FACULTY OF COMPUTING AND INFORMATICS
TERM 2430
OBJECT ORIENTED PROGRAMMING
ASSIGNMENT
Lecturer: Suraya Nurain Binti Kalid
Lab Section: TL2L
Group 5 : Transportation Management Program

Chin Wen Hui (241DC2418Q)
Aw Kak Yi (241DC240VD)
Muhammad Aiman Danish bin Ahmad Amri (1221202276)
Muhammad Danish bin Zulkipli (1221202333)

1.0 Introduction

Welcome to the Transportation Management Program! This program is designed to efficiently manage information about various types of vehicles using the principles of object-oriented programming. Whether you're an automobile enthusiast, a vehicle rental service, or simply someone interested in organizing vehicle data, this program aims to simplify the process and provide a user-friendly interface for managing vehicle information.

1.1 Objectives

The main objectives of this program are to enable users to add, edit, and delete vehicles from a collection and display and search for specific vehicles based on different criteria. By leveraging the power of object-oriented programming, the program offers flexibility and extensibility, allowing for the management of diverse vehicle types with their unique attributes and functionalities.

1.2 Scope

The program supports three main types of transportation: Air, Land, and Sea. Each vehicle type is represented by a corresponding class, encapsulating the specific attributes and behaviors associated with that type. The program provides comprehensive features to handle vehicle information efficiently, making it suitable for personal and professional use.

1.3 Target Users

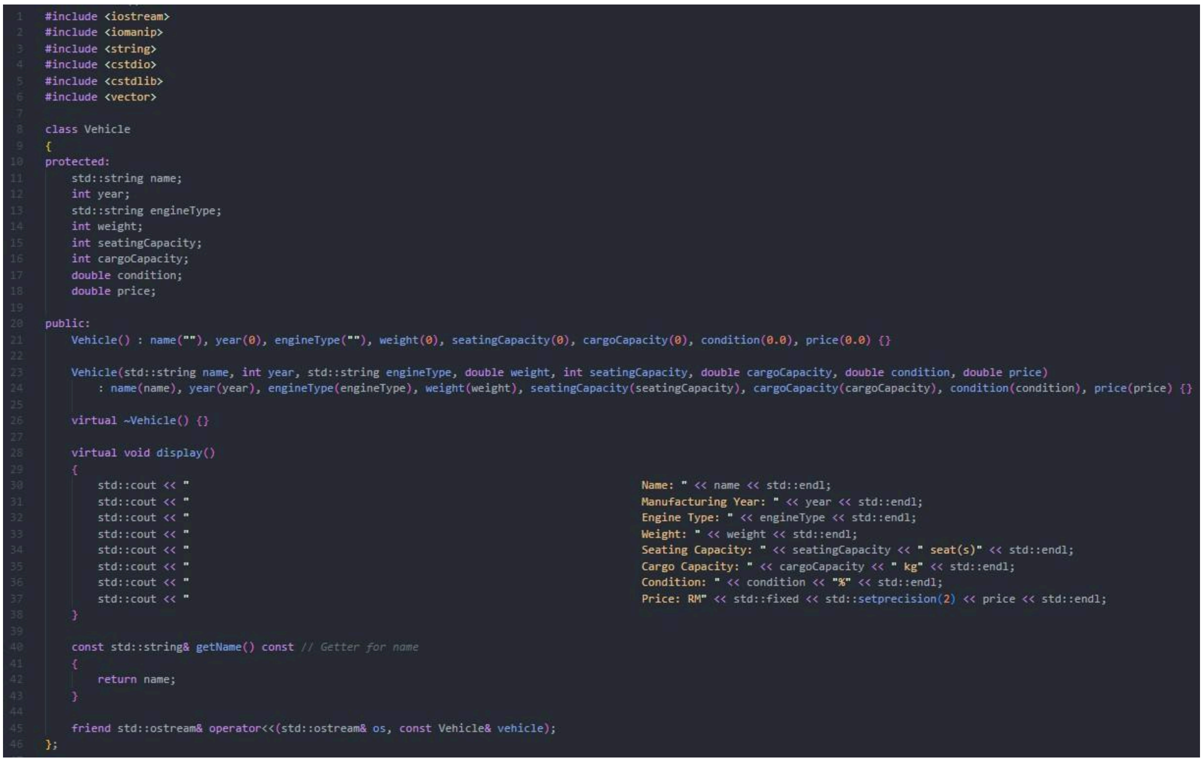
The program caters to a wide range of users, including automotive enthusiasts, vehicle rental companies, fleet managers, and anyone interested in organizing and managing vehicle data. Whether you need to keep track of your personal collection, maintain a database of available vehicles for rental services, or analyse and report on various aspects of vehicle information, this program can accommodate your needs.

1.4 Ending to a New Beginning

With its user-centric design and robust functionality, the Transportation Management Program aims to streamline the process of managing and organizing vehicle information. Whether you're an individual vehicle enthusiast or a business looking to optimize your transportation management processes, this program is here to help you efficiently handle your vehicle collection.

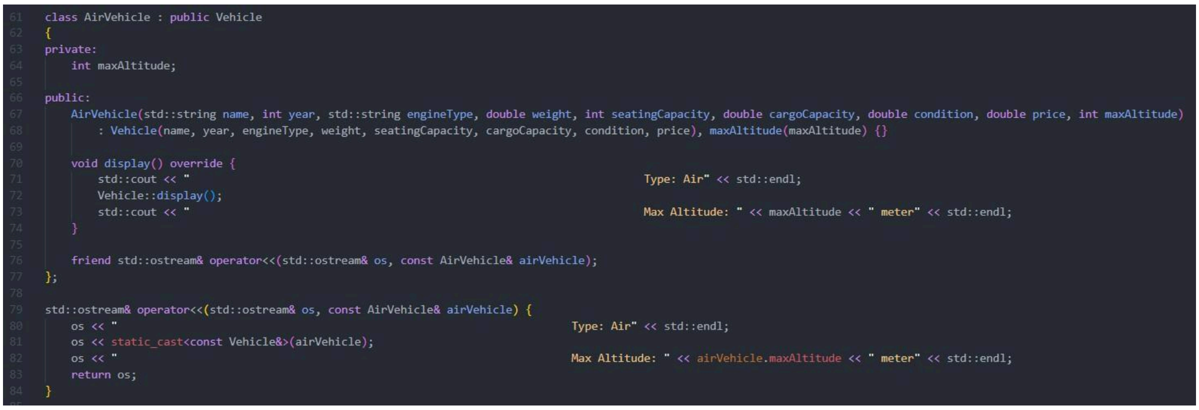
2.0 Program Features

1. Vehicle Class: This is the base class that represents a generic vehicle. It has various attributes such as name, year, engine type, weight, seating capacity, cargo capacity, condition, and price. The class provides a constructor to initialize these attributes and a display() function to print the vehicle details. It also includes a getName() function to retrieve the name attribute.



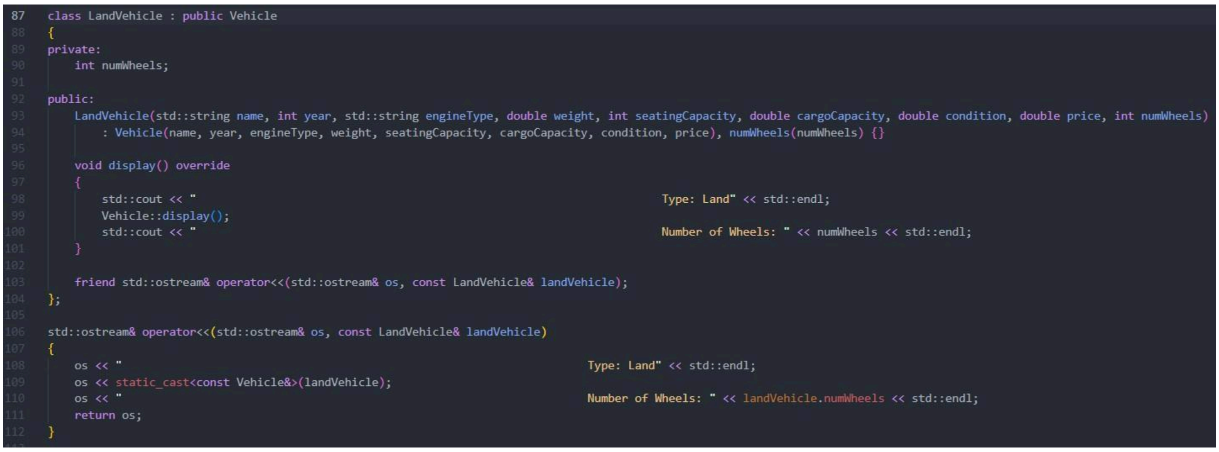
```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <vector>

class Vehicle
{
protected:
    std::string name;
    int year;
    std::string engineType;
    int weight;
    int seatingCapacity;
    int cargoCapacity;
    double condition;
    double price;

public:
    Vehicle() : name(""), year(0), engineType(""), weight(0), seatingCapacity(0), cargoCapacity(0), condition(0.0), price(0.0) {}

    Vehicle(std::string name, int year, std::string engineType, double weight, int seatingCapacity, double cargoCapacity, double condition, double price)
        : name(name), year(year), engineType(engineType), weight(weight), seatingCapacity(seatingCapacity), cargoCapacity(cargoCapacity), condition(condition), price(price) {}

    virtual ~Vehicle() {}

    virtual void display()
    {
        std::cout << "                                                                    Name: " << name << std::endl;
        std::cout << "                                                                    Manufacturing Year: " << year << std::endl;
        std::cout << "                                                                    Engine Type: " << engineType << std::endl;
        std::cout << "                                                                    Weight: " << weight << std::endl;
        std::cout << "                                                                    Seating Capacity: " << seatingCapacity << " seat(s)" << std::endl;
        std::cout << "                                                                    Cargo Capacity: " << cargoCapacity << " kg" << std::endl;
        std::cout << "                                                                    Condition: " << condition << "%" << std::endl;
        std::cout << "                                                                    Price: RM" << std::fixed << std::setprecision(2) << price << std::endl;
    }

    const std::string& getName() const // Getter for name
    {
        return name;
    }

    friend std::ostream& operator<<(std::ostream& os, const Vehicle& vehicle);
};
```

2. AirVehicle Class: This class is derived from the Vehicle class and represents an air vehicle. It adds an additional attribute called maxAltitude to represent the maximum altitude the air vehicle can reach. The class overrides the display() function to include the max altitude information.



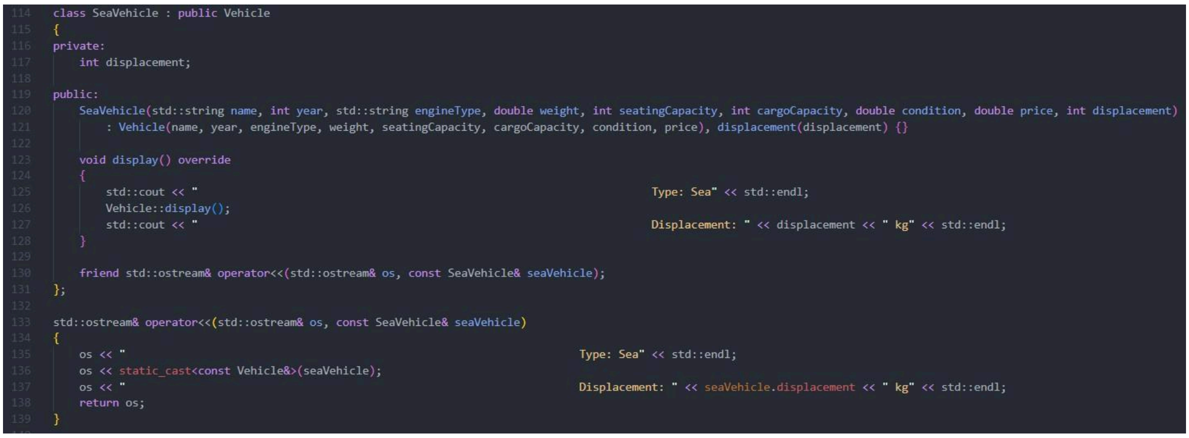
```cpp
class AirVehicle : public Vehicle
{
private:
    int maxAltitude;

public:
    AirVehicle(std::string name, int year, std::string engineType, double weight, int seatingCapacity, double cargoCapacity, double condition, double price, int maxAltitude)
        : Vehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price), maxAltitude(maxAltitude) {}

    void display() override {
        std::cout << "                                                                    Type: Air" << std::endl;
        Vehicle::display();
        std::cout << "                                                                    Max Altitude: " << maxAltitude << " meter" << std::endl;
    }

    friend std::ostream& operator<<(std::ostream& os, const AirVehicle& airVehicle);
};

std::ostream& operator<<(std::ostream& os, const AirVehicle& airVehicle) {
    os << "                                                                    Type: Air" << std::endl;
    os << static_cast<const Vehicle&>(airVehicle);
    os << "                                                                    Max Altitude: " << airVehicle.maxAltitude << " meter" << std::endl;
    return os;
}
```

3. LandVehicle Class: Similar to the AirVehicle class, the LandVehicle class is derived from the Vehicle class and represents a land vehicle. It adds an attribute called numWheels to represent the number of wheels the land vehicle has. The class overrides the display() function to include the number of wheels information.



