VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi - 590018

Technical Report

On

"FOOD ORDER MANAGEMENT SYSTEM"

by

NIREEKSHA S	**4MT21CS096**
NISARGA SURESH POOJARI	**4MT21CS097**
NISHITHA	**4MT21CS098**
OLIVIA CAROL DSOUZA	**4MT21CS099**
VINEET PREMANANDA POOJARI	**4MT21CS100**

DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING

(*Accredited by NBA*)

MANGALORE INSTITUTE OF TECHNOLOGY &ENGINEERING

Accredited by NAAC with A+ Grade, An ISO 9001: 2015 Certified Institution

(*A Unit of Rajalaxmi Education Trust®, Mangalore - 575001*)

Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi

Badaga Mijar, Moodabidri-574225, Karnataka

2022-23

Table of Contents

1. Abstract

The "Food Order Management System" is a console-based program developed in C, designed to facilitate food ordering and employee management tasks. The system comprises three primary components: Customers, Food, and Employees.

1. **Customers:**
    - Users can place food orders by selecting items from a predefined menu.
    - Each customer's order includes information such as the customer's name, address, phone number, the ordered food item, and the quantity.
    - The system stores customer orders and allows viewing the last three orders.
2. **Food Menu:**
    - The program defines a static food menu with eight items, each having a name and a price.
    - Users can select items from the menu and specify quantities for their orders.
3. **Employees:**
    - The system supports employee management features in an administrative section.
    - Administrators can add employee records, view all employee records, search for employees by ID or name, and remove employee records.
    - Employee details include name, address, phone number, department, date of birth, and joining date.
4. **Navigation:**
    - The program offers a user-friendly menu-driven interface, allowing users to choose between ordering food or accessing the admin section.
    - In the admin section, administrators can perform various employee management tasks.
5. **Data Storage:**
    - Customer orders and employee records are stored in arrays within the program's memory.
    - The system can handle up to 10 customer records and 10 employee records.
6. **User Feedback:**
    - The program provides feedback to users, such as confirming successful orders and displaying information about the most recent orders.
    - Sleep functions are used to pause execution briefly for improved user experience.
7. **Clean Screen Functionality:**
    - A custom "cls" function is used to clear the console screen for a cleaner user interface.
8. **Exit Option:**
    - The program allows users to exit gracefully when they are done using the system.

Overall, the "Food Order Management System" provides a basic framework for managing food orders and employee records within a command-line environment. It can be further extended and enhanced to include additional features and improve usability.

2. Introduction

2.1 Background:

In the modern world, efficient management systems play a pivotal role in various sectors, including the food industry and workforce management. In this context, the "Food Order Management System" is designed to streamline and automate the processes involved in food ordering and employee record management. This system caters to the needs of both customers and administrators, making it a versatile solution for a restaurant or similar establishment.

2.2 Objectives:

The primary objectives of the "Food Order Management System" are as follows:

1. **Efficient Food Ordering:** One of the key objectives is to provide customers with an easy and efficient way to place food orders. The system offers a user-friendly menu, allowing customers to select their desired items and specify quantities.
2. **Order Tracking:** The system keeps a record of customer orders, enabling them to review their past orders. This helps in ensuring order accuracy and provides a convenient reference for customers.
3. **Employee Management:** The system also serves as an employee management tool. Administrators can add, view, search for, and remove employee records, making it easier to manage the workforce within the establishment.
4. **User-Friendly Interface:** Both customers and administrators benefit from a straightforward and intuitive interface, making it easy to navigate and perform tasks.
5. **Data Accuracy:** By automating data entry and retrieval, the system reduces the chances of manual errors and ensures the accuracy of customer and employee information.
6. **Efficient Customer Service:** The system enhances customer service by expediting the food ordering process and providing a quick view of recent orders. It also helps administrators access employee records efficiently.
7. **Streamlined Operations:** With automated food ordering and employee management, the system helps streamline the day-to-day operations of a food establishment, leading to increased efficiency and reduced administrative overhead.
8. **Scalability:** The system is designed to accommodate growth. It can handle a growing database of customer orders and employee records without compromising performance.

In summary, the "Food Order Management System" aims to provide a comprehensive solution for food establishments, improving the customer experience and simplifying employee management. It brings automation and efficiency to these crucial aspects of running a successful business in the food industry.

3. Technologies Used

1. **C Programming Language**: The entire code is written in C, which is a general-purpose programming language known for its efficiency and low-level system programming capabilities.
2. **Standard Input/Output (stdio.h)**: This header file is part of the C Standard Library and provides functions like **printf** and **scanf** for input and output operations.
3. **Standard Library (stdlib.h)**: This header file is part of the C Standard Library and provides functions like **system** for executing system commands and **sleep** for introducing delays.
4. **String Manipulation (string.h)**: This header file is part of the C Standard Library and provides functions for string manipulation, such as **strcpy** for copying strings and **strcmp** for comparing strings.
5. **Structures**: The code uses C structures (**struct**) to define custom data types like **struct Food**, **struct Customer**, and **struct Employee** to organize and store related pieces of data together.
6. **Control Flow**: The code utilizes various control flow constructs like **if** statements and **switch** statements to make decisions and control the program's flow.
7. **Loops**: The code includes **for** loops and **do-while** loops for repetitive tasks and menu-driven interfaces.
8. **User Input**: It uses **scanf** to get input from the user for menu choices and data entry.
9. **Console Clearing**: The **cls** function utilizes the **system** function to clear the console screen, which is a system-specific operation typically used in command-line programs.
10. **Array**: Arrays are used to store multiple instances of data. For example, arrays of **struct Food**, **struct Customer**, and **struct Employee** are used to manage multiple records of each type.
11. **User Interaction**: The code interacts with the user through command-line menus and displays information using **printf**.
12. **Functions**: The code defines several functions to encapsulate specific tasks, improving code organization and readability.
13. **Sleep Function**: The **sleep** function is used to introduce delays in the program, creating a pause between certain actions.
14. **Conditional Statements**: The code uses conditional statements like **if** and **switch** to make decisions based on user input.
15. **Data Storage**: Data is stored in memory and manipulated using C's data structures and functions.
16. **Error Handling**: Basic error handling is performed, such as checking for invalid input choices.

