N

RIPHAH
FACULTY OF
COMPUTING

BSAI-1 (FALL 2024**)**

PROJECT :

Topic :

RESTUAURANT MANAGEMENT SYSTEM

GROUP MEMBERS :	**SAP ID:**
IZHAR ULLAH	**67O11**
HARRIS KHAN	**66427**
M AWAIS KHAN	**66610**

REPORT : RESTUAURANT MANAGEMENT SYSTEM

Table of contents:

1.INTRODUCTION:

This project is a Restaurant Management System implemented in C++. It is designed to simulate the operations of a restaurant, such as managing menus, tables, orders, and staff. The system is built using C++ principles, encapsulating key entities like MenuItem, Table, Order, Staff, and Restaurant into separate classes. Each class is designed to handle specific functionalities, enabling modularity and ease of maintenance.

Features:

1.Menu Management:

•Add and display menu items with prices.

•Allows flexibility in expanding or updating the menu.

2.Staff Management:

•Add staff members with details such as name, role, and salary.

•Manage and display a list of all staff, including roles like waiters, chefs, and managers.

3.Table Management:

•Organize tables into categories (e.g., VIP, Luxury, Normal, Special) and allocate them to different floors.

•Display a structured view of seating arrangements.

4.Order Management:

•Place customer orders by selecting menu items.

•Calculate the total price of orders and maintain a record of all orders.

5.Dynamic Addition:

•Add menu items, staff members, and table categories dynamically at runtime.

2.Summary:

The Restaurant Management System provides a foundational structure for managing a restaurant's operations programmatically. It leverages C++ OOP concepts to organize different components and ensure code reusability and scalability. This system allows restaurant owners to manage their staff, menu, and seating arrangements efficiently while handling customer orders. With proper extensions, this program could be developed further into a fully functional application for real-world use cases.

This project demonstrates:

•Strong understanding of class design and encapsulation.

•Real-world problem modeling using programming constructs.

•A hands-on approach to managing interrelated entities like staff, tables, and orders.

3.Program flow :

- **Menu items, staff members, and tables are added.**

- Orders are taken and displayed with their totals.

- Information about the menu, staff, tables, and orders is displayed.

4.IMPLEMENTION:

The project code is a Restaurant Management System implemented in C++. It uses classes to manage the menu, staff, tables, and customer orders. Key features include:

1.Menu Management: Add and display menu items with prices.

2.Staff Management: Manage staff details like name, role, and salary.

3.Table Management: Organize tables by category and floor.

4.Order Management: Place orders, calculate totals, and display summaries.

It displays the menu, staff, tables, and all orders in a structured manner.

5.Code:

```
#include <iostream>

#include <vector>

#include <string>


using namespace std;

// Class to represent a MenuItem

class MenuItem {

private:

   string name;

   double price;


public:

   // Constructor for MenuItem

   MenuItem(string n, double p) : name(n), price(p) {}


   // Getter for name

   string getName() const {

      return name;

   }


   // Getter for price

   double getPrice() const {

      return price;

   }

};


// Class to represent a Table (with categories and floors)

class Table {

private:

   string category;

   int count;
```

