**Airlines Reservation System**

****

Session: 2021 – 2025

**Submitted by:**

**Muhammad Abdullah 2021-CS-82**

**Supervised by:**

**Dr. Awais Hassan**

Department of Computer Science

**University of Engineering and Technology**

**Lahore, Pakistan**

**Airlines Reservation System**

**Abstract:**

Airlines Reservation System is basically about the ticket reservation of the passenger and the admin manages its data. Admin also manages the flights. Passenger can enter data and then reserve the ticket if flight is available. I have tried my best to put all my efforts through implementing all possible functionalities in this system.

**Users:**

There are two users of this system:

* Admin
* Passenger

**Functional Requirements:**

**Passenger:**

As passenger, I can:

* 1. Register (Enter Personal Data)
  2. Enter Flight Details
  3. Book Flights
  4. View Booked Flights
  5. Cancel Flights

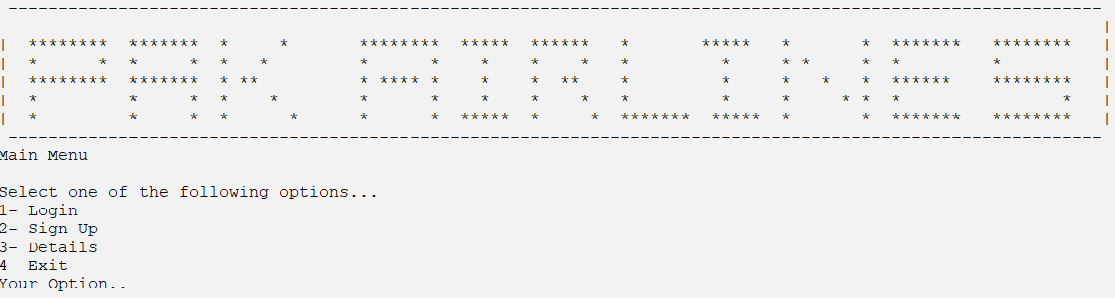
**Admin:**

As Admin, I can:

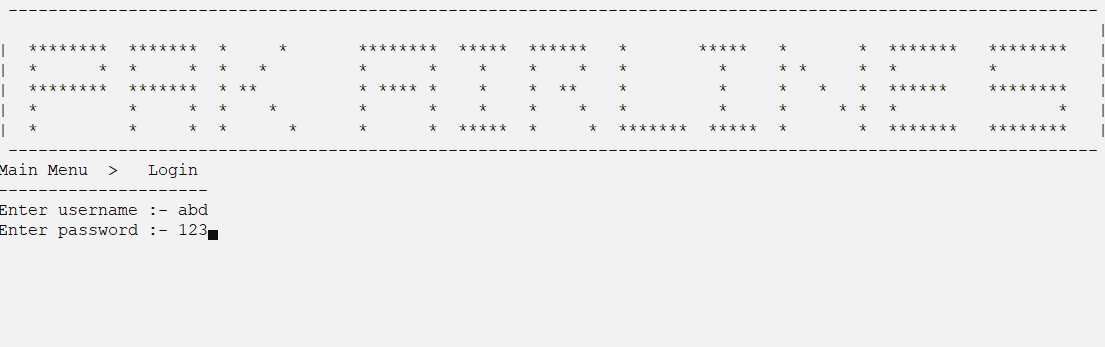
* 1. View Passengers Data
  2. View Passengers Travel Data
  3. Add new flights
  4. View Flights
  5. Allot Seats
  6. Order Passengers

**Wireframes:**

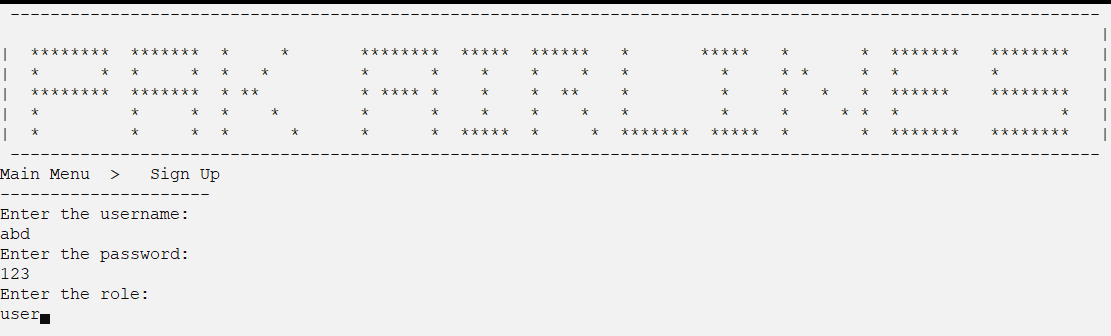
**Main Menu:**

****

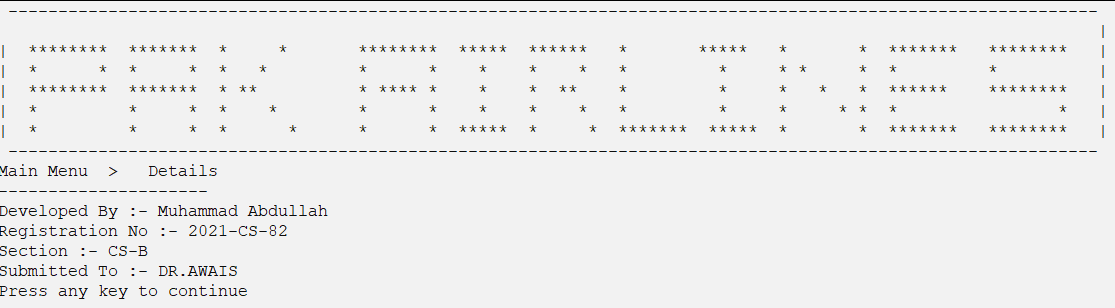
**Login Menu:**

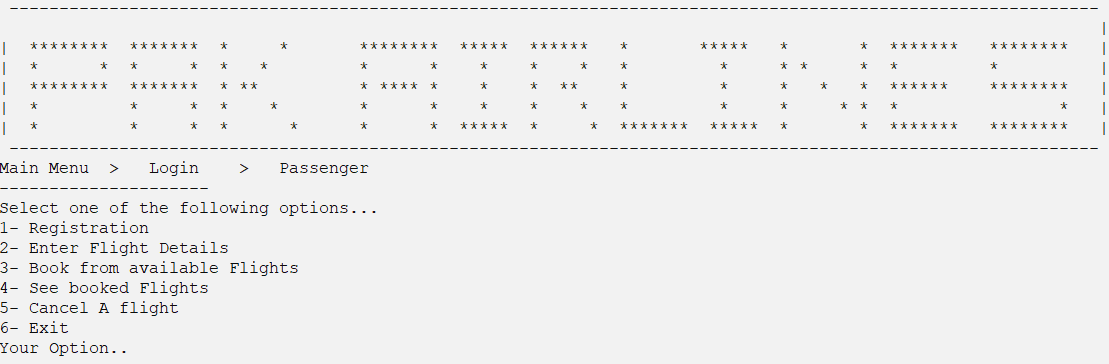
****

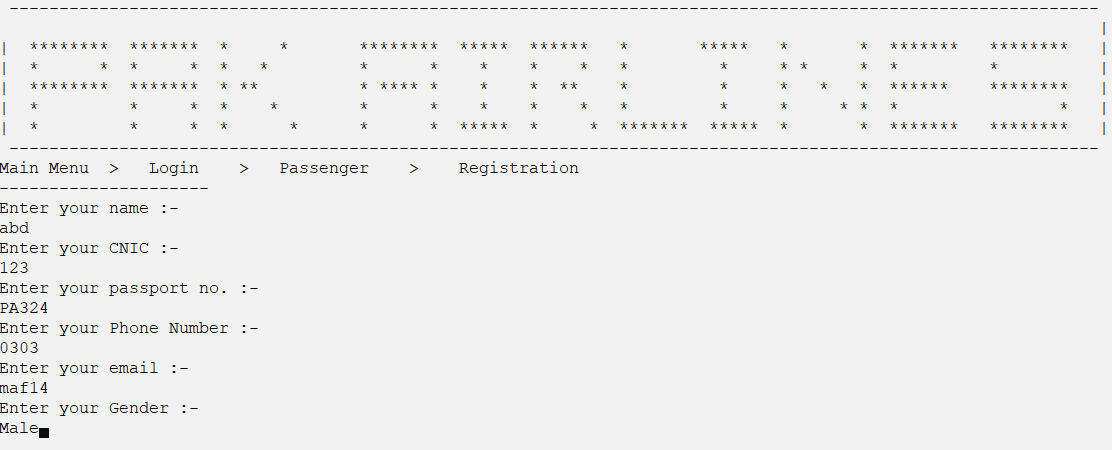
**Sign Up Menu:**

****

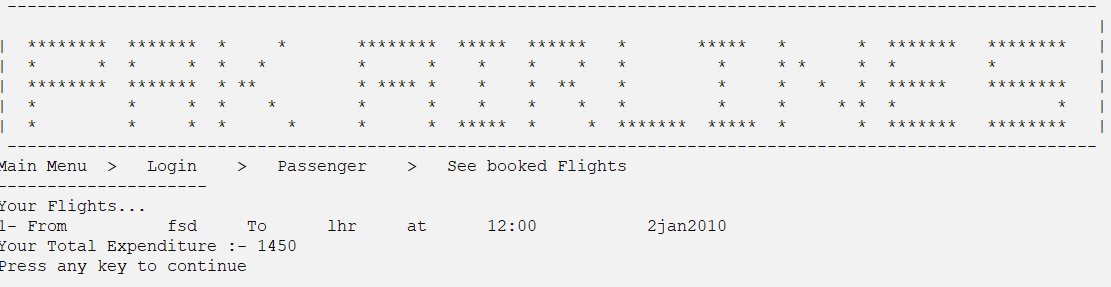
**Details:**

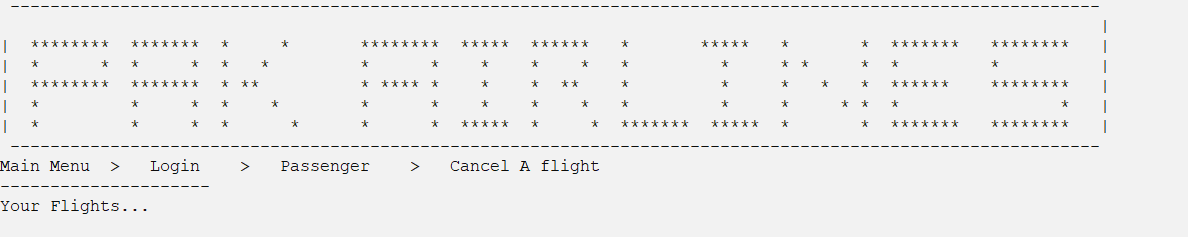
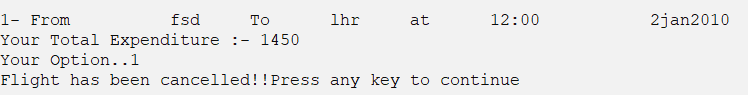
****

