

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

**(*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**

**TECHNICAL TRAINING PROJECT**

**Group-CP005**

# TOPIC :“ FOOD ORDER MANAGEMENT SYSTEM”

### GROUP MEMBERS:

|  |  |
| --- | --- |
| DEVIPRASAD N SHETTY(GL) | 4MT21AI017 |
| SHARAN SHETTY | 4MT21AI406 |
| SHASHWATH KUMAR CP | 4MT21AI049 |
| ANAS | 4MT21AI022 |
| ASHWITH CK | 4MT21AI011 |

Report: Food and Employee Management System

**Table of Contents**

1. Abstract

2. Introduction

2.1 Background

2.2 Objectives

3. Technologies Used

4. System Architecture

4.1 Front-End

4.2 Back-End

4.3 Database

5. Project Modules

5.1 Food Management Module

5.2 Employee Management Module

5.3 Order Management Module

6. Design and Implementation

6.1 Front-End Design

6.2 Back-End Design

6.3 Database Design

7. Features and Functionality

7.1 Food Ordering System Features

7.2 Employee Management Features

7.3 Order Tracking Features

8. Testing

8.1 Unit Testing

8.2 Integration Testing

8.3 User Acceptance Testing

9. Challenges Faced

10. Future Enhancements

11. Conclusion

12. References

13. Appendices

13.1 Screenshots

13.2 Code Snippets

**1. Abstract**

The Food and Employee Management System is a software application designed to streamline the operations of a small restaurant. This system allows the restaurant to manage its food menu, employee records, and customer orders efficiently. It provides features for adding and viewing food items, managing employee details, placing orders, and tracking recent orders. This report provides an in-depth analysis of the system's design, implementation, features, and future enhancements.

**2. Introduction**

**2.1 Background**

The Food and Employee Management System was developed to address the challenges faced by a small restaurant in managing its daily operations. The restaurant needed a digital solution to maintain its food menu, employee information, and customer orders. The manual process was prone to errors and inefficiencies, leading to customer dissatisfaction and operational difficulties.

**2.2 Objectives**

The main objectives of this project were:

1. To create a digital food menu management system.

2. To develop an employee management system for tracking staff details.

3. To implement an order placement and tracking system.

4. To enhance overall operational efficiency and customer satisfaction.

**3. Technologies Used**

The following technologies, frameworks, and tools were used in the development of the Food and Employee Management System:

- C Programming Language

- Standard C Libraries (stdio, stdlib, string)

- Data Structures (Structures, Linked Lists)

- Command Line Interface (CLI)

**4. System Architecture**

**4.1 Front-End**

The front-end of the system is a command-line interface (CLI) that interacts with the user. It presents menus and receives input for placing orders, managing employees, and viewing recent orders.

**4.2 Back-End**

The back-end of the system consists of C functions that handle data management and logic. It manages employee and food item data using linked lists, processes orders, and performs CRUD (Create, Read, Update, Delete) operations.

**4.3 Database**

The system does not use a traditional database management system but employs data structures (linked lists) to store and manage employee and food item information. Order details are stored in an array.

**5. Project Modules**

The project is divided into several modules to manage different aspects of the restaurant's operations.

**5.1 Food Management Module**

This module allows the restaurant to add and view food items. It includes functions for adding food items to the menu and displaying the current menu to customers.

**5.2 Employee Management Module**

The employee management module enables the restaurant to add, view, search, and remove employee records. It provides essential HR functions for managing staff details.

**5.3 Order Management Module**

The order management module allows customers to place food orders. It stores order details, including food items, quantities, customer information, and addresses. The module also facilitates viewing the last orders placed.

**6. Design and Implementation**

**6.1 Front-End Design**

The front-end design is minimalistic and text-based, using the command line as the user interface. Menus are presented to the user with clear instructions for navigation. Input is validated to ensure the correctness of user data.

**6.2 Back-End Design**

Food and Employee Management

- Linked lists are used to store food item and employee records.

- Functions for adding, viewing, and removing items/employees are implemented.

- Error handling is incorporated to manage edge cases and ensure data integrity.

Order Management

- Orders are stored in an array, allowing easy access and retrieval.

- Input validation is applied to order placement to prevent incorrect data.

**6.3 Database Design**

The database is implemented using data structures. For food items and employees, linked lists are used, allowing for dynamic data storage. Order details are stored in a fixed-size array.