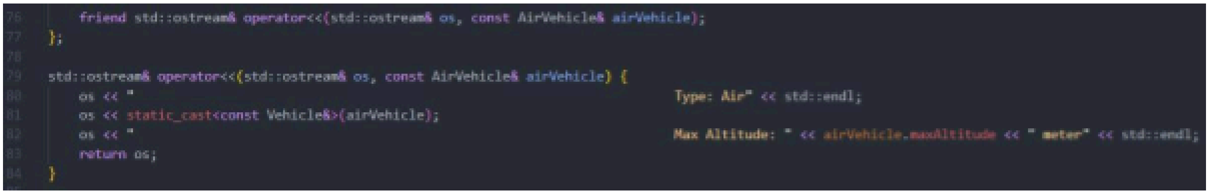
```cpp
class LandVehicle : public Vehicle
{
private:
    int numWheels;

public:
    LandVehicle(std::string name, int year, std::string engineType, double weight, int seatingCapacity, double cargoCapacity, double condition, double price, int numWheels)
        : Vehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price), numWheels(numWheels) {}

    void display() override
    {
        std::cout << "                                                                          Type: Land" << std::endl;
        Vehicle::display();
        std::cout << "                                                                          Number of Wheels: " << numWheels << std::endl;
    }

    friend std::ostream& operator<<(std::ostream& os, const LandVehicle& landVehicle);
};

std::ostream& operator<<(std::ostream& os, const LandVehicle& landVehicle)
{
    os << "                                                                       Type: Land" << std::endl;
    os << static_cast<const Vehicle&>(landVehicle);
    os << "                                                                       Number of Wheels: " << landVehicle.numWheels << std::endl;
    return os;
}
```

4. SeaVehicle Class: The SeaVehicle class is derived from the Vehicle class and represents a sea vehicle. It includes an attribute called displacement to represent the displacement of the sea vehicle. The class overrides the display() function to include the displacement information.

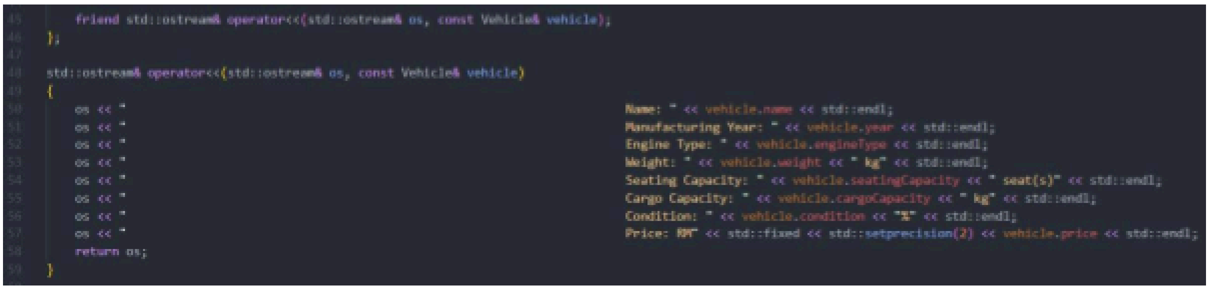


```cpp
class SeaVehicle : public Vehicle
{
private:
    int displacement;

public:
    SeaVehicle(std::string name, int year, std::string engineType, double weight, int seatingCapacity, int cargoCapacity, double condition, double price, int displacement)
        : Vehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price), displacement(displacement) {}

    void display() override
    {
        std::cout << "                                                                          Type: Sea" << std::endl;
        Vehicle::display();
        std::cout << "                                                                          Displacement: " << displacement << " kg" << std::endl;
    }

    friend std::ostream& operator<<(std::ostream& os, const SeaVehicle& seaVehicle);
};

std::ostream& operator<<(std::ostream& os, const SeaVehicle& seaVehicle)
{
    os << "                                                                       Type: Sea" << std::endl;
    os << static_cast<const Vehicle&>(seaVehicle);
    os << "                                                                       Displacement: " << seaVehicle.displacement << " kg" << std::endl;
    return os;
}
```

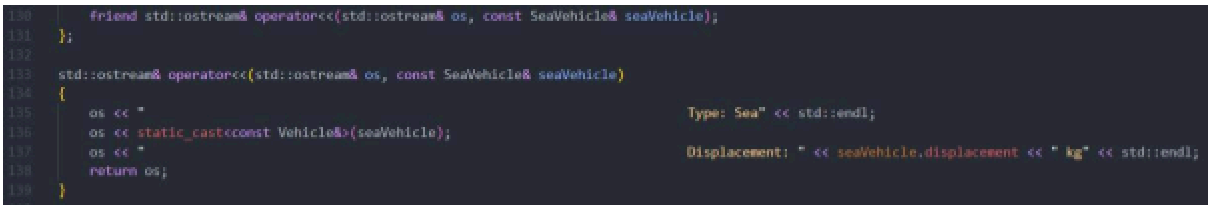
5. Operator Overloading: The program overloads the << operator for each vehicle class to allow printing the vehicle details using std::cout. It provides a formatted output for each type of vehicle, including the base class attributes and the additional attributes specific to each vehicle type.



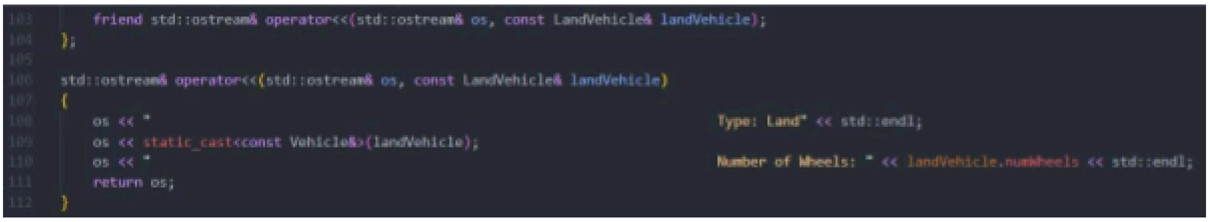
```
    friend std::ostream& operator<<(std::ostream& os, const AirVehicle& airVehicle);
};

std::ostream& operator<<(std::ostream& os, const AirVehicle& airVehicle) {
    os << "                                                      Type: Air" << std::endl;
    os << static_cast<const Vehicle&>(airVehicle);
    os << "                                                      Max Altitude: " << airVehicle.maxAltitude << " meter" << std::endl;
    return os;
}
```



```
    friend std::ostream& operator<<(std::ostream& os, const Vehicle& vehicle);
};

std::ostream& operator<<(std::ostream& os, const Vehicle& vehicle)
{
    os << "                                                   Name: " << vehicle.name << std::endl;
    os << "                                                   Manufacturing Year: " << vehicle.year << std::endl;
    os << "                                                   Engine Type: " << vehicle.engineType << std::endl;
    os << "                                                   Weight: " << vehicle.weight << " kg" << std::endl;
    os << "                                                   Seating Capacity: " << vehicle.seatingCapacity << " seat(s)" << std::endl;
    os << "                                                   Cargo Capacity: " << vehicle.cargoCapacity << " kg" << std::endl;
    os << "                                                   Condition: " << vehicle.condition << "%" << std::endl;
    os << "                                                   Price: RM" << std::fixed << std::setprecision(2) << vehicle.price << std::endl;
    return os;
}
```

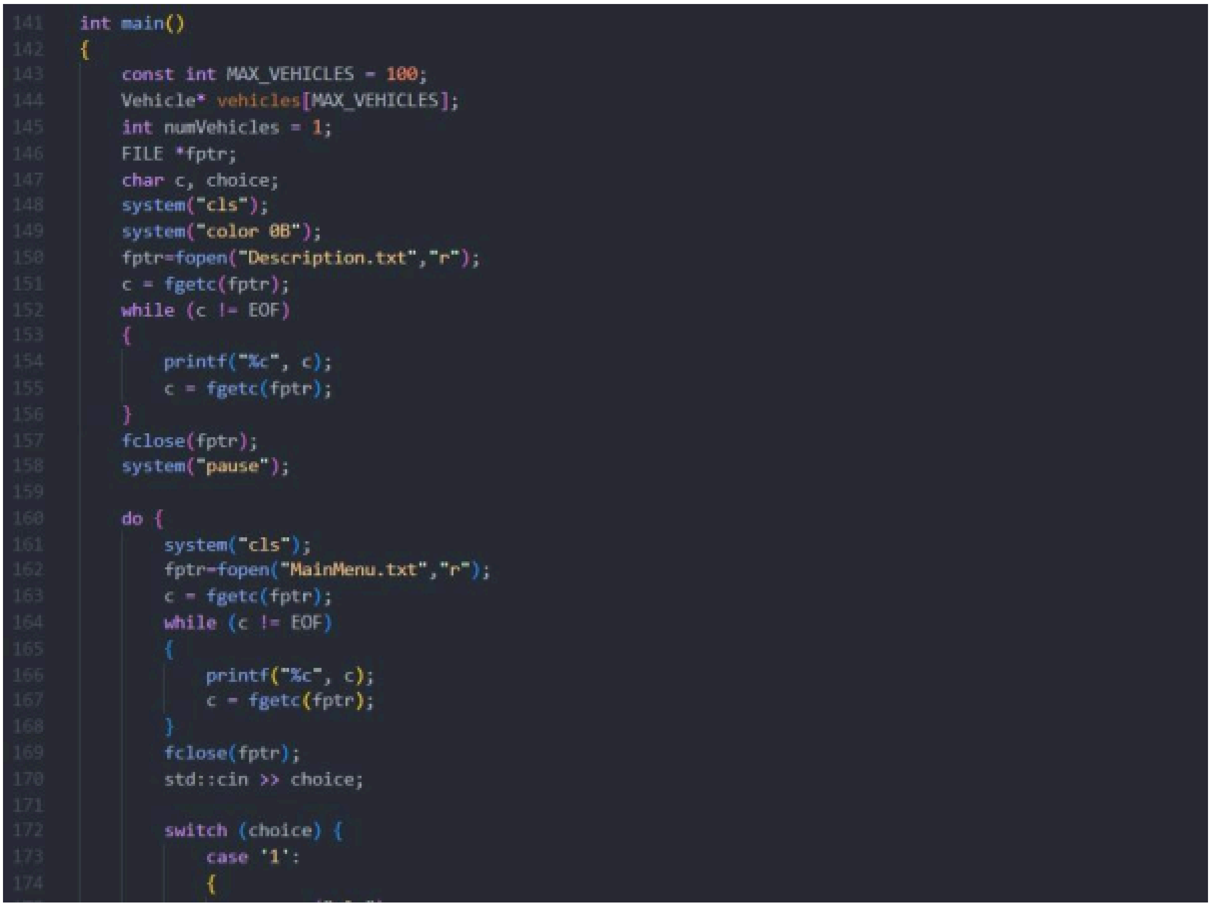


```
    friend std::ostream& operator<<(std::ostream& os, const SeaVehicle& seaVehicle);
};

std::ostream& operator<<(std::ostream& os, const SeaVehicle& seaVehicle)
{
    os << "                                                      Type: Sea" << std::endl;
    os << static_cast<const Vehicle&>(seaVehicle);
    os << "                                                      Displacement: " << seaVehicle.displacement << " kg" << std::endl;
    return os;
}
```

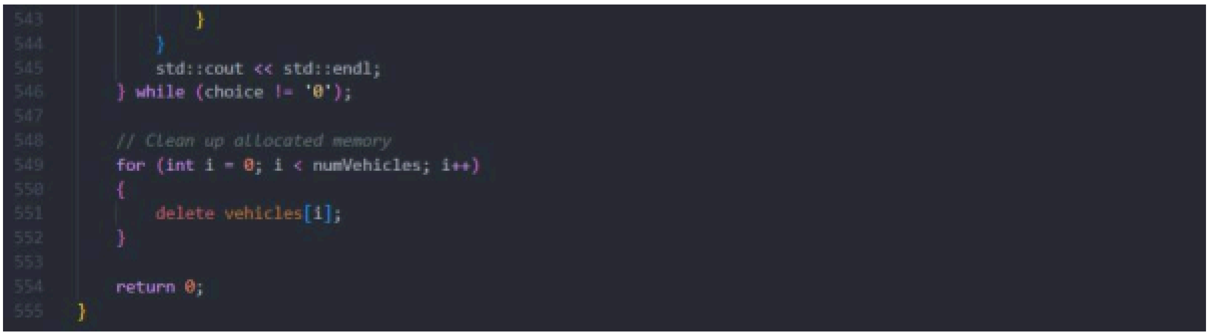


```
    friend std::ostream& operator<<(std::ostream& os, const LandVehicle& landVehicle);
};

std::ostream& operator<<(std::ostream& os, const LandVehicle& landVehicle)
{
    os << "                                                      Type: Land" << std::endl;
    os << static_cast<const Vehicle&>(landVehicle);
    os << "                                                      Number of Wheels: " << landVehicle.numWheels << std::endl;
    return os;
}
```

6. Main Function: The main function serves as the entry point of the program. It uses an array of pointers to the Vehicle base class to store multiple vehicles. It includes a constant MAX_VEHICLES to define the maximum number of vehicles the program can handle.