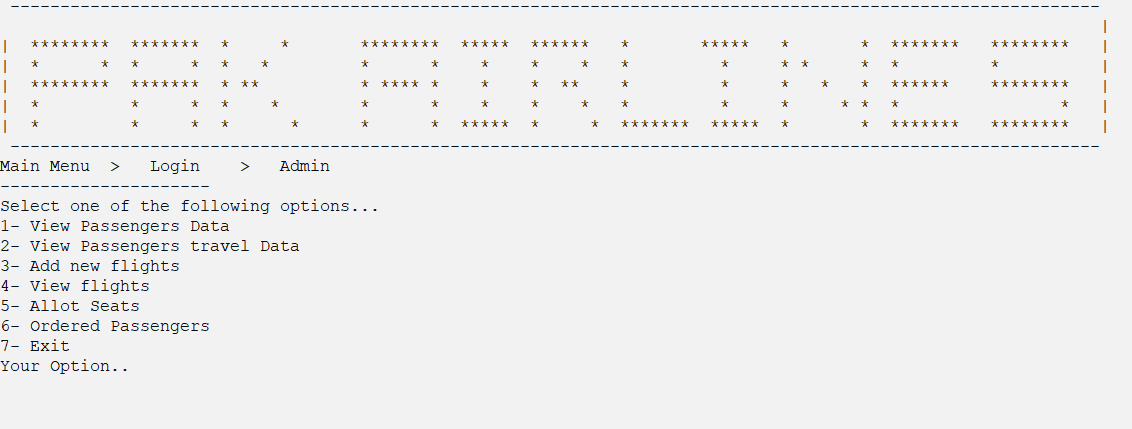
**Passenger Menu:**

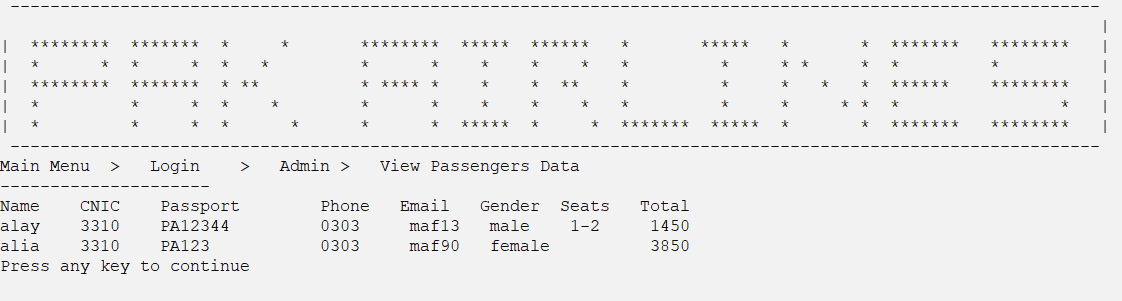
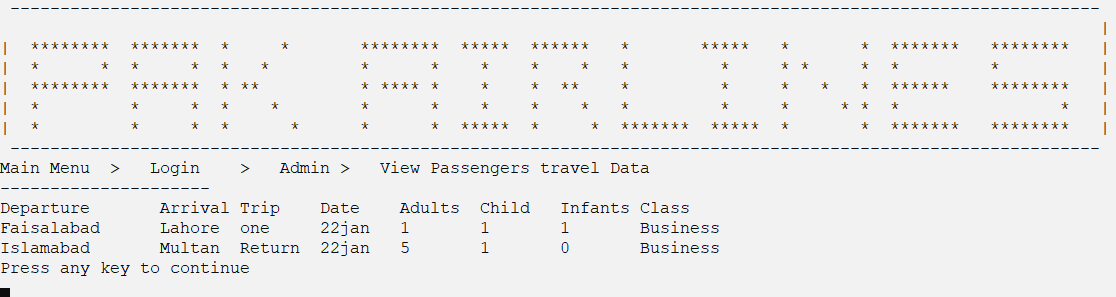
* **Registration:**
* **Enter Flight Details:**
* **Book Flights (If Available):**

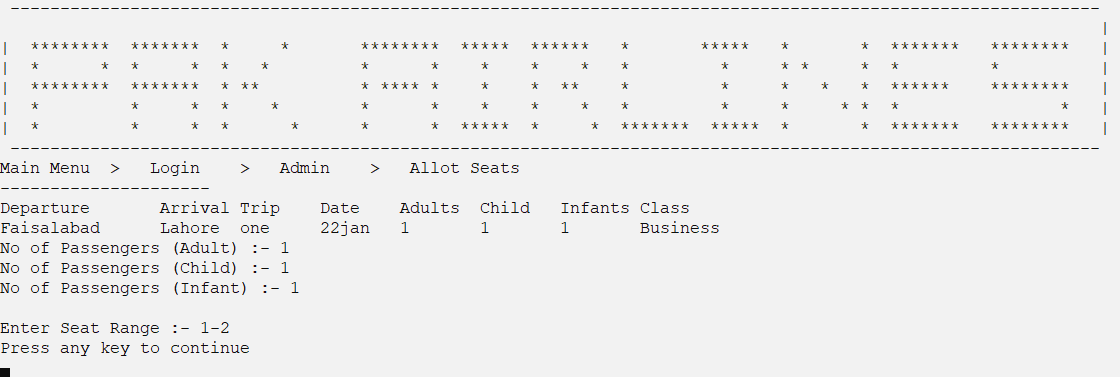


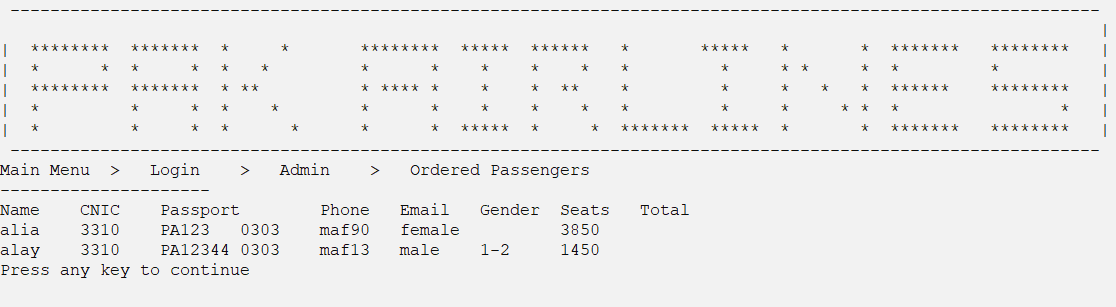
* **View Booked Flights:**
* **Cancel Booked Flights:**



**Admin Menu:**

* **View Passengers Data:**
* **View Passengers Travel Data:**

* **Add Flights:**
* **View Flights:**
* **Allot Seats:**
* **Ordered Passengers:**

****

**Data Structures:**

**Arrays:**

**Basic/main data structures**

* string usernameA[MAX\_RECORDS];
* string passwordsA[MAX\_RECORDS];
* string rolesA[MAX\_RECORDS];

**Passenger data structures**

* string pName[MAX\_RECORDS];
* string passport[MAX\_RECORDS];
* string cnic[MAX\_RECORDS];
* string mail[MAX\_RECORDS];
* string gender[MAX\_RECORDS];
* string pNum[MAX\_RECORDS];
* string pClass[MAX\_RECORDS];
* string pDepartCity[MAX\_RECORDS];
* string pArrCity[MAX\_RECORDS];
* string pTrip[MAX\_RECORDS];
* string pDepartDate[MAX\_RECORDS];
* int optionF[MAX\_RECORDS]; // Flight Option
* int adult[MAX\_RECORDS];
* int child[MAX\_RECORDS];
* int infant[MAX\_RECORDS];
* int totalSeats[MAX\_RECORDS];
* int total[MAX\_RECORDS]; // Expenditures
* string seating[MAX\_RECORDS];

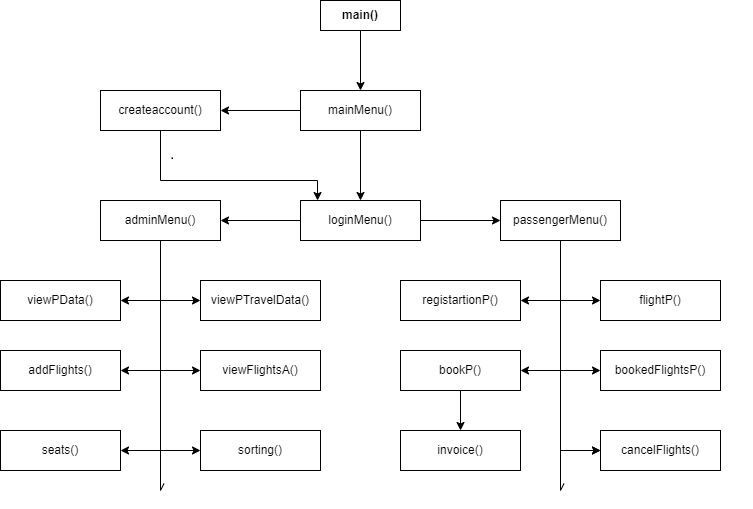
**Admin data structures**

* string aDepartDate[MAX\_RECORDS];
* string aDepartCity[MAX\_RECORDS];
* string aArrCity[MAX\_RECORDS];
* string aTrip[MAX\_RECORDS];
* string aDepartTime[MAX\_RECORDS];

**Functions:**

1. // Basic/Main Functions Prototypes
2. void header();
3. int mainMenu();
4. string loginMenu();
5. void clearScreen();
6. void storeData();
7. void storeCred();
8. void storeFlight();
9. void loadData();
10. int convert(string);
11. void addUser(string name, string pass, string role);
12. void createaccount();
13. // Passenger Functions Prototypes
14. int passengerMenu();
15. void registrationP();
16. void pDataIntoArray(string pname, string pcnic, string passport, string pnum, string pmail, string pgender);
17. void flightP();
18. void pFlightDataIntoArray(string departCity, string arrCity, string trip, string departDate, string pClassF, int adultF, int childF, int infantF, int totalSeatsF);
19. int bookP();
20. string parserecord(string line, int field);
21. void bookedFlightsP();
22. void invoice();
23. void cancelFlights();
24. // Admin Functions Prototypes
25. int adminMenu();
26. void viewPData();
27. void viewPTravelData();
28. void seats();
29. void addFlights();
30. void viewFlightsA();
31. void sorting();

**Flowchart:**

****

**Code:**

#include <iostream>

#include <conio.h>

#include <math.h>

#include <fstream>

using namespace std;

// Basic/Main Functions Prototypes

void header();

int mainMenu();

string loginMenu();

void clearScreen();

void storeData();

void storeCred();

void storeFlight();

void loadData();

int convert(string);

void addUser(string name, string pass, string role);

void createaccount();

