**Question 1 Code:**

#include<iostream>

#include<string>

using namespace std;

int const MAX\_SkillS = 5;

int const MAX\_SPORTS = 5;

class Skill

{

    private:

        int skillID;

        string skillName;

        string description;

    public:

        Skill() : skillID(0), skillName(""), description("") {}

        Skill(int sk\_ID, string sk\_name, string sk\_desc)

        {

            skillID = sk\_ID;

            skillName = sk\_name;

            description = sk\_desc;

        }

        void showSkillDetails()

        {

            cout<<"\nSkill Details are as follows....\n";

            cout<<"Skill Name: "<< skillName <<"\n";

            cout<<"Description: "<<description<<"\n";

        }

        void updateSkillDescription(string newDescription)

        {

            description = newDescription;

            cout<<"\nNew Description of the Skill Uploaded";

        }

        int get\_skillID()

        {

            return skillID;

        }

        string get\_skillName()

        {

            return skillName;

        }

        string get\_description()

        {

            return description;

        }

};

class Sport

{

    private:

        int sportID;

        string name;

        string description;

        Skill requiredSkills[MAX\_SkillS];

        int numOfSkill = 0;

    public:

        Sport() : sportID(0), name(""), description(""), numOfSkill(0) {}

        Sport(int id, string name, string sp\_desc)

        {

            sportID = id;

            this->name = name;

            description = sp\_desc;

        }

        void addSkill(Skill s)

        {

            if(numOfSkill < MAX\_SkillS)

            {

                requiredSkills[numOfSkill] = s;

                numOfSkill++;

            }

            else

            {

                cout<< "Cannot Add more skill as Maximum reached....";

            }

        }

        void removeSkill(Skill s)

        {

            int found = 0;

            for(int i=0; i<numOfSkill; i++)

            {

                if(requiredSkills[i].get\_skillID() == s.get\_skillID())

                {

                    found = 1;

                    for(int j = i; j<numOfSkill-1; j++)

                    {

                        requiredSkills[j] = requiredSkills[j+1];

                    }

                    numOfSkill--;

                    break;

                }

            }

            if(!found)

            {

                cout << "Skill with ID " << s.get\_skillID() << " not found in " << name << ".\n";

            }

        }

        int get\_sportID()

        {

            return sportID;

        }

        string get\_sportname()

        {

            return name;

        }

};

class Student; // Forward declaration

class Mentor {

private:

    int mentor\_ID;

    string name;

    int maxLearners;

    Sport sportsExpertise[MAX\_SPORTS];

    Student\* assignedLearners[MAX\_SPORTS]; // Changed to MAX\_SPORTS for consistency

    int numOfLearner;

public:

    Mentor(int id, string m\_name, int m\_Maxlearners) {

        mentor\_ID = id;

        name = m\_name;

        maxLearners = m\_Maxlearners;

        numOfLearner = 0; // Initialize number of learners

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

            assignedLearners[i] = nullptr; // Initialize pointers to nullptr

        }

    }

    bool assignLearner(Student\* s);

    void removeLearner(Student\* s);

    void viewLearners();

    void addSportExpertise(Sport s) {

        if (numOfLearner < MAX\_SPORTS) {

            sportsExpertise[numOfLearner] = s;

            numOfLearner++;

        } else {

            cout << "Cannot add more sports expertise as maximum reached....";

        }

    }

    void provideGuidance()

    {

        cout<<"Mentor: "<< name<< " is providing Guidance";

    }

    int get\_mentorID() {

        return mentor\_ID;

    }

    string get\_mentorName() {

        return name;

    }

};

class Student {

private:

    int student\_ID;

    string name;

    int age;

    Sport sportInterest[MAX\_SPORTS];

    Mentor\* mentorAssigned;

    int numOfSportInterestedIN;

public:

    Student() : student\_ID(0), name(""), age(0), mentorAssigned(nullptr), numOfSportInterestedIN(0) {}

