N.B.K.R Institute of Science &Technology

*Vidyanagar-Tirupati*

**Food Delivery Queue System Simulation**

Course: Data Structures

Branch: Computer Science and Engineering

Section: F

Year : I

Semester: II

***Submitted to***:

Ashok Selva Kumar

Submitted by:

A.SANJANA (24KB1A0511)

A.NEEHARIKA (24KB1A0515)

B.HARSHITHA (24KB1A0566)

CH.SRIVANI (24KB1A05A3)



**Acknowledgement**

I take this opportunity to express my deep gratitude to everyone who has supported and guided me throughout this project. First and foremost, I thank my project guide, *Ashok Selva Kumar*, for their consistent encouragement, valuable feedback, and expert guidance that enabled me to carry out this project successfully. I also extend my heartfelt thanks to my department and the faculty members who have provided essential knowledge and support throughout the academic session. This project would not have been possible without the resources and knowledge I acquired during my coursework. I also want to thank my peers and friends who helped me debug and test the application and motivated me during difficult phases. Lastly, I thank my family for their emotional support and encouragement, without which the completion of this work would not have been possible.

**Certificate**

This is to certify that the project titled **“Food Delivery Queue System Simulation”** is a bona fide work completed by A.Sanjana (24KB1A0511), A.Neeharika (24KB1A0515), B.Harshitha (24KB1A0566), CH.Srivani (24KB1A05A3) in partial fulfilment of the requirements of the Data Structures under the supervision of Ashok Selva Kumar during the academic year 2024 - 2025.

**Project Supervisor:** **Head of Department:**

**Abstract:**

The Food Delivery Queue System Simulation is a C-program-based project aimed at modeling a simplified food ordering and delivery management system using the Queue data structure. This simulation reflects real-world food service operations where orders are placed, managed, and delivered in a first-come-first-served (FIFO) manner. Each order contains customer details like Customer ID, Food Name, Hotel Name, and Street Name. The user is offered four primary options: placing an order, processing (delivering) the order, displaying all pending orders, and exiting the program. The queue ensures that orders are handled in the order they arrive, reflecting fairness and orderliness. This project demonstrates how linear data structures, especially queues, can be utilized in practical applications. It also provides a good understanding of basic C programming concepts like arrays, structures, functions, and user input. Educationally, it helps learners connect theory with real-world use cases in food delivery systems.

**Introduction:**

In the modern era of digitization, the food delivery industry has evolved significantly with the help of software and applications. This project titled Food Delivery Queue System Simulation aims to simulate the backend process of a food delivery service using basic C programming and queue data structures. The system follows the First-In-First-Out (FIFO) mechanism, ensuring that food orders are processed in the exact sequence they were received. The main objective of the project is to demonstrate how queues function in a real-world scenario, like managing orders efficiently and avoiding conflicts or overlaps. It also emphasizes essential programming skills such as structured programming, modularity, function handling, and user interaction in a command-line interface. The project is developed using Code::Blocks IDE and is aimed at educational purposes for beginners to understand queue logic in a practical context. It provides a base for more complex systems like online food ordering platforms and logistics applications.

**Objective:**

The primary objectives of the *Food Delivery Queue System Simulation* project are:

* To understand and implement the concept of queue data structures using C.
* To simulate a real-world food ordering and delivery service with basic functionalities.
* To allow users to place orders, view pending orders, and process deliveries in order.
* To demonstrate the application of arrays and structures in managing complex data.
* To improve logical thinking and programming skills by building a functional simulation.
* To show how FIFO logic is used in practical queue management systems.
* To help beginners connect classroom data structure concepts to real-life applications.
* To practice input/output handling and user interaction in a console application.
* To serve as a learning base for future enhancements into dynamic queue systems, linked lists, or databases.