// Passenger Functions Prototypes

int passengerMenu();

void registrationP();

void pDataIntoArray(string pname, string pcnic, string passport, string pnum, string pmail, string pgender);

void flightP();

void pFlightDataIntoArray(string departCity, string arrCity, string trip, string departDate, string pClassF, int adultF, int childF, int infantF, int totalSeatsF);

int bookP();

string parserecord(string line, int field);

void bookedFlightsP();

void invoice();

void cancelFlights();

// Admin Functions Prototypes

int adminMenu();

void viewPData();

void viewPTravelData();

void seats();

void addFlights();

void viewFlightsA();

void sorting();

// BASIC/MAIN DATA STRUCTURES

const int MAX\_RECORDS = 200;

string usernameA[MAX\_RECORDS];

string passwordsA[MAX\_RECORDS];

string rolesA[MAX\_RECORDS];

// PASSENGER DATA STRUCTURES

string pName[MAX\_RECORDS];

string passport[MAX\_RECORDS];

string cnic[MAX\_RECORDS];

string mail[MAX\_RECORDS];

string gender[MAX\_RECORDS];

string pNum[MAX\_RECORDS];

string pClass[MAX\_RECORDS];

string pDepartCity[MAX\_RECORDS];

string pArrCity[MAX\_RECORDS];

string pTrip[MAX\_RECORDS];

string pDepartDate[MAX\_RECORDS];

int optionF[MAX\_RECORDS]; // Flight Option

int adult[MAX\_RECORDS];

int child[MAX\_RECORDS];

int infant[MAX\_RECORDS];

int totalSeats[MAX\_RECORDS];

int total[MAX\_RECORDS]; // Expenditures

string seating[MAX\_RECORDS];

// int userCount = 0;

// int count = 0;

// ADMIN DATA STRUCTURES

string aDepartDate[MAX\_RECORDS];

string aDepartCity[MAX\_RECORDS];

string aArrCity[MAX\_RECORDS];

string aTrip[MAX\_RECORDS];

string aDepartTime[MAX\_RECORDS];

// float total[MAX\_RECORDS];

// Count Variables

int entry\_count = -1, flight\_count = -1, user\_count = 0, total\_count = 0;

string username;

main()

{

    loadData();

    string who = " ";

    // Options to select.. Variables

    int option = 0,

        optionP = 0,

        optionA = 0,

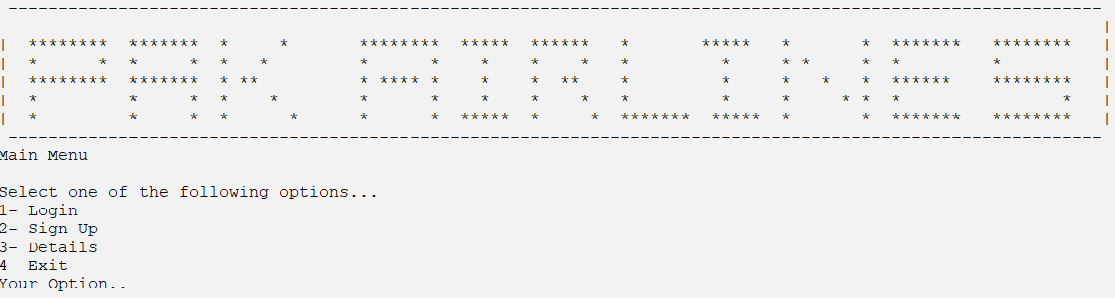
        totalSeats = 0;

    while (option < 4)

    {

        header();

        option = mainMenu();

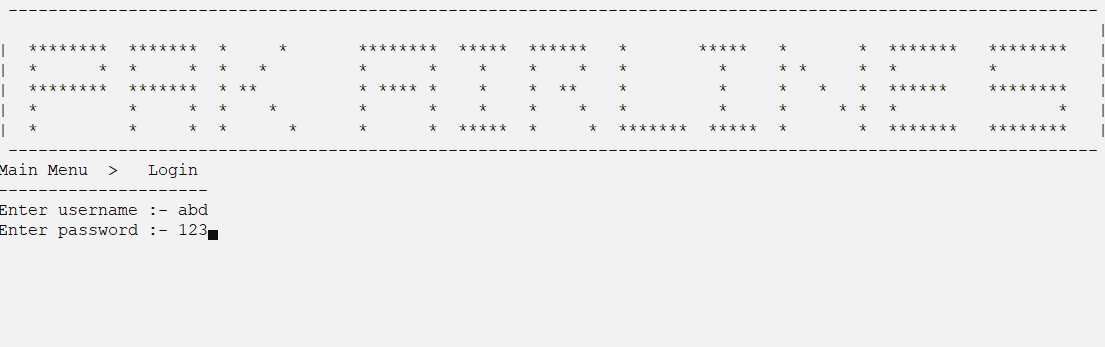


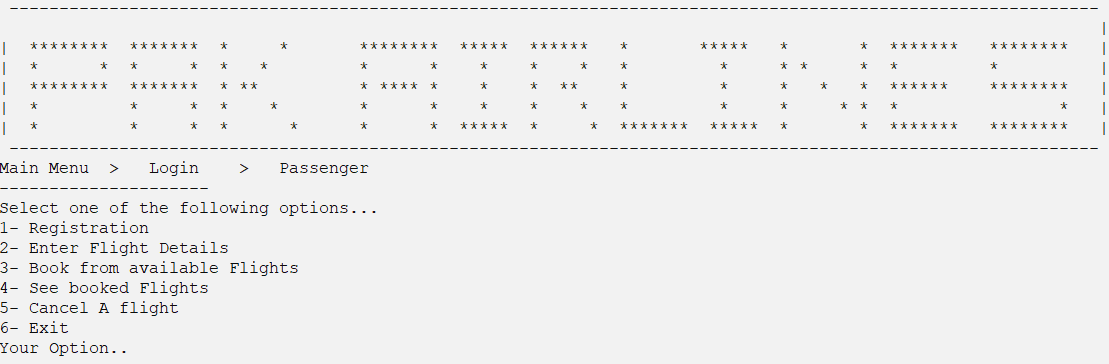
        if (option == 1)

        {

            header();

            who = loginMenu();

****

       // Passenger

            if (who == "user")

            {

                optionP = 0;

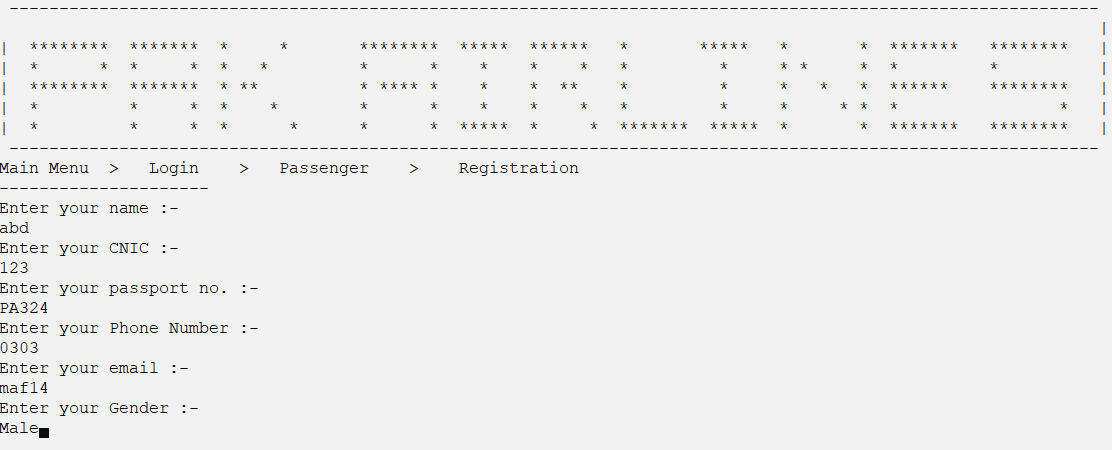
                while (optionP < 6)

                {

                    header();

                    optionP = passengerMenu();

                    // REGISTRATION



                    if (optionP == 1)

                    {

                        header();

                        registrationP();

                        clearScreen();

                    }

                    // Enter Flight Details



  else if (optionP == 2)

                    {

                        header();

                        flightP();

                        clearScreen();

                    }

                    // Book from available Flights



                    else if (optionP == 3)

                    {

                        header();

                        optionF[entry\_count] = bookP();

                        if (optionF[entry\_count] == 1)

                        {

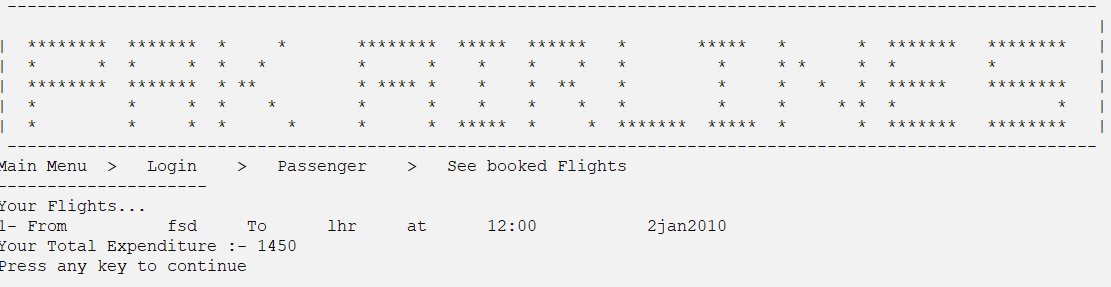
                            invoice();

                        }

                        clearScreen();

                    }

                    // View booked flights



                    else if (optionP == 4)

                    {

                        header();

                        bookedFlightsP();

                        clearScreen();

                    }

                    // Cancel booked flights



                    else if (optionP == 5)

                    {

                        header();

                        cancelFlights();

                        clearScreen();

                    }

                }

            }

            // ADMIN

            else if (who == "admin")

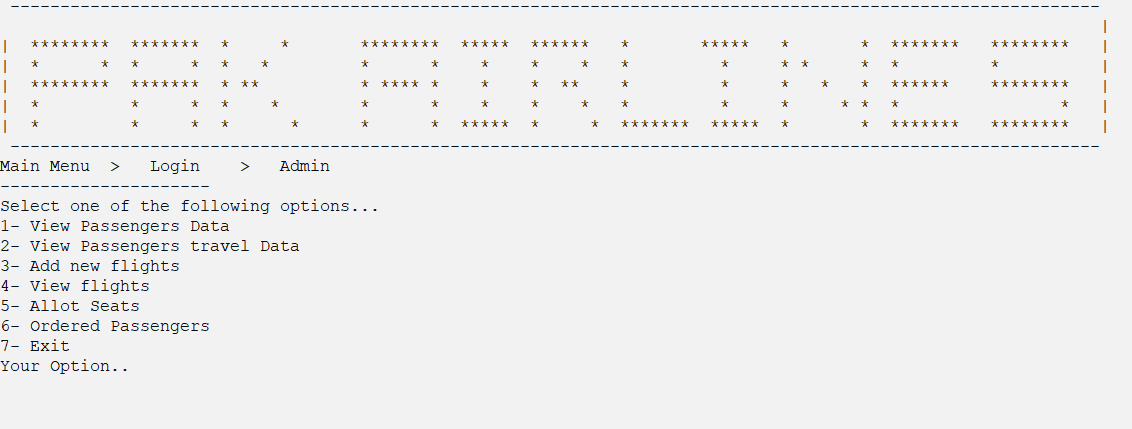
            {

                optionA = 0;

                while (optionA < 7)

