**Name: Adnan Hatim**

**Roll No: 24K-0656**

**Section: BCS-2F**

**Question 1 Code:**

#include <iostream>

#include <string>

using namespace std;

class User {

    protected:

        int user\_ID;

        string user\_Name;

        bool is\_Active;

    public:

        User() : user\_ID(0), user\_Name(""), is\_Active(false) {}

        User(int id, string name) : user\_ID(id), user\_Name(name), is\_Active(false) {}

        string getName() const {

            return user\_Name;

        }

        int getID() const {

            return user\_ID;

        }

        bool getIsActive() const {

            return is\_Active;

        }

        virtual void payFee(float amount) = 0;

};

class Student : public User {

    private:

        float semester\_fee;

    public:

        Student() : User(), semester\_fee(0.0) {}

        Student(int std\_id, string name) : User(std\_id, name), semester\_fee(0.0) {}

        void payFee(float amount) override {

            semester\_fee += amount;

            if (semester\_fee > 100) {

                is\_Active = true;

                cout << "Card Activated for Student ID: " << user\_ID << endl;

            }

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

        }

};

class Teacher : public User {

    private:

        float monthly\_fee;

    public:

        Teacher() : User(), monthly\_fee(0.0) {}

        Teacher(int tchr\_id, string name) : User(tchr\_id, name), monthly\_fee(0.0) {}

        void payFee(float amount) override {

            monthly\_fee += amount;

            is\_Active = true;

            cout << "Teacher Monthly Fee paid: " << amount << " Total Fee: " << monthly\_fee << endl;

        }

};

class Staff : public User {

    private:

        float monthly\_fee;

    public:

        Staff() : User(), monthly\_fee(0.0) {}

        Staff(int staff\_id, string name) : User(staff\_id, name), monthly\_fee(0.0) {}

        void payFee(float amount) override {

            monthly\_fee += amount;

            is\_Active = true;

            cout << "Staff Monthly Fee paid: " << amount << " Total Fee: " << monthly\_fee << endl;

        }

};

class Bus {

    private:

        int bus\_ID;

        int capacity;

        int current\_Capacity;

    public:

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

        Bus(int id, int cap) : bus\_ID(id), capacity(cap), current\_Capacity(0) {}

        void recordAttendance(int user\_id) {

            if (current\_Capacity < capacity) {

                cout << "Attendance marked for User ID: " << user\_id << endl;

                current\_Capacity++;

            } else {

                cout << "MAX CAPACITY REACHED!" << endl;

            }

        }

};

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 id, string start, string end) : route\_Id(id), start\_Location(start), end\_Location(end), stops\_Count(0) {}

        void addStops(string stop) {

            if (stops\_Count < 10) {

                stops[stops\_Count++] = stop;

                cout << "New Stop Added: " << stop << endl;

            } else {

                cout << "MAX STOPS REACHED!" << endl;

            }

        }

        void removeStops(string stop) {

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

                if (stop == stops[i]) {

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

                    stops\_Count--;

                    return;

                }

            }

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

        }

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

            return (route\_Id == other.route\_Id && start\_Location == other.start\_Location && end\_Location == other.end\_Location);

        }

};

class TransportationSystem {

    private:

        static const int MAX\_USERS = 100;

        static const int MAX\_BUSES = 10;

        static const int MAX\_ROUTES = 10;

        User\* users[MAX\_USERS];

        Bus buses[MAX\_BUSES];

        Route routes[MAX\_ROUTES];

        int userCount;

        int busCount;

        int routeCount;

    public:

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

        void registerStudent(int studentID, string name) {

            if (userCount < MAX\_USERS) {

                users[userCount++] = new Student(studentID, name);

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

            } else {

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

            }

        }

        void registerTeacher(int teacherID, string name) {

            if (userCount < MAX\_USERS) {

                users[userCount++] = new Teacher(teacherID, name);

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

            } else {

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

            }

        }

        void registerStaff(int staffID, string name) {

            if (userCount < MAX\_USERS) {

                users[userCount++] = new Staff(staffID, name);

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

            } else {

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

            }

        }

        void payFee(int userID, float amount) {

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

                if (users[i]->getID() == userID) {

                    users[i]->payFee(amount);

                    return;

                }

            }

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

        }

        void addBus(int id, int capacity) {

            if (busCount < MAX\_BUSES) {

                buses[busCount++] = Bus(id, 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.registerTeacher(2, "Mr. Smith");

    system.registerStaff(3, "Ms. Johnson");

    system.payFee(1, 50.0);

    system.payFee(2, 100.0);

    system.payFee(3, 150.0);

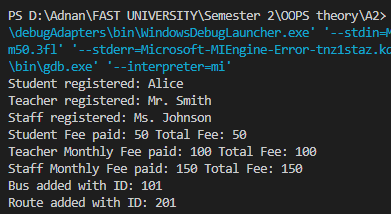
    system.addBus(101, 30);

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

    return 0;

}

**Question 1 Output:**



**Question 2 Code:**

#include<iostream>

#include<string.h>

#include<cstdlib>

#include<ctime>

using namespace std;

class Ghost

{

    protected:

        string nameOfWorker;

        string nameOfGhost;

        int scareLevel;

    public:

        Ghost(string Workername, string ghostName): nameOfWorker(Workername), nameOfGhost(ghostName)

        {

            scareLevel = rand() % 10 + 1;

        };

        virtual void hunt(){};

        friend ostream& operator << (ostream& out, const Ghost& ghost)

        {

            out << "Ghost playing worker Name: " << ghost.nameOfWorker <<", Ghost Name: " << ghost.nameOfGhost << ", Ghost Scare level: " << ghost.scareLevel<<endl;

            return out;

        }

        int getScarelevel(){return scareLevel;}

        Ghost operator+ (const Ghost& other)

        {

            string upgradedGhostWorkerName = this->nameOfWorker + other.nameOfWorker;

            string upgradedGhostName = this->nameOfGhost + other.nameOfGhost;

            Ghost upgradedGhost(upgradedGhostWorkerName, upgradedGhostName);

            upgradedGhost.scareLevel = this->scareLevel + other.scareLevel;

            if(upgradedGhost.getScarelevel() > 10)

                upgradedGhost.scareLevel = 10;

            return upgradedGhost;

        }

        string getNameofWorker(){return nameOfWorker;}

        string getNameofGhost(){return nameOfGhost;}

};

class Poltergeist: virtual public Ghost

{

    private:

        string power;

    public:

        Poltergeist(string n, string gn): Ghost(n, gn){

            power = "Moves Objects";

        };

        virtual void hunt() override

        {

            cout<< nameOfGhost<< " is using his power: "<< power;

        }

        string getPower(){return power;}

};

class Banshee: virtual public Ghost

{

    private:

        string power;

    public:

        Banshee(string n, string gn): Ghost(n, gn){

            power = "Screams Loudly";

        };

        virtual void hunt() override

        {

            cout<< nameOfGhost<< " is using his power: "<< power;

        }

        string getPower(){return power;}

};

class ShadowGhost: virtual public Ghost

{

    private:

        string power;

    public:

        ShadowGhost(string n, string gn): Ghost(n, gn){

            power = "Whisper Creepily";

        };

        virtual void hunt() override

        {

            cout<< nameOfGhost<< " is using his power: "<< power;

        }

        string getPower(){return power;}

};

class ShadowPotergeist: public Poltergeist, public ShadowGhost

{

    public:

        ShadowPotergeist(string n1, string gn1, string n2, string gn2)

            : Poltergeist(n1, gn1), ShadowGhost(n2, gn2), Ghost(n1 + n2, gn1 + gn2) {}

        void hunt() override

        {

            cout << getNameofGhost() << " is using both powers: " << Poltergeist::getPower() << " and " << ShadowGhost::getPower() << endl;

        }

};

class ShadowBanshee: public Banshee, public ShadowGhost

{

    public:

        ShadowBanshee(string n1, string gn1, string n2, string gn2)

            : Banshee(n1, gn1), ShadowGhost(n2, gn2), Ghost(n1 + n2, gn1 + gn2) {}

        void hunt() override

        {

            cout << getNameofGhost() << " is using both powers: " << Banshee::getPower() << " and " << ShadowGhost::getPower() << endl;

        }

};

class Visitor

{

    private:

        string name;

        int bravery;

    public:

        Visitor(string n, int b): name(n), bravery(b){

        };

        void visit(Ghost\* ghost)

        {

            cout << name << " is visiting the haunted house." << endl;

        ghost->hunt();

        if (ghost->getScarelevel() < bravery) {

            cout << name << " laughs at the ghost!" << endl;

        } else if (ghost->getScarelevel() > bravery) {

            cout << name << " screams and runs away!" << endl;

        } else {

            cout << name << " has a shaky voice!" << endl;

        }

        }

};

class HauntedHouse {

private:

    string name;

    Ghost\* ghosts[4];

public:

    HauntedHouse(string n, Ghost\* g1, Ghost\* g2, Ghost\* g3, Ghost\* g4) : name(n) {

        ghosts[0] = g1;

        ghosts[1] = g2;

        ghosts[2] = g3;

        ghosts[3] = g4;

    }

    void simulateVisit(Visitor\* visitors, int numVisitors) {

        cout << "Visitors are entering the " << name << "!" << endl;

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

            visitors[i].visit(ghosts[i % 4]);

        }

    }

};

int main()

{

    srand(static\_cast<unsigned int>(time(0)));

    Poltergeist\* p1 = new Poltergeist("James", "poltergeist");

    Banshee\* b1 = new Banshee("Jane", "Banshee");

    ShadowGhost\* s1 = new ShadowGhost("Mary", "Shadowy");

    ShadowPotergeist\* sp1 = new ShadowPotergeist(p1->getNameofWorker(), "Poltergeist", s1->getNameofWorker(), "ShadowGhost");

    Poltergeist\* p2 = new Poltergeist("Dove", "poltergeist");

    Banshee\* b2 = new Banshee("Hittler", "Banshee");

    ShadowGhost\* s2 = new ShadowGhost("Napolean", "Shadowy");

    ShadowPotergeist\* sp2 = new ShadowPotergeist(p2->getNameofWorker(), "Poltergeist", s2->getNameofWorker(), "ShadowGhost");

    Poltergeist\* p3 = new Poltergeist("Alexandar", "poltergeist");

    Banshee\* b3 = new Banshee("Akbar", "Banshee");

    ShadowGhost\* s3 = new ShadowGhost("Ertugrul", "Shadowy");

    ShadowBanshee\* sp3 = new ShadowBanshee(b3->getNameofWorker(), "Banshee", s3->getNameofWorker(), "ShadowGhost");

    HauntedHouse hauntedHouse1("Spooky Mansion", p1, b1, s1, sp1);

    HauntedHouse hauntedHouse2("Spooky Mansion", p2, b2, s2, sp2);

    HauntedHouse hauntedHouse3("Spooky Mansion", p3, b3, s3, sp3);

    Visitor visitors1[4] = {

        Visitor("Alice", 5),

        Visitor("Bob", 8),

        Visitor("Charlie", 3),

        Visitor("David", 6)

    };

    Visitor visitors2[4] = {

        Visitor("Yushaa", 5),

        Visitor("Murtaza", 8),

        Visitor("Abizer", 3),

        Visitor("Turab", 10)

    };

    Visitor visitors3[4] = {

        Visitor("Thesues", 5),

        Visitor("Tom", 8),

        Visitor("Robert", 3),

        Visitor("Hulk", 6)

    };

    cout<<"\n---------Merging Ghosts---------\n";

    Ghost merging = \*p1 + \*s1;

    cout << merging;

    cout<<"---------Merging Ghosts---------\n";

    cout<<"\n---------Simulating House No 1---------\n";

    hauntedHouse1.simulateVisit(visitors1, 4);

    cout<<"---------Simulating House No 1---------\n";

    cout<<"\n---------Simulating House No 2---------\n";

    hauntedHouse1.simulateVisit(visitors2, 4);

    cout<<"---------Simulating House No 2---------\n";

    cout<<"\n---------Simulating House No 3---------\n";

    hauntedHouse1.simulateVisit(visitors3, 4);

    cout<<"---------Simulating House No 3---------\n";

    delete p1;

    delete b1;

    delete s1;

    delete sp1;

    delete p2;

    delete b2;

    delete s2;

    delete sp2;

    delete p3;

    delete b3;

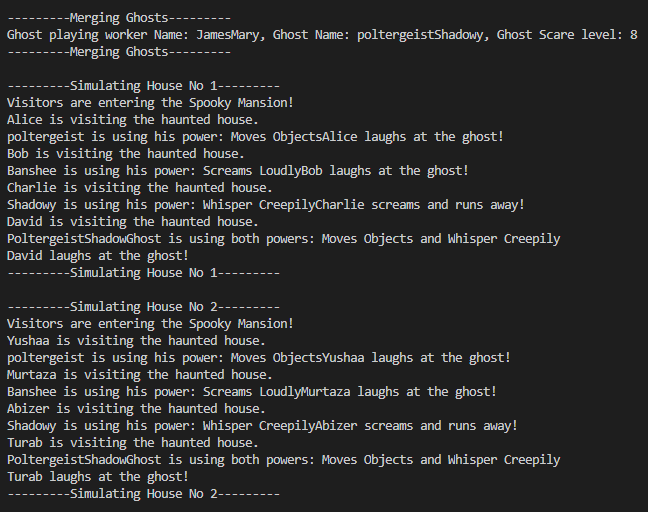
    delete s3;

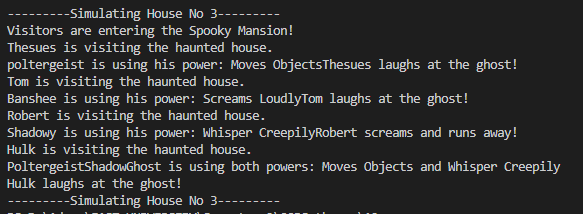
    delete sp3;

    return 0;

}

**Question 2 Output:**





**Question 3 Code:**

#include <iostream>

#include <string>

using namespace std;

class Vehicle {

protected:

    string vehicleID;

    static int activeDeliveries;

    double speed;

    double capacity;

    double energyEfficiency;

public:

    Vehicle(string id, double spd, double cap, double eff) : vehicleID(id), speed(spd), capacity(cap), energyEfficiency(eff) {

        activeDeliveries++;

    }

    virtual void calculateRoute() = 0;

    virtual void estimatedDeliveryTime() = 0;

    static int getActiveDeliveries() {

        return activeDeliveries;

    }

    bool operator==(const Vehicle& other) {

        return (this->speed == other.speed && this->capacity == other.capacity && this->energyEfficiency == other.energyEfficiency);

    }

    friend void resolveConflict(Vehicle\* v1, Vehicle\* v2);

    virtual ~Vehicle() {

        activeDeliveries--;

    }

};

int Vehicle::activeDeliveries = 0;

class RamzanDrone : public Vehicle {

public:

    RamzanDrone(string id) : Vehicle(id, 100.0, 5.0, 0.9) {}

    void calculateRoute() override {

        cout << "Calculating aerial route for RamzanDrone " << vehicleID << endl;

    }

    void estimatedDeliveryTime() override {

        cout << "Estimating delivery time for RamzanDrone " << vehicleID << endl;

    }

    void command(string action, string packageID) {

        cout << "RamzanDrone " << vehicleID << " executing command: " << action << " for package " << packageID << endl;

    }

    void command(string action, string packageID, string urgencyLevel) {

        if (urgencyLevel == "Urgent") {

            cout << "RamzanDrone " << vehicleID << " activating high-speed mode for urgent delivery of package " << packageID << endl;

        } else {

            command(action, packageID);

        }

    }

};

class RamzanTimeShip : public Vehicle {

public:

    RamzanTimeShip(string id) : Vehicle(id, 80.0, 20.0, 0.8) {}

    void calculateRoute() override {

        cout << "Verifying historical consistency for RamzanTimeShip " << vehicleID << endl;

    }

    void estimatedDeliveryTime() override {

        cout << "Estimating delivery time for RamzanTimeShip " << vehicleID << endl;

    }

    void command(string action, string packageID) {

        cout << "RamzanTimeShip " << vehicleID << " executing command: " << action << " for package " << packageID << endl;

    }

    void command(string action, string packageID, string urgencyLevel) {

        if (urgencyLevel == "Urgent") {

            cout << "RamzanTimeShip " << vehicleID << " validating historical accuracy for urgent delivery of package " << packageID << endl;

        } else {

            command(action, packageID);

        }

    }

};

class RamzanHyperPod : public Vehicle {

public:

    RamzanHyperPod(string id) : Vehicle(id, 120.0, 50.0, 0.95) {}

    void calculateRoute() override {

        cout << "Navigating underground tunnel for RamzanHyperPod " << vehicleID << endl;

    }

    void estimatedDeliveryTime() override {

        cout << "Estimating delivery time for RamzanHyperPod " << vehicleID << endl;

    }

    void command(string action, string packageID) {

        cout << "RamzanHyperPod " << vehicleID << " executing command: " << action << " for package " << packageID << endl;

    }

    void command(string action, string packageID, string urgencyLevel) {

        cout << "RamzanHyperPod " << vehicleID << " executing command: " << action << " for package " << packageID << " with urgency " << urgencyLevel << endl;

    }

};

void resolveConflict(Vehicle\* v1, Vehicle\* v2) {

    cout << "Resolving conflict between vehicles " << v1->vehicleID << " and " << v2->vehicleID << endl;

    if (v1->energyEfficiency > v2->energyEfficiency) {

        cout << "Vehicle " << v1->vehicleID << " is prioritized due to higher efficiency." << endl;

    } else if (v1->energyEfficiency < v2->energyEfficiency) {

        cout << "Vehicle " << v2->vehicleID << " is prioritized due to higher efficiency." << endl;

    } else {

        cout << "Both vehicles have equal efficiency. No priority assigned." << endl;

    }

}

int main() {

    RamzanDrone drone1("Drone1");

    RamzanTimeShip timeShip1("TimeShip1");

    RamzanHyperPod hyperPod1("HyperPod1");

    drone1.calculateRoute();

    drone1.estimatedDeliveryTime();

    drone1.command("Deliver", "Package1");

    drone1.command("Deliver", "Package1", "Urgent");

    timeShip1.calculateRoute();

    timeShip1.estimatedDeliveryTime();

    timeShip1.command("Deliver", "Package2");

    timeShip1.command("Deliver", "Package2", "Urgent");

    hyperPod1.calculateRoute();

    hyperPod1.estimatedDeliveryTime();

    hyperPod1.command("Deliver", "Package3");

    hyperPod1.command("Deliver", "Package3", "Urgent");

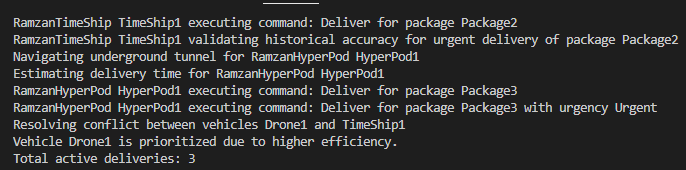
    resolveConflict(&drone1, &timeShip1);

    cout << "Total active deliveries: " << Vehicle::getActiveDeliveries() << endl;

    return 0;

}

**Question 3 Output:**



**Question 4 Code:**

#include <iostream>

#include <string>

using namespace std;

unsigned long hashPassword(const string& password) {

    unsigned long hash = 5381;

    for (char c : password) {

        hash = (hash \* 33) + c;

    }

    return hash;

}

class User {

protected:

    string name;

    string ID;

    string email;

    unsigned long hashed\_password;

public:

    User(string name, string ID, string email, string password)

