

GO ADVANCED ASSIGNMENT - WRITEUP

1. Description of Application Features

The program I have chosen to develop is a pizza ordering system. The program is made on the idea of a first-in-first-out (FIFO) concept whereby an order is processed on a first-come-first-serve queue basis i.e. the first order that comes into the system must be processed before the next one can be processed. The main menu consists of the following features:

```
=====
*   WELCOME TO AWESOME PIZZA (MAIN MENU)   *
=====
1. View Menu & Order a Pizza
2. Edit an Order
3. Remove Order in Queue
4. Search an Order in Queue
5. View All Orders in Queue
6. View Pizza Sales of the Day
7. Manage Pizza (Admin Menu)
8. Exit Program

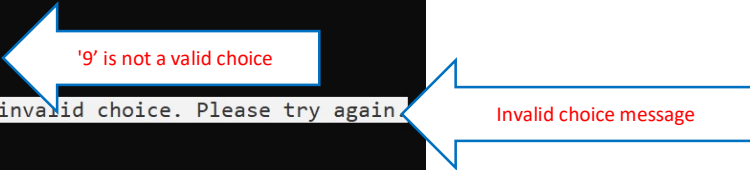
Enter your choice:
```

Basically the main menu allows user to input the selected menu choice by typing a number associated with the menu options. If a number outside of the range is entered or any other input is made, an error will be prompted and the main menu will be displayed again. This is handled by a for-loop in the code, thus the program can continue to run.

```
7. Manage Pizza (Admin Menu)
8. Exit Program

Enter your choice: 9
>> You have input an invalid choice. Please try again.

=====
*   WELCOME TO AWESOME PIZZA (MAIN MENU)   *
=====
1. View Menu & Order a Pizza
2. Edit an Order
3. Remove Order in Queue
4. Search an Order in Queue
5. View All Orders in Queue
6. View Pizza Sales of the Day
7. Manage Pizza (Admin Menu)
8. Exit Program
```



1. View Menu & Order a Pizza

In this feature, a standard pizza menu will be displayed and user will be prompted to enter the Pizza No to start making an order. I have chosen to incorporate 5 standard pizza and the standard price of each pizza is set at \$10.90 in a global variable in the package main to facilitate the start of the program. The price can be amended in the admin menu which will be explained later. If user do not wish to make an order, he/she can hit 'Enter' on the keyboard to go back to the main menu.

```

8. Exit Program

Enter your choice: 1

-----
# VIEW MENU & ORDER A PIZZA #
-----
(1) - Hawaiian Pizza - $10.90
(2) - Cheese Pizza - $10.90
(3) - Pepperoni Pizza - $10.90
(4) - Chicken Pizza - $10.90
(5) - Vegan Pizza - $10.90

Enter Pizza No
(Press 'Enter' to save order & go back to main menu)?:

```

First user will be prompted to enter the Pizza No (alternatively user can also choose to exit the feature by pressing the 'Enter' key on the keyboard. When a valid Pizza No is entered, user will be prompted to enter the Quantity next. The maximum quantity that user can entered is set as a global variable in the main package. Currently it is set to 5. Next, when a valid Quantity is entered, user will be prompted if he/she wants to continue to add more pizza. If 'Y' or 'y' is entered, the program continues in a loop prompting user to enter Pizza No and Quantity until finally 'Enter' (or any key) is pressed on the keyboard. The program then exits the loop and saves any pizza order that user has keyed in. A receipt of the order will be printed when order is added successfully.

```

Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)?: 1

Enter Order Quantity (max 5): 2

-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu): y

-----
Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)?: 3

Enter Order Quantity (max 5): 1

-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu):

-----
* RECEIPT *

Order No: 1001

2 x Hawaiian Pizza      $21.80
1 x Pepperoni Pizza     $10.90
-----
TOTAL PAYMENT           $32.70
-----

```

2. Edit an Order

User will be prompted to enter a valid Order No that is currently in the queue to edit an order. Thus it is important for users to know what is the Order Nos made previously. A way to find out the Order No in the system is by using the menu option [5. View All Orders in