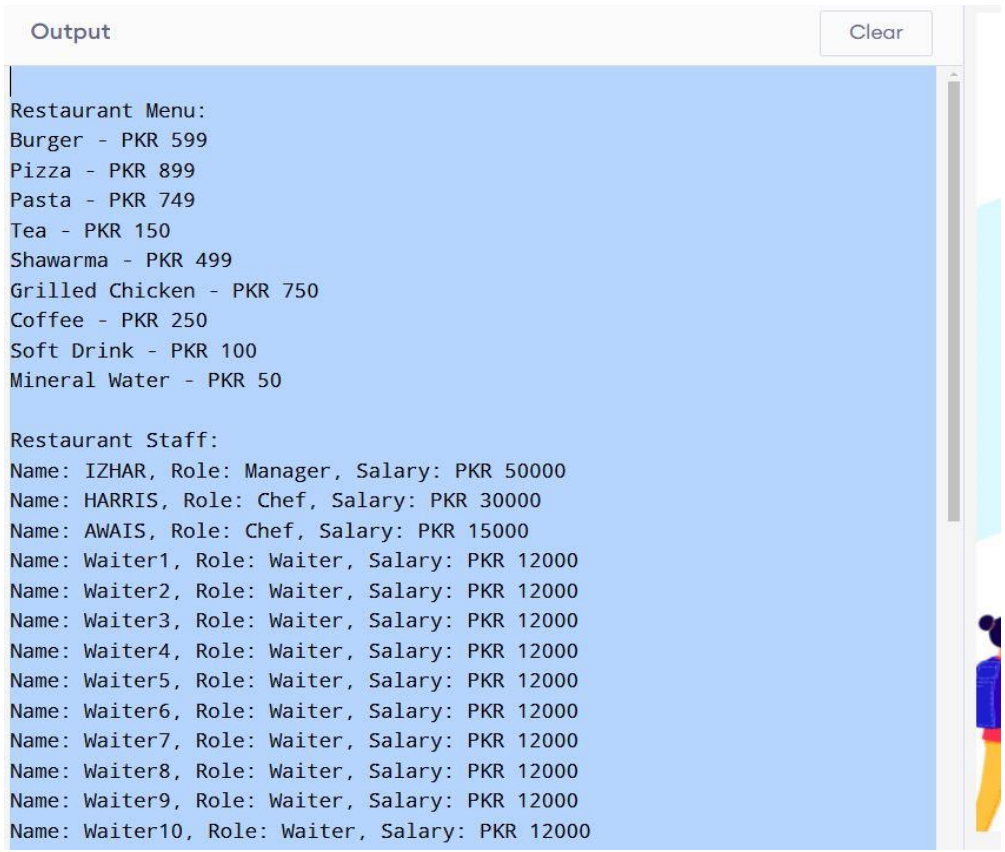
```
    string floor;

public:
    // Constructor for Table with category, count, and floor
    Table(string cat, int c, string f) : category(cat), count(c), floor(f) {}

    // Getter for category
    string getCategory() const {
        return category;
    }

    // Getter for count
    int getCount() const {
        return count;
    }

    // Getter for floor
    string getFloor() const {
        return floor;
    }

    // Method to display table category information
    void displayTableCategory() const {
        cout << "Floor: " << floor << ", Category: " << category << ", Tables: " << count << endl;
    }
};

// Class to represent an Order
class Order {
private:
    vector<MenuItem> items;

public:
    // Method to add an item to the order
    void addItem(const MenuItem& item) {
        items.push_back(item);
    }

    // Method to calculate total price of the order
    double calculateTotal() const {
        double total = 0;
        for (const auto& item : items) {
            total += item.getPrice();
        }
        return total;
    }

    // Method to display the order summary
    void displayOrder() const {
        cout << "Order Summary:" << endl;
        for (const auto& item : items) {
            cout << item.getName() << " - PKR " << item.getPrice() << endl;
```

