QUESTION NO.1:

SOURCE CODE:

#include <iostream>

using namespace std;

class Learner;

class Skill {

public:

    int skillID;

    string skillName;

    string description;

    void showSkillDetails() const {

        cout << "SKILL ID: " << skillID << endl;

        cout << "SKILL NAME: " << skillName << endl;

        cout << "SKILL DESCRIPTION: " << description << endl;

    }

    void updateSkillDescription(string newDescription) {

        description = newDescription;

    }

};

class Sports {

private:

    int sportID;

    Skill requiredSkill[20];

    int skillCount;

public:

    string name;

    string description;

    Sports() : skillCount(0) {}

    void display() const {

        cout << "SPORT ID: " << sportID << endl;

        cout << "NAME: " << name << endl;

        cout << "DESCRIPTION: " << description << endl;

        cout << "REQUIRED SKILLS: ";

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

            requiredSkill[i].showSkillDetails();

        }

        cout << endl;

    }

};

class Mentor {

private:

    int mentorID;

    int maxLearners;

    int assignedLearnersCount;

    Learner\* assignedLearners[100];

    int countExpertise;

public:

    string name;

    Sports sportExpertise[20];

    Mentor() : mentorID(0), maxLearners(0), assignedLearnersCount(0), countExpertise(0) {}

    Mentor(int ID, int maxlearners, string name) : mentorID(ID), maxLearners(maxlearners), name(name), assignedLearnersCount(0), countExpertise(0) {}

    int getStudentsAssigned() const {

        return assignedLearnersCount;

    }

    int getMaxLearners() const {

        return maxLearners;

    }

    void assignLearner(Learner\* s) {

        if (assignedLearnersCount < maxLearners) {

            assignedLearners[assignedLearnersCount++] = s;

        } else {

            cout << "MENTOR " << name << " HAS REACHED THE MAX CAPACITY." << endl;

        }

    }

    int getCountExpertise() const {

        return countExpertise;

    }

    void removeLearner(Learner\* s);

    void viewLearners();

    void provideGuidance();

};

class Learner {

private:

    int id;

    Sports sportOfInterest[20];

    int age;

    int sportOfInterestCount;

public:

    int getID() const {

        return id;

    }

    void display() const {

        cout << "ID: " << id << endl;

        cout << "NAME: " << name << endl;

        cout << "AGE: " << age << endl;

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

            sportOfInterest[i].display();

        }

    }

    Mentor\* mentorAssigned;

    string name;

    Learner() : id(0), age(0), sportOfInterestCount(0), mentorAssigned(nullptr) {}

    Learner(int id, int age, string name) : id(id), age(age), name(name), sportOfInterestCount(0), mentorAssigned(nullptr) {}

    void registerForMentor(Mentor& m) {

        if (m.getStudentsAssigned() < m.getMaxLearners()) {

            mentorAssigned = &m;

            m.assignLearner(this);

        } else {

            cout << "MENTOR " << m.name << " HAS NO AVAILABLE SLOTS." << endl;

        }

    }

    void viewMentorDetails() const {

        if (mentorAssigned) {

            cout << "MENTOR NAME: " << mentorAssigned->name << endl;

            for (int i = 0; i < mentorAssigned->getCountExpertise(); i++) {

                mentorAssigned->sportExpertise[i].display();

            }

        } else {

            cout << "NO MENTOR IS ASSIGNED." << endl;

        }

    }

    void updateSportsInterest(Sports s) {

        if (sportOfInterestCount < 20) {

            sportOfInterest[sportOfInterestCount++] = s;

        } else {

            cout << "CAN'T ADD MORE SPORT INTEREST" << endl;

        }

    }

};

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

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

        if (assignedLearners[i] == s) {

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

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

            }

            assignedLearnersCount--;

            cout << "LEARNER REMOVED FROM THE MENTORSHIP." << endl;

            return;

        }

    }

    cout << "LEARNER IS NOT FOUND UNDER THIS MENTOR." << endl;

