Department of Computer Science
Faculty of Computing

PROJECT

DATA STRUCTURE AND ALGORITHM

SECJ 2013 - 02

CASE STUDY: RESTAURANT MANAGEMENT SYSTEM
(SYSTEM TO ORDERS IN A RESTAURANT)

GROUP: TUPPERWARE

NO.	**NAME**	**MATRIC NUMBER**
1	WAN NUR SOFEA BINTI MOHD HASBULLAH	A22EC0115
2	NADHRAH NURSABRINA BINTI ZULAINI	A22EC0224
3	NUR ALEYSHA QURRATU'AINI BINTI MAT SALLEH	A22EC0241

TABLE OF CONTENT

1.0 INTRODUCTION

1.1 Problem Analysis

In this project, we will develop a restaurant management system to manage the orders in the restaurant. In a restaurant management system, there should be two main users, which are staff of the restaurant and customers. The purpose of developing this system is to address several challenges faced by both staff and customers. One of the primary challenges is the traditional and time consuming order management in a restaurant. Currently, staff must manage the menu manually before allowing the customer to make orders which lead to inefficiencies. The system proposed a solution by implementing stack operations for menu lists. Staff are allowed to add a new menu, delete current menu, and review current changes of the menu, aiming to a more organized and accurate preferences of the restaurant.

Another challenge discovered in the order system in every restaurant is in terms of the system's efficiency. By handling customer orders manually, it often leads to delays and is unorganized. In order to address this issue, the system implements queue operations for order management in the restaurant. Customers can make orders, and the system will manage their orders in a queue systematically. Staff can also view the customer order queue, which allows them to track and confirm the order. This is to enhance the overall service of the restaurant. The system also provides an interface where customers can place orders based on the available manu stack, view their order, and cancel order if needed. This resolves the challenges that often occur in order and customer management, eventually providing satisfaction to the customers.

In conclusion, by implementing stack and queue operations, the proposed restaurant management system can contribute to a more organized flow of ordering management as well as to meet the customers requirements.

1.2 Objective of The Project

- To simplify the process of taking orders in restaurant
- To apply data structure concept such as stack and queue in the system
- To manage menu by adding and deleting menu using stack operation
- To manage customer orders using queue, ensuring a smooth workflow
- To provide a seamless system for both staff and customers