**System Requirements:**

***Software Requirements*:**

* **Operating System**: Windows/Linux/macOS
* **Programming Language**: C
* **IDE/Compiler**: Code::Blocks (with GCC compiler) or any other C-supported compiler like Turbo C++ or Visual Studio Code with MinGW.
* **Version Control**: GitHub (optional, for tracking changes and storing source code)

***Hardware Requirements:***

* **Processor**: Minimum Intel Core i3 or equivalent
* **RAM**: Minimum 4 GB
* **Storage**: At least 100 MB free for project files
* **Display**: Standard resolution monitor

**Literature Review:**

In recent years, food delivery systems have become a central focus of software development due to the rise in demand for fast and efficient services. Research into queueing theory, data structure applications, and logistics optimization has led to the development of robust systems used by real-world companies such as Zomato, Swiggy, and Uber Eats. Literature on data structures, especially queues, has been widely covered in books like “Data Structures Using C” by Reema Thareja and “Fundamentals of Data Structures” by Horowitz and Sahni. These texts emphasize how queues can be used in systems where order of processing is crucial. Additionally, online learning platforms such as GeeksforGeeks, Coursera, and HackerRank provide practical exposure to queue applications. This project is a simplified simulation inspired by such literature, where a static array is used to implement a queue structure to manage orders, mimicking the backend logic of real food ordering systems.

**Methodology:**

The methodology adopted in the *Food Delivery Queue System Simulation* project follows a structured and incremental software development approach. Initially, the requirements were gathered — understanding what functionalities a basic food delivery system must include: placing an order, viewing orders, and processing them in order. The core data structure selected was a queue, implemented using arrays in the C language, to ensure First-In-First-Out (FIFO) logic. The system design included defining a struct to represent an order with fields like customer ID, food name, hotel name, and street name. Each action (place, view, process) was implemented as a separate function to ensure modularity and clarity. The main loop handles the menu-driven user interface. Basic input validation and memory-safe operations like fgets() were used to prevent buffer overflows. Code was continuously tested at each stage. The Code::Blocks IDE was used for development and debugging. This method provided a clean, educational, and functional simulation of a food delivery queue system.

**Project Description:**

**Problem Statement:**

Manual food order management often leads to inefficiencies, delays, and difficulty tracking the sequence of deliveries, especially in high-demand scenarios.

**Proposed Solution:**

This project simulates an automated food ordering and delivery system using a queue-based structure in C, ensuring orders are handled in the exact order they are placed, with features like order search, cancellation, and timestamp tracking.

**Key Features:**

1. Queue-Based Order Management

Utilizes a First-In-First-Out (FIFO) queue structure to manage food orders, ensuring fair and orderly processing.

1. Detailed Order Information

Each order includes Customer ID, Food Name, Hotel Name, Street Name, Order Time, and Delivery Time for real-time tracking.

1. Search & Cancel Functionality

Allows users to search for orders by Customer ID and cancel them before delivery—adding flexibility and control.

1. Time Stamp Integration

Automatically records the Order Time and Delivery Time using time.h library functions, simulating real-world delivery tracking.

1. Interactive Menu-Driven Interface

Provides an easy-to-use console menu system for placing, processing, displaying, searching, and canceling orders.

**Flowchart/Algorithm:**

**Flowchart:**

Start → Menu → [Place/Process/View/Exit] → Actions → Back to Menu → End

**Algorithm Steps:**

1. Start the program.
2. Initialize the queue with front and rear as -1.
3. Display the main menu:
   * 1. Place Order
     2. Process Order
     3. Show All Orders
     4. Exit
4. On placing an order:
   * Increment customer ID.
   * Get user inputs (food, hotel, street).
   * Enqueue the new order at rear.
5. On processing an order:
   * Dequeue the order at front.
   * Display processed order details.
6. On showing orders:
   * Traverse the queue from front to rear.
   * Print each order.
7. Repeat steps 3-6 until user selects exit.
8. End the program.