}

void Mentor::viewLearners() {

    if (assignedLearnersCount == 0) {

        cout << "NO LEARNERS ARE ASSIGNED." << endl;

        return;

    }

    cout << "MENTOR " << name << " HAS THE FOLLOWING LEARNERS:" << endl;

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

        assignedLearners[i]->display();

    }

}

void Mentor::provideGuidance() {

        cout << "MENTOR " << name << " IS PROVIDING GUIDANCE TO ASSIGNED LEARNERS." << endl;

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

            cout << "GUIDING LEARNER: " << assignedLearners[i]->name << endl;

        }

    }

int main() {

    Mentor mentor(1, 2, "ALI");

    Learner l1(12, 18, "MUZAMIL");

    Learner l2(13, 18, "MUZZY");

    Learner l3(14, 19, "AHMED");

    l1.registerForMentor(mentor);

    l2.registerForMentor(mentor);

    l3.registerForMentor(mentor);

    mentor.viewLearners();

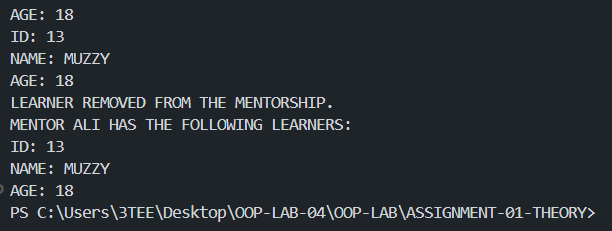
    mentor.removeLearner(&l1);

    mentor.viewLearners();

    return 0;

}

OUTPUT



QUESTION 02:

SOURCE CODE:

#include <iostream>

using namespace std;

#include <string>

#include <tuple>

string lowercase(string k)

{

    string Lowercase = "";

    int l = k.length();

    for (int i = 0; i < l; i++)

    {

        Lowercase += tolower(k[i]);

    }

    return Lowercase;

}

class Robot

{

    string name;

    int hits;

public:

    Robot(string name) : name(name) { hits = 0; }

    Robot() : name("") { hits = 0; }

    int getHits()

    {

        return hits;

    }

    string getName()

    {

        return name;

    }

    void setName(string NAME)

    {

        name = NAME;

    }

    void hitBall(int &BallX, int &BallY, string &Direction)

    {

        string d = lowercase(Direction);

        if (d == "up")

        {

            BallY++;

            hits++;

        }

        else if (d == "down")

        {

            BallY--;

            hits++;

        }

        else if (d == "right")

        {

            BallX++;

            hits++;

        }

        else if (d == "left")

        {

            BallX--;

            hits++;

        }

        else

        {

            cout << "INVALID DIRECTION";

        }

        return;

    }

};

class Ball

{

    int y;

    int x;

public:

    Ball() : x(0), y(0) {}

    int &refgetX()

    {

        return x;

    }

    int &refgetY()

    {

        return y;

    }

    int getX()

    {

        return x;

    }

    int getY()

    {

        return y;

    }

    void move(int dx, int dy)

    {

        x = dx;

        y = dy;

    }

    tuple<int, int> getPosition()

    {

        return make\_tuple(x, y);

    }

    void setBallX(int X)

    {

        x = X;

    }

    void setBallY(int Y)

    {

        y = Y;

    }

};

class Goal

{

    public:

    int x, y;

    Goal() : x(3), y(3) {}

    bool isGoalReached(int BallX, int BallY)

    {

        if (x == BallX && y == BallY)

        {

            return true;

        }

        return false;

    }

};

class Team

{

public:

    string teamName;

    Robot \*getPlayer()

    {

        return player;

    }

    void setPlayer(Robot \*p)

    {

        player = p;

    }

private:

    Robot \*player;

};

class Game

{

    Team \*teamOne;

    Team \*teamTwo;

    Goal goal;

    Ball ball;

public:

    void setTeam(Team \*t,int teamNO){

        if(teamNO == 1){

            teamOne = t;

        }else if(teamNO == 2){

            teamTwo = t;

        }

    }

    void play(Team \*team)

    {

        ball.setBallX(0);

        ball.setBallY(0);

        do

        {

            cout << "ENTER THE DIRECTION: ";

            string direction;

            cin >> direction;

            team->getPlayer()->hitBall(ball.refgetX(), ball.refgetY(), direction);

        } while (!goal.isGoalReached(ball.getX(), ball.getY()));

    }

    void declareWinner()

    {

        if (teamOne->getPlayer()->getHits() < teamTwo->getPlayer()->getHits())

        {

            cout << teamOne->teamName << "IS THE WINNER";

        }

        else if (teamOne->getPlayer()->getHits() > teamTwo->getPlayer()->getHits())

        {

            cout << teamTwo->teamName << "IS THE WINNER";

        }

        else

        {

            cout << " IT IS THE TIE";

        }

    }

    void StartGame()

    {

        cout << "TEAM ONE NAME: ";

        cin >> teamOne->teamName;

        cout << "TEAM TWO NAME: ";

        cin >> teamTwo->teamName;

        cout << "GAME STARTED"<<endl;

        cout << "TEAM "<< teamOne->teamName + " TURN: " << endl;

        play(teamOne);

        cout << "TEAM "<< teamTwo->teamName + " TURN: " << endl;

        play(teamTwo);

        declareWinner();

    }

};

int main()

{

    Team teamOne;

    Team teamTwo;

    Robot robotOne("Player1");

    Robot robotTwo("Player2");

    teamOne.setPlayer(&robotOne);

    teamOne.setPlayer(&robotTwo);

    Game game;

    game.setTeam(&teamOne,1);

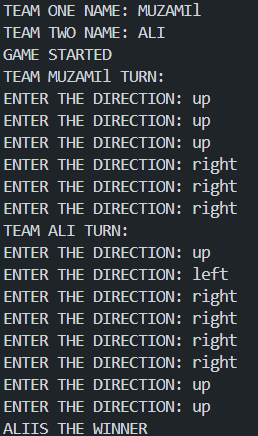
    game.setTeam(&teamTwo,2);

    game.StartGame();

    return 0;

}

OUTPUT:



QUESTION 03:

SOURCE CODE:

#include <iostream>

using namespace std;

class Vehicle {

private:

    string model;

    double rent;

    string licenseTypeRequired;

public:

    bool isAvailable;

    Vehicle() : isAvailable(true) {}

    Vehicle(string model, double rent, string licenseTypeRequired)

        : model(model), rent(rent), licenseTypeRequired(licenseTypeRequired), isAvailable(true) {}

    double getRent() { return rent; }

    string getModel() { return model; }

    string getLicenseTypeRequired() { return licenseTypeRequired; }

    void displayVehicle() {

        cout << "VEHICLE MODEL: " << model << endl;

        cout << "VEHICLE RENT PER DAY: " << rent << endl;

    }

};

class User {

private:

    int age;

    string licenseType;

    string contact;

    int userID;

public:

    User() {}

    User(int age, int userID, string contact, string licenseType)

        : age(age), userID(userID), contact(contact), licenseType(licenseType) {}

    bool rentVehicle(Vehicle \*V) {

        return licenseType == V->getLicenseTypeRequired();

    }

    void setAge(int newAge) { age = newAge; }

    void setUserID(int newUserID) { userID = newUserID; }

    void setContact(string newContact) { contact = newContact; }

    void setLicenseType(string newLicenseType) { licenseType = newLicenseType; }

    int getID() { return userID; }

};

class RentalSystem {

private:

    User users[100];

    Vehicle \*\*vehicleAvailable;

    int totalVehicle;

    int userCount;

public:

    RentalSystem() : totalVehicle(0), vehicleAvailable(nullptr), userCount(0) {}

    void ALLOCATE\_VEHICLE(string model, double rent, string licenseTypeRequired) {

        Vehicle \*\*newArray = new Vehicle \*[totalVehicle + 1];

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

            newArray[i] = vehicleAvailable[i];

        }

        newArray[totalVehicle] = new Vehicle(model, rent, licenseTypeRequired);

        if (vehicleAvailable) {

            delete[] vehicleAvailable;