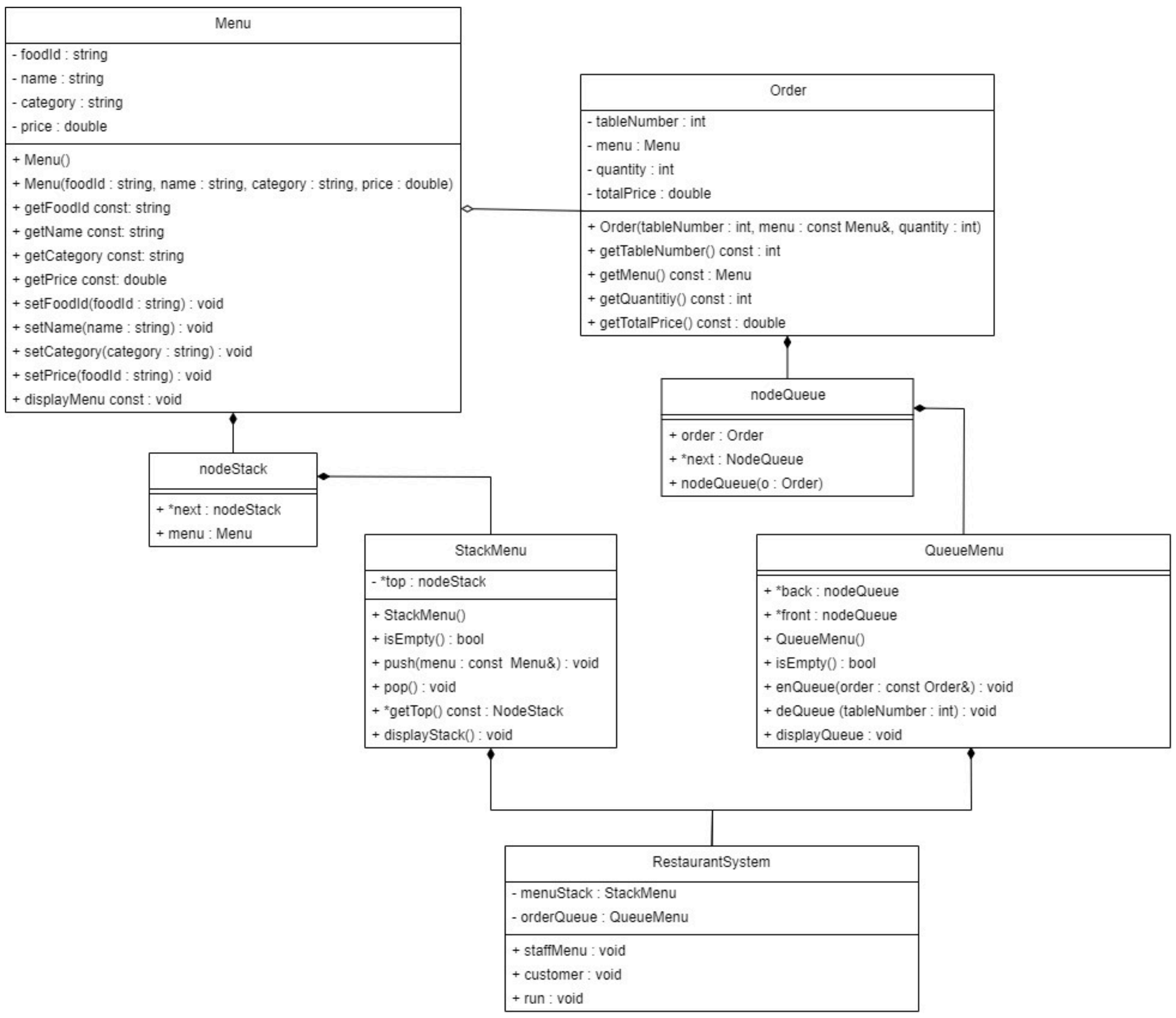
1.3 Synopsis Project

There are two types of data structure used in the Restaurant Management System, which are Stack and Queue. In the system, we have Staff and Customer as the users. Staff is the one to manage the orders queue made by customers. Staff are also able to add or delete new menus using stack operation. Basically, if the user is a staff member, he can add or delete the menu from the list, view the current list of customer orders, and confirm customer orders by using queue operation.

On the other hand, if the user is a customer, they are able to make orders based on the menu list from stack operation. Customers are required to enter their table number, food Id and the quantity of the food they want. Customers are also able to view their list of orders and cancel orders by entering the table number. The order details will be deleted from the order queue.

2.0 SYSTEM DESIGN

2.1 Class Diagram



2.2 Pseudocode

```
1. Start
2. Define a class "Menu"
3. Define a class "nodeStack"
4. Define a class "StackMenu"
5. Define a class "Order"
6. Define a class "NodeQueue"
7. Define a class "QueueMenu"
8. Define a class "RestaurantSystem"
    8.1 staffView()
        8.1.1 Do
            8.1.1.1 Case 1: Add menu
                    8.1.1.1.1 Prompt the user to enter the food ID, food name, category
                    and price
                    8.1.1.1.2 Read all food ID, food name, category and price
                    8.1.1.1.3 End switch
            8.1.1.2 Case 2: Delete menu
                    8.1.1.2.1 Menu deleted
                    8.1.1.2.2 End switch
            8.1.1.3 Case 3: Display recent changes
                    8.1.1.3.1 Display
                    8.1.1.3.2 End switch
            8.1.1.4 Case 4: View customer order
                    8.1.1.4.1 display order queue
                    8.1.1.4.2 End switch
            8.1.1.5 Case 5: confirm customer order
                    8.1.1.5.1 Prompt the user to enter table number
                    8.1.1.5.2 Read user answer
                    8.1.1.5.3 If table number less or equal to 0
                            8.1.1.1.5.3.1 Print "Invalid table number. Please enter a
                            positive integer."
                    8.1.1.5.4 Read user answer
                    8.1.1.5.5 End switch
            8.1.1.6 Case 6: Exit staff menu
                    8.1.6.1 End switch
            8.1.1.7 Default: Print "Invalid option. Please try again."
            8.1.1.8 End switch
            8.1.1.9 Print "Do you want to continue? (Y/N):"
            8.1.1.10 Read user choice
        8.1.2 End while user choice == Y or choice == y
    8.2 customerView()
        8.2.1 Do
            8.2.1.1 Case 1: Make order
                    8.2.1.1.1 Prompt the user to enter the table number
                    8.2.1.1.2 If tablenum < 0 or tablenum>10
                            8.2.1.1.2.1 Print "The table number you entered is invalid"
                    8.2.1.1.3 display Menu
                    8.2.1.1.4 Prompt the user to enter food ID and quantity
                    8.2.1.1.5 Print “"Order enqueued successfully."
                    8.2.1.1.6 End switch
               8.2.1.2 Case 2: Display Queue
                    8.2.1.2.1 Display
                    8.2.1.2.2 End switch
               8.2.1.3 Case 3: Cancel order
                   8.2.1.3.1 Prompt the user to enter table number
                   8.2.1.3.2 Read user answer
                   8.2.1.3.3 dequeue
                   8.2.1.3.4 End switch
               8.2.1.4 Case 4: Exit customer menu
                   8.2.1.4.1 End switch
               8.2.1.5 Default: Print “Invalid option. Please try again.”
               8.2.1.6 End switch
               8.2.1.7 Print “Do you want to continue? (Y/N):”
               8.2.1.8 Read user choice
          8.2.2 End while user choice == Y or choice == y
9.  Do
     9.1 Prompt the user to choose between customer or staff
     9.2 Read user choice
          9.2.1 if user == staff
               9.2.1.1 Prompt the user to enter staff ID
               9.2.1.2 Read user answer
               9.2.1.3 if staffID != restaurant’s staff ID
                    9.2.1.3.1 Print “Invalid staff ID. Redirecting to customer view…”
                    9.2.1.3.2 Customer view
               9.2.1.4 Else
                    9.2.1.4.1 Print “Welcome, staff! Redirecting to staff view..."
                    9.2.1.4.2 Staff view
          9.2.2 Else if user == customer
               9.2.2.1 Print “Welcome, customer! Redirecting to customer view…”
               9.2.2.2 Customer view
     9.3 Print “Do you want to continue in the main menu? (Y/N):”
     9.4 Read user choice
10. End while user choice == Y or choice == y
```