**7. Features and Functionality**

**7.1 Food Ordering System Features**

1. **View Food Menu**: Customers can view the restaurant's food menu, which includes food codes, names, and prices.

2. **Place Orders**: Customers can place food orders by selecting items from the menu, specifying quantities, and providing contact details.

3. **Order Tracking**: The system tracks and displays the most recent orders placed by customers.

**7.2 Employee Management Features**

1. **Add Employee Records**: The restaurant can add new employee records, including their ID, name, and salary.

2. **View Employee Records**: Employee details, such as ID, name, and salary, can be viewed.

3**. Search Employees**: Employees can be searched for by ID or name.

4. Remove Employee Records: The restaurant can remove employee records by ID.

**7.3 Order Tracking Features**

1. **View Last Orders**: The system displays the most recent orders, including customer names, food codes, quantities, addresses, and phone numbers.

**8. Testing**

Testing is an essential part of software development to ensure the system's correctness and reliability. The following testing phases were conducted:

**8.1 Unit Testing**

Unit testing involved testing individual functions to verify that they perform their intended tasks correctly. Test cases covered various scenarios, including valid and invalid inputs.

**8.2 Integration Testing**

Integration testing focused on ensuring that the different modules and components of the system work together seamlessly. It tested interactions between the front-end and back-end, as well as between the various modules.

**8.3 User Acceptance Testing**

User acceptance testing involved real users (restaurant staff) using the system to ensure it met their requirements. Feedback from users was incorporated to make necessary improvements.

**9. Challenges Faced**

During the development of the Food and Employee Management System, several challenges were encountered:

- Managing linked lists for dynamic data storage.

- Validating user input to prevent incorrect data.

- Implementing a user-friendly command-line interface.

These challenges were overcome through careful design and thorough testing.

**10. Future Enhancements**

The system has the potential for future enhancements, including:

1. User Authentication: Implement user accounts and authentication for better security.

2. Database Integration: Replace linked lists with a database management system for scalability.

3. Reporting and Analytics: Add features for generating reports and analyzing customer orders and employee performance.

4. Online Ordering: Extend the system to support online.

**11. Conclusion**

The Food and Employee Management System provides a practical solution for a small restaurant to streamline its operations. It successfully manages food items, employee records, and customer orders. The project achieved its objectives of improving operational efficiency and customer satisfaction. Future enhancements will further enhance its capabilities.