        }

        vehicleAvailable = newArray;

        totalVehicle++;

    }

    void DEALLOCATE\_ARRAY() {

        if (vehicleAvailable) {

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

                delete vehicleAvailable[i];

            }

            delete[] vehicleAvailable;

            vehicleAvailable = nullptr;

        }

    }

    void rentVehicle(int userID, string vehicleModel, int Days) {

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

            if (vehicleAvailable[i]->getModel() == vehicleModel && vehicleAvailable[i]->isAvailable) {

                for (int j = 0; j < userCount; j++) {

                    if (users[j].getID() == userID) {

                        if (users[j].rentVehicle(vehicleAvailable[i])) {

                            cout << "YOU HAVE SUCCESSFULLY RENTED THE " << vehicleModel << endl;

                            cout << "RENTED FOR DAYS: " << Days << endl;

                            cout << "TOTAL RENT: " << vehicleAvailable[i]->getRent() \* Days << endl;

                            vehicleAvailable[i]->isAvailable = false;

                        } else {

                            cout << "SORRY, LICENSE TYPE FOR " << vehicleModel << " DOES NOT MATCH." << endl;

                        }

                        return;

                    }

                }

                cout << "SORRY, USER NOT FOUND." << endl;

                return;

            }

        }

        cout << "SORRY, VEHICLE NOT AVAILABLE." << endl;

    }

    void displayAvailableVehicles() {

        int k = 1;

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

            if (vehicleAvailable[i]->isAvailable) {

                cout << k++ << ". ";

                vehicleAvailable[i]->displayVehicle();

            }

        }

    }

    void registerUser(int age, int userID, string contact, string licenseType) {

        users[userCount++] = User(age, userID, contact, licenseType);

    }

    void setAge(int userID, int newAge) {

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

            if (users[i].getID() == userID) {

                users[i].setAge(newAge);

                return;

            }

        }

        cout << "USER NOT FOUND." << endl;

    }

    void setContact(int userID, string newContact) {

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

            if (users[i].getID() == userID) {

                users[i].setContact(newContact);

                return;

            }

        }

        cout << "USER NOT FOUND." << endl;

    }

    void setLicenseType(int userID, string newLicenseType) {

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

            if (users[i].getID() == userID) {

                users[i].setLicenseType(newLicenseType);

                return;

            }

        }

        cout << "USER NOT FOUND." << endl;

    }

    ~RentalSystem() {

        DEALLOCATE\_ARRAY();

    }

};

int main() {

    RentalSystem GARAGESLOT;

    GARAGESLOT.registerUser(25, 0, "123-456-7890", "INTERMEDIATE");

    GARAGESLOT.registerUser(30, 1, "987-654-3210", "FULL");

    GARAGESLOT.ALLOCATE\_VEHICLE("COROLLA", 1500, "LEARNER");

    GARAGESLOT.ALLOCATE\_VEHICLE("CIVIC", 1800, "FULL");

    GARAGESLOT.ALLOCATE\_VEHICLE("BUGGATI CHIRON", 2500, "INTERMEDIATE");

    GARAGESLOT.rentVehicle(1,"CIVIC",5);

    cout << "AVAILABLE VEHICLES:" << endl;

    GARAGESLOT.displayAvailableVehicles();

    // modifying the vehicle

    GARAGESLOT.setAge(0, 26);

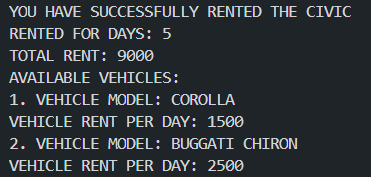
    GARAGESLOT.setContact(0, "321-654-0987");

    GARAGESLOT.setLicenseType(0, "FULL");

    return 0;

}

OUTPUT:



QUESTION 04:

SOURCE CODE:

#include <iostream>

using namespace std;

class Student {

private:

    const int studentID;

    string name;

    bool isActive;

    double balance;

    string stop;

public:

    static int totalStudents;