    Student(int s\_ID, string s\_name, int s\_age) {

        student\_ID = s\_ID;

        name = s\_name;

        age = s\_age;

        mentorAssigned = nullptr; // Initialize mentorAssigned

        numOfSportInterestedIN = 0; // Initialize number of sports interested

    }

    void registerForMentorship(Mentor\* m) {

        if (m->assignLearner(this)) {

            mentorAssigned = m;

            cout << "\nStudent: " << name << " has been assigned to instructor " << m->get\_mentorName();

        } else {

            cout << "\nStudent cannot be assigned to Instructor " << m->get\_mentorName();

        }

    }

    void updateSportsInterest(Sport s) {

        if (numOfSportInterestedIN < MAX\_SPORTS) {

            sportInterest[numOfSportInterestedIN] = s;

            numOfSportInterestedIN++;

            cout << "\nSport interest updated for student: " << name;

        } else {

            cout << "Cannot add more sports interest as maximum reached....";

        }

    }

     void viewMentorDetails() {

        if (mentorAssigned != nullptr) {

            cout << "\nMentor Details:\n";

            cout << "Mentor ID: " << mentorAssigned->get\_mentorID() << "\n";

            cout << "Mentor Name: " << mentorAssigned->get\_mentorName() << "\n";

        } else {

            cout << "No mentor assigned to student: " << name << "\n";

        }

    }

    int get\_stuID() {

        return student\_ID;

    }

    string get\_stu\_name() {

        return name;

    }

};

bool Mentor::assignLearner(Student\* s) {

    if (numOfLearner < maxLearners) {

        assignedLearners[numOfLearner] = s;

        numOfLearner++;

        return true;

    } else {

        cout << "Cannot assign learners as maximum learner reached";

        return false;

    }

}

void Mentor::removeLearner(Student\* s) {

    int found = 0;

    for (int i = 0; i < numOfLearner; i++) {

        if (assignedLearners[i] != nullptr && assignedLearners[i]->get\_stuID() == s->get\_stuID()) {

            found = 1;

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

                assignedLearners[j] = assignedLearners[j + 1];

            }

            assignedLearners[numOfLearner - 1] = nullptr; // Set last pointer to nullptr

            numOfLearner--;

            break;

        }

    }

    if (!found) {

        cout << "Student not found among " << name << "'s assigned learners.\n";

    }

}

void Mentor::viewLearners() {

    cout << "\n-----------Learner Assigned Till now-------------\n";

    if (numOfLearner == 0) {

        cout << "No learner assigned yet.....\n"; // Corrected message

    } else {

        for (int i = 0; i < numOfLearner; i++) {

            if (assignedLearners[i] != nullptr) { // Check for nullptr

                cout << "\n------------------------\n";

                cout << "Student ID: " << assignedLearners[i]->get\_stuID()

                     << ", Name: " << assignedLearners[i]->get\_stu\_name() << "\n";

                cout << "\n------------------------\n";

            }

        }

    }

}

int main()

{

    Skill skill1(1, "Serve", "Ability to serve effectively in tennis.");

    Skill skill2(2, "Forehand", "Strong forehand shot in tennis.");

    cout<<"skills added";

    Sport tennis(101, "Tennis", "A sport played with rackets and balls.");

    tennis.addSkill(skill1);

    tennis.addSkill(skill2);

    cout<<"sport added";

    Mentor mentorAli(201, "Ali", 3);

    mentorAli.addSportExpertise(tennis);

    cout<<"mentor added";

    Student studentSaad(301, "Saad", 20);

    studentSaad.updateSportsInterest(tennis);

    studentSaad.registerForMentorship(&mentorAli);

    mentorAli.viewLearners();

    studentSaad.viewMentorDetails();