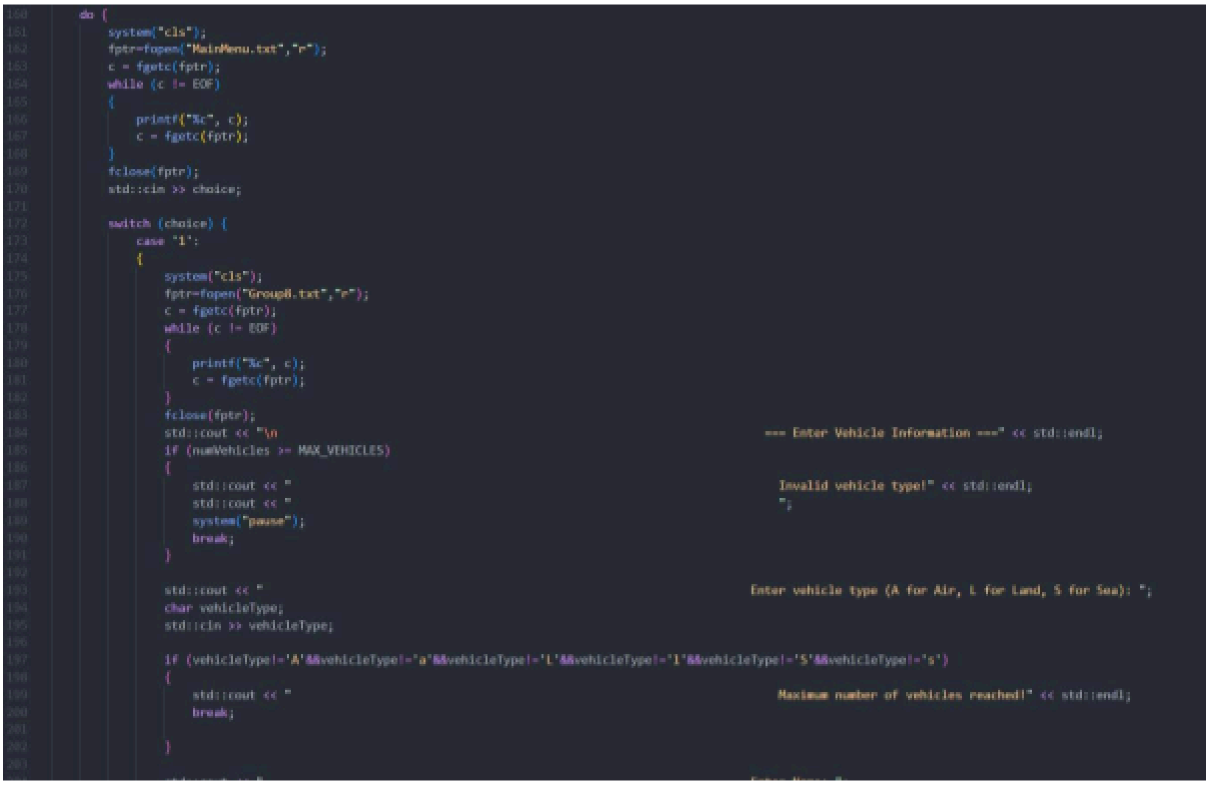


```
int main()
{
    const int MAX_VEHICLES = 100;
    Vehicle* vehicles[MAX_VEHICLES];
    int numVehicles = 1;
    FILE *fptr;
    char c, choice;
    system("cls");
    system("color 0B");
    fptr=fopen("Description.txt","r");
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    system("pause");

    do {
        system("cls");
        fptr=fopen("MainMenu.txt","r");
        c = fgetc(fptr);
        while (c != EOF)
        {
            printf("%c", c);
            c = fgetc(fptr);
        }
        fclose(fptr);
        std::cin >> choice;

        switch (choice) {
            case '1':
            {
```

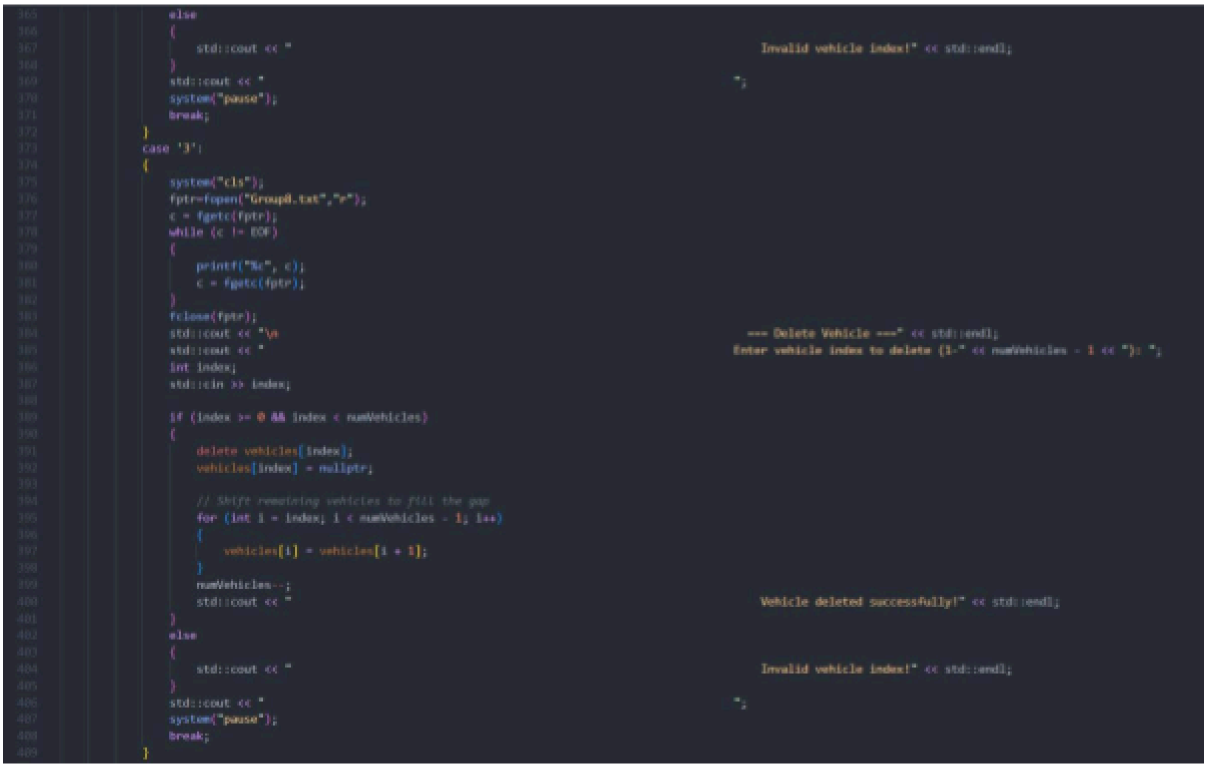


```
            }
        }
        std::cout << std::endl;
    } while (choice != '0');

    // Clean up allocated memory
    for (int i = 0; i < numVehicles; i++)
    {
        delete vehicles[i];
    }

    return 0;
}
```

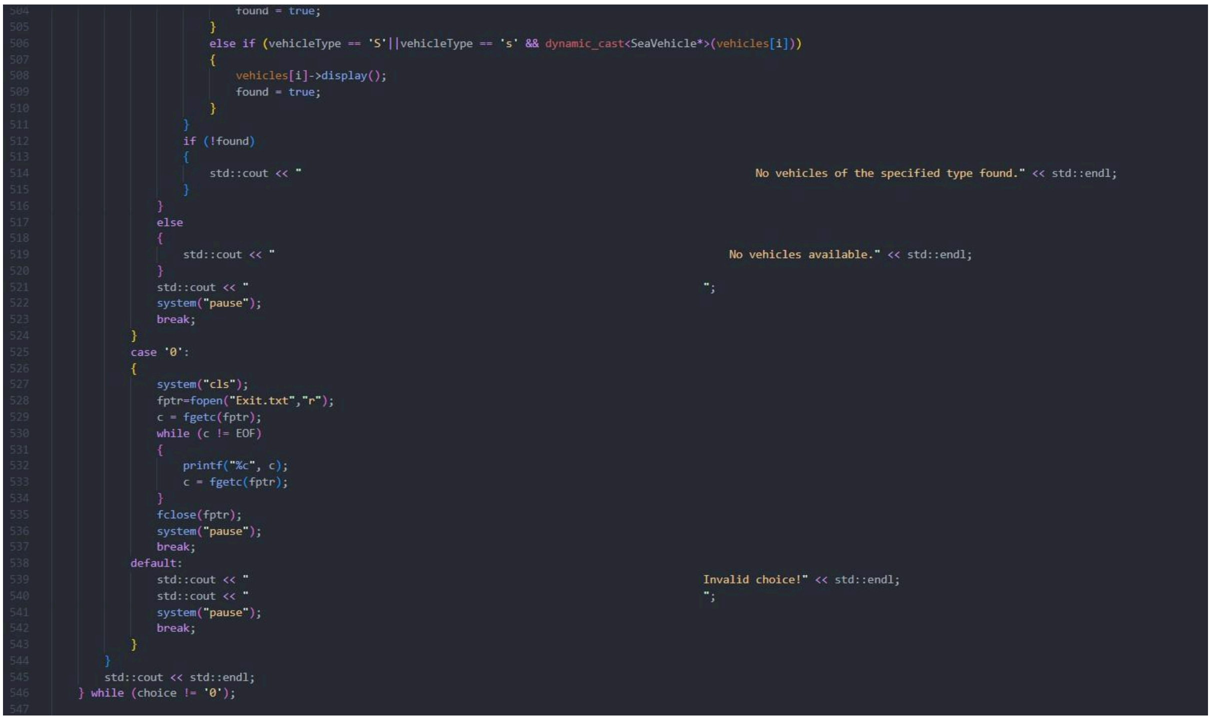
7. Menu System: The main function presents a menu to the user with options to add a vehicle, edit vehicle information, delete a vehicle, display all vehicles, and exit the program. It uses a do-while loop to repeatedly display the menu and perform the selected action until the user chooses to exit.



```
        do {
            system("cls");
            fptr=fopen("MainMenu.txt","r");
            c = fgetc(fptr);
            while (c != EOF)
            {
                printf("%c", c);
                c = fgetc(fptr);
            }
            fclose(fptr);
            std::cin >> choice;

            switch (choice) {
                case '1':
                {
                    system("cls");
                    fptr=fopen("Group8.txt","r");
                    c = fgetc(fptr);
                    while (c != EOF)
                    {
                        printf("%c", c);
                        c = fgetc(fptr);
                    }
                    fclose(fptr);
                    std::cout << "\n                                                                                    === Enter Vehicle Information ===" << std::endl;
                    if (numVehicles >= MAX_VEHICLES)
                    {
                        std::cout << "                                                                                Invalid vehicle type!" << std::endl;
                        std::cout << "                                                                                ";
                        system("pause");
                        break;
                    }

                    std::cout << "                                                                                Enter vehicle type (A for Air, L for Land, S for Sea): ";
                    char vehicleType;
                    std::cin >> vehicleType;

                    if (vehicleType!='A'&&vehicleType!='a'&&vehicleType!='L'&&vehicleType!='l'&&vehicleType!='S'&&vehicleType!='s')
                    {
                        std::cout << "                                                                                Maximum number of vehicles reached!" << std::endl;
                        break;

                    }
```

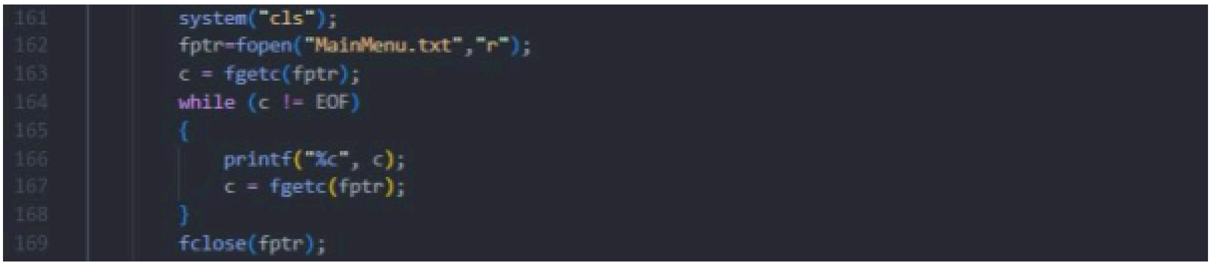


```
                    else
                    {
                        std::cout << "                                                                                Invalid vehicle index!" << std::endl;
                    }
                    std::cout << "                                                                                ";
                    system("pause");
                    break;
                }
                case '3':
                {
                    system("cls");
                    fptr=fopen("Group8.txt","r");
                    c = fgetc(fptr);
                    while (c != EOF)
                    {
                        printf("%c", c);
                        c = fgetc(fptr);
                    }
                    fclose(fptr);
                    std::cout << "\n                                                                                    === Delete Vehicle ===" << std::endl;
                    std::cout << "                                                                                Enter vehicle index to delete (1-" << numVehicles - 1 << "): ";
                    int index;
                    std::cin >> index;

                    if (index >= 0 && index < numVehicles)
                    {
                        delete vehicles[index];
                        vehicles[index] = nullptr;

                        // Shift remaining vehicles to fill the gap
                        for (int i = index; i < numVehicles - 1; i++)
                        {
                            vehicles[i] = vehicles[i + 1];
                        }
                        numVehicles--;
                        std::cout << "                                                                                Vehicle deleted successfully!" << std::endl;
                    }
                    else
                    {
                        std::cout << "                                                                                Invalid vehicle index!" << std::endl;
                    }
                    std::cout << "                                                                                ";
                    system("pause");
                    break;
                }
```

