***PROJECT REPORT***

***PROJECT NAME: RAILWAY RESERVATION SYSTEM***

*GROUP MEMBER NAME: GAURAV BISHT, SUBHAM KUMAR OJHA, PRIYANSHU SINGH, ADITYA KUMAR*

*UID: 24BCA10134, 24BCA10402, 24BCA10132, 24BCA10139*

***SUMBITTED TO: MR DHARAMPAL SINGH***

**🚄 Railway Reservation System Using Queues in C – Project Report**

**📘 Introduction**

The Railway Reservation System is one of the most essential services provided by railway authorities to manage passenger bookings efficiently. This project demonstrates a simplified simulation of a reservation system using **data structures**, particularly **queues**, implemented in the **C programming language**. It manages real-world problems such as booking confirmation, waiting list maintenance, and cancellation handling.

Queues operate on the **FIFO (First-In-First-Out)** principle, which aligns perfectly with the ticket reservation process. This system uses **two circular queues**:

* One for **confirmed bookings** (Book Queue)
* Another for the **waiting list** (Wait Queue)

The system ensures fairness in service and efficient memory utilization, making it ideal for simulating ticket reservations in a controlled environment.

**💻 Code Overview (C Language)**

The project includes the following main components:

* **Structures:** Defines a Passenger struct with fields like ticketID, name, age, and gender.
* **Queue Logic:** Circular queue logic is used for both booked and waitlist queues to prevent memory overflow.
* **Booking Function:** Adds a passenger to the booked queue if seats are available or to the waiting list otherwise.
* **Cancellation Function:** Removes a booked ticket by ticketID and promotes the first passenger from the waitlist, if available.
* **Search Function:** Allows searching for a ticket by ticketID.
* **Display Function:** Shows all booked and waiting passengers.
* **Main Menu:** Provides an interactive menu for user actions.

The code is modular, uses standard input/output functions, and avoids dynamic memory allocation for simplicity and clarity.

***CODE:***

***#include <stdio.h>***

***#include <stdlib.h>***

***#include <string.h>***

***#define MAX\_TICKETS 5 // Maximum confirmed tickets***

***// Structure for storing passenger information***

***struct Passenger {***

***char name[50];***

***int age;***

***char gender[10];***

***};***

***// Global arrays for confirmed tickets and waiting list***

***struct Passenger confirmedList[MAX\_TICKETS];***

***struct Passenger waitingList[100];***

***int confirmedCount = 0;***

***int waitingCount = 0;***

***// Function to book a ticket***

***void bookTicket() {***

***struct Passenger p;***

***printf("\nEnter Passenger Name: ");***

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

***printf("Enter Age: ");***

***scanf("%d", &p.age);***

***printf("Enter Gender: ");***

***scanf("%s", p.gender);***

***if (confirmedCount < MAX\_TICKETS) {***

***confirmedList[confirmedCount++] = p;***

***printf("Ticket confirmed for %s!\n", p.name);***

***} else {***

***waitingList[waitingCount++] = p;***

***printf("No confirmed seats available. %s added to the waiting list.\n", p.name);***

***}***

***}***

***// Function to cancel a ticket***

***void cancelTicket() {***

***if (confirmedCount == 0) {***

***printf("\nNo bookings to cancel.\n");***

***return;***

***}***

***char name[50];***

***printf("\nEnter the name of the passenger to cancel: ");***

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

***int i, found = 0;***

***for (i = 0; i < confirmedCount; ++i) {***

***if (strcmp(confirmedList[i].name, name) == 0) {***

***found = 1;***

***for (int j = i; j < confirmedCount - 1; ++j) {***

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

***}***

***confirmedCount--;***

***printf("%s's ticket has been canceled.\n", name);***

***// Move a passenger from the waiting list to confirmed list if available***

***if (waitingCount > 0) {***

***confirmedList[confirmedCount++] = waitingList[0];***

***for (int k = 0; k < waitingCount - 1; ++k) {***

***waitingList[k] = waitingList[k + 1];***

***}***

***waitingCount--;***

***printf("%s moved from waiting list to confirmed list.\n", confirmedList[confirmedCount - 1].name);***

***}***

***break;***

***}***

***}***

***if (!found) {***

***printf("Passenger not found in confirmed bookings.\n");***

***}***

***}***

***// Function to view confirmed bookings***

***void viewConfirmed() {***

***printf("\nConfirmed Bookings:\n");***

***if (confirmedCount == 0) {***

***printf("No confirmed tickets.\n");***

***return;***

***}***

***for (int i = 0; i < confirmedCount; ++i) {***

***printf("%d. %s | Age: %d | Gender: %s\n", i + 1, confirmedList[i].name, confirmedList[i].age, confirmedList[i].gender);***