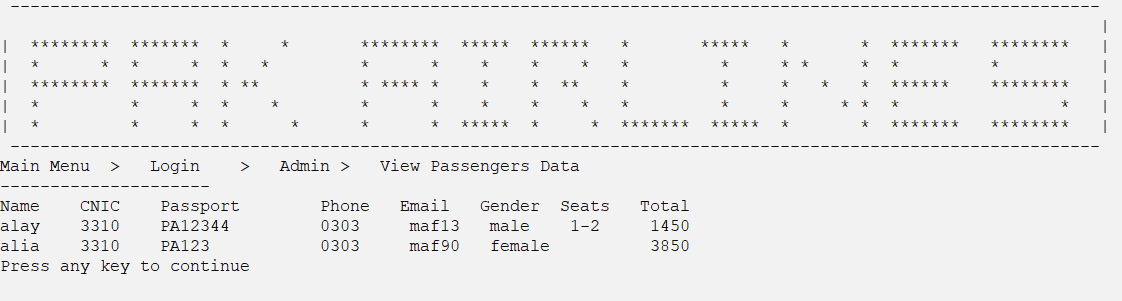
                {

                    header();



                    optionA = adminMenu();

                    // View Passengers Data



                    if (optionA == 1)

                    {

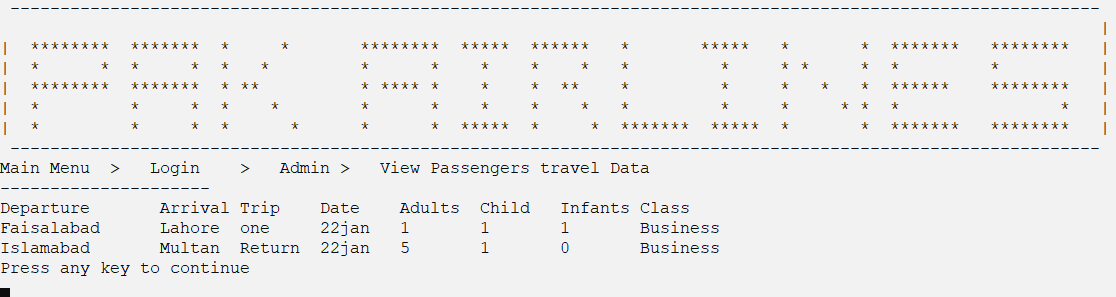
                        header();

                        viewPData();

                        clearScreen();

                    }

                    // View Passengers travel Data



                    else if (optionA == 2)

                    {

                        header();

                        viewPTravelData();

                        clearScreen();

                    }

                    // Add new flights



                    else if (optionA == 3)

                    {

                        header();

                        addFlights();

                        clearScreen();

                    }



                    else if (optionA == 4)

                    {

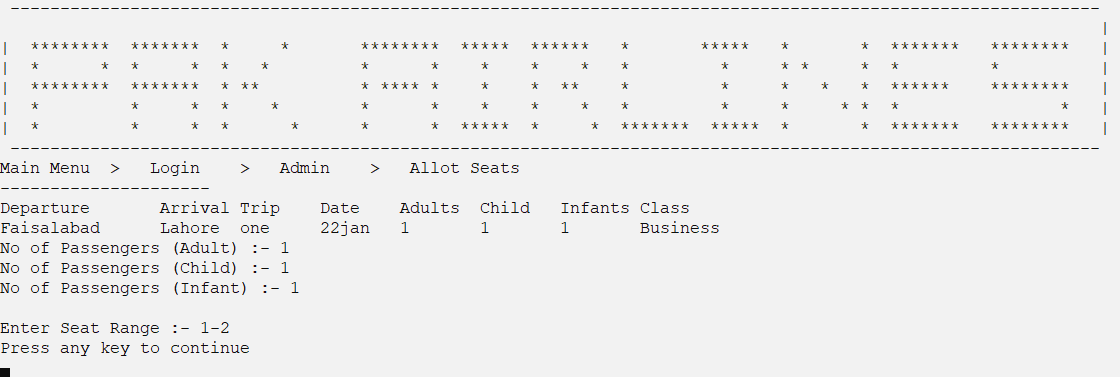
                        header();

                        viewFlightsA();

                        clearScreen();

                    }

                    // Allot Seats



                    else if (optionA == 5)

                    {

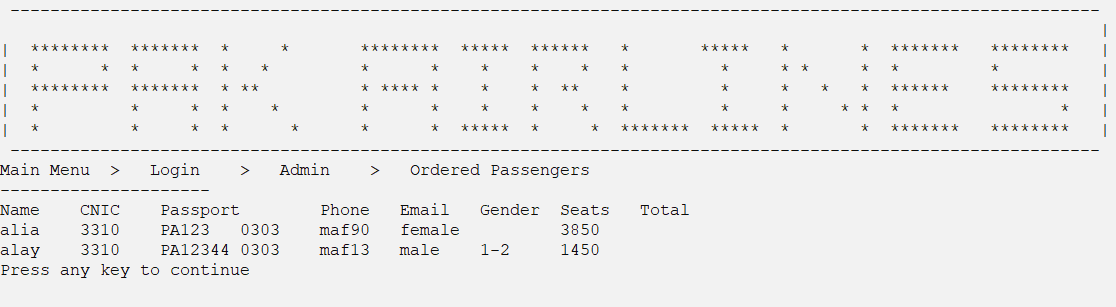
                        header();

                        seats();

                        clearScreen();

                    }

                    // Ordered Passengers

****

                    else if (optionA == 6)

                    {

                        header();

                        sorting();

                        clearScreen();

                    }

                }

            }

            else if (who == "EXIT")

            {

                cout << "Please Sign Up first" << endl;

                clearScreen();

            }

        }

        if (option == 2)

        {

            header();

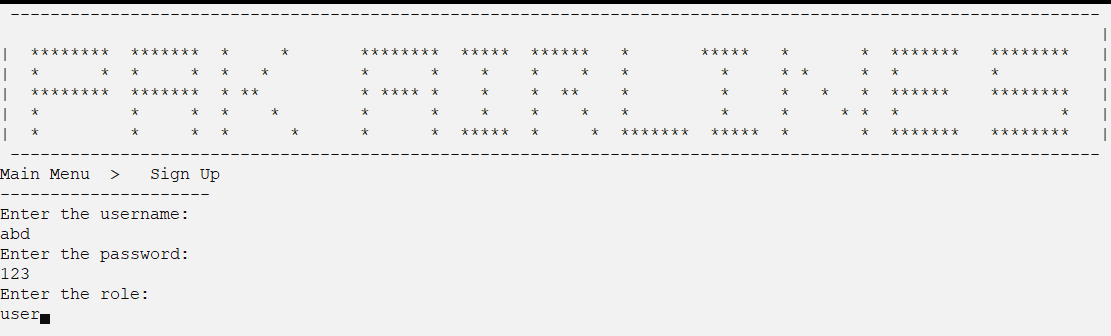
            cout << "Main Menu  >   Sign Up " << endl;

            cout << "---------------------" << endl;

            createaccount();

            clearScreen();

        }

****

 else if (option == 3)

        {

            header();

            cout << "Main Menu  >   Details " << endl;

            cout << "---------------------" << endl;

            cout << "Developed By :- Muhammad Abdullah" << endl;

            cout << "Registration No :- 2021-CS-82" << endl;

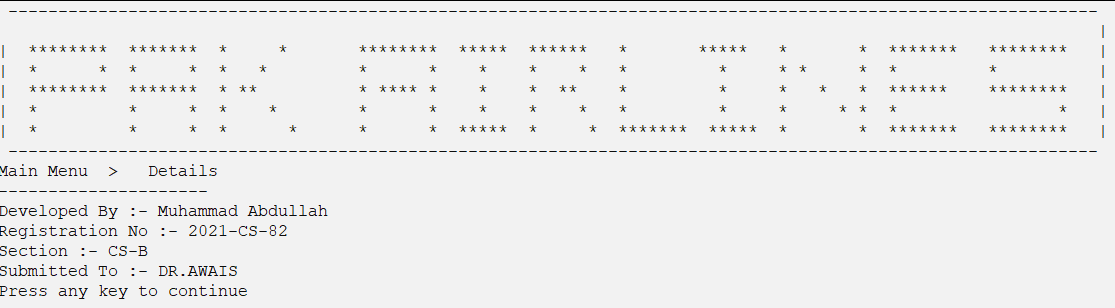
            cout << "Section :- CS-B" << endl;

            cout << "Submitted To :- DR.AWAIS" << endl;

            clearScreen();

        }

    }

****

    storeFlight();

    storeCred();

    storeData();

}

// END OF MAIN FUNCTION

void header()

{

    system("CLS");

    cout << " ------------------------------------------------------------------------------------------------------------- " << endl;

    cout << "                                                                                                              |" << endl;

    cout << "|  \*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*  \*     \*       \*\*\*\*\*\*\*\*  \*\*\*\*\*  \*\*\*\*\*\*   \*       \*\*\*\*\*   \*       \*  \*\*\*\*\*\*\*   \*\*\*\*\*\*\*\*   |" << endl;

    cout << "|  \*      \*  \*     \*  \*   \*         \*      \*    \*    \*    \*   \*         \*     \* \*     \*  \*         \*          |" << endl;

    cout << "|  \*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*  \* \*\*          \* \*\*\*\* \*    \*    \*  \*\*    \*         \*     \*   \*   \*  \*\*\*\*\*\*    \*\*\*\*\*\*\*\*   |" << endl;

    cout << "|  \*         \*     \*  \*    \*        \*      \*    \*    \*    \*   \*         \*     \*     \* \*  \*                \*   |" << endl;

    cout << "|  \*         \*     \*  \*      \*      \*      \*  \*\*\*\*\*  \*     \*  \*\*\*\*\*\*\*  \*\*\*\*\*  \*       \*  \*\*\*\*\*\*\*   \*\*\*\*\*\*\*\*   |" << endl;

    cout << " ------------------------------------------------------------------------------------------------------------- " << endl;

    // cout << entry\_count << endl;

}

// MAIN MENU

int mainMenu()

