1.

#include <iostream>

#include <vector>

#include <string>

#include <utility>

using namespace std;

class User;

class Student;

class Teacher;

class Staff;

class Stop {

    int stopID;

    string stopName;

    int routeID;

public:

    Stop(int id, string name, int route) : stopID(id), stopName(name), routeID(route) {}

    void getStopDetails() const {

        cout << "Stop ID: " << stopID << ", Name: " << stopName << ", Route: " << routeID << endl;

    }

    int getStopID(){

        return stopID;

    }

};

class AttendanceSystem {

    vector<pair<int, string>> attendanceRecords;

public:

    void recordAttendance(int userID, const string& timestamp) {

        attendanceRecords.push\_back(make\_pair(userID, timestamp));

        cout << "Attendance recorded for user ID: " << userID << " at " << timestamp << endl;

    }

    void displayAttendance() const {

        for (const auto& record : attendanceRecords) {

            cout << "User  ID: " << record.first << ", Time: " << record.second << endl;

        }

    }

};

class User {

protected:

    int userID;

    string name;

    float balance;

    bool isActive;

    int assignedStop;

public:

    User(int id, string n, float bal, bool active) : userID(id), name(n), balance(bal), isActive(active), assignedStop(0) {}

    virtual ~User () {}

    virtual void payFees(float amount) = 0;

    void assignStop(int stop) {

        assignedStop = stop;

        cout << "Stop " << stop << " assigned to user " << name << endl;

    }

    void displayUser() const {

        cout << "User  ID: " << userID << ", Name: " << name

             << ", Balance: $" << balance << ", Active: " << (isActive ? "Yes" : "No")

             << ", Assigned Stop: " << assignedStop << endl;

    }

    int getUserID(){

        return userID;

    }

    bool isUserActive(){

        return isActive;

    }

};

class Student : public User {

public:

    Student(int id, string n) : User(id, n, 0.0, false) {}

    void payFees(float amount) override {

        balance += amount;

        isActive = balance >= 0;

        cout << "Student fees paid. Balance: $" << balance << endl;

    }

};

class Teacher : public User {

public:

    Teacher(int id, string n) : User(id, n, 0.0, false) {}

    void payFees(float amount) override {

        balance += amount;

        isActive = balance >= 0;

        cout << "Teacher fees paid. Balance: $" << balance << endl;

    }

};

class Staff : public User {

public:

    Staff(int id, string n) : User(id, n, 0.0, false) {}

    void payFees(float amount) override {

        balance += amount;

        isActive = balance >= 0;

        cout << "Staff fees paid. Balance: $" << balance << endl;

    }

};

class BusRoute {

    int routeID;

    string routeName;

    vector<Stop> stops;

public:

    BusRoute(int id, string name) : routeID(id), routeName(name) {}

    void addStop(const Stop& stop) { stops.push\_back(stop); }

    void displayRoute() const {

        cout << "Route " << routeID << ": " << routeName << endl;

        for (const auto& stop : stops) {

            stop.getStopDetails();

        }

    }

    bool operator==(const BusRoute& other) {

        return routeID == other.routeID;

    }

};

class TransportationSystem {

    vector<User\*> users;

    vector<BusRoute> routes;

    AttendanceSystem attendance;

    static int userCount;

    User\* findUserByID(int userID) {

        for (auto\* user : users) {

            if (user->getUserID() == userID) {

                return user;

            }

        }

        return nullptr;

    }

public:

    void registerUser(User\* user) {

        users.push\_back(user);

        userCount++;

    }

    void assignStop(int userID, int stopID) {

        User\* user = findUserByID(userID);

        if (user) {

            user->assignStop(stopID);

        }

    }

    void recordAttendance(int userID) {

        User\* user = findUserByID(userID);

        if (user) {

            attendance.recordAttendance(userID, "current\_time");

        }

    }

    void payFees(int userID, float amount) {

        User\* user = findUserByID(userID);

        if (user) {

            user->payFees(amount);

        }

    }

    void displayAllUsers() const {

        for (const auto\* user : users) {

            user->displayUser();

        }

    }

    void displayActiveUsers() const {

        cout << "Total Users: " << userCount << endl;

        for (auto\* user : users) {

            if (user->isUserActive()) {

                user->displayUser();

            }

        }

    }

    void addBusRoute(const BusRoute& route) { routes.push\_back(route); }

    void displayAllRoutes() const {

        for (const auto& route : routes) {

            route.displayRoute();

        }

    }

};

int TransportationSystem::userCount = 0;

int main() {

    TransportationSystem system;

    Student\* s1 = new Student(1, "Alice");

    Teacher\* t1 = new Teacher(2, "Bob");

    Staff\* st1 = new Staff(3, "Charlie");

    system.registerUser(s1);

    system.registerUser(t1);

    system.registerUser(st1);

    system.payFees(1, 100);

    system.payFees(2, 150);

    system.payFees(3, 200);

    system.assignStop(1, 10);

    system.assignStop(2, 20);

    system.assignStop(3, 30);

    system.recordAttendance(1);

    system.recordAttendance(2);

    system.recordAttendance(3);

    BusRoute r1(1, "North Loop");

    r1.addStop(Stop(10, "Stop A", 1));

    r1.addStop(Stop(11, "Stop B", 1));

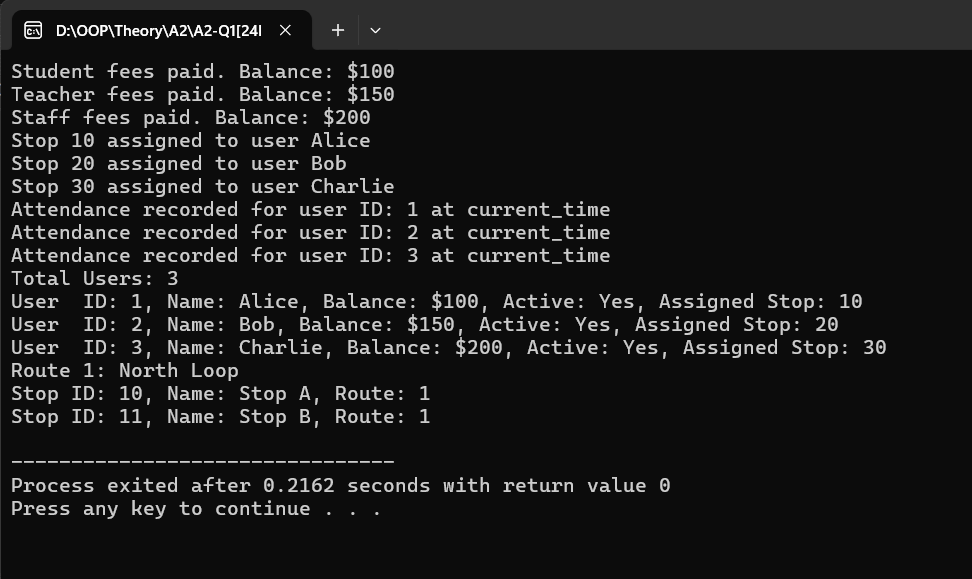
    system.addBusRoute(r1);

    system.displayActiveUsers();

    system.displayAllRoutes();

}

Output



2.

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Ghost {

protected:

    string workerName;

    int scareLevel;

public:

    Ghost(string w) : workerName(w), scareLevel(rand() % 9 + 1) {}

    virtual void haunt()

    {

        cout << workerName << " haunts with scare level " << scareLevel << endl;

    }

    friend ostream& operator<<(ostream& os, Ghost& ghost)

    {

        os << "Ghost played by: " << ghost.workerName << ", Scare Level: " << ghost.scareLevel;

        return os;

    }

    int getScareLevel()

    {

        return scareLevel;

    }

};

class Poltergeist : public Ghost {

public:

    Poltergeist(string w) : Ghost(w) {}

    void haunt() override

    {

        cout << workerName << "poltergeist moves" << endl;

    }

};

class Banshee : public Ghost {

public:

    Banshee(string w) : Ghost(w) {}

    void haunt() override

    {

        cout << workerName << "banshee screams" << endl;

    }

};

class ShadowGhost : public Ghost {

public:

    ShadowGhost(string w) : Ghost(w) {}

    void haunt() override

    {

        cout << workerName << "shadowghost whispers" << endl;

    }

};

class Visitor {

    string name;

    int braveryLevel;

public:

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

    void react(int scare)

    {

        if (scare < braveryLevel - 3)

        {

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

        }

        else if (scare > braveryLevel + 3) {

            cout << name << " screams and runs awayy" << endl;

        }

        else {

            cout << name << "shaky voice" << endl;

        }

    }

};

class HauntedHouse {

    string houseName;

    vector<Ghost\*> ghosts;

public:

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

    void addGhost(Ghost\* ghost)

    {

        ghosts.push\_back(ghost);

    }

    void hostVisitors(vector<Visitor>& visitors)

    {

        cout << "\nVisitors enter " << houseName << endl;

        for (auto& ghost : ghosts)

        {

            ghost->haunt();

        }

        for (auto& visitor : visitors)

        {

            visitor.react(ghosts[rand() % ghosts.size() + 0]->getScareLevel());

        }

    }

};

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

{

    house.hostVisitors(visitors);

}

int main() {

    HauntedHouse house1("Mansion");

    house1.addGhost(new Poltergeist("John"));

    house1.addGhost(new Banshee("Alice"));

    house1.addGhost(new ShadowGhost("Eve"));

    HauntedHouse house2("Dark Forest Cabin");

    house2.addGhost(new ShadowGhost("Mike"));

    house2.addGhost(new Poltergeist("Sarah"));

    vector<Visitor> visitors = {

        Visitor("Tom", 2),

        Visitor("Jane", 6),

        Visitor("Mark", 9)

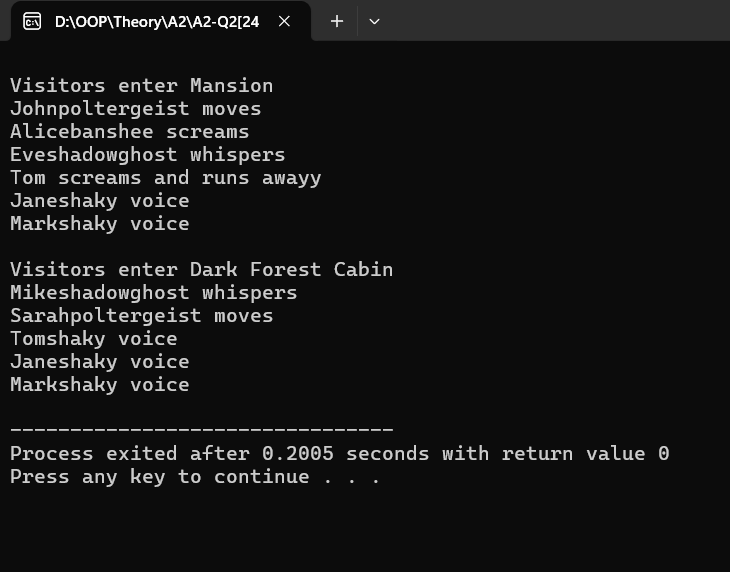
    };

    visit(visitors, house1);

    visit(visitors, house2);

}

Output



3.

#include <iostream>

using namespace std;

class Vehicle {

protected:

    int id;

    static int deliveryn;

    int eff;

public:

    Vehicle(int i, int e) : id(i), eff(e) {

        deliveryn++;

    }

    virtual ~Vehicle() {

        deliveryn--;

    }

    virtual void calculateroute(string destination) = 0;

    virtual int deliveryTime(int distance, int speed) = 0;

    virtual void move() = 0;

    virtual void command(string cmd, int packageID) {

        cout << "Base vehicle command for " << cmd << " on package " << packageID << endl;

    }

    static int getnum() {

        return deliveryn;

    }

    friend bool operator==(Vehicle &v1, Vehicle &v2) {

        return (v1.eff == v2.eff);

    }

    friend bool resolveConflict(Vehicle& v1, Vehicle& v2);

};

int Vehicle::deliveryn = 0;

class RamzanDrone : public Vehicle {

public:

    RamzanDrone(int i, int e) : Vehicle(i, e) {}

    void calculateroute(string destination) override {

        cout << "Finding route to " << destination << endl;

    }

    int deliveryTime(int distance, int speed) override {

        return (distance / speed);

    }

    void command(string cmd, int packageID) override {

        Vehicle::command(cmd, packageID);

        cout << "Drone " << id << " executing command: " << cmd << " for package " << packageID << endl;

    }

    void command(string cmd, int packageID, string urgencyLevel) {

        if (urgencyLevel == "urgent") {

            cout << "Drone " << id << " urgently delivering the package " << packageID << endl;

        } else {

            cout << "Drone " << id << " delivering at normal speed " << packageID << endl;

        }

    }

    void move() override {

        cout << "Drone " << id << " flying to destination." << endl;

    }

};

class RamzanTimeShip : public Vehicle {

public:

    RamzanTimeShip(int i, int e) : Vehicle(i, e) {}

    void calculateroute(string destination) override {

        cout << "Finding time to " << destination << endl;

    }

    int deliveryTime(int distance, int speed) override {

        return (distance / speed);

    }

    void command(string cmd, int packageID) override {

        Vehicle::command(cmd, packageID);

        cout << "TimeShip " << id << " executing command: " << cmd << " for package " << packageID << endl;

    }

    void command(string cmd, int packageID, string urgencyLevel) {

        if (urgencyLevel == "urgent") {

            cout << "TimeShip " << id << " urgently delivering through history " << packageID << endl;

        } else {

            cout << "TimeShip " << id << " normally delivering through history " << packageID << endl;

        }

    }

    void move() override {

        cout << "TimeShip " << id << " traveling through time." << endl;

    }

};

class RamzanHyperPod : public Vehicle {

public:

    RamzanHyperPod(int i, int e) : Vehicle(i, e) {}

    void calculateroute(string destination) override {

        cout << "Finding underground route to " << destination << endl;

    }

    int deliveryTime(int distance, int speed) override {

        return (distance / speed);

    }

    void command(string cmd, int packageID) override {

        Vehicle::command(cmd, packageID);

        cout << "HyperPod " << id << " executing command: " << cmd << " for package " << packageID << endl;

    }

    void move() override {

        cout << "HyperPod " << id << " speeding through underground network." << endl;

    }

};

bool resolveConflict(Vehicle& v1, Vehicle& v2) {

    cout << "Resolving conflict between " << v1.id << " and " << v2.id << endl;

    return v1.eff < v2.eff;

}

int main() {

    RamzanDrone drone(101, 20);

    RamzanTimeShip timeship(102, 30);

    RamzanHyperPod hyperpod(103, 40);

    drone.calculateroute("Shahre Faisal");

    timeship.calculateroute("1950");

    hyperpod.calculateroute("Underground Tunnel");

    cout << "Drone Delivery Time: " << drone.deliveryTime(4, 20) << " hours" << endl;

    cout << "TimeShip Delivery Time: " << timeship.deliveryTime(3, 25) << " hours" << endl;

    cout << "HyperPod Delivery Time: " << hyperpod.deliveryTime(2, 30) << " hours" << endl;

    drone.command("Deliver", 101);

    timeship.command("Deliver", 102);

    hyperpod.command("Deliver", 103);

    drone.command("Deliver", 101, "urgent");

    timeship.command("Deliver", 102, "urgent");

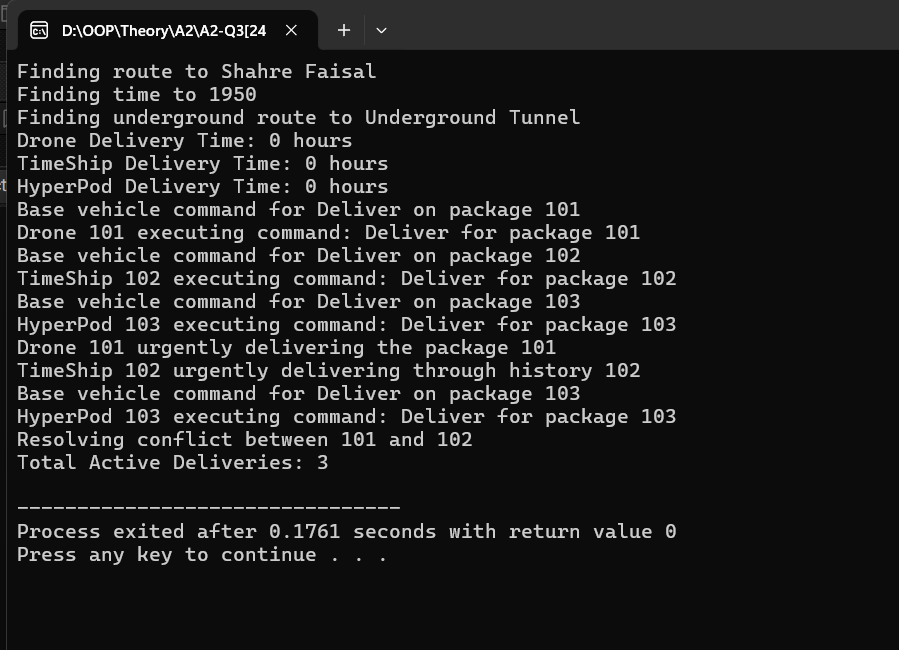
    hyperpod.command("Deliver", 103);

    resolveConflict(drone, timeship);

    cout << "Total Active Deliveries: " << Vehicle::getnum() << endl;

}

Output



4.

#include <iostream>

#include <vector>

using namespace std;

int passwordgen(string& pass) {

    int hash = 5381;

    for (char c : pass) hash = hash \* 33 + c;

    return hash;

}

class User {

protected:

    string name, id, email;

    vector<string> permissions;

    int hashedPass;

public:

    User(string n, string i, string e, string p, vector<string> perms)

        : name(n), id(i), email(e), permissions(perms) {

        hashedPass = passwordgen(p);

    }

    bool authenticate(string& p) {

        return passwordgen(p) == hashedPass;

    }

    virtual void display() {

        cout << "Name: " << name << " | ID: " << id << " | Email: " << email << endl;

    }

    bool hasPermission(const string& action) {

        for (string& perm : permissions)

            if (perm == action) return true;

        return false;

    }

    void accessLab() {

        if (hasPermission("full\_lab\_access"))

            cout << "Lab Access: Full Access Granted\n";

        else if (hasPermission("manage\_students"))

            cout << "Lab Access: Partial Access Granted\n";

        else

            cout << "Lab Access: Access Denied\n";

    }

};

class Student : public User {

protected:

    int assignments[5] = {0};

public:

    Student(string n, string i, string e, string p)

        : User(n, i, e, p, {"submit\_assignment"}) {}

    void submitAssignment(int index) {

        if (index >= 0 && index < 5) assignments[index] = 1;

    }

    void display() override {

        User::display();

        cout << "Assignments:\n";

        for (int i = 0; i < 5; ++i)

            cout << "  Assignment " << (i + 1) << ": " << (assignments[i] ? "Done" : "Not Done") << endl;

    }

};

class TA : public Student {

    vector<Student\*> assigned;

    vector<string> projects;

public:

    TA(string n, string i, string e, string p)

        : Student(n, i, e, p) {

        permissions.push\_back("view\_projects");

        permissions.push\_back("manage\_students");

    }

    void display() override {

        User::display();

        cout << "Projects:\n";

        for (size\_t i = 0; i < projects.size(); ++i)

            cout << "  Project " << (i + 1) << ": " << projects[i] << endl;

    }

    void addProject(string& proj) {

        if (projects.size() < 2) {

            projects.push\_back(proj);

            cout << "Project '" << proj << "' added.\n";

        } else {

            cout << "Max project limit reached.\n";

        }

    }

    void assignStudent(Student\* s) {

        if (hasPermission("manage\_students")) {

            if (assigned.size() < 10) {

                assigned.push\_back(s);

                cout << "Student assigned to TA.\n";

            } else {

                cout << "TA has max number of students.\n";

            }

        } else {

            cout << "Permission denied to assign students.\n";

        }

    }

};

class Professor : public User {

public:

    Professor(string n, string i, string e, string p)

        : User(n, i, e, p, {"assign\_projects", "full\_lab\_access"}) {}

    void display() override {

        User::display();

    }

    void assignProjectToTA(TA& ta, string& proj) {

        if (hasPermission("assign\_projects")) {

            ta.addProject(proj);

        } else {

            cout << "Permission denied to assign project.\n";

        }

    }

};

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

    if (user->hasPermission(action)) {

        cout << "Action '" << action << "' performed.\n";

    } else {

        cout << "Permission denied for '" << action << "'.\n";

    }

}

int main() {

    Student s("Alice", "S1", "alice@uni.edu", "pass123");

    TA t("Bob", "T1", "bob@uni.edu", "tapass");

    Professor p("Dr. Ray", "P1", "ray@uni.edu", "propass");

    s.display();

    t.display();

    p.display();

    cout << "\nAuthentication Test:\n";

    string input;

    cout << "Enter password for Alice: ";

    cin >> input;

    if (s.authenticate(input)) cout << "Access Granted\n";

    else cout << "Access Denied\n";

    cout << "\nLab Access:\n";

    s.accessLab();

    t.accessLab();

    p.accessLab();

    cout << "\nAssigning Project to TA:\n";

    string proj = "AI Lab";

    p.assignProjectToTA(t, proj);

    cout << "\nSubmitting Assignment:\n";

    s.submitAssignment(2);

    s.display();

    cout << "\nTA Assigns Student:\n";

    t.assignStudent(&s);

    cout << "\nPerforming Actions:\n";

    string action1 = "submit\_assignment";

    string action2 = "manage\_students";

    string action3 = "assign\_projects";

    authenticateAndPerformAction(&s, action1);

    authenticateAndPerformAction(&t, action2);

    authenticateAndPerformAction(&p, action3);

}

Output

