HOTEL MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

THARUN KUMAR M (2303811710621114)

in partial fulfillment of requirements for the award of the course

EGA1121 – DATA STRUCTURES

in

ELECTRONICS AND COMMUNICATION ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

May, 2024

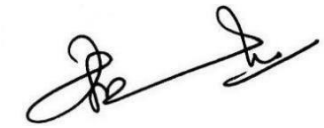
K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **"HOTEL MANAGEMENT SYSTEM"** is the bonafide work of **THARUN KUMAR M (2303811710621114),** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.



SIGNATURE

Dr.A.Delphin Carolina Rani, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR

Department of CSE

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

SIGNATURE

Ms. S. Uma Mageshwari, M.E.

SUPERVISOR

ASSISTANT PROFESSOR

Department of CSE

K. Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on …18.06.24…….

INTERNEL EXAMINER

DECLARATION

I declare that the project report on **"HOTEL MANAGEMENT SYSTEM"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfillment of the requirement of the award of the course **EGA1121-DATA STRUCTURES.**

Signature

THARUN KUMAR M

Place: Samayapuram

Date: 18.06.24

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness to our institution, "**K. Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I extend our sincere acknowledgment and appreciation to the esteemed and honorable Chairman, **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director, **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding our project and offering an adequate duration to complete it.

I would like to thank **Dr. N. VASUDEVAN, M.TECH., Ph.D.,**

Principal, who gave the opportunity to frame the project to full satisfaction.

I thank **Dr. A. DELPHIN CAROLINA RANI M.E., Ph.D.,** Head of the Department of **COMPUTER SCIENCE AND ENGINEERING,** for providing her encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our esteemed project guide **Ms. S. Uma Mageshwari.M.E.,**
Department of **COMPUTER SCIENCE AND ENGINEERING,** for her incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education.

MISSION OF THE INSTITUTION

- ➢ Produce smart technocrats with empirical knowledge who can surmount the global challenges.
- ➢ Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students.
- ➢ Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations.

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems ofsociety.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities

with an understanding of the limitations

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Hotel Management System is a comprehensive solution designed to enhance the operational efficiency of hotels and restaurants by automating key processes related to food orders and table management. Leveraging dynamic data structures such as linked lists and arrays, the system provides a flexible, scalable, and maintainable framework for managing food items and table occupancy. The system comprises several modules, including Food Item Management, Menu Display and Availability Check, Food Ordering, and Table Management. Each module is designed to handle specific tasks, from adding and viewing food items to checking availability, processing orders, and managing table statuses. The main interface ties these modules together, offering a user-friendly interface for staff to interact with the system. By automating routine tasks and providing real-time information on food availability and table occupancy, the Hotel Management System aims to reduce wait times, improve customer satisfaction, and streamline overall operations. This abstract outlines the objectives, data structures, key modules, and the advantages of implementing such a system in a hospitality setting.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

SLL	-	Singly Linked List
UI	-	User Interface
CLI	-	Command Line Interface
GUI	-	Graphical User Interface
NLP	-	Natural Language Processing
GUI	-	Graphical User Interface
APIs	-	Application Programming Interface

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION TO PROJECT

The Hotel Management System is a software solution aimed at enhancing the efficiency and service quality of hotels and restaurants. By automating key operations such as menu management, food ordering, and table occupancy tracking, the system aims to streamline the workflow for staff, reduce customer wait times, and improve overall operational efficiency. Utilizing dynamic data structures like singly linked lists and arrays, the system is designed to be flexible, scalable, and easy to maintain.

1.2 PURPOSE AND IMPORTANCE OF THE PROJECT

The primary purpose of this project is to provide hands-on experience with data structures, a crucial aspect of computer science that underpins efficient data management and retrieval in software applications. By developing a phone directory, students will learn how to apply theoretical concepts in a practical setting, enhancing their problem-solving skills and deepening their understanding of data organization and manipulation.

A Hotel management system is a useful tool that allows users to manage and organize food items efficiently. In the world of programming, implementing such an application can be an exciting and educational task. One effective way to structure the data for a phone directory is by using a singly linked list.

A singly linked list is used for managing the menu in the Hotel Management System due to its dynamic sizing, which allows the menu to grow or shrink easily as food items are added or removed. This flexibility is crucial for a dynamic environment like a restaurant. Linked lists offer efficient insertions and deletions, requiring only pointer adjustments without nee. Despite the trade-off of sequential access time, the benefits of dynamic sizing and efficient updates make singly linked lists an appropriate choice for this application.

1.3. OBJECTIVES

The primary objectives of the Hotel Management System are to streamline menu and table management for improved operational efficiency. It aims to dynamically manage a menu of food items allowing for easy additions, deletions, and modifications, and to check and display the real-time availability of these items. The system handles food orders efficiently, updating available quantitie automatically. Additionally, it manages table occupancy status to ensure

quick allocation of available tables to customers. Lastly, it provides a user-friendly interface for staff, facilitating easy interaction with the system and enhancing service quality.

1.3 PROJECT SUMMARIZATION

The Hotel management application designed to manage and organize hotel informations using a singly linked list data structure. The system offers a user-friendly interface for adding, searching, table occupancy, and ordering foodies. The Hotel Management System project is designed to automate and streamline key operations within a hotel or restaurant.

It includes four main modules: Food Item Management, Menu Display and Availability Check, Food Ordering, and Table Management. These modules work together to provide functionalities such as adding and removing food items, checking their availability, processing orders, and managing table occupancy.

The system uses a singly linked list to handle dynamic menu changes efficiently and an array to manage table statuses for quick access and updates. The integration of these modules through a main interface ensures a seamless and user-friendly experience for the staff, improving overall service quality and operational efficiency.

CHAPTER 2
PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

The system architecture of the Hotel Management System is designed to ensure modularity, scalability, and maintainability. It is composed of distinct modules for managing food items and tables, with a centralized main interface that handles user interactions. Each module is responsible for specific tasks, such as managing the menu, checking food availability, processing orders, and managing table statuses. The architecture leverages dynamic data structures to efficiently handle the varying needs of menu and table management.

A. High-Level System Architecture

The high-level system architecture for the Hotel management application typically consists of several key components:

(i) Food management

(ii) Table management

B. Components of the System Architecture

The **Food Management** is a critical component designed to handle all aspects of food item management within the hotel management system. This module employs a singly linked list data structure to dynamically manage the menu, allowing for efficient addition, deletion, and modification of food items. By using a linked list, the system can easily adapt to changes in the menu without the need for contiguous memory allocation, which is ideal for a frequently changing menu. The module provides operations to create new food items, append them to the list, remove them, and update their details, ensuring that the menu is always current and accurately reflects available inventory.

The **Table Management** focuses on managing the occupancy status of the restaurant's tables. This module uses an array data structure to keep track of up to 30 tables, with each element representing a table's occupancy status. The array allows for quick access and updates, making it easy to find and allocate available tables to customers. Operations within this module include initializing the table array, finding the first available table, marking tables as occupied or free, and viewing the current status of all tables. By efficiently managing table occupancy, this module helps optimize seating arrangements and improve customer satisfaction.

2.2 DETAILED SYSTEM ARCHITECTURE DIAGRAM

A diagram that visually represents the system architecture. The diagram should depict how each component interacts with the others. For example, it can show the User Interface sending requests to the Application Logic, which in turn interacts with the Data Management Layer and the Storage Layer.

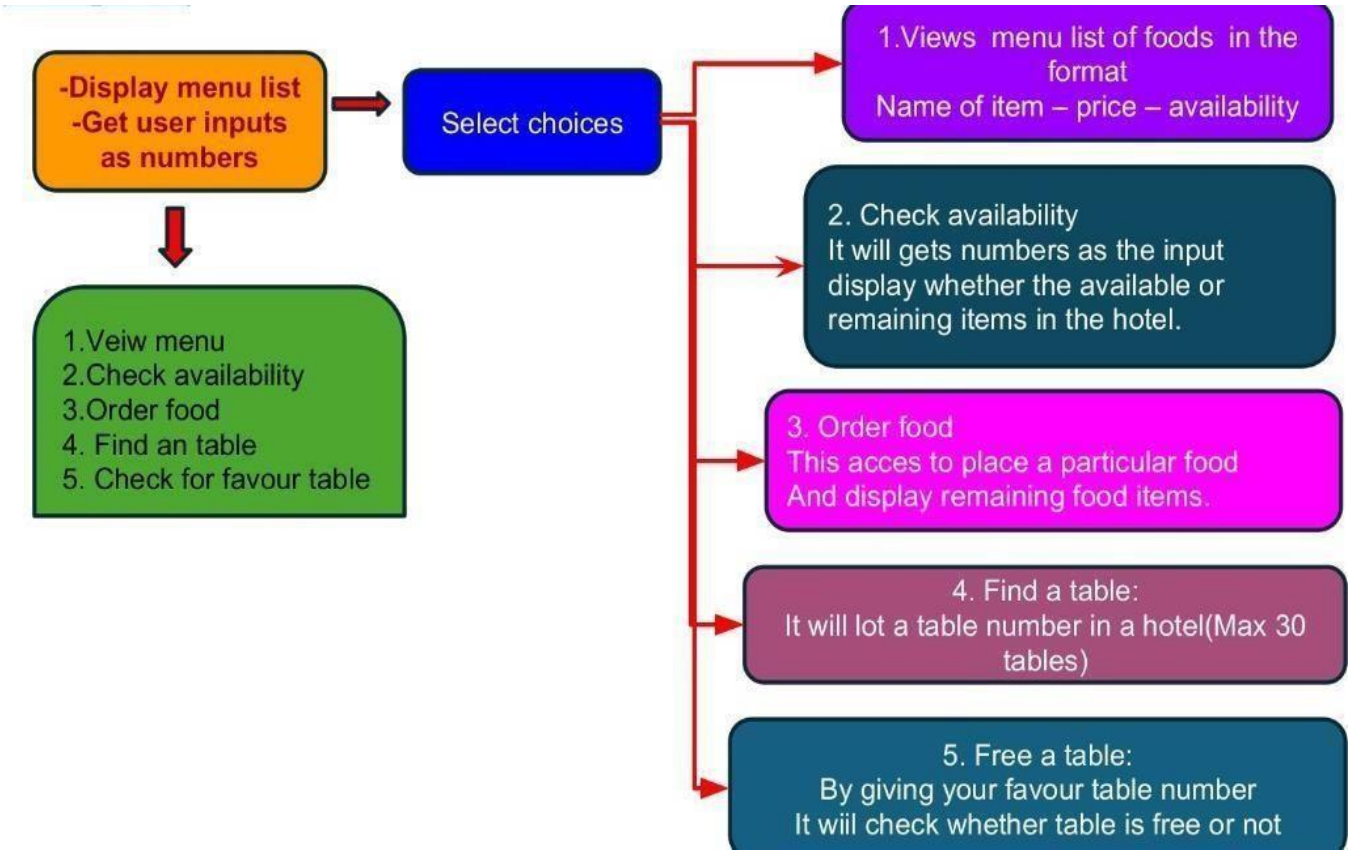


Fig 2.2 : Architecture Diagram

CHAPTER 3
DATA STRUCTURE PREFERANCE

3.1 EXPLANATION OF WHY A SINGLY LINKED LIST WAS CHOSEN

A singly linked list is chosen for managing the menu of food items due to its dynamic nature, allowing the menu to grow or shrink as needed without requiring contiguous memory allocation. This flexibility is crucial for a restaurant environment where the menu may change frequently. Linked lists also offer efficient insertion and deletion operations, especially at the beginning or end of the list, which is essential for dynamically managing the menu.

Dynamic Size:

Flexibility: The linked list can easily grow or shrink as food items are added or removed. This flexibility is crucial for a dynamic environment like a restaurant where the menu may change frequently.

Memory Efficiency: Unlike arrays, which require a fixed size and may allocate more memory than needed, linked lists allocate memory only as needed for each new food item.

Efficient Insertions and Deletions:

Insertion: Adding a new food item (node) to the end of the list can be done in constant time O(1)O(1)O(1) if the tail of the list is maintained.

Deletion: Removing a food item from the list involves adjusting pointers, which is generally more efficient than shifting elements in an array.

Ease of Implementation:

Simpler Reordering: Linked lists make it easier to reorder food items if necessary, as nodes can be rearranged by changing pointers without moving the actual data.

No Need for Contiguous Memory: Linked lists do not require a contiguous block of memory, which can be a limitation for arrays, especially as the list grows large.

Traversal:

Iterating: Traversing the list to display the menu or check availability is straightforward, although it takes O(n)O(n)O(n) time. Given the typical size of a menu, this is usually acceptable.

Consistent Performance:

Operations: Operations like insertion and deletion at the ends of the list are consistently efficient and do not degrade with the size of the list, unlike arrays which may require resizing and copying elements.

3.2 COMPARISON WITH OTHER DATA STRUCTURES

Choosing a data structure for a Hotel management application involves considering trade- offs. Singly linked lists are memory-efficient but lack efficient backward traversal. The main interface ties these data structures and their respective functions together, allowing seamless interaction between menu management and table management.

3.3 ADVANTAGES AND DISADVANTAGES OF USING A SLL

a. Advantages:

The use of a singly linked list in the Hotel Management System offers several advantages.

Its dynamic size allows it to grow or shrink as needed, accommodating changes in the menu or orders seamlessly.

Efficient insertion and deletion, especially at the beginning or end of the list, make it ideal for frequently changing data. The simplicity of a singly linked list also means it is easier to implement compared to more complex data structures.

Additionally, it does not require contiguous memory allocation, as each node is allocated individually.

b. Disadvantages:

However, there are also disadvantages to using a singly linked list. Accessing specific elements requires sequential traversal from the head, making random access less efficient.

The memory overhead is higher since each node requires additional memory for the pointer to the next node. Furthermore, singly linked lists only allow traversal in one direction, lacking the backward traversal capability found in doubly linked list.

CHAPTER -4

DATA STRUCTURE METHODOLOGY

4.1 DYNAMIC SINGLYLY LINKED LIST

Description: This type of singly linked list is dynamic in nature, meaning it allows for the efficient insertion and deletion of menu items as needed. It dynamically adjusts its size based on operations performed, such as adding new menu items or removing existing one**s.**

4.1.1 Key features of a dynamic Singly linked list

Dynamic Memory Allocation: Nodes are allocated dynamically using functions like malloc() in C, allowing nodes to be created or removed based on system needs without requiring a fixed size or pre-allocation.

Flexible Size Management: Enables the list to grow or shrink as menu items are added or removed, adapting to changes in the restaurant's menu without wasted memory.

Efficient Operations: Supports efficient operations such as insertion at the beginning or end of the list, deletion of nodes, and traversal through nodes using pointers.

4.2 NODE STRUCTURE

The Node structure's design allows for flexibility in managing menu items dynamically, accommodating changes in menu offerings and quantities without pre- allocation of memory.

The Node structure encapsulates all necessary attributes of a menu item in the Hotel Management System, facilitating efficient management, manipulation, and display of menu items through a singly linked list. It ensures that the system can handle dynamic changes in menu offerings and maintain accurate inventory information for optimal restaurant operations

4.3. INITIALIZATION, INSERTION & DELETION

Singly linked lists excel in scenarios where frequent insertion and deletion operations are required. Adding a new food item to the menu involves simply creating a new node and adjusting a few pointers, making it faster compared to arrays where elements may need to be shifted. Similarly, deleting a food item updates pointers without requiring data movement.

CHAPTER-5
MODULES

5.1. View Menu:

View Menu allows customers to browse the restaurant's offerings. This module displays all available food and beverage items, categorized for easy navigation (e.g., appetizers, main courses, desserts, drinks). It includes descriptions, prices, and images, along with dietary information such as allergensand nutritional content.

Function: viewMenu(FoodItem* head)

Description: Traverses the linked list of food items and prints the name, price,and available quantity of each item in the menu.

5.2. Check Availability :

provides real-time updates on the availability of menu items. Integrated with the inventory management system, it ensures that customers and staff areaware of which items are in stock, out of stock, or running low. This module helps manage expectations and avoid disappointment due to unavailable items.

Function: checkAvailability(FoodItem* head, char* foodName)

Description: Traverses the linked list to find the specified food item by name and prints its price and available quantity. If the item is not found, it notifies the user

5.3. Ordering:

Ordering facilitates the process of placing food and drink orders. Customers can select items from the menu, customize their orders (e.g., add extratoppings, choose cooking preferences), and submit their orders. This module supports various order types, including dine-in, takeout, and delivery, and integrates with the payment processing system to handle transactions

Function: orderFood(FoodItem* head, int index, int quantity)

Description: Traverses the linked list to find the specified food item by index. If found, itchecks if the requested quantity is available, updates the quantity, and confirms the order. If not enough items are available, it notifies the user.

5.4. Table Management:

Table Management oversees the allocation and status of tables within the restaurant. This module handles reservations, walk-ins, and waitlists, optimizing seating arrangements to maximize occupancy and minimize wait times. It tracks table statuses (e.g., reserved, occupied, available) in real-time, helping staff manage the flow of customers and ensure efficient service.

Funtions

initializeTables(Table tables[]):

Initializes the tables with default values.

findEmptyTable(Table tables[]):

Finds and returns the first empty table.

occupyTable(Table tables[], int tableNumber):

Marks the specified table as occupied.

freeTable(Table tables[], int tableNumber):

Frees the specified table

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION.

The Hotel Management System developed using a singly linked list for menu item management offers a robust foundation for efficient operations within a restaurant environment. Throughout the implementation, several key aspects were highlighted:

Functionality and Efficiency: The program effectively manages menu items through dynamic insertion, deletion, and display operations. This ensures that restaurant staff can easily update and monitor menu offerings, enhancing operational efficiency.

Data Structure Choice: The use of a singly linked list proved advantageous for its simplicity in implementation and flexibility in handling dynamic changes to the menu. It allows for efficient memory usage and straightforward traversal, essential for real-time updates in a fast-paced hospitality setting.

User Interface and Experience: The modular design of the program supports a user-friendly interface, guiding staff through menu management tasks with clear and intuitive operations. This promotes ease of use and reduces the likelihood of errors during menu updates.

Scalability and Adaptability: The system is designed to accommodate future expansions and enhancements. By leveraging a dynamic data structure like a singly linked list, it can easily scale to include additional features such as customer order management, inventory tracking, and sales reporting.

Maintenance and Sustainability: Proper memory management practices ensure the efficient use of system resources, while modular code organization facilitates ongoing maintenance and updates. This approach minimizes downtime and supports the longevity of the system. while the current implementation provides a solid foundation for menu management in a hotel setting, ongoing innovation and adaptation to industry trends will ensure the system remains competitive and aligned with evolving customer expectations and operational needs. By focusing on scalability, user experience, and technological integration, the Hotel Management System can continue to deliver value and drive efficiency in hospitality management.

6.2 FUTURE SCOPE

Several areas offer opportunities for further enhancement and development of the Hotel management system

Advanced Menu Management Features: Implementing advanced features such as categorization of menu items (appetizers, mains, desserts), nutritional information tracking, and seasonal menu updates can enrich the user experience and customer satisfaction.

Integration with Online Ordering Systems: Integrating the system with online ordering platforms can streamline operations and expand the restaurant's reach. This could involve real-time synchronization of menu updates, order processing, and inventory management.

Analytics and Reporting: Incorporating analytics tools to track sales trends, popular menu items, and customer preferences can provide valuable insights for strategic decision-making and marketing campaigns.

Mobile Compatibility: Developing a mobile-friendly version of the system or a dedicated mobile app can empower staff to manage menu items and orders on the go, enhancing operational flexibility and responsiveness.

Security Enhancements: Strengthening data security measures to protect customer information, transaction data, and system integrity against potential threats or breaches

APPENDIX-A
SOURCE CODE

S.No: 1	Exp. Name: ***Project Module***	Date: 2024-06-14

Aim:

Project Module

Source Code:

hello.c

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

#define MAX_TABLES 30

typedef struct FoodItem {
    char name[30];
    float price;
    int quantity;
    struct FoodItem* next;
} FoodItem;

typedef struct Table {
    int tableNumber;
    int isOccupied;
} Table;

// Function prototypes
FoodItem* createFoodItem(char* name, float price, int quantity);
void appendFoodItem(FoodItem** head, char* name, float price, int quantity);
void viewMenu(FoodItem* head);
void checkAvailability(FoodItem* head, char* foodName);
void orderFood(FoodItem* head, int index, int quantity);
void initializeMenu(FoodItem** menu);
void freeMenu(FoodItem* head);
void initializeTables(Table tables[]);
int findEmptyTable(Table tables[]);
void occupyTable(Table tables[], int tableNumber);
void freeTable(Table tables[], int tableNumber);

// Function to create a new food item
FoodItem* createFoodItem(char* name, float price, int quantity) {
    FoodItem* newItem = (FoodItem*)malloc(sizeof(FoodItem));
    strcpy(newItem->name, name);
    newItem->price = price;
    newItem->quantity = quantity;
    newItem->next = NULL;
    return newItem;
}

// Function to append a food item to the menu
void appendFoodItem(FoodItem** head, char* name, float price, int quantity) {
    FoodItem* newItem = createFoodItem(name, price, quantity);
    if (*head == NULL) {
        *head = newItem;
        return;
    }
    FoodItem* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newItem;
}
```

```
    FoodItem* temp = head;
    int index = 0;
    printf("Menu:\n");
    while (temp != NULL) {
        printf("%d. %s - $%.2f - Available: %d\n", index, temp->name, temp->price, temp-
>quantity);
        temp = temp->next;
        index++;
    }
}

// Function to check the availability of a food item by name
void checkAvailability(FoodItem* head, char* foodName) {
    FoodItem* temp = head;
    while (temp != NULL) {
        if (strcmp(temp->name, foodName) == 0) {
            printf("%s - $%.2f - Available: %d\n", temp->name, temp->price, temp->quantity);
            return;
        }
        temp = temp->next;
    }
    printf("Food item '%s' not found in menu.\n", foodName);
}

// Function to order a food item by index
void orderFood(FoodItem* head, int index, int quantity) {
    FoodItem* temp = head;
    int currentIndex = 0;
    while (temp != NULL) {
        if (currentIndex == index) {
            if (temp->quantity >= quantity) {
                temp->quantity -= quantity;
                printf("Order placed for %d %s(s). Remaining: %d\n", quantity, temp->name,
temp->quantity);
            } else {
                printf("Not enough %s available. Available: %d\n", temp->name, temp-
>quantity);
            }
            return;
        }
        temp = temp->next;
        currentIndex++;
    }
    printf("Food item at index '%d' not found in menu.\n", index);
}

// Function to initialize the menu with default items
void initializeMenu(FoodItem** menu) {
    appendFoodItem(menu, "Burger", 5.99, 10);
    appendFoodItem(menu, "Pizza", 8.99, 5);
    appendFoodItem(menu, "Pasta", 7.99, 7);
    appendFoodItem(menu, "Salad", 4.99, 12);
    appendFoodItem(menu, "Soup", 3.99, 8);
    appendFoodItem(menu, "Sandwich", 4.49, 15);
    appendFoodItem(menu, "Fries", 2.99, 20);
```

```
    appendFoodItem(menu, "Coffee", 2.49, 18);
}

// Function to free the allocated memory for the menu
void freeMenu(FoodItem* head) {
    FoodItem* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

// Function to initialize tables
void initializeTables(Table tables[]) {
    for (int i = 0; i < MAX_TABLES; i++) {
        tables[i].tableNumber = i + 1;
        tables[i].isOccupied = 0;
    }
}

// Function to find an empty table
int findEmptyTable(Table tables[]) {
    for (int i = 0; i < MAX_TABLES; i++) {
        if (!tables[i].isOccupied) {
            return tables[i].tableNumber;
        }
    }
    return -1;
}

// Function to occupy a table
void occupyTable(Table tables[], int tableNumber) {
    if (tableNumber > 0 && tableNumber <= MAX_TABLES) {
        tables[tableNumber - 1].isOccupied = 1;
    }
}

// Function to free a table
void freeTable(Table tables[], int tableNumber) {
    if (tableNumber > 0 && tableNumber <= MAX_TABLES) {
        tables[tableNumber - 1].isOccupied = 0;
    }
}

// Main function
int main() {
    FoodItem* menu = NULL;
    initializeMenu(&menu);

    Table tables[MAX_TABLES];
    initializeTables(tables);

    int choice;
    char foodName[30];
```

```
while (1) {
    printf("\nHotel Management System\n");
    printf("1. View Menu\n");
    printf("2. Check Food Availability\n");
    printf("3. Order Food\n");
    printf("4. Find an Empty Table\n");
    printf("5. check a favour Table\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar(); // Consume newline character

    switch (choice) {
        case 1:
            viewMenu(menu);
            break;
        case 2:
            printf("Enter food name to check availability: ");
            fgets(foodName, sizeof(foodName), stdin);
            foodName[strcspn(foodName, "\n")] = '\0'; // Remove trailing newline
            checkAvailability(menu, foodName);
            break;
        case 3:
            printf("Enter menu index to order: ");
            scanf("%d", &index);
            printf("Enter quantity to order: ");
            scanf("%d", &quantity);
            orderFood(menu, index, quantity);
            break;
        case 4: {
            int tableNumber = findEmptyTable(tables);
            if (tableNumber != -1) {
                printf("Empty table found: Table %d\n", tableNumber);
                occupyTable(tables, tableNumber);
            } else {
                printf("No empty tables available.\n");
            }
            break;
        }
        case 5: {
            int tableNumber;
            printf("Enter table number to free: ");
            scanf("%d", &tableNumber);
            freeTable(tables, tableNumber);
            printf("Table %d is now free.\n", tableNumber);
            break;
        }
        case 6:
            printf("Exiting...\n");
            freeMenu(menu);
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
    }
```

```
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Hello World

APPENDIX B - SCREENSHOTS RESULT AND DISCUSSION

1. View menu :

```
Hotel Management System
1. View Menu
2. Check Food Availability
3. Order Food
4. Find an Empty Table
5. Free a Table
6. Exit
Enter your choice: 1
Menu:
Burger - $5.99 - Available: 10
Pizza - $8.99 - Available: 5
Pasta - $7.99 - Available: 7
Salad - $4.99 - Available: 12
Soup - $3.99 - Available: 8
Sandwich - $4.49 - Available: 15
Fries - $2.99 - Available: 20
Ice Cream - $3.49 - Available: 10
Soda - $1.99 - Available: 25
Coffee - $2.49 - Available: 18
```

2. Check Availability

```
Hotel Management System
1. View Menu
2. Check Food Availability
3. Order Food
4. Find an Empty Table
5. Free a Table
6. Exit
Enter your choice: 2
Enter food name to check availability: Pizza
Pizza - $8.99 - Available: 5
```

3. Ordering food

```
Hotel Management System
1. View Menu
2. Check Food Availability
3. Order Food
4. Find an Empty Table
5. Free a Table
6. Exit
Enter your choice: 3
Enter menu index to order: 0
Enter quantity to order: 3
Order placed for 3 Burger(s). Remaining: 7
```

4. Lot a free table and check customer favour table

```
Hotel Management System
1. View Menu
2. Check Food Availability
3. Order Food
4. Find an Empty Table
5. Free a Table
6. Exit
Enter your choice: 4
Empty table found: Table 1

Hotel Management System
1. View Menu
2. Check Food Availability
3. Order Food
4. Find an Empty Table
5. Free a Table
6. Exit
Enter your choice: 5
Enter table number to free: 5
Table 5 is now free.
```