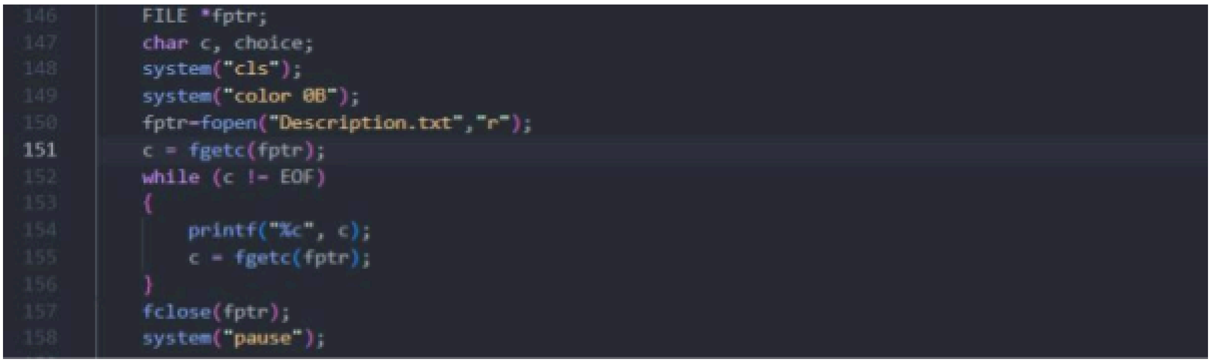


```
                        found = true;
                    }
                    else if (vehicleType == 'S'||vehicleType == 's' && dynamic_cast<SeaVehicle*>(vehicles[i]))
                    {
                        vehicles[i]->display();
                        found = true;
                    }
                }
                if (!found)
                {
                    std::cout << "                                                                     No vehicles of the specified type found." << std::endl;
                }
            }
            else
            {
                std::cout << "                                                                     No vehicles available." << std::endl;
            }
            std::cout << "                                                                     ";
            system("pause");
            break;
        }
        case '0':
        {
            system("cls");
            fptr=fopen("Exit.txt","r");
            c = fgetc(fptr);
            while (c != EOF)
            {
                printf("%c", c);
                c = fgetc(fptr);
            }
            fclose(fptr);
            system("pause");
            break;
        default:
            std::cout << "                                                                     Invalid choice!" << std::endl;
            std::cout << "                                                                     ";
            system("pause");
            break;
        }
    }
    std::cout << std::endl;
} while (choice != '0');
```

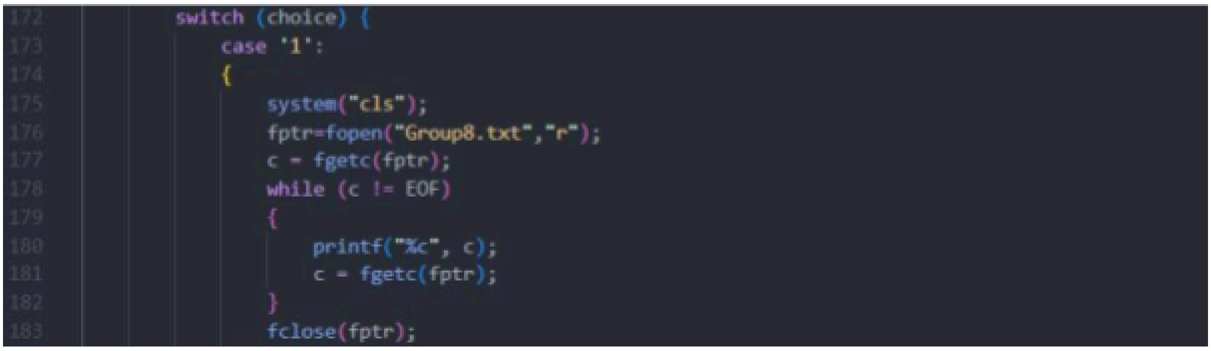
8. File Input: The program includes the use of file input to display additional information to the user. It reads and displays the contents of text files (Description.txt, MainMenu.txt, Group8.txt) using fopen and fgetc functions.



```
        system("cls");
        fptr=fopen("MainMenu.txt","r");
        c = fgetc(fptr);
        while (c != EOF)
        {
            printf("%c", c);
            c = fgetc(fptr);
        }
        fclose(fptr);
```

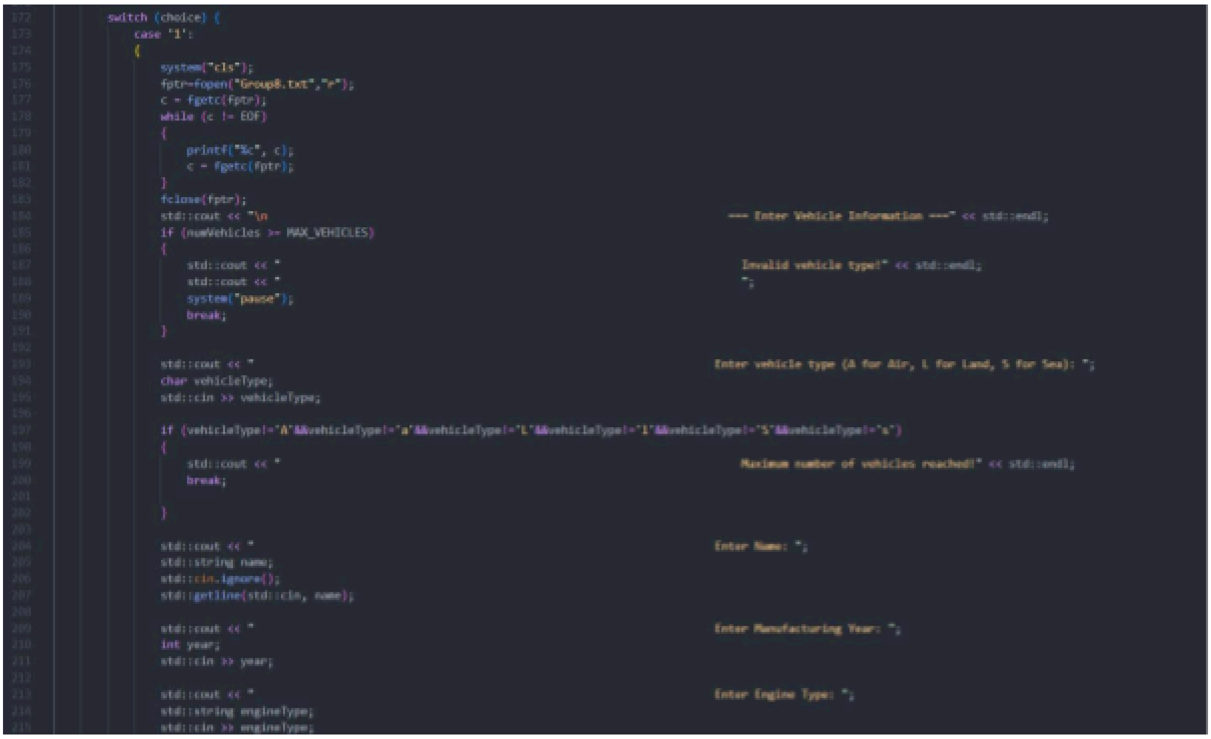


```
    FILE *fptr;
    char c, choice;
    system("cls");
    system("color 0B");
    fptr=fopen("Description.txt","r");
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    system("pause");
```

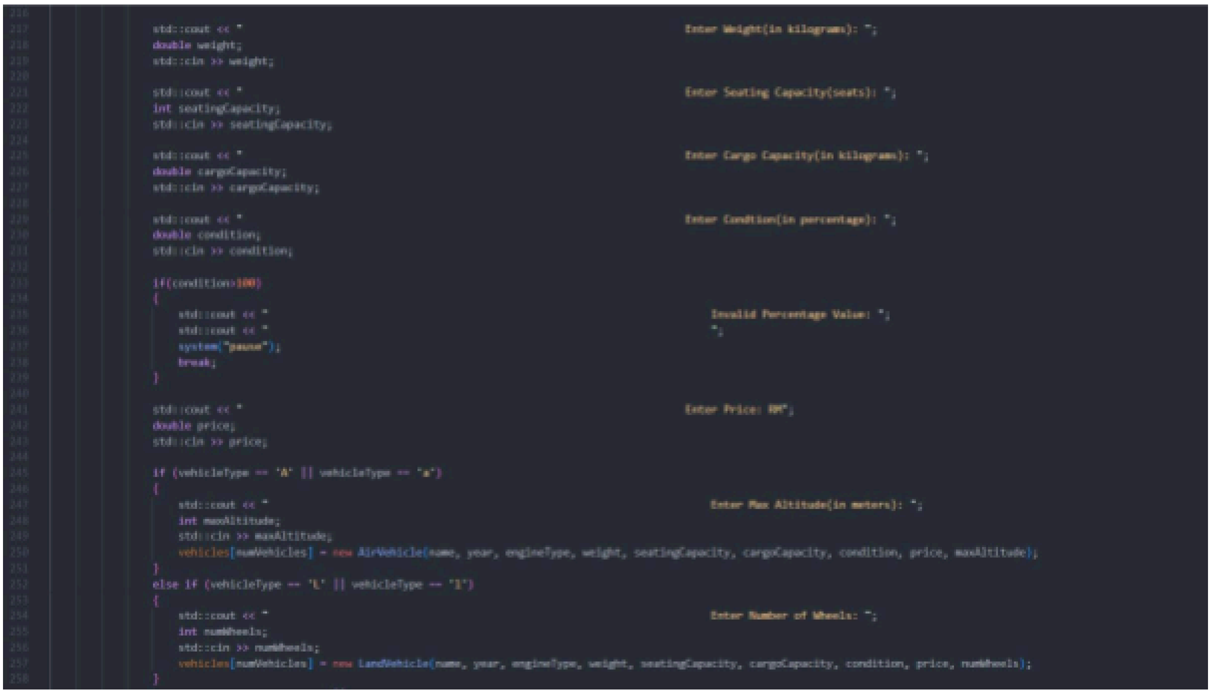


```
        switch (choice) {
            case '1':
            {
                system("cls");
                fptr=fopen("Group8.txt","r");
                c = fgetc(fptr);
                while (c != EOF)
                {
                    printf("%c", c);
                    c = fgetc(fptr);
                }
                fclose(fptr);
```

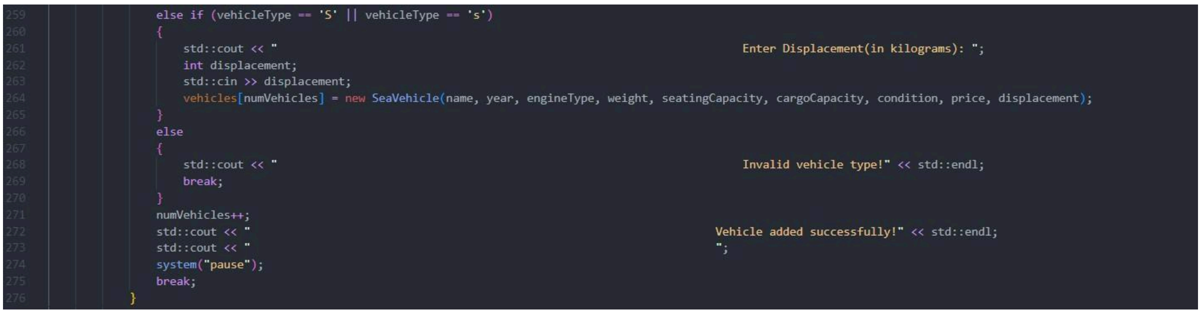
9. Adding Vehicles: When the user chooses to add a vehicle, they are prompted to enter the vehicle type (A for Air, L for Land, S for Sea) and the vehicle details such as name, year, engine type, weight, seating capacity, cargo capacity, condition, and price. Based on the vehicle type, the program dynamically creates an object of the respective derived class (AirVehicle, LandVehicle, SeaVehicle) and stores it in the vehicles array.



```
switch (choice) {
    case '1':
    {
        system("cls");
        fptr=fopen("Group8.txt","r");
        c = fgetc(fptr);
        while (c != EOF)
        {
            printf("%c", c);
            c = fgetc(fptr);
        }
        fclose(fptr);
        std::cout << "\n                                                --- Enter Vehicle Information ---" << std::endl;
        if (numVehicles >= MAX_VEHICLES)
        {
            std::cout << "                                              Invalid vehicle type!" << std::endl;
            std::cout << "                                              ";
            system("pause");
            break;
        }

        std::cout << "                                              Enter vehicle type (A for Air, L for Land, S for Sea): ";
        char vehicleType;
        std::cin >> vehicleType;

        if (vehicleType!='A'&&vehicleType!='a'&&vehicleType!='L'&&vehicleType!='l'&&vehicleType!='S'&&vehicleType!='s')
        {
            std::cout << "                                              Maximum number of vehicles reached!" << std::endl;
            break;

        }

        std::cout << "                                              Enter Name: ";
        std::string name;
        std::cin.ignore();
        std::getline(std::cin, name);

        std::cout << "                                              Enter Manufacturing Year: ";
        int year;
        std::cin >> year;

        std::cout << "                                              Enter Engine Type: ";
        std::string engineType;
        std::cin >> engineType;
```