The code is a console-based application written in C, utilizing standard C libraries and programming constructs to manage a simple food ordering and employee management system.

4. System Architecture

4.1 Front-End:

- In this context, the front-end is the text-based user interface that interacts with the user through the terminal/console. The front-end handles user input and displays information to the user. In this code, the front-end consists of the menu-driven text interface that prompts the user to make choices.

4.2 Back-End:

- The back-end of this application is responsible for processing user input, managing data (in-memory data structures), and executing the core functionality of the program.
- It includes functions such as **orderFood**, **addEmployee**, **viewLastOrders**, **viewAllEmployees**, **searchEmployee**, and **removeEmployee**, which perform various operations based on user input.
- It also manages two arrays, **customers** (for storing customer orders) and **employees** (for storing employee records), which serve as in-memory data storage.

4.3 Database:

- In this code, there is no external database system. Instead, data is stored in memory using arrays (**customers** and **employees**) for the duration of the program's execution. This is a simple form of in-memory data storage, and the data is not persisted beyond the program's runtime.

This code represents a basic command-line application that performs simple food ordering and employee management tasks. It has a text-based front-end, a back-end for processing user input and managing data in memory, but it lacks a separate database component as it uses in-memory data structures for storage during runtime.

5. Project Modules

5.1 Order Food Module (orderFood function):

- This module allows a customer to place a food order.
- It displays a menu of food items with prices.
- The customer selects a food item by entering a number.
- The customer can specify the quantity, name, address, and phone number for the order.
- The order is added to the **customers** array.

5.2 Admin Section Module:

- This module allows administrative functions to be performed.
- The admin can access several sub-modules, including:
    - **Add Employee Records (addEmployee function):**
        - Allows the admin to add employee records, including name, address, phone number, department, date of birth, and joining date.
        - The employee details are stored in the **employees** array.
    - **View Last Orders (viewLastOrders function):**
        - Displays the last three customer orders, including customer name, ordered food, quantity, total price, address, and phone number.
    - **View All Employee Records (viewAllEmployees function):**
        - Displays a list of all employee records, including employee name, department, and phone number.
    - **Search Employee (searchEmployee function):**
        - Allows the admin to search for an employee by either ID or name.
        - Displays employee details if found.
    - **Remove Employee Records (removeEmployee function):**
        - Allows the admin to remove an employee record by specifying the employee's ID.
        - The employee record is removed from the **employees** array.

5.3 Main Program Module (main function):

- This is the main module that drives the entire program.
- It provides a menu for the user to choose between ordering food, accessing the admin section, or exiting the program.
- The user can navigate through the program using a series of menu choices.

5.4 Clear Screen Function (cls function):

- This function is used to clear the console screen, making the user interface more user-friendly.

Overall, the code you provided creates a basic Food Order Management System with customer order processing and an admin section for employee management and order viewing. You can further enhance this system by adding features like data persistence, more advanced user input validation, and error handling.

6. Design and Implementation

6.1 Front-End Design:

- In this console-based application, the "front-end" is the text-based user interface that interacts with the user. It consists of text menus and prompts for user input.
- The code provides a menu-driven interface for customers to order food and for administrators to manage employee records and view customer orders.
- There are simple text-based menus and prompts for user input, but there is no graphical user interface (GUI) or web-based front-end.

6.2 Back-End Design:

- In this code, the "back-end" refers to the logic and functionality of the application. It includes functions for ordering food, managing employee records, and viewing customer orders.
- The back-end handles data structures like arrays of **Customer** and **Employee** structures to store information about customers and employees.
- Functions like **orderFood**, **addEmployee**, **viewLastOrders**, **viewAllEmployees**, **searchEmployee**, and **removeEmployee** perform various operations on these data structures.
- There is no server or extensive back-end processing in this code; everything is done within the same program.

6.3 Database Design:

- This code doesn't use a traditional database system like MySQL, PostgreSQL, or MongoDB. Instead, it relies on arrays of structures to store data in memory.
- Each **Customer** and **Employee** structure serves as a record in memory.
- When a new customer places an order or an employee record is added, the data is stored in these arrays.
- There is no persistence of data beyond the program's runtime. When the program exits, all data is lost.

7. Features and Functionality

7.1 **Order Food:**

- Users can select from a list of food items in the menu.
- They can specify the quantity of the selected food item.
- Users are prompted to provide their name, address, and phone number for the order.
- The order details are stored in the **customers** array.

7.2 Admin Section:

- Administrative users can access various features for managing employees and viewing orders.
- Requires authentication to enter the admin section.

