**OOP ASSIGNMENT 2**

**Q1:**

#include <iostream>

#include <string>

using namespace std;

class User {

protected:

    string id, name;

    bool haspaid;

    string stopAssigned;

public:

    User(string uid, string uname) : id(uid), name(uname), haspaid(false), stopAssigned("None") {}

    virtual void payfees() = 0;

    void assignstop(string stop) { stopAssigned = stop; }

    virtual void showdetails() {

        cout << "ID: " << id << "\nName: " << name

             << "\nTransport Paid: " << (haspaid ? "Yes" : "No")

             << "\nAssigned Stop: " << stopAssigned << endl;

    }

    bool isactive() { return haspaid; }

};

class Student : public User {

public:

    Student(string id, string name) : User(id, name) {}

    void payfees() override {

        haspaid = true;

        cout << name << " has paid semester fees. Transportation card activated.\n";

    }

};

class Teacher : public User {

public:

    Teacher(string id, string name) : User(id, name) {}

    void payfees() override {

        haspaid = true;

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

    }

};

class staffmember : public User {

public:

    staffmember(string id, string name) : User(id, name) {}

    void payfees() override {

        haspaid = true;

        cout << name << " has paid a discounted monthly transport fee.\n";

    }

};

class busroute {

private:

    string routename;

    string stops[3];

public:

    busroute(string name, string s1, string s2, string s3) : routename(name) {

        stops[0] = s1;

        stops[1] = s2;

        stops[2] = s3;

    }

    void showstops() {

        cout << "Route: " << routename << "\nStops: " << stops[0] << ", " << stops[1] << ", " << stops[2] << endl;

    }

    string getstop(int index) { return stops[index]; }

    bool operator==(const busroute &other) {

        return routename == other.routename &&

               stops[0] == other.stops[0] &&

               stops[1] == other.stops[1] &&

               stops[2] == other.stops[2];

    }

};

class transportsystem {

private:

    User \*registereduser;

    busroute \*routes[2];

public:

    transportsystem() {

        registereduser = nullptr;

        routes[0] = new busroute("Route A", "Stop 1", "Stop 2", "Stop 3");

        routes[1] = new busroute("Route B", "Stop A", "Stop B", "Stop C");

    }

    void registerUser(string id, string name, int type) {

        if (registereduser) {

            delete registereduser;

        }

        switch (type) {

            case 1:

                registereduser = new Student(id, name);

                break;

            case 2:

                registereduser = new Teacher(id, name);

                break;

            case 3:

                registereduser = new staffmember(id, name);

                break;

            default:

                cout << "Invalid type!\n";

                return;

        }

        cout << name << " registered successfully!\n";

    }

    void payfees() {

        if (registereduser) {

            registereduser->payfees();

        } else {

            cout << "No user registered!\n";

        }

    }

    void showavailableroutes() {

        for (int i = 0; i < 2; i++) {

            routes[i]->showstops();

        }

    }

    void assignroutes(int routeindex, int stopindex) {

        if (!registereduser) {

            cout << "No user registered!\n";

            return;

        }

        if (routeindex < 1 || routeindex > 2 || stopindex < 1 || stopindex > 3) {

            cout << "Invalid choice!\n";

            return;

        }

        registereduser->assignstop(routes[routeindex - 1]->getstop(stopindex - 1));

        cout << "Stop assigned successfully!\n";

    }

    void chackattendence() {

        if (registereduser && registereduser->isactive()) {

            registereduser->showdetails();

            cout << "Attendance Recorded!\n";

        } else {

            cout << "User has not paid fees! No access to transportation.\n";

        }

    }

    ~transportsystem() {

        delete registereduser;

        for (int i = 0; i < 2; i++) {

            delete routes[i];

        }

    }

};

int main() {

    transportsystem system;

    cout<<"Muhammad Omer Siddiqui/n24k-0022"<<endl;

    int usertype;

    cout << "Enter user type (1-Student, 2-Teacher, 3-Staff): ";

    cin >> usertype;

    system.registerUser("0022", "Omer", usertype);

    system.payfees();

    system.showavailableroutes();

    int routechoice, stopchoice;

    cout << "Enter route number (1-2): ";

    cin >> routechoice;

    cout << "Enter stop number (1-3): ";

    cin >> stopchoice;

    system.assignroutes(routechoice, stopchoice);

    system.chackattendence();

    busroute route1("Route C", "Stop X", "Stop Y", "Stop Z");

    busroute route2("Route C", "Stop X", "Stop Y", "Stop Z");

    if (route1 == route2) {

        cout << "The two routes are identical!\n";