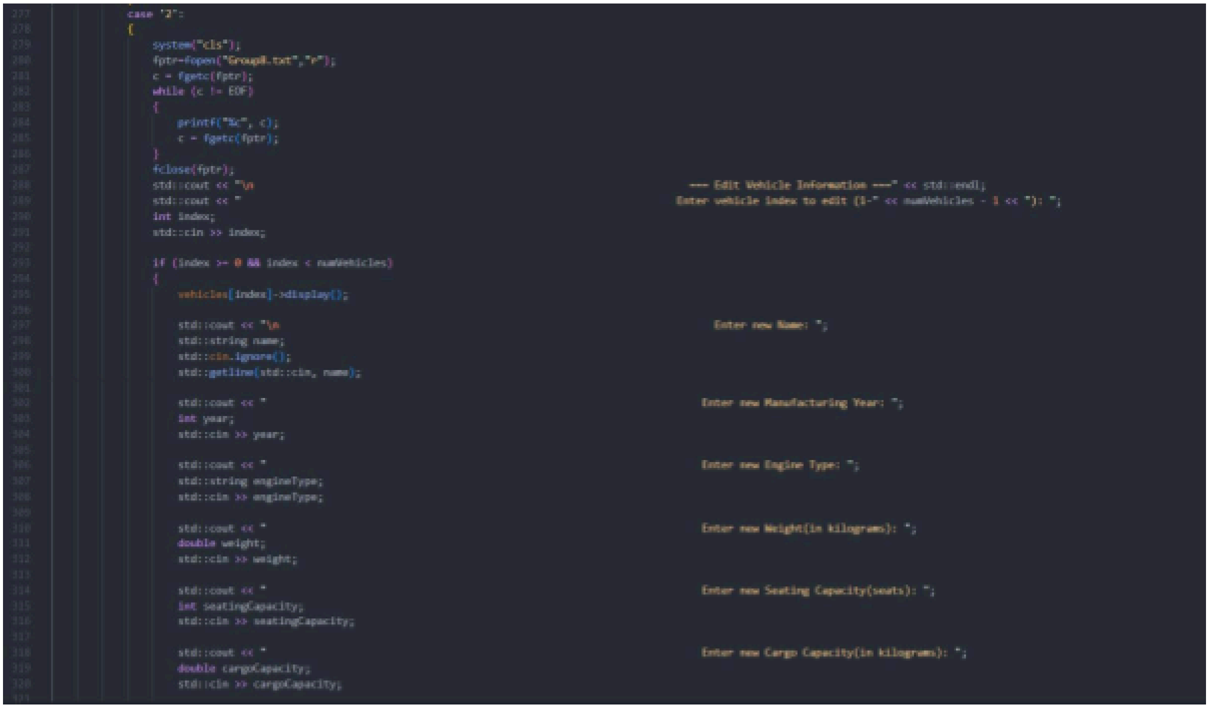


```

        std::cout << "                                              Enter Weight(in kilograms): ";
        double weight;
        std::cin >> weight;

        std::cout << "                                              Enter Seating Capacity(seats): ";
        int seatingCapacity;
        std::cin >> seatingCapacity;

        std::cout << "                                              Enter Cargo Capacity(in kilograms): ";
        double cargoCapacity;
        std::cin >> cargoCapacity;

        std::cout << "                                              Enter Condition(in percentage): ";
        double condition;
        std::cin >> condition;

        if(condition>100)
        {
            std::cout << "                                              Invalid Percentage Value: ";
            std::cout << "                                              ";
            system("pause");
            break;
        }

        std::cout << "                                              Enter Price: RM";
        double price;
        std::cin >> price;

        if (vehicleType == 'A' || vehicleType == 'a')
        {
            std::cout << "                                              Enter Max Altitude(in meters): ";
            int maxAltitude;
            std::cin >> maxAltitude;
            vehicles[numVehicles] = new AirVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, maxAltitude);
        }
        else if (vehicleType == 'L' || vehicleType == 'l')
        {
            std::cout << "                                              Enter Number of Wheels: ";
            int numWheels;
            std::cin >> numWheels;
            vehicles[numVehicles] = new LandVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, numWheels);
        }
```

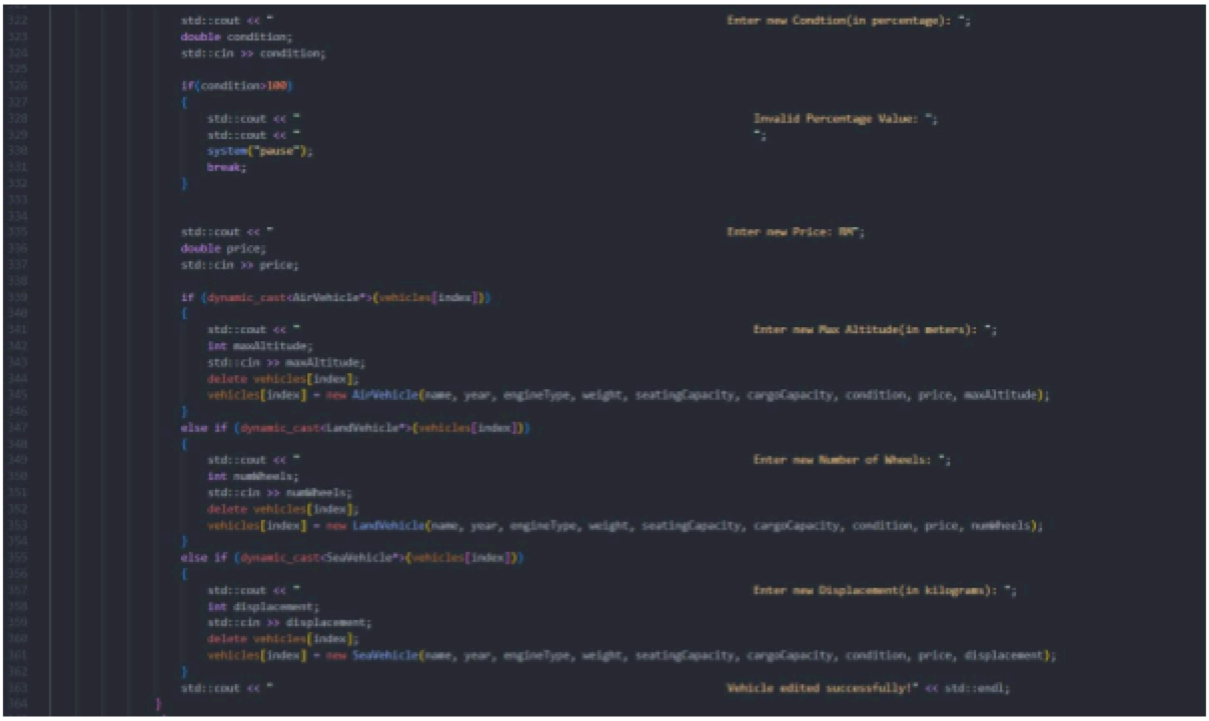


```cpp
                    else if (vehicleType == 'S' || vehicleType == 's')
                    {
                        std::cout << "                                                                      Enter Displacement(in kilograms): ";
                        int displacement;
                        std::cin >> displacement;
                        vehicles[numVehicles] = new SeaVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, displacement);
                    }
                    else
                    {
                        std::cout << "                                                                      Invalid vehicle type!" << std::endl;
                        break;
                    }
                    numVehicles++;
                    std::cout << "                                                                      Vehicle added successfully!" << std::endl;
                    std::cout << "                                                                      ";
                    system("pause");
                    break;
                }
```

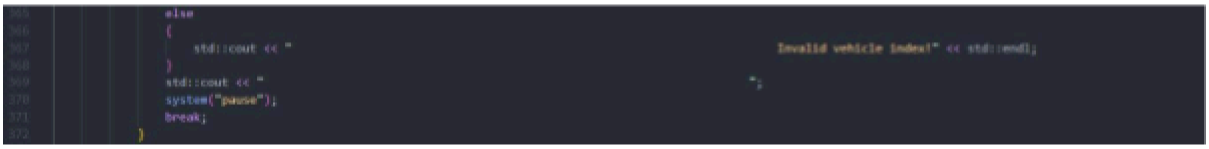
10. Editing Vehicles: If the user chooses to edit a vehicle, they need to enter the index of the vehicle they want to edit. The program checks if the index is valid and displays the current details of the vehicle. The user can then enter new information for the vehicle attributes, and the program updates the corresponding object.



```
                case '2':
                {
                    system("cls");
                    fptr=fopen("Group8.txt","r");
                    c = fgetc(fptr);
                    while (c != EOF)
                    {
                        printf("%c", c);
                        c = fgetc(fptr);
                    }
                    fclose(fptr);
                    std::cout << "\n                                                                    --- Edit Vehicle Information ---" << std::endl;
                    std::cout << "                                                                    Enter vehicle index to edit (1-" << numVehicles - 1 << "): ";
                    int index;
                    std::cin >> index;

                    if (index >= 0 && index < numVehicles)
                    {
                        vehicles[index]->display();

                        std::cout << "\n                                                                    Enter new Name: ";
                        std::string name;
                        std::cin.ignore();
                        std::getline(std::cin, name);

                        std::cout << "                                                                    Enter new Manufacturing Year: ";
                        int year;
                        std::cin >> year;

                        std::cout << "                                                                    Enter new Engine Type: ";
                        std::string engineType;
                        std::cin >> engineType;

                        std::cout << "                                                                    Enter new Weight(in kilograms): ";
                        double weight;
                        std::cin >> weight;

                        std::cout << "                                                                    Enter new Seating Capacity(seats): ";
                        int seatingCapacity;
                        std::cin >> seatingCapacity;

                        std::cout << "                                                                    Enter new Cargo Capacity(in kilograms): ";
                        double cargoCapacity;
                        std::cin >> cargoCapacity;
```

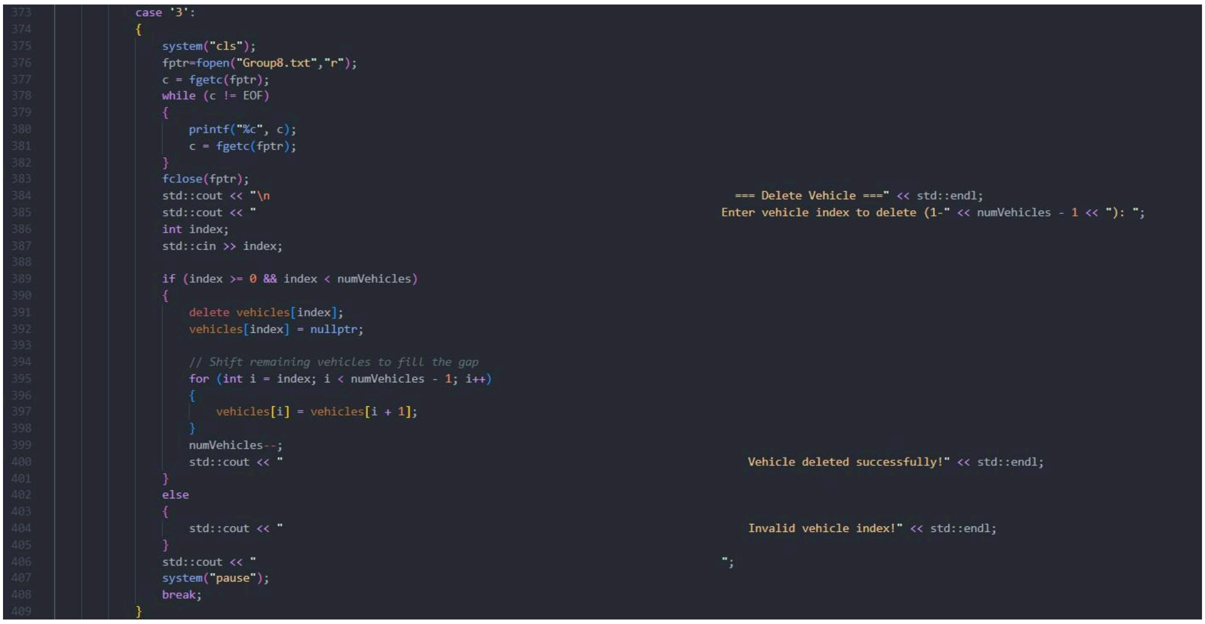


```
                        std::cout << "                                                                    Enter new Condtion(in percentage): ";
                        double condition;
                        std::cin >> condition;

                        if(condition>100)
                        {
                            std::cout << "                                                                    Invalid Percentage Value: ";
                            std::cout << "                                                                    ";
                            system("pause");
                            break;
                        }


                        std::cout << "                                                                    Enter new Price: RM";
                        double price;
                        std::cin >> price;

                        if (dynamic_cast<AirVehicle*>(vehicles[index]))
                        {
                            std::cout << "                                                                    Enter new Max Altitude(in meters): ";
                            int maxAltitude;
                            std::cin >> maxAltitude;
                            delete vehicles[index];
                            vehicles[index] = new AirVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, maxAltitude);
                        }
                        else if (dynamic_cast<LandVehicle*>(vehicles[index]))
                        {
                            std::cout << "                                                                    Enter new Number of Wheels: ";
                            int numWheels;
                            std::cin >> numWheels;
                            delete vehicles[index];
                            vehicles[index] = new LandVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, numWheels);
                        }
                        else if (dynamic_cast<SeaVehicle*>(vehicles[index]))
                        {
                            std::cout << "                                                                    Enter new Displacement(in kilograms): ";
                            int displacement;
                            std::cin >> displacement;
                            delete vehicles[index];
                            vehicles[index] = new SeaVehicle(name, year, engineType, weight, seatingCapacity, cargoCapacity, condition, price, displacement);
                        }
                        std::cout << "                                                                    Vehicle edited successfully!" << std::endl;
                    }
```



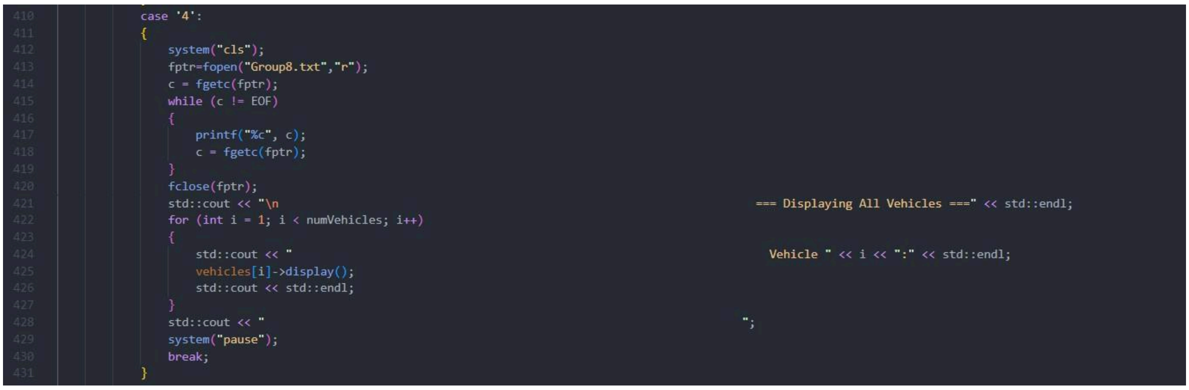
```
                    else
                    {
                        std::cout << "                                                                    Invalid vehicle index!" << std::endl;
                    }
                    std::cout << "                                                                    ";
                    system("pause");
                    break;
                }
```

11. Deleting Vehicles: When the user selects the option to delete a vehicle, they are prompted to enter the index of the vehicle they want to delete. The program checks if the index is valid, deletes the corresponding object, and shifts the remaining vehicles in the array to fill the gap.