Queue] which will be explained later. When a valid order is found based on the Order No, the order details will be displayed to the user, followed by the pizza menu which will also be printed. A prompt which asked user to enter the Pizza No, Quantity will follow as per seen in the previous feature to make an order. Here, the program is allowing user to remake his/her pizza order which will be updated upon completion with the same Order No. At the end, a new receipt will be printed and user will be informed to make difference in payment accordingly. Due to time constraints, I have not incorporated the difference in amount and will assume that this will be done by the customer and pizza shop owner in-person. But this can be an enhancement to the program in future.

```
Enter your choice: 2
-----
# EDIT AN ORDER #
-----
Enter order no: 1001
You have selected the following order to edit:
Order No: 1001
2 x Hawaiian Pizza      $21.80
1 x Pepperoni Pizza     $10.90
-----
TOTAL PAYMENT           $32.70
-----

(1) - Hawaiian Pizza - $10.90
(2) - Cheese Pizza - $10.90
(3) - Pepperoni Pizza - $10.90
(4) - Chicken Pizza - $10.90
(5) - Vegan Pizza - $10.90

Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)? 1
Enter Order Quantity (max 5): 1
-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu): y
-----
```

Current order detail is printed

Pizza menu printed for easy reference

New pizza order can be made

```
Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)?: 4

Enter Order Quantity (max 5): 1

-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu): y
-----

Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)?: 5

Enter Order Quantity (max 5): 1

-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu): y
-----

Enter Pizza No
(Or press 'Enter' to save order & go back to main menu)?: 2

Enter Order Quantity (max 5): 1

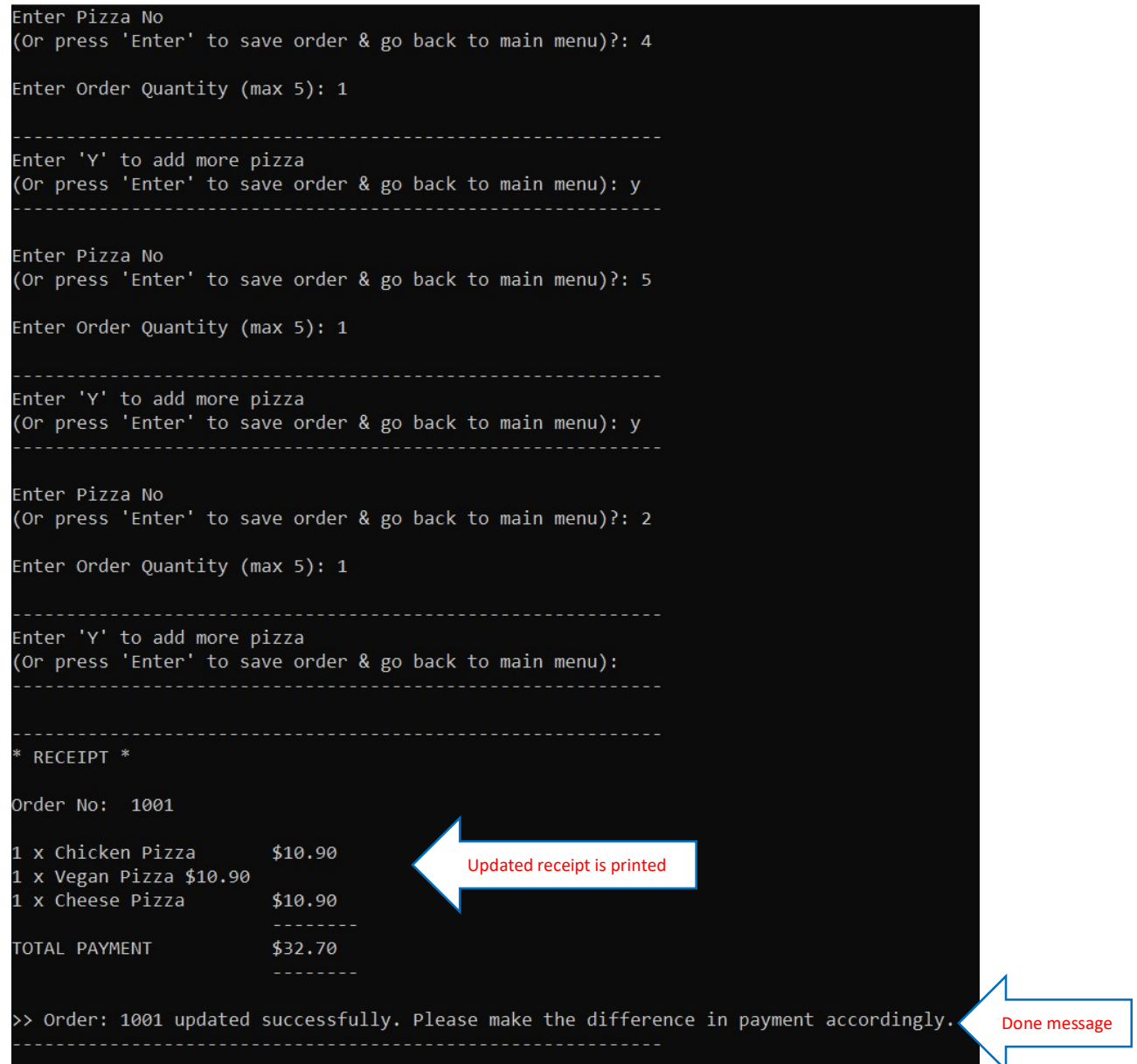
-----
Enter 'Y' to add more pizza
(Or press 'Enter' to save order & go back to main menu):
-----

* RECEIPT *

Order No: 1001

1 x Chicken Pizza      $10.90
1 x Vegan Pizza $10.90
1 x Cheese Pizza      $10.90
-----
TOTAL PAYMENT          $32.70
-----

>> Order: 1001 updated successfully. Please make the difference in payment accordingly.
```