        : name(name), ID(ID), email(email) {

        hashed\_password = hashPassword(password);

    }

    bool authenticate(const string& password) {

        return hashed\_password == hashPassword(password);

    }

    virtual void display() {

        cout << "Name: " << name << ", ID: " << ID << ", Email: " << email << endl;

    }

    virtual bool accessLab() {

        return false;

    }

    string getName(){return name;}

};

class Student : public User {

private:

    int assignments[10];

public:

    Student(string name, string ID, string email, string password)

        : User(name, ID, email, password) {

        for (int i = 0; i < 10; i++) {

            assignments[i] = 0;

        }

    }

    void submitAssignment(int index) {

        if (index >= 0 && index < 10) {

            assignments[index] = 1;

        }

    }

    void display() override {

        User::display();

        cout << "Assignments Status: ";

        for (int i = 0; i < 10; i++) {

            cout << assignments[i] << " ";

        }

        cout << endl;

    }

};

class TA : public Student {

private:

    string assignedStudents[10];

    int studentCount;

    string projects[2];

    int projectCount;

public:

    TA(string name, string ID, string email, string password)

        : Student(name, ID, email, password), studentCount(0), projectCount(0) {}

    void assignStudent(string studentName) {

        if (studentCount < 10) {

            assignedStudents[studentCount] = studentName;

            studentCount++;

        } else {

            cout << "Cannot assign more than 10 students." << endl;

        }

    }

    bool addProject(string projectName) {

        if (projectCount < 2) {

            projects[projectCount] = projectName;

            projectCount++;

            return true;

        } else {

            cout << "TA can only work on 2 projects at a time." << endl;

            return false;

        }

    }

    void display() override {

        Student::display();

        cout << "Assigned Students: ";

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

            cout << assignedStudents[i] << " ";

        }

        cout << endl;

        cout << "Current Projects: ";

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

            cout << projects[i] << " ";

        }

        cout << endl;

    }

};

class Professor : public User {

public:

    Professor(string name, string ID, string email, string password)

        : User(name, ID, email, password) {}

    void assignProjectToTA(TA& ta, string projectName) {

        if (ta.addProject(projectName)) {

            cout << "Project '" << projectName << "' assigned to TA " << ta.getName() << "." << endl;

        } else {

            cout << "Failed to assign project to TA " << ta.getName() << "." << endl;

        }

    }

    void display() override {

        User::display();

    }

};

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

    if (user->authenticate(password)) {

        cout << "Authentication successful. Performing action: " << action << endl;

        if (action == "access lab" && user->accessLab()) {

            cout << "Access granted to lab." << endl;

        } else {

            cout << "Action not permitted." << endl;

        }

    } else {

        cout << "Authentication failed." << endl;

    }

}

int main() {

    Student student1("Alice", "S001", "alice@example.com", "password123");

    TA ta1("Bob", "T001", "bob@example.com", "password456");

    Professor prof1("Dr. Smith", "P001", "smith@example.com", "password789");

    student1.display();

    ta1.display();

    prof1.display();

    prof1.assignProjectToTA(ta1, "Project A");

    prof1.assignProjectToTA(ta1, "Project B");

    prof1.assignProjectToTA(ta1, "Project C");

    ta1.display();

    authenticateAndPerformAction(&student1, "access lab", "password123");

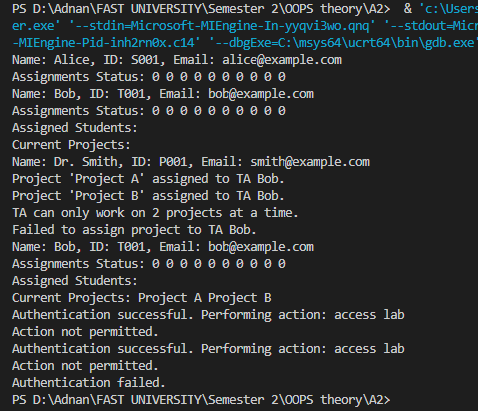
    authenticateAndPerformAction(&ta1, "access lab", "password456");

    authenticateAndPerformAction(&prof1, "access lab", "wrongpassword");

