****

**Southeast University**

**Department of Computer Science and Engineering (CSE)**

**School of Sciences and Engineering**

**Semester: (Spring, Year: 2025)**

**PROJECT REPORT**

**Course Title**: Algorithm Lab

**Course Code:** CSE266.14

**Batch**: 65

**Project Title: Developing a system for Ticket Booking.**

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Shafiqul Islam | 2023200000211 |
| **2.** | Jubair Abdullah | 2023200000773 |
| **3.** | Md. Mahmud Hossain | 2023200000799 |

**Submission Date : 18-05-25**

**Course Teacher’s Name : Maisha Muntaha**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

1. **Introduction**

Ticket management systems are essential for organizing and maintaining reservations efficiently. In traditional setups, ticket booking is often handled manually, which can lead to problems such as overbooking, lack of centralized data, and poor user experience. This project aims to develop a comprehensive ticket booking system using the C programming language to demonstrate the practical application of data structures and algorithms. The system provides both user and admin functionalities through a terminal interface. It leverages queues, stacks, arrays, and linked lists for storage and management, along with sorting and optimization algorithms.

1. **Motivation**

Manual ticket booking systems are inherently limited. Errors from double-booking, inefficient seat management, and no historical tracking create inefficiencies. The motivation for this project stems from a need to:

* Create a structured and automated reservation process.
* Implement a maximum ticket limit per passenger to ensure fairness.
* Utilize classic data structures to maintain and manipulate data effectively.
* Provide admin tools for organizing ticket data and distributing resources optimally.

The goal is to offer a reliable, efficient, and user-friendly booking system that leverages the power of C programming and data structure concepts.

1. **Implementation**

**Tools and Environment**

* **IDE:** Code::Blocks
* **Compiler:** GCC

**Programming Language:**

* **C Program**

**Technologies:**

**Array:** Stores ticket information (ID, name, seat number).

**Queue:** Maintains First In First Out (FIFO), Manages ticket processing order.

**Stack:** Maintains Last In First Out (LIFO), helpful for history and control.

**Linked List:** Keeps a record of events in the order they happened.

**Sorting Algorithms:**

* + Insertion Sort (by seat number)
  + Selection Sort (by passenger name)
  + Merge Sort (by ticket ID)
  + Quick Sort (by cost per ticket relative to number of tickets)

**Fractional Knapsack:** Distributes VIP goodies to passengers based on cost-efficiency ratio.

**Workflow:**

1. **User Interface:**

User Interface

* Seat availability
* per-user limits enforced.
* Booking
* Displaying
* Searching
* Searching
* Canceling

1. **Admin Panel:**

Admin Panel

* Shows historical logs of actions using a linked list.
* Provides sorting options and VIP goodies distribution.
* Entry with password.

1. **Booking Process:**

Logs booking in a linked list.

Adds ticket to array, queue & stack.

Checks for seat availability.

Limits booking to 3 tickets per user.

Booking Process

1. **Cancelling Process:**

* Updates history with cancellation log.enforced.
* Searches and removes the ticket from the array.enforced.

Cancelling Process

1. **Sorting:**

Organize tickets

Applied for required cases.

Sorting

1. **Optimization:**

* Fractional knapsack used to distribute goodies among top passengers based on ticket ID and efficiency ratio.