    } else {

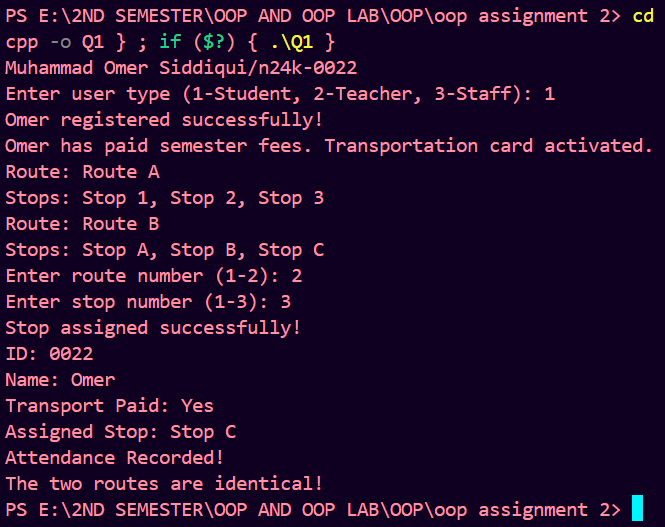
        cout << "The routes are different!\n";

    }

    return 0;

}

**OUTPUT:**



**Q2:**

#include<iostream>

#include<string>

using namespace std;

class ghost {

protected:

    string name;

    int scareLevel;

public:

    ghost(string n, int scare) {

        name = n;

        scareLevel = scare;

    }

    virtual void haunt() const = 0;

    int getScareLevel() const {

        return scareLevel;

    }

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

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

        return os;

    }

    virtual ~ghost() {}

};

class ultepaonwalichurail : public ghost {

public:

    ultepaonwalichurail() : ghost("", 0) {

        name = "Specter";

        scareLevel = 6;

    }

    void haunt() const override {

        cout << name << " hovers eerily!\n";

    }

};

class pichalperi : public ghost {

public:

    pichalperi() : ghost("", 0) {

        name = "Wraith";

        scareLevel = 8;

    }

    void haunt() const override {

        cout << name << " glides silently through walls!\n";

    }

};

class banshee : public ghost {

public:

    banshee() : ghost("", 0) {

        name = "Banshee";

        scareLevel = 7;

    }

    void haunt() const override {

        cout << name << " screams with an ear-piercing wail!\n";

    }

};

class karsazkidulhan : public ghost {

public:

    karsazkidulhan() : ghost("", 0) {

        name = "Revenant";

        scareLevel = 9;

    }

    void haunt() const override {

        cout << name << " rises from the ground in fury!\n";

    }

};

class poltergeist : public ghost {

public:

    poltergeist() : ghost("", 0) {

        name = "Poltergeist";

        scareLevel = 5;

    }

    void haunt() const override {

        cout << name << " throws objects around wildly!\n";

    }

};

class visitor {

    string name;

    int bravery;

public:

    visitor(string n, int b) {

        name = n;

        bravery = b;

    }

    void react(const ghost& g) const {

        int scare = g.getScareLevel();

        if (scare < bravery - 2) cout << name << " chuckles at the ghost!\n";

        else if (scare > bravery + 2) cout << name << " screams and runs away!\n";

        else cout << name << " trembles in fear!\n";

    }

    string getName() const {

        return name;

    }

};

class hauntedhouse {

    string houseName;

    ghost\* ghosts[5];

    int ghostCount;

public:

    hauntedhouse(string name) {

        houseName = name;

        ghostCount = 0;

    }

    void addghost(ghost\* g) {

        if (ghostCount < 5) {

            ghosts[ghostCount] = g;

            ghostCount++;

        }

    }

    void runsimulation(visitor visitors[], int visitorCount) {

        cout << "\nWelcome to " << houseName << "!\n";

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

            cout << "\nVisitor " << visitors[i].getName() << " enters...\n";

            for (int j = 0; j < ghostCount; j++) {

                ghosts[j]->haunt();

                visitors[i].react(\*ghosts[j]);

            }

        }

        cout << "\nEnd of " << houseName << " Visit\n";

    }

    ~hauntedhouse() {

        for (int i = 0; i < ghostCount; i++) delete ghosts[i];

    }

};

void visit(visitor visitors[], int count, hauntedhouse& house) {

    house.runsimulation(visitors, count);

}

int main() {

    cout<<"Muhammad Omer Siddiqui/n24k-0022"<<endl;

    hauntedhouse house1("bangali baba ki kothi");

    hauntedhouse house2("purani haweli");

    house1.addghost(new ultepaonwalichurail());

    house1.addghost(new pichalperi());

    house1.addghost(new banshee());

    house2.addghost(new karsazkidulhan());

    house2.addghost(new poltergeist());

    house2.addghost(new ultepaonwalichurail());

    visitor visitors[3] = {

        visitor("Omer", 3),

        visitor("ammar", 6),

        visitor("arham", 9)

    };

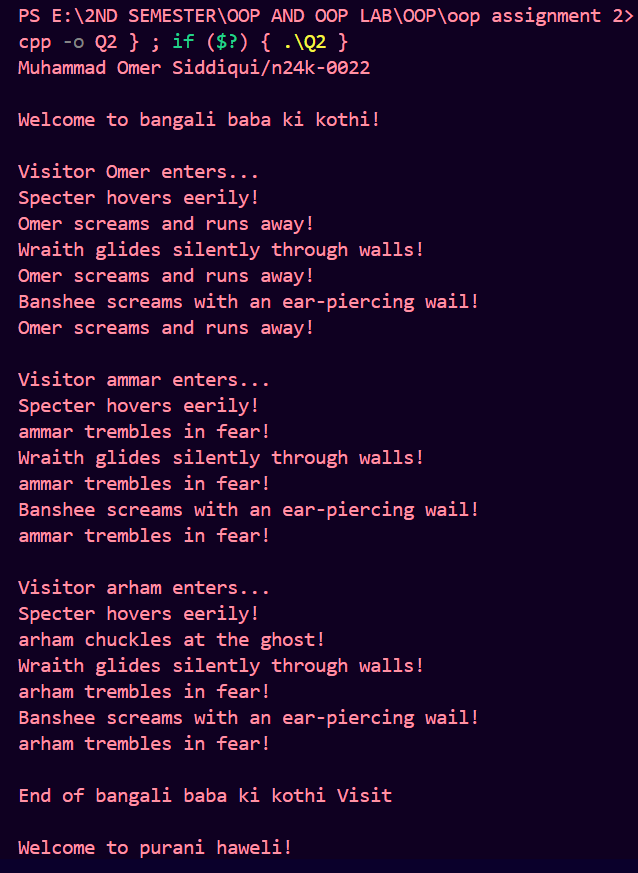
    visit(visitors, 3, house1);

    visit(visitors, 3, house2);

    return 0;

}

**OUTPUT:**



A screenshot of a computer screen

AI-generated content may be incorrect.

**Q3:**

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

class Vehicle {

protected:

    string vehicleid;

    int speed;

    int capacity;

    double efficiency;

public:

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

        : vehicleid(id), speed(spd), capacity(cap), efficiency(eff) {}

    virtual void deliver() = 0;

    virtual void move() = 0;

    bool operator==(const Vehicle &v) {

        return (*this*->efficiency == v.efficiency);

    }

    friend void resolveconflict(Vehicle &v1, Vehicle &v2);

    string getid() { return vehicleid; }

    int getspeed() { return speed; }

    int getcapacity() { return capacity; }

    double getefficiency() { return efficiency; }

    virtual ~Vehicle() {}

};

class ramazandrone : public Vehicle {

public:

    ramazandrone(string id) : Vehicle(id, 150, 10, 15.0) {}

    void deliver() override {

        cout << "Drone " << vehicleid << " is delivering small iftar meals.\n";

    }

    void move() override {

        cout << "Drone " << vehicleid << " is flying in the air.\n";

    }

};

class ramazantimeship : public Vehicle {

public:

    ramazantimeship(string id) : Vehicle(id, 50, 200, 5.0) {}

    void deliver() override {

        cout << "TimeShip " << vehicleid << " is delivering food through time.\n";

    }

    void move() override {

        cout << "TimeShip " << vehicleid << " is traveling through time.\n";

    }

};

class ramazanhyperpod : public Vehicle {

public:

    ramazanhyperpod(string id) : Vehicle(id, 300, 1000, 12.0) {}

    void deliver() override {

        cout << "HyperPod " << vehicleid << " is delivering large iftar packages.\n";

    }

    void move() override {

        cout << "HyperPod " << vehicleid << " is speeding through underground tunnels.\n";

    }

};

void resolveconflict(Vehicle &v1, Vehicle &v2) {

    if (v1 == v2) {

        cout << "Both vehicles " << v1.getid() << " and " << v2.getid() << " have equal efficiency.\n";

    } else {

        cout << "Choosing the more efficient vehicle based on efficiency:\n";

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

            cout << "Vehicle " << v1.getid() << " is more efficient.\n";

        } else {

            cout << "Vehicle " << v2.getid() << " is more efficient.\n";

        }

    }

}

int main() {

    cout<<"Muhammad Omer Siddiqui/n24k-0022"<<endl;

    ramazandrone drone1("Drone 01");

    ramazantimeship timeShip1("TimeShip 01");

    ramazanhyperpod hyperPod1("HyperPod 01");

    drone1.deliver();

    timeShip1.deliver();

    hyperPod1.deliver();

    drone1.move();

    timeShip1.move();

    hyperPod1.move();

    resolveconflict(drone1, timeShip1);

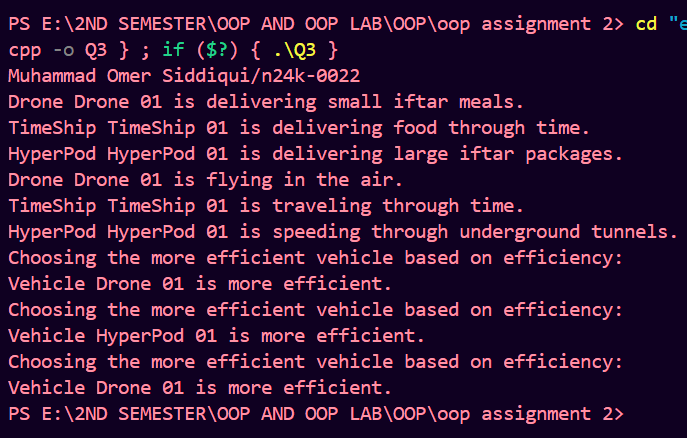
    resolveconflict(timeShip1, hyperPod1);

    resolveconflict(drone1, hyperPod1);

    return 0;

}

**OUTPUT:**

****

**Q4:**

#include <iostream>

using namespace std;

unsigned long hash\_passcode(string password)

{

    unsigned long hash\_value = 5381;

    for (char ch : password)

    {

        hash\_value = (hash\_value \* 33) + ch;

    }

    return hash\_value;

}

class User {

public:

    string user\_name;

    int user\_id;

    string user\_email;

    unsigned long hashed\_password;

    string user\_permissions[3];

    User(string name, int id, string email, string password, string perm1 = "", string perm2 = "", string perm3 = "")

        : user\_name(name), user\_id(id), user\_email(email), hashed\_password(hash\_passcode(password))

    {

        user\_permissions[0] = perm1;

        user\_permissions[1] = perm2;

        user\_permissions[2] = perm3;

    }

    virtual void display()

    {

        cout << "Name: " << user\_name << endl << "ID: " << user\_id << endl << "Email: " << user\_email << endl;

    }

    bool validate\_password(string password)

    {

        return hash\_passcode(password) == hashed\_password;

    }

    bool has\_permission(string permission)

    {

        for (int i = 0; i < 3; i++)

        {

            if (user\_permissions[i] == permission)

                return true;

        }

        return false;

    }

    virtual bool has\_lab\_access()

    {

        if (has\_permission("full\_lab\_access"))

        {

            cout << user\_name << " has full access to the lab." << endl;

            return true;

        }

        else

        {

            cout << user\_name << " does not have lab access." << endl;

            return false;

        }

    }

};

class Student : public User

{

public:

    int assignments[5];

    Student(string name, int id, string email, string password)

        : User(name, id, email, password, "submit\_assignment")

    {

        for (int i = 0; i < 5; i++)

        {

            assignments[i] = 0;

        }

    }

    void submit\_assignment(int index)

    {

        if (index >= 0 && index < 5)

        {

            assignments[index] = 1;

            cout << "Assignment " << index + 1 << " submitted by " << user\_name << "." << endl;

        }

        else

        {

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

        }

    }

    void display() override

    {

        User::display();

        cout << "Assignments: ";

        for (int i = 0; i < 5; i++)

        {

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

        }

        cout << endl;

    }

};

class Teaching\_Assistant : public Student

{

public:

    string assigned\_students[10];

    int student\_counter;

    string assigned\_projects[2];

    int project\_counter;

    Teaching\_Assistant(string name, int id, string email, string password)

        : Student(name, id, email, password)

    {

        user\_permissions[1] = "view\_projects";

        user\_permissions[2] = "manage\_students";

        student\_counter = 0;

        project\_counter = 0;

    }

    void assign\_student(string student\_name)

    {

        if (student\_counter < 10)

        {

            assigned\_students[student\_counter++] = student\_name;

            cout << student\_name << " assigned to TA " << user\_name << endl;

        }

        else

        {

            cout << "TA " << user\_name << " cannot manage more than 10 students." << endl;

        }

    }

    void add\_project(string project\_name)

    {

        if (project\_counter < 2)

        {

            assigned\_projects[project\_counter++] = project\_name;

            cout << "TA " << user\_name << " started working on project: " << project\_name << endl;

        }

        else

        {

            cout << "TA " << user\_name << " is already working on 2 projects." << endl;

        }

    }

    void display() override

    {

        Student::display();

        cout << "Assigned Students: ";

        for (int i = 0; i < student\_counter; i++)

        {

            cout << i + 1 << ". " << assigned\_students[i] << endl;

        }

        cout << "Projects: ";

        for (int i = 0; i < project\_counter; i++)

        {

            cout << assigned\_projects[i] << " ";

        }

        cout << endl;

    }

};

class Professor : public User

{

public:

    Professor(string name, int id, string email, string password)

        : User(name, id, email, password, "assign\_projects", "full\_lab\_access") {}

    void assign\_project(Teaching\_Assistant& ta, string project\_name)

    {

        ta.add\_project(project\_name);

    }

    void display() override

    {

        User::display();

        cout << "Professor with full lab access." << endl;

    }

};

void validate\_and\_perform\_action(User\* user, string password, string action)

{

    if (user->validate\_password(password))

    {

        cout << "Authentication successful for " << user->user\_name << "." << endl;

        if (user->has\_permission(action))

        {

            cout << "Action permitted: " << action << "." << endl;

        }

        else

        {

            cout << "Action denied: " << action << "." << endl;

        }

    }

    else

    {

        cout << "Authentication failed for " << user->user\_name << "." << endl;

    }

}

int main()

{

    Student student\_1("Omer", 240022, "omer@university.edu", "password123");

    Teaching\_Assistant ta\_1("Mr\_TA", 249807, "ta@university.edu", "securepass");

    Professor professor\_1("Dr\_Khalid", 224567, "khalid@university.edu", "professorpass");

    student\_1.display();

    validate\_and\_perform\_action(&student\_1, "password123", "submit\_assignment");

    validate\_and\_perform\_action(&ta\_1, "securepass", "manage\_students");

    validate\_and\_perform\_action(&professor\_1, "professorpass", "assign\_projects");

    student\_1.submit\_assignment(1);

    student\_1.display();

    ta\_1.assign\_student("Omer");

    ta\_1.add\_project("AI Research");

    ta\_1.display();

    return 0;

}

**OUTPUT:**