3.0 IMPLEMENTATION OF DATA STRUCTURE

3.1 Implementation of Stack

The system uses a Stack linked list operation to manage the menu list. This operation is basically used by the staff to add or delete menus from the stack. Stack operations required two classes which are nodeStack and StackMenu. The nodeStack class holds menu information by using the instance of the Menu class, menu. The nodeStack* next represents a pointer to point the next node in the stack.

The StackMenu class represents stack operation for the menu. nodeStack* top indicates a pointer to point to the top of the stack. StackMenu initialises top as null value indicating that the stack is empty. isEmpty() is to check whether the stack menu is empty or not. The push function in the system is to add a new menu in the stack menu. The system will prompt the user to enter menu details, if the stack is not empty it will set the new menu to the current top stack. On the other hand, the pop function is to delete a menu from the stack. If the stack is empty, the system will display "The stack is empty". If not, the top stack will be set to the next node, and the top node will be deleted. The stackTop() is to get the current of the stack menu. Lastly, displayStack() is to display all the menus in the stack. It will display according to the current menu changes.

3.2 Implementation of Queue

The system implements the queue linked-list operation to store customer’s orders. The operation includes adding a new order from the customer, viewing the current list of orders, and also cancelling orders.

The addition of a new order is by adding an order into the queue. If the queue is empty, the new order or node will be placed first in the queue and also the back of queue, otherwise, it will be added to the back of the queue. The next for back will take the new node and the back will be the new node. This means, the queue will only increment its storage at the end of the list.

The cancel order operation works by deleting the most front node in the queue. When the most front order is deleted, the second in order will be the new front. Two temporary nodes, temp and prev are needed to make this operation work. The temp node will always move to the next temp when the condition of not null and getTableNum is must not equal to tableNumber is abide. The prev on the other hand, will take the node temp before the temp move to the next node. If one of the condition does not followed, the loop will break. If temp is null, the system will output error messages. If the prev is null however, the front node will be the next temp, but if prev is not null, the next for prev will take the node from next for temp. If the next of prev is null too, hence the back node will be prev. Lastly, the next for temp will be null, and then does the temp can be deleted.

4.0 DEVELOPMENT CODE STEPS/ACTIVITIES

1. **Source code demonstrating the data structure concept employed.**

   Stack Implementation

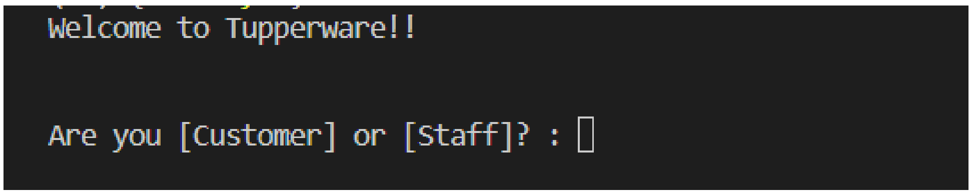
```
class nodeStack {
public:
    Menu menu;
    nodeStack* next;
    nodeStack(Menu m) : menu(m), next(nullptr) {}
};

class StackMenu {
private:
    nodeStack *top;

public:
    StackMenu() {
        top = NULL;
    }

    bool isEmpty() {
        return top == NULL;
    }

    void push(const Menu& menu) {
        nodeStack* newNode = new nodeStack(menu);

        if (!isEmpty())
            newNode->next = top;
        top = newNode;
    }

    void pop() {
        if (isEmpty())
            cout << "The stack is empty." << endl;
        else {
            nodeStack* del = top;

            top = del->next;
            del->next = NULL;
            delete del;
        }
    }

    nodeStack* getTop() const {
        return top;
    }

    void displayStack() {
        if (isEmpty())
            cout << "Sorry, no menu in the stack." << endl;
        else {
            nodeStack* temp = top;

            while (temp) {
                temp->menu.displayMenu();
                temp = temp->next;
            }
        }
    }
};
```

