QUESTION 1:

#include <iostream>

using namespace std;

class Passenger {

protected:

    int ID;

    string name;

    bool isActive;

    double balance;

    double feeToPay;

    string stop;

public:

    static int totalPassengers;

    Passenger(int id, string n, string s, double fee)

        : ID(id), name(n), stop(s), isActive(false), balance(0.0), feeToPay(fee) {

        totalPassengers++;

    }

    Passenger() : ID(-1), balance(0.0), isActive(false), feeToPay(0.0) {}

    virtual void display() {

        cout << "ID: " << ID << endl;

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

        cout << "STOP: " << stop << endl;

    }

    virtual void payFee(double amount) {

        balance += amount;

        if (balance >= feeToPay) {

            isActive = true;

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

        } else {

            cout << "PARTIAL PAYMENT RECEIVED. PAY REMAINING: "

                 << (feeToPay - balance) << " TO ACTIVATE CARD." << endl;

        }

    }

    void tapCard() const {

        cout << (isActive ? "ATTENDANCE RECORDED FOR " + name

                          : "CARD INACTIVE. PLEASE PAY THE FEE.") << endl;

    }

    int getID() const { return ID; }

    string getName() const { return name; }

    bool getIsActive() const { return isActive; }

    double getBalance() const { return balance; }

    double getFeeToPay() const { return feeToPay; }

    string getStop() const { return stop; }

    void setName(string n) { name = n; }

    void setStop(string s) { stop = s; }

    void setFeeToPay(double fee) { feeToPay = fee; }

};

int Passenger::totalPassengers = 0;

class Student : public Passenger {

    string department;

public:

    Student(int id, string n, string s, double fee, string department)

        : Passenger(id, n, s, fee), department(department) {}

    void display() override {

        Passenger::display();

        cout << "DEPARTMENT: " << department << endl;

    }

};

class Teacher : public Passenger {

public:

    int monthsPaid;

    string designation;

    Teacher(int id, string n, string s, double fee, string designation)

        : monthsPaid(0), Passenger(id, n, s, fee), designation(designation) {}

    void display() override {

        Passenger::display();

        cout << "DESIGNATION: " << designation << endl;

    }

    void payFee(double amount) override {

        balance += amount;

        monthsPaid++;

        cout << "MONTHLY PAYMENT RECEIVED: " << amount << endl;

        cout << "TOTAL MONTHS PAID: " << monthsPaid << endl;

        if (balance >= feeToPay) {

            isActive = true;

            cout << "FULL FEE PAID => CARD ACTIVATED." << endl;

        } else {

            cout << "REMAINING FEE: " << (feeToPay - balance) << endl;

        }

    }

};

class Staff : public Passenger {

private:

    string role;

    string shiftTiming;

    string assignedRoute;

public:

    Staff(int id, string n, string s, double fee, string r, string shift, string route)

        : Passenger(id, n, s, fee), role(r), shiftTiming(shift), assignedRoute(route) {}

    void display() override {

        Passenger::display();

        cout << "ROLE: " << role << endl;

        cout << "TIMINGS: " << shiftTiming << endl;

        cout << "ASSIGNED ROUTE: " << assignedRoute << endl;

    }

};

class BusRoute {

private:

    string stops[10];

    int totalStops;

public:

    string routeName;

    BusRoute() : totalStops(0) {}

    void addRoute(string name) {

        routeName = name;

        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;

    }

    bool operator==(const BusRoute& other) const {

        if (routeName != other.routeName || totalStops != other.totalStops)

            return false;

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

            if (stops[i] != other.stops[i]) return false;

        }

        return true;

    }

};

class TransportSystem {

private:

    string name;

    Passenger\* students[100];

    Staff\* StaffMembers[100];

    Teacher\* Teachers[100];

    BusRoute routes[10];

    int studentCount;

    int teacherCount;

    int staffCount;

    int routeCount;

public:

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

    void Register(Teacher \*t) {

        if (teacherCount < 100) {

            Teachers[teacherCount++] = t;

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

        } else {

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

        }

    }

    void Register(Student \*s) {

        if (studentCount < 100) {

            students[studentCount++] = s;