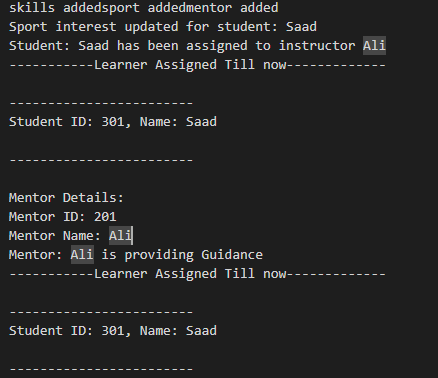
    mentorAli.provideGuidance();

    mentorAli.viewLearners();

    return 0;

}

**Question 1 Output:**



**Question 2 Code:**

#include<iostream>

#include<string.h>

using namespace std;

class Robot

{

    private:

        string name;

        int hits;

    public:

        Robot(): name(""), hits(0){};

        Robot(string \_name, int \_hits = 0)

        {

            name = \_name;

            hits = \_hits;

        }

        void hitBall(int &ballX, int &ballY, const string &direction)

        {

            if (direction == "up")

                ballY++;

            else if (direction == "down")

                ballY--;

            else if (direction == "left")

                ballX--;

            else if (direction == "right")

                ballX++;

            hits++;

        }

        int getHits()

        {

            return hits;

        }

        string getName()

        {

            return name;

        }

};

class Ball

{

    private:

        int X;

        int Y;

    public:

        Ball(): X(0), Y(0){};

        void move(int dx, int dy)

        {

            X += dx;

            Y += dy;

        }

        tuple<int, int> getPosition()

        {

            return make\_tuple(X, Y);

        }

        int getX()

        {

            return X;

        }

        void setX(int x)

        {

            X = x;

        }

        void setY(int y)

        {

            Y = y;

        }

        int getY()

        {

            return Y;

        }

};

class Goal

{

    private:

        int X;

        int Y;

    public:

        Goal(): X(3), Y(3){};

        bool isGoalReached(int ballX, int ballY)

        {

            if(ballX == X && ballY == Y)

            {

                return true;

            }

            else

            {

                return false;

            }

        }

};

class Team

{

    private:

        Robot\* r;

        string TeamName;

    public:

        Team(string \_name, Robot \*robot)

        {

            TeamName = \_name;

            r = robot;

        }

        string getTeamName()

        {

            return TeamName;

        }

        Robot\* getRobot()

        {

            return r;

        }

};

class Game

{

    private:

        Team\* t1;

        Team\* t2;

        Ball ball;

        Goal goal;

    public:

        Game(Team\* team1, Team\* team2): t1(team1), t2(team2){};

        void startGame()

        {

            play(t1);

            ball.setX(0);

            ball.setY(0);

            play(t2);

            declareWinner();

        }

        void play(Team \*team)

        {

            Robot\* r = team->getRobot();

            int ballX = ball.getX();

            int ballY = ball.getY();

            string direction = "";

            while(!goal.isGoalReached(ballX, ballY))

            {

                cout << team->getTeamName() << "'s turn. Current ball position: (" << ballX << ", " << ballY << ")\n";

                cout << "Enter direction to hit the ball (up, down, left, right): ";

                cin >> direction;

                r->hitBall(ballX, ballY, direction);

                ball.move(ballX - ball.getX(), ballY - ball.getY());

                ball.setX(ballX);

                ball.setY(ballY);

            }

            cout << team->getTeamName() << " reached the goal with " << r->getHits() << " hits!\n";

        }

        void declareWinner()

        {

            int teamOneHits = t1->getRobot()->getHits();

            int teamTwoHits = t2->getRobot()->getHits();

            if (teamOneHits < teamTwoHits)

                cout << t1->getTeamName() << " wins!\n";

            else if (teamTwoHits < teamOneHits)

                cout << t2->getTeamName() << " wins!\n";

            else

                cout << "It's a tie!\n";

        }

};

int main()

{

    Robot robot1("Robot1", 0);

    Robot robot2("Robot2", 0);

    Team team1("Team A", &robot1);

    Team team2("Team B", &robot2);