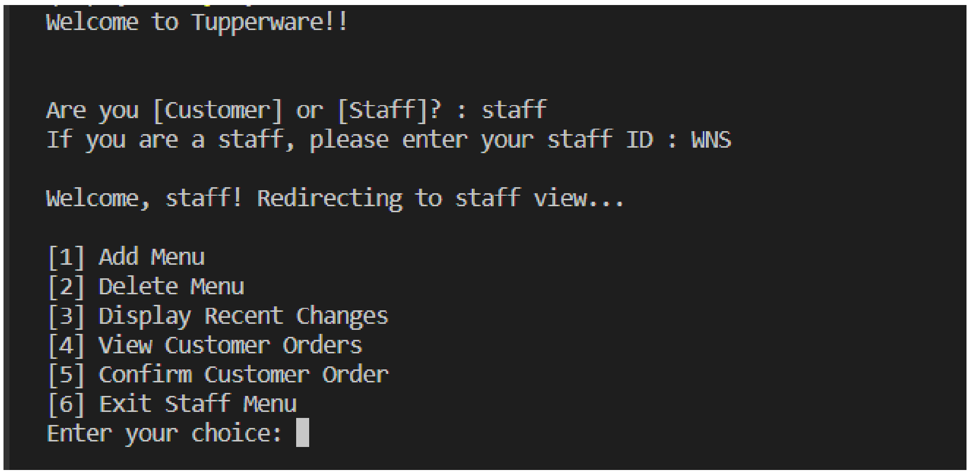
Queue Implementation

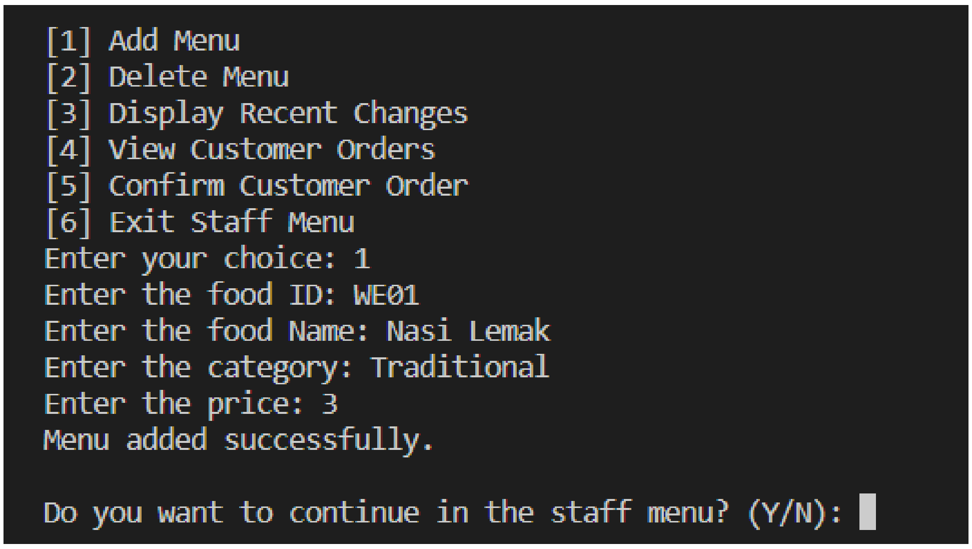
```
class nodeQueue {
public:
    Order order;
    nodeQueue* next;
    nodeQueue(Order o) : order(o), next(nullptr) {}
};

class QueueMenu {
public:
    nodeQueue *back, *front;

    QueueMenu() {
        front = NULL;
        back = NULL;
    }

    bool isEmpty() {
        return ((front == NULL) && (back == NULL));
    }

    void enQueue(const Order& order) {
        nodeQueue* newNode = new nodeQueue(order);

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
    }

    void deQueue(int tableNumber) {
        nodeQueue* temp = front;
        nodeQueue* prev = NULL;

              while (temp != NULL && temp->order.getTableNumber() !=
tableNumber) {
            prev = temp;
            temp = temp->next;
        }

        if (temp == NULL) {
             cout << "No orders found for table number " << tableNumber
<< "." << endl;
        } else {
            if (prev == NULL) {
                front = temp->next;
                if (front == NULL) {
                    back = NULL;
                }
            } else {
                prev->next = temp->next;
                if (prev->next == NULL) {
                    back = prev;
                }
            }

            temp->next = NULL;
            delete temp;
                cout << "Order for table number " << tableNumber << "
dequeued successfully." << endl;
        }
    }

    void displayQueue() {
        if (isEmpty())
            cout << "Sorry, no order in the queue." << endl;
        else {
            cout << setw(10) << "Table" << " | "
                 << left << setw(10) << "Food ID" << " | "
                 << setw(21) << "Name" << " | "
                 << setw(13) << "Category" << " | "
                 << setw(8) << "Quantity" << " | "
                 << setw(6) << "Price" << " | "
                 << setw(10) << "Total Price" << endl;
                                                             cout     <<
"-----------------------------------------------------------------------
---------------------------" << endl;

            nodeQueue* temp = front;
            while (temp) {
                Order order = temp->order;
                Menu menu = order.getMenu();
                cout << setw(10) << order.getTableNumber() << " | "
                     << setw(10) << menu.getFoodId() << " | "
                     << setw(21) << menu.getName() << " | "
                     << setw(13) << menu.getCategory() << " | "
                     << setw(8) << order.getQuantity() << " | "
                             << fixed << setprecision(2) << setw(6) <<
menu.getPrice() << " | "
                             << setw(10) << fixed << setprecision(2) <<
order.getTotalPrice() << endl;
                temp = temp->next;
            }

            cout << endl;
        }
    }
};
```