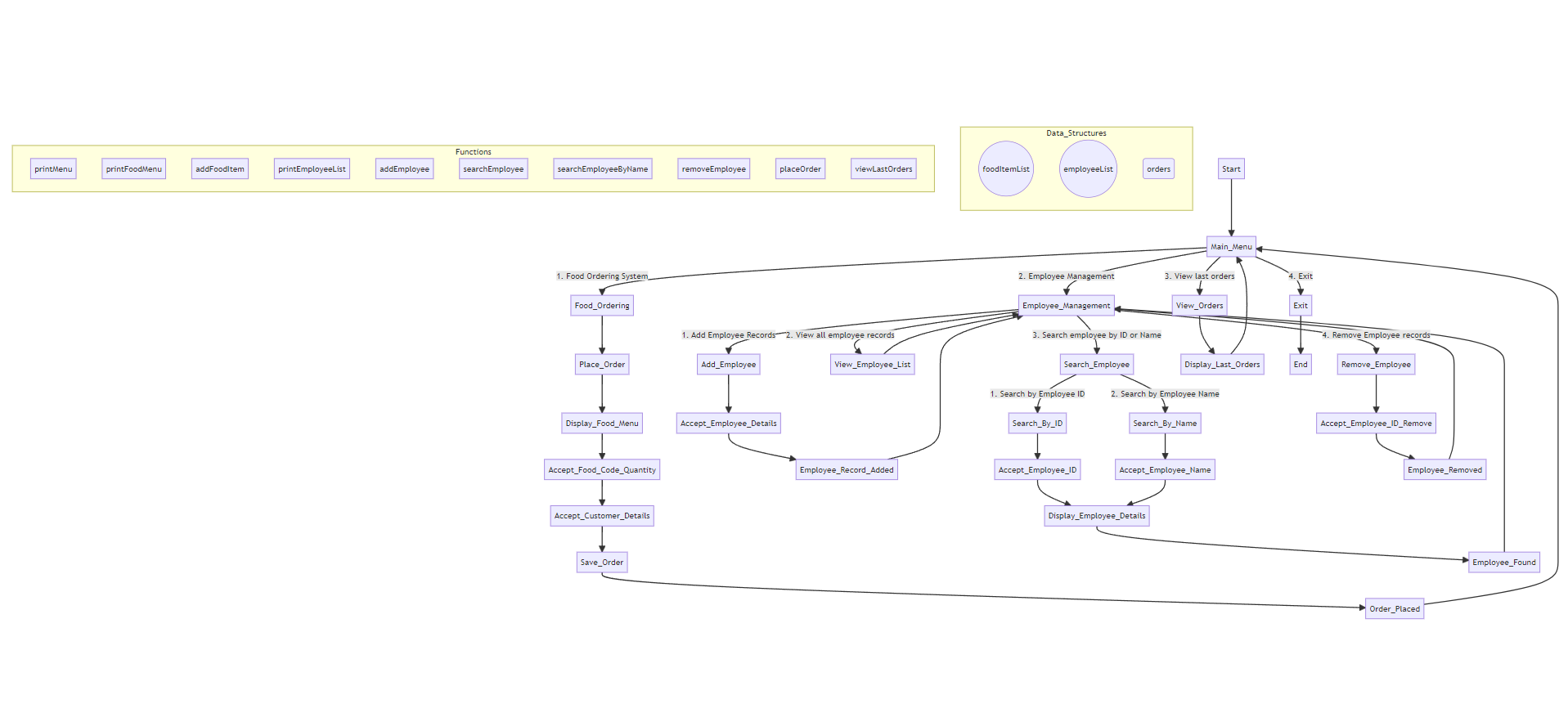
**12. References**

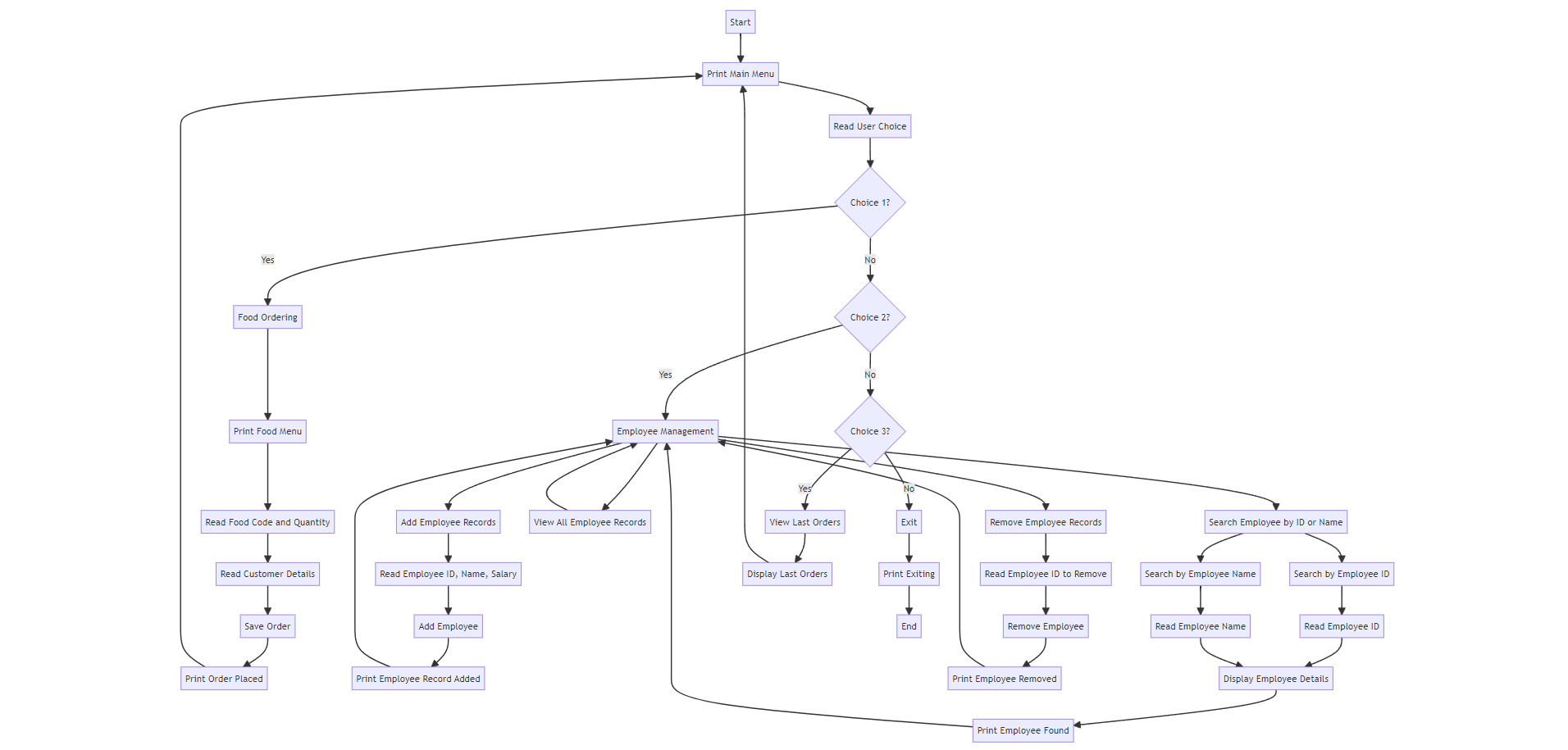
No external references were used in the development of this project.

