**Automobile Maintenance Library System**

# 1. Project Overview

This project simulates a basic automobile maintenance library system using Object-Oriented Programming in C++. It allows users to register multiple types of vehicles (electric, gas, diesel), manage their maintenance tasks, and interact with a menu-driven interface for task operations like adding/removing vehicles, assigning maintenance tasks, and viewing task lists.

# 2. System Features

• Register and remove vehicles of different fuel types  
• Assign maintenance tasks applicable to specific vehicle types  
• Remove maintenance tasks from a selected vehicle  
• List all registered vehicles  
• List all maintenance tasks for a selected vehicle  
• Prevent assignment of incompatible tasks to vehicle types  
• Menu-based interaction for user convenience

# 3. Technical Specifications

## 3.1 Automobile Class Hierarchy

- Automobile (Abstract Base Class)  
 - Pure virtual methods:  
 - getType()  
 - getMake()  
 - getModel()  
 - getYear()  
 - getOdometer()  
- Derived Classes:  
 - ElectricCar  
 - GasCar  
 - DieselCar

## 3.2 MaintenanceTask Class

Encapsulates a maintenance task:  
- Private Members:  
 - std::string taskName  
 - std::string description  
 - std::vector<std::string> applicableTypes  
- Public Methods:  
 - MaintenanceTask(std::string, std::string, std::vector<std::string>)  
 - std::string getTaskName() const  
 - std::string getDescription() const  
 - const std::vector<std::string>& getApplicableTypes() const

## 3.3 MaintenanceLibrary Class

Manages vehicles and their associated tasks:  
- Private Members:  
 - std::vector<std::shared\_ptr<Automobile>> vehicles  
 - std::unordered\_map<std::shared\_ptr<Automobile>, std::vector<MaintenanceTask>> maintenanceRecords  
- Public Methods:  
 - void addVehicle(std::shared\_ptr<Automobile> vehicle)  
 - void removeVehicle(std::shared\_ptr<Automobile> vehicle)  
 - void listVehicles() const  
 - void addMaintenanceTask(std::shared\_ptr<Automobile>, const MaintenanceTask&)  
 - void removeMaintenanceTask(std::shared\_ptr<Automobile>, const std::string&)  
 - void listMaintenanceTasks(std::shared\_ptr<Automobile>) const

# 4. User Guide

1. Add Vehicle: Specify type, make, model, year, and odometer reading.  
2. Remove Vehicle: Provide make and model.  
3. List Vehicles: View all registered vehicles.  
4. Add Maintenance Task: Provide task name and description, specify applicable types, and assign task to a vehicle.  
5. Remove Maintenance Task: By name from a specific vehicle.  
6. List Maintenance Tasks: View all tasks assigned to a vehicle.

# 5. Build and Test Instructions

Requirements:  
- C++ compiler with C++17 support  
- CMake (for cross-platform builds)

Steps:  
1. Clone the repository.  
2. Run 'cmake .' to configure the build.  
3. Run 'make' to build the executable.  
4. Run './AutomobileMaintenance' to start the system.

# 6. Sample Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case Name | Description | Input | Expected Output |
| testAddVehicle | Add new vehicles | Type = electric, make = Tesla | Vehicle added successfully |
| testInvalidVehicle | Handle invalid vehicle type | Type = hybrid | Invalid vehicle type |
| testAddTask | Add compatible maintenance task | Type = gas, task = oil change | Task added successfully |
| testIncompatibleTask | Task not valid for vehicle type | Type = electric, task for gas only | Task not applicable for electric |
| testListTasks | View maintenance tasks | Existing tasks present | Displays all tasks |
| testRemoveTask | Remove a specific task by name | Task name = battery check | Task removed from vehicle |
| testRemoveVehicle | Remove a registered vehicle | Make = Tesla, Model = Model3 | Vehicle and related tasks removed |

# 7. Notes

• Task compatibility is determined by matching the vehicle type with task’s applicable types.  
• Tasks are uniquely identified by their name within the scope of a single vehicle.  
• The application is designed using smart pointers to manage memory safely and efficiently.  
• Attempts to add duplicate or incompatible tasks are gracefully handled with user messages.