Train Ticket Booking System in C

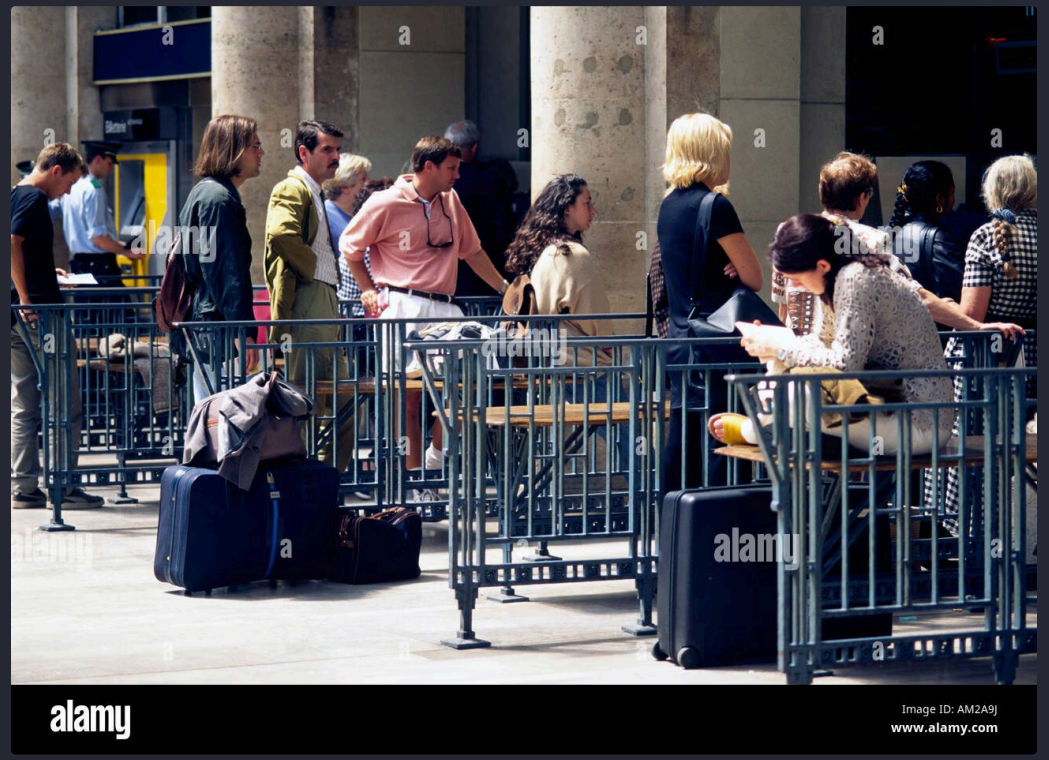
By Rudar Partap Singh

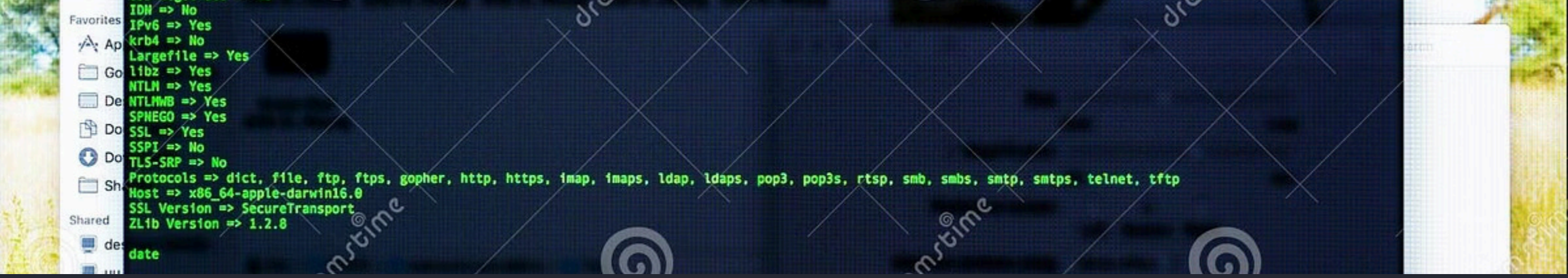
The Problem: Why Digital Booking Matters

Manual Booking Challenges

Traditional ticket booking processes are time-consuming and prone to human error. Long queues, paperwork, and manual data entry create bottlenecks that frustrate both passengers and staff.

For computer science students, understanding how to digitize real-world processes is essential. A booking system provides the perfect practical scenario to apply C programming concepts in a meaningful context.





The Solution: A Console-Based Booking Tool

01

Collect Passenger Information

Name, age, gender, contact details

02

Capture Journey Details

Source, destination, travel date

03

Generate Booking Ticket

Instant confirmation with all details

This console application streamlines the entire booking process into a simple, efficient digital workflow. By collecting six key inputs from users, the program generates a complete booking summary in seconds—transforming what once took minutes into an instantaneous operation.

System Flow: How the Program Works

Start Program

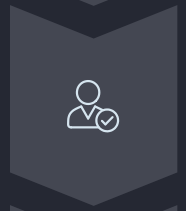
Initialize system

Get Passenger Details

Collect user inputs

Show Available Trains

Display options



Process Selection

Confirm booking

Print Ticket

Generate summary

The program follows a logical, sequential flow that mirrors real-world booking systems. Each step builds upon the previous one, ensuring data integrity and a smooth user experience from start to finish.

Smart Code Implementation: Safe Input Handling

Why fgets() Over scanf()?

Traditional scanf() functions fail when users enter names with spaces, causing program crashes and data loss. This is a critical flaw in production systems.

