**Assignment 2**

**24k-0880**

**Question 1:**

**Code:**

#include <iostream>

using namespace std;

class Person {

protected:

int id;

string pickup, dropoff;

bool active;

bool present;

public:

Person(int \_id, string PU, string DO) : id(\_id), pickup(PU), dropoff(DO), active(false), present(false) {}

virtual void showDetails() const {

cout << "ID: " << id << "\nPickup: " << pickup << "\nDropoff: " << dropoff;

cout << "\nStatus: " << (active ? "Active" : "Inactive") << "\nAttendance: " << (present ? "Present" : "Absent") << endl;

}

virtual void payFees() = 0;

void tapCard(){

if (active){

present = true;

cout << "Attendance marked for ID: " << id << ".\n";

} else{

cout << "Payment required! Cannot mark attendance.\n";

}

}

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

return id == other.id;

}

};

class Student : public Person {

public:

Student(int \_id, string PU, string DO) : Person(\_id, PU, DO) {}

void payFees() override {

if (!active) {

active = true;

cout << "Student " << id << " has paid semester fees.\n";

} else {

cout << "Fees already paid for Student " << id << ".\n";

}

}

void showDetails() const override {

cout << "[Student] ";

Person::showDetails();

}

};

class Teacher : public Person {

public:

Teacher(int \_id, string PU, string DO) : Person(\_id, PU, DO) {}

void payFees() override {

if (!active) {

active = true;

cout << "Teacher " << id << " has paid monthly transport fees.\n";

} else {

cout << "Fees already paid for Teacher " << id << ".\n";

}

}

void showDetails() const override {

cout << "[Teacher] ";

Person::showDetails();

}

};

class StaffMember : public Person {

public:

StaffMember(int \_id, string PU, string DO) : Person(\_id, PU, DO) {}

void payFees() override {

if (!active) {

active = true;

cout << "Staff Member " << id << " has paid monthly transport fees.\n";

} else {

cout << "Fees already paid for Staff Member " << id << ".\n";

}

}

void showDetails() const override {

cout << "[Staff] ";

Person::showDetails();

}

};

class Transport {

private:

int busNo;

string\* stops;

int totalStops;

Person\*\* passengers;

int capacity, passengerCount = 0;

public:

Transport(int no, string\* stopList, int totalStops, int cap) : busNo(no), totalStops(totalStops), capacity(cap) {

stops = new string[totalStops];

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

stops[i] = stopList[i];

}

passengers = new Person\*[capacity];

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

passengers[i] = nullptr;

}

}

void registerPerson(Person\* p) {

if (passengerCount < capacity) {

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

if (p->pickup == stops[i]) {

passengers[passengerCount] = p;

passengerCount++;

cout << "Person ID " << p->id << " registered for bus " << busNo << ".\n";

return;

}

}

cout << "Error: Pickup location does not match bus stops.\n";

} else {

cout << "Error: Bus " << busNo << " is full.\n";

}

}

void showTransportDetails() const {

cout << "\nBus Number: " << busNo << "\nStops: ";

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

cout << stops[i] << (i == totalStops - 1 ? "" : ", ");

}

cout << "\nCapacity: " << capacity << "\nRegistered Passengers: " << passengerCount << endl;

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

if (passengers[i] != nullptr) {

passengers[i]->showDetails();

cout << "-----------------\n";

}

}

}

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

if (totalStops != other.totalStops) return false;

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

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

}

return true;

}

~Transport() {

delete[] stops;

delete[] passengers;

}

};

int main() {

cout << "A2 - Q1 - 24K-0880 - Muhammad Umer Karim" << endl;

string stopsList[] = {"Stop A", "Stop B", "Stop C", "Stop D"};

Transport bus1(101, stopsList, 4, 5);

Student s1(1, "Stop A", "Stop C");

Student s2(2, "Stop B", "Stop D");

Teacher t1(3, "Stop A", "Stop B");

StaffMember sm1(4, "Stop C", "Stop D");

bus1.registerPerson(&s1);

bus1.registerPerson(&s2);

bus1.registerPerson(&t1);

bus1.registerPerson(&sm1);

s1.payFees();

s2.payFees();

t1.payFees();

sm1.payFees();

s1.tapCard();

t1.tapCard();

bus1.showTransportDetails();

Transport bus2(102, stopsList, 4, 5);

if (bus1 == bus2) {

cout << "Both buses have the same route.\n";