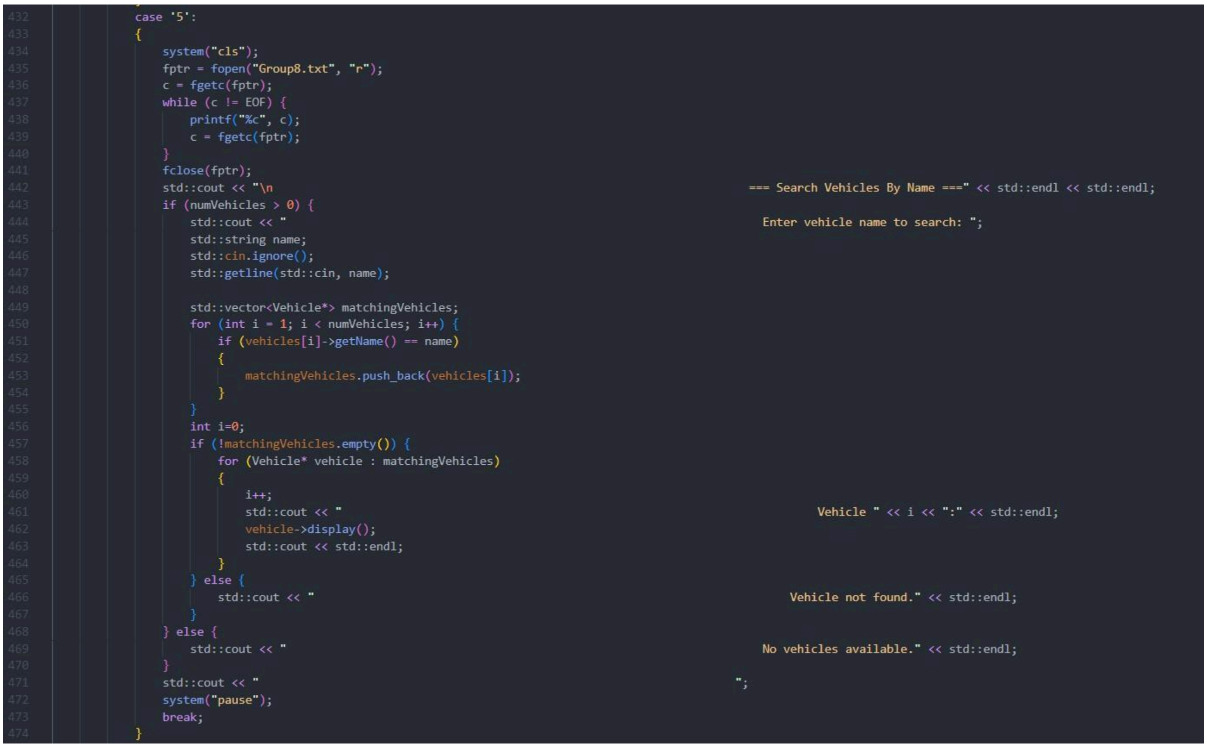
```
case '3':
{
    system("cls");
    fptr=fopen("Group8.txt","r");
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    std::cout << "\n                                                                      === Delete Vehicle ===" << std::endl;
    std::cout << "                                                                      Enter vehicle index to delete (1-" << numVehicles - 1 << "): ";
    int index;
    std::cin >> index;

    if (index >= 0 && index < numVehicles)
    {
        delete vehicles[index];
        vehicles[index] = nullptr;

        // Shift remaining vehicles to fill the gap
        for (int i = index; i < numVehicles - 1; i++)
        {
            vehicles[i] = vehicles[i + 1];
        }
        numVehicles--;
        std::cout << "                                                                      Vehicle deleted successfully!" << std::endl;
    }
    else
    {
        std::cout << "                                                                      Invalid vehicle index!" << std::endl;
    }
    std::cout << "                                                                      ";
    system("pause");
    break;
}
```

12. Displaying Vehicles: The program provides an option to display all the vehicles. It iterates over the vehicles array and calls the display() function for each non-null object, which prints the details of each vehicle.



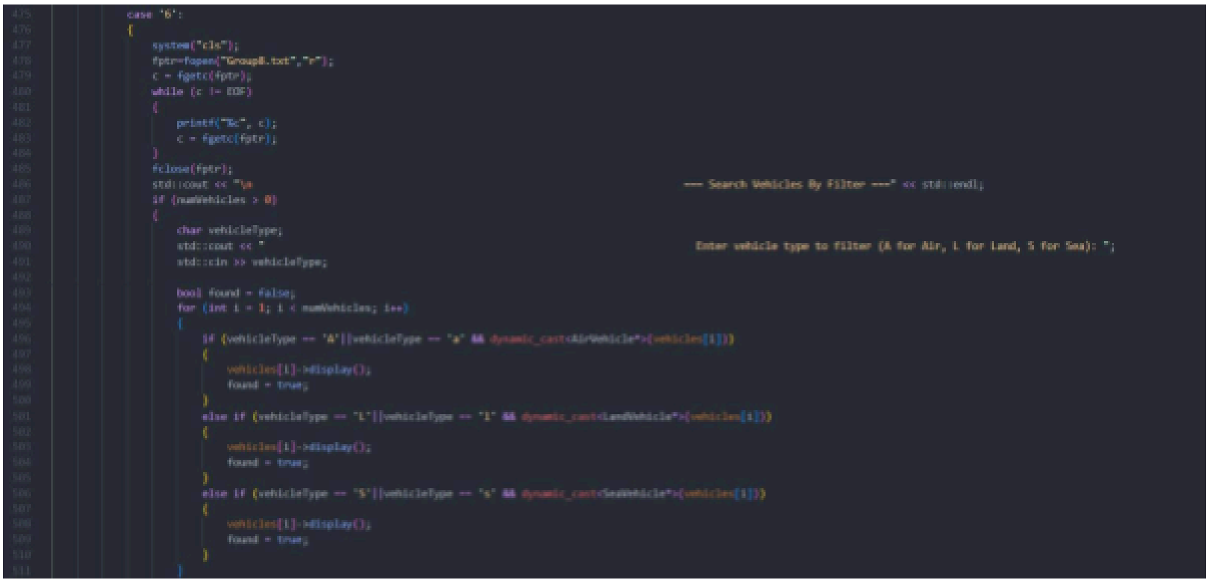
```
case '4':
{
    system("cls");
    fptr=fopen("Group8.txt","r");
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    std::cout << "\n                                                                       === Displaying All Vehicles ===" << std::endl;
    for (int i = 1; i < numVehicles; i++)
    {
        std::cout << "                                                                       Vehicle " << i << ":" << std::endl;
        vehicles[i]->display();
        std::cout << std::endl;
    }
    std::cout << "                                                                       ";
    system("pause");
    break;
}
```

13. Search by Name: This option allows the user to search for a vehicle by its name. When the user selects the search by name option from the menu, they are prompted to enter the name of the vehicle they want to search for. The program then iterates over the vehicles array and checks if the name of each vehicle matches the search query.
    - If a match is found, the program displays the details of the matching vehicle using the display() function.
    - If no match is found, the program notifies the user that the vehicle was not found.
    - This search option allows users to find a specific vehicle by its name, which can be useful when they know the name of the vehicle they are looking for.

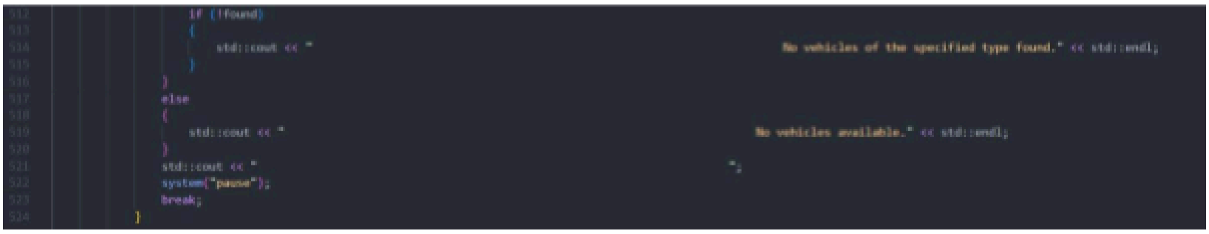


```
case '5':
{
    system("cls");
    fptr = fopen("Group8.txt", "r");
    c = fgetc(fptr);
    while (c != EOF) {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    std::cout << "\n                                                                      === Search Vehicles By Name ===" << std::endl << std::endl;
    if (numVehicles > 0) {
        std::cout << "                                                                      Enter vehicle name to search: ";
        std::string name;
        std::cin.ignore();
        std::getline(std::cin, name);

        std::vector<Vehicle*> matchingVehicles;
        for (int i = 1; i < numVehicles; i++) {
            if (vehicles[i]->getName() == name)
            {
                matchingVehicles.push_back(vehicles[i]);
            }
        }
        int i=0;
        if (!matchingVehicles.empty()) {
            for (Vehicle* vehicle : matchingVehicles)
            {
                i++;
                std::cout << "                                                                      Vehicle " << i << ":" << std::endl;
                vehicle->display();
                std::cout << std::endl;
            }
        } else {
            std::cout << "                                                                      Vehicle not found." << std::endl;
        }
    } else {
        std::cout << "                                                                      No vehicles available." << std::endl;
    }
    std::cout << "                                                                      ";
    system("pause");
    break;
}
```

14. Search by Type: This option allows the user to search for vehicles of a specific type. When the user selects the search by type option from the menu, they are prompted to enter the type of vehicle they want to search for. The program then iterates over the vehicles array and checks if the type of each vehicle matches the search query.
    - If one or more matches are found, the program displays the details of the matching vehicles using the display() function.
    - If no matches are found, the program notifies the user that no vehicles of the specified type were found.
    - This search option enables users to find vehicles based on their type, which can be helpful when they are interested in a particular category of vehicles.

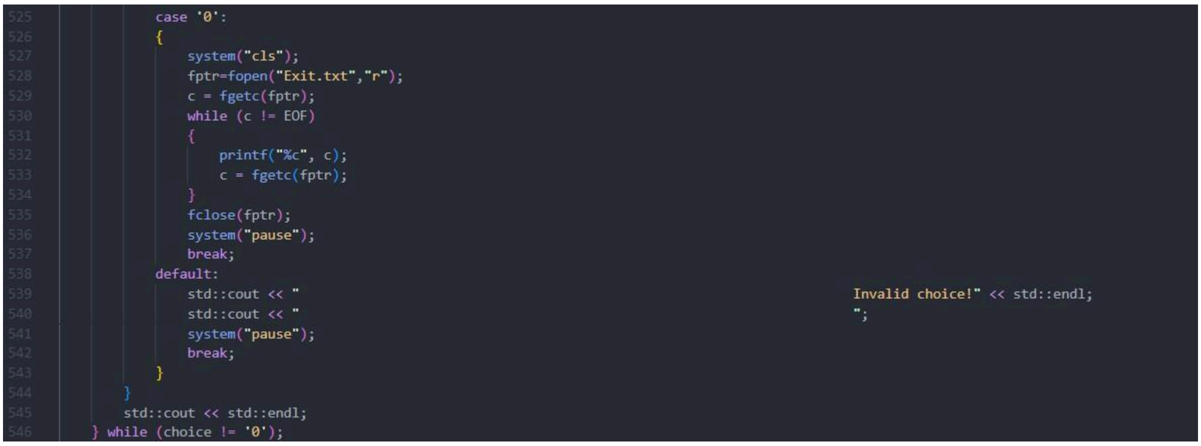


```
case '6':
{
    system("cls");
    fptr=fopen("Group8.txt","r");
    c = fgetc(fptr);
    while (c != EOF)
    {
        printf("%c", c);
        c = fgetc(fptr);
    }
    fclose(fptr);
    std::cout << "\n                                                      --- Search Vehicles By Filter ---" << std::endl;
    if (numVehicles > 0)
    {
        char vehicleType;
        std::cout << "                                                      Enter vehicle type to filter (A for Air, L for Land, S for Sea): ";
        std::cin >> vehicleType;

        bool found = false;
        for (int i = 1; i < numVehicles; i++)
        {
            if (vehicleType == 'A'||vehicleType == 'a' && dynamic_cast<AirVehicle*>(vehicles[i]))
            {
                vehicles[i]->display();
                found = true;
            }
            else if (vehicleType == 'L'||vehicleType == 'l' && dynamic_cast<LandVehicle*>(vehicles[i]))
            {
                vehicles[i]->display();
                found = true;
            }
            else if (vehicleType == 'S'||vehicleType == 's' && dynamic_cast<SeaVehicle*>(vehicles[i]))
            {
                vehicles[i]->display();
                found = true;
            }
        }
```