7.3 Employee Management:

- Admins can add employee records, including name, address, phone number, department, date of birth, and joining date.
- Admins can view all employee records.
- Admins can search for employees by ID or name.
- Admins can remove employee records by ID.

7.4 View Last Orders:

- Admins can view the last three customer orders, including customer name, ordered food, quantity, total price, address, and phone number.

7.5 User-Friendly Interface:

- The program provides clear and formatted menus for users and admins.
- It includes an option to return to the main menu at any time.

7.6 Exiting the Program:

- Users and admins can exit the program gracefully.

7.7 Clear Screen Functionality:

- The code includes a **cls()** function to clear the console screen for a cleaner user experience.

Functionality:

- The code defines three main structures: **Food**, **Customer**, and **Employee**, which store information about food items, customer orders, and employee records, respectively.
- The **main** function serves as the program's entry point and manages the primary flow of the program.
- The program allows users to place food orders, which are stored in the **customers** array.
- The admin section provides various options for managing employees and viewing customer orders. Administrative functionality includes adding employees, viewing employee records, searching for employees, and removing employees.
- The program uses a loop to continuously display menus and process user input until the user chooses to exit.
- Administrative actions and user orders are stored and managed within their respective arrays.
- The program clears the console screen (**cls()**) to maintain a clean interface and improve readability.

8. Testing

8.1 Unit Testing:

Unit testing involves testing individual components or functions of the code in isolation to ensure they work as expected. Can perform unit testing on each of the functions like **orderFood**, **addEmployee**, **viewLastOrders**, **viewAllEmployees**, **searchEmployee**, and **removeEmployee**.

8.2 Integration Testing:

Integration testing checks how different parts of the system work together. In your code, you can perform integration testing by testing how different functions interact with each other. For example, you can test the interaction between the **orderFood** function and the **viewLastOrders** function.

8.3 User Acceptance Testing:

User Acceptance Testing (UAT) involves testing the software with real users or stakeholders to ensure that it meets their requirements and expectations. This typically requires involving actual users who interact with the software and provide feedback. UAT often goes beyond code testing and includes usability testing, user interface testing, and more.

Need to have real users or stakeholders interact with the program, place orders, add employee records, view orders, search for employees, and remove employee records. They would provide feedback on whether the system meets their needs and expectations. UAT is typically done in a real-world environment and may involve non-technical users.

9. Challenges Faced

1. **Limited Data Storage**: Program has fixed-size arrays for customers and employees (10 each). This limits the number of records you can store. If you want to handle more data, you should consider dynamic memory allocation or using data structures like linked lists.
2. **Data Validation**: Program assumes that users will always provide valid input. It doesn't handle cases where users might enter incorrect or unexpected data, which could lead to crashes or incorrect results. Adding input validation and error handling would make the program more robust.
3. **Security**: Storing sensitive information like addresses and phone numbers without encryption or access controls can be a security risk. If this is intended for a real-world application, consider security measures to protect user data.
4. **Code Organization**: Code is in a single large **main** function, which can make it hard to read and maintain as it grows. Consider breaking it into smaller functions with clear responsibilities to improve readability and maintainability.
5. **Error Reporting**: When errors occur (e.g., invalid input), program should provide informative error messages to help users understand what went wrong and how to correct it.
6. **File I/O**: To make the data persist beyond a single run of the program, might want to add file input/output operations to save and load customer and employee data from files.
7. **Platform Dependency**: Use of **system("cls")** to clear the screen is platform-dependent and may not work on all systems. Consider using more cross-platform solutions for screen clearing.
8. **Lack of Documentation**: code lacks comments and documentation. Adding comments explaining the purpose of functions, variables, and the overall program flow can make it easier for others (and your future self) to understand and maintain the code.
9. **Scalability**: If you plan to expand this program to handle more complex features or larger datasets, might need to redesign parts of the code to maintain its maintainability and performance.
10. **User Experience**: The user interface is minimal, which may be acceptable for a simple application. However, for a real-world system, should consider improving the user experience, such as providing more user-friendly prompts and feedback.
11. **Sleep Function**: The use of **sleep** for pausing the program may not be user-friendly. Users might want the option to continue immediately or interact with the program during this time.
12. **Hard-Coded Values**: Magic numbers (e.g., the number of items in the food menu) are hard-coded throughout the program. It's better to define these as constants to improve code readability and maintainability.
13. **Memory Management**: Although your program uses fixed-size arrays, you might want to handle memory allocation and deallocation more gracefully in case you decide to switch to dynamic data structures in the future.