3. Remove Order in Queue

This feature will remove the first order that was made in the queue once it is processed thus there is no need to enter the Order No since it is a FIFO concept. It will prompt the user to confirm removed request and when 'Y' or 'y' is entered, the order will be removed from the queue. As an enhancement, I have added the deleted orders of the pizza to the daily sales so that a report can be generated via menu option [6. *View Pizza Sales of the Day*]. Upon deletion, the order details that was removed will be displayed followed by a done message.

```
Enter your choice: 3
-----
# REMOVE AN ORDER (DEQUEUE) #
-----
Remove completed order from queue?
(Enter 'Y' to confirm): y
Confirm deletion

Order No: 1001
1 x Hawaiian Pizza    $10.90
1 x Chicken Pizza    $10.90
1 x Vegan Pizza $10.90
-----
TOTAL PAYMENT        $32.70
-----

>> Order: 1001 removed from queue and added to pizza sales.
-----
Done message
```

4. Search an Order in Queue

This will search for an order by prompting user to enter a valid Order No. If order is found, the order details will be displayed.

```
=====
*   WELCOME TO AWESOME PIZZA (MAIN MENU)   *
=====
1. View Menu & Order a Pizza
2. Edit an Order
3. Remove Order in Queue
4. Search an Order in Queue
5. View All Orders in Queue
6. View Pizza Sales of the Day
7. Manage Pizza (Admin Menu)
8. Exit Program

Enter your choice: 4
-----
# SEARCH AN ORDER #
-----
Enter order no: 1001
Enter order no

Order No: 1001
1 x Hawaiian Pizza    $10.90
1 x Chicken Pizza    $10.90
1 x Vegan Pizza $10.90
-----
TOTAL PAYMENT        $32.70
-----
Order found and order details displayed
```

5. View All Orders in Queue

This will display all the current orders that are still in the queue waiting to be processed.


```
Enter your choice: 5

-----
# VIEW ALL ORDERS IN QUEUE #
-----
Order No: 1001
1 x Hawaiian Pizza @ $10.90
1 x Chicken Pizza @ $10.90
1 x Vegan Pizza @ $10.90
Total Amount = $32.70

Order No: 1002
1 x Pepperoni Pizza @ $10.90
1 x Vegan Pizza @ $10.90
Total Amount = $21.80

Order No: 1003
2 x Chicken Pizza @ $21.80
Total Amount = $21.80

Order No: 1004
1 x Cheese Pizza @ $10.90
3 x Hawaiian Pizza @ $32.70
Total Amount = $43.60

-----
```