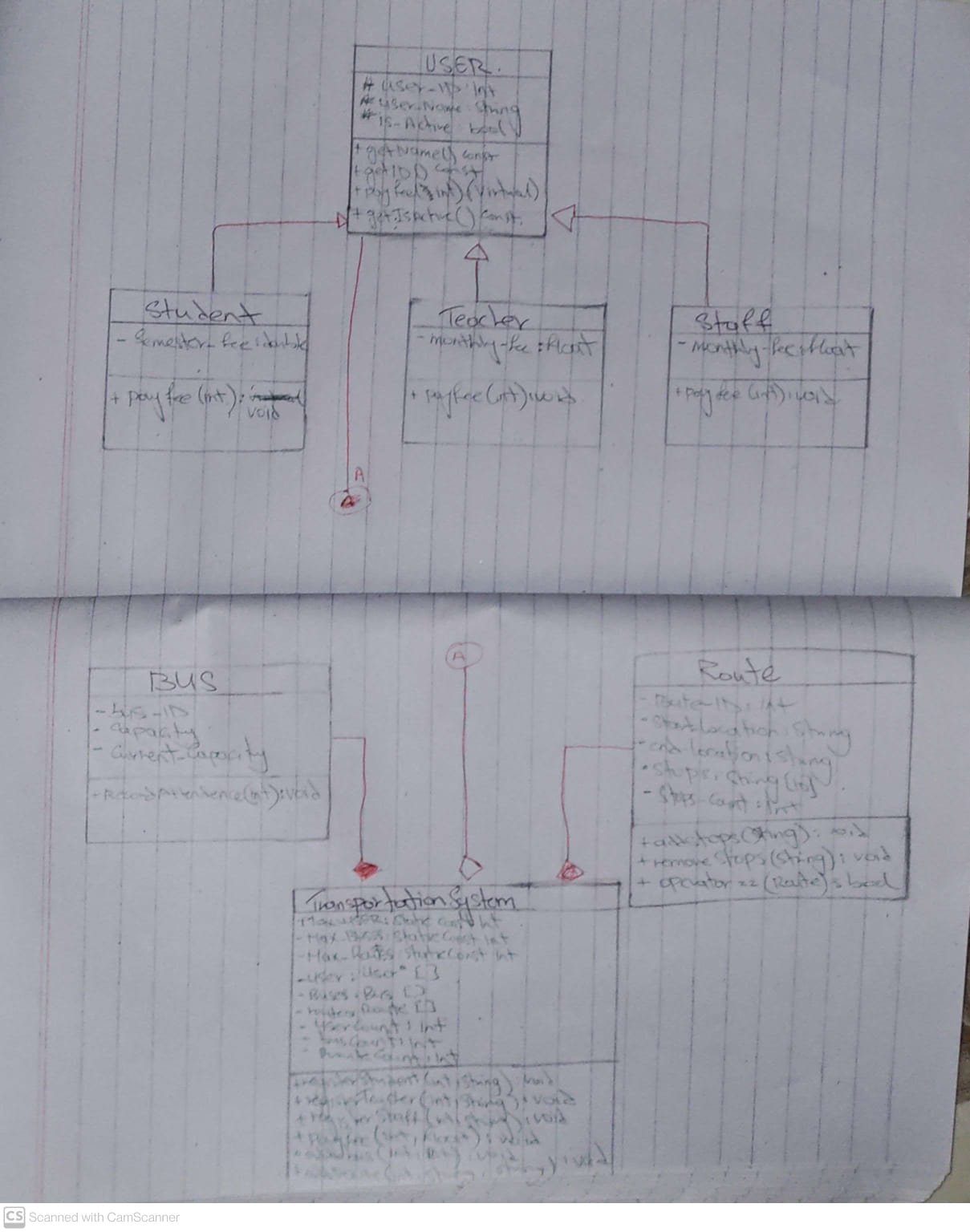
    return 0;

}

**Question 4 Output:**



**Question 4 Class Diagram:**

****