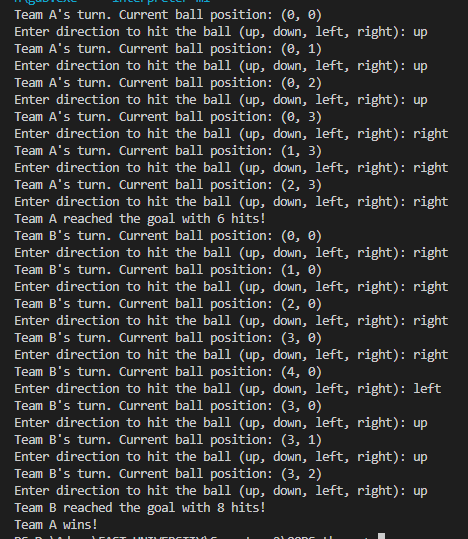
    Game game(&team1, &team2);

    game.startGame();

    return 0;

}

**Question 2 Output:**



**Question 3 Code:**

#include<iostream>

#include<string.h>

using namespace std;

class User

{

    private:

        string user\_Name;

        int age;

        string licence\_Type;

        int contact\_Info;

        int user\_ID;

    public:

        User(string \_name, int \_age, string \_license, int \_contact, int \_user\_ID)

        {

            user\_Name = \_name;

            age = \_age;

            licence\_Type = \_license;

            contact\_Info = \_contact;

            user\_ID = \_user\_ID;

        }

        void update\_user\_info(string \_name, int \_age, string \_license, int \_contact, int \_user\_ID)

        {

            user\_Name = \_name;

            age = \_age;

            licence\_Type = \_license;

            contact\_Info = \_contact;

            user\_ID = \_user\_ID;

        }

        string getName()

        {

            return user\_Name;

        }

        int getUserID()

        {

            return user\_ID;

        }

        string getLicence()

        {

            return licence\_Type;

        }

};

class Vehicle

{

    private:

        string model\_name;

        string eligibility;

        int rental\_Price\_Per\_Day;

    public:

        Vehicle(string \_model, string \_eligibility, int \_rental)

        {

            model\_name = \_model;

            eligibility = \_eligibility;

            rental\_Price\_Per\_Day = \_rental;

        }

        string getModelName()

        {

            return model\_name;

        }

        string getEligibility()

        {

            return eligibility;

        }

        int getRentalPrice()

        {

            return rental\_Price\_Per\_Day;

        }

};

class RentalSystem

{

    private:

        User\*\* users;

        Vehicle\*\* vehicles;

        int MAX\_USERS = 10;

        int MAX\_VEHICLES = 10;

        int user\_count;

        int vehicle\_count;

    public:

        RentalSystem()

        {

            users = new User\*[MAX\_USERS];

            vehicles = new Vehicle\*[MAX\_VEHICLES];

            user\_count = 0;

            vehicle\_count = 0;

        }

        void registerUser(string name, int age, string license, int contact, int user\_ID)

        {

            if(user\_count < MAX\_USERS)

            {

                users[user\_count] = new User(name, age, license, contact, user\_ID);

                cout<< "\n------------------------\n";

                cout<< "New User Created.....";

                cout<< "\n------------------------\n";

                user\_count++;

            }

            else

            {

                cout<< "\n------------------------\n";

                cout<< "MAX USER LIMIT REACHED.....";

                cout<< "\n------------------------\n";

            }

        }

        void updateUser(string new\_name, int new\_age, string new\_license, int new\_contact, int old\_user\_ID)

        {

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

            {

                if(users[i]->getUserID() == old\_user\_ID)

                {

                    users[i]->update\_user\_info(new\_name ,new\_age ,new\_license, new\_contact, old\_user\_ID);

                    cout<< "\n------------------------\n";

                    cout<< "User Profile Updated.....";

                    cout<< "\n------------------------\n";

                    return;

                }

            }

            cout<< "\n------------------------\n";

            cout<< "User with ID "<< old\_user\_ID<< " not FOUND.....";

            cout<< "\n------------------------\n";