    Student(int id, string n, string s) : studentID(id), name(n), stop(s), isActive(false), balance(0.0) {

        totalStudents++;

    }

    Student() : studentID(-1) ,balance(0.0),isActive(false){}

    void payFee(double amount) {

        balance += amount;

        isActive = true;

        cout << "PAYMENT SUCCESSFUL=> CARD ACTIVATED." << endl;

    }

    void tapCard() const {

        cout << (isActive ? "ATTENDANCE RECORDED FOR " + name : "CARD INACTIVE. PLEASE PAY THE SEMESTER FEE.") << endl;

    }

    int getID() const {

        return studentID;

    }

    string getName() const {

        return name;

    }

    bool getIsActive() const {

        return isActive;

    }

    double getBalance() const {

        return balance;

    }

    string getStop() const {

        return stop;

    }

    void setName(string n) {

        name = n;

    }

    void setStop(string s) {

        stop = s;

     }

};

int Student::totalStudents = 0;

class BusRoute {

private:

    string stops[10];

    int totalStops;

public:

    string routeName;

    BusRoute() : totalStops(0) {}

    void addRoute(string name) {

        routeName = name;

        totalStops = 0;

        cout << "ROUTE ADDED SUCCESSFULLY." << endl;

    }

    void addStop(string stop) {

        if (totalStops < 10) {

            stops[totalStops++] = stop;

            cout << "STOP " << stop << " ADDED TO ROUTE." << endl;

        } else {

            cout << "MAXIMUM NUMBER OF STOPS REACHED." << endl;

        }

    }

    bool isValidStop(string stop) const {

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

            if (stops[i] == stop) return true;

        }

        return false;

    }

};

class TransportSystem {

private:

    Student students[100];

    BusRoute routes[10];

    int studentCount;

    int routeCount;

public:

    TransportSystem() : studentCount(0), routeCount(0) {}

    void registerStudent(int id, string name, string stop) {

        if (studentCount < 100) {

            new (&students[studentCount]) Student(id, name, stop);

            studentCount++;

            cout << "STUDENT REGISTERED SUCCESSFULLY." << endl;

        } else {

            cout << "STUDENT REGISTRATION LIMIT REACHED." << endl;

        }

    }

    void addRoute(string name) {

        if (routeCount < 10) {

            routes[routeCount].addRoute(name);

            routeCount++;

        } else {

            cout << "MAXIMUM NUMBER OF ROUTES REACHED." << endl;

        }

    }

    void addStopToRoute(string routeName, string stop) {

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

            if (routes[i].routeName == routeName) {

                routes[i].addStop(stop);

                return;

            }

        }

        cout << "ROUTE NOT FOUND." << endl;

    }

    void processPayment(int id, double amount) {

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

            if (students[i].getID() == id) {

                students[i].payFee(amount);

                return;

            }

        }

        cout << "STUDENT NOT FOUND." << endl;

    }

    void tapCard(int id) const {

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

            if (students[i].getID() == id) {

                students[i].tapCard();

                return;

            }

        }

        cout << "STUDENT NOT FOUND." << endl;

    }

    static int getTotalStudents(){

        return Student::totalStudents;

    }

    ~TransportSystem(){

        cout<<"SYSTEM IS TERMINATED";

    }

};

int main() {

    TransportSystem system;

    system.registerStudent(1, "MUSTUFA", "QUIDABAD");

    system.registerStudent(2, "MUZAMIL", "GULSHAN-E-HADEED");

    system.addRoute("ROUTE 1");

    system.addStopToRoute("ROUTE 1", "QUIDABAD");

    system.addRoute("ROUTE 2");

    system.addStopToRoute("ROUTE 2", "GULSHAN-E-HADEED");

    system.addStopToRoute("ROUTE 2", "GULSHAN-E-JOHAR");

    system.processPayment(1, 5000);

    system.tapCard(1);

    system.tapCard(2);

    cout << "TOTAL STUDENTS REGISTERED: " << TransportSystem::getTotalStudents() << endl;

    return 0;

}

OUTPUT:

