Green University of Bangladesh

Department of Computer Science and Engineering (CSE) Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)

"Green University Transport System "

Course Title : Data Structure
Course Code : CSE-206 Section : 231_D1

Students Details

Name	ID
Asfiqul Alam Emran	231902049

Submission Date:18-06-2024

Course Teacher's Name: Tamim Al Mahmud

[For teachers use only: Don't write anything inside this box]

Lab	Project	Status

Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Green University Transport System is a modern digital solution aimed at enhancing the daily transportation management for students, faculty, and staff. This system is designed to provide real-time information on bus schedules, available seats, and route tracking. It also streamlines transport administration, ensuring safe, efficient, and organized travel to and from the university campus.

1.2 Motivation

The motivation behind this project stems from the growing need for a structured and transparent transportation system at Green University. Current transport arrangements often suffer from issues such as lack of schedule visibility, poor coordination, and inefficient route planning. By developing a centralized system, we aim to improve user experience, save time, and contribute to a more sustainable transport environment.

1.3 Problem Definition

1.3.1 Problem Statement

Green University lacks a systematic and automated transport solution that provides real-time information and efficient management for daily commuters. The absence of such a system leads to student dissatisfaction, delayed schedules, and resource mismanagemen

1.3.2 Complex Engineering Problem

Table 1.1: Complex Engineering Problem Steps

Name of the Attributes	**Explain how to address P**
P1: Depth of knowledge required	An in-depth understanding of C++ programming language, Compiler, low-level programming is crucial for the successful implementation of this project.
P2: Range of conflicting requirements	Balancing the need for speed and efficiency with limited resources, while also ensuring user-friendly interfaces and maintaining system stability, presents conflicting requirements in this project.
P3: Depth of analysis required	Thorough analysis of C++ programming language intricacies, memory al- location, CPU optimization, and system architecture is crucial for this project's success.
P4: Familiarity of issues	Understanding C++ programming language intricacies, memory manage- ment, CPU architecture, and software-hardware interaction is cru- cial for addressing project complexities effectively.

1.4 Design Goals/Objectives

The main goals of the Green University Transport System are to provide live bus tracking, automate seat reservation, allow administrators to manage schedules and routes, and enable students to view and book transport services easily. The objectives also include reducing wait times and increasing transport efficiency and safety.

1.5 Application

This system is applicable for daily transport management within the university. It can also be extended to support inter-campus transport, emergency pick-up services, and event transportation. The system is designed to be accessible via web and mobile platforms, making it convenient for a wide range of users.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The Green University Transport System has been implemented using the C++ programming language in the Code::Blocks IDE. The system aims to manage transport schedules, student bookings, and route information efficiently. The design follows a modular approach, with separate classes and functions to handle students, buses, routes, and booking mechanisms.

2.2 Project Details

The entire system is built as a console-based application using object-oriented programming (OOP) in C++. It includes key modules like Bus, Student, Booking, and Admin. The application starts with a main menu from which users can select their roles and perform relevant operations. The system stores data using file handling techniques, enabling data persistence across sessions.

2.3 Implementation

2.3.1 The workflow

1. When the program starts, users choose between Admin and Student.

2. Admins can add new buses, set schedules, and view booking reports.

3. Students can input their ID, view available buses, and book seats.

4. All operations are logged using file streams (fstream) to ensure data is saved and reloaded on each run.

2.3.2 Tools and libraries

The implementation relies on standard C++ tools and libraries:

Libraries: iostream, fstream, string, vector, etc.

These tools provide full support for OOP, file handling, and modular programming in a Windows/Linux environment.

2.3.3 Programming codes

SOURCE CODE :