        }

        void registerVehicle(string model, string eligibility, int rental)

        {

            if(vehicle\_count < MAX\_VEHICLES)

            {

                vehicles[vehicle\_count] = new Vehicle(model, eligibility, rental);

                cout<< "\n------------------------\n";

                cout<< "New Vehicle Added.....";

                cout<< "\n------------------------\n";

                vehicle\_count++;

            }

            else

            {

                cout<< "\n------------------------\n";

                cout<< "MAX VEHICLE LIMIT REACHED.....";

                cout<< "\n------------------------\n";

            }

        }

        void displayVehicles()

        {

            cout << "Available Vehicles:" << endl;

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

            {

                cout << "Model: " << vehicles[i]->getModelName()

                     << ", Rental Price: $" << vehicles[i]->getRentalPrice()

                     << ", Eligibility: " << vehicles[i]->getEligibility() << endl;

            }

        }

        void rentingVehicle(int user\_ID, string model\_name)

        {

            User\* user = nullptr;

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

            {

                if(users[i]->getUserID() == user\_ID)

                {

                    user = users[i];

                    break;

                }

            }

            if(user == nullptr)

            {

                cout << "User  not found!" << endl;

                return;

            }

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

            {

                if( (user->getLicence() == vehicles[i]->getEligibility()) || (user->getLicence() == "Intermediate" && vehicles[i]->getEligibility() != "Learner"))

                {

                    cout << "Rental successful! You have rented: " << vehicles[i]->getModelName()

                         << " for $" << vehicles[i]->getRentalPrice() << " per day." << endl;

                    return;

                }

                else

                {

                    cout << "You are not eligible to rent this vehicle." << endl;

                    return;

                }

            }

            cout << "Vehicle not found!" << endl;

        }

        ~RentalSystem()

        {

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

            {

                delete users[i];

            }

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

            {

                delete vehicles[i];

            }

            delete[] users;

            delete[] vehicles;

        }

};

int main()

{

    RentalSystem system;

    system.registerUser ("Alice", 25, "Full", 123456, 1);

    system.registerUser ("Bob", 20, "Learner", 987654, 2);

    system.registerVehicle("Toyota Corolla", "Full", 30.0);

    system.registerVehicle("Honda Civic", "Intermediate", 25.0);

    system.registerVehicle("Ford Fiesta", "Learner", 20.0);

    system.displayVehicles();

    system.rentingVehicle(1, "Toyota Corolla");

    system.rentingVehicle(2, "Honda Civic");

    system.rentingVehicle(2, "Ford Fiesta");

    system.updateUser ("Alice Smith", 26, "Intermediate", 123456, 1);

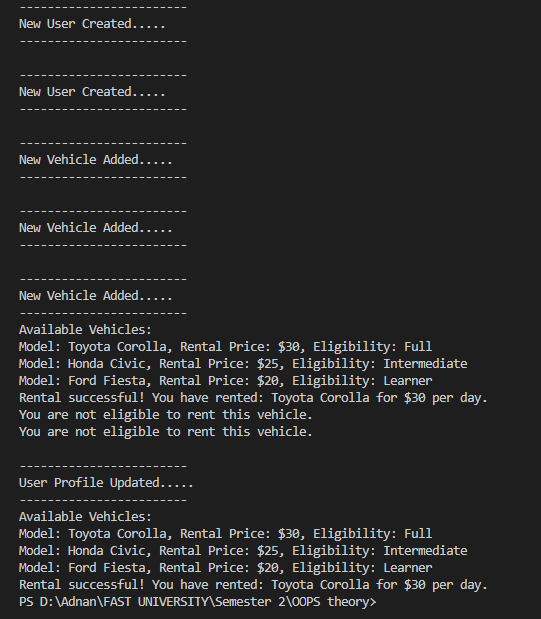
    system.displayVehicles();

    system.rentingVehicle(1, "Honda Civic");

    return 0;

