of new spot to the element we want to
add
        Catch ex As Exception 'Empty array parameter.
            Dim arrToReturn(0) As T 'Create new array with 1 element
            arrToReturn(0) = elementToAdd 'Set this element to the element we want to add.
            arrToAddTo = arrToReturn 'Set arrToReturn to this new array.
        End Try
    End Sub

    Public Sub RemoveElementFromArray(Of T)(ByRef arrToRemoveFrom() As T, ByVal
indexOfElementToRemove As Integer)
```

```
        Try
            Dim arrToReturn(UBound(arrToRemoveFrom) - 1) As T 'Create new array with 1 less
element than the parameter array.
            'Loop through the array that we want to remove from
            For i = 0 To UBound(arrToRemoveFrom)
                If i < indexOfElementToRemove Then 'If the current index is less than the index
of the element we want to remove then
                    arrToReturn(i) = arrToRemoveFrom(i) 'Copy the data from the parameter array
to the array that we want to return, keeping the same indices.
                ElseIf i > indexOfElementToRemove Then 'If the current index is greater than the
index of the element we want to remove then
                    arrToReturn(i - 1) = arrToRemoveFrom(i) 'Copy the data from the parameter
array to the array that we want to return, decreasing the index by 1
                End If
            Next
            arrToRemoveFrom = arrToReturn 'Return the final array
        Catch ex As Exception
            Debug.Print("Index doesn't exist in arrToRemoveFrom")
        End Try
    End Sub

    Public Sub PurgeArray(Of T)(ByRef arrToPurge() As T)
        arrToPurge = Nothing 'Set array to nothing
    End Sub

    Public Sub PrintItemsInArray(ByVal arrToPrint() As Integer)
        Try
            Dim printString As String = "" 'The string that we want to print
            'Loop through items of the array, and add them to printString
            For i = 0 To UBound(arrToPrint)
                If i = 0 Then
                    printString += arrToPrint(i).ToString()
                Else
                    printString += ", " & arrToPrint(i).ToString()
                End If
            Next
            Debug.Print(printString) 'Print printString
        Catch ex As Exception
            Debug.Print("Empty Array")
        End Try

    End Sub
End Module
```

The main thing that I have learnt this week is that commenting is just as important as the code itself. I found it is taking a while to comment all of my source code, as I have to re-understand the logic as I read the source code, and then convert this to a comment that is more understandable. If I had written the comments in as I was coding however, it would have been much easier to sift through my code and work out exactly what each subroutine is doing.
Next week I will continue to comment my source code and update my data dictionary to match the source code. I have the Easter weekend coming up, which will limit the amount of time I have to work on my project, but I'll see what I can get done.

Term 1 Week 9 Update
I continued to add comments to my source code this week, and update my data dictionary to match the source code. This week is only 3 days long, so it will be hard to get too much completed, but I will try my best.
I was commenting my sort algorithms this morning:

```
Module SortUtilities

#Region "OrderRecordSorts"
    Public Function InsertionSortByOrderTotal(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
```

```
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the orderSubtotal at the index that we want to sort is greater than or
equal to the orderSubtotal at currentIndexOfLoop
                If arrToSort(indexToSort).orderSubtotal >=
arrToSort(currentIndexOfLoop).orderSubtotal Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort 'Return sorted array
    End Function

    Public Function InsertionSortByTableNumber(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the tableNumber at the index that we want to sort is greater than or
equal to the tableNumber at currentIndexOfLoop
                If arrToSort(indexToSort).tableNumber >=
arrToSort(currentIndexOfLoop).tableNumber Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
```

```
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function

    Public Function InsertionSortByEmployee(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the employee at the index that we want to sort is further in the
alphabet than the employee at currentIndexOfLoop
                If arrToSort(indexToSort).employee >= arrToSort(currentIndexOfLoop).employee Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function

    Public Function InsertionSortByDate(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
```

```
            While currentIndexOfLoop < indexToSort
                'Check if the orderDate at the index that we want to sort is greater than or
equal to the orderDate at currentIndexOfLoop
                If arrToSort(indexToSort).orderDate >= arrToSort(currentIndexOfLoop).orderDate
Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort
    End Function
#End Region

End Module
```

The main thing that occurred to me is how much of the code is repeated. I know how to use generic types as parameters (See last week's post), but I cannot seem to find an efficient way of adapting my sort algorithms so that they can sort by different fields of a user-defined structure. I could combine all four sort routines into one routine and create an enum that acts as the field by which I want to sort and perform a different sorting function using multi-way selection, but that method seems like it would become very confusing rather quickly. Instead, I opted to just modify a standard insertion sort algorithm for each of the fields that I want to sort.

Next week, I plan to continue commenting code, updating pseudocode and flowcharts and updating my data dictionary.

Term 1 Week 10 Update

As I was commenting my code, I came across a logical error within my code, when trying to mark a specific order as 'paid for':

```
    Private Sub ClickedButtonConfirmPayment(sender As System.Object, e As System.EventArgs)
Handles btnConfirmPayment.Click
        If ItemIsSelected() Then
            'Add any customer comments to the order and mark it as paid.
            Dim selectedOrderInfo As String = lstOrders.SelectedItem 'The currently selected item
of the list box.
            Dim orderIDToEdit As Integer =
Integer.Parse(selectedOrderInfo.Substring(selectedOrderInfo.Length - 1, 1)) 'Declare
orderIDToEdit and set to the extracted orderID from the listbox display
            Dim allOrders() As OrderRecord = LoadOrders() 'Array of all orders ever made.
            allOrders(orderIDToEdit).orderPaid = True 'Change orderPaid flag of the appropriate
record to true.