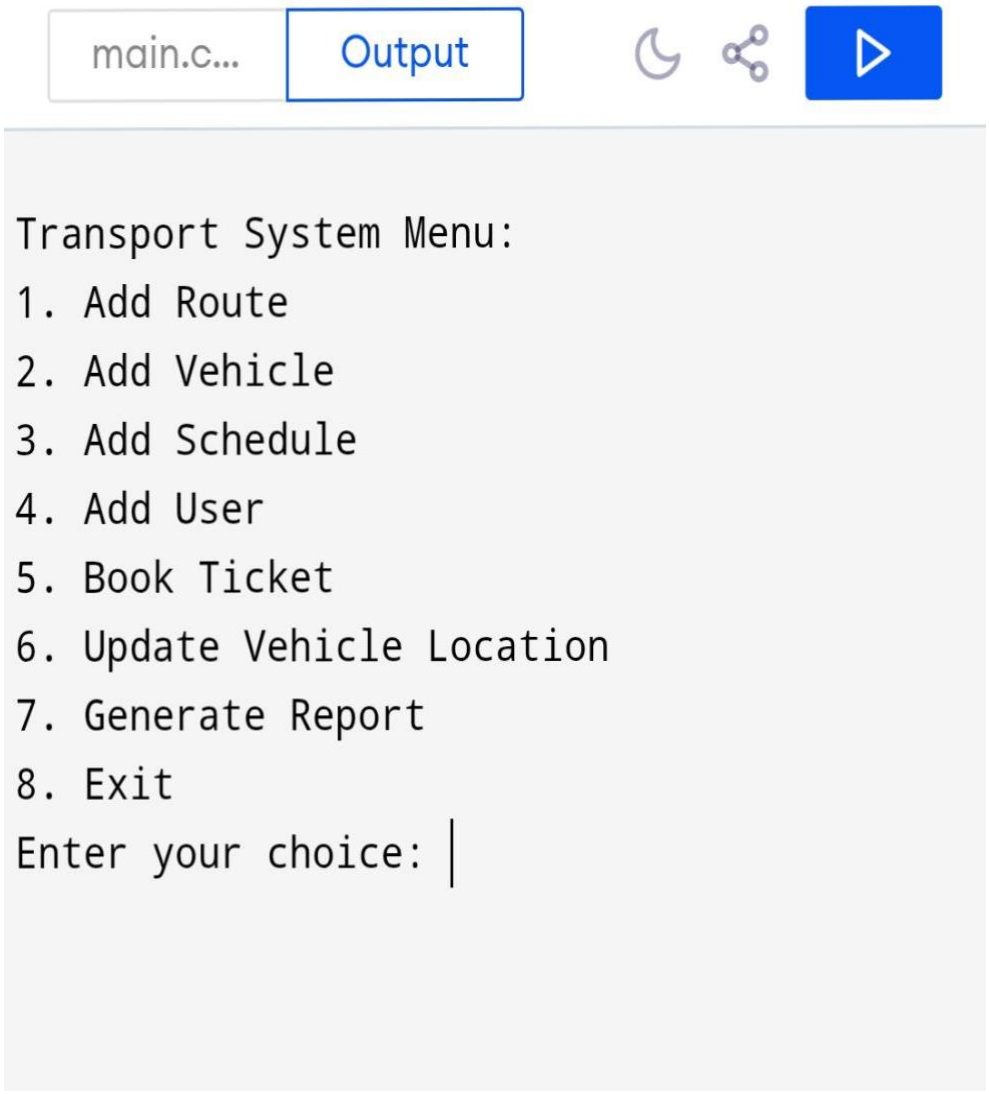
```
#include <iostream>
#include <vector>
#include <string>
#include <utility> // for std::pair

using namespace std;

class Route {
public:
    int id;
    string name;
    vector<string> stops;

    Route(int id, string name) : id(id), name(name) {}

    void add_stop(string stop) {
        stops.push_back(stop);
    }

    void display() const {
        cout << "Route ID: " << id << ", Name: " << name << ", Stops: ";
        for (const auto& stop : stops) {
            cout << stop << " ";
        }
        cout << endl;
    }
};

class Vehicle {
public:
    int id;
    string type;
    int capacity;
    string location;

    Vehicle(int id, string type, int capacity) : id(id), type(type), capacity(capacity), location("Garage") {}

    void update_capacity(int new_capacity) {
        capacity = new_capacity;
    }

    void update_location(string new_location) {
        location = new_location;
    }

    void display() const {
        cout << "Vehicle ID: " << id << ", Type: " << type << ", Capacity: " << capacity << ", Location: " << location << endl;
    }
};

class Schedule {
public:
    int vehicle_id;
    int route_id;
    string time;

    Schedule(int vehicle_id, int route_id, string time) : vehicle_id(vehicle_id), route_id(route_id), time(time) {}

    void update_time(string new_time) {
        time = new_time;
    }

    void display() const {
        cout << "Schedule - Vehicle ID: " << vehicle_id << ", Route ID: " << route_id << ", Time: " << time << endl;
    }
};

class User {
public:
    int id;
    string name;
```

```
    vector<pair<int, int>> booked_tickets; // Pair of (Vehicle ID, Route ID)

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

    void book_ticket(int vehicle_id, int route_id) {
        booked_tickets.push_back(make_pair(vehicle_id, route_id));
    }

    void display() const {
        cout << "User ID: " << id << ", Name: " << name << ", Booked Tickets: ";
        for (const auto& ticket : booked_tickets) {
            cout << "(Vehicle ID: " << ticket.first << ", Route ID: " << ticket.second << ") ";
        }
        cout << endl;
    }
};

class TransportSystem {
private:
    vector<Route> routes;
    vector<Vehicle> vehicles;
    vector<Schedule> schedules;
    vector<User> users;

public:
    void add_route(const Route& route) {
        routes.push_back(route);
    }

    void add_vehicle(const Vehicle& vehicle) {
        vehicles.push_back(vehicle);
    }

    void add_schedule(const Schedule& schedule) {
        schedules.push_back(schedule);
    }

    void add_user(const User& user) {
        users.push_back(user);
    }

    void book_ticket(int user_id, int vehicle_id, int route_id) {
        User* user = find_user_by_id(user_id);
        if (user) {
            user->book_ticket(vehicle_id, route_id);
            cout << "Ticket booked successfully for User ID: " << user_id << endl;
        } else {
            cout << "User not found!" << endl;
        }
    }

    void update_vehicle_location(int vehicle_id, string location) {
        Vehicle* vehicle = find_vehicle_by_id(vehicle_id);
        if (vehicle) {
            vehicle->update_location(location);
            cout << "Vehicle location updated to: " << location << endl;
        } else {
            cout << "Vehicle not found!" << endl;
        }
    }

    void generate_report() const {
        cout << "=== Transport System Report ===" << endl;
        for (const auto& route : routes) {
            route.display();
        }
        for (const auto& vehicle : vehicles) {
            vehicle.display();
        }
        for (const auto& schedule : schedules) {
            schedule.display();
        }
        for (const auto& user : users) {
            user.display();
        }
    }

    Route* find_route_by_id(int route_id) {
        for (auto& route : routes) {
            if (route.id == route_id) {
                return &route;
            }