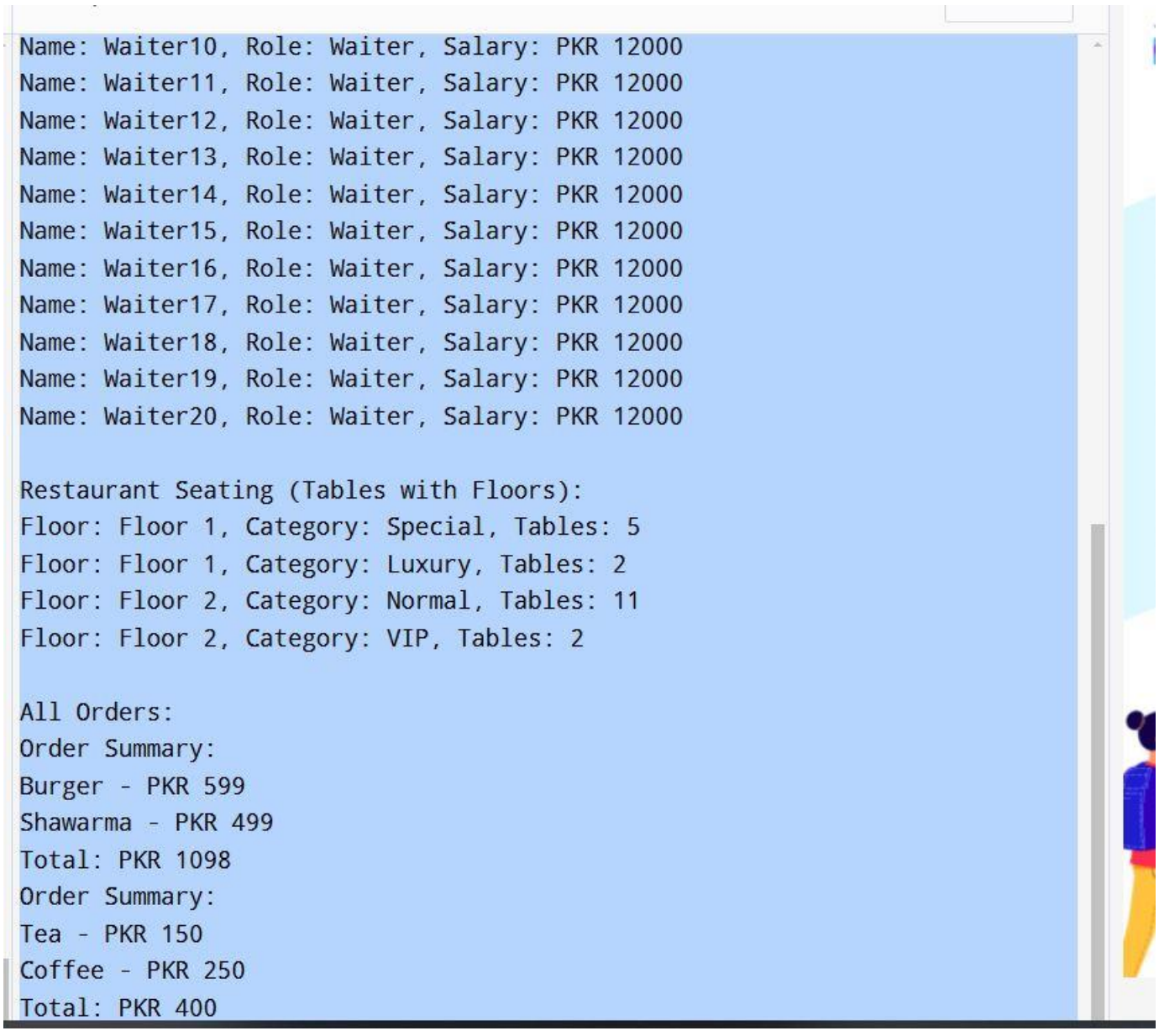
```
        }
        cout << "Total: PKR " << calculateTotal() << endl;
    }
};

// Class to represent a Staff member
class Staff {
private:
    string name;
    string role;
    double salary;

public:
    // Constructor for Staff
    Staff(string n, string r, double s) : name(n), role(r), salary(s) {}

    // Method to display staff member information
    void displayStaff() const {
        cout << "Name: " << name << ", Role: " << role << ", Salary: PKR " << salary << endl;
    }
};

// Class to manage the restaurant
class Restaurant {
private:
    vector<MenuItem> menu;
    vector<Order> orders;
    vector<Staff> staff;
    vector<Table> tables; // List of tables with categories and floors

public:
    // Method to add a menu item to the restaurant
    void addMenuItem(const MenuItem& item) {
        menu.push_back(item);
    }

    // Method to add a staff member to the restaurant
    void addStaff(const Staff& employee) {
        staff.push_back(employee);
    }

    // Method to add a table category to the restaurant
    void addTableCategory(const Table& table) {
        tables.push_back(table);
    }

    // Method to display the menu
    void displayMenu() const {
        cout << "\nRestaurant Menu:" << endl;
        for (const auto& item : menu) {
            cout << item.getName() << " - PKR " << item.getPrice() << endl;
        }
```

```
}

// Method to display the table categories with floor information
void displayTables() const {
    cout << "\nRestaurant Seating (Tables with Floors):" << endl;
    for (const auto& table : tables) {
        table.displayTableCategory();
    }
}

// Method to take an order from a customer
void takeOrder(Order order) {
    orders.push_back(order);
}

// Method to display all orders
void displayOrders() const {
    cout << "\nAll Orders:" << endl;
    for (const auto& order : orders) {
        order.displayOrder();
    }
}

// Method to display all staff members
void displayStaff() const {
    cout << "\nRestaurant Staff:" << endl;
    for (const auto& member : staff) {
        member.displayStaff();
    }
}
};

int main() {
    Restaurant myRestaurant;

    // Adding more menu items with prices in PKR
    myRestaurant.addMenuItem(MenuItem("Burger", 599));
    myRestaurant.addMenuItem(MenuItem("Pizza", 899));
    myRestaurant.addMenuItem(MenuItem("Pasta", 749));
    myRestaurant.addMenuItem(MenuItem("Tea", 150));
    myRestaurant.addMenuItem(MenuItem("Shawarma", 499));
    myRestaurant.addMenuItem(MenuItem("Grilled Chicken", 750));
    myRestaurant.addMenuItem(MenuItem("Coffee", 250));
    myRestaurant.addMenuItem(MenuItem("Soft Drink", 100));
    myRestaurant.addMenuItem(MenuItem("Mineral Water", 50));

    // Adding some staff members
    myRestaurant.addStaff(Staff("IZHAR", "Manager", 50000));
    myRestaurant.addStaff(Staff("HARRIS", "Chef", 30000));
    myRestaurant.addStaff(Staff("AWAIS", "Chef", 15000));

    // Adding 20 waiters to the staff
```

```
for (int i = 1; i <= 20; ++i) {
    string waiterName = "Waiter" + to_string(i);
    myRestaurant.addStaff(Staff(waiterName, "Waiter", 12000)); // Adding waiters with a default salary of 12,000 PKR
}

// Adding table categories (20 tables divided into 4 categories for 2 floors)
// Floor 1
myRestaurant.addTableCategory(Table("Special", 5, "Floor 1"));
myRestaurant.addTableCategory(Table("Luxury", 2, "Floor 1"));

// Floor 2
myRestaurant.addTableCategory(Table("Normal", 11, "Floor 2"));
myRestaurant.addTableCategory(Table("VIP", 2, "Floor 2"));

// Display the menu, staff, and tables
myRestaurant.displayMenu();
myRestaurant.displayStaff();
myRestaurant.displayTables();

// Taking an order
Order order1;
order1.addItem(MenuItem("Burger", 599));
order1.addItem(MenuItem("Shawarma", 499));
myRestaurant.takeOrder(order1);

// Taking another order
    Order order2;
    order2.addItem(MenuItem("Tea", 150));
    order2.addItem(MenuItem("Coffee", 250));
    myRestaurant.takeOrder(order2);

    // Display all orders
    myRestaurant.displayOrders();

    return 0;
}
```

6.USER Interface (screenshot):





7.Advantage:

1.Modular Design: The program uses classes to separate concerns (e.g., MenuItem, Order, Staff, Table), making it easy to understand and maintain.

2.Scalability: New features like additional menu items, staff roles, or order details can be added without affecting existing functionality.

3.Reusability: Objects like MenuItem and Order can be reused in other parts of the program or extended for future use.

4.Customizability: Restaurants can easily adjust the menu, staff, or table configurations based on specific requirements.

5.Comprehensive Coverage: Includes essential operations for restaurant management, such as menu display, staff management, table organization, and order handling.

6.User-Friendly Output: Clear and well-structured output helps users understand the restaurant's setup and current operations.

8.Conclusion:

1. Modular and Expandable: The modular design allows for easy additions, such as new menu items, staff roles, or table categories, making the system scalable for larger restaurants.

2.User-Friendly Interaction: The clear and organized output provides a user-friendly interface for managing and viewing restaurant operations.

3.Practical Application: This program can serve as a foundation for building a more complex restaurant management system by incorporating features like inventory tracking, customer feedback, or billing integration.

4.Structured Representation: The separation of responsibilities into distinct classes ensures better code readability and maintainability.