[**Program Code**](https://onlinegdb.com/4vRWFfLzp)**:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define SIZE 100

typedef struct {

int customerId;

char foodName[50];

char hotelName[50];

char streetName[50];

time\_t orderTime;

time\_t deliveryTime;

} Order;

Order queue[SIZE];

int front = -1, rear = -1;

void printOrderDetails(Order order) {

printf("Customer ID: %d\n", order.customerId);

printf("Food: %s\n", order.foodName);

printf("Hotel: %s\n", order.hotelName);

printf("Street: %s\n", order.streetName);

printf("Order Time: %s", ctime(&order.orderTime));

if (order.deliveryTime != 0) {

printf("Delivery Time: %s", ctime(&order.deliveryTime));

}

else {

printf("Delivery Time: Not yet delivered\n");

}

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

}

void placeOrder(Order order) {

if (rear == SIZE - 1) {

printf("Queue is full. Cannot place more orders.\n");

return;

}

if (front == -1) front = 0;

rear++;

queue[rear] = order;

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

printOrderDetails(order);

void processOrder() {

if (front == -1 || front > rear) {

printf("\nNo orders to process.\n");

return;

}

queue[front].deliveryTime = time(NULL); // Set delivery time when order is processed

printf("\nOrder Delivered:\n");

printOrderDetails(queue[front]);

front++;

}

void showOrders() {

if (front == -1 || front > rear) {

printf("\nNo pending orders.\n");

return;

}

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

for (int i = front; i <= rear; i++) {

printf("Order %d:\n", i - front + 1);

printOrderDetails(queue[i]);

}

}

void searchOrderById(int customerId) {

if (front == -1 || front > rear) {

printf("\nNo orders to search.\n");

return;

}

for (int i = front; i <= rear; i++) {

if (queue[i].customerId == customerId) {

printf("\nOrder found:\n");

printOrderDetails(queue[i]);

return;

}

}

printf("\nOrder with Customer ID %d not found.\n", customerId);

}

void cancelOrder(int customerId) {

if (front == -1 || front > rear) {

printf("\nNo orders to cancel.\n");

return;

}

int i, found = 0;

for (i = front; i <= rear; i++) {

if (queue[i].customerId == customerId) {

found = 1;

break;

}

}

if (!found) {

printf("\nOrder with Customer ID %d not found.\n", customerId);

return;

}

// Shift elements to remove the canceled order

for (int j = i; j < rear; j++) {

queue[j] = queue[j + 1];

}

rear--;

printf("\nOrder with Customer ID %d has been canceled.\n", customerId);

}

int main() {

int choice;

int orderCount = 0;

while (1) {

printf("\n==== Online Food Delivery Queue System ====\n");

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

printf("2. Process Order\n");

printf("3. Show All Orders\n");

printf("4. Search Order by Customer ID\n");

printf("5. Cancel Order by Customer ID\n");

printf("6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // Clear newline

if (choice == 1) {

Order newOrder;

newOrder.customerId = ++orderCount;

printf("Enter food name: ");

fgets(newOrder.foodName, sizeof(newOrder.foodName), stdin);

newOrder.foodName[strcspn(newOrder.foodName, "\n")] = '\0';

printf("Enter hotel name: ");

fgets(newOrder.hotelName, sizeof(newOrder.hotelName), stdin);

newOrder.hotelName[strcspn(newOrder.hotelName, "\n")] = '\0';

printf("Enter street name: ");

fgets(newOrder.streetName, sizeof(newOrder.streetName), stdin);

newOrder.streetName[strcspn(newOrder.streetName, "\n")] = '\0';

newOrder.orderTime = time(NULL); // Record order time

newOrder.deliveryTime = 0; // Initial delivery time as 0 (not yet delivered)

placeOrder(newOrder);

} else if (choice == 2) {

processOrder();

} else if (choice == 3) {

showOrders();

}

else if (choice == 4) {

int customerId;

printf("\nEnter Customer ID to search: ");

scanf("%d", &customerId);

searchOrderById(customerId);

}

else if (choice == 5) {

int customerId;

printf("\nEnter Customer ID to cancel: ");

scanf("%d", &customerId);

cancelOrder(customerId);

}

else if (choice == 6) {

printf("\nExiting system. Thank you!\n");

break;

}

else {

printf("\nInvalid choice. Please try again.\n");