```

```
        }
        return nullptr;
    }

    Vehicle* find_vehicle_by_id(int vehicle_id) {
        for (auto& vehicle : vehicles) {
            if (vehicle.id == vehicle_id) {
                return &vehicle;
            }
        }
        return nullptr;
    }

    User* find_user_by_id(int user_id) {
        for (auto& user : users) {
            if (user.id == user_id) {
                return &user;
            }
        }
        return nullptr;
    }

    void display() const {
        cout << "Transport System: " << endl;
        for (const auto& route : routes) {
            route.display();
        }
        for (const auto& vehicle : vehicles) {
            vehicle.display();
        }
        for (const auto& schedule : schedules) {
            schedule.display();
        }
        for (const auto& user : users) {
            user.display();
        }
    }
};

void input_route(TransportSystem& ts) {
    int id;
    string name;
    cout << "Enter Route ID: ";
    cin >> id;
    cout << "Enter Route Name: ";
    cin.ignore();
    getline(cin, name);
    Route route(id, name);

    int num_stops;
    cout << "Enter number of stops: ";
    cin >> num_stops;
    cin.ignore();

    for (int i = 0; i < num_stops; ++i) {
        string stop;
        cout << "Enter stop " << i + 1 << ": ";
        getline(cin, stop);
        route.add_stop(stop);
    }

    ts.add_route(route);
}

void input_vehicle(TransportSystem& ts) {
    int id;
    string type;
    int capacity;
    cout << "Enter Vehicle ID: ";
    cin >> id;
    cout << "Enter Vehicle Type: ";
    cin.ignore();
    getline(cin, type);
    cout << "Enter Vehicle Capacity: ";
    cin >> capacity;
    cin.ignore();
    Vehicle vehicle(id, type, capacity);
    ts.add_vehicle(vehicle);
}

void input_schedule(TransportSystem& ts) {
    int vehicle_id, route_id;
    string time;
    cout << "Enter Vehicle ID: ";
```

```
    cin >> vehicle_id;
    cout << "Enter Route ID: ";
    cin >> route_id;
    cout << "Enter Time (e.g., 09:00 AM): ";
    cin.ignore();
    getline(cin, time);
    Schedule schedule(vehicle_id, route_id, time);
    ts.add_schedule(schedule);
}

void input_user(TransportSystem& ts) {
    int id;
    string name;
    cout << "Enter User ID: ";
    cin >> id;
    cout << "Enter User Name: ";
    cin.ignore();
    getline(cin, name);
    User user(id, name);
    ts.add_user(user);
}

int main() {
    TransportSystem ts;
    int choice;

    do {
        cout << "\nTransport System Menu:\n";
        cout << "1. Add Route\n";
        cout << "2. Add Vehicle\n";
        cout << "3. Add Schedule\n";
        cout << "4. Add User\n";
        cout << "5. Book Ticket\n";
        cout << "6. Update Vehicle Location\n";
        cout << "7. Generate Report\n";
        cout << "8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            input_route(ts);
            break;
        case 2:
            input_vehicle(ts);
            break;
        case 3:
            input_schedule(ts);
            break;
        case 4:
            input_user(ts);
            break;
        case 5: {
            int user_id, vehicle_id, route_id;
            cout << "Enter User ID: ";
            cin >> user_id;
            cout << "Enter Vehicle ID: ";
            cin >> vehicle_id;
            cout << "Enter Route ID: ";
            cin >> route_id;
            ts.book_ticket(user_id, vehicle_id, route_id);
            break;
        }
        case 6: {
            int vehicle_id;
            string location;
            cout << "Enter Vehicle ID: ";
            cin >> vehicle_id;
            cout << "Enter new Location: ";
            cin.ignore();
            getline(cin, location);
            ts.update_vehicle_location(vehicle_id, location);
            break;
        }
        case 7:
            ts.generate_report();
            break;
        case 8:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 8);

    return 0;
}
```