}

**Question 3 Output:**



**Question 4 Code:**

#include<iostream>

#include<string.h>

using namespace std;

class Student

{

    private:

        int student\_ID;

        string student\_Name;

        float semester\_fee;

        bool is\_Active;

        bool is\_Registered;

    public:

        Student() : student\_ID(0), student\_Name(""), is\_Registered(false), semester\_fee(0.0), is\_Active(false) {};

        Student(int std\_id, string name)

        {

            student\_ID = std\_id;

            student\_Name = name;

            semester\_fee = 0;

            is\_Active = false;

            is\_Registered = false;

        }

        void paySemesterFee(int amount)

        {

            semester\_fee += amount;

            if(semester\_fee > 100)

            {

               activated() ;

            }

            cout<< "Fee paid " << amount << " Total Fee: " << semester\_fee;

        }

        void activated()

        {

            is\_Active = true;

            cout<< "\n------------------------\n";

            cout<< "Card Has been Activated for "<< student\_ID;

            cout<< "\n------------------------\n";

        }

        void Deactivated()

        {

            is\_Active = false;

            cout<< "\n------------------------\n";

            cout<< "Card Has been DeActivated for "<< student\_ID;

            cout<< "\n------------------------\n";

        }

        string getName() const

        {

            return student\_Name;

        }

        int getID() const

        {

            return student\_ID;

        }

        bool getIsActive() const

        {

            return is\_Active;

        }

};

class Bus

{

    private:

        int bus\_ID;

        int capacity;

        int current\_Capacity;

        string route;

    public:

        Bus(): bus\_ID(0), capacity(0), current\_Capacity(0), route(""){};

        Bus(int \_bus\_Id, string \_route,int \_capacity)

        {

            bus\_ID = \_bus\_Id;

            capacity = \_capacity;

            route = \_route;

        }

        void recordAttenence(int std\_id)

        {

            if(current\_Capacity < capacity)

            {

                cout<< "\n------------------------\n";

                cout<< "Attendence Marked for student " << std\_id;

                cout<< "\n------------------------\n";

                current\_Capacity++;

            }

            else

            {

                cout<< "\n------------------------\n";

                cout<< "MAX CAPACITY REACHED... ";

                cout<< "\n------------------------\n";

            }

        }

        void removeStudent(int std\_id)

        {

            cout<< "\n------------------------\n";

            cout<< "Student Removed From Bus " << std\_id;

            cout<< "\n------------------------\n";

            current\_Capacity--;

        }

        int getBusID() const

        {

            return bus\_ID;

        }

};

class Route

{

    private:

        int route\_Id;

        string start\_Location;

        string end\_Location;

        string stops[10];

        int stops\_Count;

    public:

        Route(): route\_Id(0), start\_Location(""), end\_Location(""), stops\_Count(0){};

        Route(int \_route\_ID, string \_start\_Location, string \_end\_location)

        {

            route\_Id = \_route\_ID;

            start\_Location = \_start\_Location;

            end\_Location = \_end\_location;

        }

        void addStops(string stop)

        {

            if(stops\_Count <= 10)

            {

                stops[stops\_Count] = stop;

                cout<< "\n------------------------\n";

                cout<< "New Stop Added To List: " << stop;

                cout<< "\n------------------------\n";

                stops\_Count++;

            }

            else

            {

                cout<< "\n------------------------\n";

                cout<< "MAX STOPS REACHED..... ";

                cout<< "\n------------------------\n";

            }

        }

        void removeStops(string stop)

        {

            for(int i=0; i<stops\_Count; i++)

            {

                if(stop == stops[i])

                {

                   cout<< "\n------------------------\n";

                    cout<< "STOP REMOVED: "<< stop;

                    cout<< "\n------------------------\n";

                    stops\_Count--;

                    return;

                }

            }

            cout<< "\n------------------------\n";

            cout<< "NO stop FOUND with name: "<< stop;

            cout<< "\n------------------------\n";

