Government of Nepal
National Examination Board

Computer Project Report On

RESTAURANT ORDER MANAGEMENT SYSTEM

Using C-Programming

KATHMANDU MODEL SECONDARY SCHOOL

A Project Report Submitted To
The Information Technology Department
Kathmandu Model Secondary School
Bagbazar, Kathmandu

Submitted By
Prithav Jha
Symbol No.: 812830
Shift: Day
Faculty: Science
Class: XII
Section: D2
Roll No.: 23

Submitted On: Poush 04, 2082

Government of Nepal
National Examination Board

KATHMANDU MODEL
SECONDARY SCHOOL

Computer Project Report On

RESTAURANT ORDER MANAGEMENT SYSTEM

Using C-Programming

A Project Report Submitted For The Partial Fulfillment Of Final Examination To
The Information Technology Department
Kathmandu Model Secondary School
Bagbazar, Kathmandu

Assigned On:
Mangsir 05, 2082

Submitted On:
Poush 04, 2082

Submitted By:
Prithav Jha (Symbol No.: 812830)

Submitted To:
Shekhar Chandra Paudel

Certificate of Project Assignment Authorization

This is to certify that the project "**Restaurant Order Management System**", assigned on **5th Mangsir, 2082**, has been given to the following student of Grade XII Computer Science as part of the **NEB +2** curriculum requirements. The project is to be completed and submitted according to the guidelines provided by the **Department of Computer Science**.

This report must represent authentic work carried out by the students. The project demonstrates the students' understanding of programming concepts, system design, and development practices using the **C programming language**.

This authorization certificate must be included in the report as an official acknowledgment of the assigned project, to be kept after the cover page at the time of submission.

Student:

- Mr. Prithav Jha, XII - Science, Symbol No.: 812830

Date: 14 Mangsir, 2082

Department of Computer Science
Kathmandu Model Secondary School (KMSS)

Acknowledgement

Developing this project, "**Restaurant Order Management System**," has been a significant learning curve and a rewarding experience. It required consistent effort, logical thinking, and valuable guidance from my mentors. I would like to take this opportunity to express my sincere gratitude to everyone who supported me throughout this journey.

First and foremost, I would like to thank **Kathmandu Model Secondary School**, our respected Computer Science HOD, Mr. Saroj Poudel, along with all the faculty members, for providing a sound academic environment and the necessary resources to complete this project successfully.

I am also grateful to **my friends** for their constructive feedback and moral support. A special thanks to **my parents** for their endless love and encouragement, which kept me motivated to finish this task.

Finally, I would like to express my gratitude to the **National Examination Board (NEB)** for designing a curriculum that includes practical project work, giving me the opportunity to enhance my skills in **C programming** and **system development**.

Prithav Jha
Symbol No.: 812830
Shift: Day
Faculty: Science
Grade: XII
Section: D2
Roll No.: 23

Abstract

The project entitled **"Restaurant Order Management System"** is a console-based application designed to digitize and automate the manual operations of a restaurant. In traditional settings, managing menus, recording orders, and calculating bills manually is time-consuming and prone to calculation errors. This system aims to solve these problems by providing an efficient, computerized solution.

The system is developed using the **C programming language**, applying core concepts such as **File Handling**, **Structures**, **Arrays**, and **Pointers** to manage data effectively. It features a dual-user interface: an **Admin Mode** for the restaurant manager to add, update, and delete menu items and generate sales reports; and a **Customer Mode** that allows users to view the menu, place orders, and receive an automatically calculated bill.

All data, including menu items and order history, is stored in external text files (`menu.txt` and `orders.txt`) to ensure data persistence. Through this project, I have successfully demonstrated how **C programming** can be **applied** to create a **practical**, **user-friendly** system that **simplifies** daily **business operations** and **improves service efficiency**.

Table of Contents

Certificate of Project Assignment Authorization I
Acknowledgement II
Abstract III
Table of Contents IV
List of Figures V
List of Tables VI
Acronyms, Abbreviations & Glossary of Terms VII
Chapter 1: Introduction 1
- 1.1 Background 1
- 1.2 Problem Statement 1
- 1.3 Objectives of the Project 2
- 1.4 Scope and Limitations of the System 2
  - 1.4.1 Scope 2
  - 1.4.2 Limitations 2
- 1.5 Organization of the Report 2

Chapter 2: Literature Review 3
- 2.1 Comparison of Manual vs. Computerised Systems 3
- 2.2 Software Development Life Cycle (SDLC) Model 3

Chapter 3: System Analysis & Design 5
- 3.1 Feasibility Study 5
  - 3.1.1 Technical Feasibility 5
  - 3.1.2 Operational Feasibility 5
  - 3.1.3 Economic Feasibility 5
- 3.2 System Requirements 5
  - 3.2.1 Hardware Requirements 5
  - 3.2.2 Software Requirements 5
- 3.3 Data Design (Data Structures) 5
  - 3.3.1 struct MenuItem 6
  - 3.3.2 struct OrderItem 6
  - 3.3.3 struct Order 6
- 3.4 File Design and Format 7
  - 3.4.1 menu.txt File Format 7
  - 3.4.2 orders.txt File Format 7
- 3.5 High-Level Data Flow Diagram (DFD) 7
- 3.6 Algorithmic Design 7
  - 3.6.1 Main System Flow 7
  - 3.6.2 Menu Item Management (Add, Update, Delete) 8
  - 3.6.3 Customer Order Placement 8
  - 3.6.4 Sales Report Generation 9

Chapter 4: Implementation & Testing 10
- 4.1 Implementation Environment 10
- 4.2 Source Code 10
- 4.3 Testing 31
  - 4.3.1 Unit Testing 31
  - 4.3.2 Test Cases 31

Chapter 5: Conclusion 33
- 5.1 Project Summary and Conclusion 33
- 5.2 Difficulties Encountered 33
- 5.3 Future Enhancements 33

Bibliography 35
User Manual & Snapshots 36
Team Contribution 41

List of Figures

Figure-1: Organization of the Report …………………………………………………………………. 2
Figure-2: Waterfall Model …………………………………………………………………………... 3
Figure-3: Navigate, Compile, Run ………………………………………………………………. 36
Figure-4: Main Menu ……………………………………………………………………………..... 36
Figure-5: Admin Login …………………………………………………………………………… 36
Figure-6: Admin Panel …………………………………………………………………………… 36
Figure-7: View Menu (Admin) …………………………………………………………………… 36
Figure-8: Add Menu Item ………………………………………………………………………….. 37
Figure-9: Update Menu Item ………………………………………………………………….… 37
Figure-10: Delete Menu Item ……………………………………………………………………… 37
Figure-11: View All Orders ……………………………………………………………………… 37
Figure-12: View Orders by Status ……………………………………………………………….. 38
Figure-13: Sales Report Generation ……………………………………………………………….. 38
Figure-14: Admin Logout ……………………………………………………………………………. 38
Figure-15: Customer Mode ………………………………………………………………………… 39
Figure-16: View Menu (Customer) ………………………………………………………………… 39
Figure-17: View Menu by Category ………………………………………………………………… 39
Figure-18: Place Order ………………………………………………………………………………. 40
Figure-19: Bill Calculation …………………………………………………………………………. 40
Figure-20: Greeting after Exit ……………………………………………………………………… 40

List of Tables

Table-1: Manual vs. Computerized Systems ……………….……………………………..……. 3
Table-2: Structure Fields for MenuItem …………………………………………………….……… 6
Table-3: Structure Fields for OrderItem ……….………………………………………….…. 6
Table-4: Structure Fields for Order ………………………………………………….………… 6
Table-5: Main System Flow ……………………………………..………………..……… 7
Table-6: Menu Item Management ……………...………………………………...…………... 8
Table-7: Customer Order Flow ……………………………………...……………………… 8
Table-8: Sales Report Flow ………………………...…………...………………………….. 9
Table-9: Test Results …………………………………………..…………………………… 31

Acronyms, Abbreviations & Glossary of Terms

C : A general-purpose programming language developed by Dennis Ritchie, used for the system's implementation.
IDE : Integrated Development Environment (Software that provides comprehensive facilities to computer programmers for software development, e.g., Turbo C/C++).
I/O : Input / Output
EOF : End of File
ASCII : American Standard Code for Information Interchange
CPU : Central Processing Unit
RAM : Random Access Memory
ALU : Arithmetic Logic Unit
CLI : Command Line Interface
LIB : Library (Collection of Functions or Routines)
SRC : Source
OBJ : Object File
EXE : Executable File
FN : Function
VAR : Variable
NEB : National Examination Board
KMSS : Kathmandu Model Secondary School
HOD : Head of Department
SDLC : Software Development Life Cycle (A process used by the software industry to design, develop, and test high-quality software).
DFD : Data Flow Diagram (A visual representation of the flow of data through an information system).
GCC : GNU Compiler Collection (A standard compiler system used to translate source code into executable programs).
DBMS : Database Management System (Software for storing and retrieving users' data while considering appropriate security measures, e.g., MySQL, PostgreSQL).
POS : Point of Sale (The time and place where a retail transaction is completed).
Console-based Application : A program designed to be used via a text-only interface rather than a graphical user interface (GUI).
File Handling : A mechanism in C programming used to store data (like menus and orders) in external text files (`.txt`) so data is not lost when the program closes.
Struct (Structure) : A user-defined data type in C (e.g., `struct MenuItem`) that allows combining different data types, like a price (float) and a name (string), into a single record.
Unit Testing : A software testing method where individual functions or components of the software are tested to verify they work correctly.
Waterfall Model : A linear approach to software development where each phase (Analysis, Design, Implementation) must be completed before the next begins.

Chapter 1: Introduction

1.1 Background

In the rapidly evolving service industry, particularly the culinary sector, efficiency and accuracy are paramount to customer satisfaction and business success. Traditional restaurant operations heavily rely on manual processes—taking orders using notepads, calculating bills with calculators, and managing inventory on paper. This manual approach is inherently prone to errors such as incorrect order taking, billing discrepancies, and slow service, which directly impacts the dining experience.

The **Restaurant Order Management System** project addresses these challenges by migrating core operational tasks to a computerized platform. Developed as a console-based application using the C programming language, this system automates order processing, menu management, and billing, providing a fast, reliable, and error-free solution suitable for small to medium-sized restaurants.

1.2 Problem Statement

Modern restaurant operations, particularly those relying on traditional, manual methods, face several critical challenges that impede efficiency, profitability, and customer satisfaction. The core issues that this project aims to resolve are:

1. **Billing Inaccuracy and Slowness:** Calculating bills manually, especially during peak hours, is slow and highly susceptible to human errors in addition or subtraction, leading to lost revenue or customer disputes.
2. **Inefficient Order Management:** Paper-based order taking can lead to miscommunication between the server and the kitchen, resulting in incorrect food preparation, wastage, and delays in service delivery.
3. **Lack of Data Insight:** Without a computerized system, restaurant managers cannot easily track sales history, identify popular items, or quickly generate daily or monthly sales reports, making financial analysis and decision-making difficult.
4. **Menu Management Difficulty:** Updating prices, marking items as unavailable, or adding new dishes requires manual correction or reprinting of physical menus, which is costly and time-consuming.

The central problem, therefore, is the reliance on error-prone and inefficient manual processes for order management and billing. This project seeks to replace these manual processes with a simple, robust, and accurate C-language system to streamline restaurant operations.

1.3 Objectives of the Project

The primary goal of this project is to create a functional and reliable system. The specific objectives are:

- To develop a user-friendly, console-based interface for both **Admin** (Manager) and **Customer** modes.
- To enable the dynamic creation, updating, and deletion of menu items by the administrator.
- To implement a robust **file handling** mechanism to ensure data persistence for both the menu (`menu.txt`) and order history (`orders.txt`).
- To accurately process customer orders, automatically calculate the total bill, including any applicable service charges or taxes, and generate a clear bill receipt.
- To provide the administrator with reporting features, specifically to view order history and generate sales reports.

1.4 Scope and Limitations of the System

1.4.1 Scope

The system is designed to provide comprehensive management for the core functions of a restaurant's front-end operations. Its scope covers:

- **Menu Management:** Storing and retrieving detailed menu item information (ID, Name, Category, Price).
- **Order Processing:** Accepting multiple items per order and calculating item subtotals.
- **Transaction Management:** Generating unique Order IDs and recording transaction details (customer name, date, time, total amount).

1.4.2 Limitations

As a proof-of-concept developed using C programming, the system has the following limitations:

- **Platform Dependency:** It is a console-based application and does not have a Graphical User Interface (GUI).
- **Data Storage:** It uses sequential file handling (text files) instead of a professional database management system (DBMS), limiting its capacity for handling extremely large volumes of data quickly.
- **Networking:** It is a standalone system and does not support multiple terminals or network-based ordering.
- **Inventory:** It does not integrate with back-end inventory stock management.

We can address these limitations and improve this system in its future versions.

1.5 Organization of the Report

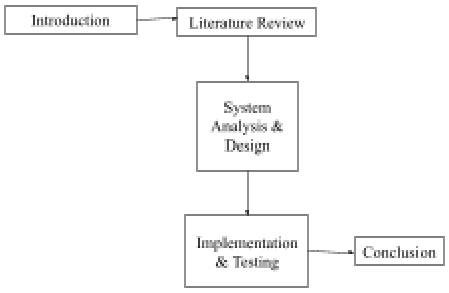


Figure-1: *Organization of the Report*

Chapter 2: Literature Review

The Literature Review involves examining existing concepts, theories, and studies relevant to the project. For a simple C-based system, this chapter focuses on the comparison between manual and computerised methods and the software development model used.

2.1 Comparison of Manual vs. Computerised Systems

Table-1: *Manual vs. Computerized Systems*

Feature	**Manual System (Paper-Based)**	**Computerised System (This Project)**
Order Taking	Hand-written notes; prone to illegible handwriting errors.	Input validated digitally; menu items selected by ID, minimizing errors.
Billing	Calculations done manually or with a calculator; high chance of arithmetical errors.	Automatic calculation via programmed functions; 100% accuracy.
Reporting	Time-consuming to compile sales reports from scattered papers.	Instantaneous generation of sales reports and order history from stored files.
Menu Updates	Requires reprinting or manual crossing out on physical menus.	Dynamic updates via Admin Mode; changes reflected instantly.
Storage	Requires physical storage space for old order receipts.	Data stored digitally on disk (orders.txt), requiring minimal physical space.

2.2 Software Development Life Cycle (SDLC) Model

The **Waterfall Model** was chosen as the methodological framework for the development of the Restaurant Order Management System. This model is ideal for small to medium-sized projects where requirements are clearly defined and unlikely to change, which perfectly describes this academic project.

