C++ Project: Airport Flight & Booking System

Student 1: Paunchici Raian

Student 2: Tudor Floristeanu

I. Project Description & Task Division

This project is a command-line-based airport management system developed in C++. It will not use any direct keyboard input (std::cin) and relies entirely on command-line arguments. The system is comprised of two core applications, with responsibilities split between two students.

- *Student 1: Flight Control (Backend)*
  - This student is responsible for the application that manages the airport's flight schedule.
  - The implementation will cover adding new flights, canceling existing flights, and updating flight details (e.g., gate number or departure time).
  - This application will also be used to view the passenger manifests for any given flight.
- *Student 2: Passenger Booking (Frontend)*
  - This student will develop the application that simulates a passenger's booking experience.
  - The implementation will allow a user to create a booking request for one or more passengers on a specific flight.
  - It will handle the final confirmation of a booking, which updates the flight's passenger list and generates a record.

II. Data Structures

The following C++ classes will be implemented to model the system's data. A key relationship is defined between the Flight and Passenger classes.

- *Passenger*: Represents a single person with a ticket.
  - string passportID
  - string fullName
- *Flight*: Represents a single flight route and its details.
  - string flightNumber
  - string destination
  - string departureTime
  - int capacity
  - Passenger[] passengers
- *Class Relationship: The *Flight* class has an **aggregation* relationship with the

Passenger class. Each Flight instance contains a collection of Passenger objects, representing the list of travelers booked on that flight. The Passenger objects can exist independently before being assigned to a flight.

III. File Structure

The two applications will communicate and persist data using a set of structured text files.

- *schedule.txt*
  - This file contains the master list of all scheduled flights.
  - The first line indicates the total number of flights.
  - Each subsequent line details a single flight: .
- *manifests.txt*
  - This file serves as a log for all confirmed passenger bookings.
  - For each flight with passengers, the file contains the flight number on one line, followed by a line with a list of passportIDs for all passengers on that flight.
- *booking.txt*
  - A temporary file used by the passenger application to stage a booking before confirmation.
  - The first line contains the flightNumber for the booking.
  - Each subsequent line contains the details for one passenger to be booked: .

IV. Application Command Reference

All system operations are executed through command-line arguments passed to the two applications.

- *Application 1 (flight_control.exe)*
  - ./flight_control.exe view_schedule: To display all scheduled flights from schedule.txt.
  - ./flight_control.exe add_flight : To add a new flight to the schedule.
  - ./flight_control.exe cancel_flight : To remove a flight from the schedule.
  - ./flight_control.exe update_time <new_time>: To change the departure time of a flight.
  - ./flight_control.exe view_manifest : To view the passenger list for a specific flight from manifests.txt.
- *Application 2 (passenger_app.exe)*
  - ./passenger_app.exe create_booking : To start a new booking or add a passenger to an existing one in booking.txt.
  - ./passenger_app.exe view_booking: To display the current booking details from booking.txt.

- ./passenger_app.exe cancel_booking: To clear the booking.txt file.
- ./passenger_app.exe confirm_booking: To finalize the booking. This action validates the request against the flight's capacity, updates the flight's available seats in schedule.txt, and appends the passenger information to manifests.txt