Smart Restaurant Management System

Submitted By

Student Name	Student ID
Sk. Azmain Zunaeid	251-15-222
Md. Delower Sarker	251-15-457
Md. Maruf Mia	251-15-042
Syeda Zawada Farah	251-15-223
Maher Afruz Mira	251-15-017

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the Course
CSE124: Data Structure Lab
in the Computer Science and Engineering Department

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

December 13, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Mr. Md. Jakaria Zobair**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

_____

Mr. Md. Jakaria Zobair
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted By

_____ **Sk. Azmain Zunaeid (Leader)** 251-15-222 Dept. of CSE, DIU	
_____ **Md. Delower Sarker** 251-15-457 Dept. of CSE, DIU	_____ **Md. Maruf Mia** 251-15-042 Dept. of CSE, DIU
_____ **Syeda Zawada Farah** 251-15-223 Dept. of CSE, DIU	_____ **Maher Afruz Mira** 251-15-017 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as follows:

Table 1: Course Outcome Statements

CO's	Statements
CO1	Design the concept of fundamentals of data structure for solving real-life problem having complex engineering attributes.
CO2	Solve a real-life problem having application of abstract data type created within the scope of complex engineering problem solving with the apply modern tools.
CO3	Apply Data Structure to solve real world problems member in diverse team.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	LD	KP	CEP
CO1	PO3	PO1, PO2, C3	K5	EP1
CO2	PO5	P2, C3	K6	EP2, EP6
CO3	PO9	P3, A4	N/A	N/A

The mapping justification of this table is provided in sections **4.3.1**, **4.3.2,** and **4.3.3**.

Table of Contents

Declaration . . . . . **i**

Course & Program Outcome . . . . . **ii**

1 Introduction **1**
- 1.1 Introduction 1
- 1.2 Motivation 1
- 1.3 Objectives 2
- 1.4 Feasibility Study 2
- 1.5 Gap Analysis 3
- 1.6 Project Outcome 3

2 Proposed Methodology/Architecture **4**
- 2.1 Requirement Analysis & Design Specification 4
  - 2.1.1 Overview 7
  - 2.1.2 Proposed Methodology/ System Design 9
  - 2.1.3 UI Design 10
- 2.2 Overall Project Plan 11

3 Implementation and Results **12**
- 3.1 Implementation 12
- 3.2 Performance Analysis 38
- 3.3 Results and Discussion 39

4 Engineering Standards and Mapping **53**
- 4.1 Impact on Society, Environment and Sustainability 53
  - 4.1.1 Impact on Life 53
  - 4.1.2 Impact on Society & Environment 53
  - 4.1.3 Ethical Aspects 54
  - 4.1.4 Sustainability Plan 54
- 4.2 Project Management and Team Work 55
- 4.3 Complex Engineering Problem 56
  - 4.3.1 Mapping of Program Outcome 56
  - 4.3.2 Complex Problem Solving 57
  - 4.3.3 Engineering Activities 58

5 Conclusion **59**
5.1 Summary 59
5.2 Limitation 60
5.3 Future Work 60

References **61**

Chapter 1

Introduction

This chapter provides a comprehensive overview of the **Smart Restaurant Management System**, detailing its purpose, motivational factors, specific objectives, feasibility considerations, identified gaps in existing solutions, and expected project outcomes.

1.1 Introduction

Smart Restaurant Management System is a console-based application developed in ANSI C that implements a comprehensive restaurant management solution. The system provides two distinct operational modes: Manager Mode for menu administration and order processing, and Customer Mode for menu viewing and placing orders.

The Manager module maintains menu items using a singly linked list data structure with persistent storage in menu.txt, supporting full CRUD (Create, Read, Update, Delete) operations. These operations include inserting items at the beginning, at the end, or after a specific node within the list, as well as deleting items from any position, updating existing entries, and searching for items by ID. All changes are automatically synchronized with the persistent text file, ensuring data integrity across sessions.

The Customer module handles order placement through a queue-based pending order system, where each order node represents a dining table and contains a nested linked list of ordered items. This design allows customers to place multi-item orders that are logically grouped by table, while the queue structure ensures that orders are processed in the sequence they are received. The manager serves completed orders using a dequeue-type function that follows the FIFO (First-In-First-Out) principle, guaranteeing fair and orderly processing of customer requests.

This project effectively demonstrates the practical application of fundamental data structures while addressing real-world restaurant management challenges. It serves as both a functional prototype and an educational tool, illustrating core computer science concepts—such as dynamic memory management, pointer manipulation, file I/O operations, and algorithmic efficiency—within an accessible, real-world context. The implementation emphasizes modular design, robust input validation, and a clear separation of concerns, making it a valuable resource for understanding how theoretical data structure concepts translate into practical, maintainable software solutions.

1.2 Motivation

The development of this system was motivated by two primary factors:

Pedagogical Motivation: Data structures often remain abstract concepts for computer science students. This project bridges the gap between theoretical knowledge and practical application by implementing **linked lists**, **queues**, and **file handling** in a realistic scenario. Students can observe how these structures operate in a meaningful context, enhancing comprehension and retention.

Practical Motivation: Small-scale restaurants require efficient order management systems that are lightweight and resource-efficient. By developing a console-based solution in C, the project demonstrates how fundamental programming concepts can solve real-world problems without the overhead of complex frameworks or expensive software.

1.3 Objectives

The project aims to achieve the following specific objectives:

1. Implement persistent menu management using linked lists with file-based storage
2. Develop a queue-based order system following FIFO principles
3. Create separate user interfaces for Manager and Customer roles
4. Demonstrate data structure concepts through practical, real-world application

1.4 Feasibility Study

The feasibility of **Smart Restaurant Management System** was assessed through multiple dimensions:

Previous Works: Several earlier projects have explored data-structure-based restaurant management solutions. One notable example is the **Restaurant Order Processing System (DataFlair) [1]**, which uses singly linked lists and queue operations to store incoming orders, maintain a dynamic list of pending items, and process them sequentially in FIFO order. This work demonstrates how basic data structures can be applied effectively to build a simple, efficient, and scalable order-management system.

Technical Feasibility: The project utilizes standard C libraries (stdio.h, stdlib.h, string.h) without external dependencies, making it highly portable and compatible with any ANSI C compliant compiler. The console-based interface eliminates complex GUI requirements.

Resource Feasibility: Minimal hardware requirements (1GB RAM, standard processor) ensure accessibility. The text-based storage system requires negligible disk space while maintaining data persistence.

Operational Feasibility: The hierarchical menu system provides intuitive navigation. Input validation and error handling mechanisms ensure robust operation even with invalid user inputs.

Economic Feasibility: As an educational prototype, development costs are minimal. The system demonstrates concepts that could scale to commercial POS systems with additional investment.

1.5 Gap Analysis

Current educational tools and commercial systems exhibit several gaps that this project addresses:

Identified Gaps in Prior Projects: Earlier restaurant management systems using linked lists and queues mainly focused on basic order handling and simple menu storage. They often lacked features such as **table-wise order tracking**, **multi-item orders**, and **persistent data storage**.
This project overcomes these limitations by implementing a complete menu-management module with CRUD operations, a nested queue for table-specific orders, persistent storage via files, mode-based user interaction (Manager and Customer), and input validation.

Educational Tools Gap: Most data structure visualizers focus on algorithm animation but neglect practical aspects like file persistence and real-world constraints. **Smart Restaurant Management System** integrates both theoretical concepts and practical implementation challenges.