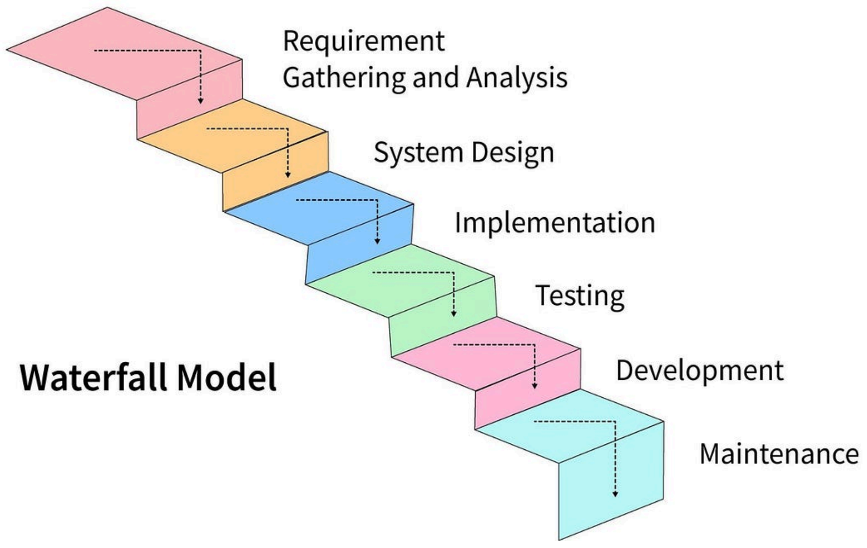


Figure-2: *Waterfall Model*

The key phases followed sequentially were:

1. **Requirement Gathering:** Defining user roles (Admin/Customer) and core functionalities (Menu, Order, Billing).
2. **System Design:** Designing the data structures (`struct MenuItem`, `struct Order`) and file layouts (`menu.txt`, `orders.txt`).
3. **Implementation (Coding):** Writing the C code (`restaurant_system.c`) for all functions.
4. **Testing:** Testing each module (login, order placement, billing) for correctness.
5. **Deployment & Maintenance:** Compiling the final report and preparing the system for presentation.

Chapter 3: System Analysis & Design

3.1 Feasibility Study

A feasibility study was conducted to evaluate the viability of the proposed system across three key dimensions: technical, operational, and economic. The findings strongly support the continuation of the project.

3.1.1 Technical Feasibility

The system is entirely feasible from a technical standpoint. It is developed using the standard C programming language, utilizing built-in libraries (`stdio.h`, `stdlib.h`, `string.h`, `time.h`). The core functionalities rely on sequential file handling and structured programming, which do not require specialized hardware, proprietary software, or complex networking infrastructure.

3.1.2 Operational Feasibility

Operationally, the system is highly feasible. It addresses the existing pain points identified in the problem statement by replacing error-prone manual ordering and billing processes with a simple, menu-driven interface. The Admin Mode is secured by a basic login (`admin/admin123`), and the Customer Mode is intuitive, making the system easy for staff and customers to adopt with minimal training.

3.1.3 Economic Feasibility

The economic feasibility is excellent. Since the project uses open-source tools (C compiler) and operates on standard computer hardware, the developmental and operational costs are negligible. The economic return is realized through reduced billing errors and improved staff efficiency, which contribute directly to increased profitability.

3.2 System Requirements

3.2.1 Hardware Requirements

- Standard Personal Computer (PC) or Laptop.
- Minimum 1 GB RAM (Low memory usage).
- Keyboard and Display (for console interaction).
- A printer (optional, for printing physical bills/reports).

3.2.2 Software Requirements

- **Operating System:** Windows, macOS, or Linux.
- **Programming Language:** C.
- **Compiler:** GCC (GNU Compiler Collection), Turbo C, or similar C compiler.
- **Text Editor/IDE:** Visual Studio Code or Embarcadero Dev C++ or Turbo C or any other text editor/IDE.

3.3 Data Design (Data Structures)

The system relies on three primary structures, defined in `restaurant_system.c`, to manage and link the data required for menu items and customer orders:

3.3.1 `struct MenuItem`

This structure is used to hold the static details of a single item offered by the restaurant.

Table-2: *Structure Fields for MenuItem*

Field Name	Data Type	Description
`item_id`	`int`	Unique identifier for the menu item.
`item_name`	`char[50]`	Name of the dish (e.g., "Momo (Veg)").
`category`	`char[30]`	Item category (e.g., "Appetizer," "Main Course").
`price`	`float`	Selling price of the item.
`available`	`int`	Status (1 for Available, 0 for Not Available).

3.3.2 `struct OrderItem`

This structure is used to capture the details of a specific item **within** a customer's current order.

Table-3: *Structure Fields for OrderItem*

Field Name	Data Type	Description
`item_id`	`int`	ID of the ordered item.
`item_name`	`char[50]`	Name of the ordered item.
`price`	`float`	Price per unit at the time of ordering.
`quantity`	`int`	Number of units ordered by the customer.
`subtotal`	`float`	Calculated price (price * quantity).

3.3.3 `struct Order`

This is the main transaction structure, holding the header information and all ordered items for one customer transaction.

Table-4: *Structure Fields for Order*

Field Name	Data Type	Description
`order_id`	`int`	Unique transaction ID (Starts at 1001).
`customer_name`	`char[50]`	Name of the person placing the order.
`date`	`char[40]`	Date and time the order was placed.
`items`	`struct OrderItem[20]`	Array to store up to 20 individual ordered items.
`item_count`	`int`	Actual number of items in the items array.
`total_amount`	`float`	Final calculated bill amount.
`status`	`char[20]`	Status of the order ("Pending," "Completed," or "Cancelled").

3.4 File Design and Format

The system uses text-based file handling to ensure data persistence, defining three critical file paths:

- `MENU_FILE: "menu.txt"`
- `ORDER_FILE: "orders.txt"`
- `TEMP_FILE: "temp.txt"` (Used temporarily for update/delete operations)

3.4.1 `menu.txt` File Format

This file stores the menu items, with each field separated by a pipe (`|`) delimiter on a single line. This format is used for efficient reading and updating of the `MenuItem` structure.

Record Format: `[item_id] | [item_name] | [category] | [price] | [available]`

Example Record: `103|Chowmein|Main Course|120.00|1`

3.4.2 `orders.txt` File Format

This file stores the order history. To link the main order details to its multiple items, a two-line record structure is used: an **Order Header** line followed by **Item Detail** lines.

A. Order Header Line: (Stores `struct Order` main fields) `[order_id] | [customer_name] | [date] | [item_count] | [total_amount] | [status]`

B. Item Detail Line(s): (Stores `struct OrderItem` fields, prefixed with "ITEM|") `ITEM | [item_id] | [item_name] | [price] | [quantity] | [subtotal]`

3.5 High-Level Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) provides a graphical representation of the information flow within the system. The Level 0 DFD, or Context Diagram, shows the system as a single process, interacting with its external entities and data stores.

The key flow demonstrates how the **Customer** places an order, which references the **Menu Data Store** (`menu.txt`), and how the final transaction is processed and stored in the **Orders Data Store** (`orders.txt`), which is later accessed by the **Administrator** for reporting.

3.6 Algorithmic Design

The core functionality of the Restaurant Order Management System is broken down into modular functions. The following algorithms detail the step-by-step logic for the main system operations.

3.6.1 Main System Flow

The system operates based on a central main menu (`mainMenu()`), which directs control flow to either the secure Administrator interface or the public Customer interface.

Table-5: *Main System Flow*

Step	Process	Description
1	Start	Call `initializeMenuFile()` to ensure `menu.txt` exists with default items.
2	Display	Display `mainMenu()` options: Admin Login, Customer Mode, Exit.
3	Input	Read user choice.
4	Process	If the choice is 1, call `adminLogin()`. If successful, call `adminMenu()`. If the

		choice is 2, call `customerMenu()`. If choice is 3, terminate the program.
5	Loop	Return to Step 2 until the program is explicitly exited.

3.6.2 Menu Item Management (Add, Update, Delete)

All menu item management is executed via file handling on the `menu.txt` data store. Updates and Deletions use the temporary file mechanism to maintain data integrity.

Table-6: *Menu Item Management*

Function	Process Description
Add Menu Item	**1.** Open `menu.txt` in append mode ("a"). **2.** Prompt Admin for new Item ID, Name, Category, and Price. **3. Validation:** Scan `menu.txt` to ensure the Item ID is unique. If not, re-prompt. **4.** Write the new record (`ID
Update Menu Item	**1.** Prompt Admin for the ID to modify. **2.** Open `menu.txt` in read mode (`"r"`) and `temp.txt` in write mode (`"w"`). **3.** Loop through records in menu.txt. **4.** If the current record's ID matches the target ID: prompt for new values (Name, Price, etc.). **5.** Write the **modified** record to `temp.txt`. **6.** If the ID does **not** match, write the **original** record to `temp.txt`. **7.** Close both files. **8. File Swap:** Use `remove(MENU_FILE)` and `rename(TEMP_FILE, MENU_FILE)` to replace the old file with the new one.
Delete Menu Item	**1.** Prompt Admin for the ID to delete and confirm. **2.** Open `menu.txt` in read mode (`"r"`) and `temp.txt` in write mode (`"w"`). **3.** Loop through records in `menu.txt`. **4.** If the current record's ID matches the target ID, skip writing it to `temp.txt` (effectively deleting it). **5.** If the ID does not match, write the original record to `temp.txt`. **6.** Close both files. **7. File Swap:** Use `remove(MENU_FILE)` and `rename(TEMP_FILE, MENU_FILE)`.

3.6.3 Customer Order Placement

The `placeOrder()` function manages the entire transaction from item selection to final bill generation and record keeping.

Table-7: *Customer Order Flow*

Step	Process	Description
1	**Initialization**	Call `generateOrderID()`. Initialize a `struct Order` (e.g., `currentOrder`) with the new ID, `item_count = 0`, and `status = "Pending"`. Capture current date/time.
2	**Customer Details**	Prompt and read the customer's name.
3	**Item Selection**	**Loop:** Display the menu. Prompt customer for Item ID and Quantity.
4	**Item Lookup**	Search `menu.txt` for the Item ID. If found and available: **a.** Calculate `subtotal = price * quantity`. **b.** Populate a `struct OrderItem`. **c.** Store `struct OrderItem` in `currentOrder.items[]`. **d.** Increment `currentOrder.item_count`.
5	**Loop**	Repeat loop until the customer selects 'n' or the item limit is reached.

6	**Billing**	Call `calculateBill(&currentOrder)` to sum all subtotals and set `currentOrder.total_amount`.
7	**Bill Display**	Call `displayOrder(currentOrder)` to show the final bill summary to the customer.
8	**Record Save**	Open `orders.txt` in append mode (`"a"`). Write the **Order Header** line, followed by a separate **Item Detail** line for each ordered item.
9	**End**	Close file and confirm order placement.

3.6.4 Sales Report Generation

The `generateSalesReport()` function reads the `orders.txt` file and aggregates the data for financial and performance analysis.

Table-8: *Sales Report Flow*

Step	Process	Description
1	**Initialization**	Initialize counters for `total_orders`, `completed_orders`, `total_revenue`, and an array of `ItemSales` structs.
2	**Read Data**	Open `orders.txt` in read mode (`"r"`). Loop through the file, reading the Order Header line first, then the corresponding number of Item Detail lines.
3	**Order Tally**	For each Order Header read: a. Increment `total_orders` and `total_revenue`. b. Tally status (Pending/Completed/Cancelled). c. If status is "Completed," add `total_amount` to `completed_revenue`.
4	**Item Tally**	For each Item Detail read **within a Completed order**: **a.** Check if the Item ID is already in the `ItemSales` array. **b.** If found, update its `total_quantity` and `total_sales`. **c.** If not found, add a new entry to the array.
5	**Output**	Display summary statistics (Total Orders, Revenue Breakdown).
6	**Sort**	Sort the ItemSales array by `total_quantity` (using a simple sort like Bubble Sort).
7	**Display**	Display the **Item-wise Sales Report** and the **Top 5 Best Sellers** from the sorted list.

Chapter 4: Implementation & Testing

This chapter details the specific environment used for coding and compiling the system, presents the complete source code, and documents the testing process undertaken to ensure the system meets all defined requirements.

4.1 Implementation Environment

The **Restaurant Order Management System** was implemented using a Structured Programming paradigm within a standard development environment.

- **Programming Language: C Language**. C was chosen for its efficiency, direct control over memory, and suitability for console-based applications and file handling.
- **Development Tools:** The code was written and compiled using a standard **C Compiler** (such as GCC or Turbo C/C++ IDE) on a Windows or Linux operating system.
- **Key Libraries Used:** The program utilizes standard C libraries for essential operations:
  - `stdio.h`: For standard input/output operations (e.g., `printf`, `scanf`, file operations).
  - `stdlib.h`: For general utility functions (e.g., `system` calls for clearing the screen, `exit`).
  - `string.h`: For string manipulation (e.g., `strcmp`, `strcpy`, `strcspn`).
  - `time.h`: For generating time and date stamps for orders.

4.2 Source Code

The complete source code for the **Restaurant Order Management System** is provided below. The program is implemented as a single file, `restaurant_system.c`, containing all structure definitions, function prototypes, and implementation logic.