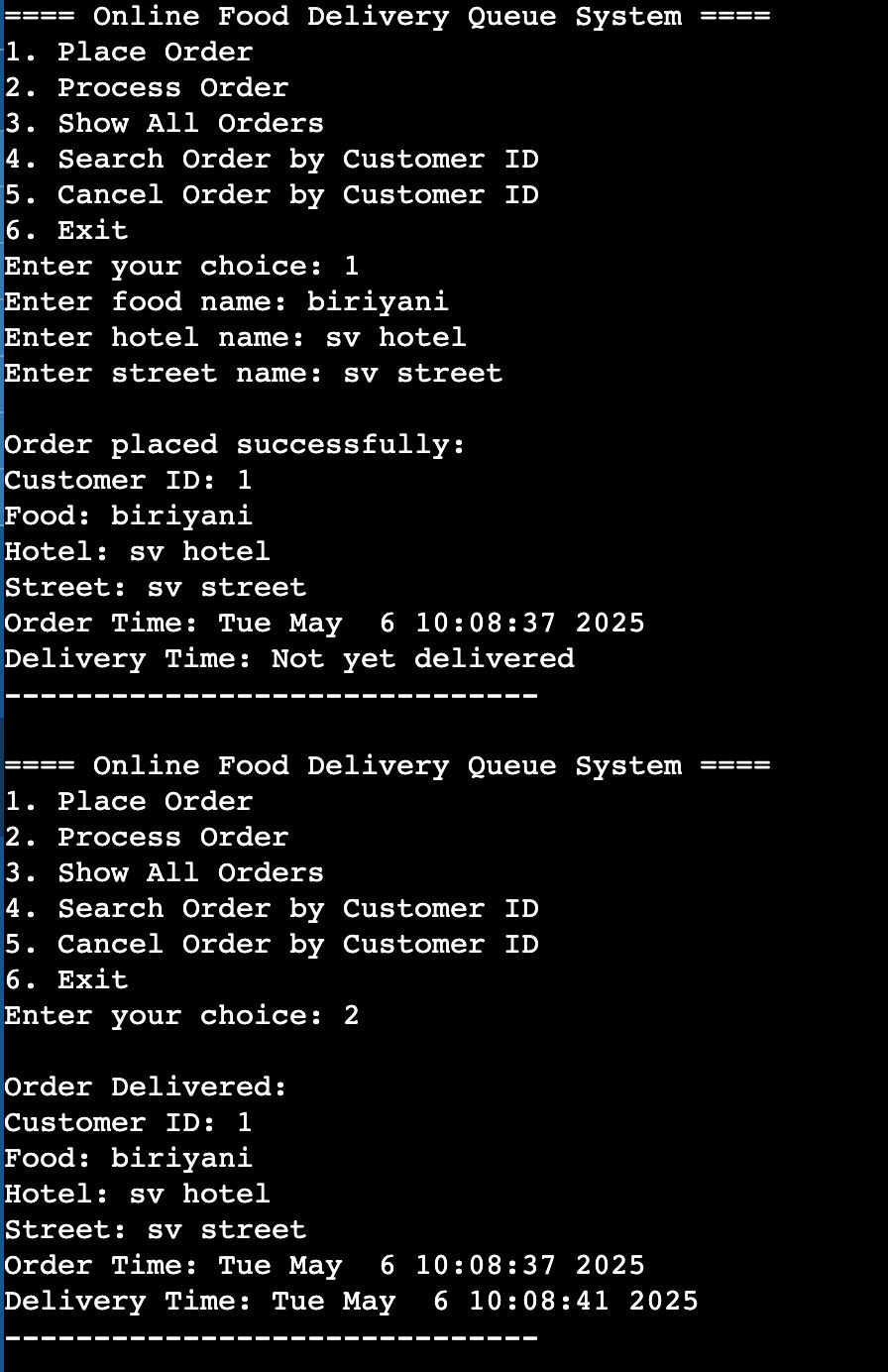
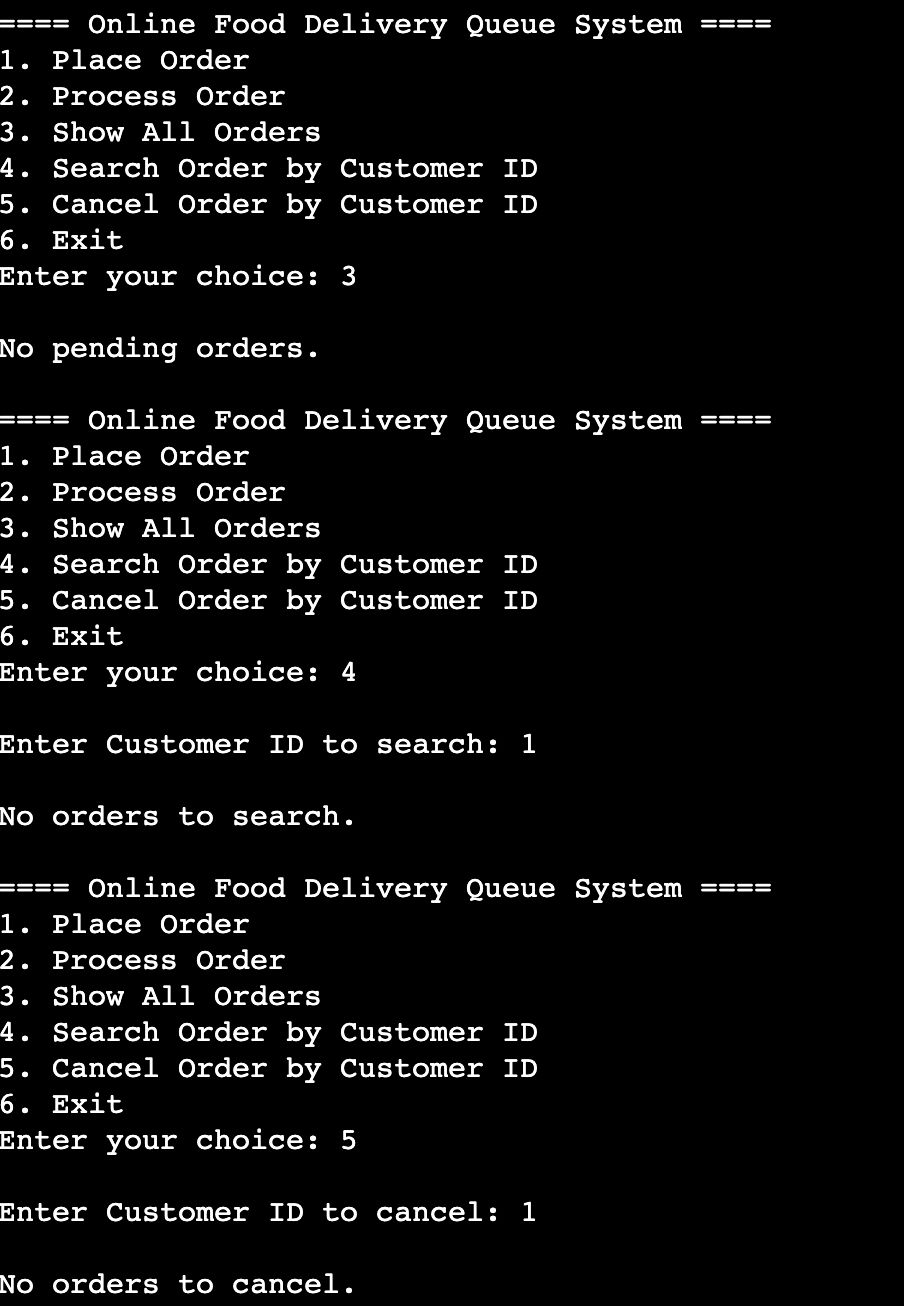
}

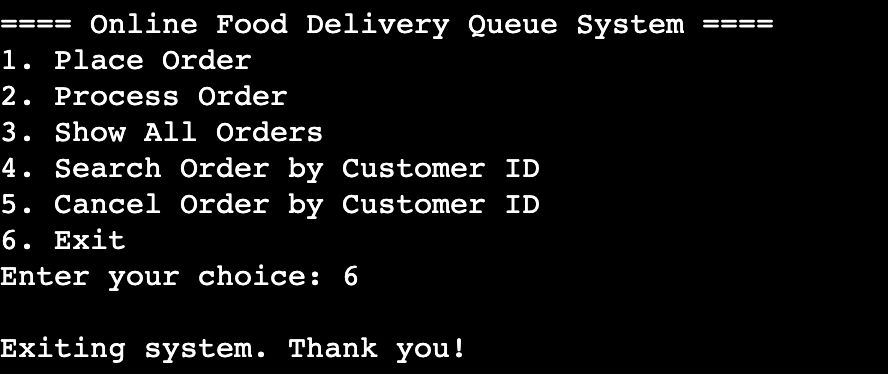
}

return 0;

}

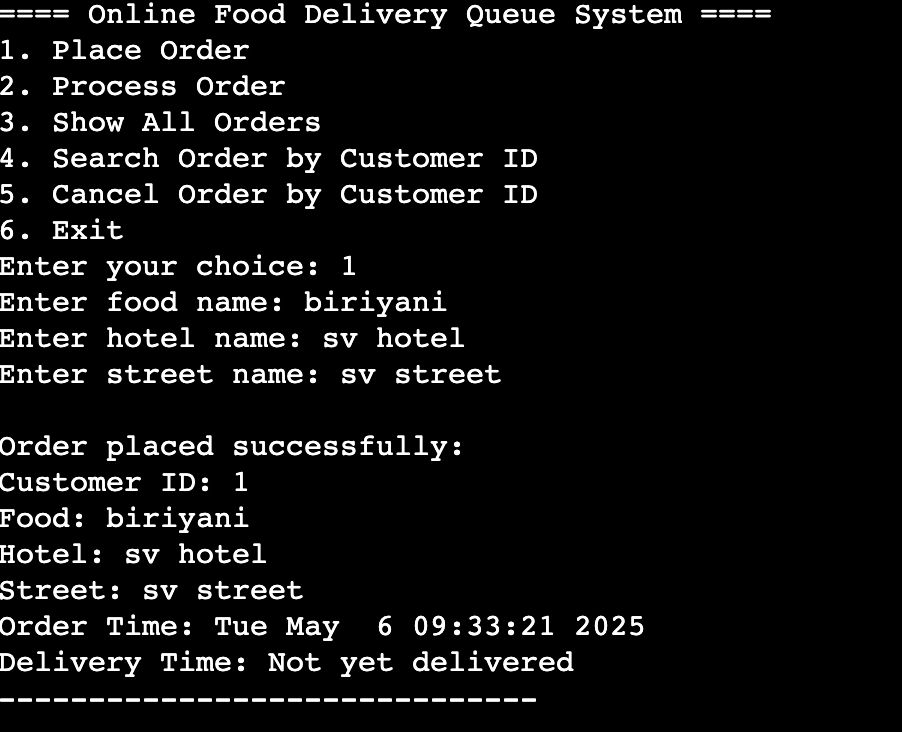
**Output screenshots:**

** **

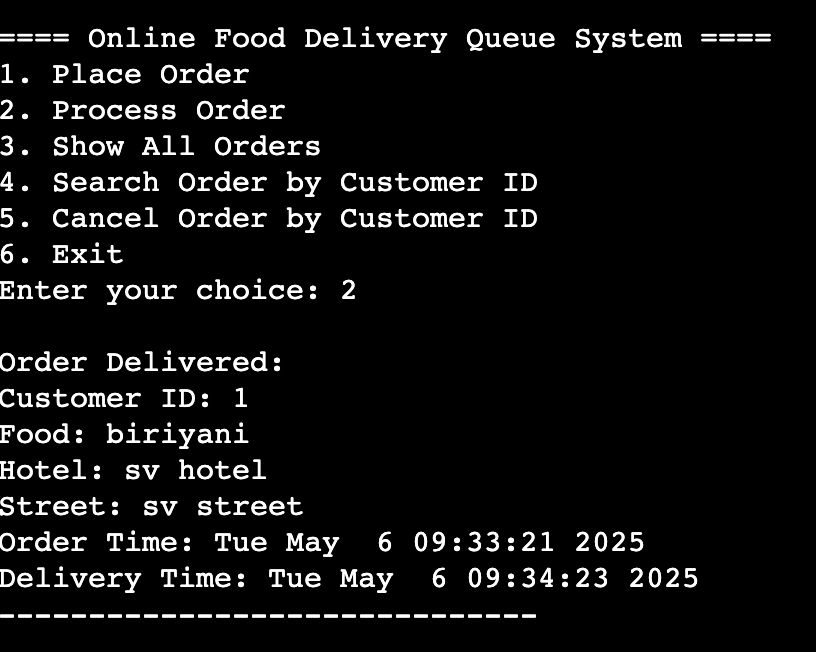
****

**Testing and validation:**

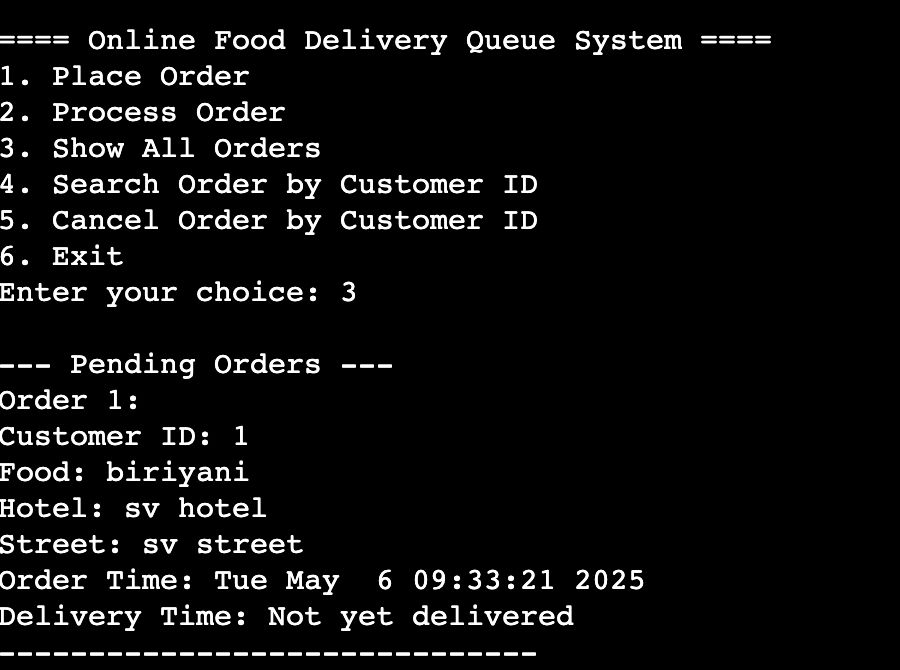
**Test 1:** Place Order



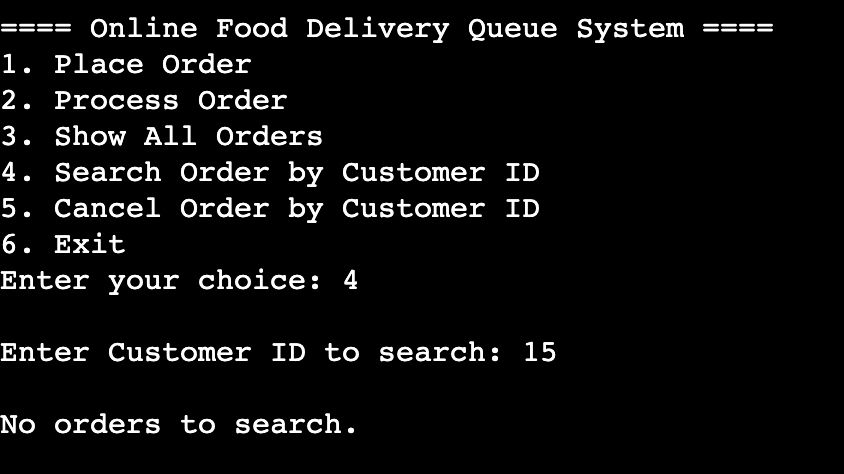
**Test 2**: Process Order



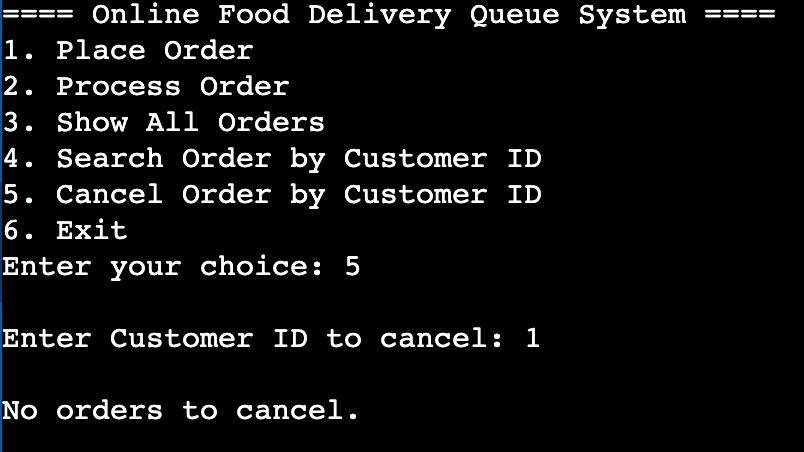
**Test 3:** Show all Orders



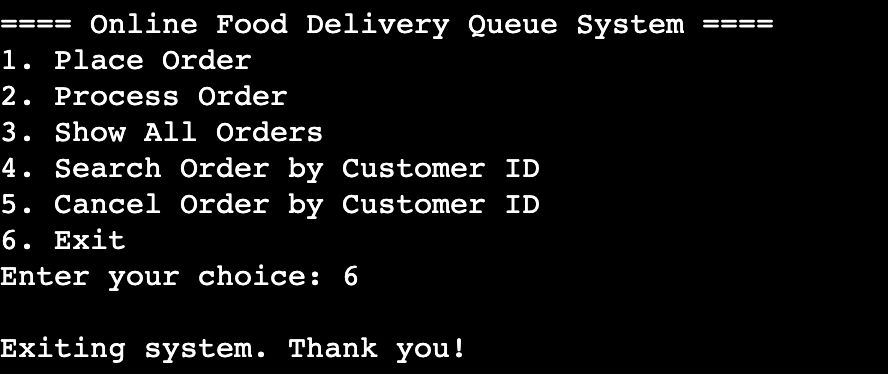
**Test 4**: Search order by customer ID



**Test 5**: Cancel order by Customer ID



**Test 6**: Exit



**Limitations:**

* **Static Queue Size**: The array has a fixed size (100 orders), limiting scalability.
* **No Persistent Storage**: Once the program exits, all data is lost. It does not use file storage.
* **No Error Logging**: There is no mechanism to log invalid inputs or system errors.
* **Limited User Interface**: It's console-based only; lacks graphical or web-based interaction.