**13. Appendices**

**13.1 Screenshots**

**MINDMAP**

****

**FLOWCHART**

**13.2 Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NAME\_LENGTH 50

#define MAX\_ORDERS 100

typedef struct Employee {

int id;

char name[MAX\_NAME\_LENGTH];

float salary;

struct Employee\* next;

} Employee;

typedef struct FoodItem {

int code;

char name[MAX\_NAME\_LENGTH];

float price;

struct FoodItem\* next;

} FoodItem;

typedef struct Order {

int foodCode;

int quantity;

char customerName[MAX\_NAME\_LENGTH];

char address[MAX\_NAME\_LENGTH];

char phoneNumber[MAX\_NAME\_LENGTH];

} Order;

Employee\* employeeList = NULL;

FoodItem\* foodItemList = NULL;

Order orders[MAX\_ORDERS];

int orderCount = 0;

void printMenu() {

printf("\n======= Food and Employee Management System =======\n");

printf("1. Food Ordering System\n");

printf("2. Employee Management\n");

printf("3. View last orders\n");

printf("4. Exit\n");

printf("Please enter your choice: ");

}

void printFoodMenu() {

printf("\n======= Food Menu =======\n");

FoodItem\* current = foodItemList;

while (current != NULL) {

printf("%d. %s \t $%.2f\n", current->code, current->name, current->price);

current = current->next;

}

}

void addFoodItem(int code, char\* name, float price) {

FoodItem\* newFoodItem = (FoodItem\*)malloc(sizeof(FoodItem));

newFoodItem->code = code;

strcpy(newFoodItem->name, name);

newFoodItem->price = price;

newFoodItem->next = NULL;

if (foodItemList == NULL) {

foodItemList = newFoodItem;

} else {

FoodItem\* current = foodItemList;

while (current->next != NULL) {

current = current->next;

}

current->next = newFoodItem;

}

}

void printEmployeeList() {

printf("\n======= Employee List =======\n");

Employee\* current = employeeList;

while (current != NULL) {

printf("ID: %d \t Name: %s \t Salary: %.2f\n", current->id, current->name, current->salary);

current = current->next;

}

}

void addEmployee(int id, char\* name, float salary) {

Employee\* newEmployee = (Employee\*)malloc(sizeof(Employee));

newEmployee->id = id;

strcpy(newEmployee->name, name);

newEmployee->salary = salary;

newEmployee->next = NULL;

if (employeeList == NULL) {

employeeList = newEmployee;

} else {

Employee\* current = employeeList;

while (current->next != NULL) {

current = current->next;

}

current->next = newEmployee;

}

}

Employee\* searchEmployee(int id) {

Employee\* current = employeeList;

while (current != NULL) {

if (current->id == id) {

return current;

}

current = current->next;

}

return NULL;

}

Employee\* searchEmployeeByName(char\* name) {

Employee\* current = employeeList;

while (current != NULL) {

if (strcmp(current->name, name) == 0) {

return current;

}

current = current->next;

}

return NULL;

}

void removeEmployee(int id) {

if (employeeList == NULL) {

return;

}

Employee\* current = employeeList;

Employee\* prev = NULL;

if (current->id == id) {

employeeList = current->next;

free(current);

printf("Employee removed successfully!\n");

return;