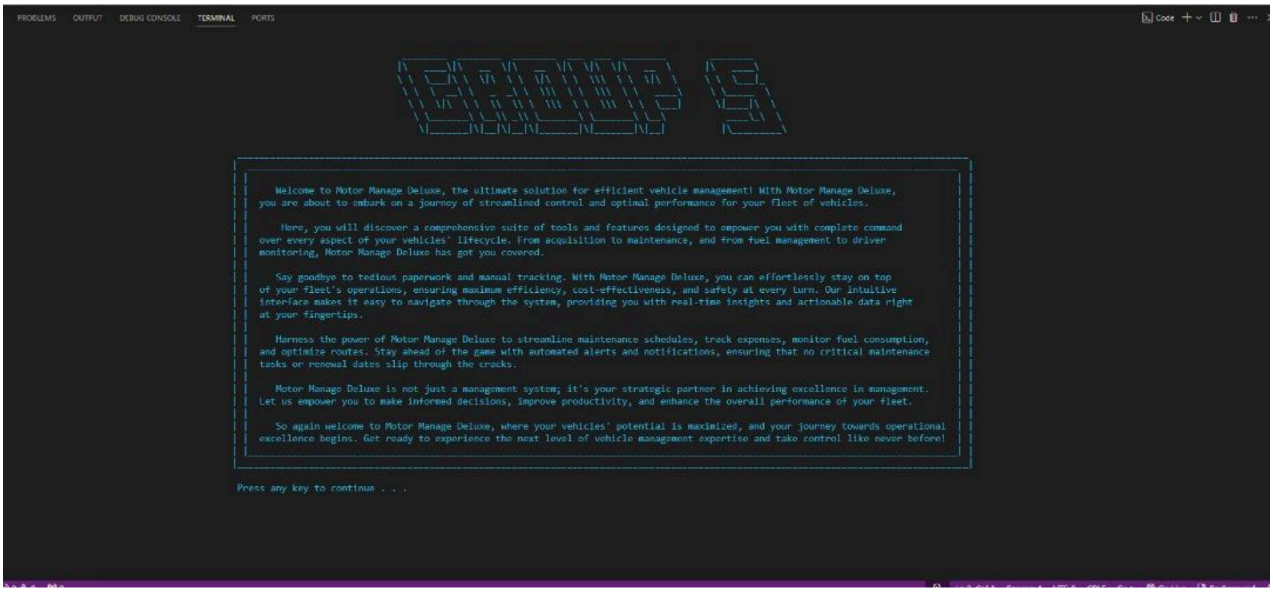
```
        if (!found)
        {
            std::cout << "                                                      No vehicles of the specified type found." << std::endl;
        }
    }
    else
    {
        std::cout << "                                                      No vehicles available." << std::endl;
    }
    std::cout << "                                                      ";
    system("pause");
    break;
}
```

15. Exit: The exit feature allows the user to gracefully terminate the program. When the user selects the exit option from the menu, the program ends the execution and the user is returned to the operating system or the calling program.Exiting the program is a standard way to conclude the execution when the user has completed their tasks or wants to terminate the program intentionally.

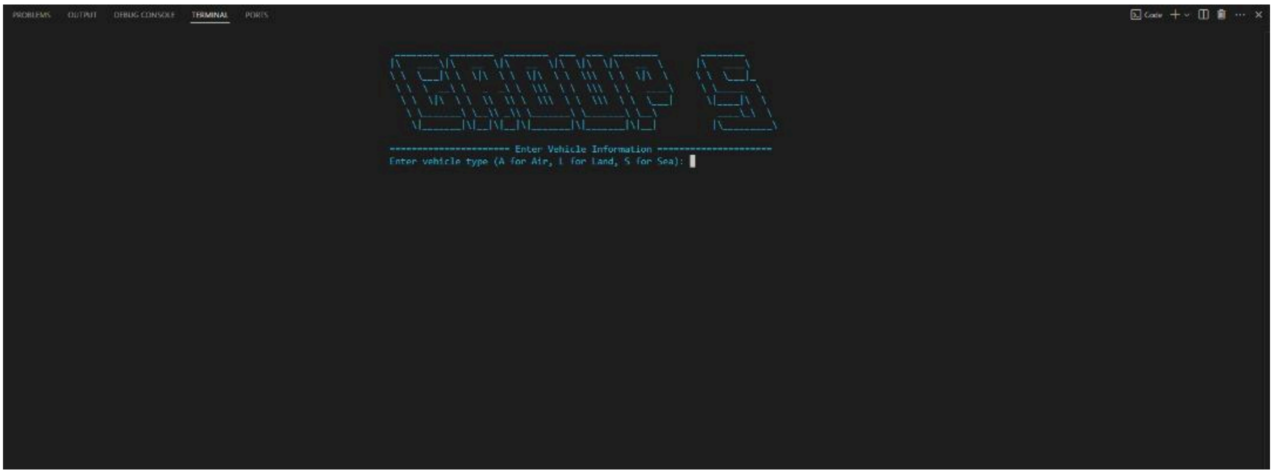


```
        case '0':
        {
            system("cls");
            fptr=fopen("Exit.txt","r");
            c = fgetc(fptr);
            while (c != EOF)
            {
                printf("%c", c);
                c = fgetc(fptr);
            }
            fclose(fptr);
            system("pause");
            break;
        default:
            std::cout << "                                                                     Invalid choice!" << std::endl;
            std::cout << "                                                                     ";
            system("pause");
            break;
        }
    }
    std::cout << std::endl;
} while (choice != '0');
```

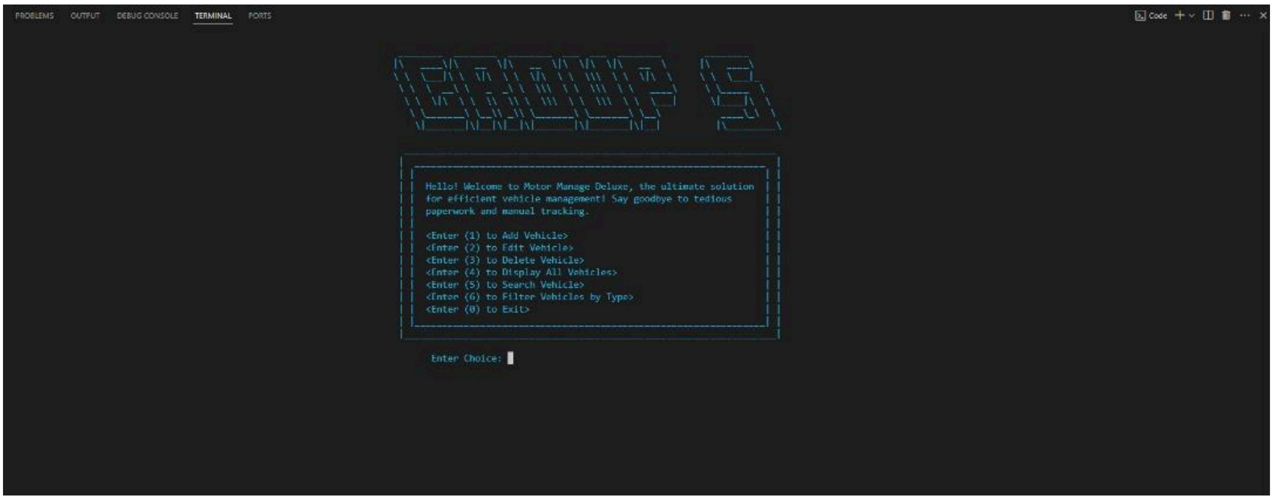
Output of the program



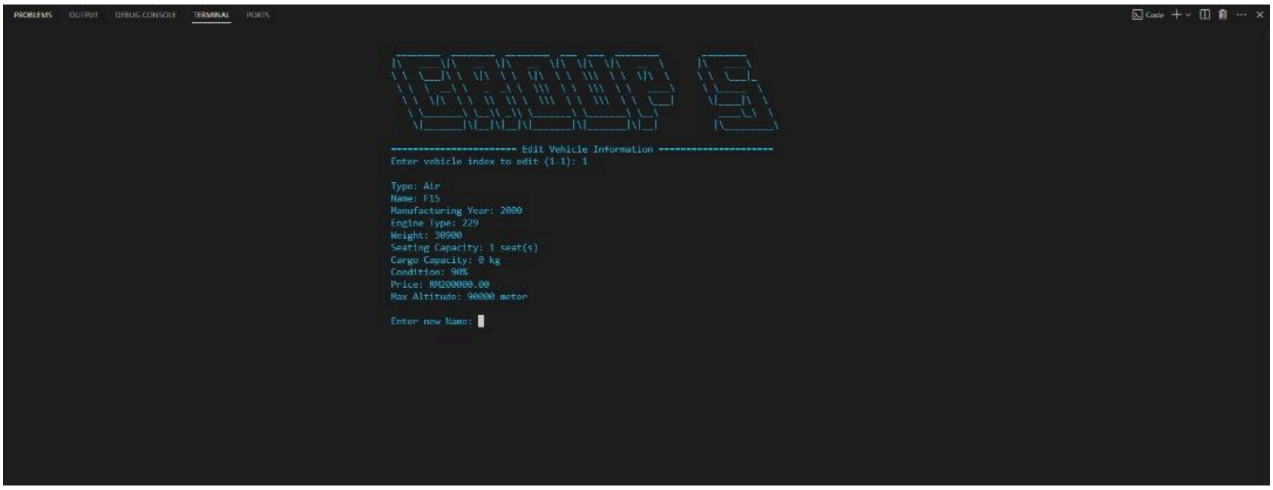
```
   Welcome to Motor Manage Deluxe, the ultimate solution for efficient vehicle management! With Motor Manage Deluxe,
you are about to embark on a journey of streamlined control and optimal performance for your fleet of vehicles.

    Here, you will discover a comprehensive suite of tools and features designed to empower you with complete command
over every aspect of your vehicles' lifecycle. From acquisition to maintenance, and from fuel management to driver
monitoring, Motor Manage Deluxe has got you covered.

   Say goodbye to tedious paperwork and manual tracking. With Motor Manage Deluxe, you can effortlessly stay on top
of your fleet's operations, ensuring maximum efficiency, cost-effectiveness, and safety at every turn. Our intuitive
interface makes it easy to navigate through the system, providing you with real-time insights and actionable data right
at your fingertips.

   Harness the power of Motor Manage Deluxe to streamline maintenance schedules, track expenses, monitor fuel consumption,
and optimize routes. Stay ahead of the game with automated alerts and notifications, ensuring that no critical maintenance
tasks or renewal dates slip through the cracks.

   Motor Manage Deluxe is not just a management system; it's your strategic partner in achieving excellence in management.
Let us empower you to make informed decisions, improve productivity, and enhance the overall performance of your fleet.

   So again welcome to Motor Manage Deluxe, where your vehicles' potential is maximized, and your journey towards operational
excellence begins. Get ready to experience the next level of vehicle management expertise and take control like never before!

Press any key to continue . . .
```



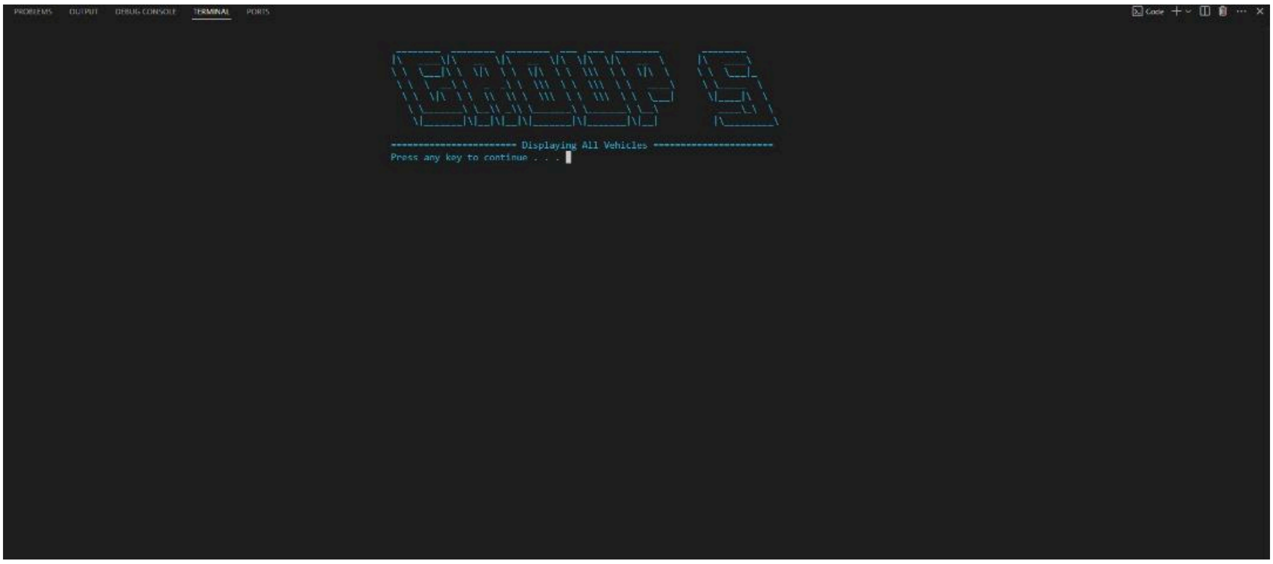
```
--------------------- Enter Vehicle Information ---------------------
Enter vehicle type (A for Air, L for Land, S for Sea):
```