2. **User manual/guide: provide examples of input and output (if any) for each task.**

Staff view	
Steps	Output and Explanation

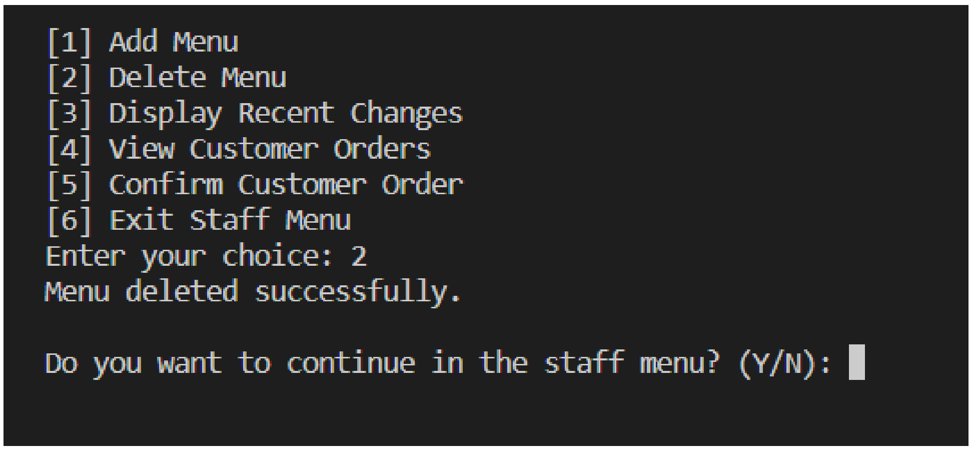
1.	
	```
	Welcome to Tupperware!!

	Are you [Customer] or [Staff]? :
	```
	 The page displays the main menu with 2 options for user to choose either staff or customer.

2.	
	```
	Welcome to Tupperware!!

	Are you [Customer] or [Staff]? : staff
	If you are a staff, please enter your staff ID : WNS

	Welcome, staff! Redirecting to staff view...

	[1] Add Menu
	[2] Delete Menu
	[3] Display Recent Changes
	[4] View Customer Orders
	[5] Confirm Customer Order
	[6] Exit Staff Menu
	Enter your choice:
	```
	 Staff need to enter the staff ID. After that, it will go to staff view. The system will display 6 options as shown above and staff need to enter their choice.

3.	
	```
	[1] Add Menu
	[2] Delete Menu
	[3] Display Recent Changes
	[4] View Customer Orders
	[5] Confirm Customer Order
	[6] Exit Staff Menu
	Enter your choice: 1
	Enter the food ID: WE01
	Enter the food Name: Nasi Lemak
	Enter the category: Traditional
	Enter the price: 3
	Menu added successfully.

	Do you want to continue in the staff menu? (Y/N):
	```
	 <u>Choice [1] Add menu</u> Staff need to enter the food Id, food name, category and price. The menu

will be added to the stack menu.

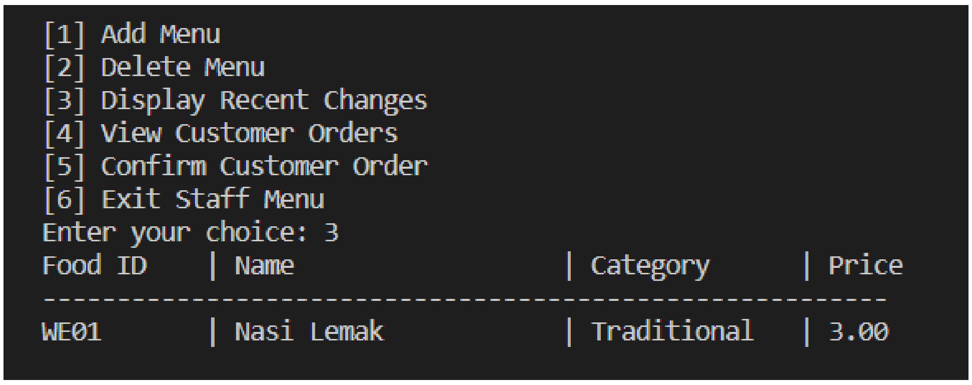
4.



```
[1] Add Menu
[2] Delete Menu
[3] Display Recent Changes
[4] View Customer Orders
[5] Confirm Customer Order
[6] Exit Staff Menu
Enter your choice: 2
Menu deleted successfully.

Do you want to continue in the staff menu? (Y/N):
```

<u>Choice [2] Delete Menu</u>
The menu at the top of the stack will be deleted.

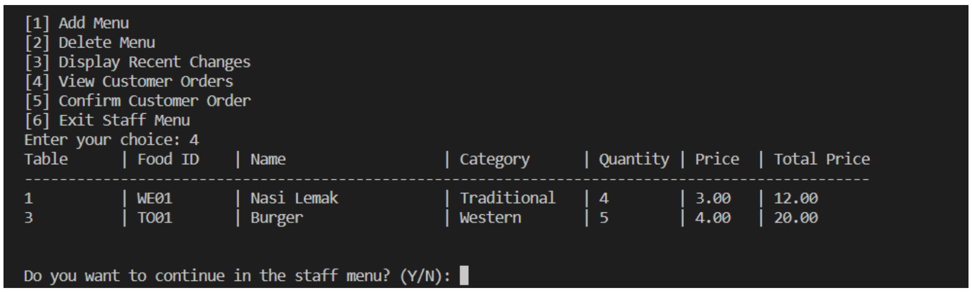
5.



```
[1] Add Menu
[2] Delete Menu
[3] Display Recent Changes
[4] View Customer Orders
[5] Confirm Customer Order
[6] Exit Staff Menu
Enter your choice: 3
Food ID    | Name                 | Category      | Price
----------------------------------------------------------
WE01       | Nasi Lemak           | Traditional   | 3.00
```

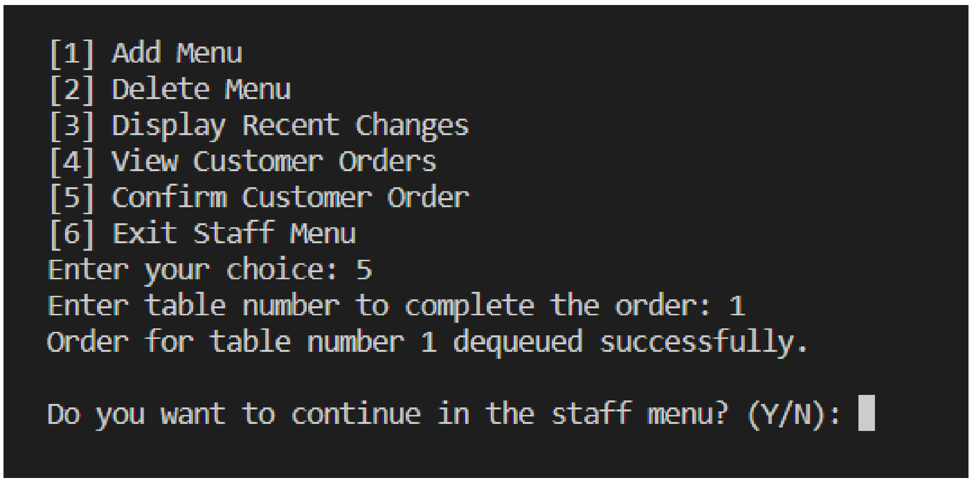
<u>Choice [3] Display Recent Changes</u>
The system will display the menu list in the stack.

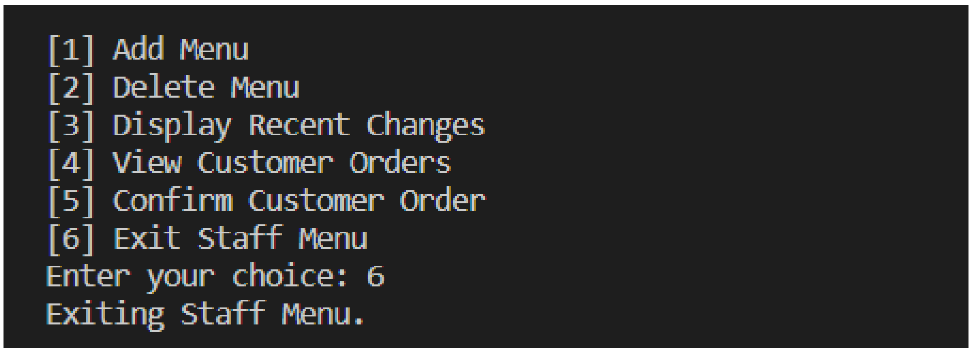
6.



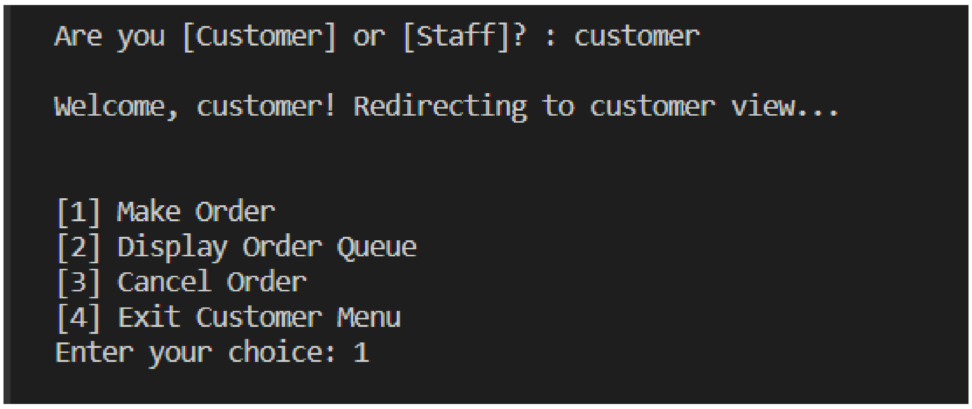
```
[1] Add Menu
[2] Delete Menu
[3] Display Recent Changes
[4] View Customer Orders
[5] Confirm Customer Order
[6] Exit Staff Menu
Enter your choice: 4
Table      | Food ID    | Name                 | Category      | Quantity | Price    | Total Price
-----------------------------------------------------------------------------------------------
1          | WE01       | Nasi Lemak           | Traditional   | 4        | 3.00     | 12.00
3          | TO01       | Burger               | Western       | 5        | 4.00     | 20.00


Do you want to continue in the staff menu? (Y/N):
```

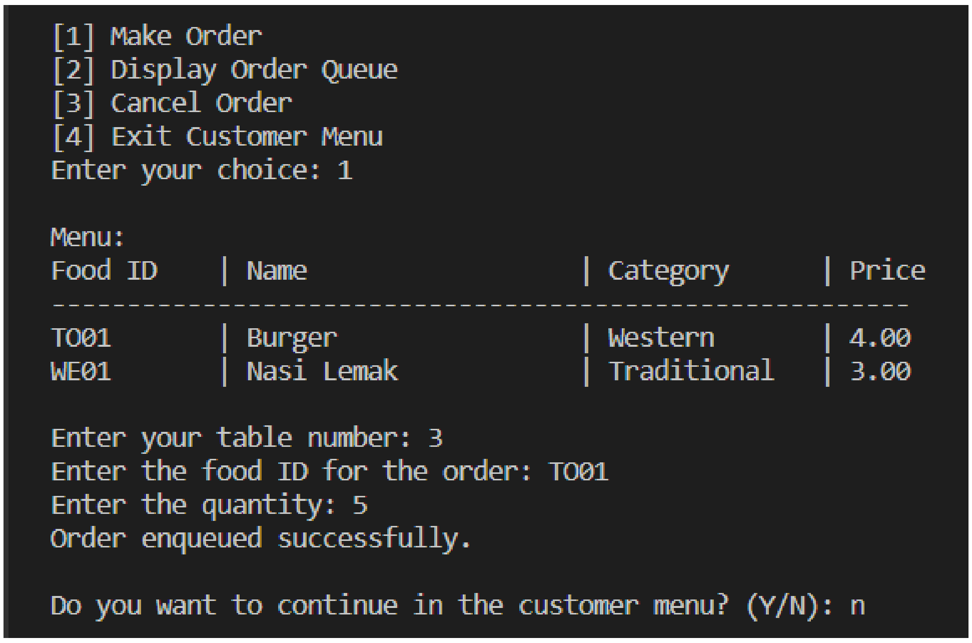
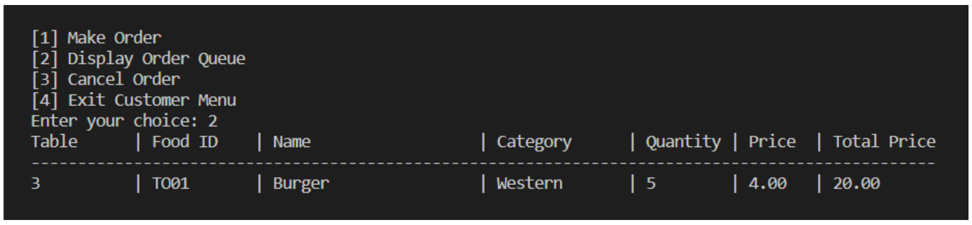
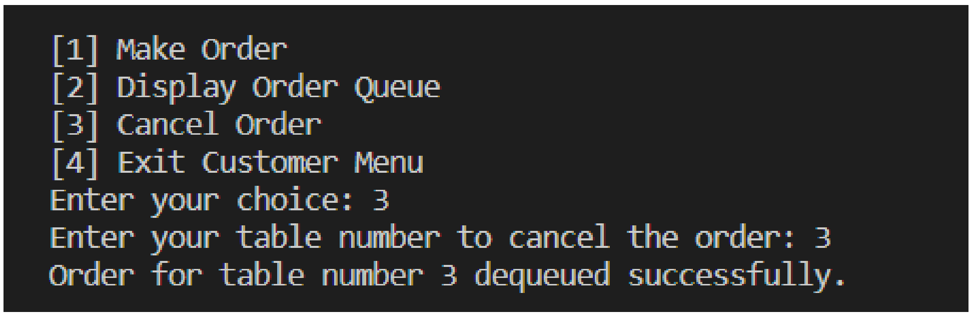
<u>Choice [4] View Customer Order</u>
The system will display the queue of customer orders. If there is no queue order, it will display “Sorry, no order in queue.”