Commercial Systems Gap: Existing POS systems are often feature-heavy and implemented in high-level languages, obscuring the fundamental data structure operations. This project maintains transparency in demonstrating core computer science concepts.

Implementation Gap: Many educational projects use abstract examples, whereas **Smart Restaurant Management System** models a realistic restaurant scenario with nested data structures (order nodes containing item lists), providing a more comprehensive learning experience.

1.6 Project Outcome

The project delivers several significant outcomes:

1. Working menu system with full CRUD operations and automatic saving to/loading from menu.txt
2. Functional order queue where orders are stored and served in sequence, demonstrated through serveNextOrder()
3. Two operational modes with Manager authentication and distinct workflows for admin tasks and customer ordering
4. Educational prototype showing linked lists, queues, memory management, and file I/O in a tangible restaurant context

Chapter 2

Proposed Methodology/Architecture

This chapter details the system requirements, architectural design, user interface specifications, and project planning methodology employed in the development of **Smart Restaurant Management System**.

2.1 Requirement Analysis & Design Specification

Hardware Requirements

- **Processor:** 1 GHz or higher
- **Memory:** 1 GB RAM minimum
- **Storage:** 10 MB available space
- **Input:** Standard keyboard
- **Output:** Console/terminal display

Software Requirements

- **Operating System:** Windows/Linux/macOS
- **Compiler:** GCC, Clang, or any ANSI C compliant compiler
- **Dependencies:** Standard C libraries only (no external dependencies)

Functional Requirements - The functions and features the system must have

- **Authentication System:** Password-protected manager access
- **Menu Management:** Complete CRUD operations on food items
- **Order Processing:** Table-based order placement and management
- **Persistence:** Automatic saving and loading of menu data
- **Order Queue:** FIFO-based order serving system
- **Input Validation:** Robust handling of invalid user inputs

Non-Functional Requirements - The quality, performance, and constraints of the system

- **Performance:** Responsive operation with immediate feedback
- **Reliability:** Consistent operation without crashes
- **Usability:** Intuitive menu-driven interface
- **Maintainability:** Well-documented, modular code structure
- **Memory Safety:** Proper allocation and deallocation of dynamic memory

Stakeholder Engagement and Field Research

To ensure our system addressed real-world restaurant needs, we conducted direct stakeholder engagement by visiting a restaurant. This field research provided invaluable insights into actual operational workflows, pain points in existing order management, and practical requirements that informed our system design.

Key insights gathered from stakeholder interviews:

- The importance of quick menu updates during peak hours
- Need for clear table-wise order tracking
- Priority of FIFO order serving during busy periods
- Requirement for simple, error-resistant interfaces for staff

This direct engagement with end-users ensured our project remained grounded in practical requirements rather than theoretical assumptions, significantly enhancing the relevance and usability of our final system.

Figure 2.1: Stakeholder Requirement Gathering
This photograph shows team Binary Brains conducting field research at a local restaurant to understand real-world operational needs and validate system requirements with actual stakeholders.

Figure 2.2: On-site Design Discussion
This photograph captures the team immediately analyzing gathered requirements and brainstorming initial system design concepts while still at the restaurant, ensuring practical insights directly informed the architectural planning.

Figure 2.3: Team Binary Brains at Research Site
This group photograph documents the complete project team at the restaurant after concluding stakeholder interviews and preliminary design discussions, marking the successful completion of the initial requirement gathering phase.

2.1.1 Overview

The system architecture comprises **three** core modules (how the code is divided for better organization):

Menu Management Module: Handles all food item operations using a **singly linked list** structure. Each node contains food ID, name, price, and next pointer. The module interfaces with "menu.txt" for data persistence.

Order Processing Module: Manages customer orders through a **queue** of order nodes. Each order node contains table number and a linked list of ordered items. Implements **FIFO** serving principle.

User Interface Module: Provides console-based interfaces for both **manager** and **customer** modes with hierarchical menu navigation and comprehensive input validation.

Figure 2.4: System Architecture of Our Project
This shows the three internal code Modules powering the two user Modes. Both Modes share the same underlying Modules but have different access and interfaces.

Project Development Work Flow

Figure 2.5: Project Development Work Flow
This diagram shows the step-by-step development process of our project, from initial planning through coding and testing to final documentation and submission.

2.1.2 Proposed Methodology/System Design

Figure 2.6: Function Flow of Our Project
The diagram illustrates the overall functional flow of the **Smart Restaurant Management System**, showing the interaction between the main program, data loading, and operational modes. It highlights how manager and customer operations invoke their respective functions to manage menu items and process orders.

2.1.3 UI Design

The user interface employs a console-based hierarchical menu system:

Main Menu:
- Manager Mode (Password Protected)
- Customer Mode
- Exit System

Manager Interface:
- Menu display and management operations
- Order viewing and serving capabilities
- Comprehensive CRUD operations

Customer Interface:
- Table selection with validation (1-50)
- Menu browsing and order placement
- Pending order viewing

UI Characteristics:
- Consistent formatting and clear section headers
- Informative prompts and error messages
- Responsive feedback for all user actions
- Input validation with helpful error guidance

```
##################################################
--------------------------------------------------
|       SMART RESTAURANT MANAGEMENT SYSTEM       |
--------------------------------------------------
##################################################

1. MANAGER
2. CUSTOMER
3. EXIT

Enter Choice (1-3):
```

Figure 2.7: Main Menu

```
##################################################
--------------------------------------------------
|       SMART RESTAURANT MANAGEMENT SYSTEM       |
--------------------------------------------------
##################################################

1. MANAGER
2. CUSTOMER
3. EXIT

Enter Choice (1-3): 1
Enter The Password: 222

========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13):
```

Figure 2.8: Manager Interface

```
##################################################
--------------------------------------------------
|       SMART RESTAURANT MANAGEMENT SYSTEM       |
--------------------------------------------------
##################################################

1. MANAGER
2. CUSTOMER
3. EXIT

Enter Choice (1-3): 2
Enter Your Table Number (1-50): 1
Table Number 1 Selected.

========= CUSTOMER MODE =========
1. Show The Restaurant Menu
2. Select & Order Items
3. View ALL Pending Orders
4. Back To Main Menu

Enter Your Choice (1-4):
```

Figure 2.9: Customer Interface

2.2 Overall Project Plan

The project followed a structured development timeline:

Phase 1: Requirements & Design (1 Week)
- Requirement analysis and specification
- Data structure design and system architecture
- Interface design and user flow mapping

Phase 2: Core Implementation (4 Weeks)
- Linked list implementation for menu management
- File I/O operations for data persistence
- Order queue implementation
- Basic user interfaces

Phase 3: Validation & Enhancement (2 Weeks)
- Comprehensive input validation
- Error handling mechanisms
- Memory management optimization
- Edge case handling

Phase 4: Testing & Debugging (1 Week)
- Unit testing of individual modules
- Integration testing
- User acceptance testing
- Performance optimization

Phase 5: Documentation & Finalization (2 Weeks)
- Code documentation and commenting
- Report preparation
- Presentation development
- Final submission preparation

Table 2.1: Overall Project Plan

Weeks	Phases
Week 1	P-1- Requirements & Design
Week 2-5	P-2- Core Implementation
Week 6-7	P-3- Validation & Enhancement
Week 8	P-4- Testing & Debugging
Week 9-10	P-5- Documentation & Finalization

Total: 10 Weeks

Chapter 3

Implementation and Results

This chapter provides detailed insights into the system implementation, performance characteristics, and operational results of the **Smart Restaurant Management System**.