All orders in the queue will be displayed

6. View Pizza Sales of the Day

This feature will display all the pizza sales of the day. Note that if the orders are still in the queue, there will not be any data being displayed here. Only orders that are removed from queue in [3. Remove Order in Queue], will be added to the total sales. I am assuming here that if orders do not get processed and handled to the customer, it does not constitute as a sales because in real life, the customers will not be happy if they did not receive their orders and will definitely ask for refund. The system does not handle refund as of now, so we will assume that all orders will be processed in order and get dequeued one by one, adding to the total sales ultimately. When orders are added to the total sales, they will be sorted by the pizza names in alphabetical order. It will display the total quantities made on the selected pizza for all orders and the total cost (quantity * price per pizza). Total sales is then displayed by adding all the total cost of every pizza.

```
Enter your choice: 6

-----
# VIEW PIZZA SALES OF THE DAY #
-----
PIZZA NAME          ORDER QUANTITY  TOTAL COST
-----
Cheese Pizza         1               $10.90
Chicken Pizza        3               $32.70
Hawaiian Pizza       4               $43.60
Pepperoni Pizza      1               $10.90
Vegan Pizza          2               $21.80

-----
TOTAL SALES OF THE DAY:                $119.90
-----
```

Report will be displayed by pizza name, order quantity and total cost (quantity * price per pizza)

Total sales is displayed by adding all the total cost

7. Manage Pizza (Admin Menu)

The admin menu allows user to manage the pizza in the program. The admin menu consists of the following features:

```
Enter your choice: 7
=====
*   MANAGE PIZZA (ADMIN MENU)   *
=====
1. Add New Pizza
2. Edit Pizza
3. Delete Pizza
4. View All Pizza
5. Go Back to Main Menu

Enter your choice:
```

1. Add New Pizza

Allow user to add a new pizza on top of the 5 standard pizza that was already created. User will be prompted to enter the Pizza Name and Price.

```
Enter your choice: 1
-----
# ADD NEW PIZZA #
-----

Enter Pizza Name: BBQ Pizza
Enter Pizza Price: 9.90

Are you sure you want to add BBQ Pizza @ $9.90? (Enter 'Y' to confirm): y
>> BBQ Pizza @ $9.90 added successfully.
-----
```

2. Edit Pizza

Allow user to edit a pizza by keying in the Pizza No. If a valid pizza is found, the selection will be displayed and user will be prompted to enter the new Pizza Name and Price. **Note: A pizza cannot be edited if an order has been made on the pizza!**

```
Enter your choice: 2
-----
# EDIT PIZZA #
-----

Enter Pizza No: 6

You have selected to update: BBQ Pizza @ $9.90

Enter New Pizza Name: Spicy BBQ Pizza
Enter New Pizza Price: 11.90

Are you sure you want to update to Spicy BBQ Pizza @ $11.90? (Enter 'Y' to confirm): y
>> Spicy BBQ Pizza @ $11.90 updated successfully.
-----
```

```
Enter your choice: 2
-----
# EDIT PIZZA #
-----

Enter Pizza No: 1

>> Orders have been made on the selected pizza. Cannot update.
-----
```

3. Delete Pizza

Allow user to delete a pizza by keying in the Pizza No. If a valid pizza is found, the selection will be displayed and user will be prompted to confirm delete. **Note: A pizza cannot be deleted if an order has been made on the pizza!**

```
Enter your choice: 3
-----
# DELETE PIZZA #
-----

Enter Pizza No: 6

Are you sure you want to delete Spicy BBQ Pizza @ $11.90? (Enter 'Y' to confirm): y

>> Spicy BBQ Pizza @ $11.90 deleted successfully.
-----
```

```
Enter your choice: 3
-----
# DELETE PIZZA #
-----

Enter Pizza No: 1

>> Orders have been made on the selected pizza. Cannot delete.
-----
```