            If Not iptCustomerComments.Text = "" Then
                allOrders(orderIDToEdit).customerComments = iptCustomerComments.Text 'Add any
customer comments to the appropriate record.
            End If

            EditOrderRecord(allOrders(orderIDToEdit)) 'Make edits to the order at orderIDToEdit
```

```
            DisplayArrayInListBox(GetCurrentUnpaidOrders()) 'Display all current unpaid orders in
the list box. (Note this removes the order we just edited, as it is now paid for)
            iptCustomerComments.Text = "" 'Reset customer comments
            MsgBox("Order Marked as Paid.")
        End If
    End Sub
```

As I tested the code, and went to pay for an order, the order record was not disappearing from the list box, as the 'orderPaid' flag was not being set to true. Upon closer inspection, I noticed that the orderRecord I was trying to edit had an ID of 10, which was not working with my code for extracting the orderID from the listBox:

```
Dim orderIDToEdit As Integer = Integer.Parse(selectedOrderInfo.Substring(selectedOrderInfo.Length
- 1, 1)) 'Declare orderIDToEdit and set to the extracted orderID from the listbox display
```

This is because I was extracting it by removing all of the letters in the string except for the last one. In this instance, I was getting an orderIDToEdit of '0' instead of '10'.

This has set me back a little bit, as I have to now fix this code, so that it works independently of the number if characters in the orderID. Admittedly my original solution was a bit of a 'dodgy fix', but this was more a result of lack of planning for this stage.
I plan to fix this issue (and any related issues) this week, and continue working on my project over the holiday break (This is the last week of term). I am hoping to begin the testing phase early next term.

Term 2 Week 3 Update
I have had a slow start to this term due to two weeks of exams, but I am making some steady progress now that I have some time to work on my project. I have finished commenting all of my code, and have fixed all (known) bugs in the code. I am currently updating my data dictionary, making sure every variable and parameter from code is documented adequately. It would take too much room to put the whole dictionary here, so here's a snippet:

NAME	DATA TYPE	SCOPE	DESCRIPTION	EXAMPLE	VALIDATION	FORMAT
GENERAL (Used multiple times)						
users	Array of UserRecords	Local	Local variable for holding data of all users.	0 "Alex" "myPassword" True 1 "Tom" "hisPassword" False	N/A	N/A
menuItems	Array of MenuItem Records	Local	Local variable for holding data of all menu items	0 "Pie" 3.5 True 1 "Coke" 3 False	N/A	N/A
allOrders	Array of Order Records	Local	Local variable for holding data of all orders ever made.	0 12/05/16 12 13.5 True "Alex" "Great nachos"	N/A	N/A

				1 13/05/16 5 40 False “Tom” “None”		
nextID	Integer	Local	Local variable for keeping track of the next ID that we want to open up for use in an array.	4	N/A	N/A
arrToReturn	Generic	Local	The array of items that we want to return to the user.	0	N/A	N/A
sender	Object	Local	Variable used as a parameter for subroutines that are called by user interactions, indicates the UI object that called the subroutine.	togTable2	N/A	N/A
e	Event	Local	Variable used as a parameter for subroutines that are called by user interactions, indicates the exact event that took place when calling the subroutine.	togTable2. Checked Changed	N/A	N/A
LOGIN						
currentUser	String	Public	Public variable for keeping track of the user that is currently logged in.	“Alex”	N/A	N/A
inputUsername	String	Local	Parameter passed into CheckLogin ( )	“Alex”	Length < maxUserLength	N/A

			function, the username we want to check against the users database.			
inputPassword	String	Local	Parameter passed into CheckLogin ( ) function, the password we want to check against the users database.	“myPassword”	N/A	N/A
NEW LOGIN						
maxUserLength	Integer	Private	Constant for deciding the maximum username length when creating new users	16	N/A	N/A
userToSearch	String	Local	Parameter passed into UserAlreadyExists ( ) function, the username we want to check if it exists or not.	“Tom”	N/A	N/A
usernames	Array of Strings	Local	Array of strings for holding the usernames of all users.	“Alex” “Tom”	Same length as Users ( )	N/A

The main thing I learned about data dictionaries is the difference between validation and format. Format refers to how the data is displayed/represented, whereas validation refers to the specific restrictions that are placed on the data so as to avoid runtime errors.
Another thing that I clarified was the difference between different access modifiers;
"Local" refers to a variable or parameter that is specific to a certain subroutine/function
"Private" refers to a variable or constant that is only accessible from within the class in which it is declared.
"Public" refers to a variable or constant that can be accessed by any class.

I plan to continue updating all of my diagrams in the planning and designing phase to make sure they match my source code properly (This includes adding in a few extra parts to my structure chart, as well as checking DFDs) before moving onto the testing phase. I hope to begin testing in early week 5.

Term 2 Week 4 Update

We covered some theory content this week, mainly the benefits of developing good coding practices. Practices such as developing standard reusable subroutines and functions, commenting code, using data structures appropriately as well as version control all help to simplify the development process and testing phase, as well as making the code easier to understand later, thus improving maintainability. I decided to have a look through my code and make sure that I was employing all of these strategies; Here is an example of a reusable function:

```
    Public Function IsManager(ByVal userToCheck As String)
        Dim users() As UserRecord = LoadUsers()
        Try
            'Loop through users until we find the correct one.
            For i = 0 To UBound(users)
                If users(i).username = userToCheck Then
                    'Found the user
                    Return users(i).manager
                End If
            Next
        Catch ex As Exception
            'i.e. There is nothing in the users array
        End Try
        Return False
    End Function
```

This function checks if a user is a manager or not. It is located in a module, and can be accessed from anywhere within the project. The try/catch statement ensures that no runtime errors occur even if no users exist yet. Because of this, it doesn't matter what data is passed into the function, it will always return an appropriate value. This function does however lack some intrinsic commenting to explain exactly what is occurring. A better example of commenting can be found in my sorting subroutines:

```
    Public Function InsertionSortByOrderTotal(ByRef arrToSort() As OrderRecord)
        Dim firstIndex As Integer = 0 'The starting index of the array
        Dim lastIndex As Integer = UBound(arrToSort) 'The last index of the array
        Dim indexToSort As Integer = firstIndex + 1 'The index of the array that we are trying to
insert into the correct position. Note we start from 1 more than the start index as we are going
to assume that the first element is already sorted
        Dim currentIndexOfLoop As Integer = firstIndex 'The current index of the inner loop, for
keeping track of which item we are currently comparing to the item we are sorting.

        'While the array isnt sorted yet
        While indexToSort <= lastIndex
            'Loop through array, starting from the beginning of the array until the index that we
want to sort.
            While currentIndexOfLoop < indexToSort
                'Check if the orderSubtotal at the index that we want to sort is greater than or
equal to the orderSubtotal at currentIndexOfLoop
                If arrToSort(indexToSort).orderSubtotal >=
arrToSort(currentIndexOfLoop).orderSubtotal Then
                    currentIndexOfLoop = currentIndexOfLoop + 1 'Increase currentIndexOfLoop
                Else
                    'we want to insert arrToSort(indexToSort) into the array at the index
currentIndexOfLoop
                    Dim temp As OrderRecord = arrToSort(indexToSort) 'Bring the element to be
sorted into a temporary spot while we shift all previous items up into the spot
                    'Loop through items in reverse order, starting from the index that we want to
sort.
                    For i = indexToSort To currentIndexOfLoop + 1 Step -1
                        arrToSort(i) = arrToSort(i - 1) 'Shift elements up through array
                    Next
                    arrToSort(currentIndexOfLoop) = temp 'Insert arrToSort(indexToSort) into the
array at the index currentIndexOfLoop
                End If
            End While
            currentIndexOfLoop = firstIndex 'Reset currentINdexOfLoop to the first index of the
array.
```

```
            indexToSort = indexToSort + 1 'Increment indexToSort by 1
        End While

        Return arrToSort 'Return sorted array
    End Function
```

More comments are required, as it is a much more complex algorithm. By commenting, I can easily see how the algorithm works, making testing and maintaining the software solution easier. It should also be noted that many of my sort algorithms are repeated. I discussed this issue in my Term 1 Week 9 Update, as it is the main example in my project of subroutines that could be improved to avoid the repetition of code. I did however come to the conclusion to have four separate but similar sorting algorithms in order to improve understandability despite the repeated code.

We also covered testing and debugging methods this week, such as console output statements, stubs, breakpoints, peer checking, desk checking and drivers. I have used a variety of these during the implementing phase to ensure loops repeat the correct number of times and statements are being executed in the correct order. Here is an example of a subroutine that I wrote for printing the contents of an array to the console for examination:

```
    Public Sub PrintItemsInArray(ByVal arrToPrint() As Integer)
        Try
            Dim printString As String = "" 'The string that we want to print
            'Loop through items of the array, and add them to printString
            For i = 0 To UBound(arrToPrint)
                If i = 0 Then
                    printString += arrToPrint(i).ToString()
                Else
                    printString += ", " & arrToPrint(i).ToString()
                End If
            Next
            Debug.Print(printString) 'Print printString
        Catch ex As Exception
            Debug.Print("Empty Array")
        End Try