3.1 Implementation

The system was implemented as **a single-file C program** (SRMS_CSE124_Project_Binary_Brains_68_A1_DIU.c) following ANSI C standards. The implementation emphasizes modularity, memory safety, and robust error handling.

Development Environment

- **Compiler:** GCC (GNU Compiler Collection)
- **IDE:** Visual Studio Code with C/C++ extension
- **Debugger:** GDB for debugging sessions
- **Platform:** Cross-platform (Windows/Linux/macOS compatible)

Source Code:

```c
/* ################### SRMS CSE124 Project Binary Brains 68_A1 DIU ################### */

/* ---------------- Libraries (3) - Both ---------------- */

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

```

```
/* ---------------- Structures (3) - Both ---------------- */

struct food // For the descriptions of the menu items; For the structure of the menu
{
    char id[20];
    char name[30];
    float price;
    int stockCount; // NEW ADDED CODE: For managing stock quantity
    struct food *next;
};

struct food *start = NULL;

struct orderItem // For the order of a particular table
{
    char foodID[20];
    int qty;
    struct orderItem *next;
};

struct orderNode // For the queue of the tables gave order
{
    int tableNo;
    struct orderItem *items;
    struct orderNode *next;
};

struct orderNode *pendingOrders = NULL;

```

```c
/* ---------------- File Operations (2) - Both ---------------- */

void saveData() // For saving all the operational data permanently
{
    FILE *fp = fopen("menu.txt", "w");
    if(fp == NULL)
        return;

    struct food *i = start;
    while(i != NULL)
    {
        // UPDATED CODE: Added stockCount to save file
        fprintf(fp, "%s;%s;%.2f;%d\n", i->id, i->name, i->price, i->stockCount);
        i = i->next;
    }
    fclose(fp);
}

void loadData() // For loading all the previous operational data just after starting the program
{
    FILE *fp = fopen("menu.txt", "r");
    if(fp == NULL) return;

    char line[200];
    while(fgets(line, sizeof(line), fp))
    {
        char id[20], name[30];
        float price;
        int stock; // NEW ADDED CODE: Variable for reading stock

        // UPDATED CODE: Reading 4 items now including stock
        if(sscanf(line, "%19[^;];%29[^;];%f;%d", id, name, &price, &stock) == 4)
        {
            struct food *newnode = (struct food*)malloc(sizeof(struct food));
            strcpy(newnode->id, id);
            strcpy(newnode->name, name);
            newnode->price = price;
            newnode->stockCount = stock; // NEW ADDED CODE: Assigning stock
            newnode->next = NULL;

            if(start == NULL)
            {
                start = newnode;
            }
            else
            {
                struct food *t = start;
                while(t->next != NULL) t = t->next;
                t->next = newnode;
            }
        }
    }
    fclose(fp);
}
```

```c
/* ################### SINGLY LINKED LIST OPERATIONS (10) ################### */

/* ---------------- Menu Print (1) - Both ---------------- */

void printLinkList() // For printing the menu
{
    printf("\n-------------------- Smart Restaurant Menu --------------------\n");
    struct food *i = start;
    if(i == NULL)
    {
        printf("\nNo items available.\n");
        printf("\n-------------------------------------------------------------\n");
        return;
    }
    while(i != NULL)
    {
        // UPDATED CODE: Added Logic to Show Stock or Stock Out
        if(i->stockCount > 0)
        {
            printf("\nID: %-10s | %-20s | Price: %.2f Tk | Stock: %d\n", i->id, i->name, i->price, i->stockCount);
        }
        else
        {
            printf("\nID: %-10s | %-20s | Price: %.2f Tk | *** STOCK OUT ***\n", i->id, i->name, i->price);
        }
        i = i->next;
    }
    printf("\n-------------------------------------------------------------\n");
}

/* ---------------- Insert Functions for Inserting Food Items in the Menu (3) - Manager ---------------- */

void InsertBeginFood() // For inserting in the beginning of the menu
{
    // UPDATED CODE: ID and Name validation loop
    struct food *newnode = (struct food*)malloc(sizeof(struct food));
    struct food *temp;
    int exists;

    // STEP 1: ID Validation Loop
    while(1)
    {
        printf("Enter Food ID: ");
        scanf("%19s", newnode->id);

        temp = start;
        exists = 0;

        // Check if ID exists
        while(temp != NULL)
        {
            if(strcmp(temp->id, newnode->id) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists) printf("Error: This ID already exists! Please enter a unique ID.\n");

        else  break;
    }

```

```c
    // STEP 2: Name Validation Loop
    while(1)
    {
        printf("Enter Food Name: ");
        scanf(" %[^\n]", newnode->name);

        temp = start;
        exists = 0;

        // Check if Name exists
        while(temp != NULL)
        {
            if(strcmp(temp->name, newnode->name) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists)   printf("Error: This Name already exists! Please enter a unique Name.\n");

        else break;

    }
    //printf("Enter Food ID: ");
    //scanf("%19s", newnode->id);
    // printf("Enter Food Name: ");
    // scanf(" %[^\n]", newnode->name);
    printf("Enter Food Price: ");
    scanf("%f", &newnode->price);

    printf("Enter Stock Quantity: "); // NEW ADDED CODE
    scanf("%d", &newnode->stockCount); // NEW ADDED CODE

```

```c
    newnode->next = start;
    start = newnode;

    printf("Item added at beginning.\n");
    saveData();
}

```

```c
void InsertLastFood() // For inserting in the last of the menu
{
    // UPDATED CODE: ID and Name validation loop
    struct food *newnode = (struct food*)malloc(sizeof(struct food));
    struct food *temp;
    int exists;

    // STEP 1: ID Validation Loop
    while(1)
    {
        printf("Enter Food ID: ");
        scanf("%19s", newnode->id);

        temp = start;
        exists = 0;

        // Check if ID exists
        while(temp != NULL)
        {
            if(strcmp(temp->id, newnode->id) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists) printf("Error: This ID already exists! Please enter a unique ID.\n");

        else  break;
    }

```

```c
    // STEP 2: Name Validation Loop
    while(1)
    {
        printf("Enter Food Name: ");
        scanf(" %[^\n]", newnode->name);

        temp = start;
        exists = 0;

        // Check if Name exists
        while(temp != NULL)
        {
            if(strcmp(temp->name, newnode->name) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists)   printf("Error: This Name already exists! Please enter a unique Name.\n");

        else break;

    }

    // printf("Enter Food ID: ");
    // scanf("%19s", newnode->id);
    // printf("Enter Food Name: ");
    // scanf(" %[^\n]", newnode->name);
    printf("Enter Food Price: ");
    scanf("%f", &newnode->price);

    printf("Enter Stock Quantity: "); // NEW ADDED CODE
    scanf("%d", &newnode->stockCount); // NEW ADDED CODE

```

```c
    newnode->next = NULL;

    if(start == NULL) start = newnode;

    else
    {
        struct food *i = start;
        while(i->next != NULL) i = i->next;
        i->next = newnode;
    }

    printf("Item added at last.\n");
    saveData();
}