4. View All Pizza

Display all the pizza in the system.

```
Enter your choice: 4
-----
# VIEW ALL PIZZA #
-----

(1) - Hawaiian Pizza - $10.90
(2) - Cheese Pizza - $10.90
(3) - Pepperoni Pizza - $10.90
(4) - Chicken Pizza - $10.90
(5) - Vegan Pizza - $10.90
(6) - Spicy BBQ Pizza - $11.90
-----
```

5. Go Back to Main Menu

Allow user to go back to the main menu.


```

Enter your choice: 5

=====
*   WELCOME TO AWESOME PIZZA (MAIN MENU)   *
=====
1. View Menu & Order a Pizza
2. Edit an Order
3. Remove Order in Queue
4. Search an Order in Queue
5. View All Orders in Queue
6. View Pizza Sales of the Day
7. Manage Pizza (Admin Menu)
8. Exit Program

Enter your choice:

```

8. Exit Program

Allow user to exit the program nicely. User will be prompted that all data will be lost and to confirm his/her exit request.

```

=====
*   WELCOME TO AWESOME PIZZA (MAIN MENU)   *
=====
1. View Menu & Order a Pizza
2. Edit an Order
3. Remove Order in Queue
4. Search an Order in Queue
5. View All Orders in Queue
6. View Pizza Sales of the Day
7. Manage Pizza (Admin Menu)
8. Exit Program

Enter your choice: 8

Are you sure you want to exit the program?
Note: All data will be lost!
(Enter 'Y' to confirm): y
>>>>> Exiting program ..... Goodbye!

```

2. Description of Data Structures Used

■ Queue

As orders for my pizza ordering program is processed on a first-come-first-serve basis, I have selected to use a Queue data structures to manage this. The Queue data structure is appropriate because it has a first-in-first-out (FIFO) concept, which works well in my program because when order is made we can enqueue the order. In real life usually an order that comes in first will be processed first. Thus once the order is fulfilled, we can then dequeue it.

The structure of the Queue is as follows. The item in the Node struct will take in an Order struct which contains the OrderNo (int), OrderSlice ([]OrderItem) and TotalCost (float64). I have included a slice for the OrderItem which is a struct that contains the PizzaNo (int) and OrderQty (int) because in an order, user can make purchase of more than one pizza.

```

type OrderItem struct {
    PizzaNo int
    OrderQty int
}

type Order struct {
    OrderNo int
    OrderSlice []OrderItem
    TotalCost float64
}

type Node struct {
    Item Order
    Next *Node
}

type Queue struct {
    Front *Node
    Back *Node
    Size int
}

```

■ **Pointer-based Linked List**

Next, for the management of the pizza menu, I have selected to use a pointer-based LinkedList data structure. This is an appropriate data structure to use because the operations for add, edit, delete and view pizzas have simple and straight-forward functionalities. The structure of the LinkedList is as follows. The item in the Node struct will take in a Pizza struct which contains the PizzaNo (int), PizzaName (string) and PizzaPrice (float64).

```

type Pizza struct {
    PizzaNo int
    PizzaName string
    PizzaPrice float64
}

type Node struct {
    Item Pizza
    Next *Node
}

// Create a linkedlist for the pizza menu
type LinkedList struct {
    Head *Node
    Size int
}

```

■ **Binary Search Tree**

As an enhancement to the program, I have created a feature to report on the total pizza sales of the day. To do this, I have incorporated a Binary Search Tree data structure because I wanted the pizza names to be ordered in alphabetical order. Thus, by using a Binary Search Tree, I can ensure that the pizza names are inserted in a sorted order. The data structure is as follows. The item in the BinaryNode struct will take in a PizzaSales struct which contains the PizzaNo (int), PizzaName (string), OrderQty (int) and PizzaPrice (float64).

```

type PizzaSales struct {
    PizzaNo    int
    PizzaName  string
    OrderQty   int
    PizzaPrice float64
}

type BinaryNode struct {
    Item  PizzaSales // to store the data item
    Left  *BinaryNode // pointer to point to left node
    Right *BinaryNode // pointer to point to right node
}

type BST struct {
    Root *BinaryNode
}

```

3. Description of Data and Format in Various Files

My program consists of the following files and folder:

- main.go

This file is where the package main and main () function code is located. It will also import the packages that I have created here. Inside this file, I have also declared my global variables especially the ones that I need to access my data structures. In this file, I will also print out the main menu and it will call to respective functions based on the menu options selected.

```

import (
    aBST  "GoAdvancedAssignment/adminBST"
    aOrder "GoAdvancedAssignment/adminOrder"
    aPizza "GoAdvancedAssignment/adminPizza"
    "fmt"
    "sync"
)

```

```

var {
    wg sync.WaitGroup

    mu sync.Mutex

    standardPizza = []string{"Hawaiian Pizza", "Cheese Pizza", "Pepperoni Pizza", "Chicken Pizza", "Vegan Pizza"}
    pizzaNo = len(standardPizza)

    orderNo = 1000

    pizzaList = &aPizza.LinkedList{
        Head: nil,
        Size: 0,
    }

    orderQueue = &aOrder.Queue{
        Front: nil,
        Back: nil,
        Size: 0,
    }

    salesBST = &aBST.BST{
        Root: nil,
    }
}

const (
    standardPrice = 10.90 // standard price of all pizza start at $10.90
    maxOrderQty   = 5     // max order is 5 per selected pizza
)

```

■ admin.go

This file is an extension of the package main. In this file, I will print out the admin menu and it will contain codes to manage the functionality of add, edit, delete and view pizza. It will also generate the PizzaNo for creation of a new pizza.

■ order.go

This file is an extension of the package main. In this file, I will manage the functionality of adding orders (enqueue), editing orders, deleting orders (dequeue) and printing of order details. It will also generate the OrderNo for creation of a new order.

■ validations.go

This file is an extension of the package main. It contains validation checks that are performed on most of the user inputs.

■ adminPizza/adminPizza.go

This is a package created called adminPizza. In this package, I will manage the data structure for LinkedList which will perform add, edit and delete of pizza functionalities. Inside the package, it will also print the pizza menu.

■ adminOrder/adminOrder.go

This is a package created called adminOrder. In this package, I will manage the data structure for Queue which will perform enqueue and dequeue of the orders. I will also use this package to search for a particular order and print all the order details. An update order function is also added to handle edit of an order.

■ adminBST/adminBST.go

This is a package created called adminBST. In this package, I will manage the data structure for the Binary Search Tree which will insert data upon order dequeue to keep track of pizza sales. It will insert data based on the pizza names in sorted order and will do a print using in-order traverse while summing up the total sales amount of all pizzas. Everytime an order is

dequeued, the UpdateNode function in this package will also update the quantity of an existing pizza.

4. Description of Error Handling and Concurrency Mechanism

■ Error Handling

Error handling is managed in most of the functions used in the three packages I have created by incorporating errors.New() and returning the error to the calling program. Errors will be printed for the user if it is not nil.

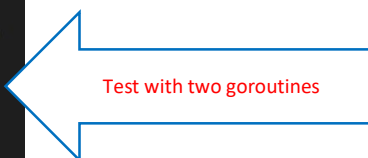
Capturing of panics runtime errors has been added into EditPizza(), DeletePizza() and SearchPizza() of the adminPizza package. This is to handle the runtime error that might occurs when nil cannot be assigned to a currentNode. This is also done for the same reason in SearchOrder(), SearchPizzaInOrder() and UpdateOrder() of the adminOrder package.

■ Concurrency

To handle concurrency, I have decided to use WaitGroup and mutex in func manageUserSelection(choice string) of main.go file. For testing, I created two goroutines to the func by passing in the menu choice of "2" (Edit an Order) and "3" (Remove an Order in Queue) respectively. The output became overlapping with each of the choice's functionalities as shown below.

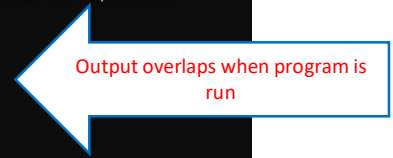
```
wg.Add(2)
//manageUserSelection(choice)
go manageUserSelection("2")
go manageUserSelection("3")

wg.Wait()
```



```
-----
# REMOVE AN ORDER (DEQUEUE) #
-----
# ORDER A PIZZA #
-----
Remove completed order from queue? (Enter 'Y' to confirm): (1) - Hawaiian Pizza - $10.90
(2) - Cheese Pizza - $10.90
(3) - Pepperoni Pizza - $10.90
(4) - Chicken Pizza - $10.90
(5) - Vegetarian Pizza - $10.90

Enter Pizza No (Press 'Enter' to go back to main menu?):
```



Then I added a Mutex Lock() and Unlock() within each of this option in my switch statements.


```

case "2":
    mu.Lock()
    printDividerLine()
    fmt.Println("# ORDER A PIZZA #")
    printDividerLine()

    // Display the pizza menu so that it is easier to make an order
    err := pizzaList.PrintPizzaMenu()

    if err != nil {
        fmt.Println(">> Sorry. No pizza on the menu today.")
    } else {
        fmt.Println()
        /* Go to function in main() to enqueue a pizza order */
        enqueueOrder()
    }
    printDividerLine()
    mu.Unlock()

```

```

case "3":
    mu.Lock()
    printDividerLine()
    fmt.Println("# REMOVE AN ORDER (DEQUEUE) #")
    printDividerLine()

    /* Go to function in main() to dequeue an order that came
    in first in the queue */
    dequeueOrder()

    printDividerLine()
    mu.Unlock()

```

Now, when the program is run, it will display the functionalities of menu choice “2”, then it will display functionalities of menu choice “3” as shown below one after the other.

```

-----
# ORDER A PIZZA #
-----
(1) - Hawaiian Pizza - $10.90
(2) - Cheese Pizza - $10.90
(3) - Pepperoni Pizza - $10.90
(4) - Chicken Pizza - $10.90
(5) - Vegetarian Pizza - $10.90

Enter Pizza No (Press 'Enter' to go back to main menu)? 1

Enter Order Quantity: 1

Enter 'Y' to add more pizza or any key to go back to main menu:

-----
* RECEIPT *
-----
Order No: 1001

1 x Hawaiian Pizza      $10.90
-----
TOTAL PAYMENT           $10.90
-----

-----
# REMOVE AN ORDER (DEQUEUE) #
-----
Remove completed order from queue? (Enter 'Y' to confirm):

```

Because the program will run each menu choice one at a time, I have modify the WaitGroup and Mutex to the following.

```

wg.Add(1)
go manageUserSelection(choice)
//go manageUserSelection("2")
//go manageUserSelection("3")

wg.Wait()

```

And in manageUserSelection(), Mutex Lock() and Unlock() will be placed before and after my switch cases.

```

func manageUserSelection(choice string) {

    defer wg.Done()

    mu.Lock()

    switch choice {
    case "1":

```

5. Instructions on How to Run Application

- For information, my program is currently running Go version - Go1.16.3
- My project files are located in **C:\Projects\Go\src\GoAdvancedAssignment**
- To run the program, run in cmd prompt: **go install** in the respective folders of the 3 packages that I have created as follows.

- Install package in GoAdvancedAssignment/adminPizza

```
C:\Projects\Go\src\GoAdvancedAssignment\adminPizza>go install
```

- Install package in GoAdvancedAssignment/adminOrder

```
C:\Projects\Go\src\GoAdvancedAssignment\adminOrder>go install
```

- Install package in GoAdvancedAssignment/adminBST

```
C:\Projects\Go\src\GoAdvancedAssignment\adminBST>go install
```

- It is important to use the same folder name structure because the import of the packages in main() takes the following format.

```
import (  
    aBST "GoAdvancedAssignment/adminBST"  
    aOrder "GoAdvancedAssignment/adminOrder"  
    aPizza "GoAdvancedAssignment/adminPizza"  
    "fmt"  
    "sync"  
)
```

- When the packages are run successfully, run in cmd prompt: **go run.** (package main has been split into 4 different files - admin.go, main.go, orders.go and validations.go)

```
C:\Projects\Go\src\GoAdvancedAssignment>go run .
```

Remember to input the period "."

- The program will start.