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

        } else {

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

        }

    }

    void Register(Staff \*sf) {

        if (staffCount < 100) {

            StaffMembers[staffCount++] = sf;

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

        } else {

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

        }

    }

    void tapCard(int id) const {

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

            if (students[i]->getID() == id) {

                students[i]->tapCard();

                return;

            }

        }

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

            if (Teachers[i]->getID() == id) {

                Teachers[i]->tapCard();

                return;

            }

        }

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

            if (StaffMembers[i]->getID() == id) {

                StaffMembers[i]->tapCard();

                return;

            }

        }

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

    }

};

int main() {

    cout << "NAME: MUZAMIL ALI" << endl;

    cout << "ROLL NO: 24K-1023" << endl;

    TransportSystem system;

    Student\* student1 = new Student(1, "MUSTUFA", "QUIDABAD", 5000, "Computer Science");

    system.Register(student1);

    Teacher\* teacher1 = new Teacher(2, "MUZAMIL", "GULSHAN-E-HADEED", 5000, "Professor");

    system.Register(teacher1);

    BusRoute route1, route2;

    route1.addRoute("ROUTE 1");

    route1.addStop("QUIDABAD");

    route2.addRoute("ROUTE 1");

    route2.addStop("QUIDABAD");

    if (route1 == route2) cout << "ROUTES ARE SAME." << endl;

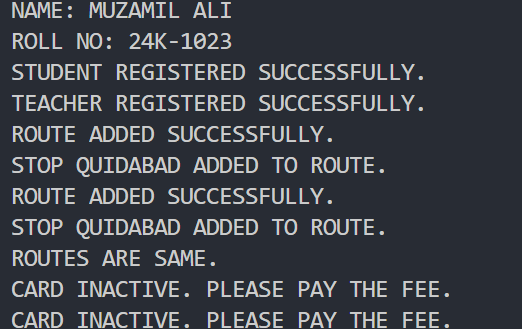
    system.tapCard(1);

    system.tapCard(2);

    return 0;

}

OUTPUT:



QUESTION 2:

#include <iostream>

using namespace std;

class Visitor;

class HauntedHouse;

class Ghost

{

protected:

    string name;

    int scareLevel;

public:

    Ghost() : name(""), scareLevel(0) {}

    Ghost(string name, int scareLevel) : name(name), scareLevel(scareLevel) {}

    virtual ~Ghost() {}

    string getName() const { return name; }

    int getScareLevel() const { return scareLevel; }

    virtual void haunt() { cout << name << " IS HAUNTING !!!" << endl; }

    friend void visit(Visitor \*visitor, int totalVisitors, HauntedHouse &house);

};

class Poltergeist : virtual public Ghost

{

public:

    Poltergeist(string name, int scareLevel) : Ghost(name, scareLevel) {}

    void haunt() override { cout << name << " IS MOVING OBJECTS !!!" << endl; }

};

class Banshee : public Ghost

{

public:

    Banshee(string name, int scareLevel) : Ghost(name, scareLevel) {}

    void haunt() override { cout << name << " IS SCREAMING !!! " << endl; }

};

class ShadowGhost :  virtual public Ghost

{

public:

    ShadowGhost(string name, int scareLevel) : Ghost(name, scareLevel) {}

    void haunt() override { cout << name << " IS WHISPERING CREEPILY... " << endl; }

};

class ShadowPoltergeist : public ShadowGhost, public Poltergeist

{

public:

    ShadowPoltergeist(string name, int scareLevel) : Ghost(name, scareLevel), ShadowGhost(name, scareLevel), Poltergeist(name, scareLevel) {}

    void haunt() override

    {

        ShadowGhost::haunt();

        Poltergeist::haunt();

    }

};

class HauntedHouse

{

private:

    string name;

    Poltergeist \*pGhost;

    Banshee \*bGhost;

    ShadowGhost \*sGhost;

    ShadowPoltergeist \*spGhost;

public:

    HauntedHouse(string name, Poltergeist \*p, Banshee \*b, ShadowGhost \*s, ShadowPoltergeist \*sp) : name(name), pGhost(p), bGhost(b), sGhost(s), spGhost(sp) {}

    ~HauntedHouse()

    {

        delete pGhost;

        delete bGhost;

        delete sGhost;

        delete spGhost;

    }

    friend void visit(Visitor \*visitor, int totalVisitors, HauntedHouse &house);

};

class Visitor

{

    friend void visit(Visitor \*visitor, int totalVisitors, HauntedHouse &house);

private:

    string name;

    int braveryPoints;

public:

    Visitor(string name, int braveryPts) : name(name), braveryPoints(braveryPts) {}

    string getName() { return name; }

    bool isCowardly() const { return braveryPoints >= 1 && braveryPoints <= 4; }

    bool isAverage() const { return braveryPoints >= 5 && braveryPoints <= 7; }

    bool isFearless() const { return braveryPoints >= 8 && braveryPoints <= 10; }

    void react(Ghost &ghost)

    {

        if (isFearless() && ghost.getScareLevel() <= 4)

        {

            cout << getName() << " TAUNTS " << ghost.getName() << " FEARLESSLY!" << endl;

        }

        else if (isCowardly() && ghost.getScareLevel() >= 8)

        {

            cout << getName() << " SCREAMS AND RUNS AWAY FROM " << ghost.getName() << "!" << endl;

        }

        else if (isAverage() && (ghost.getScareLevel() >= 5 && ghost.getScareLevel() <= 7))

        {

            cout << getName() << " GETS A SHAKY VOICE NEAR " << ghost.getName() << "." << endl;

        }

        else

        {

            cout << getName() << " DOESN'T KNOW HOW TO REACT TO " << ghost.getName() << "!" << endl;

        }

    }

};

void visit(Visitor \*visitor, int totalVisitors, HauntedHouse &house)

{

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

    {

        cout << "\nVISITORS: " << visitor[i].getName() << " IS EXPLORING THE " << house.name << "..." << endl;

        house.pGhost->haunt();

        visitor[i].react(\*house.pGhost);

        house.bGhost->haunt();

        visitor[i].react(\*house.bGhost);

        house.sGhost->haunt();

        visitor[i].react(\*house.sGhost);

        house.spGhost->haunt();

        visitor[i].react(\*house.spGhost);

    }

}

int main()

{

    cout << "NAME: MUZAMIL ALI" << endl;

    cout << "ROLL NO: 24K-1023" << endl;

    HauntedHouse house1("BHOOT BANGLA",

                         new Poltergeist("YURO", 3),

                         new Banshee("NOKU", 7),

                         new ShadowGhost("ROKU", 5),

                         new ShadowPoltergeist("HANSE", 9));

    HauntedHouse house2("GHOST MANSION",

                         new Poltergeist("MORGAN", 4),

                         new Banshee("TERRA", 10),

                         new ShadowGhost("DRAKE", 6),

                         new ShadowPoltergeist("CREEPER", 8));

    Visitor visitors[3] = {

        Visitor("MUZZY", 3),

        Visitor("JANE", 6),

        Visitor("LUCAS", 9)};

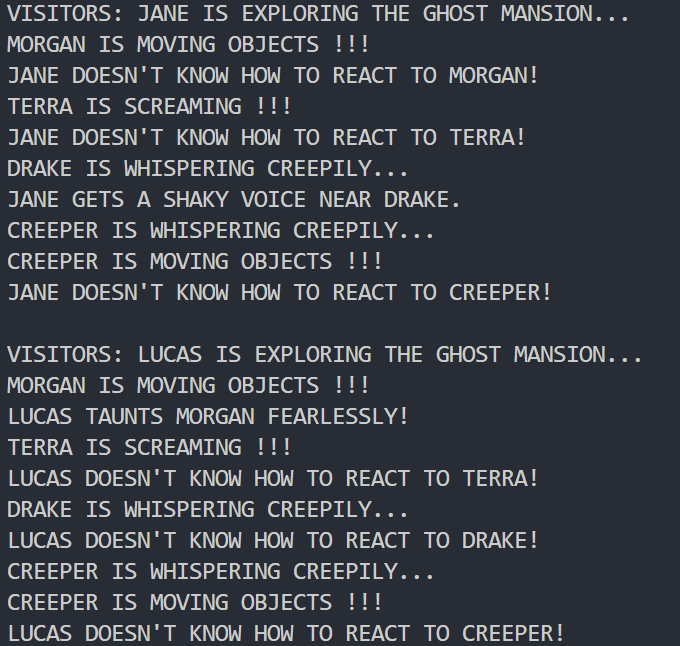
    visit(visitors, 3, house1);

    visit(visitors, 3, house2);

    return 0;

}

OUTPUT:



QUESTION 3:

#include <iostream>

#include <string>

using namespace std;

class Vehicle

{

protected:

    string vehicleID;

    int vehicleSpeed;

    int vehicleCapacity;

    int energyEfficiency;

    static int totalDeliveries;

public:

    Vehicle(string id) : vehicleID(id) {}

    virtual void calculateRoute() = 0;

    virtual double estimateTime(double distance) = 0;

    virtual void command(string packageID)

    {

        totalDeliveries++;

        cout << "BASIC DELIVERY COMMAND FOR PACKAGE: " << packageID << endl;

    }

    virtual void command(string packageID, string urgency)

    {

        totalDeliveries++;

        cout << "URGENT DELIVERY COMMAND FOR PACKAGE: " << packageID << endl;

    }

    static int getTotalDeliveries() { return totalDeliveries; }

    friend bool operator==(const Vehicle &a, const Vehicle &b);

    friend Vehicle \*resolveConflict(Vehicle \*a, Vehicle \*b);

};

int Vehicle::totalDeliveries = 0;

bool operator==(const Vehicle &a, const Vehicle &b)

{

    return (a.vehicleSpeed == b.vehicleSpeed) &&

           (a.vehicleCapacity == b.vehicleCapacity) &&

           (a.energyEfficiency == b.energyEfficiency);

}

Vehicle \*resolveConflict(Vehicle \*a, Vehicle \*b)

{

    int aScore = a->vehicleSpeed \* a->energyEfficiency + a->vehicleCapacity;

    int bScore = b->vehicleSpeed \* b->energyEfficiency + b->vehicleCapacity;

    return aScore >= bScore ? a : b;

}

class RamzanDrone : public Vehicle

{

public:

    RamzanDrone(string id) : Vehicle(id)

    {

        vehicleSpeed = 100;

        vehicleCapacity = 10;

        energyEfficiency = 80;

    }

    void calculateRoute() override

    {

        cout << "CALCULATING AERIAL ROUTE FOR HIGH-SPEED DELIVERY" << endl;

    }

    double estimateTime(double distance) override

    {

        return distance / vehicleSpeed;

    }

    void command(string packageID, string urgency) override

    {

        if (urgency == "urgent")

        {

            cout << "DRONE ACTIVATING TURBO MODE FOR IFTAR MEAL: " << packageID << endl;

            vehicleSpeed \*= 2;

        }

        Vehicle::command(packageID, urgency);

    }

};

class RamzanTimeShip : public Vehicle

{

public:

    RamzanTimeShip(string id) : Vehicle(id)

    {

        vehicleSpeed = 80;

        vehicleCapacity = 50;

        energyEfficiency = 70;

    }

    void calculateRoute() override

    {

        cout << "VERIFYING HISTORICAL CONSISTENCY FOR TIME TRAVEL" << endl;

    }

    double estimateTime(double distance) override

    {

        return distance / vehicleSpeed;

    }

    void command(string packageID, string urgency) override

    {

        if (urgency == "urgent")

        {

            cout << "TIMESHIP VALIDATING HISTORICAL RECORDS FOR: " << packageID << endl;

        }

        Vehicle::command(packageID, urgency);

    }

};

class RamzanHyperPod : public Vehicle

{

public:

    RamzanHyperPod(string id) : Vehicle(id)

    {

        vehicleSpeed = 120;

        vehicleCapacity = 1000;

        energyEfficiency = 90;

    }

    void calculateRoute() override

    {

        cout << "NAVIGATING UNDERGROUND HYPERLOOP NETWORK" << endl;

    }

    double estimateTime(double distance) override

    {

        return distance / vehicleSpeed;

    }

};

int main()

{

    cout << "NAME: MUZAMIL ALI" << endl;

    cout << "ROLL NO: 24K-1023" << endl;

    RamzanDrone drone("DRONE");

    RamzanTimeShip timeship("TIMESHIP");

    RamzanHyperPod pod("POD");

    drone.command("IFTAR\_MEAL1", "urgent");

    drone.command("IFTAR\_MEAL2", "urgent");

    timeship.command("HISTORICAL\_ITEM", "urgent");

    pod.calculateRoute();

    cout << "ESTIMATED DELIVERY TIME: " << pod.estimateTime(360) << " HOURS" << endl;

    Vehicle \*result = resolveConflict(&drone, &pod);

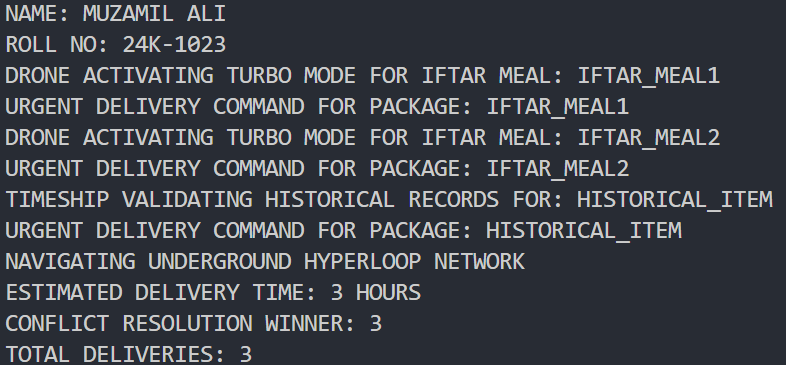
    cout << "CONFLICT RESOLUTION WINNER: " << result->getTotalDeliveries() << endl;

    cout << "TOTAL DELIVERIES: " << Vehicle::getTotalDeliveries() << endl;

    return 0;

}

OUTPUT:



QUESTION 4:

#include <iostream>

#include <string>

using namespace std;

int hashPassword(const string &password) {

    unsigned long hash = 5381;

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

        hash = ((hash \* 33) + password[i]);

    return hash;

}

class User {

protected:

    string userName;

    string listOfPermission[10];

    int hashedPassword;

    string userEmail;

public:

    User() {}

    User(string name, string email, string LOF[], string password) : userName(name), userEmail(email) {

        for (int i = 0; i < 10; i++)

            listOfPermission[i] = LOF[i];

        hashedPassword = hashPassword(password);

    }

    bool authenticate(string pass) {

        return hashPassword(pass) == hashedPassword;

    }

    virtual void display() {

        cout << "NAME: " << userName << endl;

        cout << "EMAIL: " << userEmail << endl;

    }

    bool access(int lab) {

        if (listOfPermission[lab] == "Full\_Lab\_access") {

            cout << userName << " HAS BEEN GRANTED LAB ACCESS\n" << endl;

            return true;

        }

        else {

            cout << userName << " HAS BEEN GRANTED LAB ACCESS\n" << endl;

        }

        return false;

    }

    virtual void performAction(string action) = 0;

};

class Student : public User {

protected:

    int\* assignment;

    static int totalAssignment;

public:

    Student() {}

    Student(string name, string email, string LOF[], string password) : User(name, email, LOF, password) {

        assignment = nullptr;

    }

    void addAssignment() {

        int\* temp;

        if (assignment) {

            temp = new int[totalAssignment];

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

                temp[i] = assignment[i];

            delete[] assignment;

        }

        assignment = new int[++totalAssignment];

        for (int i = 0; i < totalAssignment - 1; i++)

            assignment[i] = temp[i];

        delete[] temp;

        assignment[totalAssignment - 1] = 0;

    }

    void submitAssignment(int assignmentNumber) {

        assignment[assignmentNumber - 1] = 1;

    }

    void display() {

        User::display();

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

            cout << "ASSIGNMENT " << i + 1 << ": " << assignment[i] << endl;

    }

    void performAction(string action) {

        if (action == "submit\_assignment")

            cout << userName << " SUBMITTED AN ASSIGNMENT" << endl;

        else

            cout << "ACTION NOT PERMITTED FOR STUDENT" << endl;

    }

};

int Student::totalAssignment = 1;

class TA : public Student {

    Student assignedStudent[10];

    int currentStudent;

    string project[2];

    bool taAssigned;

    int projectNumber;

public:

    TA() {}

    TA(string name, string email, string LOF[], string password) : Student(name, email, LOF, password), projectNumber(0), currentStudent(0), taAssigned(false) {}

    void viewProject() {

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

            cout << i + 1 << " PROJECT: " << project[i] << endl;

    }

    void addProject(string newProject) {

        if (projectNumber != 2) {

            project[projectNumber++] = newProject;

            cout << "PROJECT ASSIGNED" << endl;

        }

        else

            cout << "NEW PROJECT LIMIT EXCEEDED!" << endl;

    }

    void display() {

        Student::display();

        viewProject();

    }

    void assignedAsTA(bool aat) { taAssigned = aat; }

    void assignStudent(Student& s) {

        if (taAssigned) {

            if (currentStudent != 10)

                assignedStudent[currentStudent++] = s;

            else

                cout << "MORE THAN 10 CAN'T BE ASSIGNED" << endl;

        }

        else

            cout << "STUDENT IS NOT TA SO HE CAN'T MANAGE STUDENTS" << endl;

    }

    void performAction(string action) {

        if (action == "view\_projects")

            cout << userName << " IS VIEWING PROJECTS" << endl;

        else if (action == "manage\_students")

            cout << userName << " IS MANAGING STUDENTS" << endl;

        else

            Student::performAction(action);

    }

};

class Professor : public User {

    TA ta[4];

public:

    Professor(string name, string email, string LOF[], string password) : User(name, email, LOF, password) {}

    void assignedProject(int n, string project) {

        ta[n - 1].addProject(project);

    }

    void display() {

        User::display();

        for (int i = 0; i < 4; i++) {

            cout << "TA " << i + 1 << ":" << endl;

            ta[i].viewProject();

        }

    }

    void performAction(string action) {

        if (action == "assign\_projects")

            cout << userName << " IS ASSIGNING PROJECTS" << endl;

        else if (action == "full\_lab\_access")

            cout << userName << " HAS FULL LAB ACCESS" << endl;

        else

            cout << "ACTION NOT PERMITTED FOR PROFESSOR" << endl;

    }

};

void authenticateAndPerformAction(User\* user, string action) {

    user->performAction(action);

}

int main() {

    cout << "NAME: MUZAMIL ALI" << endl;

    cout << "ROLL NO: 24K-1023" << endl;

    string lf[10] = { "Full\_Lab\_access", "Full\_Lab\_access", "NOT\_ACCESS", "Full\_Lab\_access", "NOT\_ACCESS", "NOT\_ACCESS", "Full\_Lab\_access", "NOT\_ACCESS", "Full\_Lab\_access", "NOT\_ACCESS" };

    Student s[3] = {

        {"RAHUL", "rahulpirwani704@gmail.com", lf, "&we"},

        {"ROHAN", "rohannlkumar@gmail.com", lf, "&e2234"},

        {"MANAV", "manav@gmail.com", lf, "&we"}

    };

    s[2].addAssignment();

    s[2].addAssignment();

    s[2].submitAssignment(1);

    s[2].display();

    TA t("KARAN", "karanshukari@gmail.com", lf, "iwe\*2!");

    t.addProject("COMPUTER ENGINEERING");

    t.addProject("CYASD");

    t.access(3);

    t.addProject("HOW");

    t.assignStudent(s[1]);

    authenticateAndPerformAction(&t, "xyz");

    t.assignStudent(s[2]);

    Professor p("ALI", "Alikhan777@gmail.com", lf, "were");

    p.assignedProject(1, "XYZ");

    p.assignedProject(2, "XYZ");

    p.assignedProject(2, "DASUF");

    p.assignedProject(3, "XYZ");

    p.display();

}

OUTPUT:

