NUCES Airline Flight System (NAFS)

Submitted by:
Umar Mehmood
22I-2617
SE-D

Submitted to:
Sir Bilal Khalid Dar
OOP Lecturer
SE-D

CONTENTS

Overall Description

Explanation

The Project was to make a Flight Management System for a newly established airline named NUCES Airlines. I have implemented several functionalities in the NAFS according to the provided requirements. The system starts showing a main menu that repetitively displays itself until the exit option is selected by which the system turns off. Other options of the main menu includes Selection as an admin or as a passenger, see public information about NUCES Airlines and search flights.

The selection as admin and as passenger options takes the user to the admin and passenger menu respectively according to the selection user makes.

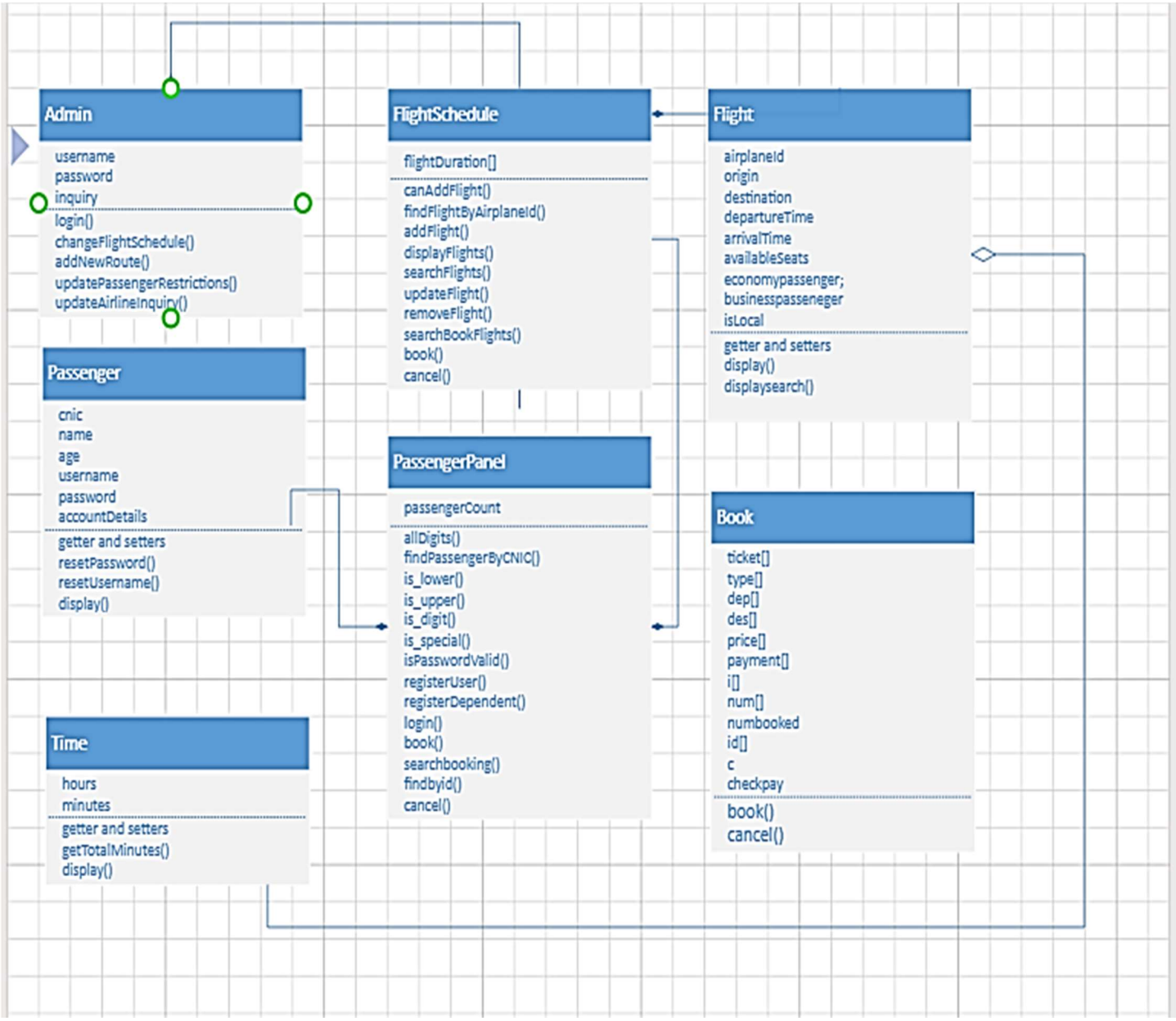
In the admin menu the user is first asked for the username and password which has been stored already for the admins. If the admin successfully enters correct username and password then the admin will be directed to the Admin Panel otherwise the user gets back to the main menu with a message that the user has entered incorrect username or password. In the Admin Panel, the admin is shown 5 options: Change flight schedule to change the flights schedule, add a new route, update the airline inquiry and logout. The change flight schedule option takes the admin to a new menu in which the admin can search a flight which is scheduled previously from a specific departure place, find a flight that where currently an airplane is, add a flight that adds a flight from a departure place to a destination place, update flight that updates already present flight, display flights that displays all the already scheduled flights and remove a flight that removes a flight from the schedule. Admin can add a new route by add new route option, update the airline inquiries and logout that will take the admin back to the main menu.

In the passenger menu the user has 6 options including the registration option to enter all the required information and entering the username and password according to the rules defined so that he can login afterwards, register dependent option for users having no CNIC can register on their parent/guardian CNIC, login that the user can do after registering, estimate the travelling cost by NUCES cost calculator and exit that takes back to the main menu.

After login the passenger is taken to a passenger panel that shows until the user logouts. The panel has check route details that will show the flights scheduled so that the user can decide accordingly, search a flight that allows the user to search all the flights scheduled from the departure place the user enters, check cost option again gives a chance to the user to check flight prices, book a flight books the user the number of seats in the airplane the user wishes for and generates a ticket, cancel flight option cancels the seat booked for the passenger with the deduction of 25% payment and logout that takes back to the passenger menu.

Class Diagram

The class diagram made on Microsoft visio has been added below:



Classes Description

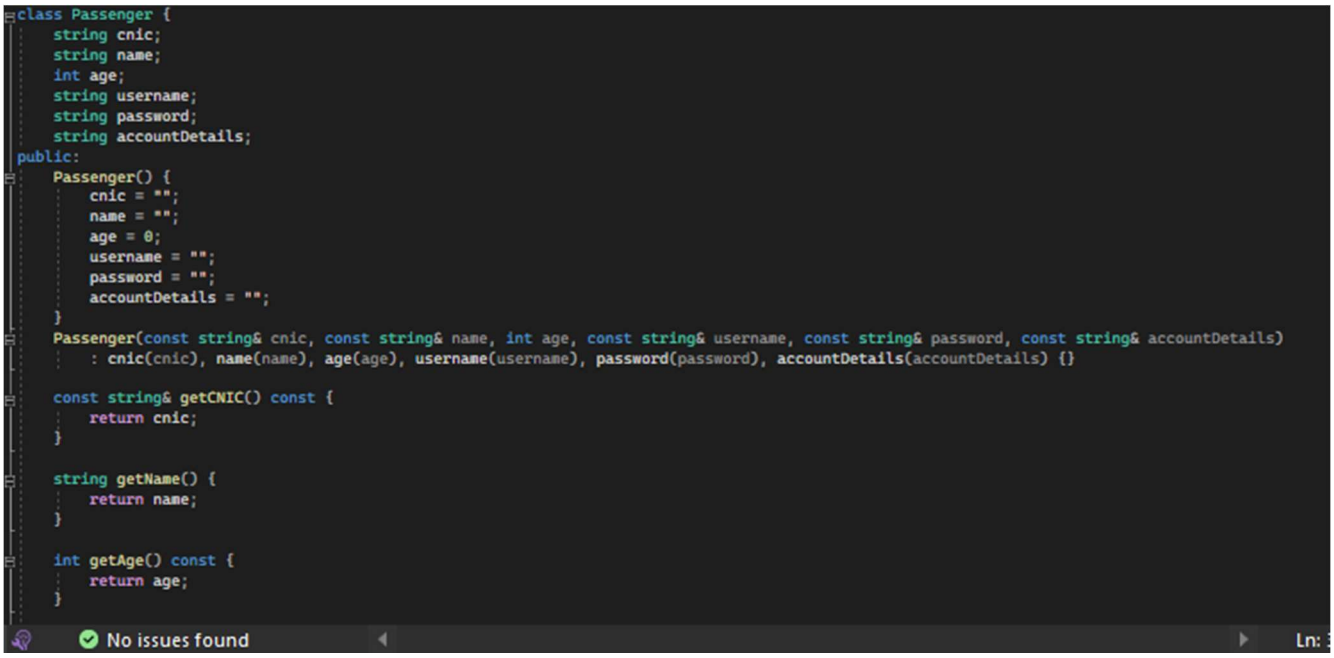
Admin Class

```
class Admin {
    FlightSchedule f;
    string username = "Umar";
    string password = "Umar@123";
    string inquiry;
public:
    Admin(){}
    Admin(const string& username, const string& password)
        : username(username), password(password) {}

    bool login(const string& username, const string& password) const {
        return this->username == username && this->password == password;
    }

    void changeFlightSchedule() {
        int choice = 0;
        // Code for changing flight schedule
        Sleep(1000);
        system("CLS");
        cout << "The flights already present are:" << endl;
        f.displayFlights();
        Sleep(4000);
        system("CLS");
        cout << "Select option to schhedule the Flight:" << endl;
        cout << "1.    Search a Flight" << endl;
        cout << "2.    Find a Flight" << endl;
        cout << "3.    Add a Flight" << endl;
        cout << "4.    Update a Flight" << endl;
        cout << "5.    Display Flights" << endl;
        cout << "6.    Remove a Flight" << endl;
        cin >> choice;
```

No issues found

Admin is one of the main classes of the management system. The Admin class has a composition of the FlightSchedule class so that the schedule management can be done. Admin has three variables of string type namely username, password and inquiry so that the admin can login. In the Admin class there are 3 main functions: changeFlightSchedule() that will further ask the admin to select an option from a menu and the switch statement will call the functions of FlightSchedule class that implements the requirements, addNewRoute() that will add a new route by calling a addFlight() function of FlightSchedule after getting the required information from the Admin, and updateAirlineIquiry() that will update the inquiry string.

Passenger Class



```
class Passenger {
    string cnic;
    string name;
    int age;
    string username;
    string password;
    string accountDetails;
public:
    Passenger() {
        cnic = "";
        name = "";
        age = 0;
        username = "";
        password = "";
        accountDetails = "";
    }
    Passenger(const string& cnic, const string& name, int age, const string& username, const string& password, const string& accountDetails)
        : cnic(cnic), name(name), age(age), username(username), password(password), accountDetails(accountDetails) {}

    const string& getCNIC() const {
        return cnic;
    }

    string getName() {
        return name;
    }

    int getAge() const {
        return age;
    }
```


The Passenger class has cnic, name, username, password, accountDetails variables of string type and age of int type. Passenger class has all the getter/setters and a display() function that displays the user's basic information.

PassengerPanel Class

PassengerPanel class has composition of Passenger and FlightSchedule class. It also has a variable of int type named passengerCount to count the number of passengers added. The class has various functions explained below: allDigits() function that takes password as a parameter and check if there is a number present in it or not return true if it has else returns false. findPassengerByCNIC() function takes cnic as input and compares it with all the passengers' cnic and return the iterator having cnic similar to the parameter else return -1. is_lower() functions takes a char as parameter and checks if it is a lower-case char or not returns true if it is else false. is_upper() is similar to is_lower the difference is that it checks for upper-case char. is_digit() checks for numbers but this time takes char as a parameter and checks a single

character. is_special() function checks if the char taken as parameter is a special character or not and returns true if it is. isPasswordValid() function takes as parameter the string password and checks if it is valid or not according to the rules defined in the project and by using the above functions and returns true if it is according to the rules. registerUser() function registers the user by taking the required information and storing it in the system. registerDependent() function registers the passenger having no cnic and registers on the behalf of an adult already registered. login() function compares the username and password send as parameters with the registered user's username and password and returns true if login is successful. book() function calls a FlightSchedule function to book a flight for a passenger. searchbooking() searches the airplaneId that has flight scheduled from the entered departure and destination place and returns the id. findbyid() function calls a FlightSchedule function namely findFlightByAirplaneId() which return the iterator having the id. cancel() function calls a FlightSchedule function named as cancel to cancel the flight booking and returns true if the booking is cancelled.

Time Class

```
class Time {
    int hours;
    int minutes;

public:
    Time() : hours(0), minutes(0) {}
    Time(int hours, int minutes) : hours(hours), minutes(minutes) {}

    void setTime(int hours, int minutes) {
        this->hours = hours;
        this->minutes = minutes;
    }

    int getHours() const { return hours; }
    void setHours(int hours) { this->hours = hours; }

    int getMinutes() const { return minutes; }
    void setMinutes(int minutes) { this->minutes = minutes; }

    int getTotalMinutes() const { return ((hours * 60) + minutes); }

    void display() const {
        cout << (hours < 10 ? "0" : "") << hours << ":" << (minutes < 10 ? "0" : "") << minutes;
    }
};

class Flight { ... };
```

Time class has two int type variables namely hours and minutes. This class only contains getter/setters and a display function that displays the time.

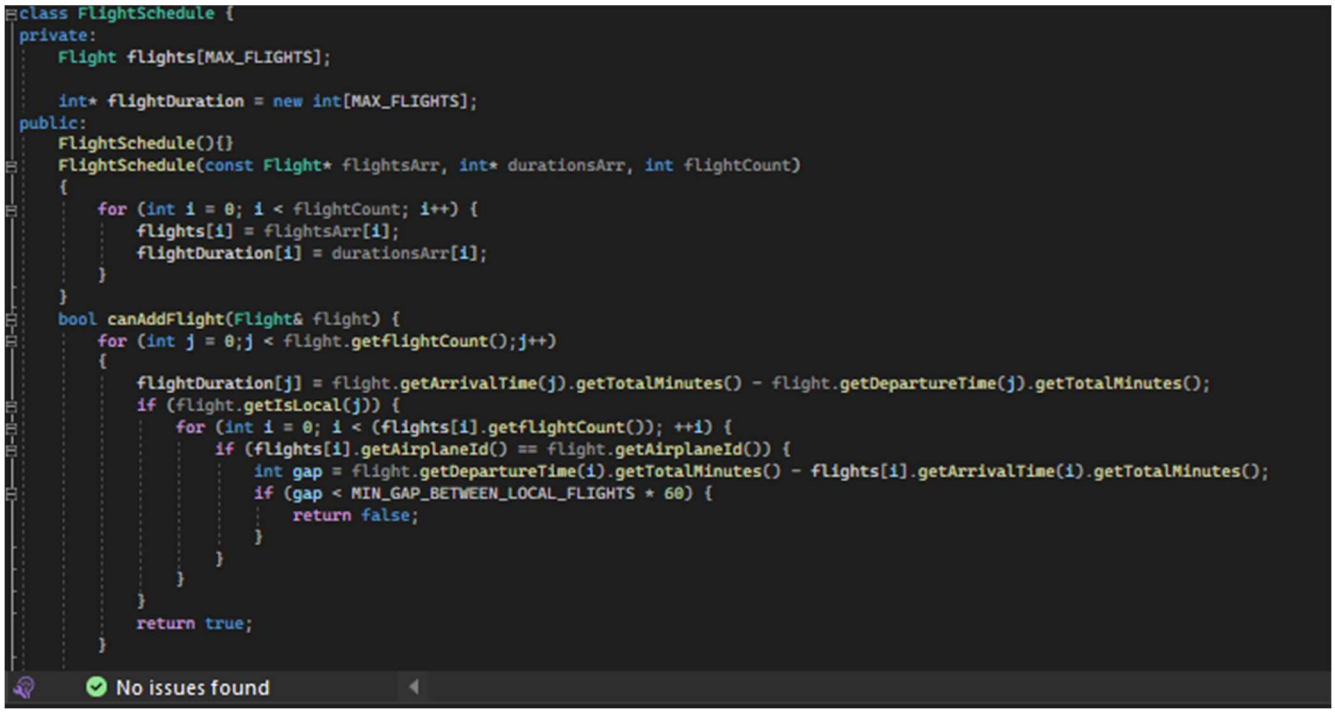
Flight Class

```
class Flight {
    string* airplaneId;
    string* origin;
    string* destination;
    Time* departureTime;
    Time* arrivalTime;
    int availableSeats;
    int economypassenger = 50;
    int businesspassenger = 10;
    bool* isLocal;


public:
    int flightCount = 0;
    Flight()
    {
        economypassenger = 50;
        businesspassenger = 10;
        airplaneId = new string[60];
        origin = new string[60];
        destination = new string[60];
        departureTime = new Time[60];
        arrivalTime = new Time[60];
        /*airplaneId = NULL;
        origin = NULL;
        destination = NULL;*/
        departureTime[0].setTime(1, 1);
        departureTime[1].setTime(2, 2);
        arrivalTime[0].setTime(3, 3);
        arrivalTime[1].setTime(3, 3);
        airplaneId[0] = { "123" };
```

No issues found Ln: 389 Ch: 45 SPC

There are dynamic arrays of string type namely airplaneId, origin, destination. Flight class has a composition of Time class making two objects namely departureTime and arrivalTime. Three variables namely availableSeats, economypassengers and businesspassengers and a variable of bool type namely isLocal is also present. A number of getter/setters and two functions are present. display() function displays all the flights scheduled and displaysearch() function displays only the information of the specific flight searched.

FlightSchedule Class

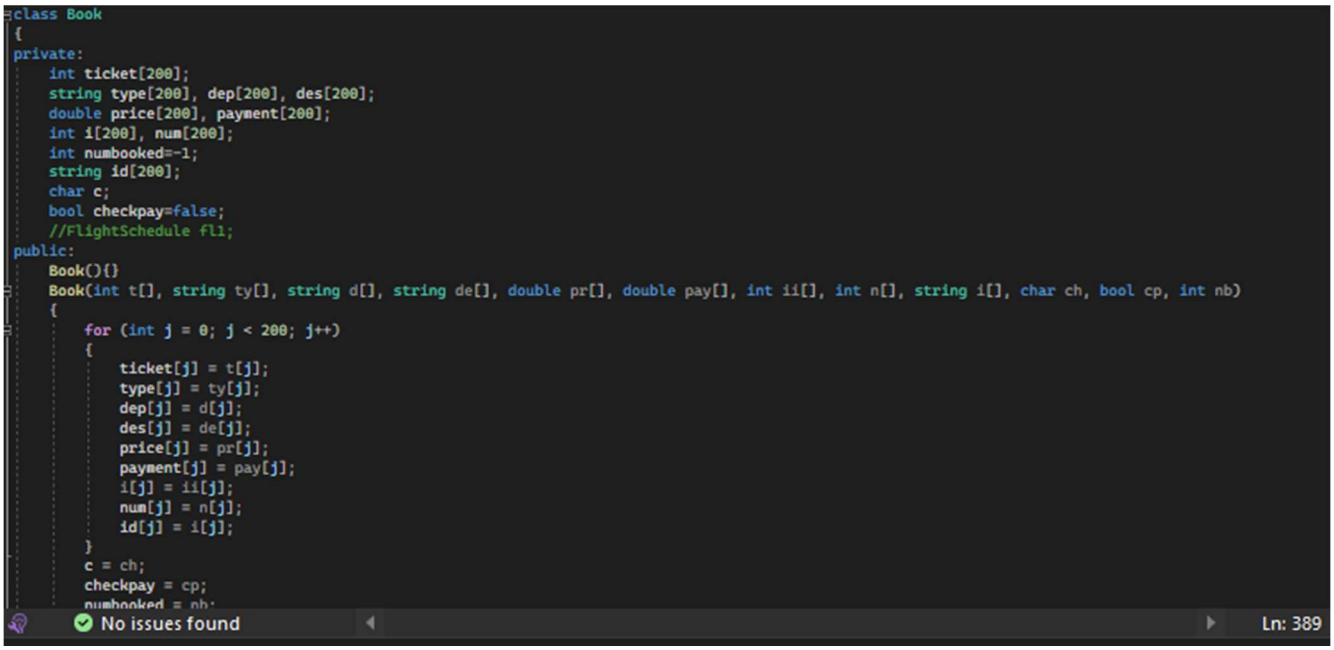


```
class FlightSchedule {
private:
    Flight flights[MAX_FLIGHTS];

    int* flightDuration = new int[MAX_FLIGHTS];
public:
    FlightSchedule(){}
    FlightSchedule(const Flight* flightsArr, int* durationsArr, int flightCount)
    {
        for (int i = 0; i < flightCount; i++) {
            flights[i] = flightsArr[i];
            flightDuration[i] = durationsArr[i];
        }
    }
    bool canAddFlight(Flight& flight) {
        for (int j = 0;j < flight.getflightCount();j++)
        {
            flightDuration[j] = flight.getArrivalTime(j).getTotalMinutes() - flight.getDepartureTime(j).getTotalMinutes();
            if (flight.getIsLocal(j)) {
                for (int i = 0; i < (flights[i].getflightCount()); ++i) {
                    if (flights[i].getAirplaneId() == flight.getAirplaneId()) {
                        int gap = flight.getDepartureTime(i).getTotalMinutes() - flights[i].getArrivalTime(i).getTotalMinutes();
                        if (gap < MIN_GAP_BETWEEN_LOCAL_FLIGHTS * 60) {
                            return false;
                        }
                    }
                }
            }
            return true;
        }
```

No issues found

FlightSchedule class has a one to many composition of Flight class and has a dynamic array of int type namely flightDuration. canAddFlight() function return true if a flight taken as parameter can be added according to certain rules else returns false. findFlightByAirplaneId() function takes a string as parameter and returns the location having the id. addFlight() function adds the flight given as parameter after checking from canAddFlight() function. displayFlights() function displays the flight information scheduled from the departure provided as parameter. updateFlight() function checks if a flight is present then updates the info of the flight else says that flight not found. removeFlight() function similarly searches for flight and removes it. searchBookFlights() function searches for the flight scheduled from the departure and destination places provided and prints the information regarding the specific flight. book() function after booking and subtracting available seats returns true else the available seats are less or the information provided is wrong returns false. cancel() function on the other hand adds the available seats and returns true after cancelling the booking.

Book Class



```
class Book
{
private:
    int ticket[200];
    string type[200], dep[200], des[200];
    double price[200], payment[200];
    int i[200], num[200];
    int numbooked=-1;
    string id[200];
    char c;
    bool checkpay=false;
    //FlightSchedule fli;
public:
    Book(){}
    Book(int t[], string ty[], string d[], string de[], double pr[], double pay[], int ii[], int n[], string i[], char ch, bool cp, int nb)
    {
        for (int j = 0; j < 200; j++)
        {
            ticket[j] = t[j];
            type[j] = ty[j];
            dep[j] = d[j];
            des[j] = de[j];
            price[j] = pr[j];
            payment[j] = pay[j];
            i[j] = ii[j];
            num[j] = n[j];
            id[j] = i[j];
        }
        c = ch;
        checkpay = cp;
        numbooked = nb;
```

No issues found Ln: 389

Book class has all the information stored regarding the booking. It has three int type arrays namely ticket, i, num, four string type arrays namely type, dep, des, id, two double type arrays namely price and payment. It also have one int type variable namely numbooked, one char type variable namely c and one bool type variable namely checkpay.

There are only two functions namely book() and cancel() both work by taking a PassengerPanel object as a parameter and calling it's functions and adding the important information in the arrays.