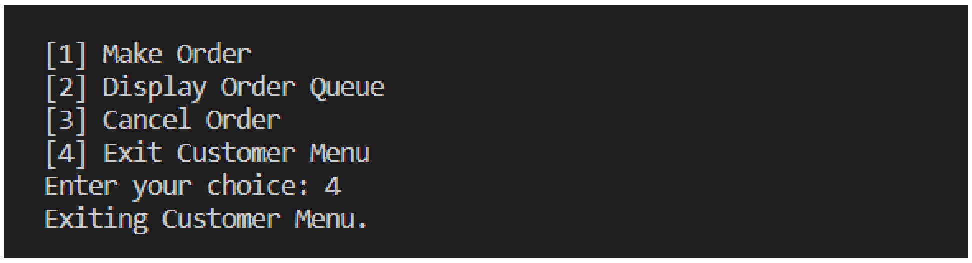
7.	
	`[1] Add Menu`
	`[2] Delete Menu`
	`[3] Display Recent Changes`
	`[4] View Customer Orders`
	`[5] Confirm Customer Order`
	`[6] Exit Staff Menu`
	`Enter your choice: 5`
	`Enter table number to complete the order: 1`
	`Order for table number 1 dequeued successfully.`
	
	`Do you want to continue in the staff menu? (Y/N):`
	<u>Choice [5] Confirm Customer Order</u>
	The staff need to enter the table number they want to remove from the order queue. Then, the customer order will be removed from the queue.

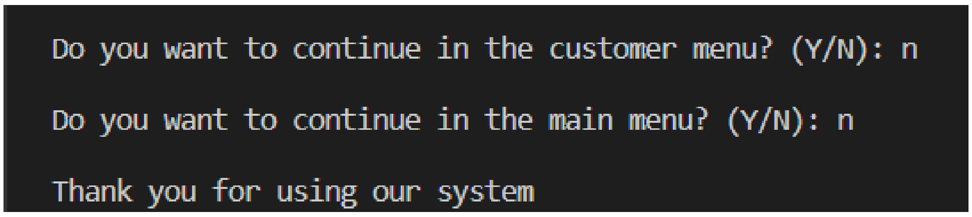
8.	
	`[1] Add Menu`
	`[2] Delete Menu`
	`[3] Display Recent Changes`
	`[4] View Customer Orders`
	`[5] Confirm Customer Order`
	`[6] Exit Staff Menu`
	`Enter your choice: 6`
	`Exiting Staff Menu.`
	<u>Choice [6] Exit Staff Menu</u>
	Exiting staff menu

Customer view	
Steps	Output and Explanation

1.	
	`Are you [Customer] or [Staff]? : customer`
	
	`Welcome, customer! Redirecting to customer view...`
	
	`[1] Make Order`
	`[2] Display Order Queue`
	`[3] Cancel Order`
	`[4] Exit Customer Menu`
	`Enter your choice: 1`
	If the user is a customer, it goes to the customer view. There are 4 options for the customer to choose.

2.	 [1] Make Order [2] Display Order Queue [3] Cancel Order [4] Exit Customer Menu Enter your choice: 1 Menu: Food ID | Name | Category | Price -- TO01 | Burger | Western | 4.00 WE01 | Nasi Lemak | Traditional | 3.00 Enter your table number: 3 Enter the food ID for the order: TO01 Enter the quantity: 5 Order enqueued successfully. Do you want to continue in the customer menu? (Y/N): n **<u>Choice [1] Make Order</u>** The system will display the menu in the stack for customers to refer. Customers need to enter the table number, food Id and the quantity. The order will add into the queue.
3.	 [1] Make Order [2] Display Order Queue [3] Cancel Order [4] Exit Customer Menu Enter your choice: 2 Table | Food ID | Name | Category | Quantity | Price | Total Price -- 3 | TO01 | Burger | Western | 5 | 4.00 | 20.00 **<u>Choice [2] Display Order Queue</u>** The system will display the queue of customer orders with the total price.
4.	 [1] Make Order [2] Display Order Queue [3] Cancel Order [4] Exit Customer Menu Enter your choice: 3 Enter your table number to cancel the order: 3 Order for table number 3 dequeued successfully. **<u>Choice [3] Cancel Order</u>** The order will be removed from the queue based on the table number entered.

5.	
	```
	[1] Make Order
	[2] Display Order Queue
	[3] Cancel Order
	[4] Exit Customer Menu
	Enter your choice: 4
	Exiting Customer Menu.
	```
	Choice [4] Exit Customer Menu
	Exiting the customer view.

6.	
	```
	Do you want to continue in the customer menu? (Y/N): n

	Do you want to continue in the main menu? (Y/N): n

	Thank you for using our system
	```
	Exiting the system.