```

```c
void InsertAnyFood() // For inserting in any position of the menu; For inserting after any item of the menu
{
    if(start == NULL)
    {
        printf("No items available in the menu.\n");
        return;
    }

    char x[20];
    printf("After which food ID you want to insert: ");
    scanf("%19s", x);

    struct food *i = start;
    while(i != NULL && strcmp(i->id, x) != 0) i = i->next;
    if(i == NULL)
    {
        printf("Given ID not found.\n");
        return;
    }

    // UPDATED CODE: ID and Name validation loop
    struct food *newnode = (struct food*)malloc(sizeof(struct food));
    struct food *temp;
    int exists;

    // STEP 1: ID Validation Loop
    while(1)
    {
        printf("Enter Food ID: ");
        scanf("%19s", newnode->id);

        temp = start;
        exists = 0;

```

```
        // Check if ID exists
        while(temp != NULL)
        {
            if(strcmp(temp->id, newnode->id) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists) printf("Error: This ID already exists! Please enter a unique ID.\n");

        else  break;
    }

    // STEP 2: Name Validation Loop
    while(1)
    {
        printf("Enter Food Name: ");
        scanf(" %[^\n]", newnode->name);

        temp = start;
        exists = 0;

        // Check if Name exists
        while(temp != NULL)
        {
            if(strcmp(temp->name, newnode->name) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists)   printf("Error: This Name already exists! Please enter a unique Name.\n");

        else break;

    }
    //printf("Enter Food ID: ");
    //scanf("%19s", newnode->id);
    // printf("Enter Food Name: ");
    // scanf(" %[^\n]", newnode->name);
    printf("Enter Food Price: ");
    scanf("%f", &newnode->price);

    printf("Enter Stock Quantity: "); // NEW ADDED CODE
    scanf("%d", &newnode->stockCount); // NEW ADDED CODE

    newnode->next = i->next;
    i->next = newnode;

    printf("Item added after ID %s.\n", x);
    saveData();
}
```

```c
/* ---------------- Delete Functions for Deleting Food Items from the Menu (3) - Manager ---------------- */

void deleteBeginFood() // For deleting from the beginning of the menu
{
    if(start == NULL)
    {
        printf("No items available in the menu.\n");
        return;
    }

    struct food *temp = start;
    start = start->next;
    free(temp);

    printf("Item deleted from beginning.\n");
    saveData();
}

```

```c
void deleteLastFood() // For deleting from the last of the menu
{
    if(start == NULL)
    {
        printf("No items available in the menu.\n");
        return;
    }

    if(start->next == NULL)
    {
        struct food *temp = start;
        start = NULL;
        free(temp);

        printf("Item deleted from last.\n");
        saveData();
        return;
    }

    struct food *i = start;
    while(i->next->next != NULL) i = i->next;
    struct food *temp = i->next;
    i->next=NULL;
    free(temp);

    printf("Item deleted from last.\n");
    saveData();
}

```

```c
void deleteAnyFood() // For deleting from any position of the menu; For deleting by ID number
{
    if(start == NULL)
    {
        printf("No items available in the menu.\n");
        return;
    }

    char y[20];
    printf("Which food ID you want to delete: ");
    scanf("%19s", y);

    if(strcmp(start->id, y) == 0)
    {
        struct food *temp = start;
        start = start->next;
        free(temp);

        printf("Item deleted having ID %s.\n", y);
        saveData();
        return;
    }

    struct food *i = start;
    while(i->next != NULL && strcmp(i->next->id, y) != 0) i = i->next;
    if(i->next == NULL)
    {
        printf("Given ID not found.\n");
        return;
    }

    struct food *temp = i->next;
    i->next = i->next->next;
    free(temp);
    printf("Item deleted having ID %s.\n", y);
    saveData();
}
```

```c
/* ---------------- Update, Search and Count Functions for Menu (3) - Manager ---------------- */

void updateNodeFood() // For updating any food item in the menu by the ID number
{
    char x[20];
    printf("Enter the food ID to update: ");
    scanf("%19s", x);

    struct food *i = start;
    while(i != NULL && strcmp(i->id, x) != 0)
        i = i->next;

    if(i == NULL)
    {
        printf("Food ID %s not found.\n", x);
        return;
    }
// ID and Name validation starting
    char temp_id[20];
    char temp_name[50];

    struct food *temp;
    int exists;

    int id_valid = 0;
    int name_valid = 0;

    while(1)
    {
        // ID checking
        if(id_valid == 0)
        {
            printf("Enter New Food ID: ");
            scanf("%19s", temp_id);

            temp = start;
            exists = 0;
```

```
        while(temp != NULL)
        {

            if(temp != i && strcmp(temp->id, temp_id) == 0)
            {
                exists = 1;
                break;
            }
            temp = temp->next;
        }

        if(exists)
        {
            printf("Error: This ID already exists in the menu! Please try again.\n");
            continue;
        }
        else
        {
            id_valid = 1;
        }
    }

    // STEP 2: Name Check
    if(name_valid == 0)
    {
        printf("Enter New Food Name: ");
        scanf(" %[^\n]", temp_name);

        temp = start;
        exists = 0;
```

```
            exists = 0;
            while(temp != NULL)
            {
                if(temp != i && strcmp(temp->name, temp_name) == 0)
                {
                    exists = 1;
                    break;
                }
                temp = temp->next;
            }

            if(exists)
            {
                printf("Error: This Name already exists in the list! Please try again.\n");
                continue;
            }
            else
            {
                name_valid = 1;
            }
        }

        // Finalize
        if(id_valid == 1 && name_valid == 1)
        {

            strcpy(i->id, temp_id);
            strcpy(i->name, temp_name);
            break;
        }
    }

```

```
    printf("Enter New Food Price: ");
    scanf("%f", &i->price);

    printf("Enter New Stock Quantity: "); // NEW ADDED CODE
    scanf("%d", &i->stockCount); // NEW ADDED CODE

    printf("Description of the food item is updated.\n");
    saveData();
}

```

```c
void searchNodeFood() // For searching any food item in the menu by the ID number
{
    char key[20];
    printf("Enter the food ID to search: ");
    scanf("%19s", key);
    struct food *i = start;
    int pos = 1;
    while(i != NULL)
    {
        if(strcmp(i->id, key) == 0)
        {
            printf("Found at the position %d. Name: %s, Price: %.2f Tk, Stock: %d\n", // UPDATED CODE (Shown Stock)
                    pos, i->name, i->price, i->stockCount);
            return;
        }
        i = i->next;
        pos++;
    }
    printf("Food ID %s not found.\n", key);
}
```

```c
void countNodeFood() // For counting the food items of the menu
{
    struct food *i = start;
    int count = 0;
    while(i != NULL)
    {
        count++;
        i = i->next;
    }
    printf("Total food items: %d\n", count);
}
```

```c
/* ################### QUEUE AND NESTED NODE OPERATIONS (3) ################### */

/* ---------------- Confirming Placed Orders - Enqueue and Nested Node Creation (1) - Customer ---------------- */

void addPendingOrder(int table, char *foodID, int qty) // Confirming placed orders by providing the table number, food id and quantity
{
    struct orderNode *ptr = pendingOrders;
    while(ptr != NULL)
    {
        if(ptr->tableNo == table) break;
        ptr = ptr->next;
    }

    struct orderItem *newIt = (struct orderItem*)malloc(sizeof(struct orderItem));
    strcpy(newIt->foodID, foodID);
    newIt->qty = qty;
    newIt->next = NULL;

    if(ptr != NULL)
    {
        if(ptr->items == NULL)
        {
            ptr->items = newIt;
        }

        else
        {
            struct orderItem *it = ptr->items;
            while(it->next != NULL) it = it->next;
            it->next = newIt;
        }
    }

```

```c
    else
    {
        struct orderNode *newOrd = (struct orderNode*)malloc(sizeof(struct orderNode));
        newOrd->tableNo = table;
        newOrd->items = newIt;
        newOrd->next = NULL;

        if(pendingOrders == NULL) pendingOrders = newOrd;

        else
        {
            struct orderNode *t = pendingOrders;
            while(t->next != NULL) t = t->next;
            t->next = newOrd;
        }
    }
}

```

```c
/* ---------------- Viewing Pending Orders - Printing Queue with Nested Node (1) - Both ---------------- */

void viewAllPendingOrders() // For viewing pending orders
{
    printf("\n====== Pending Orders ======\n");

    if(pendingOrders == NULL)
    {
        printf("\nNo pending orders available.\n");
        return;
    }

    struct orderNode *p = pendingOrders;

    while(p != NULL)
    {
        printf("\nTable %d: \n", p->tableNo);

        struct orderItem *it = p->items;
        float tableTotal = 0; // Per table total

        while(it != NULL)
        {
            struct food *f = start;

            while(f != NULL && strcmp(f->id, it->foodID) != 0)
                f = f->next;

            if(f != NULL)
            {
                float itemTotal = f->price * it->qty;
                tableTotal += itemTotal;

                printf("  %s x %d (%.2f Tk)\n", f->name, it->qty, itemTotal);
            }
```

```c
            else
                printf("  Unknown Food ID %s x %d\n", it->foodID, it->qty);

            it = it->next;
        }

        printf("--- Total Bill: %.2f Tk ---\n", tableTotal);

        p = p->next;
    }

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

```c
/* ----------------- Serving Orders - Dequeue (1) - Manager ----------------- */

void serveNextOrder() // For serving pending orders
{
    if(pendingOrders == NULL)
    {
        printf("No pending orders available.\n");
        return;
    }

    struct orderNode *serve = pendingOrders;
    printf("\nServing Table %d: \n", serve->tableNo);
    struct orderItem *it = serve->items;

    while(it != NULL)
    {
        struct food *f = start;
        while(f != NULL && strcmp(f->id, it->foodID) != 0) f = f->next;
        if(f != NULL) printf("  %s x %d (%.2f Tk)\n", f->name, it->qty, f->price * it->qty);
        else printf("  Unknown Food ID %s x %d\n", it->foodID, it->qty);
        it = it->next;
    }

    pendingOrders = pendingOrders->next;

    it = serve->items;
    while(it != NULL)
    {
        struct orderItem *tmp = it;
        it = it->next;
        free(tmp);
    }
    free(serve);
    printf("Order served.\n");
}
```

```c
/* #################### MODES (4) #################### */

/* ----------------- Customer Mode (2) - Customer ----------------- */

void orderFood(int table) // Placing orders by providing the table number, food id and quantity
{
    char id[25];
    int qty;
    char more = 'y';
    float total = 0.0f;

    while(more == 'y' || more == 'Y')
    {

        printf("Enter the food ID to order: ");
        scanf("%24s", id);
        printf("Enter the quantity: ");

        if(scanf("%d", &qty) != 1)
        {
            while(getchar()!='\n');
            printf("Invalid quantity!\n");
            continue;
        }

        if(qty <= 0)
        {
            printf("Quantity must be a positive number.\n");
            continue;
        }

        struct food *i = start;
        int found = 0;

```

```c
        while(i != NULL)
        {
            if(strcmp(i->id, id) == 0)
            {
                // NEW ADDED CODE: Checking Stock Logic Started
                if(i->stockCount >= qty)
                {
                    i->stockCount = i->stockCount - qty; // Decreasing stock
                    saveData(); // Saving Updated Stock to file

                    total += i->price * qty;
                    printf("Added %d x %s (%.2f Tk)\n", qty, i->name, i->price * qty);
                    addPendingOrder(table, id, qty);
                }
                else if(i->stockCount == 0)
                {
                    printf("Sorry! This item is OUT OF STOCK.\n");
                }
                else
                {
                    printf("Insufficient Stock! Only %d items available.\n", i->stockCount);
                }
                // NEW ADDED CODE: Checking Stock Logic Ended

                found = 1;
                break;
            }
            i = i->next;
        }
        if(!found) printf("Given ID not found.\n");

        printf("Order more? (y/n): ");
        scanf(" %c", &more);
    }
```

```c

    if(total > 0) // UPDATED CODE: Added this check
    {
        printf("\nTotal payable: %.2f Tk.\n", total);
        printf("Your order will be served shortly.\n");
    }
}

```

```c
void customerMode() // ##### CUSTOMER INTERFACE #####
{
    int table;
    printf("Enter Your Table Number (1-50): ");
    scanf("%d", &table);

    if(table < 1 || table > 50)
    {
        printf("Invalid Table Number! (Choice From 1 to 50.)\n");
        //return;
        customerMode(); // Updated

    }
    else // Updated
    {
        printf("Table Number %d Selected.\n", table);

        int choice;
        while(1)
        {
            printf("\n========= CUSTOMER MODE =========\n");
            printf("1. Show The Restaurant Menu\n");
            printf("2. Select & Order Items\n");
            printf("3. View ALL Pending Orders\n");
            printf("4. Back To Main Menu\n");
            printf("\n");
            printf("Enter Your Choice (1-4): ");

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

```

```
            switch(choice)
            {
            case 1:
                printLinkList();
                break;
            case 2:
                orderFood(table);
                break;
            case 3:
                viewAllPendingOrders();
                break;
            case 4:
                return;
            default:
                printf("Invalid Choice!\n");
                break;
            }
        }
    }
}

```

```
/* ---------------- Manager Mode (2) - Manager ---------------- */

int managerLogin()  // For passwording the manager mode
{
    char pass[50];
    printf("Enter The Password: ");
    scanf("%49s", pass);
    if((strcmp(pass, "222") == 0) || (strcmp(pass, "223") == 0) || (strcmp(pass, "042") == 0) || (strcmp(pass, "017") == 0) || (strcmp(pass, "457") == 0)) return 1;
    printf("Invalid Password. Try Again.\n");
    managerLogin(); // Updated 2 lines
    // return 0;
}

```

```c
void managerMode()  // ##### MANAGER INTERFACE #####
{
    if(!managerLogin()) return;
    int ch;

    while(1)
    {
        printf("\n========= MANAGER MODE =========\n");
        printf("1. Show The Restaurant Menu\n");
        printf("2. Add Item (First)\n");
        printf("3. Add Item (Last)\n");
        printf("4. Add Item (Any-ID)\n");
        printf("5. Delete Item (First)\n");
        printf("6. Delete Item (Last)\n");
        printf("7. Delete Item (Any-ID)\n");
        printf("8. Update Item (By ID)\n");
        printf("9. Search Item (By ID)\n");
        printf("10. Count Items\n");
        printf("11. View All Pending Orders\n");
        printf("12. Serve Next Order\n");
        printf("13. Back To Main Menu\n");
        printf("\n");
        printf("Enter Your Choice (1-13): ");

        if(scanf("%d", &ch) != 1)
        {
            while(getchar()!='\n');
            printf("Invalid Choice!\n");
            continue;
        }

```

```
        switch(ch)
        {
        case 1:
            printLinkList();
            break;
        case 2:
            InsertBeginFood();
            break;
        case 3:
            InsertLastFood();
            break;
        case 4:
            InsertAnyFood();
            break;
        case 5:
            deleteBeginFood();
            break;
        case 6:
            deleteLastFood();
            break;
        case 7:
            deleteAnyFood();
            break;
        case 8:
            updateNodeFood();
            break;
        case 9:
            searchNodeFood();
            break;
        case 10:
            countNodeFood();
            break;
        case 11:
            viewAllPendingOrders();
            break;
        case 12:
            serveNextOrder();
```

```
            break;
        case 13:
            return;
        default:
            printf("Invalid Choice.\n");
            break;
        }
    }
}

/* ----------------- Main (1) ----------------- */

int main()
{
    loadData();
    int choice;
    while(1)
    {
        printf("##################################################\n");
        printf("--------------------------------------------------\n");
        printf("|       SMART RESTAURANT MANAGEMENT SYSTEM       |\n");
        printf("--------------------------------------------------\n");
        printf("##################################################\n");
        printf("\n1. MANAGER\n2. CUSTOMER\n3. EXIT\n\nEnter Choice (1-3): ");


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

        switch(choice)
        {
        case 1:
            managerMode();
            break;
        case 2:
            customerMode();
            break;
        case 3:
            printf("-------Thank you for visiting our restaurant-------");
            exit(0);
        default:
            printf("Invalid Choice.\n");
            break;
        }
```

```
        }
    }
    return 0;
}
```

Figure 3.1: Complete Source Code Implementation
This comprehensive series of screenshots presents the full source code implementation of the Smart Restaurant Management System, showcasing the detailed development of data structures including linked lists and queues, alongside modular function organization and systematic error handling across the entire 1000+ lines of C programming.

3.2 Performance Analysis

Time Complexity Analysis

File Operations:
- saveData(): O(n) – Iterates through menu list once
- loadData(): O(n) – Reads and processes each line in file

Menu Display:
- printLinkList(): O(n) – Traverses entire menu list

Insert Functions:
- InsertBeginFood(): $O(n^2)$ – Validates ID and name uniqueness (traversal each) then O(1) insertion
- InsertLastFood(): $O(n^2)$ – Validation (traversal) + O(n) to find tail
- InsertAnyFood(): $O(n^2)$ – Validation + O(n) to find insertion point

Delete Functions:
- deleteBeginFood(): O(1) – Direct head removal
- deleteLastFood(): O(n) – Traversal to find second-last node
- deleteAnyFood(): O(n) – Linear search for target ID

Update & Search:
- updateNodeFood(): $O(n^2)$ – Search + ID/name validation
- searchNodeFood(): O(n) – Linear search
- countNodeFood(): O(n) – Complete traversal

Order Operations:
- addPendingOrder(): O(m + k) – Find table (m orders) + append to items list (k items)
- viewAllPendingOrders(): O(m × k × n) – For each order (m), each item (k), lookup in menu (n)
- serveNextOrder(): O(k + n) – Process k items, each requiring menu lookup
- orderFood(): O(n) per item – Stock lookup in menu

User Interface:
- customerMode(): O(1) per operation (calls other functions)
- managerMode(): O(1) per operation (calls other functions)
- managerLogin(): O(1) – Fixed string comparisons

Main Function:
- main(): O(1) per iteration (calls other functions)

Space Complexity per Function

- All functions: O(1) auxiliary space (except those that inherently handle lists)
- Functions creating nodes: Temporary allocations freed after use
- Persistent memory: Only start and pendingOrders pointers maintain allocated nodes

3.3 Results and Discussion

Functional Testing Results
The system underwent comprehensive testing with the following results:

Menu Management Module Testing:
- All CRUD operations function correctly
- Data persistence maintained across sessions
- Input validation prevents invalid operations
- Memory properly managed during insertions and deletions

Order Processing Module Testing:
- Table-based order placement working correctly
- Pending order queue maintains FIFO order
- Order serving properly deallocates memory
- Bill calculation accurate across various scenarios

User Interface Module Testing:
- Manager authentication functions as expected
- Customer table selection with proper validation
- Menu navigation intuitive and responsive
- Error messages informative and helpful

Performance Validation

Execution Speed:
- Menu operations: Instantaneous response for $n < 1000$ items
- Order operations: Linear scaling with order complexity
- File I/O: Minimal overhead with efficient buffering

Memory Usage:
- Stable memory footprint during extended operation
- Proper cleanup after order serving
- No observable memory leaks in testing

User Experience Assessment
The console-based interface proved effective for the intended educational purpose:

Strengths:
- Clear menu structure and navigation
- Comprehensive input validation
- Informative feedback for all operations
- Consistent formatting and display

Areas for Improvement:
- Limited visual appeal compared to GUI systems
- No advanced features like order modification
- Basic authentication system

Sample Test Scenarios

Test Case 1: Complete Order Cycle

1. Manager adds 3 menu items via InsertLastFood()
2. Customer at table 5 orders multiple items
3. viewAllPendingOrders() displays correct items and totals
4. Manager serves order via serveNextOrder()
5. Memory properly freed, queue updated correctly

Test Case 2: Error Handling

1. Attempt to order non-existent food ID
2. Input invalid quantity values
3. Try to delete from empty menu
4. All scenarios handled gracefully with appropriate error messages

Test Case 3: Persistence Verification

1. Add menu items and exit program
2. Restart program and verify menu loaded correctly
3. Confirm data integrity maintained

The testing results confirm that the system meets all specified requirements and performs reliably under normal operating conditions.

Source Code Output:

```
##################################################
--------------------------------------------------
|       SMART RESTAURANT MANAGEMENT SYSTEM       |
--------------------------------------------------
##################################################

1. MANAGER
2. CUSTOMER
3. EXIT

Enter Choice (1-3): 1
Enter The Password: 222

========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13):
```

Figure 3.2: Manager Mode Interface
This screenshot demonstrates the Manager Mode console interface, showing the administrative menu with options for CRUD operations, order management, and system maintenance functions.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 2
Enter Food ID: A001
Enter Food Name: Biriyani
Enter Food Price: 250
Enter Stock Quantity: 200
Item added at beginning.
```

Figure 3.3: Insert at Beginning Operation
Process of adding a new menu item at the start of the list with confirmation.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 3
Enter Food ID: A002
Enter Food Name: Borhani
Enter Food Price: 90
Enter Stock Quantity: 350
Item added at last.
```

Figure 3.4: Insert at Last Operation
Adding a new item at the end of the menu list and confirmation of successful insertion.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 4
After which food ID you want to insert: A001
Enter Food ID: A003
Enter Food Name: Morog Polao
Enter Food Price: 180
Enter Stock Quantity: 300
Item added after ID A001.
```

Figure 3.5: Insert After Specific ID
Inserting a new menu item after a specified food ID with position confirmation.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 1

-------------------- Smart Restaurant Menu --------------------

ID: A001       | Biriyani             | Price: 250.00 Tk | Stock: 200

ID: A003       | Morog Polao          | Price: 180.00 Tk | Stock: 300

ID: A002       | Borhani              | Price: 90.00 Tk | Stock: 350

-----------------------------------------------------------------
```

Figure 3.6: Full Menu Display
Complete restaurant menu showing all items with IDs, names, prices, and stock quantities.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 8
Enter the food ID to update: A003
Enter New Food ID: A001
Error: This ID already exists in the menu! Please try again.
Enter New Food ID: A005
Enter New Food Name: Tehari
Enter New Food Price: 400
Enter New Stock Quantity: 260
Description of the food item is updated.
```

Figure 3.7: Update Menu Item
Modifying existing menu item details including price and stock updates.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 9
Enter the food ID to search: A002
Found at the position 3. Name: Borhani, Price: 90.00 Tk, Stock: 350
```

Figure 3.8: Search Operation
Finding a specific menu item by ID and displaying its complete details.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 10
Total food items: 3
```

Figure 3.9: Item Count Display
Showing the total number of menu items currently available.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 11

====== Pending Orders ======

Table 1:
  Biriyani x 5 (1250.00 Tk)
--- Total Bill: 1250.00 Tk ---

============================
```

Figure 3.10: View All Pending Orders
Displaying all current pending orders with table-wise itemization and totals.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 12

Serving Table 1:
  Biriyani x 5 (1250.00 Tk)
Order served.
```

Figure 3.11: Serve Next Order
Processing and removing the first order from the queue following FIFO principle.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 5
Item deleted from beginning.
```

Figure 3.12: Delete First Item
Removing the first menu item from the list and updated menu display.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 6
Item deleted from last.
```

Figure 3.13: Delete Last Item
Deleting the final menu item and confirmation of successful removal.

```
========= MANAGER MODE =========
1. Show The Restaurant Menu
2. Add Item (First)
3. Add Item (Last)
4. Add Item (Any-ID)
5. Delete Item (First)
6. Delete Item (Last)
7. Delete Item (Any-ID)
8. Update Item (By ID)
9. Search Item (By ID)
10. Count Items
11. View All Pending Orders
12. Serve Next Order
13. Back To Main Menu

Enter Your Choice (1-13): 7
Which food ID you want to delete: A005
Item deleted having ID A005.
```

Figure 3.14: Delete Specific Item by ID
Targeted deletion of a menu item using its unique ID number.

```
#################################################
-------------------------------------------------
|       SMART RESTAURANT MANAGEMENT SYSTEM      |
-------------------------------------------------
#################################################

1. MANAGER
2. CUSTOMER
3. EXIT

Enter Choice (1-3): 2
Enter Your Table Number (1-50): 1
Table Number 1 Selected.

========= CUSTOMER MODE =========
1. Show The Restaurant Menu
2. Select & Order Items
3. View ALL Pending Orders
4. Back To Main Menu

Enter Your Choice (1-4):
```

Figure 3.15: Customer Mode Interface
This screenshot displays the main Customer Mode console interface, showing the selection menu with options to view the restaurant menu, place orders, check pending orders, or return to the main menu.

```
========= CUSTOMER MODE =========
1. Show The Restaurant Menu
2. Select & Order Items
3. View ALL Pending Orders
4. Back To Main Menu

Enter Your Choice (1-4): 1

-------------------- Smart Restaurant Menu --------------------

ID: A001       | Biriyani            | Price: 250.00 Tk | Stock: 200

ID: A003       | Morog Polao         | Price: 180.00 Tk | Stock: 300

ID: A002       | Borhani             | Price: 90.00 Tk | Stock: 350

----------------------------------------------------------------
```

Figure 3.16: Customer Menu View
Display of available menu items for customer browsing with current stock status.

```
========= CUSTOMER MODE =========
1. Show The Restaurant Menu
2. Select & Order Items
3. View ALL Pending Orders
4. Back To Main Menu

Enter Your Choice (1-4): 2
Enter the food ID to order: A001
Enter the quantity: 5
Added 5 x Biriyani (1250.00 Tk)
Order more? (y/n): n

Total payable: 1250.00 Tk.
Your order will be served shortly.
```

Figure 3.17: Order Placement Process
Step-by-step ordering interface showing item selection, quantity specification, total payable and order confirmation.

```
========= CUSTOMER MODE =========
1. Show The Restaurant Menu
2. Select & Order Items
3. View ALL Pending Orders
4. Back To Main Menu

Enter Your Choice (1-4): 3

====== Pending Orders ======

Table 1:
  Biriyani x 5 (1250.00 Tk)
--- Total Bill: 1250.00 Tk ---

============================
```

Figure 3.18: View All Pending Orders
Displaying all current pending orders with table-wise itemization and totals.

Chapter 4

Engineering Standards and Mapping

This chapter examines the engineering standards, societal impact, environmental considerations, and program outcome mapping of the **Smart Restaurant Management System** project.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

The **Smart Restaurant Management System** demonstrates significant positive impact on educational experiences and practical skill development:

Educational Enhancement: By providing a concrete implementation of data structures in a real-world context, the project bridges the gap between theoretical computer science concepts and practical applications. Students can observe how linked lists, queues, and memory management operate in a meaningful scenario, leading to deeper understanding and better knowledge retention.

Skill Development: The project fosters essential programming skills including:

- Algorithm design and implementation
- Memory management and pointer operations
- File I/O and data persistence
- Input validation and error handling
- Modular programming and code organization

Accessibility: The console-based nature and minimal system requirements make the educational resource accessible to students with limited computational resources, promoting inclusive computer science education.

4.1.2 Impact on Society & Environment

Societal Benefits:

- **Educational Advancement:** Contributes to developing better-prepared software engineers by providing practical learning tools
- **Small Business Demonstration:** Shows how fundamental programming concepts can solve real business problems efficiently
- **Open Knowledge:** Serves as a reference implementation for students learning data structures and C programming

Environmental Considerations:

- **Minimal Footprint:** As a software-only solution, the environmental impact is negligible
- **Energy Efficiency:** The algorithms are optimized for performance, reducing computational resource requirements
- **Digital Distribution:** Eliminates paper-based systems it could potentially replace.

4.1.3 Ethical Aspects

Educational Integrity: The system is designed as a learning tool that complements rather than replaces fundamental understanding. It encourages students to study the implementation details and underlying concepts rather than using it as a black-box solution.

Data Privacy: The current implementation does not collect or store personal user information. Any future enhancements involving user data would require implementing proper privacy safeguards and compliance with data protection regulations.

Academic Honesty: The code serves as a reference implementation that demonstrates proper programming practices while encouraging students to develop their own solutions and understanding.

4.1.4 Sustainability Plan

Technical Sustainability:
- Modular architecture allows easy maintenance and extension
- Well-documented code facilitates future development
- Standard C implementation ensures long-term compatibility

Educational Sustainability:
- Comprehensive documentation supports continued use as teaching resource
- Clear structure enables adaptation for different learning objectives
- Extensible design allows incorporation of additional features and data structures

Development Sustainability:
- Version control practices enable collaborative improvement
- Testing protocols ensure reliability across modifications
- Documentation standards maintain code understandability

4.2 Project Management and Team Work

Our team **Binary Brains**, consisting of **five** members—

1. Sk. Azmain Zunaeid (ID: 251-15-222) - Leader,
2. Md. Delower Sarker (ID: 251-15-457)
3. Md. Maruf Mia (ID: 251-15-042),
4. Syeda Zawada Farah (ID: 251-15-223),
5. Maher Afruz Mira (ID: 251-15-017), and

—worked collaboratively to successfully complete the **Smart Restaurant Management System** project.

As a team, we understood the importance of dividing tasks while maintaining clear communication and shared goals. We all sat together regularly to discuss the progress, tackle challenges, and ensure that each part of the project was on track.

Team Structure and Role Allocation:

- **Project Leader: Zunaeid**, as the team leader, responsible for overall project coordination, task delegation, and timeline management. The leader provided guidance at every stage, performed the initial system design verification, and conducted the final review, correction, and integration of all work —including the code implementation, documentation, and presentation—to ensure quality and coherence.

- **System Implementation Lead: Zunaeid** and **Delower** took the primary responsibility for the hands-on code implementation. This involved writing the core functions, integrating modules, debugging, and verifying the practical functionality of the design.

- **Documentation Team: Zunaeid** was primarily responsible for the comprehensive project report. He diligently compiled the research, wrote detailed descriptions of the methodology, and structured the report as per the required guidelines.

- **Presentation Team: Delower** was responsible for creating the project presentation slides. He worked to distill the complex project details into a clear, concise, and visually engaging format suitable for demonstration.

- **Testing & Quality Assurance: Maruf** focused on validating the system, designing test cases, and ensuring the robustness of error handling and input validation.

- **Research & Requirements Analysis: Farah** and **Mira** collaborated on conducting the feasibility study, gap analysis, and compiling reference materials and best practices for the project.

Through regular team meetings, we ensured that everyone's ideas and contributions were valued, which allowed us to maintain a balanced approach to complete this project. We collaborated seamlessly to tackle issues, make decisions, and ensure that the project progressed smoothly towards completion.

4.3 Complex Engineering Problem

4.3.1 Mapping of Program Outcome

Table 4.1: Justification of Program Outcomes

PO's	Justification
PO3 - Design/Development of Solutions	Complete system architecture with three modules; custom data structures modeling restaurant workflow; full CRUD operations with persistence
PO5 - Modern Tool Usage	Deliberate selection of ANSI C for educational transparency; use of GCC, debugging tools, and systematic documentation practices
PO9 - Individual and Team Work	Effective 5-member collaboration with defined roles; regular team coordination; balanced individual contributions and collective problem-solving

4.3.2 Complex Problem Solving

Table 4.2: Mapping with Complex Problem Solving

EP1 Dept of Knowledge	EP2 Range of Conflicting Require-ments	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent of Stake-holder Involve-ment	EP7 Inter-dependence
√	√	N/A	N/A	N/A	√	N/A

EP1 — Depth of Knowledge

The Smart Restaurant Management System requires extensive knowledge of core computer science concepts, including linked lists, queues, pointers, dynamic memory allocation, file I/O, and algorithmic efficiency. Implementing nested data structures for table-wise orders and ensuring correct traversal, insertion, deletion, and deallocation requires mastery beyond introductory-level skills. The system demonstrates a complex relationship between theoretical knowledge and practical implementation, fulfilling EP1.

EP2 — Range of Conflicting Requirements

For this project, several alternative approaches were possible for each system component. The menu could be stored using arrays or different linked list structures, and the order system could be implemented with stacks, priority queues, or simple lists. Similarly, data persistence could rely on text files or databases. Each option involved trade-offs in complexity, performance, and suitability for the course objectives. We selected singly linked lists, a FIFO queue, and text-file storage as the most balanced and effective solutions. Choosing the most appropriate design among many valid alternatives demonstrates the conflicting requirements addressed in EP2.

EP6 — Extent of Stakeholder Involvement

The system incorporates requirements gathered directly from restaurant staff during stakeholder engagement. The team observed real workflows and interviewed operational personnel to understand practical needs such as fast menu updates, efficient order serving, and clear table-wise tracking. These insights shaped design decisions, including the structuring of order queues and the simplicity of the user interface. The active involvement of real stakeholders ensures alignment with authentic user needs, satisfying EP6.

4.3.3 Engineering Activities

Table 4.3: Mapping with Complex Engineering Activities

EA1 Range of re-sources	EA2 Level of Interac-tion	EA3 Innovation	EA4 Consequences for society and envi-ronment	EA5 Familiarity
N/A	N/A	N/A	N/A	N/A

Chapter 5

Conclusion

This chapter provides a comprehensive summary of the **Smart Restaurant Management System** project, discusses its limitations, and outlines potential directions for future enhancement and extension.

5.1 Summary

The **Smart Restaurant Management System** successfully demonstrates the practical application of fundamental data structures in solving real-world problems. The project implements a complete restaurant management solution using singly linked lists for menu management and a queue-based system for order processing, all developed in ANSI C.

Key Achievements:

- **Comprehensive Menu Management:** Implemented full CRUD operations on food items with persistent storage in "menu.txt"
- **Efficient Order Processing:** Developed a queue-based order system with table management and nested data structures
- **Robust Error Handling:** Incorporated comprehensive input validation and error management throughout the system
- **Proper Memory Management:** Demonstrated correct allocation and deallocation of dynamic memory
- **Modular Architecture:** Organized code into distinct, reusable modules following software engineering best practices
- **Educational Value:** Created a practical learning tool that bridges theoretical concepts and real-world applications

The system successfully meets all specified objectives, providing both functional restaurant management capabilities and valuable educational insights into data structure implementation. The console-based interface, while simple, effectively demonstrates the core concepts while maintaining accessibility and ease of use.

5.2 Limitation

Despite achieving its primary objectives, the system has several limitations that represent opportunities for improvement:

1. **Functionality Gaps:**
    a. No order modification or cancellation capabilities
    b. Basic user interface without graphical elements
2. **Scalability Constraints:**
    a. Linear search operations (O(n)) become inefficient with large menu sizes
3. **Security Limitations:**
    a. Static manager password without encryption or secure storage

5.3 Future Work

The **Smart Restaurant Management System** provides a solid foundation for enhancements and extensions:

1. Enhancing System Functionality

To address the current lack of order modification and cancellation features, future development can introduce a more flexible order management module. This would allow customers or the manager to adjust or remove items from an active order before it is served. Furthermore, the system can be extended with a graphical user interface (GUI) using frameworks such as GTK, Qt, or a simple web-based frontend. A GUI would significantly improve usability, visual clarity, and ease of navigation compared to the existing console interface.

2. Improving Scalability and Performance

The present system relies on linear search operations for menu and order processing, which becomes inefficient as the data size grows. Future work can incorporate more advanced data structures, such as hash tables for constant-time menu lookups or balanced trees to improve overall operation efficiency. These enhancements would allow the system to scale effectively and maintain high performance even with large datasets.

3. Strengthening System Security

The current implementation uses a static, unencrypted manager password, which presents a security vulnerability. Future versions should integrate secure authentication mechanisms, including encrypted password storage, salted hashing, and possibly role-based access control. Implementing these measures would improve system reliability, protect administrative access, and align the project with standard software security practices.

References

1. DataFlair. "Restaurant Order Processing System."
2. International Organization for Standardization. (2018). *ISO/IEC 9899:2018 — Programming Languages — C.* ISO.
3. Kanetkar, Y. (2003). Data Structures Through C (2nd ed.). BPB Publications.
4. Kernighan, B. W., & Ritchie, D. M. (1988). The C Programming Language (2nd ed.). Prentice Hall.
5. OpenAI. (2023). ChatGPT (GPT-4 version) [Large language model]. https://chat.openai.com
6. 深度求索 (DeepSeek). (2024). DeepSeek (Latest version) [Large language model]. https://www.deepseek.com