By implementing `fgets()` for string input, the program safely captures complete names like "Rudar Partap Singh" without truncation or errors. The custom `removeNewline()` function strips trailing newline characters, ensuring clean data processing.

```
fgets(passenger.name, 50, stdin);
removeNewline(passenger.name);
```

This approach demonstrates defensive programming—anticipating user behavior and building resilient code that handles edge cases gracefully.

Modular Architecture: Function-Based Design

getPassengerDetails()

Collects and validates all passenger information including name, age, gender, and contact number through safe input methods.

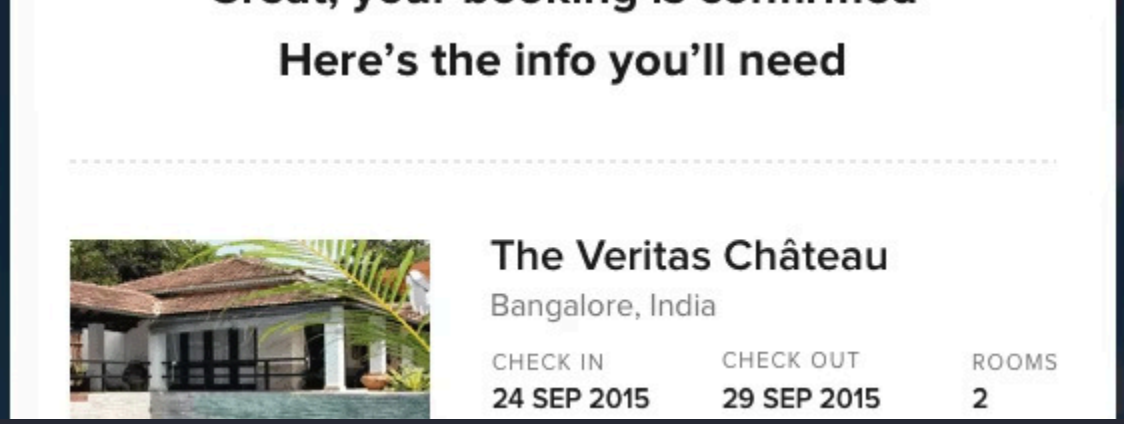
showTrains()

Displays available train options with route information, allowing users to make informed selection decisions.

processBooking()

Handles the booking logic, generates confirmation numbers, and formats the final ticket output.

Rather than cramming everything into main(), the code is organized into distinct, reusable functions. This modular approach improves readability, simplifies debugging, and makes future enhancements much easier to implement. Each function has a single, well-defined responsibility—a fundamental principle of clean code architecture.



The Demo: Booking Summary Output

Formatted Ticket Display

The booking summary presents all passenger details in a clean, professional format. Information is organized logically with clear labels, making it easy to verify at a glance.

Complete Information

Every critical detail is included: passenger name, age, gender, contact number, journey route (source to destination), travel date, and a unique booking reference number for tracking purposes.

The output demonstrates successful data flow through the entire system. Users receive immediate visual confirmation of their booking, with all information formatted in a readable, ticket-like structure that mimics real-world booking receipts.

Testing & Validation

Test Case Execution

Passenger: John Doe

Age: 32

Route: Delhi to Mumbai

Date: May 15, 2024

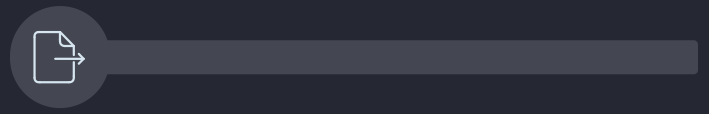
Status: ✓ Success

Validation Results

- Input handling: All fields accepted correctly
- String processing: Multi-word names captured fully
- Data flow: Information passed between functions accurately
- Output generation: Ticket formatted properly

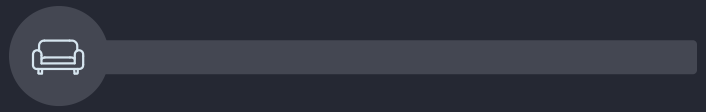
Rigorous testing with real-world data confirms the program's reliability. The system successfully handles various input scenarios, demonstrating robust logic and proper error handling throughout the booking workflow.

Future Improvements & Roadmap



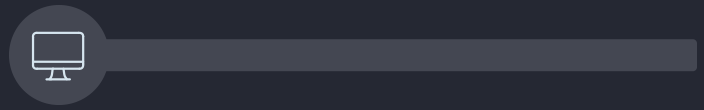
Persistent Data Storage

Implement file I/O operations to save booking records permanently. Currently, data exists only during runtime—adding CSV or database storage would enable booking history and retrieval.



Seat Availability System

Introduce dynamic seat tracking to prevent overbooking. Track remaining capacity per train and provide real-time availability feedback to users during the selection process.



Graphical User Interface

Transition from console to GUI using frameworks like GTK+ or Qt. A visual interface would dramatically improve usability and make the system accessible to non-technical users.

While the current implementation successfully demonstrates core C programming concepts, these enhancements would transform it from an educational project into a production-ready application.

Key Learnings & Takeaways

Pointer Mastery

Understanding memory addresses and how to pass data between functions efficiently using pointers and references.

String Handling

Working with character arrays, implementing safe input methods, and manipulating string data without buffer overflows.

Program Logic

Designing sequential workflows, managing state, and creating modular code architecture that's maintainable and scalable.

This project reinforced fundamental C programming principles while demonstrating how to build practical, real-world applications. The experience of handling user input safely, organizing code modularly, and thinking through edge cases has been invaluable for developing strong software engineering habits.

Thank you for your time and attention!