Note: *As per the project guidelines, the following source code is presented in* ***Consolas*** *font,* ***10pt*** *size, with a line spacing of* ***0.8****.*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

// File paths
#define MENU_FILE "menu.txt"
#define ORDER_FILE "orders.txt"
#define TEMP_FILE "temp.txt"

// Structure definitions
struct MenuItem {
 int item_id;
 char item_name[50];
 char category[30];
 float price;
 int available; // 1 = available, 0 = not available
};

struct OrderItem {
 int item_id;
 char item_name[50];
 float price;
 int quantity;
 float subtotal;
};

```

```c
struct Order {
 int order_id;
 char customer_name[50];
 char date[40];
 struct OrderItem items[20];
 int item_count;
 float total_amount;
 char status[20]; // Pending, Completed, Cancelled
};

// Function prototypes
void mainMenu();
int adminLogin();
void adminMenu();
void customerMenu();

// Admin functions
void addMenuItem();
void viewMenu();
void updateMenuItem();
void deleteMenuItem();
void viewAllOrders();
void viewOrdersByStatus();
void updateOrderStatus();
void generateSalesReport();

// Customer functions
void placeOrder();
void viewCustomerMenu();
void viewMenuByCategory();

// Utility functions
void calculateBill(struct Order *order);
void displayOrder(struct Order order);
void pressEnterToContinue();
void clearScreen();
int generateOrderID();
void initializeMenuFile();

// Global variables
struct Order currentOrder;

int main() {
 int choice;

 // Initialize menu file if it doesn't exist
 initializeMenuFile();

 while (1) {
   clearScreen();
   mainMenu();

   printf("\nEnter your choice: ");
   if (scanf("%d", &choice) != 1) {
     while (getchar() != '\n')
       ;
     printf("\nInvalid input! Please enter a number.\n");
     pressEnterToContinue();
     continue;
   }
   while (getchar() != '\n')
     ;

   switch (choice) {
   case 1:
     if (adminLogin()) {
       adminMenu();
     }
     break;
   case 2:
     customerMenu();
     break;
   case 3:
     clearScreen();
     printf("\n========================================\n");
     printf("   Thank you for using our system!\n");
     printf("========================================\n\n");
     exit(0);
   default:
     printf("\nInvalid choice! Please try again.\n");
     pressEnterToContinue();
   }
 }

 return 0;
}

// ==================== MAIN MENU ====================
void mainMenu() {
 printf("\n========================================\n");
 printf("  RESTAURANT ORDER MANAGEMENT SYSTEM\n");
 printf("========================================\n");
 printf("\n         MAIN MENU\n");
 printf("----------------------------------------\n");
 printf("  1. Admin Login\n");
 printf("  2. Customer Mode\n");
 printf("  3. Exit\n");
 printf("----------------------------------------\n");
}

// ==================== ADMIN LOGIN ====================
int adminLogin() {
 char username[30], password[30];
 int attempts = 0;

 clearScreen();
 printf("\n========================================\n");
 printf("           ADMIN LOGIN\n");
 printf("========================================\n");

 while (attempts < 3) {
   printf("\nEnter Username: ");
   scanf("%s", username);
   printf("Enter Password: ");
   scanf("%s", password);
   while (getchar() != '\n')
     ;

   if (strcmp(username, "admin") == 0 && strcmp(password, "admin123") == 0) {
     printf("\n✓ Login Successful!\n");
     printf("Welcome, Administrator!\n");
     pressEnterToContinue();
     return 1;
   } else {
     attempts++;
     printf("\n✗ Invalid credentials! ");
     if (attempts < 3) {
       printf("Attempts remaining: %d\n", 3 - attempts);
     }
   }
 }

 printf("\n✗ Too many failed attempts! Returning to main menu.\n");
 pressEnterToContinue();
 return 0;
}

// ==================== ADMIN MENU ====================
void adminMenu() {
 int choice;

 while (1) {
   clearScreen();
   printf("\n========================================\n");
   printf("           ADMIN PANEL\n");
   printf("========================================\n");
   printf("\n  1. View Menu\n");
```

```
    printf("  2. Add Menu Item\n");
    printf("  3. Update Menu Item\n");
    printf("  4. Delete Menu Item\n");
    printf("  5. View All Orders\n");
    printf("  6. View Orders by Status\n");
    printf("  7. Update Order Status\n");
    printf("  8. Generate Sales Report\n");
    printf("  9. Logout\n");
    printf("----------------------------------------\n");
    printf("\nEnter your choice: ");

    if (scanf("%d", &choice) != 1) {
      while (getchar() != '\n')
        ;
      printf("\nInvalid input!\n");
      pressEnterToContinue();
      continue;
    }
    while (getchar() != '\n')
      ;

    switch (choice) {
    case 1:
      viewMenu();
      pressEnterToContinue();
      break;
    case 2:
      addMenuItem();
      break;
    case 3:
      updateMenuItem();
      break;
    case 4:
      deleteMenuItem();
      break;
    case 5:
      viewAllOrders();
      pressEnterToContinue();
      break;
    case 6:
      viewOrdersByStatus();
      pressEnterToContinue();
      break;
    case 7:
      updateOrderStatus();
      break;
    case 8:
      generateSalesReport();
      pressEnterToContinue();
      break;
    case 9:
      printf("\n✓ Logged out successfully!\n");
      pressEnterToContinue();
      return;
    default:
      printf("\nInvalid choice!\n");
      pressEnterToContinue();
    }
  }
}

// ==================== CUSTOMER MENU ====================
void customerMenu() {
 int choice;

 while (1) {
    clearScreen();
    printf("\n========================================\n");
    printf("         CUSTOMER MODE\n");
    printf("========================================\n");
    printf("\n  1. View Full Menu\n");
    printf("  2. View Menu by Category\n");
    printf("  3. Place Order\n");
    printf("  4. Return to Main Menu\n");
```

```
     printf("----------------------------------------\n");
     printf("\nEnter your choice: ");

     if (scanf("%d", &choice) != 1) {
       while (getchar() != '\n')
         ;
       printf("\nInvalid input!\n");
       pressEnterToContinue();
       continue;
     }
     while (getchar() != '\n')
       ;

     switch (choice) {
     case 1:
       viewCustomerMenu();
       pressEnterToContinue();
       break;
     case 2:
       viewMenuByCategory();
       pressEnterToContinue();
       break;
     case 3:
       placeOrder();
       break;
     case 4:
       return;
     default:
       printf("\nInvalid choice!\n");
       pressEnterToContinue();
     }
   }
 }

// ==================== ADD MENU ITEM ====================
void addMenuItem() {
 clearScreen();

 // First display current menu
 printf("\n==========================================\n");
 printf("       CURRENT MENU\n");
 printf("========================================\n");

 FILE *fp_read = fopen(MENU_FILE, "r");
 if (fp_read) {
   struct MenuItem item;
   int count = 0;

   printf("\n%-6s %-30s %-20s %-12s %-10s\n", "ID", "Item Name", "Category",
          "Price", "Status");
   printf("------------------------------------------------------------------"
          "-------------\n");

   while (fscanf(fp_read, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id,
                 item.item_name, item.category, &item.price,
                 &item.available) != EOF) {
     printf("%-6d %-30s %-20s Rs. %-8.2f %-10s\n", item.item_id,
            item.item_name, item.category, item.price,
            item.available ? "Available" : "Not Available");
     count++;
   }

   printf("------------------------------------------------------------------"
          "-------------\n");
   printf("Total Items: %d\n\n", count);
   fclose(fp_read);
 } else {
   printf("\nNo existing menu items.\n\n");
 }

 // Now add new item
 FILE *fp = fopen(MENU_FILE, "a");
 if (!fp) {
   printf("\n✗ Error opening menu file!\n");
   pressEnterToContinue();
   return;
 }

 struct MenuItem item;

 printf("========================================\n");
 printf("        ADD NEW MENU ITEM\n");
 printf("========================================\n");

 int id_exists;
 do {
   id_exists = 0;
   printf("\nEnter Item ID: ");
   scanf("%d", &item.item_id);
   while (getchar() != '\n')
     ;

   // Check if ID exists
   FILE *check_fp = fopen(MENU_FILE, "r");
   if (check_fp) {
     struct MenuItem temp_item;
     while (fscanf(check_fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &temp_item.item_id,
                   temp_item.item_name, temp_item.category, &temp_item.price,
                   &temp_item.available) != EOF) {
       if (temp_item.item_id == item.item_id) {
         id_exists = 1;
         printf(
             "Error: Item ID %d already exists! Please enter a unique ID.\n",
             item.item_id);
         break;
       }
     }
     fclose(check_fp);
   }
 } while (id_exists);

 printf("Enter Item Name: ");
 fgets(item.item_name, sizeof(item.item_name), stdin);
 item.item_name[strcspn(item.item_name, "\n")] = '\0';

 printf("Enter Category (Appetizer/Main Course/Dessert/Beverage): ");
 fgets(item.category, sizeof(item.category), stdin);
 item.category[strcspn(item.category, "\n")] = '\0';

 printf("Enter Price: Rs. ");
 scanf("%f", &item.price);

 item.available = 1;

 fprintf(fp, "%d|%s|%s|%.2f|%d\n", item.item_id, item.item_name, item.category,
         item.price, item.available);

 fclose(fp);

 printf("\n✓ Menu item added successfully!\n");
 pressEnterToContinue();
}

// ==================== VIEW MENU ====================
void viewMenu() {
 clearScreen();
 FILE *fp = fopen(MENU_FILE, "r");
```

```
  if (!fp) {
    printf("\n✗ Menu file not found!\n");
    return;
  }

  struct MenuItem item;
  int count = 0;

  printf("\n================================================================="
         "=============\n");
  printf("                              RESTAURANT MENU\n");
  printf("=================================================================="
         "===========\n");
  printf("%-6s %-30s %-20s %-12s %-10s\n", "ID", "Item Name", "Category",
         "Price", "Status");
  printf("------------------------------------------------------------------"
         "-----------\n");

  while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id, item.item_name,
                item.category, &item.price, &item.available) != EOF) {
    printf("%-6d %-30s %-20s Rs. %-8.2f %-10s\n", item.item_id, item.item_name,
           item.category, item.price,
           item.available ? "Available" : "Not Available");
    count++;
  }

  printf("------------------------------------------------------------------"
         "-----------\n");
  printf("Total Items: %d\n", count);
  printf("=================================================================="
         "===========\n");

  fclose(fp);
}

// =================== UPDATE MENU ITEM ===================
void updateMenuItem() {
  viewMenu();
  int id, found = 0, choice;

  printf("\n========================================\n");
  printf("        UPDATE MENU ITEM\n");
  printf("========================================\n");

  printf("\nEnter Item ID to update: ");
  scanf("%d", &id);

  FILE *fp = fopen(MENU_FILE, "r");
  FILE *temp = fopen(TEMP_FILE, "w");

  if (!fp || !temp) {
    printf("\n✗ Error opening file!\n");
    pressEnterToContinue();
    return;
  }

  struct MenuItem item;

  while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id, item.item_name,
                item.category, &item.price, &item.available) != EOF) {

    if (item.item_id == id) {
      found = 1;
      printf("\nCurrent Details:\n");
      printf("Name: %s\n", item.item_name);
      printf("Category: %s\n", item.category);
      printf("Price: Rs. %.2f\n", item.price);
      printf("Status: %s\n", item.available ? "Available" : "Not Available");

      printf("\nWhat do you want to update?\n");
      printf("1. Name\n");
      printf("2. Category\n");
      printf("3. Price\n");
      printf("4. Availability\n");
      printf("5. All Details\n");
      printf("\nEnter choice: ");
      scanf("%d", &choice);
      while (getchar() != '\n')
        ;

      switch (choice) {
      case 1:
        printf("Enter new name: ");
        fgets(item.item_name, sizeof(item.item_name), stdin);
        item.item_name[strcspn(item.item_name, "\n")] = '\0';
        break;
      case 2:
        printf("Enter new category: ");
        fgets(item.category, sizeof(item.category), stdin);
        item.category[strcspn(item.category, "\n")] = '\0';
        break;
      case 3:
        printf("Enter new price: Rs. ");
        scanf("%f", &item.price);
        break;
      case 4:
        printf("Available? (1=Yes, 0=No): ");
        scanf("%d", &item.available);
        break;
      case 5:
        printf("Enter new name: ");
        fgets(item.item_name, sizeof(item.item_name), stdin);
        item.item_name[strcspn(item.item_name, "\n")] = '\0';

        printf("Enter new category: ");
        fgets(item.category, sizeof(item.category), stdin);
        item.category[strcspn(item.category, "\n")] = '\0';

        printf("Enter new price: Rs. ");
        scanf("%f", &item.price);

        printf("Available? (1=Yes, 0=No): ");
        scanf("%d", &item.available);
        break;
      default:
        printf("\nInvalid choice! No changes made.\n");
      }
    }

    fprintf(temp, "%d|%s|%s|%.2f|%d\n", item.item_id, item.item_name,
            item.category, item.price, item.available);
  }

  fclose(fp);
  fclose(temp);

  remove(MENU_FILE);
  rename(TEMP_FILE, MENU_FILE);

  if (found) {
    printf("\n✓ Menu item updated successfully!\n");
```

```
 } else {
   printf("\n✗ Item ID not found!\n");
 }

 pressEnterToContinue();
}

// ==================== DELETE MENU ITEM ====================
void deleteMenuItem() {
 viewMenu();
 int id, found = 0;
 char confirm;

 printf("\n=========================================\n");
 printf("         DELETE MENU ITEM\n");
 printf("========================================\n");

 printf("\nEnter Item ID to delete: ");
 scanf("%d", &id);

 FILE *fp = fopen(MENU_FILE, "r");
 FILE *temp = fopen(TEMP_FILE, "w");

 if (!fp || !temp) {
   printf("\n✗ Error opening file!\n");
   pressEnterToContinue();
   return;
 }

 struct MenuItem item;

 while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id, item.item_name,
               item.category, &item.price, &item.available) != EOF) {

   if (item.item_id == id) {
     found = 1;
     printf("\nItem Found:\n");
     printf("ID: %d\n", item.item_id);
     printf("Name: %s\n", item.item_name);
     printf("Price: Rs. %.2f\n", item.price);

     printf("\nAre you sure you want to delete? (y/n): ");
     scanf(" %c", &confirm);

     if (confirm == 'y' || confirm == 'Y') {
       printf("\n✓ Item deleted successfully!\n");
       continue; // Skip writing this item
     } else {
       printf("\n✗ Deletion cancelled!\n");
     }
   }

   fprintf(temp, "%d|%s|%s|%.2f|%d\n", item.item_id, item.item_name,
           item.category, item.price, item.available);
 }

 fclose(fp);
 fclose(temp);

 remove(MENU_FILE);
 rename(TEMP_FILE, MENU_FILE);

 if (!found) {
   printf("\n✗ Item ID not found!\n");
 }

 pressEnterToContinue();
}

// ==================== PLACE ORDER ====================
void placeOrder() {
 clearScreen();
```

```

  // Initialize order
  currentOrder.order_id = generateOrderID();
  currentOrder.item_count = 0;
  currentOrder.total_amount = 0;
  strcpy(currentOrder.status, "Pending");

  // Get current date
  time_t t = time(NULL);
  struct tm tm = *localtime(&t);
  sprintf(currentOrder.date, "%02d/%02d/%04d", tm.tm_mday, tm.tm_mon + 1,
          tm.tm_year + 1900);

  printf("\n========================================\n");
  printf("          PLACE NEW ORDER\n");
  printf("========================================\n");

  printf("\nEnter Customer Name: ");
  fgets(currentOrder.customer_name, sizeof(currentOrder.customer_name), stdin);
  currentOrder.customer_name[strcspn(currentOrder.customer_name, "\n")] = '\0';

