**OOP Assignment 2**

Question 1:

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

    string name;

    string rollno;

    bool feesPaid;

    Person(string n, string id, bool fees) {

        name = n;

        rollno = id;

        feesPaid = fees;

    }

    virtual void registerPerson() = 0;

    virtual void payFees() = 0;

};

class Student : public Person {

public:

    string assignedStop;

    Student(string n, string id, bool fees, string stop)

        : Person(n, id, fees), assignedStop(stop) {}

    void registerPerson() override {

        cout << "Student Registered: " << name << " with Roll No: " << rollno << endl;

    }

    void payFees() override {

        if (feesPaid) {

            cout << "Student " << name << " has already paid the fees for this semester!" << endl;

        } else {

            cout << "Student " << name << " pays semester fees." << endl;

            feesPaid = true;

        }

    }

};

class Teacher : public Person {

public:

    Teacher(string n, string id, bool fees)

        : Person(n, id, fees) {}

    void registerPerson() override {

        cout << "Teacher Registered: " << name << " with ID: " << rollno << endl;

    }

    void payFees() override {

        if (feesPaid) {

            cout << "Teacher " << name << " has already paid monthly fees!" << endl;

        } else {

            cout << "Teacher " << name << " pays monthly fees." << endl;

            feesPaid = true;

        }

    }

};

class Staff : public Person {

public:

    Staff(string n, string id, bool fees)

        : Person(n, id, fees) {}

    void registerPerson() override {

        cout << "Staff Registered: " << name << " with ID: " << rollno << endl;

    }

    void payFees() override {

        if (feesPaid) {

            cout << "Staff " << name << " has already paid monthly fees!" << endl;

        } else {

            cout << "Staff " << name << " pays monthly fees." << endl;

            feesPaid = true;

        }

    }

};

class Stop {

public:

    string stopName;

    int studentsAssigned[10];

    Stop(string name) {

        stopName = name;

    }

    Stop() {}

    void assignStudent(int rollno, int index) {

        if (index < 10) {

            studentsAssigned[index] = rollno;

            cout << "Student assigned to stop: " << stopName << endl;

        }

        else {

            cout << "Stop is full!" << endl;

        }

    }

    void removeStudent(int rollno) {

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

            if (studentsAssigned[i] == rollno) {

                studentsAssigned[i] = 0;

                cout << "Student removed from stop: " << stopName << endl;

                return;

            }

        }

        cout << "Student not found at this stop!" << endl;

    }

};

class Route {

public:

    string routeName;

    Stop stops[10];

    Route() {}

    Route(string name, Stop stops[10]) {

        routeName = name;

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

            this->stops[i] = stops[i];

        }

    }

    void addStop(string stopName, int index) {

        stops[index] = Stop(stopName);

        cout << "Stop " << stopName << " added to route " << routeName << endl;

    }

    void removeStop(int index) {

        stops[index] = Stop("");

        cout << "Stop removed from route!" << endl;

    }

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

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

            if (this->stops[i].stopName != other.stops[i].stopName) {

                return false;

            }

        }

        return true;

    }

};

class Point {

private:

    int pointNumber;

    int capacity;

    Route assignedRoute;

public:

    Point(int number, int cap, Route r) : pointNumber(number), capacity(cap), assignedRoute(r) {}

    void assignRoute(Route r) {

        assignedRoute = r;

    }

    void boardStudent(Person& p) {

        cout << p.name << " boarded!" << endl;

    }

};

int main() {

    Student s1("Alice", "24K-1111", false, "Gulshan");

    Teacher t1("Mr. Smith", "T-001", false);

    Staff st1("Mrs. Johnson", "S-001", false);

    s1.registerPerson();

    t1.registerPerson();

    st1.registerPerson();

    s1.payFees();

    t1.payFees();

    st1.payFees();

    Stop stop1("Gulshan");

    Stop stop2("Malir");

    stop1.assignStudent(1111, 0);

    stop2.assignStudent(2222, 0);

    Stop stops[] = {stop1, stop2};

    Route r1("University Road", stops);

    Point p1(1, 20, r1);

    p1.boardStudent(s1);

    Route r2("University Road", stops);

    if (r1 == r2) {

        cout << "Routes are the same!" << endl;

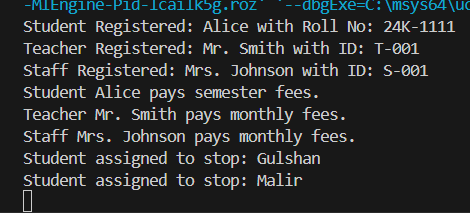
    } else {

        cout << "Routes are different!" << endl;

    }

    return 0;

}



Question 2:

#include <iostream>

#include <vector>

#include <string>

#include <cstdlib>

#include <ctime>

using namespace std;

class Ghost {

public:

    string name;

    int scareLevel;

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

    virtual void performHaunting() const = 0;

    virtual ~Ghost() {}

    friend ostream& operator<<(ostream& os, const Ghost& g) {

        os << g.name << " (Scare Level: " << g.scareLevel << ")";

        return os;

    }

};

class Poltergeist : public virtual Ghost {

public:

    Poltergeist(string n, int scare) : Ghost(n, scare) {}

    void performHaunting() const override {

        cout << name << " is moving objects around! Watch out!" << endl;

    }

};

class Banshee : public virtual Ghost {

public:

    Banshee(string n, int scare) : Ghost(n, scare) {}

    void performHaunting() const override {

        cout << name << " lets out a terrifying scream!" << endl;

    }

};

class ShadowGhost : public virtual Ghost {

public:

    ShadowGhost(string n, int scare) : Ghost(n, scare) {}

    void performHaunting() const override {

        cout << name << " is whispering creepily in the dark..." << endl;

    }

};

class ShadowPoltergeist : public Poltergeist, public ShadowGhost {

public:

    ShadowPoltergeist(string n, int scare)

        : Ghost(n, scare), Poltergeist(n, scare), ShadowGhost(n, scare) {}

    void performHaunting() const override {

        Poltergeist::performHaunting();

        ShadowGhost::performHaunting();

    }

};

class Visitor {

public:

    string name;

    int bravery;

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

    void reactToHaunting(int scareLevel) const {

        if (scareLevel < bravery - 2) {

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

        } else if (scareLevel > bravery + 2) {

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

        } else {

            cout << name << " has a shaky voice, but stands their ground!" << endl;

        }

    }

};

class HauntedHouse {

public:

    string houseName;

    vector<Ghost\*> ghosts;

    HauntedHouse(string name) : houseName(name) {}

    void addGhost(Ghost\* ghost) {

        ghosts.push\_back(ghost);

    }

    void runSimulation(const vector<Visitor>& visitors) const {

        cout << "Welcome to the haunted house: " << houseName << "!" << endl;

        for (const auto& ghost : ghosts) {

            cout << "\n" << \*ghost << " is haunting...\n";

            ghost->performHaunting();

            for (const auto& visitor : visitors) {

                visitor.reactToHaunting(ghost->scareLevel);

            }

        }

    }

};

void visit(const vector<Visitor>& visitors, HauntedHouse& house) {

    house.runSimulation(visitors);

}

int main() {

    srand(time(0));

    HauntedHouse house1("Spooky Mansion");

    house1.addGhost(new Poltergeist("Ghost1", rand() % 10 + 1));

    house1.addGhost(new Banshee("Ghost2", rand() % 10 + 1));

    house1.addGhost(new ShadowGhost("Ghost3", rand() % 10 + 1));

    HauntedHouse house2("Haunted Castle");

    house2.addGhost(new Poltergeist("Ghost4", rand() % 10 + 1));

    vector<Visitor> visitors = {

        Visitor("Alice", 3),

        Visitor("Bob", 6),

        Visitor("Charlie", 9)

    };

    cout << "\nVisiting Haunted House 1:\n";

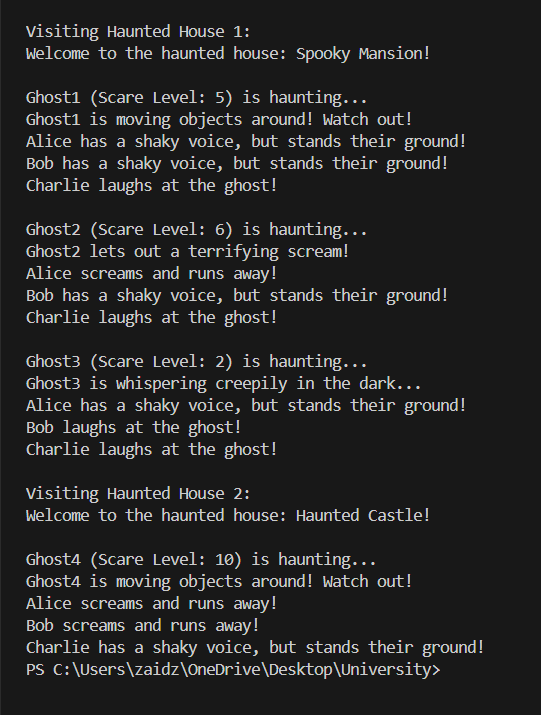
    visit(visitors, house1);

    cout << "\nVisiting Haunted House 2:\n";

    visit(visitors, house2);

    return 0;

}



Question 3:

#include <iostream>

#include <string>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

class Vehicle {

public:

    static int activeDeliveries;

    string vehicleID;

    double speed;

    double capacity;

    double energyEfficiency;

    Vehicle(string id, double spd, double cap, double eff)

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

        activeDeliveries++;

    }

    virtual ~Vehicle() {

        activeDeliveries--;

    }

    virtual void move() const = 0;

    virtual void command(const string& cmd) const {

        cout << "Command: " << cmd << " received by " << vehicleID << endl;

    }

    static int getActiveDeliveries() {

        return activeDeliveries;

    }

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

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

               (this->energyEfficiency == other.energyEfficiency);

    }

    virtual void calculateRoute() const {

        cout << "Calculating optimal route for " << vehicleID << "..." << endl;

    }

    virtual double estimateDeliveryTime(double distance) const {

        return distance / speed;

    }

};

int Vehicle::activeDeliveries = 0;

class RamzanDrone : public Vehicle {

public:

    RamzanDrone(string id) : Vehicle(id, 100, 10, 15) {}

    void move() const override {

        cout << vehicleID << " is flying at high speed to deliver Iftar!" << endl;

    }

    void command(const string& cmd) const override {

        if (cmd == "Deliver") {

            cout << vehicleID << " is delivering food for Iftar!" << endl;

        } else if (cmd == "DeliverUrgent") {

            cout << vehicleID << " is delivering an urgent Iftar package at high speed!" << endl;

        }

    }

};

class RamzanTimeShip : public Vehicle {

public:

    RamzanTimeShip(string id) : Vehicle(id, 30, 500, 10) {}

    void move() const override {

        cout << vehicleID << " is traveling through time to ensure historical accuracy!" << endl;

    }

    void command(const string& cmd) const override {

        if (cmd == "Deliver") {

            cout << vehicleID << " is delivering food but ensuring historical consistency!" << endl;

        } else if (cmd == "DeliverUrgent") {

            cout << vehicleID << " is prioritizing urgent historical delivery!" << endl;

        }

    }

};

class RamzanHyperPod : public Vehicle {

public:

    RamzanHyperPod(string id) : Vehicle(id, 200, 1000, 20) {}

    void move() const override {

        cout << vehicleID << " is racing underground to deliver bulk supplies!" << endl;

    }

    void command(const string& cmd) const override {

        if (cmd == "Deliver") {

            cout << vehicleID << " is delivering bulk supplies!" << endl;

        } else if (cmd == "DeliverUrgent") {

            cout << vehicleID << " is delivering bulk supplies with urgency!" << endl;

        }

    }

};

class ConflictResolutionSystem {

public:

    static Vehicle\* resolveConflict(Vehicle& v1, Vehicle& v2, double distance) {

        double time1 = v1.estimateDeliveryTime(distance);

        double time2 = v2.estimateDeliveryTime(distance);

        if (time1 < time2) {

            return &v1;

        } else {

            return &v2;

        }

    }

};

int main() {

    RamzanDrone drone1("Drone1");

    RamzanTimeShip timeShip1("TimeShip1");

    RamzanHyperPod hyperPod1("HyperPod1");

    drone1.command("Deliver");

    timeShip1.command("DeliverUrgent");

    hyperPod1.command("Deliver");

    double deliveryDistance = 100;

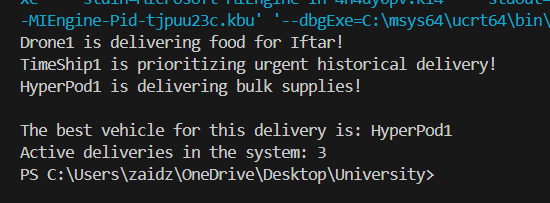
    Vehicle\* bestVehicle = ConflictResolutionSystem::resolveConflict(drone1, hyperPod1, deliveryDistance);

    cout << "\nThe best vehicle for this delivery is: " << bestVehicle->vehicleID << endl;

    cout << "Active deliveries in the system: " << Vehicle::getActiveDeliveries() << endl;

    return 0;

}



Question 4:

#include <iostream>

#include <vector>

#include <string>

#include <unordered\_map>

using namespace std;

unsigned long hashPassword(const string& password) {

    unsigned long hash = 5381;

    for (char c : password) {

        hash = (hash \* 33) + c;

    }

    return hash;

}

enum Permission {

    STUDENT = 1,

    TA = 2,

    PROFESSOR = 4

};

class User {

protected:

    string name;

    string userID;

    string email;

    string hashed\_password;

    int permissions;

public:

    User(string n, string id, string em, string password, int perm)

        : name(n), userID(id), email(em), permissions(perm) {

        hashed\_password = to\_string(hashPassword(password));

    }

    bool authenticate(string password) {

        return hashed\_password == to\_string(hashPassword(password));

    }

    virtual void display() const {

        cout << "Name: " << name << "\nID: " << userID << "\nEmail: " << email << endl;

    }

    virtual void accessLab() {

        cout << "User has access to the lab." << endl;

    }

    int getPermissions() const {

        return permissions;

    }

    string getUserID() const {

        return userID;

    }

};

class Student : public User {

protected:

    vector<int> assignments;

public:

    Student(string n, string id, string em, string password)

        : User(n, id, em, password, STUDENT) {}

    void display() const override {

        User::display();

        cout << "Role: Student" << endl;

    }

    void addAssignment() {

        assignments.push\_back(0);

    }

    void submitAssignment(int assignmentID) {

        if (assignmentID >= 0 && assignmentID < assignments.size()) {

            assignments[assignmentID] = 1;

            cout << "Assignment " << assignmentID << " submitted." << endl;

        } else {

            cout << "Invalid assignment ID." << endl;

        }

    }

    bool isAssignmentSubmitted(int assignmentID) const {

        if (assignmentID >= 0 && assignmentID < assignments.size()) {

            return assignments[assignmentID] == 1;

        }

        return false;

    }

};

class tAssistant : public Student {

private:

    vector<Student\*> assignedStudents;

    vector<string> projects;

public:

    tAssistant(string n, string id, string em, string password)

        : Student(n, id, em, password) {

        permissions = STUDENT | TA;

    }

    void display() const override {

        Student::display();

        cout << "Role: Teaching Assistant" << endl;

    }

    void assignStudent(Student\* student) {

        if (assignedStudents.size() < 10) {

            assignedStudents.push\_back(student);

            cout << "Assigned Student " << student->getUserID() << " to TA." << endl;

        } else {

            cout << "TA can only manage up to 10 students!" << endl;

        }

    }

    void assignProject(string project) {

        if (projects.size() >= 2) {

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

            return;

        }

        projects.push\_back(project);

        cout << "Assigned project: " << project << endl;

    }

    void viewProjects() const {

        cout << "Projects TA is working on:" << endl;

        for (const string& project : projects) {

            cout << project << endl;

        }

    }

};

class Professor : public User {

private:

    vector<tAssistant\*> assignedTAs;

    vector<string> researchProjects;

public:

    Professor(string n, string id, string em, string password)

        : User(n, id, em, password, PROFESSOR) {}

    void display() const override {

        User::display();

        cout << "Role: Professor" << endl;

    }

    void assignProjectToTA(tAssistant\* ta, string project) {

        ta->assignProject(project);

    }

    void collaborateWithTA(tAssistant\* ta, string project) {

        cout << "Professor collaborating with TA on project: " << project << endl;

        ta->assignProject(project);

    }

};

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

    if (user->authenticate(password)) {

        cout << "Authenticated successfully!" << endl;

        if (action == "submit assignment" && (user->getPermissions() & STUDENT)) {

            Student\* student = dynamic\_cast<Student\*>(user);

            if (student) {

                student->submitAssignment(0);

            }

        } else if (action == "assign project" && (user->getPermissions() & PROFESSOR)) {

            Professor\* professor = dynamic\_cast<Professor\*>(user);

            if (professor) {

                cout << "Professor assigning project..." << endl;

                tAssistant\* existingTA = new tAssistant("Jane Smith", "TA1", "jane@example.com", "password123");

                professor->assignProjectToTA(existingTA, "AI Research");

                existingTA->viewProjects();

                delete existingTA;

            }

        } else {

            cout << "Action not allowed for this user role!" << endl;

        }

    } else {

        cout << "Authentication failed!" << endl;

    }

}

int main() {

    Student s1("John Doe", "S1", "john@example.com", "password123");

    tAssistant ta1("Jane Smith", "TA1", "jane@example.com", "password123");

    Professor p1("Dr. Brown", "P1", "drbrown@example.com", "profpass");

    s1.display();

    ta1.display();

    p1.display();

    authenticateAndPerformAction(&s1, "submit assignment", "password123");

    authenticateAndPerformAction(&p1, "assign project", "profpass");

    p1.assignProjectToTA(&ta1, "Machine Learning Research");

    ta1.viewProjects();

    return 0;

}