* **No Real-Time Data Handling**: There's no multi-user or real-time processing capability.

**Future Enhancements**:

 **Dynamic Queue Implementation**: Using linked lists to avoid fixed size limitations.

 **File Storage Integration**: Saving order data to files for retrieval after program exit.

 **GUI Development**: Creating a graphical interface using tools like GTK, Qt, or web frameworks.

 **Multiple Order Types**: Supporting food categories, prices, and special instructions.

 **Sorting or Filtering**: Allow sorting by hotel or food type for better delivery optimization.

 **Admin Panel**: Adding features for admin-level actions like clearing all orders or managing history.

**Conclusion:**

The Food Delivery Queue System Simulation project successfully demonstrates how queues operate in real-life service-based systems. Through a simple and interactive console program, users are introduced to key concepts such as struct usage, queue logic, array handling, and menu-driven design in C programming. It bridges the gap between academic theory and practical application, offering a strong foundation for further study in data structures and system development. The system was thoroughly tested and validated, and although limited in scope, it fulfilled its educational purpose effectively. This project can serve as a base model for more advanced applications, and its future enhancements open opportunities for broader use cases.

**References:**

* Reema Thareja, Data Structures Using C, Oxford University Press
* Seymour Lipschutz, Data Structures, McGraw-Hill
* Horowitz and Sahni, Fundamentals of Data Structures
* GeeksforGeeks – <https://www.geeksforgeeks.org>
* TutorialsPoint – <https://www.tutorialspoint.com>
* CProgramming.com
* Stack Overflow