  // Display menu
  viewCustomerMenu();

  int item_id, quantity;
  char more;

  do {
    printf("\nEnter Item ID: ");
    if (scanf("%d", &item_id) != 1) {
      while (getchar() != '\n')
        ;
      printf("Invalid input!\n");
      continue;
    }

    printf("Enter Quantity: ");
    if (scanf("%d", &quantity) != 1 || quantity <= 0) {
      while (getchar() != '\n')
        ;
      printf("Invalid quantity!\n");
      continue;
    }
    while (getchar() != '\n')
      ;

    // Search for item in menu
    FILE *fp = fopen(MENU_FILE, "r");
    if (!fp) {
      printf("\n✗ Error opening menu file!\n");
      pressEnterToContinue();
      return;
    }

    struct MenuItem item;
    int found = 0;

    while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id,
                  item.item_name, item.category, &item.price,
                  &item.available) != EOF) {

      if (item.item_id == item_id && item.available == 1) {
        found = 1;

        // Add to order
        int idx = currentOrder.item_count;
        currentOrder.items[idx].item_id = item.item_id;
        strcpy(currentOrder.items[idx].item_name, item.item_name);
        currentOrder.items[idx].price = item.price;
        currentOrder.items[idx].quantity = quantity;
        currentOrder.items[idx].subtotal = item.price * quantity;

        currentOrder.item_count++;

        printf("\n✓ Added: %s x %d = Rs. %.2f\n", item.item_name, quantity,
               item.price * quantity);
        break;
      }
    }

    fclose(fp);

    if (!found) {
      printf("\n✗ Item not found or not available!\n");
    }

    printf("\nAdd more items? (y/n): ");
    scanf(" %c", &more);
    while (getchar() != '\n')
      ;

  } while ((more == 'y' || more == 'Y') && currentOrder.item_count < 20);

  if (currentOrder.item_count == 0) {
    printf("\n✗ No items added! Order cancelled.\n");
    pressEnterToContinue();
    return;
  }

  // Calculate bill
  calculateBill(&currentOrder);

  // Display order summary
  clearScreen();
  displayOrder(currentOrder);

  // Save order to file
  FILE *fp = fopen(ORDER_FILE, "a");
  if (!fp) {
    printf("\n✗ Error saving order!\n");
    pressEnterToContinue();
    return;
  }

  fprintf(fp, "%d|%s|%s|%d|%.2f|%s\n", currentOrder.order_id,
          currentOrder.customer_name, currentOrder.date,
          currentOrder.item_count, currentOrder.total_amount,
          currentOrder.status);

  // Save order items
  for (int i = 0; i < currentOrder.item_count; i++) {
    fprintf(fp, "ITEM|%d|%s|%.2f|%d|%.2f\n", currentOrder.items[i].item_id,
            currentOrder.items[i].item_name, currentOrder.items[i].price,
            currentOrder.items[i].quantity, currentOrder.items[i].subtotal);
  }

  fclose(fp);

  printf("\n✓ Order placed successfully!\n");
  printf("Your Order ID: %d\n", currentOrder.order_id);
  pressEnterToContinue();
}

// ==================== VIEW CUSTOMER MENU ====================
void viewCustomerMenu() {
  FILE *fp = fopen(MENU_FILE, "r");
  if (!fp) {
    printf("\n✗ Menu not available!\n");
    return;
  }

  struct MenuItem item;

  printf("\n======================================================================="
         "============\n");
  printf("                              OUR MENU\n");
  printf("========================================================================"
         "===========\n");
  printf("%-6s %-35s %-20s %-10s\n", "ID", "Item Name", "Category", "Price");
  printf("------------------------------------------------------------------------"
         "-----------\n");

  while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id, item.item_name,
                item.category, &item.price, &item.available) != EOF) {
    if (item.available == 1) {
      printf("%-6d %-35s %-20s Rs. %.2f\n", item.item_id, item.item_name,
             item.category, item.price);
    }
  }

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

  fclose(fp);
}

// =================== VIEW MENU BY CATEGORY ===================
void viewMenuByCategory() {
  clearScreen();
  FILE *fp = fopen(MENU_FILE, "r");
  if (!fp) {
    printf("\n✗ Menu not available!\n");
    return;
  }

  int choice;
  char selected_category[30];

  printf("\n========================================\n");
  printf("      SELECT CATEGORY\n");
  printf("========================================\n");
  printf("\n  1. Appetizer\n");
  printf("  2. Main Course\n");
  printf("  3. Dessert\n");
  printf("  4. Beverage\n");
  printf("----------------------------------------\n");
  printf("\nEnter your choice: ");

  if (scanf("%d", &choice) != 1) {
    while (getchar() != '\n')
      ;
    printf("\n✗ Invalid input!\n");
    fclose(fp);
    return;
  }
  while (getchar() != '\n')
    ;

  switch (choice) {
  case 1:
    strcpy(selected_category, "Appetizer");
    break;
```

```c
 case 2:
   strcpy(selected_category, "Main Course");
   break;
 case 3:
   strcpy(selected_category, "Dessert");
   break;
 case 4:
   strcpy(selected_category, "Beverage");
   break;
 default:
   printf("\n✗ Invalid choice!\n");
   fclose(fp);
   return;
 }

 struct MenuItem item;
 int count = 0;

 printf("\n===================================================================="
        "=============\n");
 printf("                         MENU - %s\n", selected_category);
 printf("====================================================================="
        "===========\n");
 printf("%-6s %-35s %-20s %-10s\n", "ID", "Item Name", "Category", "Price");
 printf("---------------------------------------------------------------------"
        "-----------\n");

 rewind(fp);

 while (fscanf(fp, "%d|%49[^|]|%29[^|]|%f|%d\n", &item.item_id, item.item_name,
               item.category, &item.price, &item.available) != EOF) {
   if (item.available == 1 && strcmp(item.category, selected_category) == 0) {
     printf("%-6d %-35s %-20s Rs. %.2f\n", item.item_id, item.item_name,
            item.category, item.price);
     count++;
   }
 }

 if (count == 0) {
   printf("No items available in this category.\n");
 }

 printf("---------------------------------------------------------------------"
        "-----------\n");
 printf("Total Items in %s: %d\n", selected_category, count);
 printf("====================================================================="
        "===========\n");

 fclose(fp);
}

// =================== GENERATE SALES REPORT ===================
void generateSalesReport() {
 clearScreen();
 FILE *fp = fopen(ORDER_FILE, "r");
 if (!fp) {
   printf("\n✗ No orders found! Cannot generate report.\n");
   return;
 }
```

```

  printf("\n===================================================================="
         "============\n");
  printf("                           SALES REPORT\n");
  printf("====================================================================="
         "===========\n");

  struct Order order;
  char line[200];

  int total_orders = 0;
  int pending_orders = 0;
  int completed_orders = 0;
  int cancelled_orders = 0;
  float total_revenue = 0.0;
  float completed_revenue = 0.0;

  // Item sales tracking
  struct ItemSales {
    int item_id;
    char item_name[50];
    int total_quantity;
    float total_sales;
  } item_sales[100];
  int item_count = 0;

  // Read all orders
  while (fgets(line, sizeof(line), fp)) {
    if (sscanf(line, "%d|%49[^|]|%39[^|]|%d|%f|%19[^\n]", &order.order_id,
               order.customer_name, order.date, &order.item_count,
               &order.total_amount, order.status) == 6) {

      total_orders++;
      total_revenue += order.total_amount;

      if (strcmp(order.status, "Pending") == 0) {
        pending_orders++;
      } else if (strcmp(order.status, "Completed") == 0) {
        completed_orders++;
        completed_revenue += order.total_amount;
      } else if (strcmp(order.status, "Cancelled") == 0) {
        cancelled_orders++;
      }

      // Read item details for this order
      for (int i = 0; i < order.item_count; i++) {
        char item_line[200];
        int item_id, quantity;
        char item_name[50];
        float price, subtotal;

        if (fgets(item_line, sizeof(item_line), fp)) {
          if (sscanf(item_line, "ITEM|%d|%49[^|]|%f|%d|%f", &item_id, item_name,
                     &price, &quantity, &subtotal) == 5) {

            // Only count completed orders for item sales
            if (strcmp(order.status, "Completed") == 0) {
              // Check if item already exists in tracking
              int found = 0;
              for (int j = 0; j < item_count; j++) {
                if (item_sales[j].item_id == item_id) {
                  item_sales[j].total_quantity += quantity;
                  item_sales[j].total_sales += subtotal;
                  found = 1;
                  break;
                }
              }

```

```
                // If not found, add new item
                if (!found && item_count < 100) {
                  item_sales[item_count].item_id = item_id;
                  strcpy(item_sales[item_count].item_name, item_name);
                  item_sales[item_count].total_quantity = quantity;
                  item_sales[item_count].total_sales = subtotal;
                  item_count++;
                }
              }
            }
          }
        }
      }
    }

    fclose(fp);

    // Display Summary
    printf("\n--- ORDER SUMMARY ---\n");
    printf("Total Orders: %d\n", total_orders);
    printf("  - Pending: %d\n", pending_orders);
    printf("  - Completed: %d\n", completed_orders);
    printf("  - Cancelled: %d\n", cancelled_orders);

    printf("\n--- REVENUE SUMMARY ---\n");
    printf("Total Revenue (All Orders): Rs. %.2f\n", total_revenue);
    printf("Completed Orders Revenue: Rs. %.2f\n", completed_revenue);

    if (completed_orders > 0) {
      printf("Average Order Value: Rs. %.2f\n",
             completed_revenue / completed_orders);
    }

    // Display Item Sales
    if (item_count > 0) {
printf("\n=================================================================="
             "===============\n");
      printf("                         ITEM-WISE SALES REPORT\n");
printf("==================================================================="
             "=============\n");
      printf("%-6s %-35s %-15s %-15s\n", "ID", "Item Name", "Qty Sold",
             "Revenue");
printf("-------------------------------------------------------------------"
             "-------------\n");

      // Sort items by quantity sold (bubble sort - simple implementation)
      for (int i = 0; i < item_count - 1; i++) {
        for (int j = 0; j < item_count - i - 1; j++) {
          if (item_sales[j].total_quantity < item_sales[j + 1].total_quantity) {
            struct ItemSales temp = item_sales[j];
            item_sales[j] = item_sales[j + 1];
            item_sales[j + 1] = temp;
          }
        }
      }

      // Display sorted items
      for (int i = 0; i < item_count; i++) {
        printf("%-6d %-35s %-15d Rs. %.2f\n", item_sales[i].item_id,
               item_sales[i].item_name, item_sales[i].total_quantity,
               item_sales[i].total_sales);
      }

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

      // Display Top 5 Best Sellers
      printf("\n--- TOP 5 BEST SELLING ITEMS ---\n");
      int top_count = (item_count < 5) ? item_count : 5;
      for (int i = 0; i < top_count; i++) {
        printf("%d. %s - %d units sold (Rs. %.2f)\n", i + 1,
               item_sales[i].item_name, item_sales[i].total_quantity,
               item_sales[i].total_sales);
      }
    } else {
      printf("\n--- ITEM-WISE SALES REPORT ---\n");
      printf("No completed orders to generate item sales report.\n");
    }

printf("\n=================================================================="
           "============\n");
    printf("                         END OF REPORT\n");
printf("==================================================================="
           "===========\n");
  }

  // =================== VIEW ALL ORDERS ===================
  void viewAllOrders() {
    clearScreen();
    FILE *fp = fopen(ORDER_FILE, "r");
    if (!fp) {
      printf("\n✗ No orders found!\n");
      return;
    }

printf("\n=================================================================="
           "============\n");
    printf("                           ALL ORDERS\n");
printf("==================================================================="
           "===========\n");

    struct Order order;
    char line[200];
    int order_count = 0;

    while (fgets(line, sizeof(line), fp)) {
      if (sscanf(line, "%d|%49[^|]|%39[^|]|%d|%f|%19[^\n]", &order.order_id,
                 order.customer_name, order.date, &order.item_count,
                 &order.total_amount, order.status) == 6) {

        order_count++;
        printf("\n--- Order #%d ---\n", order.order_id);
        printf("Customer: %s\n", order.customer_name);
        printf("Date: %s\n", order.date);
        printf("No. of Items: %d\n", order.item_count);

        // Read and display item details
        printf("Items: ");
        for (int i = 0; i < order.item_count; i++) {
          char item_line[200];
          char item_name[50];
          int item_id, quantity;
          float price, subtotal;

          if (fgets(item_line, sizeof(item_line), fp)) {
            if (sscanf(item_line, "ITEM|%d|%49[^|]|%f|%d|%f", &item_id, item_name,
                       &price, &quantity, &subtotal) == 5) {
              if (i > 0)
                printf(", ");
              printf("%s(x%d) (Rs.%.2f)", item_name, quantity, subtotal);
            }
          }
        }
        printf("\n");

        printf("Total: Rs. %.2f\n", order.total_amount);
        printf("Status: %s\n", order.status);

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

    printf("\nTotal Orders: %d\n", order_count);
    printf("=====================================================================" 
           "===========\n");

    fclose(fp);
   }

   // =================== VIEW ORDERS BY STATUS ===================
   void viewOrdersByStatus() {
    clearScreen();
    char search_status[20];

    printf("\n========================================\n");
    printf("      VIEW ORDERS BY STATUS\n");
    printf("========================================\n");
    printf("\nEnter Status (Pending/Completed/Cancelled): ");
    fgets(search_status, sizeof(search_status), stdin);
    search_status[strcspn(search_status, "\n")] = '\0';

    FILE *fp = fopen(ORDER_FILE, "r");
    if (!fp) {
      printf("\n✗ No orders found!\n");
      return;
    }

    struct Order order;
    char line[200];
    int found = 0;

    printf("\n==================================================================="
           "============\n");
    printf("Orders with Status: %s\n", search_status);
    printf("====================================================================="
           "===========\n");

    while (fgets(line, sizeof(line), fp)) {
      if (sscanf(line, "%d|%49[^|]|%39[^|]|%d|%f|%19[^\n]", &order.order_id,
                 order.customer_name, order.date, &order.item_count,
                 &order.total_amount, order.status) == 6) {

        if (strcmp(order.status, search_status) == 0) {
          found++;
          printf("\n--- Order #%d ---\n", order.order_id);
          printf("Customer: %s\n", order.customer_name);
          printf("Date: %s\n", order.date);
          printf("No. of Items: %d\n", order.item_count);