    End Sub
```

This subroutine made the implementation of searching and sorting algorithms much easier, as I could examine the contents of the array after each loop.

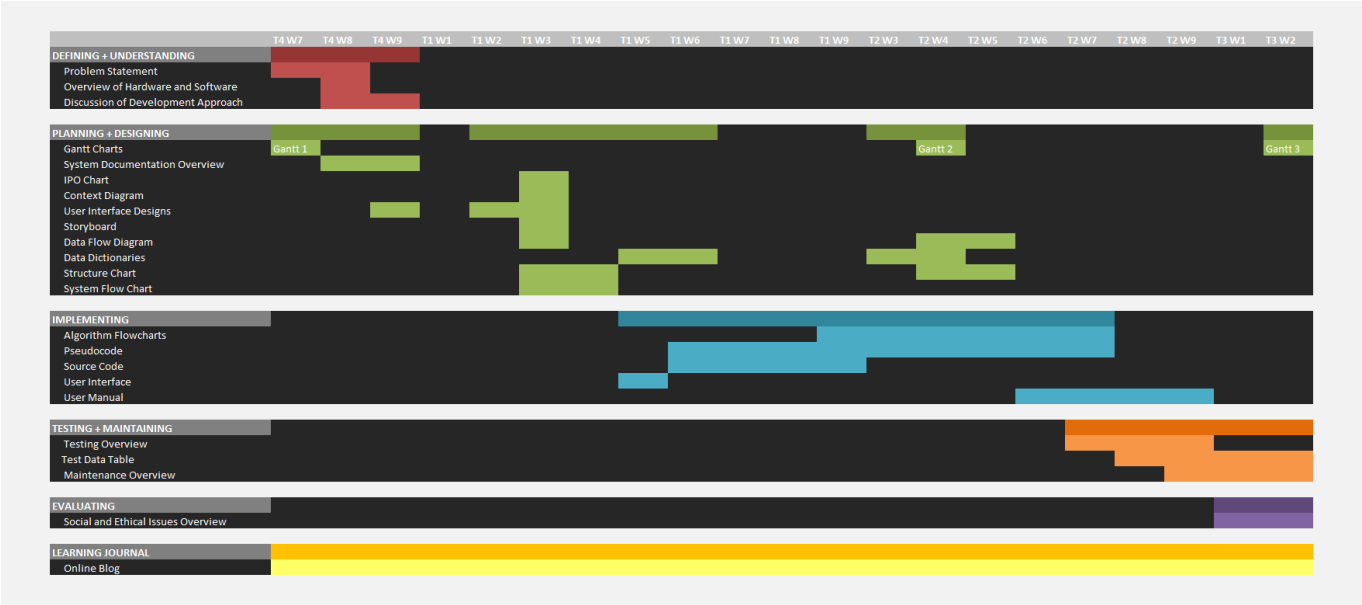
I am continuing to work through my pseudocode and flowcharts, however they are taking longer than expected, as formatting them takes a long time. I am also currently producing a revised Gantt chart to help manage the remainder of my time, this will be finalised next week.

Term 2 Week 5 Update

I finalised my second Gantt Chart today, splitting up the remainder of my time amongst the tasks I have left. I have managed my time surprisingly well and I have completed all of the tasks that I was aiming to have done by this time on my initial Gantt Chart. Here is a comparison of my initial and revised Gantt charts:

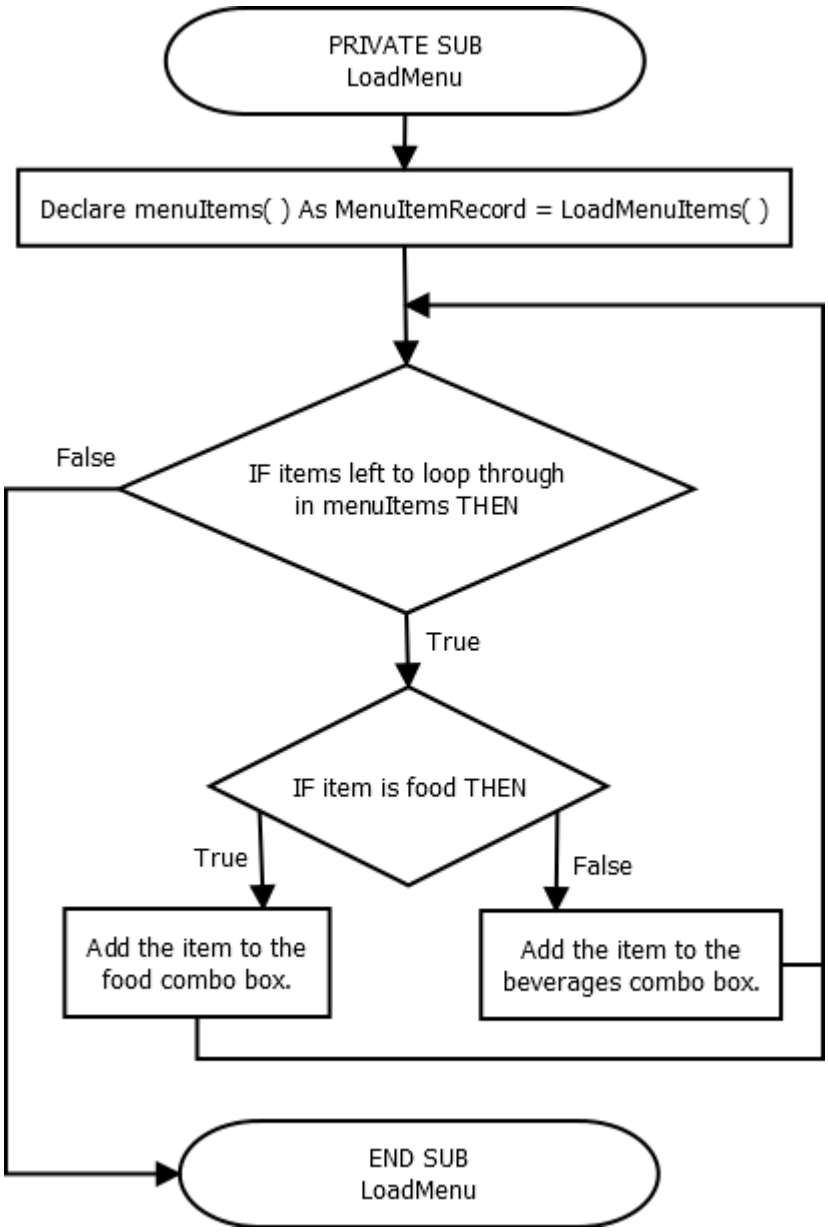
INITIAL

REVISED



I am continuing to work on my pseudocode and flowcharts, like I said last week, formatting them takes a lot of time. I am however making steady progress. Here is an example of a pseudocode and matching flowchart algorithm for a module of code in my TakeOrders form:

```
PRIVATE SUB LoadMenu( )
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   FOR EACH MenuItemRecord item IN menuItems
      IF item is food THEN
         Add the item to the food combo box.
      ELSE
         Add the item to the beverages combo box.
      END IF
   NEXT
END SUB
```



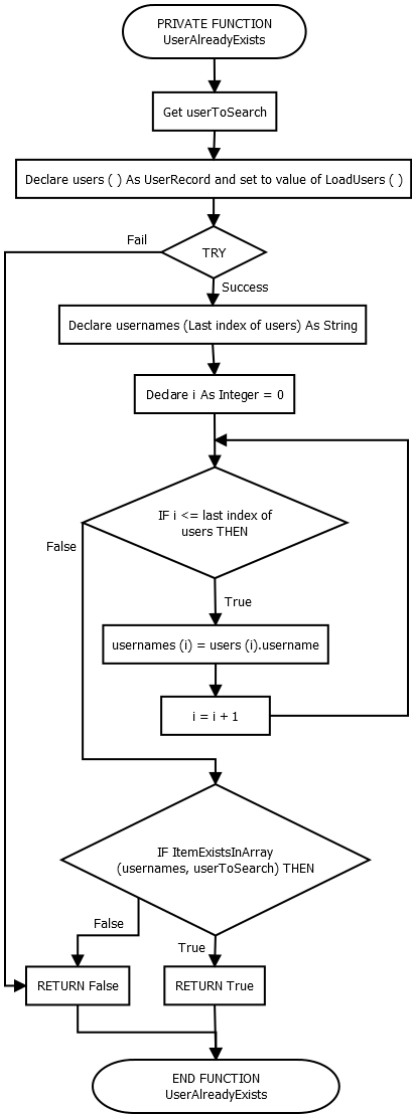
The thing I am finding hardest and learning most about is representing repetition structures such as pre-test loops in flowchart form, as there is not explicit symbol to represent a looping structure; Instead a binary decision symbol has to be used in a specific way. This can be seen in the examples above. Apart from this main trouble, the production of my pseudocode and flowcharts is going quite smoothly.

Next week, I will keep working on my pseudocode and flowcharts.

Term 2 Week 6 Update
I am continuing to work on my pseudocode and flowcharts this week, however I have run into a few problems, mainly the best way to represent functions in flowchart format. The pseudocode below was straightforward to come up with and understand:

```
PRIVATE FUNCTION UserAlreadyExists(userToSearch As String)
   Declare users ( ) As UserRecord and set to value of LoadUsers ( )
   TRY
      Declare usernames (Last index of users) As String
      FOR i = 0 TO Last index of users
         usernames (i) = users (i).username
      NEXT
      IF ItemExistsInArray(usernames, userToSearch) THEN
         RETURN True
      ELSE
         RETURN False
      END IF
   CATCH
      RETURN False
   END TRY
END FUNCTION
```

The flowchart counterpart was howver harder to design. This is due to the fact that the function actually terminates after returning a value. This is easy to represent through pseudocode, but in flowcharts this is difficult to make sure lines don't cross and the flowchart still makes sense. Here is the solution that I came up with:



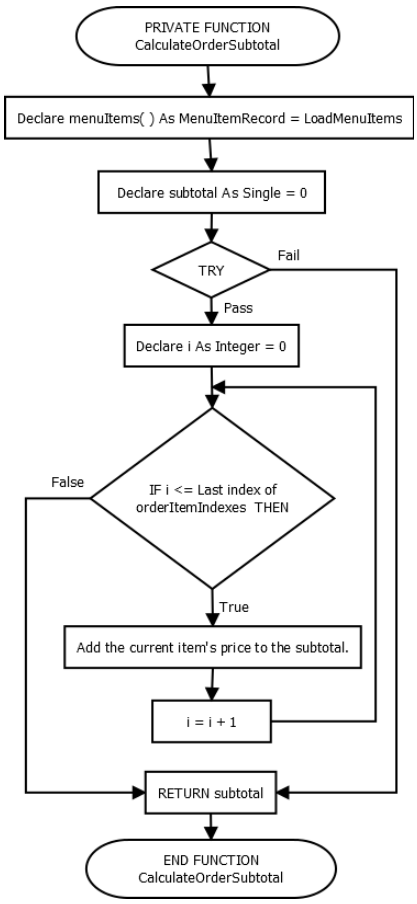
Such flowcharts require extra planning to create and thus take much more time. Again, this is slowing me down, and although I am trying to work through this part of the project as quickly as possible, it is very important to get it accurate, considering it describes the exact logic of the program.

Next week, I will keep working on my pseudocode and flowcharts.

Term 2 Week 7 Update

Despite sorting out a few problems with flowcharts in the presvious couple of weeks, I managed to come across some more (that's always the way, isn't it). My main problem that I had to overcome this week was how to represent try/catch statements in flowchart form. In pseudocode, it is obvious that each line within the 'try' part of the statement tries to execute, and if any problems occur on any one of these statements, the 'catch' part of the statement executes. This is however unpheasible to represent in flowchart form, as there would have to be 'fail' arrows coming off each statement in the 'try' part of the statement. Instead I opted for a more simple approach, representing the try/catch pair as a simple decision statement, whereby failing executes the 'catch' part of the statement. Although not technically ideal, this representation dramatically improves readability whilst still indicating that a check for errors is still taking place. Here is my solution, in pseudocode and flowchart form:

```
PRIVATE FUNCTION CalculateOrderSubtotal (orderItemIndexes( ) As Integer)
   Declare menuItems( ) As MenuItemRecord = LoadMenuItems( )
   Declare subtotal As Single = 0
   TRY
      FOR i = 0 TO Last index of orderItemIndexes
         Add the current item's price to the subtotal.
      NEXT
   CATCH
   END TRY
   RETURN subtotal
END FUNCTION
```



Again, next week I will continue to work on flowcharts and pseudocode.

Term 2 Week 8 Update
We began the theory of testing and evaluating software solutions this week. We learned about the importance of alpha and beta testing, black box and white box testing, as well as the different levels of testing: module level, program level and system level.

We also discussed the relevance of each of these concepts to our major projects, in preparation for the testing phase. I have conducted plenty of informal white box, black box and alpha testing during the implementation phase of my project, but this needs to be documented in a test data table to ensure that my solution performs correctly in all situations. This test data table will include both module and program level tests, however system level tests are beyond the scope of the project.

This is due to the fact that my solution is only really a prototype for a café management system, and does not run on mobile devices like it would in a proper system. System tests involving hardware, software, data, personnel and procedures are therefore difficult to preform.

Beta tests are also not necessary in the scope of the major project, as I am not planning to release the software solution to real-world clients.

I will be working on a test data table along with my algorithm flowcharts, as I recently finished all of my pseudcode. I am hoping to have these elements of the project done before coming back to school next term.

Term 2 Week 9 Update
I finished the overview for the testing section of my project this week, addressing the various aspects of testing and maintaining that are relevant to my project:

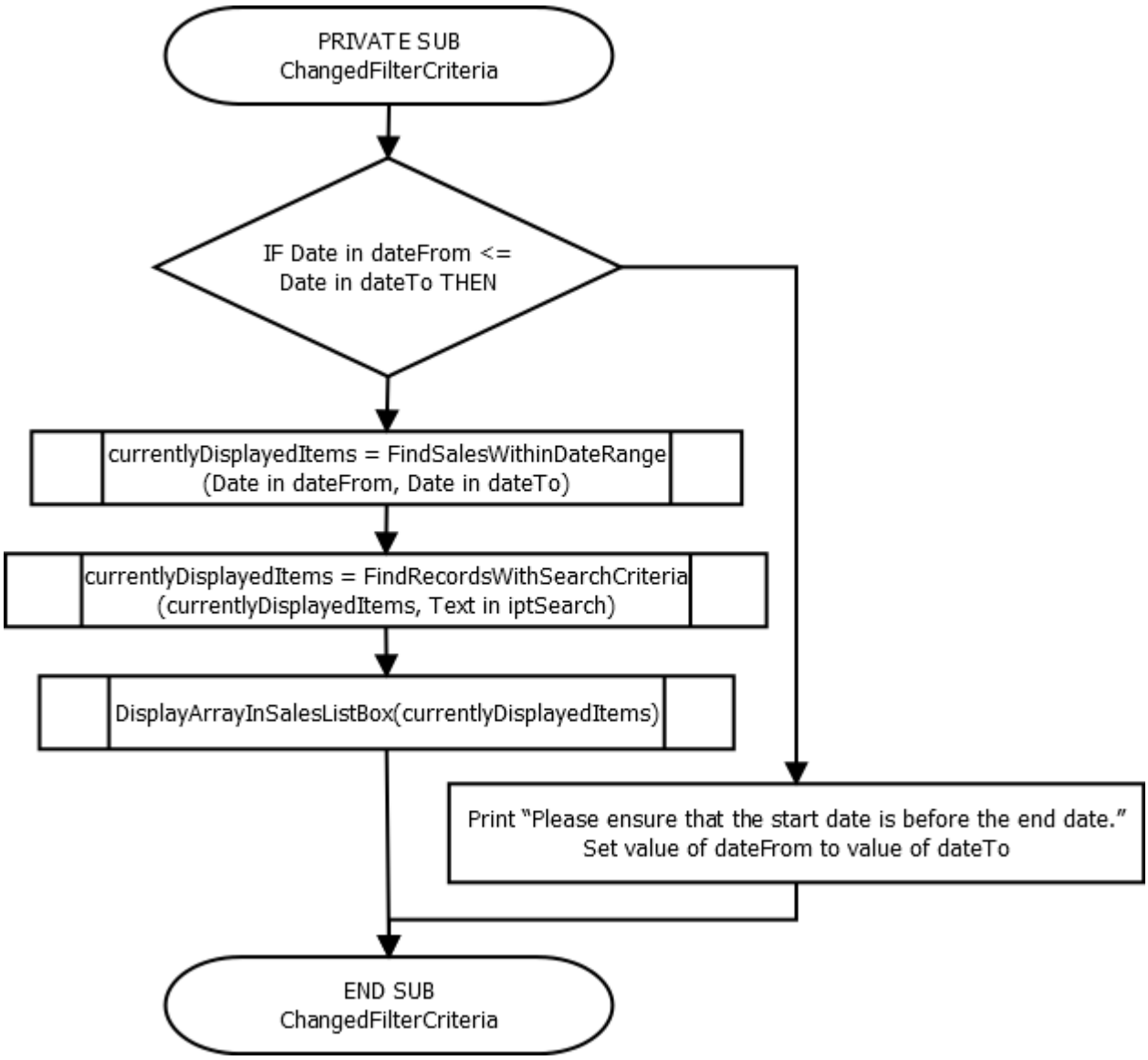
> Testing is essential to the software development process, as it enables the production of a high-quality final solution that meets all initial requirements and functions properly in a variety of environments.
>
> There are various types of testing that are used to achieve different goals. Black box testing is the testing of a software product whereby the user is aware of the inputs and expected outputs, but is unsure of the explicit logic behind the program. This is useful for determining a user's reaction to the software package. White box testing is the opposite, and is usually carried out by the developer with the intent of removing logic, syntax and runtime errors; The exact processing that occurs is known to the user in this case.
>
> Similarities can be drawn between white box testing and alpha testing, a method of testing conducted by the developer prior to the release of a product to make sure it performs as intended. Beta testing is similar to black box testing, as it is a way of gauging user experience and interaction through the release to the public.
>
> Both alpha/white box and beta/black box testing are essential to the testing phase of software development, as together they provide a well-rounded view of the overall quality of the solution. However, not all of these methods of testing are applicable to the scope of my project. Alpha and white box testing are completed both informally during the implementation phase, as well as formally through a test data table. Beta testing is not within the scope of my project, as I am not planning to release the software solution to real-world clients. It is however necessary for me to gauge the effectiveness of elements such as the user interface, help sections andinclusivity of the software solution using relevant black box testing techniques. Whilst these techniques were inflormal, often just observations of a user's experience, they were extremely valuable in the testing process, providing feedback on the elements of my project that needed improving.
>
> In addition to the different types of testing, there are also multiple levels of testing that need to be completed. There are three main levels, namely module level testing, program level, and system level testing. System level testing involves all aspects of the final system, including hardware, software, data, personnel and procedures. It is therefore obsolete in the scope of my project, as my solution is only really a prototype for a café management system, and does not run on mobile devices like it would in a proper system. System tests involving hardware, software, data, personnel and procedures are therefore difficult to perform.

Module level testing is however highly relevant, as it is the testing of individual modules of code to ensure it is free of syntax, logic and runtime errors. Program level testing is also highly relevant, as it makes sure each module passes parameters to other modules correctly and the whole program works as intended. These types of tests are recorded formally using a test data table, as seen on the next page.

I have also been working steadily on my algorithm flowcharts this week. The main problem that I ran into is in regards to the use of private class variables in subroutines, and how this should be represented in flowchart form. In my 'ViewSales' form, for example, I declare a private variable called currentlyDisplayedItems( ), which I use in various other subroutines in the class. This is easily seen in the pseudocode below:

```
Declare currentlyDisplayedItems( ) As OrderRecord = LoadOrders( )

PRIVATE SUB ChangedFilterCriteria( ) When iptSearch is changed, dateFrom is changed, dateTo is changed
    IF Date in dateFrom <= Date in dateTo THEN
        currentlyDisplayedItems = FindSalesWithinDateRange(Date in dateFrom, Date in dateTo)
        currentlyDisplayedItems = FindRecordsWithSearchCriteria(currentlyDisplayedItems, Text in iptSearch)
        DisplayArrayInSalesListBox(currentlyDisplayedItems)
    ELSE
        Print "Please ensure that the start date is before the end date."
        Set value of dateFrom to value of dateTo
    END IF
END SUB
```

Creating a flowchart equivalent is however interesting, as it is often unclear as to where the variable came from. Here is the flowchart for the ChangedFilterCriteria subroutine:



Since the array currentlyDisplayedItems( ) is declared outside of the cubroutine, it is incorrect to declare it in this flowchart, but it is also incorrect not to show evidence of the declaration at all.

The solution I decided upon is to do a 'class flowchart', showing all variable, subroutine and function declarations, to give each subroutine a context to which it can be compared.

I will continue to work on my flowcharts over the holidays and hope to be well into the testing phase by the end of Week 1 Term 3.

Term 3 Week 1 Update
I finished the last few flowcharts this week, as well as completed my discussions of social and ethical issues and maintenance. I still have to complete my user manual and test data table, however these are entirely achievable before the due date next week.

I am also currently revising my Gantt chart, and am quite impressed with how well I have managed to stick with my original plans. It was never expected that I would be able to match the initial Gantt chart requirements entirely, due to the many interruptions of school.

The main thing I have learned this week is the importance of creating a high quality, ergonomic and inclusive software product. A failure to do so leads to a failure of the software product itself, as it will not be accepted by its user base. Although I do not plan on selling my project, these concepts are essential to think about, especially if I decide to release a software solution in the near future.

Term 3 Week 2 Update
The main thing I learned this week is just how long it takes to put everything into an ordered format. I had plenty of odds and ends to tidy up, as well as a user menual to complete.

One of my other struggles this week was to switch from producing technical documents like those in my portfolio to user-based documents like the user manual. It is important to use language that the user can understand whilst explaining instructions, and it is necessary to explain everything in detail. This can get tedious, as I often think "Surely it is obvious what the 'Back' button does", but it is important to mention it in the user manual so that the user has an understanding of where they are in the software solution navigation-wise.

I completed my final Gantt chart and social and ethical issues discussions this week, as well as tidied up some earlier extended responses to make sure all aspects of the questions were addressed.

Overall the project has been a pleasure to work on over the last few months, and I am rather happy with how it turned out. Of course, given more time, I would have added in more features, including:

- Additional metadata for special requests from customers.
    - For example, a customer might order a burger with no cheese – it is important that this extra message is passed through to the kitchen.
    - This could be implemented by adding an extra field to the orderRecord structure for keeping track of any special requests. These special requests would be printed on the receipt that is passed through to the kitchen.
- Include the ability to complete multiple orders for a table and automatically merge the data from the orders to calculate a grand total for the table.
    - This would be useful in the case that a customer wants to order drinks, then meals, then desserts. Instead of putting in three separate orders for the same table that need to be marked as paid, adding items to the original order would make it easier to complete the transaction process.
    - This is a major change that would require a major redesign of the system for taking and paying for orders, but could be completed by doing the following:
        - When a table is selected during the order taking process, the system checks if there is an existing unpaid order from today for that table.
        - If there isn't, then the order process continues as normal.
        - If there is, then when the order process continues as normal until the order is confirmed (in order to reduce the learning curve for the waiters). Wen the order is confirmed, a receipt is generated as usual (displaying only the items ordered this time around, i.e. not the items from the previous order that has already been served), but instead of generating a new orderRecord and writing this data to file, the order subtotal for the second order is added to the order subtotal of the first, and the updated orderRecord is saved to fie. This way when it is time for the customers to pay for their order, there will be only one order to be marked as paid, and the subtotal of multiple orders will not need to be added up.
- Adding the ability to keep track of currently used tables, in order to avoid confusion.
    - This could be achieved by implementing an additional form for viewing and updating the state of tables.
    - Each table would have one of two states: 'Empty' or 'Taken' (represented by an array of Boolean variables stored in a text file.)
    - As customers arrive, the waiters can easily see which tables are taken from this new form and seat the customers at an empty table. They then mark the table as being 'Taken'.
    - Similarly, as customers leave, waiters mark the table as 'Empty', ready for the next customer. If this change was implemented alongside the previous one (Merging data from multiple orders), then tables could automatically be marked as 'Empty' once the orderRecord for that table is marked as 'Paid', increasing efficiency.

The software industry is always changing and it is vital to consistently update and maintain a software package in order to profit from it. Such is the nature of software development.