10. Future Enhancements

1. **User Authentication**: Implement user authentication for administrators to ensure secure access to the admin section. This can involve username and password validation.
2. **Database Integration**: Currently, the system stores data in memory, which means that data is lost when the program exits. Consider integrating a database (e.g., SQLite, MySQL) to persist customer orders and employee records between program runs.
3. **Menu Management**: Allow administrators to dynamically add, update, or remove items from the food menu. This would make the system more flexible and adaptable to changes in the menu.
4. **Order History**: Implement a comprehensive order history feature that allows customers to view their past orders and administrators to access order statistics and trends.
5. **Payment Processing**: Add payment processing capabilities so that customers can pay for their orders online. This may involve integrating with payment gateways like PayPal or Stripe.
6. **Inventory Management**: Integrate inventory tracking to ensure that items are available in stock. Update the inventory count when orders are placed and trigger alerts when items are running low.
7. **User Interface Improvements**: Enhance the user interface with a graphical user interface (GUI) for a more user-friendly experience, especially for non-technical users.
8. **Order Tracking**: Implement order tracking for customers so they can check the status of their orders, including estimated delivery times.
9. **Reporting and Analytics**: Create reports and analytics tools for administrators to gain insights into sales, customer preferences, and employee performance.
10. **Customer Feedback**: Add a feature for customers to provide feedback and ratings for the food they ordered and the service they received.
11. **Localization**: If the system is used in different regions, consider adding support for multiple languages and currencies.
12. **Mobile App**: Develop a mobile application version of the system to reach a broader audience and provide on-the-go ordering and management.
13. **Security Enhancements**: Ensure that sensitive data, such as customer information and payment details, is securely stored and transmitted. Implement encryption and security best practices.
14. **Error Handling and Logging**: Improve error handling by providing meaningful error messages to users and implementing a logging system to track system errors and events.
15. **Testing and Quality Assurance**: Implement automated testing to ensure the system functions correctly and undergoes regular quality assurance reviews.
16. **Scalability**: Design the system architecture to be scalable, allowing it to handle increased traffic and data as the business grows.
17. **Feedback Mechanism**: Include a feedback mechanism for users to report issues or suggest improvements, helping to continuously enhance the system.

11. Conclusion

1. **Data Structures**: The code uses three custom-defined structures: **Food**, **Customer**, and **Employee** to represent food items, customer information, and employee records, respectively.
2. **Functionality**:
    - Customers can place orders from a predefined food menu.
    - Customer details, such as name, address, and phone number, are collected during the order process.
    - The program maintains a list of customer orders.
    - Administrators can add employee records, view the last customer orders, view all employee records, search for employees by ID or name, and remove employee records.
    - The program utilizes a simple text-based menu system to navigate these functionalities.
    - The code includes a **cls** function to clear the console screen, enhancing user experience.
3. **Code Organization**: The code is well-structured with functions for each major operation, making it easy to understand and maintain.
4. **Input Validation**: Basic input validation is performed to check for valid user inputs, such as order choices and employee IDs.
5. **User Interface**: The program provides clear prompts and menus for users, making it user-friendly.
6. **Exit Condition**: Users can exit the program gracefully by choosing option 3 in the main menu.
7. **Error Handling**: The code provides error messages for invalid inputs or actions.

However, there is room for improvement and further development, such as enhancing input validation, error handling, and potentially expanding the functionality of the system. Additionally, ensuring data integrity and security could be a consideration for real-world applications.

12. References

1. **C Standard Library**: The code makes use of several standard C library functions, such as **printf**, **scanf**, **strcpy**, **sleep**, **system**, and **strcmp**. These functions are part of the C standard library and are documented in various C programming resources.
2. **Structures in C**: The code defines three structures: **struct Food**, **struct Customer**, and **struct Employee**. C programming tutorials or textbooks on structures to understand how they work.
3. **Control Structures (if, switch, do-while)**: The code uses control structures like **if**, **switch**, and **do-while** for decision-making and looping. You can learn about these control structures in any C programming resource.
4. **Arrays in C**: The code uses arrays to store food menu items and customer/employee records. Learn about arrays in C to understand how they are used.
5. **User Input and Output**: The code relies heavily on user input and output through **scanf** and **printf**. Refered to C programming books or tutorials on how to handle user input and display output.

13. Appendices

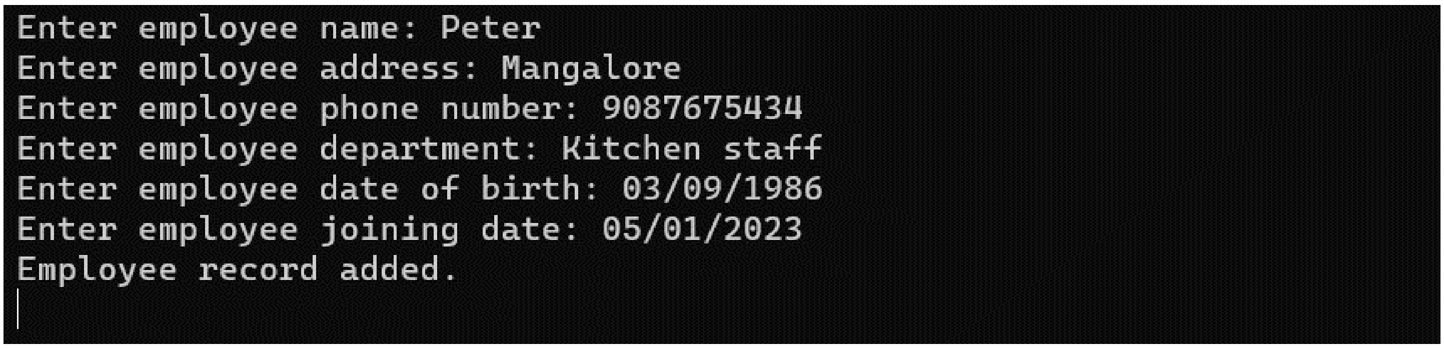
1.Code Snippets

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Food {

    char name[50];

    float price;

};


struct Customer {

    char name[50];

    char address[100];

    char phone[15];

    struct Food orderedFood;

    int quantity;