          // Read and display item details
          printf("Items: ");
```

```
       for (int i = 0; i < order.item_count; i++) {
         char item_line[200];
         char item_name[50];
         int item_id, quantity;
         float price, subtotal;

         if (fgets(item_line, sizeof(item_line), fp)) {
           if (sscanf(item_line, "ITEM|%d|%49[^|]|%f|%d|%f", &item_id,
                      item_name, &price, &quantity, &subtotal) == 5) {
             if (i > 0)
               printf(", ");
             printf("%s(x%d) (Rs.%.2f)", item_name, quantity, subtotal);
           }
         }
       }
       printf("\n");

       printf("Total: Rs. %.2f\n", order.total_amount);
       printf("----------------------------------------------------------------"
              "-----------------\n");
     } else {
       // Skip item lines for non-matching orders
       for (int i = 0; i < order.item_count; i++) {
         fgets(line, sizeof(line), fp);
       }
     }
   }
 }

 if (found == 0) {
   printf("\nNo orders found with status: %s\n", search_status);
 } else {
   printf("\nTotal Orders Found: %d\n", found);
 }

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

 fclose(fp);
}

// ==================== UPDATE ORDER STATUS ====================
void updateOrderStatus() {
 viewAllOrders();
 int order_id, found = 0;
 char new_status[20];

 printf("\n==========================================\n");
 printf("      UPDATE ORDER STATUS\n");
 printf("==========================================\n");

 printf("\nEnter Order ID to update: ");
 if (scanf("%d", &order_id) != 1) {
   while (getchar() != '\n')
     ;
   printf("\n✗ Invalid input!\n");
   pressEnterToContinue();
   return;
 }
 while (getchar() != '\n')
   ;

 FILE *fp = fopen(ORDER_FILE, "r");
 FILE *temp = fopen(TEMP_FILE, "w");

 if (!fp || !temp) {
   printf("\n✗ Error opening file!\n");
   pressEnterToContinue();
```

```
    return;
  }

  struct Order order;
  char line[200];

  while (fgets(line, sizeof(line), fp)) {
    if (sscanf(line, "%d|%49[^|]|%39[^|]|%d|%f|%19[^\n]", &order.order_id,
               order.customer_name, order.date, &order.item_count,
               &order.total_amount, order.status) == 6) {

      if (order.order_id == order_id) {
        found = 1;
        printf("\nCurrent Order Details:\n");
        printf("Order ID: %d\n", order.order_id);
        printf("Customer: %s\n", order.customer_name);
        printf("Date: %s\n", order.date);
        printf("No. of Items: %d\n", order.item_count);

        // Read and display item details
        printf("Items: ");
        char item_lines[20][200]; // Store item lines temporarily
        for (int i = 0; i < order.item_count; i++) {
          char item_name[50];
          int item_id, quantity;
          float price, subtotal;

          if (fgets(item_lines[i], sizeof(item_lines[i]), fp)) {
            if (sscanf(item_lines[i], "ITEM|%d|%49[^|]|%f|%d|%f", &item_id,
                       item_name, &price, &quantity, &subtotal) == 5) {
              if (i > 0)
                printf(", ");
              printf("%s(x%d) (Rs.%.2f)", item_name, quantity, subtotal);
            }
          }
        }
        printf("\n");

        printf("Total: Rs. %.2f\n", order.total_amount);
        printf("Current Status: %s\n", order.status);

        printf("\nSelect New Status:\n");
        printf("1. Pending\n");
        printf("2. Completed\n");
        printf("3. Cancelled\n");
        printf("\nEnter choice: ");

        int choice;
        if (scanf("%d", &choice) != 1) {
          while (getchar() != '\n')
            ;
          printf("\n✗ Invalid input!\n");
          fprintf(temp, "%s", line);
          // Copy remaining item lines that were already read
          for (int i = 0; i < order.item_count; i++) {
            fprintf(temp, "%s", item_lines[i]);
          }
          found = 1; // Mark as found to avoid "not found" message
          break;
        }
        while (getchar() != '\n')
          ;

        switch (choice) {
        case 1:
          strcpy(new_status, "Pending");
          break;
        case 2:
          strcpy(new_status, "Completed");
          break;
```

```
       case 3:
         strcpy(new_status, "Cancelled");
         break;
       default:
         printf("\n✗ Invalid choice! No changes made.\n");
         strcpy(new_status, order.status);
       }

       // Write updated order
       fprintf(temp, "%d|%s|%s|%d|%.2f|%s\n", order.order_id,
               order.customer_name, order.date, order.item_count,
               order.total_amount, new_status);

       // Write item lines
       for (int i = 0; i < order.item_count; i++) {
         fprintf(temp, "%s", item_lines[i]);
       }

       printf("\n✓ Order status updated to: %s\n", new_status);
     } else {
       // Write unchanged order
       fprintf(temp, "%s", line);
     }

     // Copy item lines
     for (int i = 0; i < order.item_count; i++) {
       fgets(line, sizeof(line), fp);
       fprintf(temp, "%s", line);
     }
   } else {
     // Write any other lines as-is
     fprintf(temp, "%s", line);
   }
 }

 fclose(fp);
 fclose(temp);

 remove(ORDER_FILE);
 rename(TEMP_FILE, ORDER_FILE);

 if (!found) {
   printf("\n✗ Order ID not found!\n");
 }

 pressEnterToContinue();
}

// ==================== CALCULATE BILL ====================
void calculateBill(struct Order *order) {
 order->total_amount = 0;
 for (int i = 0; i < order->item_count; i++) {
   order->total_amount += order->items[i].subtotal;
 }
}

// ==================== DISPLAY ORDER ====================
void displayOrder(struct Order order) {
printf("\n==================================================================="
        "=============\n");
 printf("                              ORDER BILL\n");
printf("====================================================================="
        "===========\n");
 printf("Order ID: %d\n", order.order_id);
 printf("Customer Name: %s\n", order.customer_name);
 printf("Date: %s\n", order.date);
printf("====================================================================="
        "===========\n");
```

```
  printf("%-6s %-30s %-10s %-10s %-12s\n", "ID", "Item", "Price", "Qty",
         "Subtotal");
  printf("---------------------------------------------------------------------"
         "-----------\n");

  for (int i = 0; i < order.item_count; i++) {
    printf("%-6d %-30s Rs. %-7.2f %-10d Rs. %.2f\n", order.items[i].item_id,
           order.items[i].item_name, order.items[i].price,
           order.items[i].quantity, order.items[i].subtotal);
  }

  printf("====================================================================="
         "===========\n");
  printf("                                           TOTAL: Rs. %.2f\n",
         order.total_amount);
  printf("====================================================================="
         "===========\n");
}

// ==================== GENERATE ORDER ID ====================
int generateOrderID() {
  FILE *fp = fopen(ORDER_FILE, "r");
  if (!fp) {
    return 1001; // First order ID
  }

  int max_id = 1000;
  char line[200];
  int id;

  while (fgets(line, sizeof(line), fp)) {
    if (sscanf(line, "%d|", &id) == 1) {
      if (id > max_id) {
        max_id = id;
      }
    }
  }

  fclose(fp);
  return max_id + 1;
}

// ==================== INITIALIZE MENU FILE ====================
void initializeMenuFile() {
  FILE *fp = fopen(MENU_FILE, "r");
  if (fp) {
    fclose(fp);
    return; // File exists
  }

  // Create file with sample items
  fp = fopen(MENU_FILE, "w");
  if (!fp)
    return;

  fprintf(fp, "101|Momo (Veg)|Appetizer|150.00|1\n");
  fprintf(fp, "102|Momo (Chicken)|Appetizer|180.00|1\n");
  fprintf(fp, "103|Chowmein|Main Course|120.00|1\n");
  fprintf(fp, "104|Fried Rice|Main Course|140.00|1\n");
  fprintf(fp, "105|Dal Bhat|Main Course|200.00|1\n");
  fprintf(fp, "106|Pizza|Main Course|350.00|1\n");
  fprintf(fp, "107|Burger|Main Course|180.00|1\n");
  fprintf(fp, "108|Ice Cream|Dessert|100.00|1\n");
  fprintf(fp, "109|Gulab Jamun|Dessert|80.00|1\n");
  fprintf(fp, "110|Coca Cola|Beverage|60.00|1\n");
```

```
    fprintf(fp, "111|Fresh Juice|Beverage|100.00|1\n");

    fclose(fp);
}

// ==================== UTILITY FUNCTIONS ====================
void pressEnterToContinue() {
    printf("\nPress Enter to continue...");
    getchar();
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}
```

4.3 Testing

Testing is a critical phase to verify that the implemented system functions correctly according to the user requirements. This phase involved **Unit Testing** (testing individual functions) and **System Testing** (testing the entire application flow).

4.3.1 Unit Testing

Each core module, such as `adminLogin()`, `addMenuItem()`, `placeOrder()`, and `calculateBill()`, was tested independently to confirm it performed its specific function accurately. The `calculateBill()` function was particularly tested to ensure float arithmetic for total amounts was correct.

4.3.2 Test Cases

The following table documents key test cases run on the system, demonstrating its functionality, validation, and error handling.

Table-9: *Test Results*

S. No.	Module	Test Scenario (Input Condition)	Expected Result	Actual Result	Status
1	**Admin Login**	Enter correct credentials (Username: admin, Password: admin123).	Successful login; display Admin Menu.	Successful login; Admin Menu displayed.	Success
2	**Admin Login**	Enter incorrect password on all three attempts.	Deny access; display "Too many failed attempts"; return to Main Menu.	Access denied; returns to Main Menu.	Success
3	**Add Menu Item**	Add a new item with a unique ID (e.g., ID 200, Name: Soda, Price: 50).	Item successfully appended to menu.txt; displayed in menu list.	Item added; validation for unique ID passed.	Success

4	**Add Menu Item**	Attempt to add a new item with an existing ID (e.g., ID 101).	System displays "Error: Item ID 101 already exists!"; prompts for new ID.	ID conflict handled; prompts for unique ID.	Success
5	**Update Menu Item**	Update the price of an existing item (e.g., ID 104, New Price: 160.00).	menu.txt record for ID 104 is updated; old price replaced.	Price updated successfully using a temporary file.	Success
6	**Delete Menu Item**	Delete an item by ID (e.g., ID 109) and confirm deletion.	Item removed from menu.txt; menu count decreases by one.	Item successfully deleted from file.	Success
7	**Place Order**	Place an order for three items with varying quantities (e.g., Item 101 x 2, Item 103 x 1, Item 110 x 3).	A new record is created in orders.txt with a unique ID (e.g., 1001); Bill amount calculated correctly (300 + 120 + 180 = 600.00).	Order saved; total amount calculated and displayed accurately.	Success
8	**Place Order**	Attempt to order an item that is set as unavailable (available = 0).	System displays "Item not found or not available!"; item not added to the order.	Unavailable item skipped; order continues.	Success
9	**Update Order Status**	Change the status of a Pending order (e.g., ID 1001) to Completed.	orders.txt header record for order ID 1001 is updated to Completed.	Status updated successfully.	Success
10	**Sales Report**	Generate reports after several orders, including Completed and Cancelled ones.	Report accurately reflects Total Orders, segregates revenue by Completed status, and lists item sales for Completed orders only.	Correct breakdown of order status and accurate revenue figures displayed.	Success

Chapter 5: Conclusion

5.1 Project Summary and Conclusion

The **Restaurant Order Management System** project was successfully designed, developed, and tested in accordance with the specifications outlined in the feasibility study and project guidelines. Using the **C programming language** and robust **file handling** techniques, the system effectively meets its primary objective: to replace the manual, error-prone processes of a restaurant with an efficient and computerized alternative.

The system's dual-mode functionality—a secure **Admin Panel** for menu management and sales analysis, and an intuitive **Customer Mode** for viewing and placing orders—demonstrates a strong application of structured programming principles. All core functions, including item creation, order processing, dynamic billing, and report generation, were implemented and validated through comprehensive testing, ensuring high accuracy and reliability. The use of delimited text files (`menu.txt`, `orders.txt`) ensures that all data remains persistent across program sessions.

In conclusion, this project successfully provides a functional, cost-effective, and practical solution for streamlining the essential front-end operations of a small-to-medium-sized restaurant.

5.2 Difficulties Encountered

Throughout the implementation phase, several challenges inherent to developing a large-scale C program were encountered:

1. **File Handling Complexity:** Managing multiple related data items (Order Header and multiple Order Item details) within a single sequential file (`orders.txt`) required complex logic to read, write, and ensure the corresponding lines were properly linked, especially when updating order statuses.
2. **String Manipulation:** Handling user input, particularly strings containing spaces (like item names or customer names), required careful use of functions like `fgets()` and `strcspn()` to prevent buffer overflows and ensure consistent data formatting (using the pipe `|` delimiter) for successful file parsing with `sscanf()`.
3. **Data Persistence for Order IDs:** Ensuring the `order_id` generator always found the highest existing ID in `orders.txt` and incremented it correctly, even if the file was large or contained fragmented data, required meticulous reading of the file's first field.

5.3 Future Enhancements

While the current system fulfills all project objectives, several enhancements could be implemented in future iterations to improve functionality and usability:

1. **Graphical User Interface (GUI):** Replacing the current console-based interface with a GUI (using libraries like GTK or migrating to a language like Python/Java) would greatly enhance user experience and modern appeal.

2. **Database Integration:** Migrating data storage from plain text files (`.txt`) to a full-fledged Database Management System (DBMS) like MySQL or PostgreSQL would significantly improve data integrity, transaction speed, and support for high-volume operations and concurrent users.
3. **Networking Capability:** Implementing a network layer would allow for multiple terminals (e.g., separate waiter and kitchen displays) to interact with the same database simultaneously, turning it into a true Point of Sale (POS) system.
4. **Inventory Management:** Integrating the system with a back-end inventory module would allow for automatic stock deduction upon order completion, triggering low-stock alerts to the administrator.

Bibliography

The following resources, including textbooks, reference manuals, and online tutorials, were consulted and utilized for the research, analysis, design, and implementation of the **Restaurant Order Management System**.

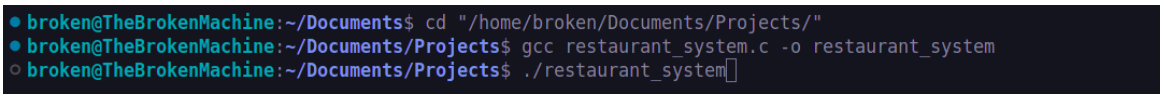
Books and Textbooks

1. **Kanetkar, Y. (2018).** *Let Us C* (16th ed.). BPB Publications.
    - *(Reference for core C programming concepts, structures, pointers, and file management techniques.)*
2. **Balagurusamy, E. (2019).** *Programming in ANSI C* (8th ed.). McGraw Hill Education.
    - *(Reference for algorithmic development and standard C library functions.)*
3. **NEB Curriculum (2080).** *Computer Science Textbook, Grade XII*. Janak Education Materials Centre Ltd.
    - *(Reference for project report structuring and core software development life cycle (SDLC) concepts.)*

Websites and Online Resources

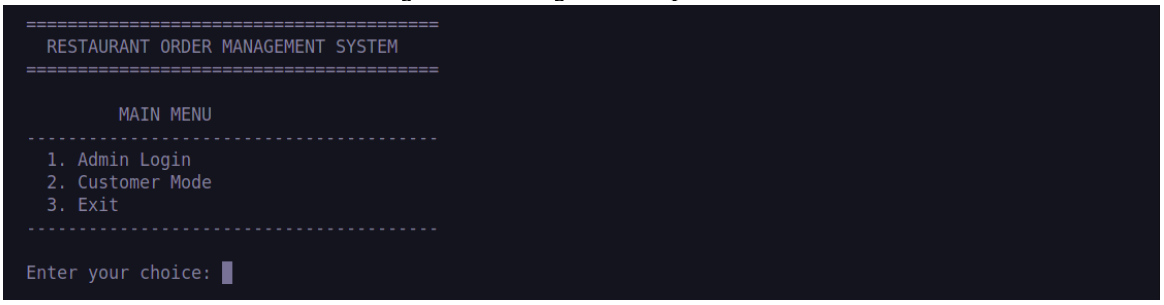
4. **TutorialsPoint.** *C - File Handling*. Available at: `https://www.tutorialspoint.com/cprogramming/c_file_io.htm`
    - *(Consulted for the implementation of read/write operations and the logic for the temporary file mechanism used in item update/deletion.)*
5. **GeeksforGeeks.** *C Programming Articles*. Available at: `https://www.geeksforgeeks.org/c-programming-language/`
    - *(Consulted for specific function syntax, particularly for time/date functions (`time.h`) and string manipulation (`string.h`).)*
6. **CPlusPlus.com.** *C Standard Library Reference*. Available at: `http://www.cplusplus.com/reference/clibrary/`
    - *(Used as a reference for verifying the behavior and parameters of standard I/O functions like `sscanf()` and `fprintf()`.)*

User Manual & Snapshots



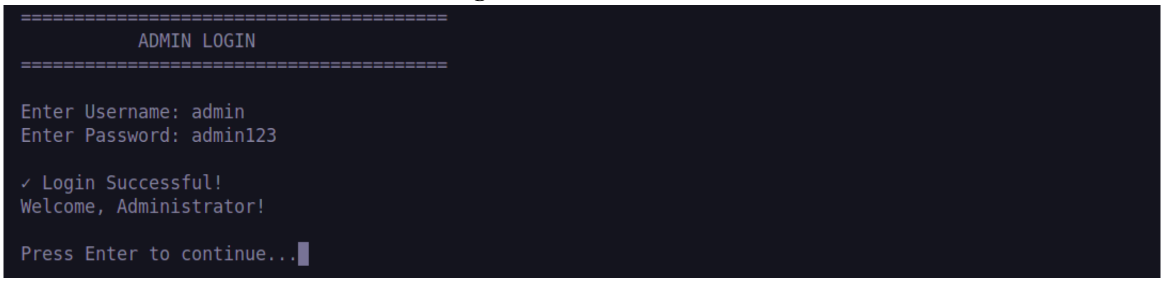
```
● broken@TheBrokenMachine:~/Documents$ cd "/home/broken/Documents/Projects/"
● broken@TheBrokenMachine:~/Documents/Projects$ gcc restaurant_system.c -o restaurant_system
○ broken@TheBrokenMachine:~/Documents/Projects$ ./restaurant_system
```

Figure-3: *Navigate, Compile, Run*



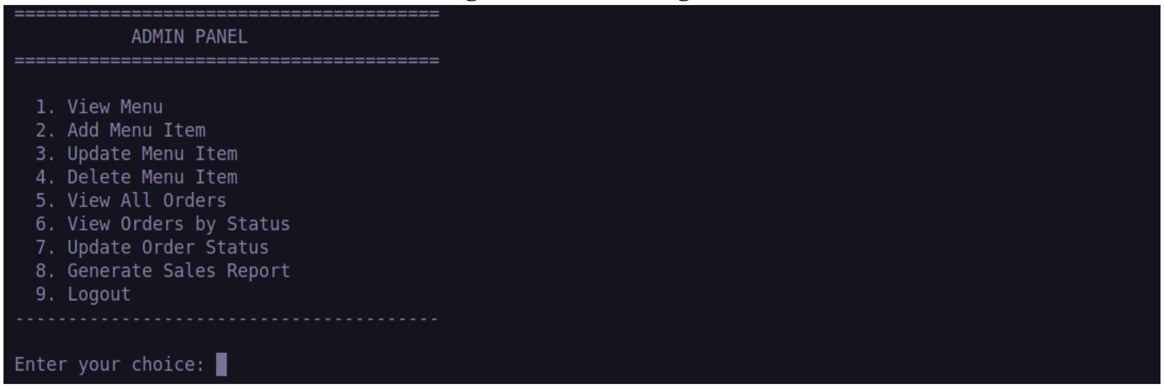
```
========================================
  RESTAURANT ORDER MANAGEMENT SYSTEM
========================================

         MAIN MENU
----------------------------------------
  1. Admin Login
  2. Customer Mode
  3. Exit
----------------------------------------

Enter your choice:
```

Figure-4: *Main Menu*



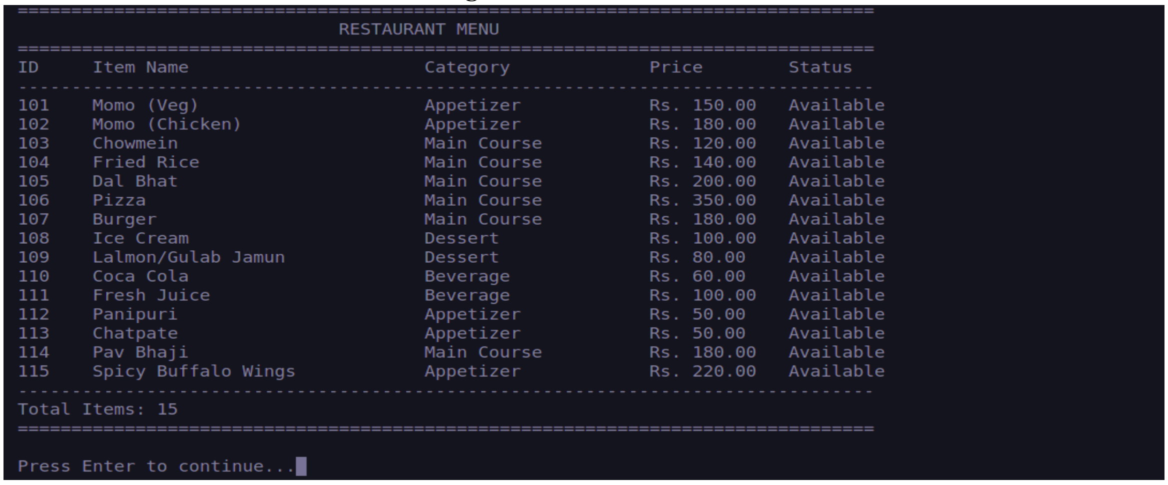
```
========================================
           ADMIN LOGIN
========================================

Enter Username: admin
Enter Password: admin123

✓ Login Successful!
Welcome, Administrator!

Press Enter to continue...
```

Figure-5: *Admin Login*



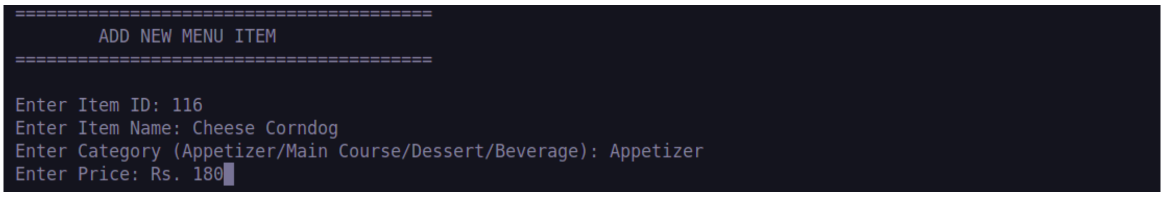
```
========================================
           ADMIN PANEL
========================================

  1. View Menu
  2. Add Menu Item
  3. Update Menu Item
  4. Delete Menu Item
  5. View All Orders
  6. View Orders by Status
  7. Update Order Status
  8. Generate Sales Report
  9. Logout
----------------------------------------

Enter your choice:
```

Figure-6: *Admin Panel*



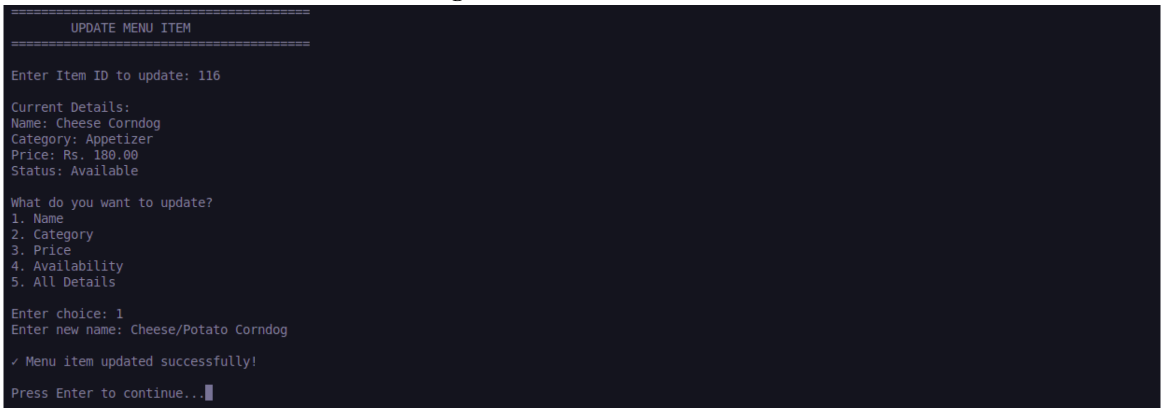
```
================================================================================
                              RESTAURANT MENU
================================================================================
ID     Item Name                      Category             Price         Status
--------------------------------------------------------------------------------
101    Momo (Veg)                     Appetizer            Rs. 150.00    Available
102    Momo (Chicken)                 Appetizer            Rs. 180.00    Available
103    Chowmein                       Main Course          Rs. 120.00    Available
104    Fried Rice                     Main Course          Rs. 140.00    Available
105    Dal Bhat                       Main Course          Rs. 200.00    Available
106    Pizza                          Main Course          Rs. 350.00    Available
107    Burger                         Main Course          Rs. 180.00    Available
108    Ice Cream                      Dessert              Rs. 100.00    Available
109    Lalmon/Gulab Jamun             Dessert              Rs. 80.00     Available
110    Coca Cola                      Beverage             Rs. 60.00     Available
111    Fresh Juice                    Beverage             Rs. 100.00    Available
112    Panipuri                       Appetizer            Rs. 50.00     Available
113    Chatpate                       Appetizer            Rs. 50.00     Available
114    Pav Bhaji                      Main Course          Rs. 180.00    Available
115    Spicy Buffalo Wings            Appetizer            Rs. 220.00    Available
--------------------------------------------------------------------------------
Total Items: 15
================================================================================

Press Enter to continue...
```

Figure-7: *View Menu (Admin)*



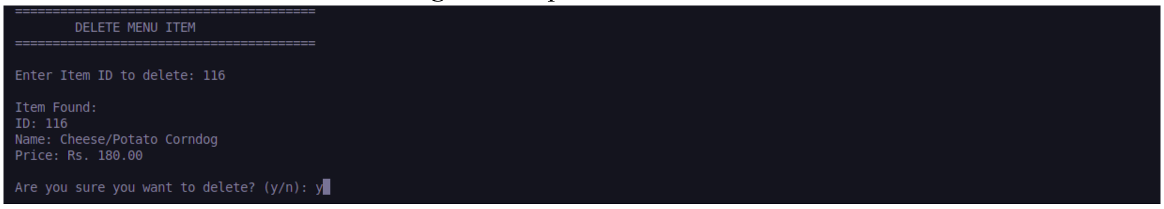
```
========================================
        ADD NEW MENU ITEM
========================================

Enter Item ID: 116
Enter Item Name: Cheese Corndog
Enter Category (Appetizer/Main Course/Dessert/Beverage): Appetizer
Enter Price: Rs. 180
```

Figure-8: *Add Menu Item*



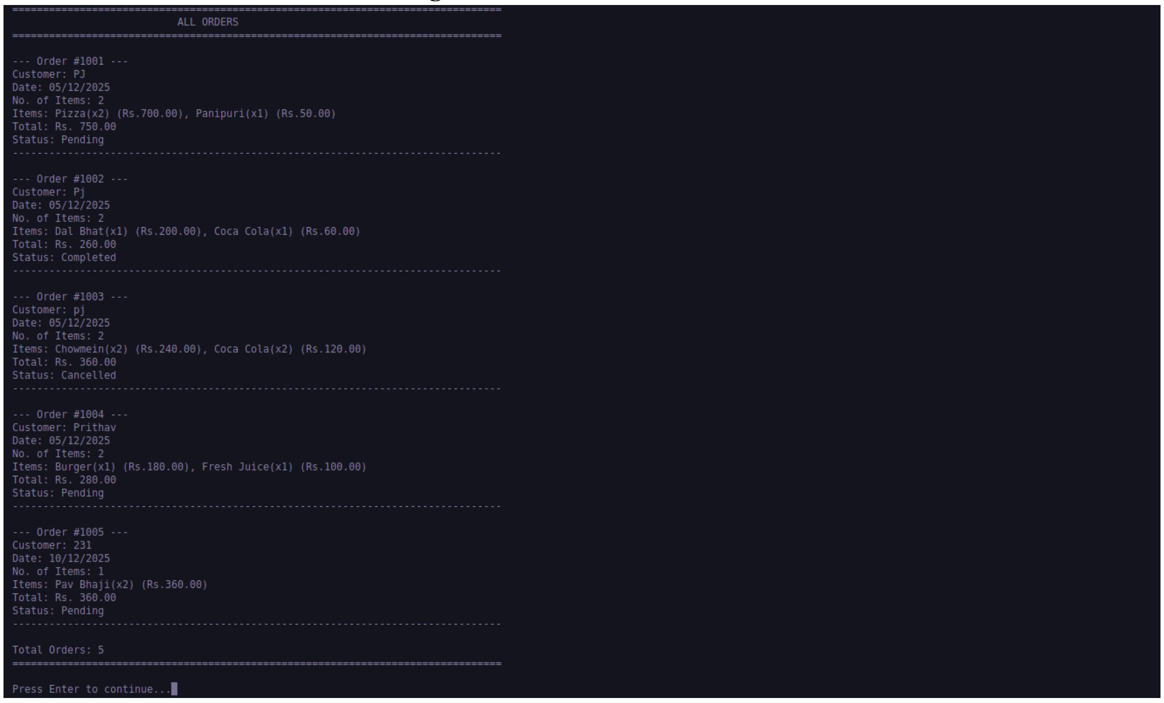
```
========================================
        UPDATE MENU ITEM
========================================

Enter Item ID to update: 116

Current Details:
Name: Cheese Corndog
Category: Appetizer
Price: Rs. 180.00
Status: Available

What do you want to update?
1. Name
2. Category
3. Price
4. Availability
5. All Details

Enter choice: 1
Enter new name: Cheese/Potato Corndog

✓ Menu item updated successfully!

Press Enter to continue...
```

Figure-9: *Update Menu Item*



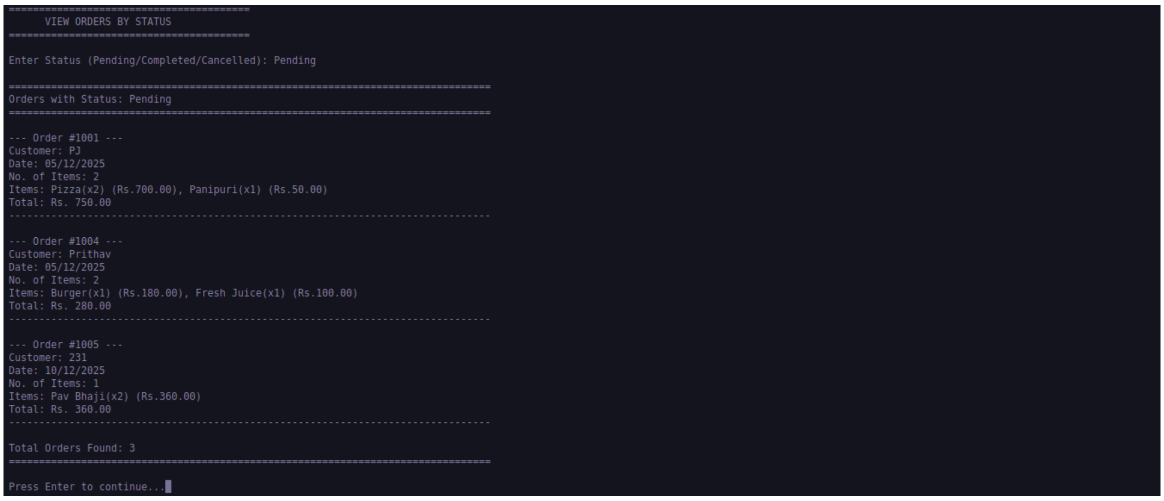
```
========================================
        DELETE MENU ITEM
========================================

Enter Item ID to delete: 116

Item Found:
ID: 116
Name: Cheese/Potato Corndog
Price: Rs. 180.00

Are you sure you want to delete? (y/n): y
```

Figure-10: *Delete Menu Item*



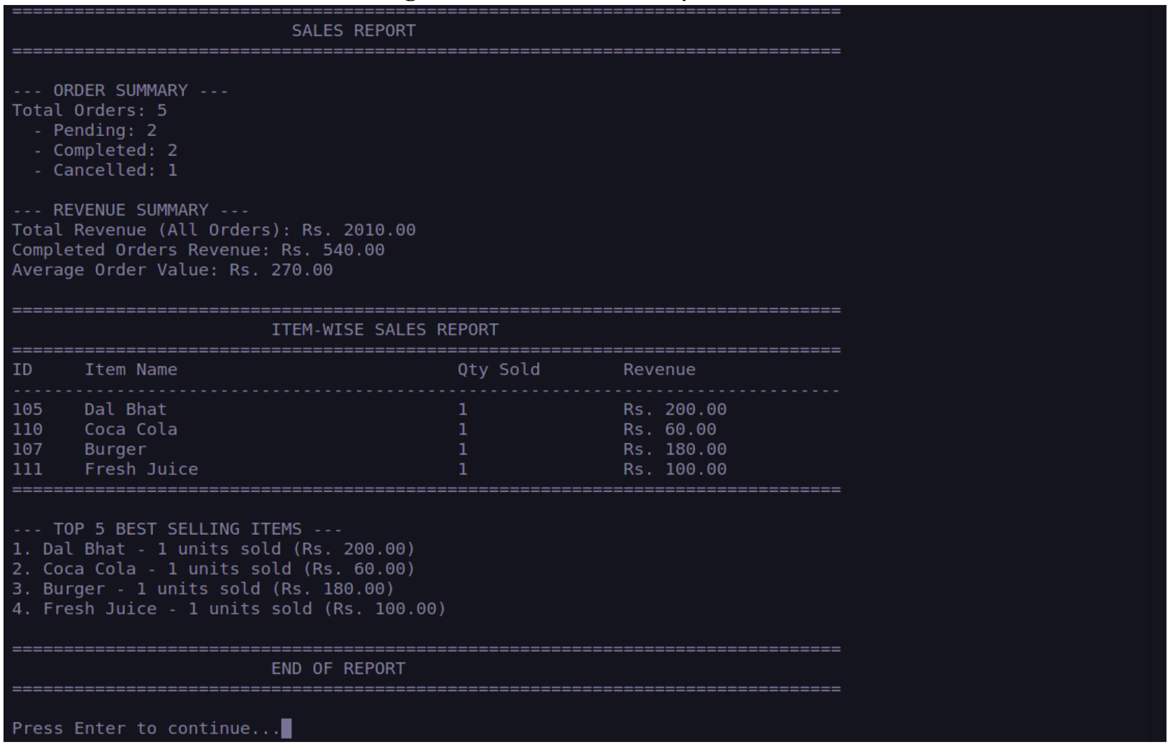
```
================================================================================
                           ALL ORDERS
================================================================================

--- Order #1001 ---
Customer: PJ
Date: 05/12/2025
No. of Items: 2
Items: Pizza(x2) (Rs.700.00), Panipuri(x1) (Rs.50.00)
Total: Rs. 750.00
Status: Pending
--------------------------------------------------------------------------------

--- Order #1002 ---
Customer: Pj
Date: 05/12/2025
No. of Items: 2
Items: Dal Bhat(x1) (Rs.200.00), Coca Cola(x1) (Rs.60.00)
Total: Rs. 260.00
Status: Completed
--------------------------------------------------------------------------------

--- Order #1003 ---
Customer: pj
Date: 05/12/2025
No. of Items: 2
Items: Chowmein(x2) (Rs.240.00), Coca Cola(x2) (Rs.120.00)
Total: Rs. 360.00
Status: Cancelled
--------------------------------------------------------------------------------

--- Order #1004 ---
Customer: Prithav
Date: 05/12/2025
No. of Items: 2
Items: Burger(x1) (Rs.180.00), Fresh Juice(x1) (Rs.100.00)
Total: Rs. 280.00
Status: Pending
--------------------------------------------------------------------------------

--- Order #1005 ---
Customer: 231
Date: 10/12/2025
No. of Items: 1
Items: Pav Bhaji(x2) (Rs.360.00)
Total: Rs. 360.00
Status: Pending
--------------------------------------------------------------------------------

Total Orders: 5
================================================================================

Press Enter to continue...
```

Figure-11: *View All Orders*



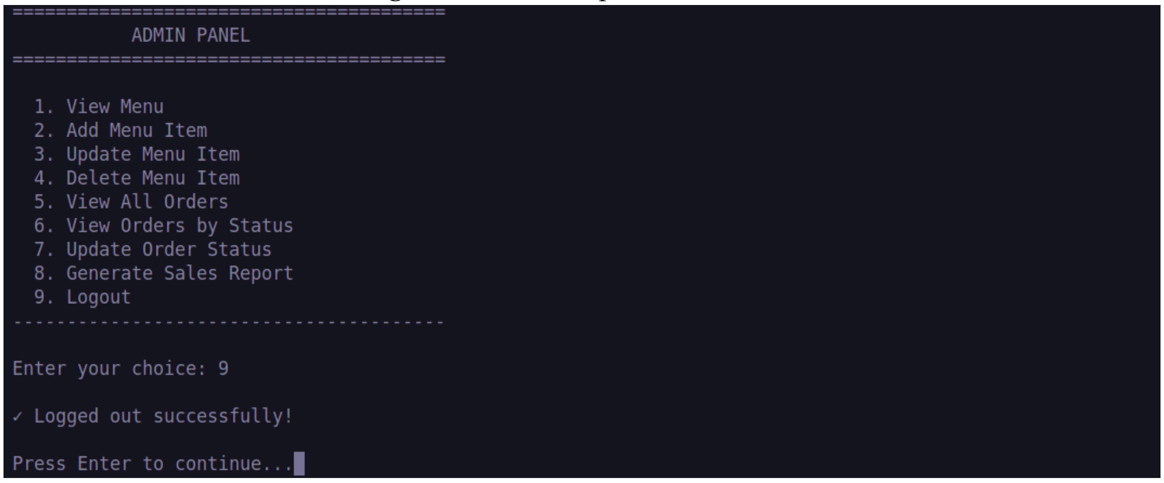
```
========================================
      VIEW ORDERS BY STATUS
========================================

Enter Status (Pending/Completed/Cancelled): Pending

================================================================================
Orders with Status: Pending
================================================================================

--- Order #1001 ---
Customer: PJ
Date: 05/12/2025
No. of Items: 2
Items: Pizza(x2) (Rs.700.00), Panipuri(x1) (Rs.50.00)
Total: Rs. 750.00
--------------------------------------------------------------------------------

--- Order #1004 ---
Customer: Prithav
Date: 05/12/2025
No. of Items: 2
Items: Burger(x1) (Rs.180.00), Fresh Juice(x1) (Rs.100.00)
Total: Rs. 280.00
--------------------------------------------------------------------------------

--- Order #1005 ---
Customer: 231
Date: 10/12/2025
No. of Items: 1
Items: Pav Bhaji(x2) (Rs.360.00)
Total: Rs. 360.00
--------------------------------------------------------------------------------

Total Orders Found: 3
================================================================================

Press Enter to continue...
```

Figure-12: *View Orders by Status*



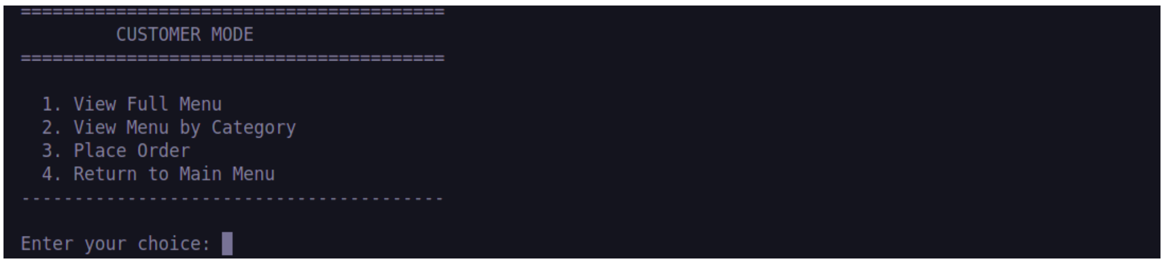
```
================================================================================
                           SALES REPORT
================================================================================

--- ORDER SUMMARY ---
Total Orders: 5
  - Pending: 2
  - Completed: 2
  - Cancelled: 1

--- REVENUE SUMMARY ---
Total Revenue (All Orders): Rs. 2010.00
Completed Orders Revenue: Rs. 540.00
Average Order Value: Rs. 270.00

================================================================================
                         ITEM-WISE SALES REPORT
================================================================================
ID     Item Name                                   Qty Sold        Revenue
--------------------------------------------------------------------------------
105    Dal Bhat                                    1               Rs. 200.00
110    Coca Cola                                   1               Rs. 60.00
107    Burger                                      1               Rs. 180.00
111    Fresh Juice                                 1               Rs. 100.00
================================================================================

--- TOP 5 BEST SELLING ITEMS ---
1. Dal Bhat - 1 units sold (Rs. 200.00)
2. Coca Cola - 1 units sold (Rs. 60.00)
3. Burger - 1 units sold (Rs. 180.00)
4. Fresh Juice - 1 units sold (Rs. 100.00)

================================================================================
                         END OF REPORT
================================================================================

Press Enter to continue...
```

Figure-13: *Sales Report Generation*



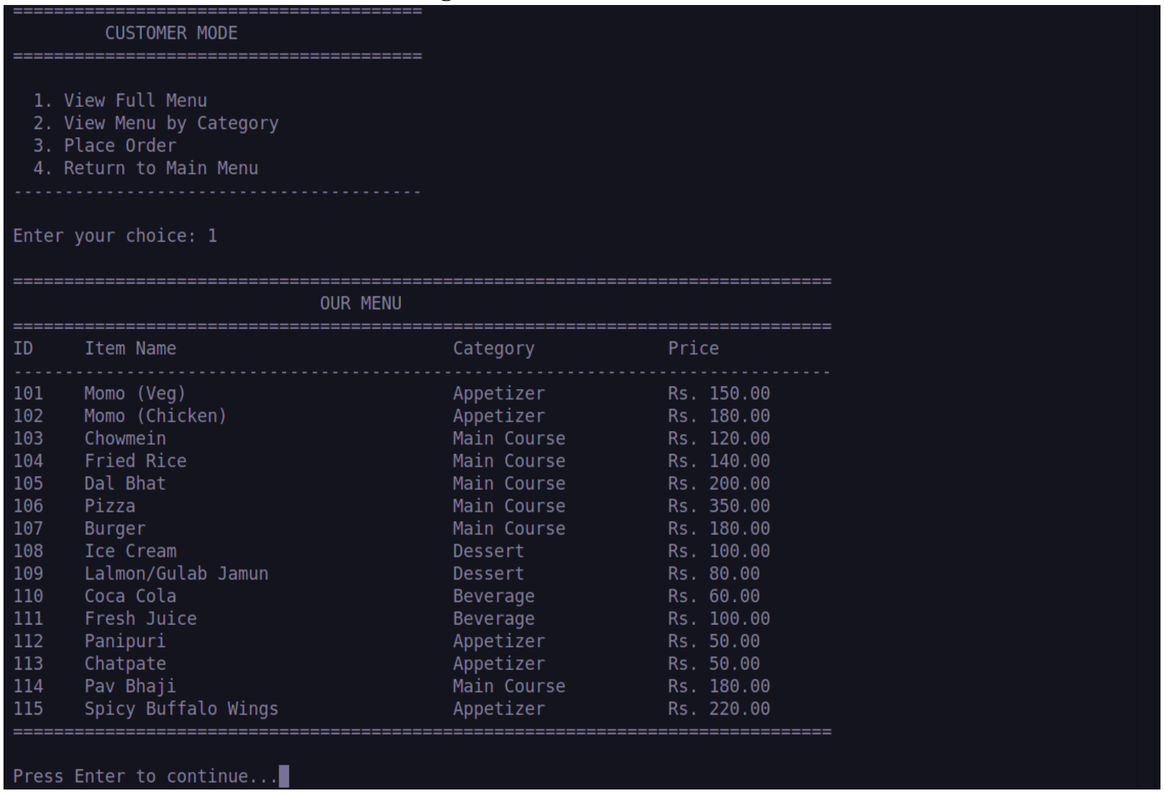
```
========================================
           ADMIN PANEL
========================================

  1. View Menu
  2. Add Menu Item
  3. Update Menu Item
  4. Delete Menu Item
  5. View All Orders
  6. View Orders by Status
  7. Update Order Status
  8. Generate Sales Report
  9. Logout
----------------------------------------

Enter your choice: 9

✓ Logged out successfully!

Press Enter to continue...
```

Figure-14: *Admin Logout*



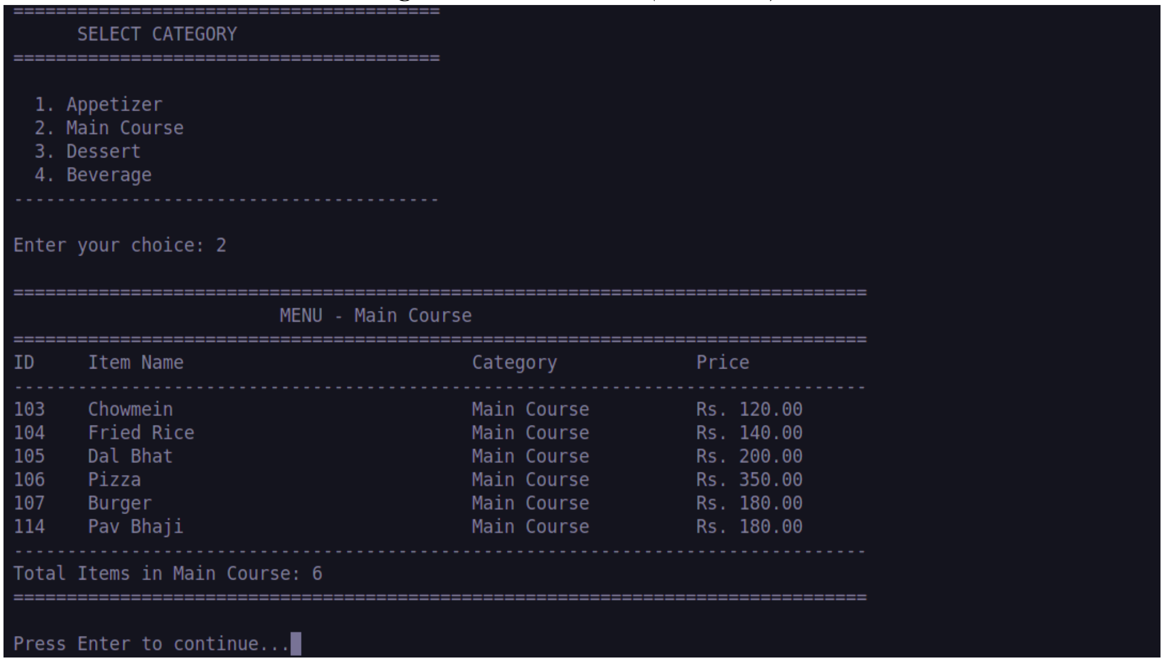
```
========================================
         CUSTOMER MODE
========================================

  1. View Full Menu
  2. View Menu by Category
  3. Place Order
  4. Return to Main Menu
----------------------------------------

Enter your choice: 
```

Figure-15: *Customer Mode*



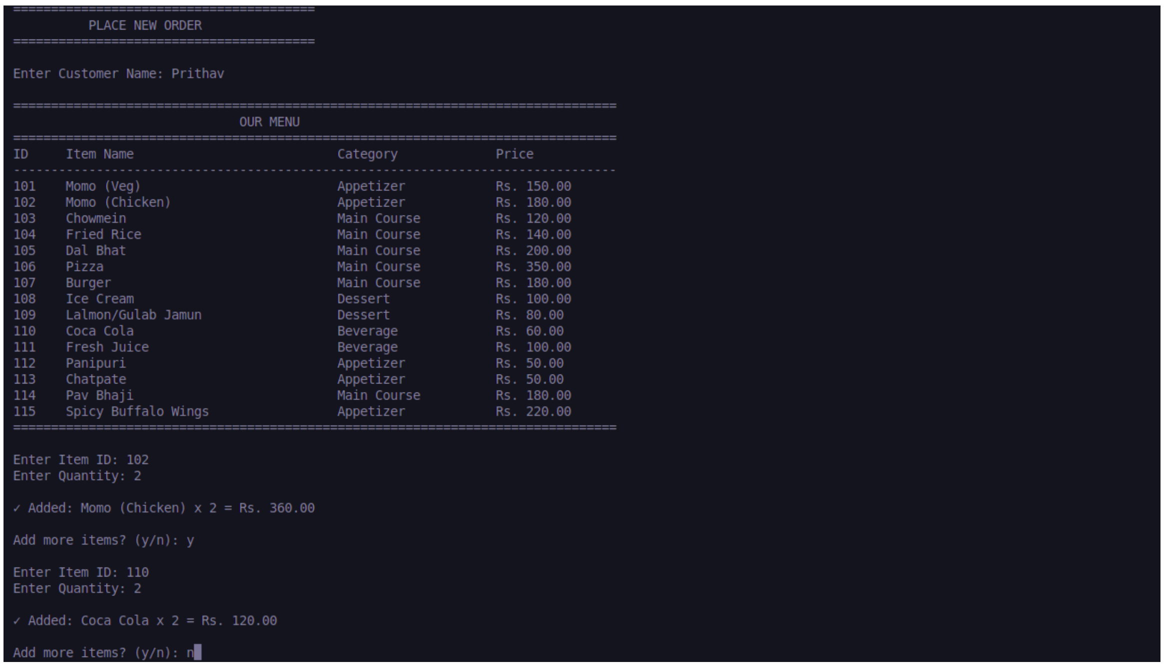
```
========================================
         CUSTOMER MODE
========================================

  1. View Full Menu
  2. View Menu by Category
  3. Place Order
  4. Return to Main Menu
----------------------------------------

Enter your choice: 1

================================================================================
                              OUR MENU
================================================================================
ID     Item Name                           Category            Price
--------------------------------------------------------------------------------
101    Momo (Veg)                          Appetizer           Rs. 150.00
102    Momo (Chicken)                      Appetizer           Rs. 180.00
103    Chowmein                            Main Course         Rs. 120.00
104    Fried Rice                          Main Course         Rs. 140.00
105    Dal Bhat                            Main Course         Rs. 200.00
106    Pizza                               Main Course         Rs. 350.00
107    Burger                              Main Course         Rs. 180.00
108    Ice Cream                           Dessert             Rs. 100.00
109    Lalmon/Gulab Jamun                  Dessert             Rs. 80.00
110    Coca Cola                           Beverage            Rs. 60.00
111    Fresh Juice                         Beverage            Rs. 100.00
112    Panipuri                            Appetizer           Rs. 50.00
113    Chatpate                            Appetizer           Rs. 50.00
114    Pav Bhaji                           Main Course         Rs. 180.00
115    Spicy Buffalo Wings                 Appetizer           Rs. 220.00
================================================================================

Press Enter to continue...
```

Figure-16: *View Menu (Customer)*



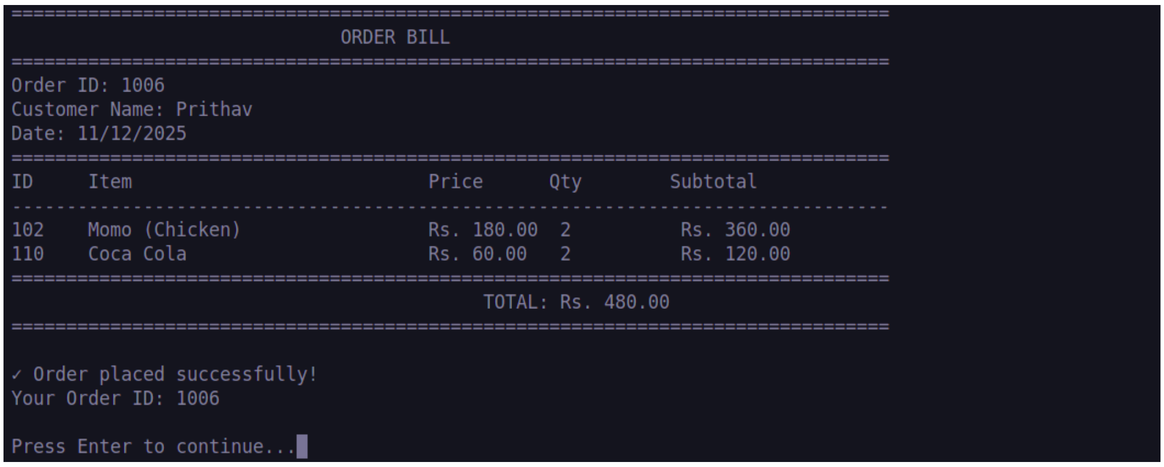
```
========================================
      SELECT CATEGORY
========================================

  1. Appetizer
  2. Main Course
  3. Dessert
  4. Beverage
----------------------------------------

Enter your choice: 2

================================================================================
                         MENU - Main Course
================================================================================
ID     Item Name                           Category            Price
--------------------------------------------------------------------------------
103    Chowmein                            Main Course         Rs. 120.00
104    Fried Rice                          Main Course         Rs. 140.00
105    Dal Bhat                            Main Course         Rs. 200.00
106    Pizza                               Main Course         Rs. 350.00
107    Burger                              Main Course         Rs. 180.00
114    Pav Bhaji                           Main Course         Rs. 180.00
--------------------------------------------------------------------------------
Total Items in Main Course: 6
================================================================================

Press Enter to continue...
```

Figure-17: *View Menu by Category*



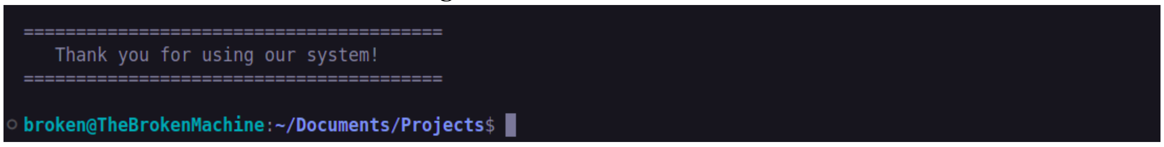
```
========================================
          PLACE NEW ORDER
========================================

Enter Customer Name: Prithav

================================================================================
                              OUR MENU
================================================================================
ID     Item Name                           Category             Price
--------------------------------------------------------------------------------
101    Momo (Veg)                          Appetizer            Rs. 150.00
102    Momo (Chicken)                      Appetizer            Rs. 180.00
103    Chowmein                            Main Course          Rs. 120.00
104    Fried Rice                          Main Course          Rs. 140.00
105    Dal Bhat                            Main Course          Rs. 200.00
106    Pizza                               Main Course          Rs. 350.00
107    Burger                              Main Course          Rs. 180.00
108    Ice Cream                           Dessert              Rs. 100.00
109    Lalmon/Gulab Jamun                  Dessert              Rs. 80.00
110    Coca Cola                           Beverage             Rs. 60.00
111    Fresh Juice                         Beverage             Rs. 100.00
112    Panipuri                            Appetizer            Rs. 50.00
113    Chatpate                            Appetizer            Rs. 50.00
114    Pav Bhaji                           Main Course          Rs. 180.00
115    Spicy Buffalo Wings                 Appetizer            Rs. 220.00
================================================================================

Enter Item ID: 102
Enter Quantity: 2

✓ Added: Momo (Chicken) x 2 = Rs. 360.00

Add more items? (y/n): y

Enter Item ID: 110
Enter Quantity: 2

✓ Added: Coca Cola x 2 = Rs. 120.00

Add more items? (y/n): n
```

Figure-18: *Place Order*



```
================================================================================
                              ORDER BILL
================================================================================
Order ID: 1006
Customer Name: Prithav
Date: 11/12/2025
================================================================================
ID     Item                           Price      Qty        Subtotal
--------------------------------------------------------------------------------
102    Momo (Chicken)                 Rs. 180.00  2          Rs. 360.00
110    Coca Cola                      Rs. 60.00   2          Rs. 120.00
================================================================================
                                           TOTAL: Rs. 480.00
================================================================================

✓ Order placed successfully!
Your Order ID: 1006

Press Enter to continue...
```

Figure-19: *Bill Calculation*



```
  ========================================
     Thank you for using our system!
  ========================================

○ broken@TheBrokenMachine:~/Documents/Projects$
```

Figure-20: *Greeting after Exit*

Team Contribution

As per the project assignment, this report documents the development of the **Restaurant Order Management System**. This project was completed by a single individual.

Prithav Jha
Symbol No.: 812830
Shift: Day
Faculty: Science
Grade: XII
Section: D2
Roll No.: 23

Total Contribution: 100%

Detailed Contribution Statement:

The entire development life cycle of the **Restaurant Order Management System** was handled individually. The contribution spanned all technical and documentation phases, including:

- **Analysis and Design (Chapter 3):** Conceptualization of the system, requirement gathering, determining the feasibility study, and designing the data structures (`struct MenuItem`, `struct Order`) and file formats (`menu.txt`, `orders.txt`).
- **Implementation (Chapter 4):** Writing the complete source code (`restaurant_system.c`) in C language, including all core functionalities (Admin login, Menu Management, Order Placement, Sales Reporting), and implementing the file handling logic.
- **Testing and Validation (Chapter 4):** Development and execution of all unit and system-level test cases to ensure the accuracy of billing, data integrity, and error handling.
- **Documentation:** Drafting and compiling all chapters of the project report, generating diagrams, collecting screenshots, and adhering to all formatting standards specified in the project guidelines.

THE END