***}***

***}***

***// Function to view waiting list***

***void viewWaitingList() {***

***printf("\nWaiting List:\n");***

***if (waitingCount == 0) {***

***printf("No passengers in waiting list.\n");***

***return;***

***}***

***for (int i = 0; i < waitingCount; ++i) {***

***printf("%d. %s | Age: %d | Gender: %s\n", i + 1, waitingList[i].name, waitingList[i].age, waitingList[i].gender);***

***}***

***}***

***int main() {***

***int choice;***

***while (1) {***

***printf("\n===== Railway Reservation System =====");***

***printf("\n1. Book Ticket\n2. Cancel Ticket\n3. View Confirmed Bookings\n4. View Waiting List\n5. Exit\n");***

***printf("Enter your choice: ");***

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

***switch (choice) {***

***case 1:***

***bookTicket();***

***break;***

***case 2:***

***cancelTicket();***

***break;***

***case 3:***

***viewConfirmed();***

***break;***

***case 4:***

***viewWaitingList();***

***break;***

***case 5:***

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

***exit(0);***

***default:***

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

***}***

***}***

***return 0;***

***}***

**🖥️ Output Overview**

The system performs the following during execution:

1. **Booking:** Accepts user input and confirms booking or adds to waiting.
2. **Waiting List:** Once seats are full, the system correctly handles a queue of pending requests.
3. **Cancellation:** Cancelling a confirmed ticket automatically promotes the next in line from the waiting queue.
4. **Ticket Search:** Helps users locate booking status with a ticket ID.
5. **Display Status:** Provides a real-time view of the booking and waitlist queues.

**Sample Scenario**:

* 5 seats booked successfully
* 2 passengers added to waitlist
* 1 cancellation processed
* 1 waitlisted passenger promoted to confirmed list

This confirms queue functionality and real-life behavior mimicry.

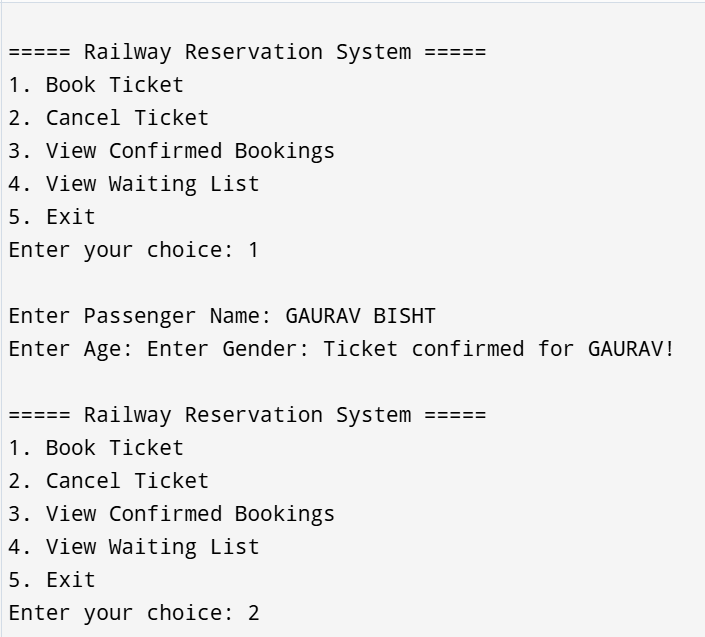
**🎓 Learning Outcome**

Through this project, the following concepts and skills were developed:

* **In-depth understanding of Queues**, particularly circular queue implementation in C.
* Practical use of **structures and arrays** to manage complex data.
* **Modular programming** and clean code practices.
* Application of DSA to real-world systems like reservations, scheduling, and customer service queues.
* Gained exposure to **logical problem solving**, **memory management**, and **user interface design in CLI**.

This project highlights how **Data Structures & Algorithms (DSA)** are not just academic concepts but core to building real, usable systems.

***OUTPUT IMAGE:***

******