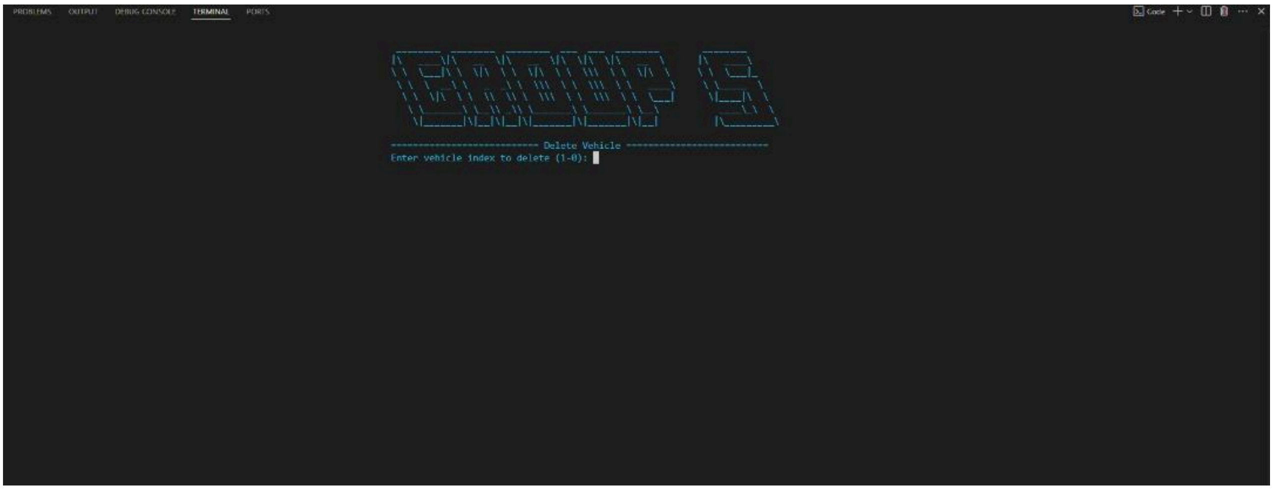
```
Hello! Welcome to Motor Manage Deluxe, the ultimate solution
for efficient vehicle management! Say goodbye to tedious
paperwork and manual tracking.

<Enter (1) to Add Vehicle>
<Enter (2) to Edit Vehicle>
<Enter (3) to Delete Vehicle>
<Enter (4) to Display All Vehicles>
<Enter (5) to Search Vehicle>
<Enter (6) to Filter Vehicles by Type>
<Enter (0) to Exit>

  Enter Choice:
```



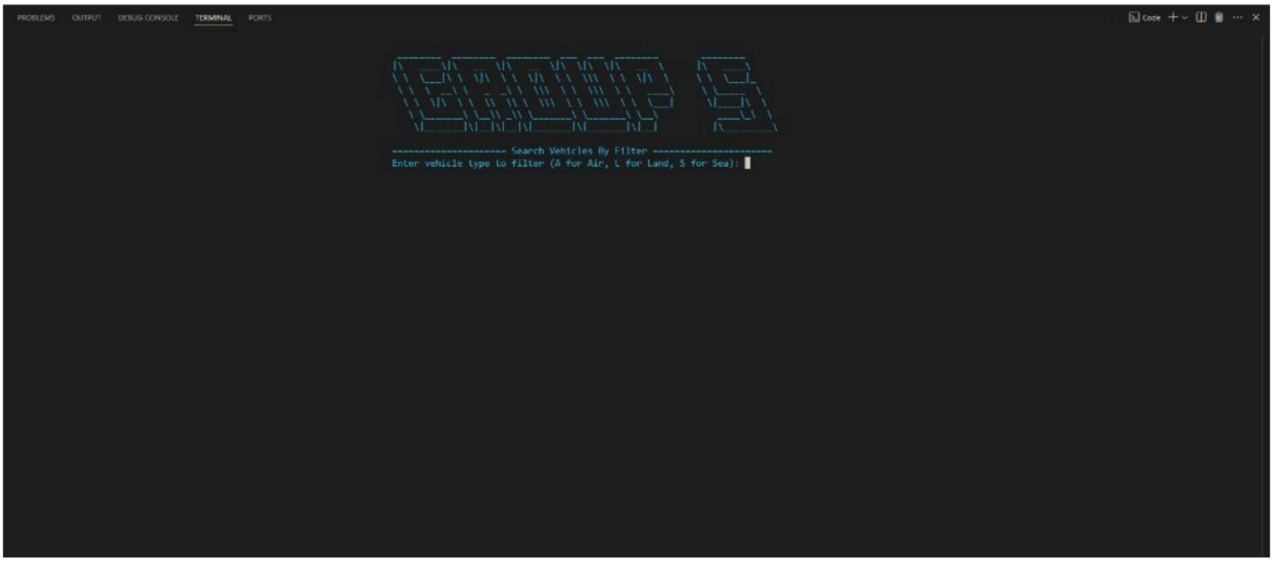
```
===================== Edit Vehicle Information =====================
Enter vehicle index to edit (1-1): 1

Type: Air
Name: F15
Manufacturing Year: 2000
Engine Type: 229
Weight: 30900
Seating Capacity: 1 seat(s)
Cargo Capacity: 0 kg
Condition: 90%
Price: RM200000.00
Max Altitude: 90000 meter

Enter new Name: 
```



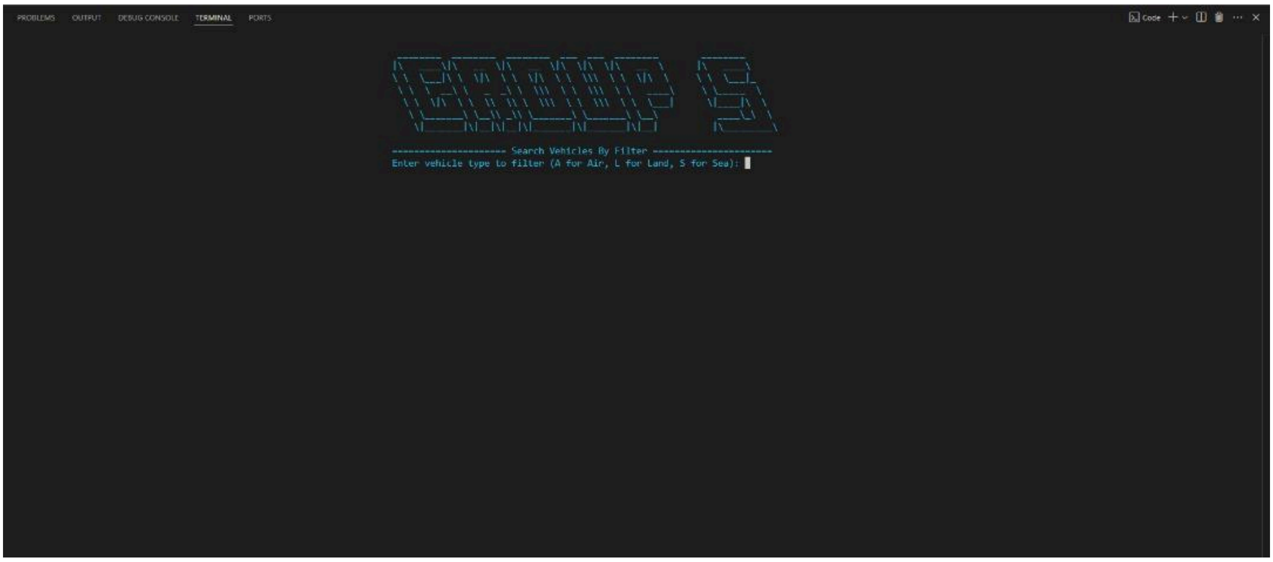
```
===================== Displaying All Vehicles =====================
Press any key to continue . . . 
```



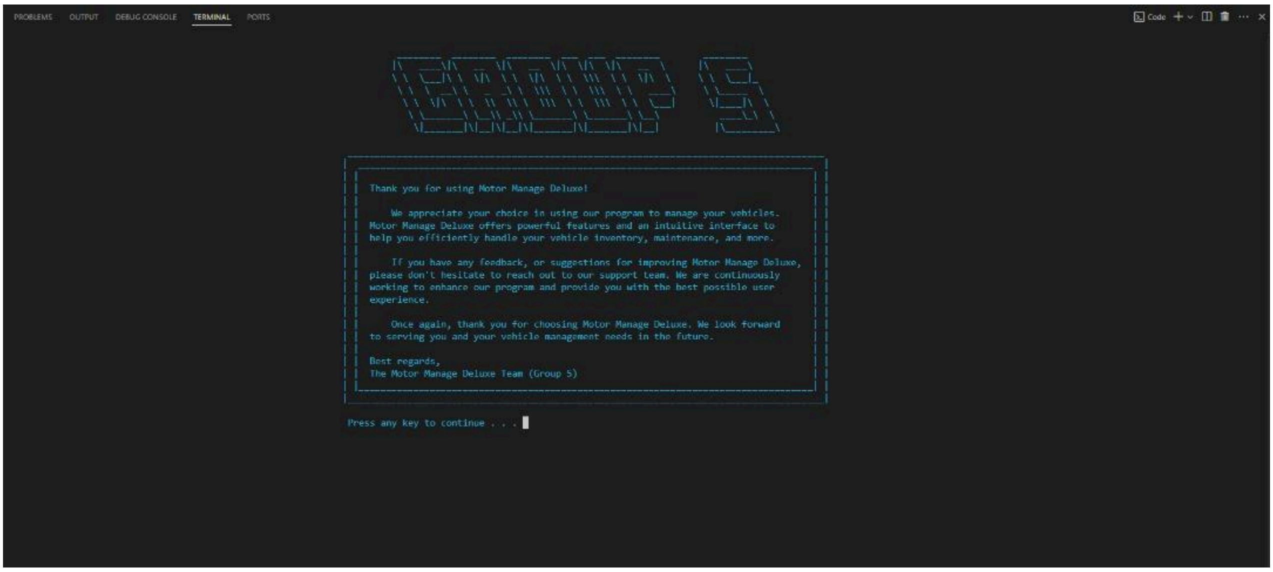
```
========================= Delete Vehicle =========================
Enter vehicle index to delete (1-0): 
```



```
==================== Search Vehicles By Filter ====================
Enter vehicle type to filter (A for Air, L for Land, S for Sea): 
```



```
==================== Search Vehicles By Filter ====================
Enter vehicle type to filter (A for Air, L for Land, S for Sea): 
```



```
Thank you for using Motor Manage Deluxe!

    We appreciate your choice in using our program to manage your vehicles.
Motor Manage Deluxe offers powerful features and an intuitive interface to
help you efficiently handle your vehicle inventory, maintenance, and more.

    If you have any feedback, or suggestions for improving Motor Manage Deluxe,
please don't hesitate to reach out to our support team. We are continuously
working to enhance our program and provide you with the best possible user
experience.

    Once again, thank you for choosing Motor Manage Deluxe. We look forward
to serving you and your vehicle management needs in the future.

Best regards,
The Motor Manage Deluxe Team (Group 5)

Press any key to continue . . . 
```

3.0 Object Oriented Programming Concept

First, we need to understand that object-oriented programming (OOP) is a programming paradigm that organizes data and behavior into objects, which are instances of classes. It promotes modularity, reusability, and encapsulation. In this program, the following OOP concepts are used:

1. Classes: Classes are the building blocks of object-oriented programming. They define the blueprint for creating objects with similar properties and behaviors. In the given program, several classes are defined: Vehicle, AirVehicle, LandVehicle, and SeaVehicle. Each class represents a specific type of vehicle.

2. Inheritance: Inheritance is a mechanism that allows a class to inherit properties and behaviors from another class. It promotes code reuse and hierarchical relationships. In the program, the classes AirVehicle, LandVehicle, and SeaVehicle inherit from the base class Vehicle. This means that the derived classes inherit the attributes and methods defined in the Vehicle class, such as name, year, display(), and the overloaded insertion operator <<.

3. Encapsulation: Encapsulation is the process of bundling data and methods together within a class, hiding internal implementation details and providing public interfaces for interacting with the object. In the program, the member variables of the classes (name, year, engineType, etc.) are declared as protected, which means they can be accessed by the derived classes. The member functions (display(), getName(), etc.) provide the public interface for accessing and manipulating the object's data.

4. Polymorphism: Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables the use of a single interface to represent multiple related classes. In the program, the display() function is declared as virtual in the base class Vehicle and overridden in the derived classes (AirVehicle, LandVehicle, and SeaVehicle). This allows calling the display() function on a base class pointer to invoke the appropriate version of the function based on the actual object type.

5. Operator Overloading: Operator overloading enables defining the behavior of operators (+, -, <<, etc.) for user-defined types. In the program, the insertion operator
   << is overloaded for the classes Vehicle, AirVehicle, LandVehicle, and SeaVehicle. This allows printing the object's information using the cout stream and simplifies the code for displaying the objects.

3.1 Peeling Back the Layers

By utilizing these OOP concepts, the program provides a modular and extensible structure for managing different types of vehicles. It allows adding, editing, and deleting vehicles, as well as displaying their information. The inheritance hierarchy ensures code reuse and promotes a clear and organized design. The encapsulation of data and methods within classes ensures data integrity and provides a clean interface for interacting with the objects. Overall, the program demonstrates the effective use of object-oriented programming concepts to model real-world entities and their relationships.