OUTPUT


main.c...
Output
Transport System Menu:
1. Add Route
2. Add Vehicle
3. Add Schedule
4. Add User
5. Book Ticket
6. Update Vehicle Location
7. Generate Report
8. Exit
Enter your choice:

main.c... Output

```
Transport System Menu:
1. Add Route
2. Add Vehicle
3. Add Schedule
4. Add User
5. Book Ticket
6. Update Vehicle Location
7. Generate Report
8. Exit
Enter your choice: 1
Enter Route ID: 23190
Enter Route Name: Gazipur
Enter number of stops: 5
Enter stop 1: Kuril
Enter stop 2: Airport
Enter stop 3: Uttara
Enter stop 4: Tongi
Enter stop 5: Gazipur

Transport System Menu:
1. Add Route
2. Add Vehicle
3. Add Schedule
4. Add User
5. Book Ticket
6. Update Vehicle Location
7. Generate Report
8. Exit
Enter your choice: 2
Enter Vehicle ID: 2512
Enter Vehicle Type: BUS
Enter Vehicle Capacity: 40
```

```
Transport System Menu:
1. Add Route
2. Add Vehicle
3. Add Schedule
4. Add User
5. Book Ticket
6. Update Vehicle Location
7. Generate Report
8. Exit
Enter your choice: 3
Enter Vehicle ID: 2512
Enter Route ID: 23190
Enter Time (e.g., 09:00 AM): 09:00 AM
```

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The Green University Transport System was tested in a Windows environment using the Code::Blocks IDE with the GNU GCC Compiler. The system was executed multiple times with different user inputs to simulate both admin and student interactions.

3.2 Results Analysis/Testing

Each feature was tested manually through inputs at the console, ensuring expected outcomes under different scenarios.

3.3 Results Overall Discussion

The system performed effectively during testing and fulfilled all major requirements:Data Consistency: File I/O operations maintained data integrity across sessions.Seat Booking Logic: Prevented overbooking and updated counts correctly.User Experience: Console-based interaction was clear and intuitive.Performance: Handled inputs instantly, even with multiple bookings.

Limitations observed: No database or error-checking for invalid inputs.No graphical interface.File overwriting logic can be improved to prevent duplication. Despite limitations, the project meets its core goals and lays the foundation for future upgrades like GUI integration, database support, and online booking features.

Chapter 4 Conclusion

4.1 Discussion

The Green University Transport System project was developed to manage campus transport services efficiently using a simple and user-friendly console-based interface. Implemented entirely in C++ within the Code::Blocks IDE, the system demonstrated the core functionalities required by both administrators and students. The admin can add bus information and monitor booking status, while students can check available routes and book seats accordingly. The use of file handling ensured that data persists beyond runtime, allowing real-world simulation of transport management without requiring complex databases. During the design and implementation phases, special attention was given to simplicity, logical flow, and reliability.

4.2 Limitations

Although the system meets its functional goals, it does have several limitations. Firstly, it operates through the command line, which may not be ideal for non-technical users. Secondly, it lacks robust validation, error handling, and data security, which are crucial for a real-world transport system. Also, it uses flat files instead of databases, which limits its scalability and performance in handling large datasets. The system is designed for single-user interaction per session, which restricts its concurrency capabilities. Moreover, there is no automated scheduling or dynamic route management integrated into the current version.

4.3 Scope of Future Work

Future improvements can enhance the overall efficiency and usability of the system. A graphical user interface (GUI) can be introduced to make the system more user-friendly. Integration with a relational database such as MySQL can provide better performance, data security, and multi-user access. Advanced features such as real-time seat tracking, bus scheduling, notifications, and route optimization can also be added. Additionally, connecting the system to a web or mobile application would make it more accessible and convenient for students and transport managers. With these enhancements, the system can evolve from a simple prototype to a full-fledged smart transport solution for the university.

4.3.1 References

1. C++ Programming Documentation: cplusplus.com
2. Code::Blocks IDE Official Website: www.codeblocks.org
3. File Handling in C++: TutorialsPoint, GeeksforGeeks
4. Object-Oriented Programming Concepts: GeeksforGeeks OOPs
5. Green University Official Website (for route references)