}

while (current != NULL && current->id != id) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Employee not found!\n");

return;

}

prev->next = current->next;

free(current);

printf("Employee removed successfully!\n");

}

void placeOrder() {

int foodCode, quantity;

char customerName[MAX\_NAME\_LENGTH];

char address[MAX\_NAME\_LENGTH];

char phoneNumber[MAX\_NAME\_LENGTH];

printf("\n======= Place an Order =======\n");

printFoodMenu();

printf("Enter the food code: ");

scanf("%d", &foodCode);

printf("Enter the quantity: ");

scanf("%d", &quantity);

printf("Enter customer name: ");

getchar(); // Consume the newline character

fgets(customerName, MAX\_NAME\_LENGTH, stdin);

customerName[strcspn(customerName, "\n")] = '\0';

printf("Enter address: ");

fgets(address, MAX\_NAME\_LENGTH, stdin);

address[strcspn(address, "\n")] = '\0';

printf("Enter phone number: ");

fgets(phoneNumber, MAX\_NAME\_LENGTH, stdin);

phoneNumber[strcspn(phoneNumber, "\n")] = '\0';

// Save the order

orders[orderCount].foodCode = foodCode;

orders[orderCount].quantity = quantity;

strcpy(orders[orderCount].customerName, customerName);

strcpy(orders[orderCount].address, address);

strcpy(orders[orderCount].phoneNumber, phoneNumber);

orderCount++;

printf("\nOrder placed successfully!\n");

}

void viewLastOrders() {

printf("\n======= Last Orders =======\n");

for (int i = orderCount - 1; i >= 0; i--) {

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

printf("Food Code: %d\n", orders[i].foodCode);

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

printf("Address: %s\n", orders[i].address);

printf("Phone Number: %s\n\n", orders[i].phoneNumber);

}

}

int main() {

int choice;

// Adding some sample food items

addFoodItem(1, "Pizza", 10.99);

addFoodItem(2, "Burger", 5.99);

addFoodItem(3, "Pasta", 8.99);

// Adding some sample employees

addEmployee(1, "John Doe", 2000);

addEmployee(2, "Jane Smith", 2500);

addEmployee(3, "David Brown", 3000);

do {

printMenu();

scanf("%d", &choice);

getchar(); // Consume the newline character

switch (choice) {

case 1:

placeOrder();

break;

case 2: {

int adminChoice;

printf("\n======= Employee Management =======\n");

printf("1. Add Employee Records\n");

printf("2. View all employee records\n");

printf("3. Search employee by ID or Name\n");

printf("4. Remove Employee records\n");

printf("Please enter your choice: ");

scanf("%d", &adminChoice);

getchar(); // Consume the newline character

switch (adminChoice) {

case 1: {

int id;

char name[MAX\_NAME\_LENGTH];

float salary;

printf("Enter Employee ID: ");

scanf("%d", &id);

getchar(); // Consume the newline character

printf("Enter Employee Name: ");

fgets(name, MAX\_NAME\_LENGTH, stdin);

name[strcspn(name, "\n")] = '\0'; // Remove trailing newline character

printf("Enter Salary: ");

scanf("%f", &salary);

getchar(); // Consume the newline character

addEmployee(id, name, salary);

break;

}

case 2:

printEmployeeList();

break;

case 3: {

int searchChoice;

printf("1. Search by Employee ID\n");

printf("2. Search by Employee Name\n");

printf("Please enter your choice: ");

scanf("%d", &searchChoice);

getchar(); // Consume the newline character

switch (searchChoice) {

case 1: {

int id;

printf("Enter Employee ID: ");

scanf("%d", &id);

getchar(); // Consume the newline character

Employee\* result = searchEmployee(id);

if (result != NULL) {

printf("\nEmployee found!\n");

printf("ID: %d \t Name: %s \t Salary: %.2f\n", result->id, result->name, result->salary);

} else {

printf("\nEmployee not found!\n");

}

break;

}

case 2: {

char name[MAX\_NAME\_LENGTH];

printf("Enter Employee Name: ");

fgets(name, MAX\_NAME\_LENGTH, stdin);

name[strcspn(name, "\n")] = '\0'; // Remove trailing newline character

Employee\* result = searchEmployeeByName(name);

if (result != NULL) {

printf("\nEmployee found!\n");

printf("ID: %d \t Name: %s \t Salary: %.2f\n", result->id, result->name, result->salary);