};


struct Employee {

    char name[50];

    char address[100];

    char phone[15];

    char department[50];

    char dob[15];

    char joiningDate[15];
```

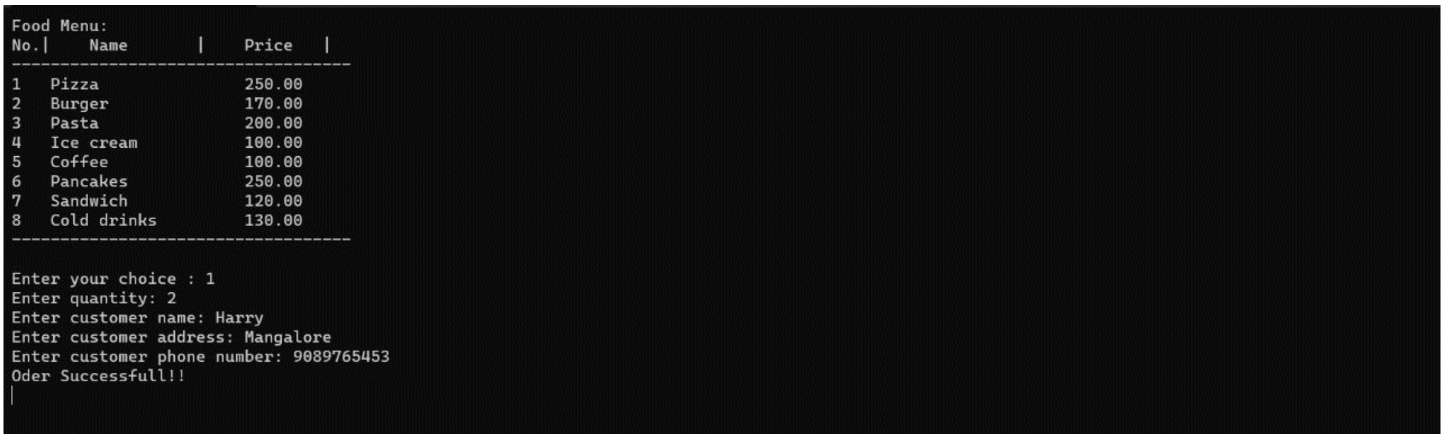
```
};

void cls();

int i;

void orderFood(struct Customer *customer) {

    struct Food foodMenu[8];

    strcpy(foodMenu[0].name, "Pizza");

    foodMenu[0].price = 250;

    strcpy(foodMenu[1].name, "Burger");

    foodMenu[1].price = 170;

    strcpy(foodMenu[2].name, "Pasta");

    foodMenu[2].price = 200;

    strcpy(foodMenu[3].name, "Ice cream");

    foodMenu[3].price = 100;

    strcpy(foodMenu[4].name, "Coffee");

    foodMenu[4].price = 100;

    strcpy(foodMenu[5].name, "Pancakes");

    foodMenu[5].price = 250;

    strcpy(foodMenu[6].name, "Sandwich");

    foodMenu[6].price = 120;

    strcpy(foodMenu[7].name, "Cold drinks");

    foodMenu[7].price = 130;



    printf("Food Menu:\n");

            printf( "No.|    Name        |    Price   |\n");

    printf("-----------------------------------\n");

    for (i = 0; i < 8; i++) {

        printf("%-4d%-20s%-9.2f\n", i + 1, foodMenu[i].name, foodMenu[i].price);
    }
            printf("-----------------------------------\n");
    int choice;
    printf("\nEnter your choice : ");
    scanf("%d", &choice);
    if (choice >= 1 && choice <= 8) {
        customer->orderedFood = foodMenu[choice - 1];
        printf("Enter quantity: ");
        scanf("%d", &customer->quantity);
        printf("Enter customer name: ");
        scanf(" %s", customer->name);
        printf("Enter customer address: ");
        scanf(" %s", customer->address);
        printf("Enter customer phone number: ");
        scanf(" %s", customer->phone);
    } else {
        printf("Invalid choice.\n");
    }
    printf("Oder Successfull!!\n");
    sleep(3);
    cls();
}


void addEmployee(struct Employee *employees, int *numEmployees) {
            cls();
    printf("Enter employee name: ");
    scanf(" %s", employees[*numEmployees].name);
    printf("Enter employee address: ");

    scanf(" %s", employees[*numEmployees].address);

    printf("Enter employee phone number: ");

    scanf(" %s", employees[*numEmployees].phone);

    printf("Enter employee department: ");

    scanf(" %s", employees[*numEmployees].department);

    printf("Enter employee date of birth: ");

    scanf(" %s", employees[*numEmployees].dob);

    printf("Enter employee joining date: ");

    scanf(" %s", employees[*numEmployees].joiningDate);



    (*numEmployees)++;

    printf("Employee record added.\n");

     sleep(3);

     cls();

}



void viewLastOrders(struct Customer *customers, int numCustomers) {

            cls();

    if (numCustomers > 0) {

        printf("Last orders:\n");

        for (i = numCustomers - 1; i >= 0 && i >= numCustomers - 3; i--) {

            printf("Customer: %s\n", customers[i].name);

            printf("Ordered Food: %s\n", customers[i].orderedFood.name);

            printf("Quantity: %d\n", customers[i].quantity);

            printf("Total Price: %.2f\n", customers[i].orderedFood.price * customers[i].quantity);

            printf("Address: %s\n", customers[i].address);
            printf("Phone: %s\n", customers[i].phone);
        }
    } else {
        printf("No orders available.\n");
    }
    sleep(3);
         cls();
}

void viewAllEmployees(struct Employee *employees, int numEmployees) {
            cls();
    if (numEmployees >0){
            printf("Employee records:\n");
            for (i = 0; i < numEmployees; i++) {
        printf("\nEmployee Name: %s\n", employees[i].name);
        printf("Department: %s\n", employees[i].department);
        printf("Phone: %s\n", employees[i].phone);
      }
  }
     else {
                        printf("No employees found.\n");
    }
    sleep(3);
            cls();
}

void searchEmployee(struct Employee *employees, int numEmployees) {
        cls();

    int choice;

    printf("Search employee by:\n");

    printf("1. ID\n");

    printf("2. Name\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);


    if (choice == 1) {

        int id;

        printf("Enter employee ID: ");

        scanf("%d", &id);

        if (id >= 1 && id <= numEmployees) {

            printf("Employee Name: %s\n", employees[id - 1].name);

            printf("Department: %s\n", employees[id - 1].department);

            printf("Phone: %s\n", employees[id - 1].phone);

        } else {

            printf("Invalid ID.\n");

        }

    } else if (choice == 2) {

        char name[50];

        printf("Enter employee name: ");

        scanf(" %s", name);


        for (i = 0; i < numEmployees; i++) {

            if (strcmp(name, employees[i].name) == 0) {

                printf("Employee Name: %s\n", employees[i].name);
                printf("Department: %s\n", employees[i].department);

                printf("Phone: %s\n", employees[i].phone);

                break;

            }

        }

    } else {

        printf("Invalid choice.\n");

    }

     sleep(3);

            cls();

}



void removeEmployee(struct Employee *employees, int *numEmployees) {

            cls();

    int id;

    printf("Enter employee ID to remove: ");

    scanf("%d", &id);



    if (id >= 1 && id <= *numEmployees) {

        for (i = id - 1; i < *numEmployees - 1; i++) {

            employees[i] = employees[i + 1];

        }

        (*numEmployees)--;

        printf("Employee record removed.\n");

    } else {

        printf("Invalid ID.\n");

    }
    sleep(3);

        cls();

}


int main() {

        printf("\n\n\n\n\n\n\n\n\n\n\n\t\t\t\t\tFOOD ORDER MANAGEMENT SYSTEM ");

        sleep(2);

        cls();

   struct Customer customers[10];

   int numCustomers = 0;

   struct Employee employees[10];

   int numEmployees = 0;


   int choice;

   do {

        cls();

       printf("1. Order Food\n");

       printf("2. Admin Section\n");

       printf("3. Exit\n");

       printf("Enter your choice: ");

       scanf("%d", &choice);

       cls();

       int adminChoice;

       switch (choice) {

           case 1:

               orderFood(&customers[numCustomers]);

               numCustomers++;
    break;
  case 2:


    do {
        cls();
      printf("\nAdmin Section\n");
      printf("1. Add Employee Records\n");
      printf("2. View Last Orders\n");
      printf("3. View All Employee Records\n");
      printf("4. Search Employee\n");
      printf("5. Remove Employee Records\n");
      printf("6. Back to Main Menu\n");
      printf("Enter your choice: ");
      scanf("%d", &adminChoice);

      switch (adminChoice) {
        case 1:
          addEmployee(employees, &numEmployees);
          break;
        case 2:
          viewLastOrders(customers, numCustomers);
          break;
        case 3:
          viewAllEmployees(employees, numEmployees);
          break;
        case 4:
          searchEmployee(employees, numEmployees);
```

```
                        break;

                    case 5:

                        removeEmployee(employees, &numEmployees);

                        break;

                    case 6:

                        break;

                    default:

                        printf("Invalid choice.\n");

                        break;

                }

            } while (adminChoice != 6);

            break;

        case 3:

            printf("Exiting...\n");

            break;

        default:

            printf("Invalid choice.\n");

            break;

    }

} while (choice != 3);


return 0;

}

void cls(){


system("cls");

}
```

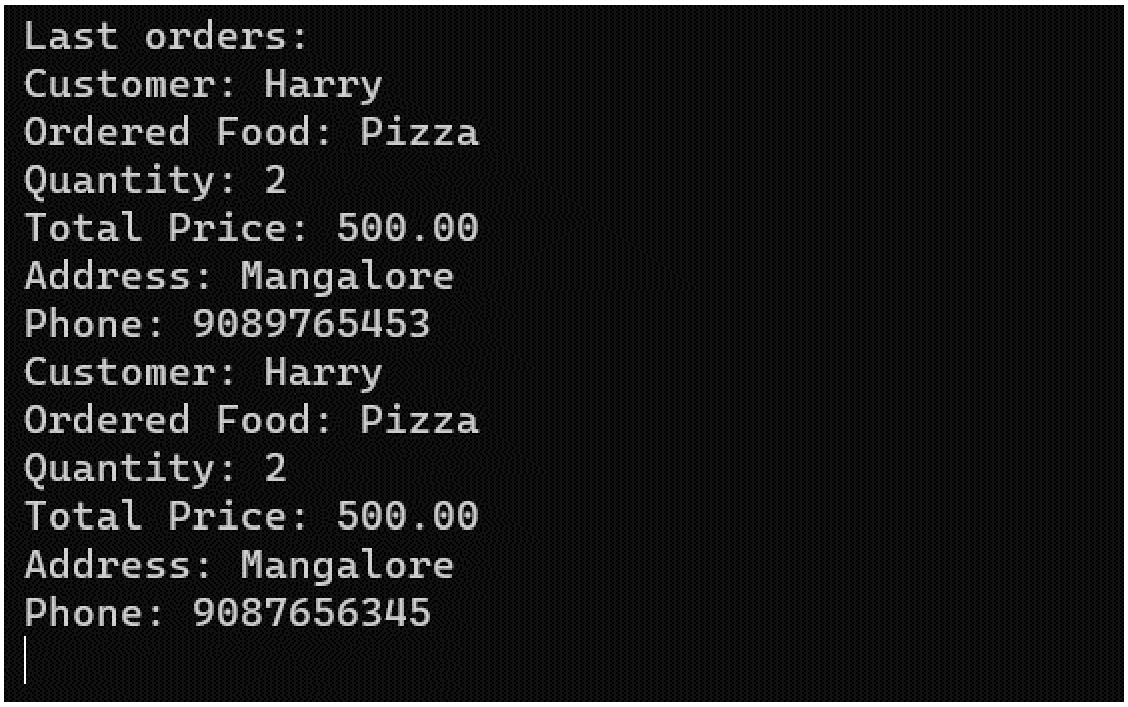
2. Screenshots



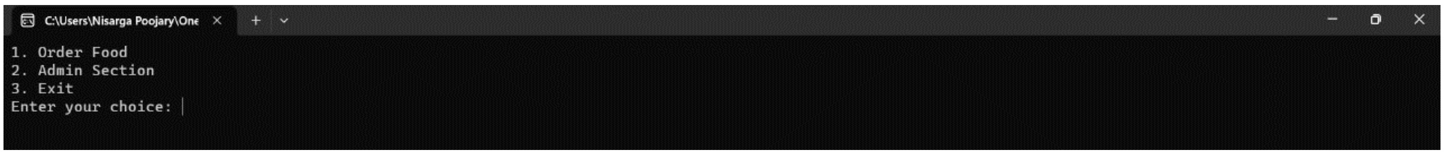
```
Enter employee name: Peter
Enter employee address: Mangalore
Enter employee phone number: 9087675434
Enter employee department: Kitchen staff
Enter employee date of birth: 03/09/1986
Enter employee joining date: 05/01/2023
Employee record added.
```



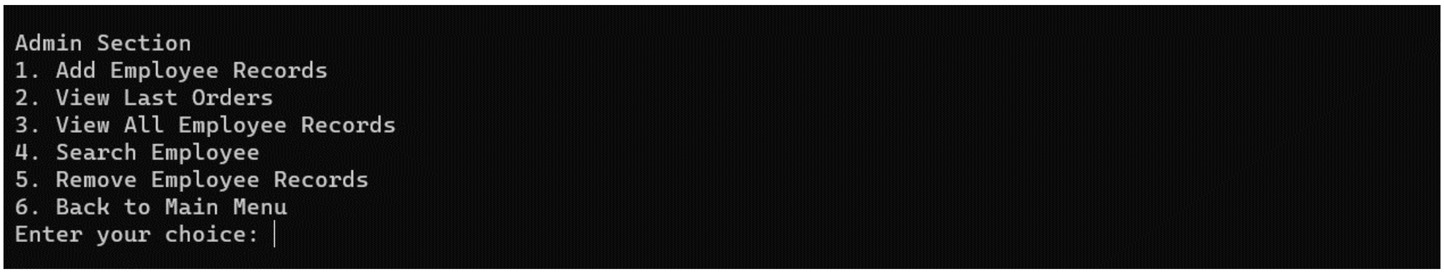
```
Food Menu:
No.|    Name       |    Price   |
-----------------------------------
1   Pizza               250.00
2   Burger              170.00
3   Pasta               200.00
4   Ice cream           100.00
5   Coffee              100.00
6   Pancakes            250.00
7   Sandwich            120.00
8   Cold drinks         130.00
-----------------------------------

Enter your choice : 1
Enter quantity: 2
Enter customer name: Harry
Enter customer address: Mangalore
Enter customer phone number: 9089765453
Oder Successfull!!
```



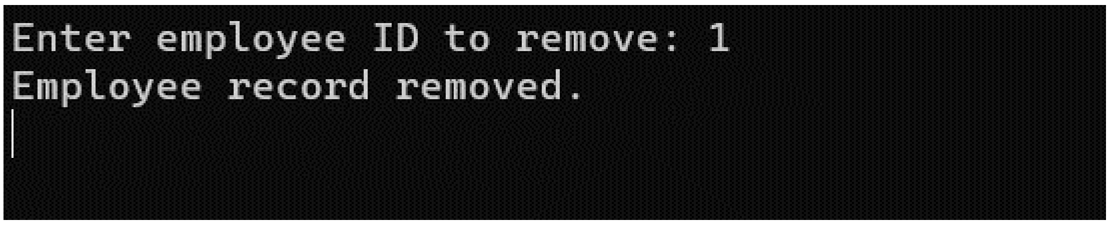
```
Last orders:
Customer: Harry
Ordered Food: Pizza
Quantity: 2
Total Price: 500.00
Address: Mangalore
Phone: 9089765453
Customer: Harry
Ordered Food: Pizza
Quantity: 2
Total Price: 500.00
Address: Mangalore
Phone: 9087656345
```



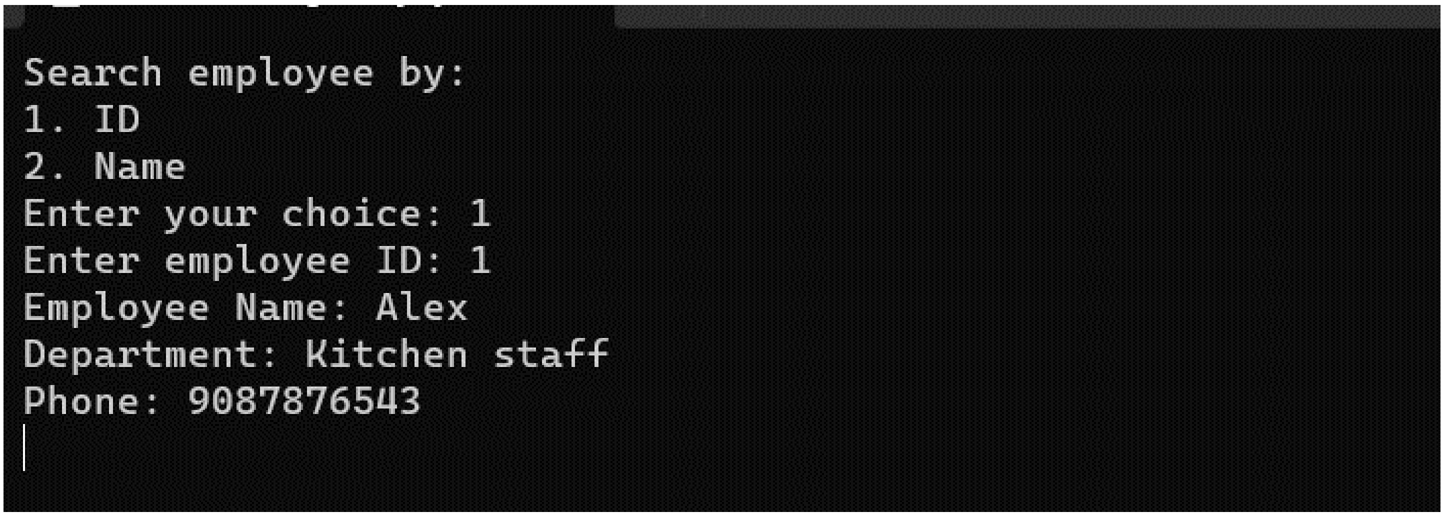
```
1. Order Food
2. Admin Section
3. Exit
Enter your choice:
```



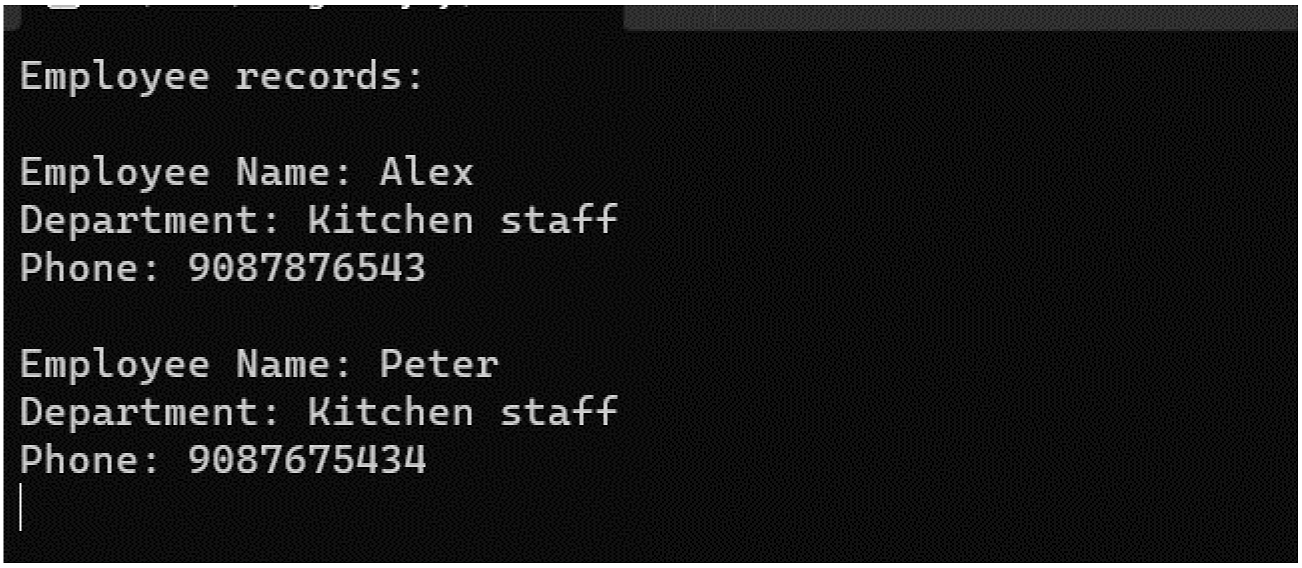
```
Admin Section
1. Add Employee Records
2. View Last Orders
3. View All Employee Records
4. Search Employee
5. Remove Employee Records
6. Back to Main Menu
Enter your choice:
```



```
Enter employee ID to remove: 1
Employee record removed.
```



```
Search employee by:
1. ID
2. Name
Enter your choice: 1
Enter employee ID: 1
Employee Name: Alex
Department: Kitchen staff
Phone: 9087876543
```



```
Employee records:

Employee Name: Alex
Department: Kitchen staff
Phone: 9087876543

Employee Name: Peter
Department: Kitchen staff
Phone: 9087675434
```