} else {

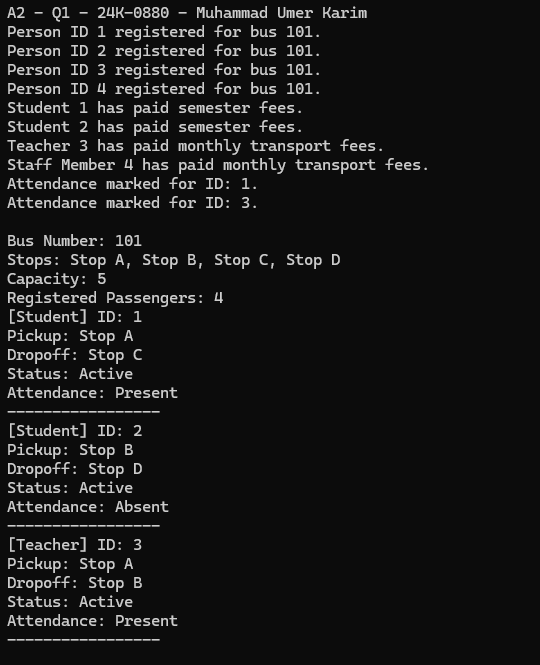
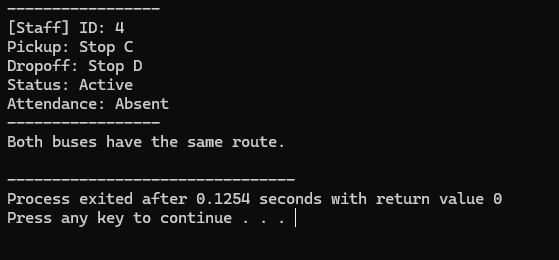
cout << "Buses have different routes.\n";

}

return 0;

}

**Output:**

**** ****

**Question 2:**

**Code:**

#include <iostream>

#include <string>

using namespace std;

class Ghost {

public:

string name;

int scareLevel;

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

virtual void haunt() const = 0;

int getScareLevel() const { return scareLevel; }

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

os << "Ghost: " << g.name << ", Scare Level: " << g.scareLevel;

return os;

}

virtual ~Ghost() {}

};

class Poltergeist : public Ghost {

public:

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

void haunt() const override {

cout << name << " moves objects violently!" << endl;

}

Poltergeist operator+(const Ghost& other) {

return Poltergeist("Hybrid of " + name + " & " + other.name, scareLevel + other.scareLevel);

}

};

class Banshee : public Ghost {

public:

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

void haunt() const override {

cout << name << " screams loudly!" << endl;

}

};

class ShadowGhost : public Ghost {

public:

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

void haunt() const override {

cout << name << " whispers creepily..." << endl;

}

};

class ShadowPoltergeist : public Ghost {

ShadowGhost shadowGhost;

Poltergeist poltergeist;

public:

ShadowPoltergeist(string n, int s1, int s2)

: Ghost(n, s1 + s2), shadowGhost(n + "\_shadow", s1), poltergeist(n + "\_poltergeist", s2) {}

void haunt() const override {

shadowGhost.haunt();

poltergeist.haunt();

}

};

class Visitor {

string name;

int bravery;

public:

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

string getName() const { return name; }

int getBravery() const { return bravery; }

void reactTo(int scare) const {

if (scare < bravery - 2)

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

else if (scare > bravery + 2)

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

else

cout << name << " is shaking with fear..." << endl;

}

};

class HauntedHouse {

string houseName;

Ghost\* ghosts[10];

int count;

public:

HauntedHouse(string name) : houseName(name), count(0) {}

void addGhost(Ghost\* g) {

if (count < 10) ghosts[count++] = g;

}

friend void visit(Visitor\* v[], int vCount, HauntedHouse& h) {

cout << "\n--- Visiting " << h.houseName << " ---\n";

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

cout << v[i]->getName() << " enters...\n";

for (int j = 0; j < h.count; ++j) {

h.ghosts[j]->haunt();

v[i]->reactTo(h.ghosts[j]->getScareLevel());

}

}

}

~HauntedHouse() {

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

delete ghosts[i];

}

};

int main() {

cout << "A2 - Q2 - 24K-0880 - Muhammad Umer Karim" << endl;

HauntedHouse hh1("Ghost House");

hh1.addGhost(new Poltergeist("Polty", 5));

hh1.addGhost(new Banshee("Bansha", 7));

hh1.addGhost(new ShadowGhost("Shadowy", 6));

Visitor\* visitors[3] = {

new Visitor("Alice", 3),

new Visitor("Bob", 6),

new Visitor("Charlie", 9)

};

visit(visitors, 3, hh1);

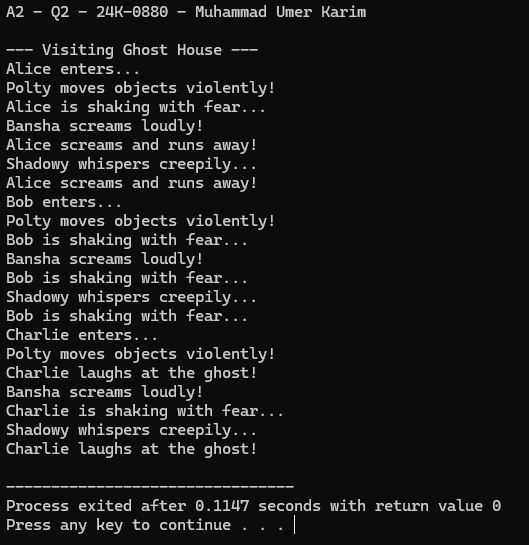
for (int i = 0; i < 3; ++i)

delete visitors[i];

return 0;

}

**Output:**



**Question 3:**

**Code:**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

class Vehicle {

protected:

string vehicleID;

float speed;

float capacity;

float energyEfficiency;

static int totalDeliveries;

public:

Vehicle(string id, float s, float c, float ee)

: vehicleID(id), speed(s), capacity(c), energyEfficiency(ee) {

totalDeliveries++;

}

virtual void calculateRoute() const = 0;

float calculateDeliveryTime(float distance) const {

return distance / speed;

}

static int getTotalDeliveries() {

return totalDeliveries;

}

virtual void move() const = 0;

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

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

}

virtual ~Vehicle() {}

};

int Vehicle::totalDeliveries = 0;

class RamzanTimeShip : public Vehicle {

public:

RamzanTimeShip(string id, float s, float c, float ee)

: Vehicle(id, s, c, ee) {}

void calculateRoute() const override {

cout << "Time-Ship is verifying historical accuracy before proceeding...\n";

}

void move() const override {

cout << "RamzanTimeShip moves through time to ensure historical accuracy.\n";

}

void command(const string& cmd, int urgencyLevel = 0) {

if (cmd == "Deliver" && urgencyLevel == 1) {

cout << "Time-Ship treats urgent deliveries as historically sensitive.\n";

}

else {

cout << "Time-Ship delivers on time with historical precision.\n";

}

}

};

class RamzanDrone : public Vehicle {

public:

RamzanDrone(string id, float s, float c, float ee)

: Vehicle(id, s, c, ee) {}

void calculateRoute() const override {

cout << "RamzanDrone calculates an aerial route for high-speed delivery.\n";

}

void move() const override {

cout << "RamzanDrone flies through the sky to deliver quickly.\n";

}

void command(const string& cmd, int urgencyLevel = 0) {

if (cmd == "Deliver" && urgencyLevel == 1) {

cout << "RamzanDrone activates high-speed mode for urgent delivery.\n";

}

else {

cout << "RamzanDrone delivers efficiently with low energy consumption.\n";

}

}

};

class RamzanHyperPod : public Vehicle {

public:

RamzanHyperPod(string id, float s, float c, float ee)

: Vehicle(id, s, c, ee) {}

void calculateRoute() const override {

cout << "RamzanHyperPod navigates through underground tunnels for efficient bulk delivery.\n";

}

void move() const override {

cout << "RamzanHyperPod travels through high-speed underground networks.\n";

}

void command(const string& cmd, int urgencyLevel = 0) {

if (cmd == "Deliver" && urgencyLevel == 1) {

cout << "RamzanHyperPod accelerates through the underground tunnels.\n";

}

else {

cout << "RamzanHyperPod delivers large quantities efficiently.\n";

}

}

};

class ConflictResolution {

public:

static void resolveConflict(const Vehicle& v1, const Vehicle& v2) {

if (v1 == v2) {

cout << "The vehicles are equally efficient.\n";

} else {

if (v1.getTotalDeliveries() > v2.getTotalDeliveries()) {

cout << "Vehicle 1 has a better delivery track record.\n";

} else {

cout << "Vehicle 2 has a better delivery track record.\n";

}

}

}

};

int main() {

cout << "A2 - Q3 - 24K-0880 - Muhammad Umer Karim" << endl;

cout << "AI-driven RamzanBox Delivery System - 2030" << endl;

RamzanTimeShip timeShip("TimeShip001", 100, 2000, 15);

RamzanDrone drone("Drone001", 250, 50, 30);

RamzanHyperPod hyperPod("HyperPod001", 500, 5000, 10);

timeShip.command("Deliver", 1);

drone.command("Deliver", 0);

hyperPod.command("Deliver", 1);

ConflictResolution::resolveConflict(timeShip, drone);

ConflictResolution::resolveConflict(drone, hyperPod);

timeShip.move();

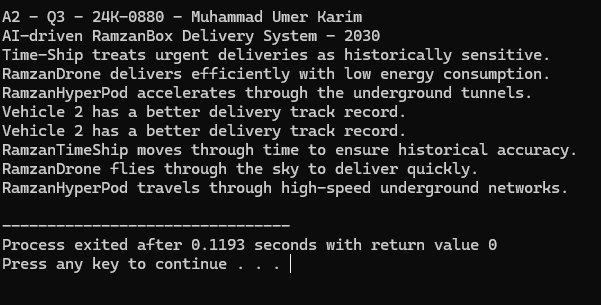
drone.move();

hyperPod.move();

return 0;

}

**Output:**



**Question 4:**

**Code:**

#include <iostream>

#include <string>

using namespace std;

const int MAX\_PERMISSIONS = 5;

const int MAX\_ASSIGNMENTS = 10;

const int MAX\_STUDENTS = 10;

const int MAX\_PROJECTS = 2;

int calculatePasswordHash(const string& password) {

int hash = 5381;

for (char c : password) {

hash = hash \* 33 + c;

}

return hash;

}

class User {

protected:

string name;

string id;

string email;

int hashedPassword;

string permissions[MAX\_PERMISSIONS];

int permissionCount;

public:

User(string n, string i, string e, string p, string perms[], int permCount)

: name(n), id(i), email(e), permissionCount(permCount) {

hashedPassword = calculatePasswordHash(p);

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

permissions[i] = perms[i];

}

}

bool authenticate(string password) {

return calculatePasswordHash(password) == hashedPassword;

}

bool hasPermission(string perm) {

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

if (permissions[i] == perm) return true;

}

return false;

}

virtual void display() {

cout << "Name: " << name << "\nID: " << id << "\nEmail: " << email << "\nPermissions: ";

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

cout << permissions[i] << " ";

}

cout << endl;

}

bool performAction(string action, string password) {

if (!authenticate(password)) {

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

return false;

}

if (!hasPermission(action)) {

cout << "Permission denied!" << endl;

return false;

}

cout << "Action '" << action << "' performed successfully" << endl;

return true;

}

};

class Student : public User {

private:

int assignments[MAX\_ASSIGNMENTS];

int assignmentCount;

public:

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

: User(n, i, e, p, new string[1]{"submit assignment"}, 1), assignmentCount(0) {

for (int i = 0; i < MAX\_ASSIGNMENTS; i++) assignments[i] = 0;

}

void display() override {

cout << "STUDENT DETAILS\n";

User::display();

cout << "Assignments: ";

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

cout << (assignments[i] ? "Submitted" : "Pending") << " ";

}

cout << endl;

}

void addAssignment() {

if (assignmentCount < MAX\_ASSIGNMENTS) {

assignments[assignmentCount++] = 0;

}

}

bool submitAssignment(int num, string password) {

if (!performAction("submit assignment", password)) return false;

if (num < 1 || num > assignmentCount) {

cout << "Invalid assignment number!" << endl;

return false;

}

assignments[num-1] = 1;

cout << "Assignment " << num << " submitted" << endl;

return true;

}

string getName(){

return name;

}

};

class TA : public Student {

private:

Student\* students[MAX\_STUDENTS];

int studentCount;

string projects[MAX\_PROJECTS];

int projectCount;

public:

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

: Student(n, i, e, p), studentCount(0), projectCount(0) {

permissions[permissionCount++] = "view projects";

permissions[permissionCount++] = "manage\_students";

}

void display() override {

cout << "TEACHING ASSISTANT DETAILS\n";

User::display();

cout << "Managing " << studentCount << " students\n";

cout << "Projects: ";

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

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

}

cout << endl;

}

bool addStudent(Student\* s, string password) {

if (!performAction("manage\_students", password)) return false;

if (studentCount >= MAX\_STUDENTS) {

cout << "Cannot manage more students!" << endl;

return false;

}

students[studentCount++] = s;

cout << "Added student " << s->getName() << endl;

return true;

}

bool addProject(string name, string password) {

if (!performAction("view projects", password)) return false;

if (projectCount >= MAX\_PROJECTS) {

cout << "Cannot take more projects!" << endl;

return false;

}

projects[projectCount++] = name;

cout << "Added project " << name << endl;

return true;

}

};

class Professor : public User {

public:

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

: User(n, i, e, p, new string[2]{"assign projects", "full\_lab\_access"}, 2) {}

void display() override {

cout << "PROFESSOR DETAILS\n";

User::display();

}

bool assignProject(TA\* ta, string name, string password) {

if (!performAction("assign projects", password)) return false;

return ta->addProject(name, password);

}

};

int main() {

cout << "A2 - Q3 - 24K-0880 - Muhammad Umer Karim" << endl;

Student s("Umar", "0880", "Umar@school.edu", "22");

TA ta("Akbar", "0696", "Akbar@school.edu", "333");

Professor p("Zahid", "0543", "Zahid@school.edu", "55555");

s.addAssignment();

s.addAssignment();

s.submitAssignment(1, "22");

ta.addStudent(&s, "333");

p.assignProject(&ta, "Quantum Research", "55555");

cout << "\n";

s.display();

cout << "\n";

ta.display();

cout << "\n";

p.display();

return 0;

}

**Output:**