} else {

printf("\nEmployee not found!\n");

}

break;

}

default:

printf("\nInvalid choice!\n");

}

break;

}

case 4: {

int id;

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

scanf("%d", &id);

getchar(); // Consume the newline character

removeEmployee(id);

break;

}

default:

printf("\nInvalid choice!\n");

}

break;

}

case 3:

viewLastOrders();

break;

case 4:

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

break;

default:

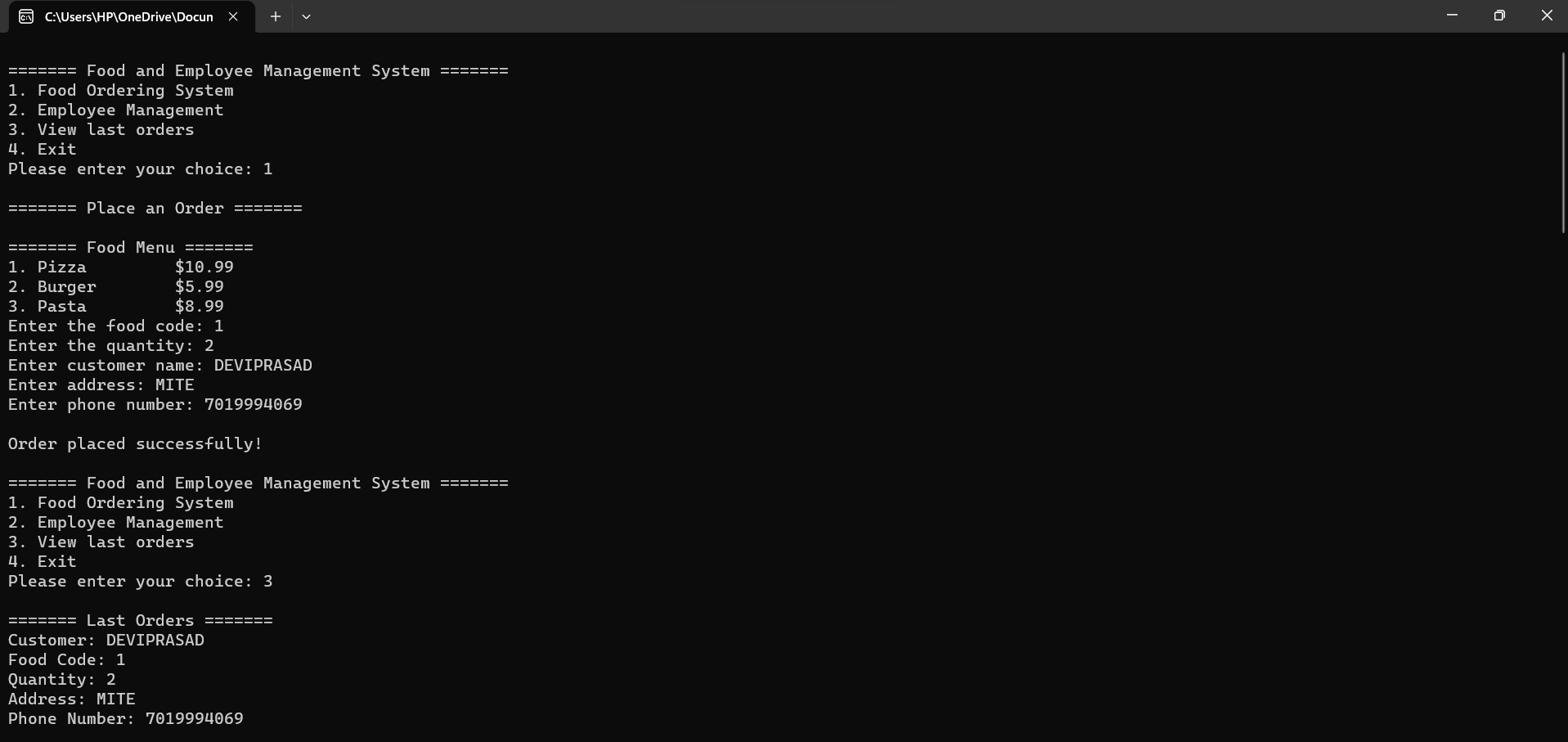
printf("\nInvalid choice!\n");

}

} while (choice != 4);

return 0;

}

**OUTPUT :**

# 