Optimization

1. **Source Code:**

**//reg add HKCU\Console /v VirtualTerminalLevel /t REG\_DWORD /d 1**

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#include <conio.h>**

**#define MAX\_TICKETS 100 // Maximum number of tickets**

**#define MAX\_NAME\_LEN 50 // Maximum length of passenger name**

**#define MAX\_TICKETS\_PER\_PERSON 3 // Maximum tickets a person can book**

**#define ADMIN\_PASSWORD "0000" // Admin panel password**

**#define TICKET\_PRICE 200 // Price per ticket**

**// Color Functions**

**void reset()**

**{**

**printf("\033[0m");**

**}**

**void yellow()**

**{**

**printf("\033[0;33m");**

**}**

**void purple()**

**{**

**printf("\033[0;35m");**

**}**

**void red()**

**{**

**printf("\033[0;31m");**

**}**

**void green()**

**{**

**printf("\033[0;32m");**

**}**

**void cyan()**

**{**

**printf("\033[0;36m");**

**}**

**// Structures for Tickets, Queue, Stack, and Linked List**

**struct Ticket**

**{**

**int ticket\_id;**

**char passenger\_name[MAX\_NAME\_LEN];**

**int seat\_number;**

**};**

**struct Ticket tickets[MAX\_TICKETS];**

**int ticket\_count = 0;**

**// Queue Structure**

**struct TicketQueue**

**{**

**struct Ticket data[MAX\_TICKETS];**

**int front;**

**int rear;**

**} queue = { .front = -1, .rear = -1 };**

**// Stack Structure**

**struct TicketStack**

**{**

**struct Ticket data[MAX\_TICKETS];**

**int top;**

**} stack = { .top = -1 };**

**// Linked List for History**

**struct HistoryNode**

**{**

**char action[100];**

**struct HistoryNode\* next;**

**} \*history\_head = NULL;**

**// All functions starts from here**

**// Queue Functions**

**void enqueueTicket(struct Ticket ticket)**

**{**

**if ((queue.rear + 1) % MAX\_TICKETS == queue.front)**

**{**

**red();**

**printf("Tickets not Available.\n"); //Queue overflow**

**reset();**

**}**

**if (queue.front == -1) queue.front = 0;**

**queue.rear = (queue.rear + 1) % MAX\_TICKETS;**

**queue.data[queue.rear] = ticket;**

**}**

**// Stack Functions**

**void pushToStack(struct Ticket ticket)**

**{**

**if (stack.top == MAX\_TICKETS - 1)**

**{**

**red();**

**printf("Please wait until anyone cancel a ticket.\nHave a nice day!");//Stack overflow**

**reset();**

**getch();**

**exit(0);**

**}**

**stack.data[++stack.top] = ticket;**

**}**

**// Linked List Functions**

**void addToHistory(const char\* action)**

**{**

**struct HistoryNode\* new\_node = (struct HistoryNode\*)malloc(sizeof(struct HistoryNode));**

**strncpy(new\_node->action, action, 99);**

**new\_node->action[99] = '\0';**

**new\_node->next = history\_head;**

**history\_head = new\_node;**

**}**

**void displayHistory()**

**{**

**struct HistoryNode\* current = history\_head;**

**if (!current)**

**{**

**red();**

**printf("No actions in history.\n");**

**reset();**

**return;**

**}**

**purple();**

**printf("\nAction History:\n");**

**reset();**

**while (current)**

**{**

**printf("%s\n", current->action);**

**current = current->next;**

**}**

**}**

**// Ticket Booking Functions**

**int isSeatAvailable(int seat\_number)**

**{**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**if (tickets[i].seat\_number == seat\_number) return 0;**

**}**

**return 1;**

**}**

**int countTicketsByPerson(const char\* name)**

**{**

**int count = 0;**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**if (strcmp(tickets[i].passenger\_name, name) == 0) count++;**

**}**

**return count;**

**}**

**void ticket\_price(int total\_tickets)**

**{**

**int cost = TICKET\_PRICE \* total\_tickets;**

**printf("\nTotal cost: %d Taka\nPlease insert money: ", cost);**

**int payment;**

**scanf("%d", &payment);**

**while (payment < cost)**

**{**

**red();**

**printf("Insufficient amount! Add more money: ");**

**reset();**

**int extra;**

**scanf("%d", &extra);**

**payment += extra;**

**}**

**green();**

**printf("Payment successful!!\nReturned amount: %d Taka.\n", payment - cost);**

**reset();**

**}**

**void addTicket()**

**{**

**if (ticket\_count >= MAX\_TICKETS)**

**{**

**red();**

**printf("All tickets are booked.\n");**

**reset();**

**return;**

**}**

**char name[MAX\_NAME\_LEN];**

**int num\_tickets;**

**printf("\nEnter passenger name: ");**

**scanf("%s", name);**

**cyan();**

**printf ("\nThe price of each ticket is 200 taka only\n\n");**

**reset();**

**int current\_tickets = countTicketsByPerson(name);**

**if (current\_tickets >= MAX\_TICKETS\_PER\_PERSON)**

**{**

**red();**

**printf("You can only book up to %d tickets.\n", MAX\_TICKETS\_PER\_PERSON);**

**reset();**

**return;**

**}**

**printf("How many tickets (1-%d)? ", MAX\_TICKETS\_PER\_PERSON - current\_tickets);**

**scanf("%d", &num\_tickets);**

**if (num\_tickets < 1 || num\_tickets > (MAX\_TICKETS\_PER\_PERSON - current\_tickets))**

**{**

**red();**

**printf("Invalid number of tickets.\n");**

**reset();**

**return;**

**}**

**for (int i = 0; i < num\_tickets; i++)**

**{**

**int seat\_number;**

**printf("Enter seat number for ticket %d (1-99): ", i + 1);**

**scanf("%d", &seat\_number);**

**if(seat\_number <=0 || seat\_number >=100)**

**{**

**red();**

**printf ("Invalid seat number!!");**

**return;**

**}**

**else if (!isSeatAvailable(seat\_number))**

**{**

**red();**

**printf("Seat %d is already booked.\n", seat\_number);**

**reset();**

**i--;**

**continue;**

**}**

**struct Ticket new\_ticket = {ticket\_count + 1, "", seat\_number};**

**strncpy(new\_ticket.passenger\_name, name, MAX\_NAME\_LEN - 1);**

**tickets[ticket\_count++] = new\_ticket;**

**enqueueTicket(new\_ticket);**

**pushToStack(new\_ticket);**

**char action[MAX\_NAME\_LEN + 50];**

**sprintf(action, "Booked ticket: ID: %d, Name: %s, Seat: %d", new\_ticket.ticket\_id, name, seat\_number);**

**addToHistory(action);**

**green();**

**printf("Ticket booked successfully.\n");**

**reset();**

**}**

**ticket\_price(num\_tickets);**

**}**

**void displayTickets()**

**{**

**if (ticket\_count == 0)**

**{**

**red();**

**printf("No tickets booked yet.\n");**

**reset();**

**return;**

**}**

**purple();**

**printf("List of Tickets:\nID\tName\tSeat\n");**

**reset();**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**printf("%d\t%s\t%d\n", tickets[i].ticket\_id, tickets[i].passenger\_name, tickets[i].seat\_number);**

**}**

**}**

**void cancelTicket()**

**{**

**int seat\_number;**

**printf("Enter seat number to cancel: ");**

**scanf("%d", &seat\_number);**

**int found = 0;**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**if (tickets[i].seat\_number == seat\_number)**

**{**

**green();**

**printf("Cancelling ticket: ID: %d, Name: %s, Seat: %d\n", tickets[i].ticket\_id, tickets[i].passenger\_name, tickets[i].seat\_number);**

**reset();**

**printf("\nReturned amount: %d\n", TICKET\_PRICE);**

**// Adding to history**

**char action[MAX\_NAME\_LEN + 50];**

**sprintf(action, "Cancelled ticket: ID: %d, Name: %s, Seat: %d", tickets[i].ticket\_id, tickets[i].passenger\_name, tickets[i].seat\_number);**

**addToHistory(action);**

**// Shift all tickets one step to the left**

**for (int j = i; j < ticket\_count - 1; j++)**

**{**

**tickets[j] = tickets[j + 1];**

**}**

**green();**

**printf("\nThanks for using our system!\n");**

**reset();**

**ticket\_count--;**

**found = 1;**

**break;**

**}**

**}**

**if (!found)**

**{**

**red();**

**printf("No ticket found for seat number: %d\n", seat\_number);**

**reset();**

**}**

**}**

**/\*// Sorting Tickets**

**void sortTicketsBySeat()**

**{**

**for (int i = 0; i < ticket\_count - 1; i++)**

**{**

**for (int j = i + 1; j < ticket\_count; j++)**

**{**

**if (tickets[i].seat\_number > tickets[j].seat\_number)**

**{**

**struct Ticket temp = tickets[i];**

**tickets[i] = tickets[j];**

**tickets[j] = temp;**

**}**

**}**

**}**

**green();**

**printf("Tickets sorted by seat number.\n");**

**reset();**

**}**

**\*/**

**// Insertion Sort – Sort by seat number**

**void insertionSortBySeat()**

**{**

**for (int i = 1; i < ticket\_count; i++)**

**{**

**struct Ticket key = tickets[i];**

**int j = i - 1;**

**// Move elements of tickets[0..i-1] that are greater than key.seat\_number**

**// to one position ahead of their current position**

**while (j >= 0 && tickets[j].seat\_number > key.seat\_number)**

**{**

**tickets[j + 1] = tickets[j];**

**j = j - 1;**

**}**

**tickets[j + 1] = key;**

**}**

**green();**

**printf("Tickets sorted by seat number.\n");**

**reset();**

**}**

**// Add necessary sorting and fractional knapsack functions**

**// Selection Sort – Sort by passenger name**

**void selectionSortByName()**

**{**

**for (int i = 0; i < ticket\_count - 1; i++)**

**{**

**for (int j = i + 1; j < ticket\_count; j++)**

**{**

**if (strcmp(tickets[i].passenger\_name, tickets[j].passenger\_name) > 0)**

**{**

**struct Ticket temp = tickets[i];**

**tickets[i] = tickets[j];**

**tickets[j] = temp;**

**}**

**}**

**}**

**green();**

**printf("Tickets sorted by passenger name.\n");**

**reset();**

**}**

**// Merge Sort – Sort by ticket ID**

**void merge(struct Ticket arr[], int left, int right)**

**{**

**if (left >= right)**

**return;**

**int mid = left + (right - left) / 2;**

**merge(arr, left, mid);**

**merge(arr, mid + 1, right);**

**int n1 = mid - left + 1, n2 = right - mid;**

**struct Ticket L[n1], R[n2];**

**for (int i = 0; i < n1; i++)**

**L[i] = arr[left + i];**

**for (int j = 0; j < n2; j++)**

**R[j] = arr[mid + 1 + j];**

**int i = 0, j = 0, k = left;**

**while (i < n1 && j < n2)**

**{**

**if (L[i].ticket\_id <= R[j].ticket\_id)**

**arr[k++] = L[i++];**

**else**

**arr[k++] = R[j++];**

**}**

**while (i < n1)**

**arr[k++] = L[i++];**

**while (j < n2)**

**arr[k++] = R[j++];**

**}**

**void mergeSortByTicketID()**

**{**

**merge(tickets, 0, ticket\_count - 1);**

**green();**

**printf("Tickets sorted by ticket ID.\n");**

**reset();**

**}**

**// Quick Sort – Sort by cost/number of tickets**

**int compareTicketsByCostRatio(const void \*a, const void \*b)**

**{**

**struct Ticket \*ticket1 = (struct Ticket \*)a;**

**struct Ticket \*ticket2 = (struct Ticket \*)b;**

**float cost\_ratio1 = (float)TICKET\_PRICE / countTicketsByPerson(ticket1->passenger\_name);**

**float cost\_ratio2 = (float)TICKET\_PRICE / countTicketsByPerson(ticket2->passenger\_name);**

**return (cost\_ratio1 > cost\_ratio2) - (cost\_ratio1 < cost\_ratio2);**

**}**

**void quickSortByCost()**

**{**

**qsort(tickets, ticket\_count, sizeof(struct Ticket), compareTicketsByCostRatio);**

**green();**

**printf("Tickets sorted by cost/number of tickets.\n");**

**reset();**

**}**

**// Fractional Knapsack – VIP goodies distribution based on ticket ID or cost/value ratio**

**void fractionalKnapsack()**

**{**

**int knapsack\_capacity = 500; // Assuming a fixed knapsack capacity for VIP goodies**

**float total\_value = 0;**

**printf("\nVIP goodies allocation based on ticket ID or cost/value ratio:\n");**

**// Sort tickets by cost/number of tickets (this helps to maximize the value)**

**quickSortByCost();**

**// Allocate goodies based on the sorted order**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**if (knapsack\_capacity >= TICKET\_PRICE)**

**{**

**knapsack\_capacity -= TICKET\_PRICE;**

**total\_value += TICKET\_PRICE;**

**printf("Allocated goodies to Ticket ID %d, Value: %d Taka\n", tickets[i].ticket\_id, TICKET\_PRICE);**

**}**

**else**

**{**

**float fraction = (float)knapsack\_capacity / TICKET\_PRICE;**

**total\_value += fraction \* TICKET\_PRICE;**

**printf("Allocated partial goodies to Ticket ID %d, Fractional Value: %.2f Taka\n", tickets[i].ticket\_id, fraction \* TICKET\_PRICE);**

**break;**

**}**

**}**

**printf("\nTotal Value of allocated goodies: %.2f Taka\n", total\_value);**

**}**

**// Searching for a Ticket by Name**

**void searchTicketByName()**

**{**

**char name[MAX\_NAME\_LEN];**

**printf("Enter passenger name to search: ");**

**scanf("%s", name);**

**int found = 0;**

**for (int i = 0; i < ticket\_count; i++)**

**{**

**if (strcmp(tickets[i].passenger\_name, name) == 0)**

**{**

**green();**

**printf("Ticket Found:\nID: %d, Name: %s, Seat: %d\n", tickets[i].ticket\_id, tickets[i].passenger\_name, tickets[i].seat\_number);**

**reset();**

**found = 1;**

**}**

**}**

**if (!found)**

**{**

**red();**

**printf("No ticket found for passenger: %s\n", name);**

**reset();**

**}**

**}**

**// Admin Panel Update with Insertion Sort by Seat Number**

**void adminPanel()**

**{**

**char password[50];**

**printf("Enter admin password: ");**

**scanf("%s", password);**

**if (strcmp(password, ADMIN\_PASSWORD) == 0)**

**{**

**green();**

**printf("Access granted.\n");**

**reset();**

**int admin\_choice;**

**do**

**{**

**printf("\n~~~~ Admin Panel ~~~~\n\n");**

**printf("1. Display All Tickets\n");**

**printf("2. Sort Tickets by Seat Number\n");**

**printf("3. Sort Tickets by Passenger Name\n");**

**printf("4. Sort Tickets by Ticket ID\n");**

**printf("5. Sort Tickets by Cost/Number of Tickets\n");**

**printf("6. Discount Allocation (Fractional Knapsack)\n");**

**printf("7. View Total History\n");**

**printf("8. Exit Admin Panel\n");**

**printf("\nEnter your choice: ");**

**scanf("%d", &admin\_choice);**

**switch (admin\_choice)**

**{**

**case 1:**

**displayTickets();**

**break;**

**case 2:**

**insertionSortBySeat();**

**displayTickets();**

**break;**

**case 3:**

**selectionSortByName();**

**displayTickets();**

**break;**

**case 4:**

**mergeSortByTicketID();**

**displayTickets();**

**break;**

**case 5:**

**quickSortByCost();**

**displayTickets();**

**break;**

**case 6:**

**fractionalKnapsack();**

**break;**

**case 7:**

**displayHistory();**

**break;**

**case 8:**

**green();**

**printf("Exiting admin panel.\n");**

**reset();**

**break;**

**default:**

**red();**

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

**reset();**

**}**

**}**

**while (admin\_choice != 8);**

**}**

**else**

**{**

**red();**

**printf("Incorrect password. Access denied.\n");**

**reset();**

**}**

**}**

**//Main Function**

**int main()**

**{**

**int choice;**

**do**

**{**

**cyan();**

**printf("\n~~~ Ticket Booking System ~~~\n\n");**

**reset();**

**printf("1. Book a Ticket\n2. Display Tickets\n3. Cancel Ticket\n4. Search Ticket (Name)\n5. Admin Panel\n6. Exit\n");**