        }

        int getRouteID() const

        {

            return route\_Id;

        }

};

class TransportationSystem {

    private:

        static const int MAX\_STUDENTS = 100;

        static const int MAX\_BUSES = 10;

        static const int MAX\_ROUTES = 10;

        Student students[MAX\_STUDENTS];

        Bus buses[MAX\_BUSES];

        Route routes[MAX\_ROUTES];

        int studentCount;

        int busCount;

        int routeCount;

    public:

        TransportationSystem() : studentCount(0), busCount(0), routeCount(0) {}

        void registerStudent(int studentID, string name)

        {

            if (studentCount < MAX\_STUDENTS)

            {

                students[studentCount++] = Student(studentID, name);

                cout << "Student registered: " << name << endl;

            }

            else

            {

                cout << "Maximum student limit reached!" << endl;

            }

    }

    void payFee(int studentID, float amount)

    {

        for (int i = 0; i < studentCount; i++)

        {

            if (students[i].getID() == studentID)

            {

                students[i].paySemesterFee(amount);

                return;

            }

        }

        cout << "Student ID not found!" << endl;

    }

    void assignRoute(int studentID, int routeID)

    {

        cout << "Route ID: " << routeID << " assigned to student ID: " << studentID << endl;

    }

     void recordAttendance(int studentID, int busID) {

        for (int i = 0; i < busCount; i++) {

            if (buses[i].getBusID() == busID) {

                buses[i].recordAttenence(studentID);

                return;

            }

        }

        cout << "Bus ID not found!" << endl;

    }

    void addBus(int id, string route, int capacity) {

        if (busCount < MAX\_BUSES) {

            buses[busCount++] = Bus(id, route, capacity);

            cout << "Bus added with ID: " << id << endl;

        } else {

            cout << "Maximum bus limit reached!" << endl;

        }

    }

    void addRoute(int id, string start, string end) {

        if (routeCount < MAX\_ROUTES) {

            routes[routeCount++] = Route(id, start, end);

            cout << "Route added with ID: " << id << endl;

        } else {

            cout << "Maximum route limit reached!" << endl;

        }

    }

};

int main() {

    TransportationSystem system;

    system.registerStudent(1, "Alice");

    system.registerStudent(2, "Bob");

    system.payFee(1, 50.0);

    system.payFee(1, 60.0);

    system.addBus(101, "Route A", 30);

    system.addRoute(201, "Location A", "Location B");

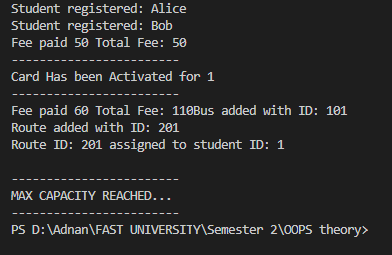
    system.assignRoute(1, 201);

    system.recordAttendance(1, 101);

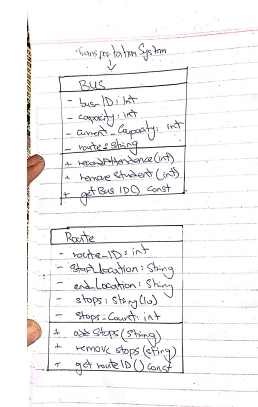
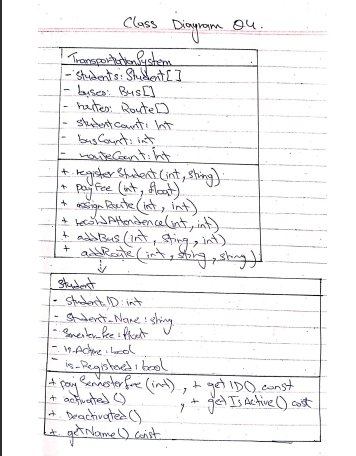
    return 0;

}

**Question 4 Output:**



**Question 4 Class Diagram:**

****