{

    int option;

    cout << "Main Menu " << endl;

    cout << "---------------------" << endl;

    cout << "Select one of the following options..." << endl;

    cout << "1- Login" << endl;

    cout << "2- Sign Up" << endl;

    cout << "3- Details" << endl;

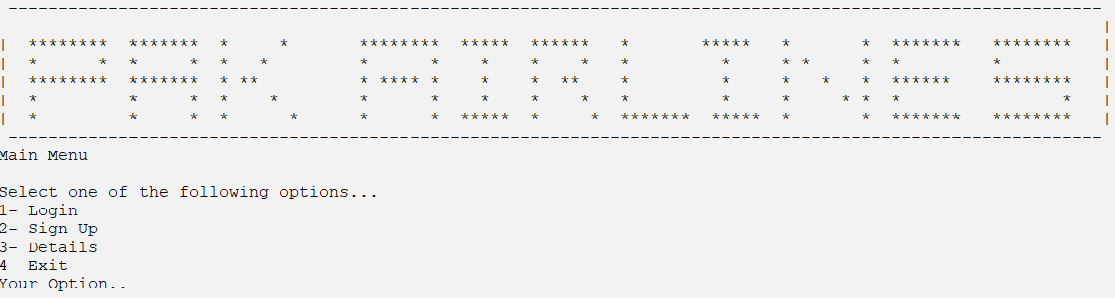
    cout << "4- Exit" << endl;

    cout << "Your Option..";

    cin >> option;

    return option;

}



string loginMenu()

{

    cout << "Main Menu  >   Login" << endl;

    cout << "---------------------" << endl;

    cout << "Enter username :- ";

    cin >> username;

    cout << "Enter password :- ";

    string pass;

    cin >> pass;

    for (int i = 0; i < MAX\_RECORDS; i++)

    {

        if (usernameA[i] == username && passwordsA[i] == pass)

        {

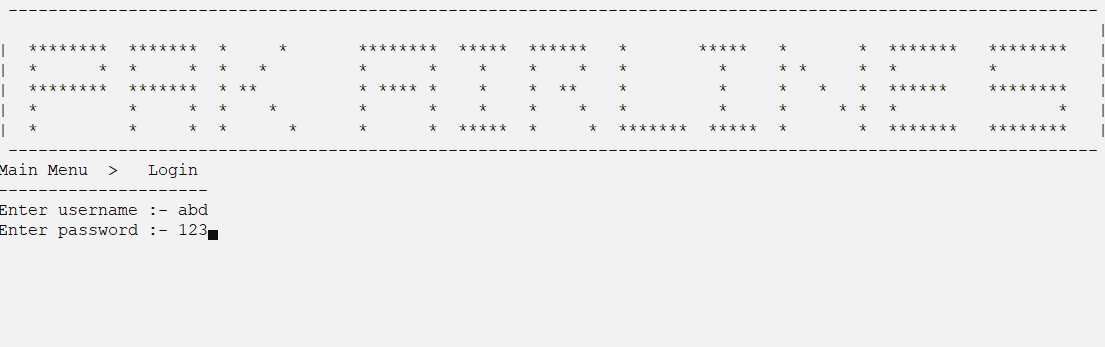
            return rolesA[i];

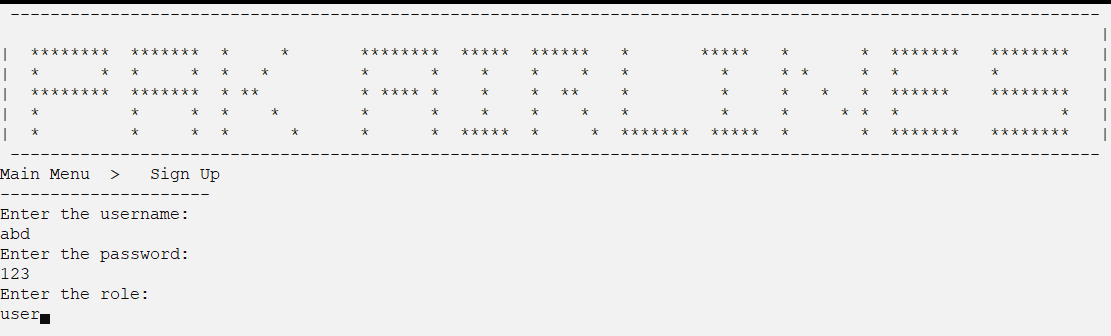
        }

    }

    return "EXIT";

}

****

****

void createaccount()

{

    bool flag = true;

    int c = 0, x;

    char ch;

    string user, pass, role;

    cout << "Enter the username: " << endl;

    cin >> user;

    while (flag)

    {

        cout << "Enter the password: " << endl;

        cin >> pass;

        for (int i = 0; i < pass.length(); i++)

        {

            ch = pass[i];

            x = ch;

            if (x <= 48 && x >= 57)

            {

                c++;

            }

        }

        if (c == pass.length() - 1)

        {

            flag = true;

            cout << "Password Should Contain a Number and should be more than 1 character" << endl;

        }

        else

        {

            flag = false;

        }

    }

    cout << "Enter the role: " << endl;

    cin >> role;

    addUser(user, pass, role);

}

void addUser(string username, string password, string role)

{

    if (total\_count < MAX\_RECORDS)

    {

        usernameA[total\_count] = username;

        passwordsA[total\_count] = password;

        rolesA[total\_count] = role;

        if (role == "user")

        {

            entry\_count++;

        }

        total\_count++;

        cout

            << endl

            << "Registered Successfully . You could login now" << endl

            << endl;

    }

    else

    {

        cout << "No More Space To Add Users." << endl;

    }

}

// MENU OF PASSENGER

int passengerMenu()

{

    int optionP;

    cout << "Main Menu  >   Login    >   Passenger" << endl;

    cout << "---------------------" << endl;

    cout << "Select one of the following options..." << endl;

    cout << "1- Registration" << endl;

    cout << "2- Enter Flight Details" << endl;

    cout << "3- Book from available Flights" << endl;

    cout << "4- See booked Flights " << endl;

    cout << "5- Cancel A flight  " << endl;

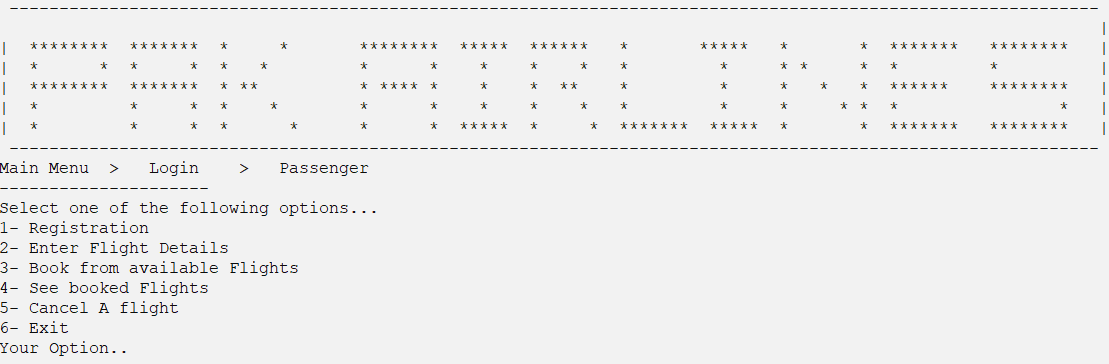
    cout << "6- Exit  " << endl;

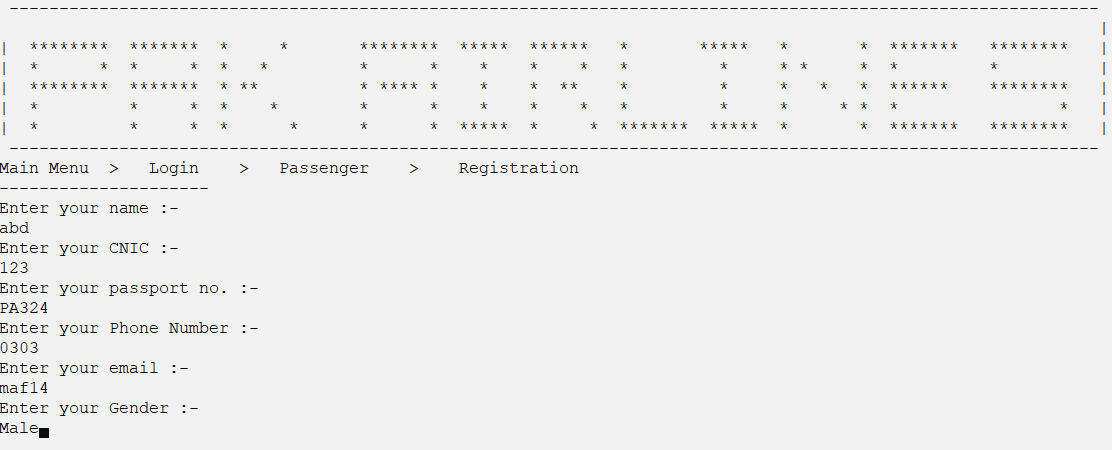
    cout << "Your Option..";

    cin >> optionP;

    return optionP;

}





// REGISTRATION OF PASSENGER

void registrationP()

{

    string pName, cnic, pNum, mail, gender, passport;

    if (entry\_count < MAX\_RECORDS)

    {

        cout << "Main Menu  >   Login    >   Passenger    >    Registration" << endl;

        cout << "---------------------" << endl;

        cout << "Enter your name :- " << endl;

        cin >> pName;

        cout << "Enter your CNIC :- " << endl;

        cin >> cnic;

        cout << "Enter your passport no. :- " << endl;

        cin >> passport;

        cout << "Enter your Phone Number :- " << endl;

        cin >> pNum;

        cout << "Enter your email :- " << endl;

        cin >> mail;

        cout << "Enter your Gender :- " << endl;

        cin >> gender;

        pDataIntoArray(pName, cnic, passport, pNum, mail, gender);

    }

    else

    {

        cout << "No more space to enter records";

    }

}

void pDataIntoArray(string pname, string pcnic, string passportx, string pnum, string pmail, string pgender)

{

    pName[entry\_count] = pname;

    cnic[entry\_count] = pcnic;

    passport[entry\_count] = passportx;

    pNum[entry\_count] = pnum;

    mail[entry\_count] = pmail;

    gender[entry\_count] = pgender;

}



// FLIGHT DETAILS OF PASSENGER

void flightP()

{

    string departCity, arrCity, trip, departDate, pClass;

    int adult, child, infant, totalSeats;

    if (entry\_count < MAX\_RECORDS)

