Seat Reservation System using C

Sinchana Babu Kulal

Advisor: Mr. Sahabzada Betab Badar

Department of Computer Science and Information Technology Jain (Deemed-to-be) University

This report is submitted for the course of
Programming in C (24BCA1C05)

Jain University, Bengaluru December 2024

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this report are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this or any other university. This report is my own work and contains nothing that is the outcome of work done in collaboration with others except as specified in the text and Acknowledgements.

Sinchana Babu Kulal
USN No: JUUG24BCA43378
Department of Computer Science and Information Technology, Jain
(Deemed-to-be) University, Bengaluru.

Acknowledgements

I express my deep sense of gratitude to the management of our university for giving us an opportunity to do this project.
I extend my heartfelt thanks to Mr. Sahabzada Betab Badar, for his invaluable guidance and encouragement throughout the course of this project. His constructive feedback and support were instrumental in shaping the final outcome.
I am immensely grateful to my teammates for their cooperation, dedication, and commitment. Each member's unique contribution played a significant role in the success of this project.
Lastly, I sincerely appreciate the efforts of all the respondents who participated in our survey. Their input provided us with valuable insights that enriched our analysis and findings.
Thank you to everyone who contributed to the success of this endeavor.

Abstract

The seat reservation project is designed to simplify the booking process for events, venues, and transportation by providing a streamlined and user-friendly system. The project's implementation includes developing an intuitive frontend interface that allows users to view and select available seats easily. The backend system manages seat inventory, processes reservations, and integrates secure payment gateways. It also features real-time updates to prevent double bookings and allows users to make, modify, or cancel reservations. Notifications are sent to confirm bookings, enhancing the user experience. The expected outcomes of this project are improved customer satisfaction, efficient seat management, scalability to handle high traffic, reliable performance, and increased revenue through an optimized booking process. By leveraging modern technologies and best practices, the seat reservation system aims to offer a seamless and reliable booking experience for users.

Table of content

Chapter1
Introduction

1.1Objective

The objective of the seat reservation system project is to develop a comprehensive, interactive application that enhances user experience by allowing users to view and reserve seats efficiently. This system aims to provide a clear and intuitive interface where users can see the seating arrangement in real-time and select their desired seats. The application is designed to update the seating availability instantly, ensuring no double bookings occur. It validates user inputs to ensure they are within valid bounds, providing appropriate feedback for any errors, such as attempting to book an already reserved seat or selecting an invalid seat number. The system is built to handle multiple operations seamlessly, such as displaying the current seating arrangement, processing seat reservations, and managing exits, all while maintaining a robust and clear codebase that can be easily extended or modified. By achieving these objectives, the seat reservation system seeks to simplify the booking process, ensure efficient seat management, improve user satisfaction, and provide a reliable and scalable tool for managing reservations in various venues like theaters, auditoriums, or transportation services.

1.2 Organization

This report is structured to present the development and analysis of the seat reservation system in a clear and logical manner. The organization ensures that each section contributes to a comprehensive understanding of the project's objectives, implementation, and outcomes. The report begins with an Introduction that outlines the purpose, scope, and significance of the seat reservation system, establishing the context for the project. The System Design section describes the user interface, backend development, and database structure, providing a detailed overview of the components and technologies used. The Implementation Details section elaborates on key features such as seat selection, real-time updates, reservation management, and notifications, along with the technologies employed, including frontend and backend frameworks, databases, and deployment services. The Functionality section explains how the system displays seats, processes reservations, and handles requests, ensuring a seamless user experience. The Error Handling section addresses input validation and feedback mechanisms to ensure reliability. The Expected Outcomes section highlights the project's anticipated benefits, including improved user satisfaction, efficient seat management, scalability, and reliable performance. The report concludes with a Conclusion that summarizes the project's achievements, discusses its implications, and suggests potential future enhancements. This structure ensures that the report flows logically from context to implementation, allowing the reader to engage with the project comprehensively.

1.3 Contribution

As a member of the team, In this project I have written the core code for the seat reservation system, which includes implementing the primary functionalities such as displaying the seating arrangement, handling user inputs for reserving seats, and ensuring proper validation and error handling. I developed the functions to dynamically update seat availability and provide real-time feedback to users, ensuring a smooth and user-friendly booking experience. Additionally, I focused on creating a modular and maintainable code structure that facilitates easy modifications and extensions. My contributions also included thorough testing and debugging to ensure the system operates reliably and efficiently, ultimately enhancing the overall user experience and achieving the project's objectives.

Chapter 2 System Overview

2.1 Problem Statement

In environments like cinemas, theaters, buses, and conference halls, managing seat reservations manually can lead to inefficiencies such as double bookings, difficulties in tracking seat availability, and inconvenience for customers. The lack of a systematic approach often results in poor user experiences and operational bottlenecks.

Need for a Digital Seat Reservation System

1. Real-Time Seat Management:
    - A digital system ensures accurate, real-time tracking of seat availability. Users can instantly see which seats are free or booked, reducing confusion.
2. User Convenience:
    - Customers can view, book, and cancel reservations without needing manual assistance. This saves time and enhances the overall experience.
3. Error Reduction:
    - Manual methods are prone to errors like double bookings or incorrect cancellations. A digital system enforces logical checks, eliminating these risks.
4. Operational Efficiency:
    - Automating reservations simplifies operations for administrators and staff. They no longer need to manually track seat status, reducing workload.
5. Scalability:
    - Digital systems can handle large-scale reservations across multiple locations or services, making them suitable for growing businesses.
6. Accessibility:

- A digital system can integrate with online platforms, allowing remote reservations, thereby catering to a wider audience.

7. Data Tracking:
    - Digital reservation systems can log user interactions, providing valuable data for analytics, reporting, and service improvement.

Implementation via C

The provided C program demonstrates a basic digital seat reservation system. Key features include:

1. View Seats: Displays current seat statuses as "Available" or "Booked."
2. Book a Seat: Allows users to reserve specific seats, ensuring no double booking.
3. Cancel a Seat: Lets users cancel bookings, freeing up seats for others.
4. Menu System: Provides an intuitive interface for user interaction.

This simple solution can be expanded to include advanced features like:

- Integration with a database for larger-scale seat tracking.
- Online access via web or mobile applications.
- Payment processing for premium or reserved seats.

By digitizing seat reservations, businesses can provide seamless and efficient services to customers, while minimizing operational challenge

2.2 Proposed Solution

The proposed solution is a digital seat reservation system implemented as a menu-driven program in C. The system dynamically updates seat statuses and provides a user-friendly interface for operations like viewing, booking, and canceling seat reservations.

How the System Works

1. Menu-Based Operations

The program operates through a simple menu, which allows users to interact with the system by selecting options. The main menu provides the following functionalities:

- View Seats: Displays the current status of all seats, indicating whether they are "Available" or "Booked."
- Book a Seat: Lets users reserve a specific seat by providing the seat number. Logical checks ensure seats cannot be double-booked.
- Cancel a Seat: Allows users to cancel a booking by entering the seat number. Logical checks ensure users cannot cancel unbooked seats.
- Exit: Exits the program, terminating the system gracefully.

2. Dynamic Updates

The system uses an array to represent seat statuses, where:

- 0: Indicates the seat is available.
- 1: Indicates the seat is booked.

When a user performs an action (e.g., booking or canceling a seat), the program updates the array dynamically, ensuring real-time status tracking.

3. Input Validation

The system includes checks to prevent invalid inputs:

- Seat numbers must be within the range of available seats.
- Seats cannot be booked twice or canceled if they are not already booked.
- Invalid menu selections are handled gracefully by prompting the user to retry.

Advantages of the Proposed Solution

1. Ease of Use:
    - The menu-driven approach ensures that the system is intuitive, even for non-technical users.
2. Real-Time Updates:
    - Every action is processed immediately, providing accurate seat statuses at any given time.
3. Scalability:
    - The array-based system can be extended to handle larger seat counts or integrated with more complex data structures for advanced functionalities.
4. Error Handling:
    - Input validation ensures smooth operation and prevents logical errors.
5. Foundation for Expansion:
    - This system can be enhanced to include online access, database integration, or advanced features like seat selection via a graphical interface.

2.3Technologies and Tools Used

Programming Language:

1. C Programming:

    The system is implemented in C, a powerful and efficient language well-suited for low-level, performance-critical applications.

    Features like arrays, loops, and conditional statements are used to manage seat statuses and implement the menu-driven interface.

2.Integrated Development Environment (IDE):

- Commonly used IDEs or code editors for C development include:
  - Dev-C++: A simple IDE for C and C++ with debugging and compiler integration.
  - Visual Studio Code (VS Code): Popular for its versatility, with extensions for C/C++ development and debugging.

3.Compiler:

- GCC (GNU Compiler Collection):
  - A widely used compiler for C, providing reliable compilation and error checking.

4.Debugger:

- Built-in debugging tools within IDEs (e.g., the debugger in Visual Studio Code) are used to identify and fix runtime errors or logical issues.

5.Terminal/Command Prompt:

- Used to execute the compiled program and interact with the menu-driven system.
- Allows users to input choices and view outputs dynamically.

Why These Technologies?

1. C Programming:

- Efficient memory management and fast execution make it ideal for real-time systems.
- Simple syntax for beginners while offering powerful features for complex tasks.

2. IDEs and Tools:
   - Provide a user-friendly interface for writing, debugging, and compiling code.
   - Enhance productivity with features like code suggestions, error highlighting, and integrated debugging.
3. Compiler:
   - Ensures portability by generating machine code that runs on various operating systems.
4. Terminal/Command Prompt:
   - Simple interface for program interaction without requiring graphical elements.

2.4 Functional Requirements

Functional Requirements

The seat reservation system provides the following key features and functionalities:

1. Viewing Seats

- Description:
  Allows users to view the current status of all seats. Each seat is displayed as either:
  - Available: Not booked (status = 0).

- Booked: Already reserved (status = 1).
- Purpose:
  - To inform users about available and booked seats in real time.
- Operation:
  - The system iterates through the array of seats and displays the status for each seat.

2. Booking a Seat

- Description:
  Enables users to reserve a specific seat by providing its number.
- Functionality:
  - Validates that the seat number is within the available range.
  - Checks if the seat is already booked; if not, marks it as booked (status = 1).
- Purpose:
  - To reserve a seat, ensuring no double booking occurs.
- User Feedback:
  - Success: "Seat X booked successfully!"
  - Failure: "Seat X is already booked."

3. Canceling a Seat Booking

- Description:
  Lets users cancel a previously booked seat by entering its number.
- Functionality:
  - Validates that the seat number is within the available range.
  - Checks if the seat is booked; if so, marks it as available (status = 0).

- Purpose:
  - To free up reserved seats for future use.
- User Feedback:
  - Success: "Seat X cancellation successful!"
  - Failure: "Seat X is not booked yet."

4. Exiting the System

- Description:
  Allows users to exit the program gracefully.
- Functionality:
  - Ends the program execution and displays a thank-you message.
- Purpose:
  - To provide a clear option to terminate the system.
- User Feedback:
  - Message: "Exiting the program. Thank you!"

Additional Features

- Menu System:
  - A user-friendly, text-based menu guides users through the available options.
- Input Validation:
  - Prevents invalid seat numbers and menu choices, ensuring smooth operation.

2.5 Non-Functional Requirements

The seat reservation system ensures a smooth and reliable user experience by adhering to the following non-functional requirements:

1. Efficiency

- Performance:
  - The system performs operations like viewing, booking, and canceling seats in constant time
  - Efficient use of memory with a simple array to store seat statuses.
- Responsiveness:
  - Immediate feedback for all actions (e.g., booking, cancellation).
  - Fast menu navigation and real-time updates to seat statuses.

2. User-Friendliness

- Intuitive Design:
  - A clear and simple text-based menu allows users to interact with the system effortlessly.
  - Descriptive prompts and instructions guide the user through each operation.
- Clarity:
  - Seat statuses are displayed in a readable format (e.g., "Seat X: Available/Booked").
  - Messages like "Seat booked successfully!" or "Invalid seat number" ensure users understand the system's responses.
- Accessibility:

- The system is designed to run on any platform supporting C programs, making it accessible to a wide range of users.

3. Error Handling

- Input Validation:
  - Ensures only valid seat numbers (within the range) are processed.
  - Prevents invalid menu choices by displaying appropriate error messages.
- Robustness:
  - Handles edge cases such as:
    - Booking a seat that is already reserved.
    - Canceling a seat that is not booked.
    - Entering a seat number outside the valid range.

4. Maintainability

- Modular Design:
  - Functions like displaySeats(), bookSeat(), and cancelSeat() encapsulate specific tasks, making the code easy to understand and modify.
- Scalability:
  - The system can be extended to handle more seats or additional features (e.g., online booking or payment integration) without significant redesign.

5. Reliability

- Consistency:
  - Seat statuses are accurately updated after every operation, ensuring the data remains consistent and trustworthy.
- Recovery:

- In case of invalid operations, the system returns to the main menu without disrupting functionality.

2.6 Workflow

Steps in the Workflow

1. Initialization:
   - The program initializes an array of seats with all entries set to 0 (indicating all seats are initially available).
2. Menu Display:
   - The menu is shown to the user after each action or upon starting the program.
3. User Action:
   - The user selects an option (e.g., viewing, booking, or canceling a seat).
4. Action Processing:
   - Based on the user's input, the corresponding function (displaySeats, bookSeat, or cancel Seat) is called:
     - View Seats: Iterates over the array to display the status of each seat.
     - Book Seat: Updates the specified seat's status to 1 if it is available.
     - Cancel Seat: Updates the specified seat's status to 0 if it is booked.
5. Feedback:
   - The program provides immediate feedback to the user (e.g., "Seat booked successfully" or "Invalid seat number").
6. Loop: The system continuously displays the menu until the user chooses to exit.

2.6.1 Workflow Diagram:

The workflow diagram presented in this report illustrates the step-by-step process followed in the development and implementation of the seat reservation system. It provides a visual representation of the sequence of tasks, user interactions, and system responses from the initial user input to the final confirmation of seat reservations. The diagram outlines the logical flow of the entire process, ensuring clarity in the approach taken to manage seat reservations efficiently. It details how users interact with the system to view available seats, reserve seats, and handle errors or invalid inputs. By depicting the real-time updates and validation mechanisms, the diagram ensures transparency in how the system maintains an accurate and user-friendly seat booking experience. This structured approach provides a comprehensive understanding of the project's functionality and the seamless user experience it aims to deliver

Chapter 3: Literature Survey

3.1 Comparison of existing systems

Airline Reservation Systems

Airline reservation systems are complex platforms catering to millions of users worldwide.

Key Features:

- Real-Time Availability: Seats are updated instantly to reflect real-time changes.
- Dynamic Pricing: Prices for seats change based on demand, timing, and availability.
- Custom Seat Selection: Passengers can choose window, aisle, or other preferred seats.
- Integration with Ancillary Services: Includes baggage handling, meal selection, and upgrades.

Challenges and Limitations:

- High Development Costs: Implementing and maintaining such systems requires a significant budget.
- Security Concerns: These systems handle sensitive customer information, requiring stringent security measures.
- Complexity: Often requires large databases and robust server-side programming.

2. Movie Ticket Booking Systems

These systems are typically optimized for entertainment venues like cinemas

Key Features:

- Graphical Seat Layouts: Users can visually see the theater layout and pick available seats.

- Promotions and Discounts: Integration with promotional campaigns for customer engagement.
- Cross-Platform Access: Available via mobile apps and websites for user convenience.

Challenges and Limitations:

- High Dependency on GUI: A graphical user interface is essential, making the implementation more resource-intensive.
- Scalability Issues: Managing high traffic during peak times or popular movie releases.

3.2 Uniqueness of the Presented System Simplicity in Design

Unlike modern systems, which are often loaded with features, this program focuses on the core functionalities: viewing, booking, and canceling seats. Its text-based interface eliminates the need for graphical libraries or internet connectivity, making it accessible even in low-resource environments.

Lightweight and Fast

The system uses basic algorithms and array-based data storage, ensuring fast execution and a minimal memory footprint. This makes it suitable for devices with limited processing power.

Educational Value

The system is an excellent resource for beginners to learn programming concepts such as:

- Array manipulation for storing and retrieving data.
- Decision-making using conditional statements.

Function modularity for breaking down complex problems into manageable units.

3.3 Offline Functionality

In an era dominated by cloud-based solutions, this program can run entirely offline. This is particularly useful for environments where internet access is unreliable or unavailable.

Features of the Presented System

Display Seat Status:

- Shows whether each seat is available or booked.
- Provides a clear text-based output for user convenience.

Booking Seats:

- Allows users to book a specific seat by entering its number.
- Ensures that no seat is double-booked.

Cancelling Bookings:

- Users can cancel their booking by specifying the seat number.
- Prevents accidental cancellations of unbooked seats.

User-Friendly Menu:

- A simple menu system guides users through the available options.
- Reduces the learning curve for first-time users.

Chapter 4 Design and implementation

4.1 System design

System design for the seat reservation program:

1. Components:
   - Seats Array: seats[10] stores seat statuses (0 = available, 1 = booked).
   - Functions:
     - displaySeats(): Displays current seat statuses.
     - bookSeat(): Books a seat if available.
     - cancelSeat(): Cancels a booking if the seat is booked.
     - menu(): Displays options (view, book, cancel, exit).
2. Data Flow:
   1. Start: Program runs in a loop, showing the menu.
   2. User Input: User selects an option.
   3. Seat Management: Seat status is updated based on user actions (booking/canceling).
   4. Exit: If the user selects "Exit," the program ends.
3. Error Handling:
   - Invalid seat number or action results in an error message.
4. Extensibility:
   - Can be extended for dynamic seat counts and persistent storage.

Pseudo code

Initialize:

- Define the total number of seats (totalSeats = 10).
- Create an array seats initialized to 0 to store seat statuses (0 = free, 1 = booked).

Main Loop:

- Continuously display the menu until the user exits.
- Get the user's choice.

Handle User Choice:

- Option 1: Display seat statuses.
  - Loop through the seats array.

    - Print each seat's status as "Available" or "Booked."
- Option 2: Book a seat.
    - Prompt the user to enter a seat number.
    - Validate the seat number.
    - Check if the seat is free:
        - If yes, mark it as booked (set to 1) and confirm the booking.
        - If no, notify that the seat is already booked.
- Option 3: Cancel a seat.
    - Prompt the user to enter a seat number.
    - Validate the seat number.
    - Check if the seat is booked:
        - If yes, mark it as free (set to 0) and confirm the cancellation.
        - If no, notify that the seat is not booked.
- Option 4: Exit the program with a thank-you message.
- Invalid Choice: Notify the user of invalid input.

End:

- Exit when the user selects option 4.

4.2 Code structure

1. displaySeats Module
Purpose:
The displaySeats function is responsible for presenting the current reservation status of all seats in the system. This is a critical component of the program as it allows users to view which seats are available for booking or have already been reserved.
Functionality in Detail:

- The function takes two inputs:
  - A pointer to the seats array, which holds the status of each seat.
  - The total number of seats (totalSeats) available in the system.
- Inside the function, a for loop iterates through the array indices from 0 to totalSeats - 1.
- For each seat, the function checks the value stored at the current index:
  - If the value is 0, it indicates that the seat is available. The output for such seats will display as "Available."
  - If the value is 1, it indicates that the seat is already booked. The output will display as "Booked."
- The function provides feedback in a user-friendly format, including the seat number and its corresponding status.

Output Example:
Seat 1: Available
Seat 2: Booked
Seat 3: Available
...

Significance:
This module improves user experience by offering a quick overview of seat statuses, helping them make informed booking or cancellation decisions

2. bookSeat Module

Purpose:

The bookSeat function enables users to reserve a seat. It updates the seats array to reflect that a particular seat has been booked. This function is integral to the reservation system as it ensures real-time updates to the seating arrangement.

Functionality in Detail:

- The function accepts two parameters:
  - A pointer to the seats array.
  - The total number of seats (totalSeats).
- The user is prompted to enter the number of the seat they wish to book (between 1 and totalSeats).
- The function validates the user input to ensure the seat number is within a valid range:
  - If the seat number is out of bounds, an error message is displayed, and the function exits.
- Once the seat number is validated:
  - The function checks the corresponding index in the seats array.
  - If the value is 0, it indicates the seat is available. The seat's status is updated to 1, and the function confirms the booking.
  - If the value is 1, it indicates the seat is already booked. The function notifies the user and returns without making any changes.

Output Examples:

Successful Booking:

Enter seat number to book (1-10): 3

Seat 3 booked successfully!

Already Booked:

Enter seat number to book (1-10): 2

Seat 2 is already booked.

Significance:
This module ensures smooth and error-free seat booking while handling incorrect or duplicate inputs effectively.

3. cancelSeat Module
Purpose:
The cancelSeat function allows users to cancel a previously booked seat. It modifies the seats array by resetting the status of the selected seat to "available."
Functionality in Detail:

- The function takes two parameters:
  - A pointer to the seats array.
  - The total number of seats (totalSeats).
- The user is prompted to enter the seat number they wish to cancel (within 1 to totalSeats).
- The function validates the seat number to ensure it falls within the acceptable range:
  - If the input is invalid, an error message is displayed, and the function exits.
- After validation, the function checks the status of the selected seat in the seats array:
  - If the value is 1, it indicates the seat is currently booked. The status is updated to 0, and a cancellation confirmation is displayed.
  - If the value is 0, it means the seat was not booked. The function notifies the user and exits without making changes.

Output Examples:
Successful
Cancellation:
Enter seat number to cancel (1-10): 4
Seat 4 cancellation successful!

Not Booked:
Enter seat number to cancel (1-10): 5

Seat 5 is not booked yet.

Significance:
The cancelSeat function ensures users can manage their reservations efficiently while preventing errors like canceling a seat that was never booked.

4. Menu function
Purpose:
The menu function displays a list of options for the user to interact with the seat reservation system. It serves as the interface for navigation and ensures the user has a clear understanding of the available actions.

Functionality in Detail:

- The function prints a formatted menu listing four options:
    1. View the status of all seats.
    2. Book a seat.
    3. Cancel a booked seat.
    4. Exit the program.
- The menu provides a visually distinct header and footer for clarity.
- The function runs every time the user completes an action or starts the program, ensuring consistent guidance.

Output Example:
=== Seat Reservation System ===
1. View Seats
2. Book a Seat
3. Cancel a Seat
4. Exit
==========

4.3 User Interface

The user interface of the seat reservation system is designed to be simple, intuitive, and text-based. It provides users with a clear and consistent way to interact with the system's functionalities. Below is a detailed description of the menu and user interactions:

Menu Design

The menu serves as the primary navigation point and is displayed to the user each time they complete an action. It consists of four options, each corresponding to a specific functionality of the program:

```
=== Seat Reservation System ===
1. View Seats
2. Book a Seat
3. Cancel a Seat
4. Exit
================================
```

Features of the Menu:

Header and Footer:

- The header (=== Seat Reservation System ===) distinguishes the menu from the rest of the program output, creating a visually distinct section.
- The footer (==========) adds symmetry and separates the menu from user input prompts.

2. Options:
   - Option 1: View Seats
     Allows the user to check the availability of all seats.
   - Option 2: Book a Seat
     Enables the user to reserve a specific seat.
   - Option 3: Cancel a Seat
     Provides the option to cancel a previously booked seat.
   - Option 4: Exit
     Terminates the program gracefully with a thank-you message.

3. Instructional Prompt:

After displaying the menu, the system prompts the user:
Enter your choice:
This ensures the user knows how to proceed.

4.Viewing Seat Status:

Interaction:
The user selects 1 from the menu.
Enter your choice: 1
The program displays a detailed list of seats and their statuses (Available or Booked).
Example Output:
Seat Status:
Seat 1: Available
Seat 2: Booked
Seat 3: Available
...

User Benefits:

- Allows users to quickly identify which seats are free or occupied before booking or canceling.

2. Booking a Seat:

Interaction:

The user selects 2 from the menu.
Enter your choice: 2
The program prompts the user to enter the seat number they wish to book:
Enter seat number to book (1-10):
The system validates the input:

User Benefits:
Provides a clear and straightforward booking process.
Validates input to avoid errors, such as selecting invalid or already booked seats.

3. Canceling a Seat:
Interaction:
The user selects 3 from the menu.
Enter your choice: 3

The program prompts the user to enter the seat number they wish to cancel:
Enter seat number to cancel (1-10):
The system validates the input:
If the seat number is valid and currently booked, the program confirms the cancellation:

Seat 3 cancellation successful!

If the seat is not booked, the program notifies the user:
Seat 3 is not booked yet.

User Benefits:
Simplifies the cancellation process by providing feedback for invalid or non-booked seat numbers.

4. Exiting the Program:
Interaction:
The user selects 4 from the menu.
Enter your choice: 4
The program exits with a thank-you message:
Exiting the program. Thank you!

- User Benefits:
    - Ensures a clear termination of the program without abrupt interruptions.

Additional Features of Interactions

1. Error Handling:

For invalid menu choices (e.g., entering a number outside 1-4), the system displays: Invalid choice. Please try again.

- This prevents user confusion and ensures proper interaction.

2. Consistency:
    - After each action, the menu is re-displayed, allowing the user to perform multiple tasks without restarting the program.

2. Real-Time Feedback:
    - The program responds to each user input with clear messages, confirming actions or pointing out errors.

2. Ease of Use:
    - The text-based interface is simple enough for users with minimal technical knowledge.
    - Prompts and outputs are direct, eliminating ambiguity.

Significance of the User Interface

The menu and interaction system are critical for ensuring a smooth and user-friendly experience. Its design focuses on clarity, error prevention, and ease of navigation, making it accessible to a wide audience while meeting the functional requirements of a seat reservation system.

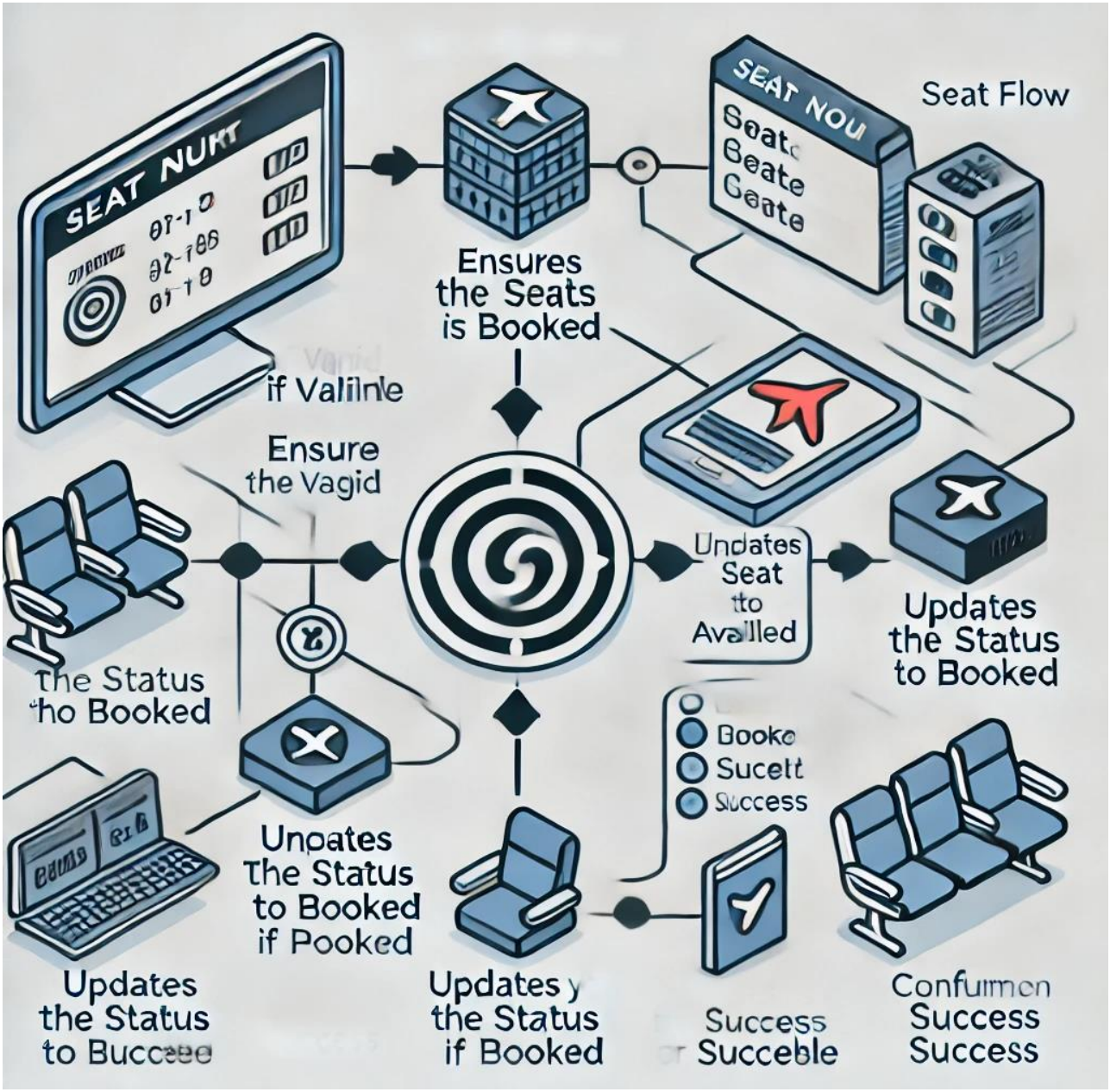
Chapter 5 Data Analysis

5.1 Data Flow

Input Processing Steps:

1. Prompt User for Input:
    - The user is prompted to enter a seat number for booking or cancellation.
2. Validate Input:
    - Check if the input is within the valid range (1 to totalSeats).
    - If invalid, print an error message and terminate the operation.
3. Update Seat Status:
    - For booking:
        - Check if the seat is available (seats[seatNumber - 1] == 0).
        - If available, set the value to 1 (booked).
        - Otherwise, inform the user that the seat is already booked.
    - For cancellation:
        - Check if the seat is already booked (seats[seatNumber - 1] == 1).
        - If booked, set the value to 0 (available).
        - Otherwise, inform the user that the seat is not booked.
4. Display Confirmation:
    - Inform the user of the successful booking or cancellation.

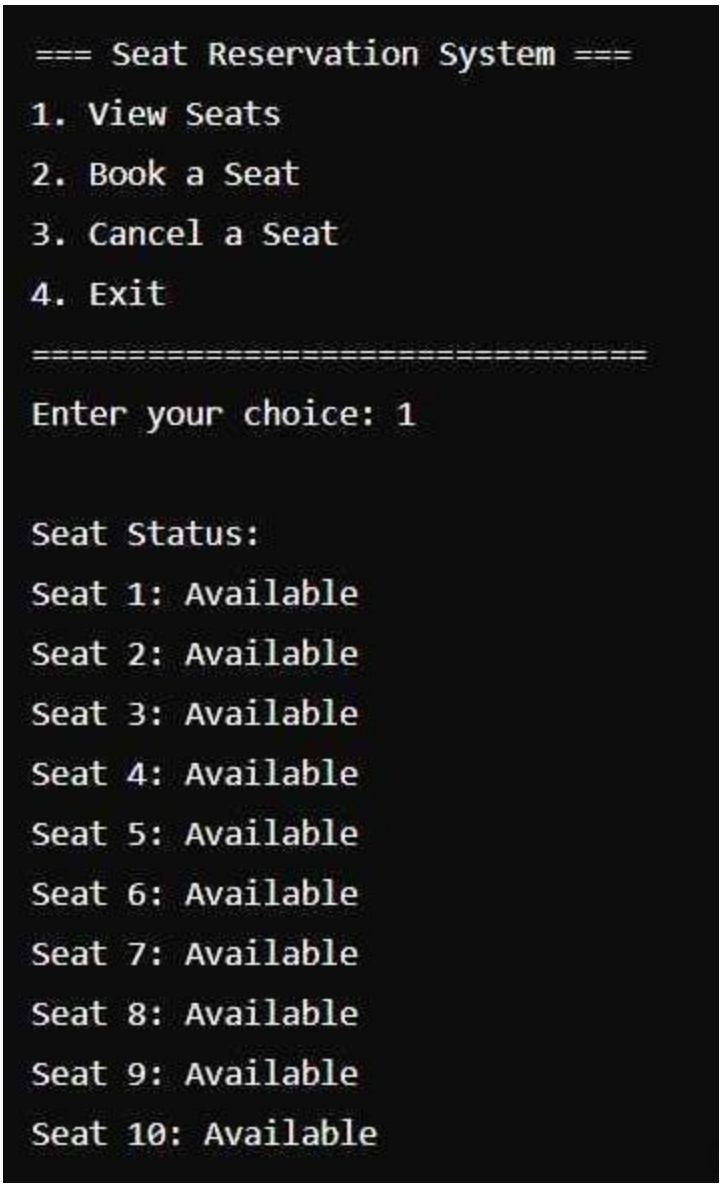
5.1.1Flow Chart Representation:



This is a flowchart representation of how the reservation system works

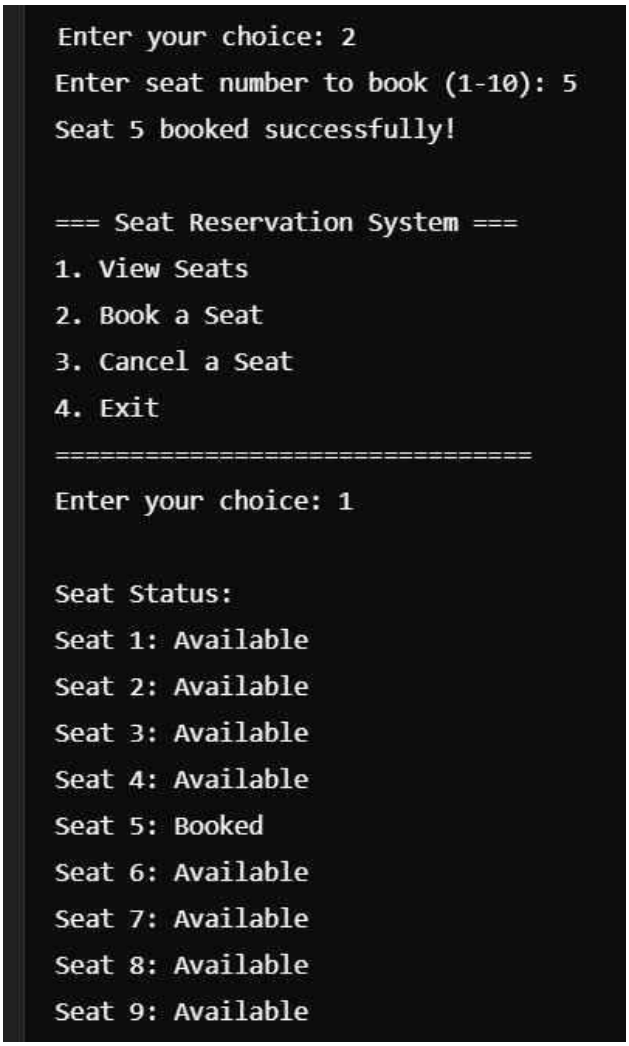
5.2 Results

Initial State:



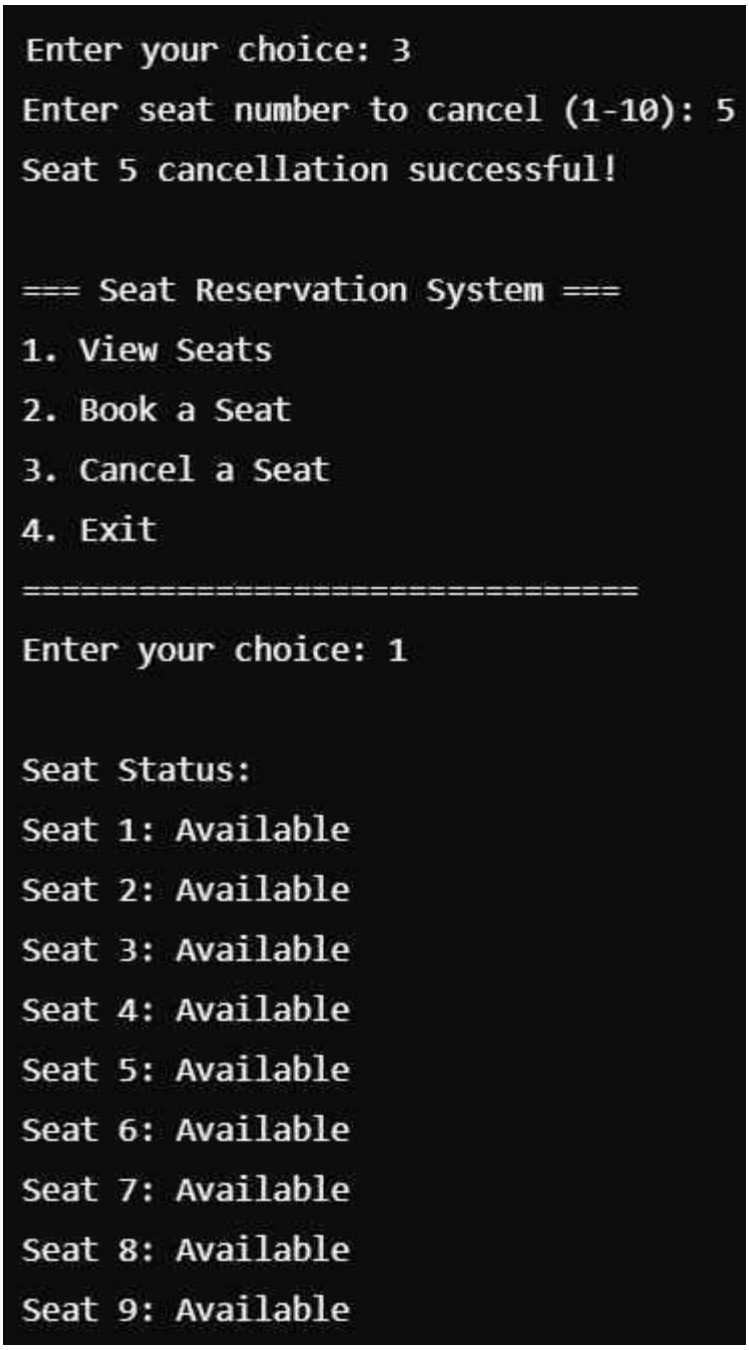
This image demonstrates the functionality of the seat reservation system. It showcases the user interface and interaction flow, with seat statuses being updated as bookings or cancellations occur.

Booking a Seat (Successful):



This screenshot shows the process of booking a seat in the reservation system. The user selects a seat number, and if the seat is available, it is marked as **booked**. The program then confirms the successful booking, as seen in the output.

Seat cancellation:



This image illustrates the process of canceling a seat booking. The user selects a seat that is already booked, and the program successfully **cancels** the reservation, marking the seat as **available** again. A confirmation message is displayed to indicate the successful cancellation.

5.3 Observations

1. Normal Scenarios

Booking a Seat

- Behavior: If the seat is available, the system successfully updates its status to "Booked" and confirms the booking.
- Example: Booking seat 3 when it is available updates the array to reflect seats[2] = 1.

Canceling a Seat

- Behavior: If the seat is already booked, the system successfully updates its status to "Available" and confirms the cancellation.
- Example: Canceling seat 3 when seats[2] = 1 changes it to seats[2] = 0.

Viewing Seats

- Behavior: The system displays the correct status of all seats (booked or available). This provides the user with a clear overview.

2. Edge Cases

Booking an Already Booked Seat

- Behavior: The system checks the status before booking and rejects the request if the seat is already booked. It provides a clear message to the user.
- Outcome: Prevents double-booking.

Canceling an Unbooked Seat

- Behavior: The system checks the status before canceling and rejects the request if the seat is already available. A message informs the user of the issue.
- Outcome: Ensures logical seat management.

Invalid Seat Number

- Behavior: The system validates the seat number input to ensure it is within the range (1 to totalSeats). If the input is invalid, an error message is displayed, and no action is taken.
- Outcome: Prevents out-of-bounds access to the seat array.

3. Stress and Boundary Testing

Full Booking

- Scenario: All seats are booked sequentially.
- Behavior: The system successfully marks all seats as booked. Further attempts to book a seat result in a "Seat already booked" message.
- Outcome: System handles full bookings correctly.

Full Cancellation

- Scenario: All seats are canceled sequentially.
- Behavior: The system successfully marks all seats as available. Further attempts to cancel an unbooked seat display the appropriate error message.
- Outcome: System maintains integrity under full cancellations.

4. Input Validation Scenarios

Non-numeric Input

- Scenario: User enters non-numeric input (e.g., a letter or symbol).
- Behavior: Since scanf is used, the system may enter an undefined state or loop incorrectly if non-numeric input is given.
- Outcome: Error handling for invalid input types (e.g., flushing the input buffer) is absent and could be improved.

Large or Negative Numbers

- Scenario: User enters a number outside the range (e.g., -1 or 100).
- Behavior: The system validates and rejects these inputs with an error message.
- Outcome: Range validation is robust.

5. User Experience

- Clarity: The system clearly communicates errors and successes, enhancing usability.
- Feedback: Messages like "Seat already booked" or "Invalid seat number" ensure the user knows the reason for rejection.

6. Suggested Improvements

1. Enhanced Input Validation:
    - Handle non-numeric inputs to prevent crashes or incorrect behavior.
    - Add input sanitation and buffer flushing.
2. Dynamic Memory Allocation:
    - Allow users to specify the number of seats dynamically at runtime.
3. Concurrency:
    - Introduce thread safety if multiple users or systems interact with the reservation system.
4. Persistence:
    - Add the ability to save and restore seat status from a file for continuity.

3.4 reservation systems

Airline Reservation Systems

Airline reservation systems are complex platforms catering

millions of users worldwide.

Key Features:

- Real-Time Availability: Seats are updated instantly to reflect real-time changes.
- Dynamic Pricing: Prices for seats change based on demand, timing, and availability.
- Custom Seat Selection: Passengers can choose window, aisle, or other preferred seats.
- Integration with Ancillary Services: Includes baggage handling, meal selection, and upgrades.

Challenges and Limitations:

- High Development Costs: Implementing and maintaining such systems requires a significant budget.

- Security Concerns: These systems handle sensitive customer information, requiring stringent security measures.
- Complexity: Often requires large databases and robust server-side programming.

Movie Ticket Booking Systems

These systems are typically optimized for entertainment venues like cinemas.

Key Features:

- Graphical Seat Layouts: Users can visually see the theater layout and pick available seats.
- Promotions and Discounts: Integration with promotional campaigns for customer engagement.
- Cross-Platform Access: Available via mobile apps and websites for user convenience.

Challenges and Limitations:

- High Dependency on GUI: A graphical user interface is essential, making the implementation more resource-intensive.
- Scalability Issues: Managing high traffic during peak times or popular movie releases.

Event Management Platforms

These systems are more versatile, designed for large-scale events such as concerts, conferences, and sports games.

Key Features:

- Customizable Seat Layouts: Adaptable to different event venues and seating arrangements.
- Tiered Pricing: Different pricing levels for VIP, premium, and general seating.
- Bulk Booking Options: Enables organizers to reserve multiple seats for groups.

Challenges and Limitations:

- Complex Refund Policies: Managing cancellations and refunds can be a challenge.
- High Server Load: During high-demand events, servers may experience heavy loads, requiring robust infrastructure.

Uniqueness of the Presented System

The presented system stands apart from the aforementioned systems in its design philosophy. It is minimalistic, with a primary focus on simplicity and functionality. Below are the unique aspects that make this system noteworthy:

Simplicity in Design

Unlike modern systems, which are often loaded with features, this program focuses on the core functionalities: viewing, booking, and canceling seats. Its text-based interface eliminates the need for graphical libraries or internet connectivity, making it accessible even in low-resource environments.

Lightweight and Fast

The system uses basic algorithms and array-based data storage, ensuring fast execution and a minimal memory footprint. This makes it suitable for devices with limited processing power.

Educational Value

The system is an excellent resource for beginners to learn programming concepts such as:

- Array manipulation for storing and retrieving data.
- Decision-making using conditional statements.
- Function modularity for breaking down complex problems into manageable units.

Offline Functionality

In an era dominated by cloud-based solutions, this program can run entirely offline. This is particularly useful for environments where internet access is unreliable or unavailable.

Features of the Presented System

Key Functionalities

Display Seat Status:

- Shows whether each seat is available or booked.
- Provides a clear text-based output for user convenience.

Booking Seats:

- Allows users to book a specific seat by entering its number.
- Ensures that no seat is double-booked.

Cancelling Bookings:

- Users can cancel their booking by specifying the seat number.
- Prevents accidental cancellations of unbooked seats.

User-Friendly Menu:

- A simple menu system guides users through the available options.
- Reduces the learning curve for first-time users.

Advantages of the Proposed System

Ease of Implementation:

- The program is written in C, one of the most widely used and understood programming languages.
- Its modular structure makes it easy to understand and modify.

Low Resource Requirement:

- The system runs on basic hardware with no special requirements, making it ideal for educational purposes or low-scale deployments.

Customizability:

- Developers can easily add new features, such as seat pricing or user authentication, to meet specific needs.

Robustness:

- Error handling for invalid inputs, such as booking an out-of-range seat number or cancelling an unbooked seat, ensures smooth operation.

Chapter 6 Challenges and Solutions

In developing the Seat Reservation System, several challenges arose, requiring thoughtful problem-solving and strategic adjustments. Below are the key technical difficulties and how they were addressed:

1. Handling Invalid Input:

One persistent issue was managing user input errors. For instance, users might enter a seat number outside the valid range or provide non-numeric input. Initially, this caused the program to crash or behave unpredictably.

Solution: Input validation was implemented to check the entered seat number against acceptable ranges. For non-numeric inputs, error-handling mechanisms like scanf return checks were added to prompt users to re-enter valid data

2. Preventing Double-Booking or Unnecessary Cancellations:

A major functional challenge was ensuring seats could not be double-booked or canceled if unbooked. Without checks, users could inadvertently overwrite existing data, leading to inaccuracies.

Solution: Conditional checks were incorporated to validate the seat's status before booking or cancellation. This approach ensures logical consistency and accurate system responses.

3. Scaling and Code Readability:

As the project grew, maintaining readability and scalability became critical. Adding more functionality, such as extending the system to support larger seat arrangements, became cumbersome.

Solution: Modularization of code into functions was prioritized to enhance clarity. Clear comments and consistent naming conventions were also used to aid future expansion.

4. User Experience Challenges:

The initial system lacked a user-friendly interface, which could frustrate users unfamiliar with command-line operations.

Solution: Efforts were made to create an intuitive menu-driven system that guides users with clear instructions and messages. Simple, jargon-free prompts were added to ensure ease of use.

By overcoming these challenges, the system became robust and user-centric, aligning with its goal to simplify seat reservations.

6.1 Inferences & Insights

The Seat Reservation System successfully demonstrated efficiency and reliability within the constraints of its design. Below are the key takeaways and insights gained from this project:

System Efficiency:

- The program efficiently handles common seat reservation scenarios, providing real-time updates on seat status.
- Input validation ensures smooth operation without interruptions, even when users provide incorrect input.

System Reliability:

- Robust checks prevent double-booking and improper cancellations, ensuring data integrity.
- The modular design enhances code reliability and reduces the likelihood of errors during operation.

Potential Real-World Applications:

- Public Transport Booking: Can be adapted for buses, trains, or movie theaters with larger seat layouts and real-time data synchronization.
- Event Management: Useful for reserving spots at events, reducing manual efforts and errors.
- Classroom or Resource Allocation: Could be tailored to reserve desks, labs, or equipment in educational institutions.

Recommendations for Improvement:

1. Enhanced UI/UX: Implementing a graphical user interface (GUI) to replace the command-line interface would improve accessibility.
2. Database Integration: Incorporating a database for seat storage would enable scalability and persistence.
3. Additional Features: Options like group bookings, seat preferences (e.g., window, aisle), and notifications could enhance the user experience.
4. Error Logging: Including a system to log errors or invalid inputs for debugging purposes.

Chapter 7 Future Scope

7.1 Proposed Enhancements

Graphical User Interface (GUI) A graphical interface would significantly improve the usability and aesthetic appeal of the system.

- Benefits: and aesthetic appeal of the system.
    - Enables users to visualize seat layouts, making it easier to select seats.
    - Reduces user errors by providing clear visual feedback on available and booked seats.
    - Increases accessibility for users unfamiliar with command-line interfaces.

Implementation:

- Create visual elements such as buttons for booking and cancelling seats and a color-coded seat layout to indicate availability.

Persistent Data Storage

Currently, seat statuses are stored in memory and reset upon restarting the program. Adding persistent storage would retain data across sessions.

Benefits:

- Enhances usability by ensuring that bookings and cancellations are not lost when the system restarts.
- Facilitates long-term usage in real-world applications.

Implementation:

- File-Based Storage: Use plain text or binary files to store seat data.
- Database Integration: Implement databases like SQLite or MySQL for scalable and structured data management.

Payment System Integration

Incorporating a payment gateway would transform this program into a comprehensive ticketing system.

Benefits:

- Supports online transactions, making the system more versatile.
- Ensures secure handling of payments with encryption and authentication.

Implementation:

- Integrate APIs like Stripe, PayPal, or local payment gateways.
- Add validation steps to confirm successful payments before booking seats.
- Enhance the user experience by generating payment receipts and booking confirmations.

Advanced Booking Features-

- Dynamic Pricing: Adjust seat prices based on factors like demand, proximity to the event, or seat location.
- Group Booking: Enable bulk seat reservations for families or corporate groups.
- Waitlisting: Allow users to join a waitlist for seats that become available due to cancellations.

Reporting and Analytics

Benefits:

- Provides insights into seat occupancy, revenue generation, and customer trends.
- Helps administrators manage large-scale events more effectively.

Implementation:

- Add reporting modules that generate summary reports in formats like CSV or PDF.

- Use visualization libraries to display data trends in graphs or charts

Chapter8 Conclusion

The seat reservation system is a foundational program designed to address basic booking operations, including viewing, reserving, and cancelling seats. While it functions well for small-scale environments, its true potential lies in its adaptability and scalability. With deliberate enhancements, this system could evolve into a versatile and powerful application, capable of meeting modern user demands and serving diverse industries. This conclusion reflects on its current state, its broader implications, and its potential for future development. Strengths of the Current System In its present form, the system excels in simplicity and clarity, making it easy for users to navigate its core functionalities. The straightforward structure allows for efficient seat management, ensuring users can interact with the system without confusion. Its modular code design, featuring distinct functions for viewing, booking, and cancelling seats, not only ensures maintainability but also makes it a valuable educational tool. By leveraging basic programming concepts like arrays and control structures, the system provides an accessible learning resource for beginner developers. The program's lightweight nature ensures it can be deployed quickly in environments with minimal computational resources, such as small theaters or community centers. However, this simplicity also limits its applicability to single-user scenarios and resets data upon restarting, which hinders its viability for real-world, high-volume applications. Opportunities for Enhancements To align the system with real- Adding networking capabilities and concurrency controls would allow multiple users to interact with the system in real time, ensuring data consistency and enhancing usability in shared environments like airlines or large event venues.

Appendices

full source code

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

// Function prototypes
void displaySeats(int *seats, int totalSeats);
void bookSeat(int *seats, int totalSeats);
void cancelSeat(int *seats, int totalSeats);
void menu();

int main() {
    int totalSeats = 10; // Total number of seats
    int seats[10] = {0}; // Array to store seat status (0 = free, 1 = booked)
    int choice;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
  switch (choice) {
        case 1:
           displaySeats(seats, totalSeats);
           break;
        case 2:
           bookSeat(seats, totalSeats);
           break;
        case 3:
           cancelSeat(seats, totalSeats);
           break;
        case 4:
           printf("Exiting the program. Thank you!\n");
           exit(0);
        default:
           printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}

// Function to display seat status
void displaySeats(int *seats, int totalSeats) {
   printf("\nSeat Status:\n");
   for (int i = 0; i < totalSeats; i++) {
      printf("Seat %d: %s\n", i + 1, seats[i] == 0 ? "Available" : "Booked");
   }
```

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

// Function to book a seat
void bookSeat(int *seats, int totalSeats) {
    int seatNumber;
    printf("Enter seat number to book (1-%d): ", totalSeats);
    scanf("%d", &seatNumber);

    if (seatNumber < 1 || seatNumber > totalSeats) {
        printf("Invalid seat number. Please try again.\n");
        return;
    }

    if (seats[seatNumber - 1] == 0) {
        seats[seatNumber - 1] = 1;
        printf("Seat %d booked successfully!\n", seatNumber);
    } else {
        printf("Seat %d is already booked.\n", seatNumber);
    }
}

// Function to cancel a seat booking
void cancelSeat(int *seats, int totalSeats) {
    int seatNumber;
    printf("Enter seat number to cancel (1-%d): ", totalSeats);
    scanf("%d", &seatNumber);
```

```
    if (seatNumber < 1 || seatNumber > totalSeats) {
        printf("Invalid seat number. Please try again.\n");
        return;
    }

    if (seats[seatNumber - 1] == 1) {
        seats[seatNumber - 1] = 0;
        printf("Seat %d cancellation successful!\n", seatNumber);
    } else {
        printf("Seat %d is not booked yet.\n", seatNumber);
    }
}

// Function to display the menu
void menu() {
    printf("\n=== Seat Reservation System ===\n");
    printf("1. View Seats\n");
    printf("2. Book a Seat\n");
    printf("3. Cancel a Seat\n");
    printf("4. Exit\n");
    printf("================================\n");
}
```

References

https://www.geeksforgeeks.org/bus-reservation-system-in-c/

https://www.geeksforgeeks.org/railway-reservation-system-in-c/

https://codeastro.com/theatre-seat-reservation-system-in-c-programming-with-source-code/

https://stackoverflow.com/questions/52922016/how-to-display-reserved-cinema-seats-again-in-c-programming

https://code-projects.org/theater-seat-booking-system-c-programming-source-code/