**printf("\nEnter choice: ");**

**scanf("%d", &choice);**

**switch (choice)**

**{**

**case 1:**

**addTicket();**

**break;**

**case 2:**

**displayTickets();**

**break;**

**case 3:**

**cancelTicket ();**

**break;**

**case 4:**

**searchTicketByName();**

**break;**

**case 5:**

**adminPanel();**

**break;**

**case 6:**

**green();**

**printf("\nGoodbye! Have a nice day.\n");**

**reset();**

**break;**

**default:**

**red();**

**printf("Invalid choice.\n");**

**reset();**

**}**

**}**

**while (choice != 6);**

**getch ();**

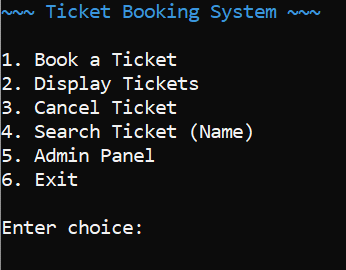
**return 0;**

**}**

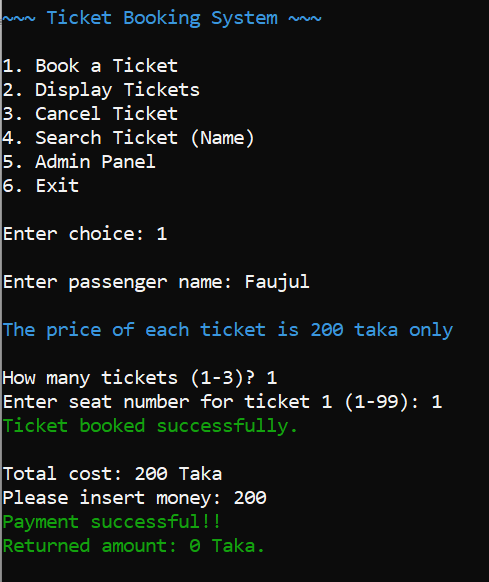
1. **Output**

The system generates outputs for all major operations including ticket booking, display, search, and sorting. Sample outputs:

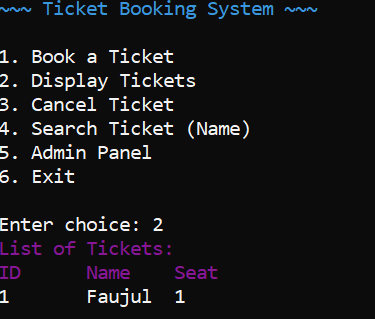
**Main Interface:**



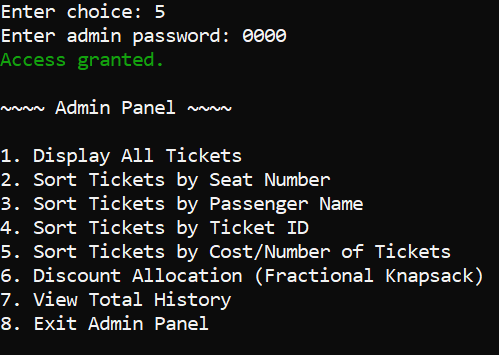
**Successful Booking:**



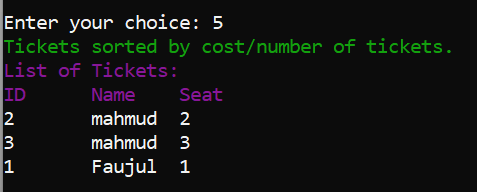
**Successful Displaying Ticket:**



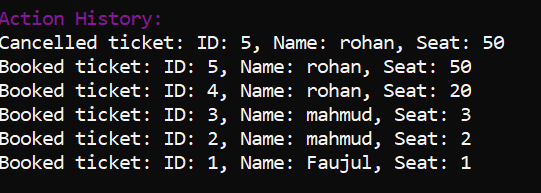
**Admin Panel:**



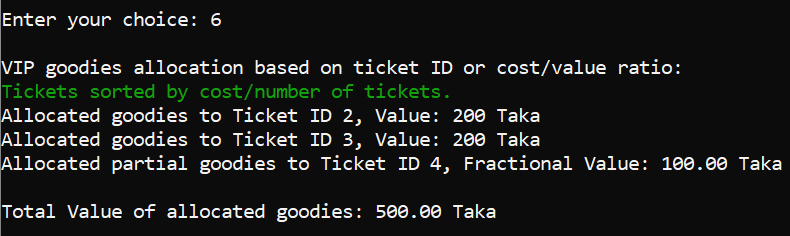
**Sort Tickets by Cost/Number of Tickets:**



**History:**



**Discount Allocation (Fractional Knapsack):**

****

1. **Conclusion**

This Ticket Booking System effectively illustrates the power of data structures in building a real-world application. Through the use of arrays, queues, stacks, and linked lists, the system manages reservations efficiently and transparently. Sorting algorithms allow for organized viewing, while the fractional knapsack implementation showcases optimization strategies in C. Despite being console-based, the system is modular, extensible, and ready for future upgrades like GUI integration or database support. It meets its educational and functional goals and serves as a foundational software solution for ticket management.