    {

        cout << "Main Menu  >   Login    >   Passenger    >    Enter Flight Details" << endl;

        cout << "---------------------" << endl;

        cout << "Departure from :- " << endl;

        cin >> departCity;

        cout << "Arrival To :- " << endl;

        cin >> arrCity;

        cout << "Trip Type :- " << endl;

        cin >> trip;

        cout << "Depart Date :-" << endl;

        cin >> departDate;

        cout << "---------Passengers----------" << endl;

        cout << "No of Passengers (Adult) :- " << endl;

        cin >> adult;

        cout << "No of Passengers (Child) :- " << endl;

        cin >> child;

        cout << "No of Passengers (Infant) :- " << endl;

        cin >> infant;

        cout << "---------Class---------" << endl;

        cout << "Enter Class :- " << endl;

        cin >> pClass;

        totalSeats = adult + child;

        pFlightDataIntoArray(departCity, arrCity, trip, departDate, pClass, adult, child, infant, totalSeats);

    }

    else

    {

        cout << "No more space to enter records";

    }

}

void pFlightDataIntoArray(string departCity, string arrCity, string trip, string departDate, string pClassF, int adultF, int childF, int infantF, int totalSeatsF)

{

    pDepartCity[entry\_count] = departCity;

    pArrCity[entry\_count] = arrCity;

    pTrip[entry\_count] = trip;

    pDepartDate[entry\_count] = departDate;

    pClass[entry\_count] = pClassF;

    adult[entry\_count] = adultF;

    child[entry\_count] = childF;

    infant[entry\_count] = infantF;

    totalSeats[entry\_count] = totalSeatsF;

}

// Book from available Flights



int bookP()

{

    cout << "Main Menu  >   Login    >   Passenger    >   Book from available Flights" << endl;

    cout << "---------------------" << endl;

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

    {

   if (pDepartCity[entry\_count] == aDepartCity[i] && pArrCity[entry\_count] == aArrCity[i] && pDepartDate[entry\_count] == aDepartDate[i])

        {

            cout << "1- From \t " << pDepartCity[entry\_count] << "\t To \t " << pArrCity[entry\_count] << " \t at \t " << aDepartTime[i] << "\t \t " << pDepartDate[entry\_count] << endl;

            cin >> optionF[entry\_count];

            return optionF[entry\_count];

        }

        else

        {

            optionF[entry\_count] = 0;

        }

    }

    return optionF[entry\_count];

}

void invoice()

{

    cout << endl

         << "Departure "

         << "\t"

         << "Arrival"

         << "\t"

         << "Trip"

         << "\t"

         << "Date"

         << "\t"

         << "Adults"

         << "\t"

         << "Child"

         << "\t"

         << "Infants"

         << "\t"

         << "Class " << endl;

    cout << pDepartCity[entry\_count] << "\t" << pArrCity[entry\_count] << "\t" << pTrip[entry\_count] << "\t" << pDepartDate[entry\_count] << "\t" << adult[entry\_count] << "\t" << child[entry\_count] << "\t" << infant[entry\_count] << "\t" << pClass[entry\_count] << endl;

    cout << endl

         << "--------Expenditures---------" << endl;

    cout << "No of Passengers (Adult)(600$) :- " << adult[entry\_count] << endl;

    cout << "No of Passengers (Child) (450$):- " << child[entry\_count] << endl;

    cout << "No of Passengers (Infant)(0$) :- " << infant[entry\_count] << endl

         << endl;

    total[entry\_count] = (adult[entry\_count] \* 600) + (child[entry\_count] \* 450);

    if (pClass[entry\_count] == "Business" || pClass[entry\_count] == "business")

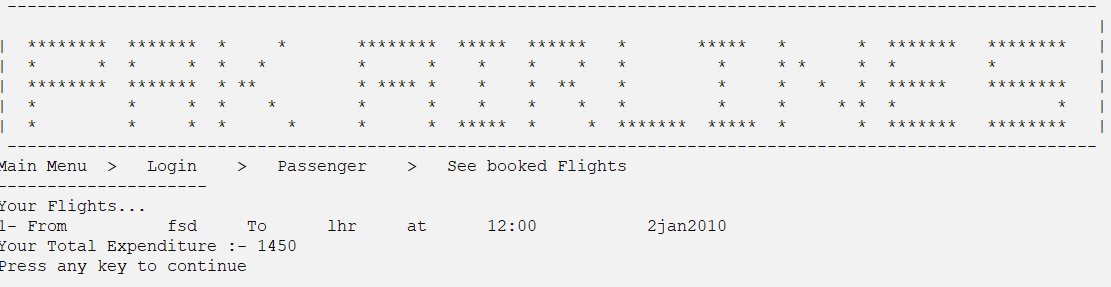
    {

        total[entry\_count] = total[entry\_count] + 400;

    }

    cout << "Your Total Expenditure :- " << total[entry\_count] << "$" << endl;

}



// See booked Flights

void bookedFlightsP()

{

    int i = 0;

    cout << "Main Menu  >   Login    >   Passenger    >   See booked Flights" << endl;

    cout << "---------------------" << endl;

    cout << "Your Flights..." << endl;

    for (int j = entry\_count; j < MAX\_RECORDS; j++)

    {

        if (rolesA[j] == "admin")

        {

            i--;

        }

        else if (usernameA[j] == username)

        {

            i = j + i;

            // cout << j << endl

            //      << i << endl;

            if (optionF[i] == 1)

            {

                cout << "1- From \t " << pDepartCity[i] << "\t To \t " << pArrCity[i] << " \t at \t 12:00 \t \t " << pDepartDate[i] << endl;

                cout << "Your Total Expenditure :- " << total[i] << endl;

            }

 else if (optionF[i] == 0)

            {

                cout << "No booked flights yet!!" << endl;

            }

        }

    }

}



// Cancel Flights

void cancelFlights()

{

    int cancel;

    cout << "Main Menu  >   Login    >   Passenger    >   Cancel A flight" << endl;

    cout << "---------------------" << endl;

    cout << "Your Flights..." << endl

         << endl;

    for (int i = 0; i < MAX\_RECORDS; i++)

    {

        if (usernameA[i] == username)

        {

            if (optionF[i] == 1)

            {

                cout << "1- From \t " << pDepartCity[i] << "\t To \t " << pArrCity[i] << " \t at \t 12:00 \t \t " << pDepartDate[i] << endl;

                cout << "Your Total Expenditure :- " << total[i] << endl;

                cout << "Your Option..";

                if (cancel == 1)

                {

                    cout << "Flight has been cancelled!!";

                    optionF[entry\_count] = 0;

                }

            }

            else if (optionF[i] == 0)

            {

                cout << "No booked flights yet!!" << endl

                     << endl;

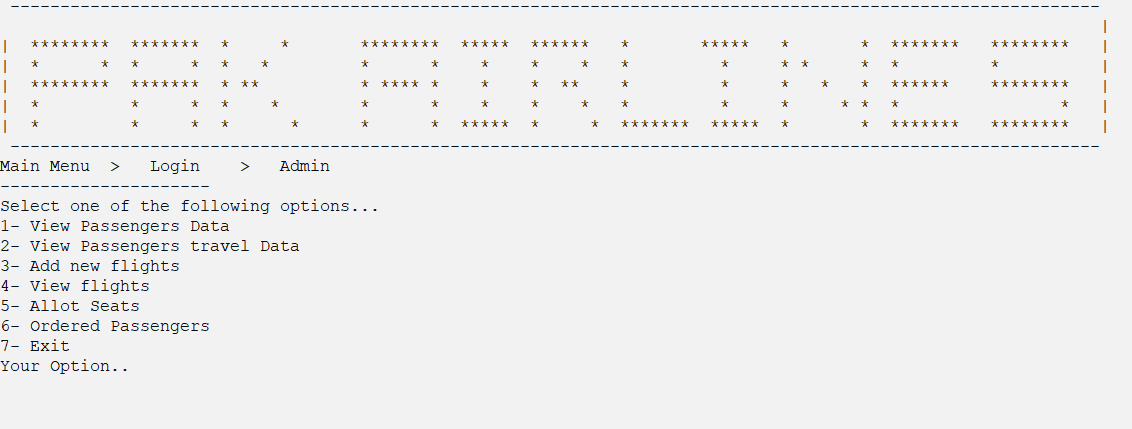
            }

        }

    }

}

// ADMIN



// MENU OF ADMIN

int adminMenu()

{

    int optionA;

    cout << "Main Menu  >   Login    >   Admin " << endl;

    cout << "---------------------" << endl;

    cout << "Select one of the following options..." << endl;

    cout << "1- View Passengers Data" << endl;

    cout << "2- View Passengers travel Data" << endl;

    cout << "3- Add new flights " << endl;

    cout << "4- View flights " << endl;

    cout << "5- Allot Seats  " << endl;

    cout << "6- Ordered Passengers  " << endl;

    cout << "7- Exit" << endl;

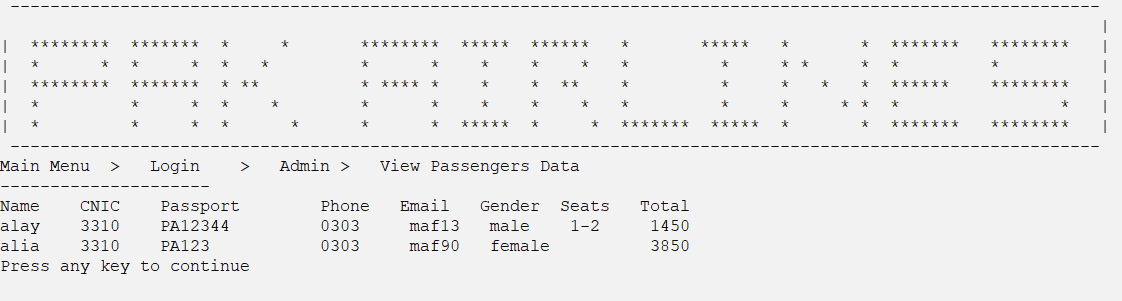
    cout << "Your Option..";

    cin >> optionA;

    return optionA;

}

// VIEW PASSENGERS DATA



void viewPData()

{

    cout << "Main Menu  >   Login    >   Admin >   View Passengers Data" << endl;

    cout << "---------------------" << endl;

    cout << "Name"

         << "\t"

         << "CNIC"

         << "\t"

         << "Passport"

         << "\t"

         << "Phone"

         << "\t"

         << "Email"

 << "\t"

         << "Gender"

         << "\t"

         << "Seats"

         << "\t"

         << "Total" << endl;

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

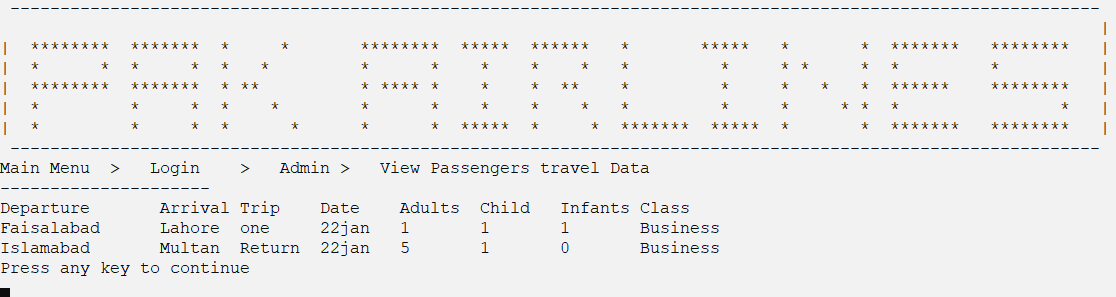
    {

        cout << pName[i] << "\t" << cnic[i] << "\t" << passport[i] << "\t\t" << pNum[i] << "\t " << mail[i] << "\t " << gender[i] << "\t " << seating[i] << "\t " << total[i] << endl;

    }

}

// VIEW PASSENGERS TRAVEL DATA



void viewPTravelData()

{

    cout << "Main Menu  >   Login    >   Admin >   View Passengers travel Data" << endl;

    cout << "---------------------" << endl;

    cout << "Departure"

         << "\t"

         << "Arrival"

         << "\t"

         << "Trip"

         << "\t"

         << "Date"

         << "\t"

         << "Adults"

         << "\t"

         << "Child"

         << "\t"

         << "Infants"

         << "\t"

         << "Class " << endl;

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

    {

        cout << pDepartCity[i] << "\t" << pArrCity[i] << "\t" << pTrip[i] << "\t" << pDepartDate[i] << "\t" << adult[i] << "\t" << child[i] << "\t" << infant[i] << "\t" << pClass[i] << endl;

    }

}

// ADD FLIGHTS



void addFlights()

{

    cout << flight\_count << endl;

    cout << "Main Menu  >   Login    >   Admin    >   Add new flights" << endl;

    cout << "---------------------" << endl;

    cout << "Departure from :- " << endl;

    cin >> aDepartCity[flight\_count];

    cout << "Arrival To :- " << endl;

    cin >> aArrCity[flight\_count];

    cout << "Trip Type :- " << endl;

    cin >> aTrip[flight\_count];

    cout << "Depart Date :-" << endl;

    cin >> aDepartDate[flight\_count];

    cout << "Depart Time :- " << endl;

    cin >> aDepartTime[flight\_count];

    flight\_count++;

}



void viewFlightsA()

{

    cout << flight\_count << endl;

    cout << "Main Menu  >   Login    >   Admin >   View Flights" << endl;

    cout << "---------------------" << endl;

    cout << "Departure"

         << "\t"

         << "Arrival"

         << "\t"

         << "Trip"

         << "\t"

         << "DepartDate"

         << "\t"

         << "DepartTime"

         << endl;

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

    {

        cout << aDepartCity[i] << "\t" << aArrCity[i] << "\t" << aTrip[i] << "\t" << aDepartDate[i] << "\t" << aDepartTime[i] << endl;

    }

}

// ALLOT SEATS

void seats()

{

    int total = 0;

    string seat;

    cout << "Main Menu  >   Login    >   Admin    >   Allot Seats" << endl;

    cout << "---------------------" << endl;

    cout << "Departure"

         << "\t"

         << "Arrival"

         << "\t"

         << "Trip"

         << "\t"

         << "Date"

         << "\t"

         << "Adults"

         << "\t"

         << "Child"

         << "\t"

         << "Infants"

         << "\t"

         << "Class" << endl;

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

    {

        cout << pDepartCity[i] << "\t" << pArrCity[i] << "\t" << pTrip[i] << "\t" << pDepartDate[i] << "\t" << adult[i] << "\t" << child[i] << "\t" << infant[i] << "\t" << pClass[i] << endl;

        cout << "No of Passengers (Adult) :- " << adult[i] << endl;

        cout << "No of Passengers (Child) :- " << child[i] << endl;

        cout << "No of Passengers (Infant) :- " << infant[i] << endl

             << endl;

        total = (adult[i]) + (child[i]);

        cout << "Enter Seat Range :- ";

        cin >> seating[i];

    }

}

void clearScreen()

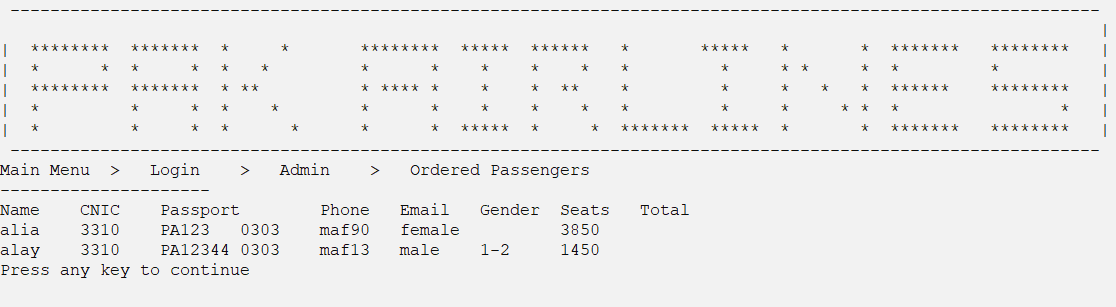
{

    cout << "Press any key to continue " << endl;

    getch();

    system("cls");

}

****

// SORTED PASSENGERS

void sorting()

{

    cout << "Main Menu  >   Login    >   Admin    >   Ordered Passengers" << endl;

    cout << "---------------------" << endl;

    cout << "Name"

         << "\t"

         << "CNIC"

         << "\t"

         << "Passport"

         << "\t"

         << "Phone"

         << "\t"

         << "Email"

         << "\t"

         << "Gender"

         << "\t"

         << "Seats"

         << "\t"

         << "Total" << endl;

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

    {

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

        {

            if (total[j] > total[i])

            { // sort total

                int temp = total[i];

                total[i] = total[j];

                total[j] = temp;

                // sort pName

                string namex = pName[i];

                pName[i] = pName[j];

                pName[j] = namex;

                // sort cnic array

                string name1 = cnic[i];

                cnic[i] = cnic[j];

                cnic[j] = name1;

                // sort passport

                string name2 = passport[i];

                passport[i] = passport[j];

                passport[j] = name2;

                // sort pNum

                string name3 = pNum[i];

                pNum[i] = pNum[j];

                pNum[j] = name3;

                // sort mail

                string name4 = mail[i];

                mail[i] = mail[j];

                mail[j] = name4;

                // sort gender

                string name5 = gender[i];

                gender[i] = gender[j];

                gender[j] = name5;

                // sort gender

                string name6 = seating[i];

                seating[i] = seating[j];

                seating[j] = name6;

            }

        }

    }

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

    {

        cout << pName[i] << "\t" << cnic[i] << "\t" << passport[i] << "\t" << pNum[i] << "\t" << mail[i] << "\t" << gender[i] << "\t" << seating[i] << "\t" << total[i] << endl;

    }

}

string parserecord(string line, int field)

{

    int commacount = 0;

    string item;

    for (int i = 0; i < line.length(); i++)

    {

        if (line[i] == ',')

        {

            commacount++;

        }

        if (commacount == field)

        {

            if (line[i] != ',')

            {

                item = item + line[i];

            }

        }

    }

    return item;

}

int convert(string n) // to convert a string into integer if its interger

{

    if (n == "0")

    {

        return 0;

    }

   float num = 0;

    int x;

    for (int i = 0; i < n.length(); i++)

    {

        x = (n[n.length() - i - 1]) - 48;

        num = num + x \* pow(10, i);

    }

    return num;

}

void loadData()

{

    string word1;

    fstream credFile;

    credFile.open("cred.txt", ios::in);

    int j = 0;

    while (!credFile.eof())

    {

        getline(credFile, word1);

        usernameA[j] = parserecord(word1, 0);

        passwordsA[j] = parserecord(word1, 1);

        rolesA[j] = parserecord(word1, 2);

        if (rolesA[j] == "user")

        {

            entry\_count++;

        }

        j++;

    }

    credFile.close();

    string temp, temp1;

    string word;

    fstream file;

    file.open("admin.txt", ios::in);

    int i = 0;

    while (!file.eof())

    {

        getline(file, word);

        pName[i] = parserecord(word, 0);

        cnic[i] = parserecord(word, 1);

        passport[i] = parserecord(word, 2);

        pNum[i] = parserecord(word, 3);

        mail[i] = parserecord(word, 4);

        gender[i] = parserecord(word, 5);

        seating[i] = parserecord(word, 6);

        temp = parserecord(word, 7);

        total[i] = convert(temp);

        pDepartCity[i] = parserecord(word, 8);

        pArrCity[i] = parserecord(word, 9);

        pTrip[i] = parserecord(word, 10);

        pDepartDate[i] = parserecord(word, 11);

        temp = parserecord(word, 12);

        adult[i] = convert(temp);

        temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

    file.close();

    int z = 0;

    fstream fFile;

    string word2;

    fFile.open("flight.txt", ios::in);

    while (!fFile.eof())

    {

   getline(fFile, word2);

        aDepartCity[z] = parserecord(word2, 0);

        aArrCity[z] = parserecord(word2, 1);

        aTrip[z] = parserecord(word2, 2);

        aDepartDate[z] = parserecord(word2, 3);

        aDepartTime[z] = parserecord(word2, 4);

        z++;

        flight\_count++;

    }

    fFile.close();

}

void storeData()

{

    // ADMIN STORE

    fstream adminFile;

    adminFile.open("admin.txt", ios::out);

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

    {

        adminFile << pName[i];

        adminFile << ",";

        adminFile << cnic[i];

        adminFile << ",";

        adminFile << passport[i];

        adminFile << ",";

        adminFile << pNum[i];

        adminFile << ",";

        adminFile << mail[i];

        adminFile << ",";

        adminFile << gender[i];

        adminFile << ",";

        adminFile << seating[i];

        adminFile << ",";

        adminFile << total[i];

        adminFile << ",";

        adminFile << pDepartCity[i];

        adminFile << ",";

        adminFile << pArrCity[i];

        adminFile << ",";

        adminFile << pTrip[i];

        adminFile << ",";

        adminFile << pDepartDate[i];

        adminFile << ",";

        adminFile << adult[i];

        adminFile << ",";

        adminFile << child[i];

        adminFile << ",";

        adminFile << infant[i];

        adminFile << ",";

        adminFile << pClass[i];

        adminFile << ",";

        adminFile << optionF[i];

        adminFile << endl;

    }

    adminFile.close();

}

void storeCred()

{

    // ADMIN STORE

    fstream credFile;

    credFile.open("cred.txt", ios::out);

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

    {

        credFile << usernameA[i];

        credFile << ",";

        credFile << passwordsA[i];

        credFile << ",";

        credFile << rolesA[i];

        credFile << endl;

    }

    credFile.close();

}

void storeFlight()

{

    // ADMIN STORE

    fstream fFile;

    fFile.open("flight.txt", ios::out);

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

    {

        fFile << aDepartCity[i];

        fFile << ",";

        fFile << aArrCity[i];

        fFile << ",";

        fFile << aTrip[i];

        fFile << ",";

        fFile << aDepartDate[i];

        fFile << ",";

        fFile << aDepartTime[i];

        fFile << endl;

    }

    fFile.close();

}

         << "\t"

         << "Gender"

         << "\t"

         << "Seats"

         << "\t"

         << "Total" << endl;

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

    {

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

        {

            if (total[j] > total[i])

            { // sort total

                int temp = total[i];

                total[i] = total[j];

                total[j] = temp;

                // sort pName

                string namex = pName[i];

                pName[i] = pName[j];

                pName[j] = namex;

                // sort cnic array

                string name1 = cnic[i];

                cnic[i] = cnic[j];

                cnic[j] = name1;

                // sort passport

                string name2 = passport[i];

                passport[i] = passport[j];

                passport[j] = name2;

                // sort pNum

                string name3 = pNum[i];

                pNum[i] = pNum[j];

                pNum[j] = name3;

                // sort mail

                string name4 = mail[i];

                mail[i] = mail[j];

                mail[j] = name4;

                // sort gender

                string name5 = gender[i];

                gender[i] = gender[j];

                gender[j] = name5;

                // sort gender

                string name6 = seating[i];

                seating[i] = seating[j];

                seating[j] = name6;

            }

 }

        }

    }

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

    {

        cout << pName[i] << "\t" << cnic[i] << "\t" << passport[i] << "\t" << pNum[i] << "\t" << mail[i] << "\t" << gender[i] << "\t" << seating[i] << "\t" << total[i] << endl;

    }

}

string parserecord(string line, int field)

{

    int commacount = 0;

    string item;

    for (int i = 0; i < line.length(); i++)

    {

        if (line[i] == ',')

        {

            commacount++;

        }        if (commacount == field)

        {

            if (line[i] != ',')

            {

                item = item + line[i];

            }

        }

    }

    return item;

}

int convert(string n) // to convert a string into integer if its interger

{

    if (n == "0")

    {

        return 0;

    }

    float num = 0;

    int x;

    for (int i = 0; i < n.length(); i++)

    {

        x = (n[n.length() - i - 1]) - 48;

        num = num + x \* pow(10, i);

    }

    return num;

}

void loadData()

{

    string word1;

    fstream credFile;

    credFile.open("cred.txt", ios::in);

    int j = 0;

    while (!credFile.eof())

    {

        getline(credFile, word1);

        usernameA[j] = parserecord(word1, 0);

        passwordsA[j] = parserecord(word1, 1);

        rolesA[j] = parserecord(word1, 2);

        if (rolesA[j] == "user")

        {

            entry\_count++;

        }

        j++;

    }

    credFile.close();

    string temp, temp1;

    string word;

    fstream file;

    file.open("admin.txt", ios::in);

    int i = 0;

    while (!file.eof())

    {

        getline(file, word);

        pName[i] = parserecord(word, 0);

        cnic[i] = parserecord(word, 1);

        passport[i] = parserecord(word, 2);

        pNum[i] = parserecord(word, 3);

        mail[i] = parserecord(word, 4);

        gender[i] = parserecord(word, 5);

        seating[i] = parserecord(word, 6);

        temp = parserecord(word, 7);

        total[i] = convert(temp);

        pDepartCity[i] = parserecord(word, 8);

        pArrCity[i] = parserecord(word, 9);

        pTrip[i] = parserecord(word, 10);

        pDepartDate[i] = parserecord(word, 11);

        temp = parserecord(word, 12);

        adult[i] = convert(temp);

        temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

void loadData()

{

    string word1;

    fstream credFile;

    credFile.open("cred.txt", ios::in);

    int j = 0;

    while (!credFile.eof())

    {

        getline(credFile, word1);

        usernameA[j] = parserecord(word1, 0);

        passwordsA[j] = parserecord(word1, 1);

        rolesA[j] = parserecord(word1, 2);

        if (rolesA[j] == "user")

        {

            entry\_count++;

        }

        j++;

    }

    credFile.close();

    string temp, temp1;

    string word;

    fstream file;

    file.open("admin.txt", ios::in);

    int i = 0;

    while (!file.eof())

    {

        getline(file, word);

        pName[i] = parserecord(word, 0);

        cnic[i] = parserecord(word, 1);

        passport[i] = parserecord(word, 2);

        pNum[i] = parserecord(word, 3);

        mail[i] = parserecord(word, 4);

        gender[i] = parserecord(word, 5);

        seating[i] = parserecord(word, 6);

        temp = parserecord(word, 7);

        total[i] = convert(temp);

        pDepartCity[i] = parserecord(word, 8);

        pArrCity[i] = parserecord(word, 9);

        pTrip[i] = parserecord(word, 10);

        pDepartDate[i] = parserecord(word, 11);

        temp = parserecord(word, 12);

        adult[i] = convert(temp);

        temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

    file.close();

    int z = 0;

    fstream fFile;

    string word2;

    fFile.open("flight.txt", ios::in);

    while (!fFile.eof())

    {

        getline(fFile, word2);

        aDepartCity[z] = parserecord(word2, 0);

        aArrCity[z] = parserecord(word2, 1);

        aTrip[z] = parserecord(word2, 2);

        aDepartDate[z] = parserecord(word2, 3);

        aDepartTime[z] = parserecord(word2, 4);

        z++;

        flight\_count++;

    }

    fFile.close();

}

   temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

    file.close();

    int z = 0;

    fstream fFile;

    string word2;

    fFile.open("flight.txt", ios::in);

    while (!fFile.eof())

    {

        getline(fFile, word2);

        aDepartCity[z] = parserecord(word2, 0);

        aArrCity[z] = parserecord(word2, 1);

        aTrip[z] = parserecord(word2, 2);

        aDepartDate[z] = parserecord(word2, 3);

        aDepartTime[z] = parserecord(word2, 4);

        z++;

        flight\_count++;

    }

    fFile.close();

}

void storeData()

{

    // ADMIN STORE

    fstream adminFile;

    adminFile.open("admin.txt", ios::out);

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

    {

        adminFile << pName[i];

        adminFile << ",";

        adminFile << cnic[i];

        adminFile << ",";

        adminFile << passport[i];

        adminFile << ",";

        adminFile << pNum[i];

        adminFile << ",";

        adminFile << mail[i];

        adminFile << ",";

        adminFile << gender[i];

        adminFile << ",";

        adminFile << seating[i];

        adminFile << ",";

        adminFile << total[i];

        adminFile << ",";

        adminFile << pDepartCity[i];

        adminFile << ",";

        adminFile << pArrCity[i];

        adminFile << ",";

        adminFile << pTrip[i];

        adminFile << ",";

        adminFile << pDepartDate[i];

        adminFile << ",";

        adminFile << adult[i];

        adminFile << ",";

        adminFile << child[i];

        adminFile << ",";

        adminFile << infant[i];

        adminFile << ",";

        adminFile << pClass[i];

        adminFile << ",";

        adminFile << optionF[i];

        adminFile << endl;

    }

    adminFile.close();

}

void storeCred()

{

    // ADMIN STORE

    fstream credFile;

    credFile.open("cred.txt", ios::out);

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

    {

        credFile << usernameA[i];

        credFile << ",";

        credFile << passwordsA[i];

        credFile << ",";

        credFile << rolesA[i];

        credFile << endl;

    }

    credFile.close();

}

void storeFlight()

{

    // ADMIN STORE

    fstream fFile;

    fFile.open("flight.txt", ios::out);

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

    {

        temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

    file.close();

    int z = 0;

    fstream fFile;

    string word2;

    fFile.open("flight.txt", ios::in);

    while (!fFile.eof())

    {

        getline(fFile, word2);

        aDepartCity[z] = parserecord(word2, 0);

        aArrCity[z] = parserecord(word2, 1);

        aTrip[z] = parserecord(word2, 2);

        aDepartDate[z] = parserecord(word2, 3);

        aDepartTime[z] = parserecord(word2, 4);

        z++;

        flight\_count++;

    }

    fFile.close();

}

       adminFile << ",";

        adminFile << gender[i];

        adminFile << ",";

        adminFile << seating[i];

        adminFile << ",";

        adminFile << total[i];

        adminFile << ",";

        adminFile << pDepartCity[i];

        adminFile << ",";

        adminFile << pArrCity[i];

        adminFile << ",";

        adminFile << pTrip[i];

        adminFile << ",";

        adminFile << pDepartDate[i];

        adminFile << ",";

        adminFile << adult[i];

        adminFile << ",";

        adminFile << child[i];

        adminFile << ",";

        adminFile << infant[i];

        adminFile << ",";

        adminFile << pClass[i];

        adminFile << ",";

        adminFile << optionF[i];

        adminFile << endl;

    }

    adminFile.close();

}

void storeCred()

{

    // ADMIN STORE

    fstream credFile;

    credFile.open("cred.txt", ios::out);

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

    {

        credFile << usernameA[i];

        credFile << ",";

        credFile << passwordsA[i];

        credFile << ",";

        credFile << rolesA[i];

        credFile << endl;

    }

    credFile.close();

}

void storeFlight()

{

    // ADMIN STORE

    fstream fFile;

    fFile.open("flight.txt", ios::out);

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

    {

        fFile << aDepartCity[i];

        fFile << ",";

        fFile << aArrCity[i];

        fFile << ",";

        fFile << aTrip[i];

        fFile << ",";

        fFile << aDepartDate[i];

        fFile << ",";

        fFile << aDepartTime[i];

        fFile << endl;

    }

    fFile.close();

}

        temp = parserecord(word, 13);

        child[i] = convert(temp);

        temp = parserecord(word, 14);

        infant[i] = convert(temp);

        pClass[i] = parserecord(word, 15);

        temp = parserecord(word, 16);

        optionF[i] = convert(temp);

        i++;

    }

    file.close();

    int z = 0;

    fstream fFile;

    string word2;

    fFile.open("flight.txt", ios::in);

    while (!fFile.eof())

    {

        getline(fFile, word2);

        aDepartCity[z] = parserecord(word2, 0);

        aArrCity[z] = parserecord(word2, 1);

        aTrip[z] = parserecord(word2, 2);

        aDepartDate[z] = parserecord(word2, 3);

        aDepartTime[z] = parserecord(word2, 4);

        z++;

        flight\_count++;

    }

    fFile.close();

}

void storeFlight()

{

    // ADMIN STORE

    fstream fFile;

    fFile.open("flight.txt", ios::out);

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

    {

        fFile << aDepartCity[i];

        fFile << ",";

        fFile << aArrCity[i];

        fFile << ",";

        fFile << aTrip[i];

        fFile << ",";

        fFile << aDepartDate[i];

        fFile << ",";

        fFile << aDepartTime[i];

        fFile << endl;

    }

    fFile.close();

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | A | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | B | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | A | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | A | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | A | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | A | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | A | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | A | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | A | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | C | Not implemented |
| Presentation and Demo  **Grade:** | A | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | A | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |