**Screenshots and User Walkthroughs**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Person {

public:

    string name;

    string rollno;

    bool feesPaid;

    Person(string n = "", string id = "", bool fees = false)

        : name(n), rollno(id), feesPaid(fees) {}

    virtual void registerPerson() = 0;

    virtual void payFees() = 0;

    virtual void login() = 0;

    virtual void writeToFile(ofstream& out) = 0;

    virtual void readFromFile(ifstream& in) = 0;

};

template <typename T>

class FileHandler {

public:

    void saveToFile(const char\* filename, T\* arr, int count) {

        try {

            ofstream out(filename, ios::binary);

            if (!out) throw runtime\_error("Cannot open file for writing");

            out.write((char\*)&count, sizeof(count));

            for (int i = 0; i < count; i++) arr[i].writeToFile(out);

            out.close();

        } catch (exception& e) {

            cerr << "File write error: " << e.what() << endl;

        }

    }

    int loadFromFile(const char\* filename, T\* arr, int maxCount) {

        int count = 0;

        try {

            ifstream in(filename, ios::binary);

            if (!in) throw runtime\_error("Cannot open file for reading");

            in.read((char\*)&count, sizeof(count));

            if (count > maxCount) count = maxCount;

            for (int i = 0; i < count; i++) arr[i].readFromFile(in);

            in.close();

        } catch (exception& e) {

            cerr << "File read error: " << e.what() << endl;

        }

        return count;

    }

};

class Student : public Person {

public:

    string assignedStop;

    Student(string n = "", string id = "", bool fees = false, 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 << "Fees already paid.\n";

        else {

            cout << "Student pays fees.\n";

            feesPaid = true;

        }

    }

    void login() override {

        cout << "Student " << name << " logs in.\n";

    }

    void writeToFile(ofstream& out) override {

        size\_t len;

        len = name.size(); out.write((char\*)&len, sizeof(len)); out.write(name.c\_str(), len);

        len = rollno.size(); out.write((char\*)&len, sizeof(len)); out.write(rollno.c\_str(), len);

        out.write((char\*)&feesPaid, sizeof(feesPaid));

        len = assignedStop.size(); out.write((char\*)&len, sizeof(len)); out.write(assignedStop.c\_str(), len);

    }

    void readFromFile(ifstream& in) override {

        size\_t len; char buffer[100];

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; name = buffer;

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; rollno = buffer;

        in.read((char\*)&feesPaid, sizeof(feesPaid));

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; assignedStop = buffer;

    }

};

class Teacher : public Person {

public:

    Teacher(string n = "", string id = "", bool fees = false)

        : Person(n, id, fees) {}

    void registerPerson() override {

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

    }

    void payFees() override {

        if (feesPaid) cout << "Fees already paid.\n";

        else {

            cout << "Teacher pays monthly fees.\n";

            feesPaid = true;

        }

    }

    void login() override {

        cout << "Teacher " << name << " logs in.\n";

    }

    void writeToFile(ofstream& out) override {

        size\_t len;

        len = name.size(); out.write((char\*)&len, sizeof(len)); out.write(name.c\_str(), len);

        len = rollno.size(); out.write((char\*)&len, sizeof(len)); out.write(rollno.c\_str(), len);

        out.write((char\*)&feesPaid, sizeof(feesPaid));

    }

    void readFromFile(ifstream& in) override {

        size\_t len; char buffer[100];

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; name = buffer;

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; rollno = buffer;

        in.read((char\*)&feesPaid, sizeof(feesPaid));

    }

};

class Admin : public Person {

public:

    Admin(string n = "", string id = "", bool fees = false)

        : Person(n, id, fees) {}

    void registerPerson() override {

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

    }

    void payFees() override {

        cout << "Admin does not pay fees.\n";

    }

    void login() override {

        cout << "Admin " << name << " logs in.\n";

    }

    void writeToFile(ofstream& out) override {

        size\_t len;

        len = name.size(); out.write((char\*)&len, sizeof(len)); out.write(name.c\_str(), len);

        len = rollno.size(); out.write((char\*)&len, sizeof(len)); out.write(rollno.c\_str(), len);

        out.write((char\*)&feesPaid, sizeof(feesPaid));

    }

    void readFromFile(ifstream& in) override {

        size\_t len; char buffer[100];

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; name = buffer;

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; rollno = buffer;

        in.read((char\*)&feesPaid, sizeof(feesPaid));

    }

};

class Stop {

public:

    string stopName;

    Stop(string name = "") : stopName(name) {}

};

class Route {

public:

    string routeName;

    Stop stops[10];

    Route(string name = "") : routeName(name) {}

};

class Point {

private:

    int pointNumber;

    int capacity;

    Route assignedRoute;

public:

    Point(int number = 0, int cap = 0, Route r = Route())

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

    void boardStudent(Person& p) {

        cout << p.name << " boarded the point.\n";

    }

    void writeToFile(ofstream& out) {

        out.write((char\*)&pointNumber, sizeof(pointNumber));

        out.write((char\*)&capacity, sizeof(capacity));

        size\_t len = assignedRoute.routeName.size();

        out.write((char\*)&len, sizeof(len)); out.write(assignedRoute.routeName.c\_str(), len);

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

            len = assignedRoute.stops[i].stopName.size();

            out.write((char\*)&len, sizeof(len));

            out.write(assignedRoute.stops[i].stopName.c\_str(), len);

        }

    }

    void readFromFile(ifstream& in) {

        in.read((char\*)&pointNumber, sizeof(pointNumber));

        in.read((char\*)&capacity, sizeof(capacity));

        size\_t len; char buffer[100];

        in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0'; assignedRoute.routeName = buffer;

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

            in.read((char\*)&len, sizeof(len)); in.read(buffer, len); buffer[len] = '\0';

            assignedRoute.stops[i].stopName = buffer;

        }

    }

};

int main() {

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

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

    Admin a1("Principal", "A-01", true);

    Person\* users[3] = { &s1, &t1, &a1 };

    cout << "=== Registering and Logging In ===\n";

    for (int i = 0; i < 3; i++) {

        users[i]->registerPerson();

        users[i]->login();

        users[i]->payFees();

        cout << endl;

    }

    FileHandler<Student> studentFH;

    studentFH.saveToFile("students.bin", &s1, 1);

    FileHandler<Teacher> teacherFH;

    teacherFH.saveToFile("teachers.bin", &t1, 1);

    FileHandler<Admin> adminFH;

    adminFH.saveToFile("admins.bin", &a1, 1);

    Student loadedStudent;

    studentFH.loadFromFile("students.bin", &loadedStudent, 1);

    cout << "Loaded Student: " << loadedStudent.name << ", Stop: " << loadedStudent.assignedStop << "\n";

    Route r1("University Route");

    r1.stops[0] = Stop("Gulshan");

    r1.stops[1] = Stop("Malir");

    Point p1(101, 20, r1);

    p1.boardStudent(s1);

    FileHandler<Point> pointFH;

    pointFH.saveToFile("points.bin", &p1, 1);

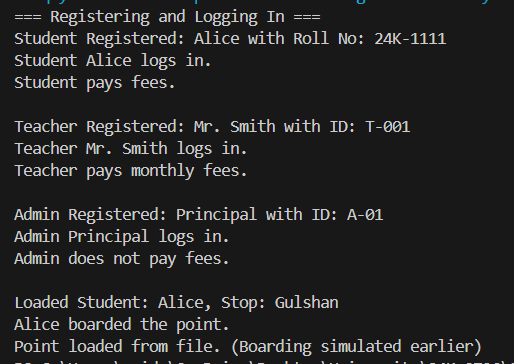
    Point loadedPoint;

    pointFH.loadFromFile("points.bin", &loadedPoint, 1);

    cout << "Point loaded from file. (Boarding simulated earlier)\n";

}

Output:



**Walkthrough:**

Student’s Perspective:

1. **Register**

Action: Student enters their name, roll number, and assigned stop.

Result: The system registers the student and links them to the transport network.

1. **Pay Fees**

Action: Student pays transport fees.

Result: System checks if fees were already paid. If not, the system marks the fees as paid.

1. **Board Transport**

Action: Student boards from their assigned stop.

Result: System confirms boarding and ensures capacity is not exceeded.

Teacher’s Perspective:

1. **Register**

Action: Teacher enters name and ID.

Result: Teacher is registered in the system.

1. **Pay Monthly Fees**

Action: Teacher pays their transport fee.

Result: System confirms the payment status.

1. **Board Transport**

Action: Teacher boards from a designated stop.

Result: System allows boarding based on availability.

Admin’s Perspective:

1. **Add New Users (Students, Teachers, Staff)**

Action: Admin inputs user data to create new entries.

Result: New users are added to the system.

1. **Assign/Remove Stops**

Action: Admin assigns students to stops or removes them.

Result: Stops update their records accordingly.

1. **Create/Edit Routes**

Action: Admin creates a transport route made up of multiple stops.

Result: Route becomes part of the active network.

1. **Assign Routes to Points**

Action: Admin connects a transport point (like a bus) to a specific route.

Result: Point uses the route for daily pickups.

1. **View User and Point Data**

Action: Admin retrieves stored data about users or points.

Result: System loads info from the files.