**Car Functionality System**

**Team Members:**

Tejashri Adsure

Harish Belande

Tejas Kamble

**1. Overview**

The **Car Functionality System** simulates the behaviour of a real car, it has features to:

* Accelerate and brake the car
* Shift gears between various states
* Track speed, gear, and location
* Set and retrieve destinations
* Calculate travel distance and estimated time to the destination

This system provides a simple interface to simulate and monitor a car’s behaviour and journey, including checking arrival and time-to-destination.

**2. System Architecture**

**Classes and Components:**

**Car Class:**

This is the core class and responsible for cars operation, including:

* **Attributes:**
  + gear: The current gear of the car (using the GearType enum).
  + speed: The current speed of the car (in km/h).
  + currentLocation: The current geographical location of the car (using the Location struct).
  + destination: The target destination of the car.
  + destinationSet: A flag to indicate if the destination has been set.
* **Methods:**
  + accelerate(double acceleration): Increases the car’s speed.
  + brake(double brakeForce): Decreases the car’s speed.
  + shiftGear(GearType newGear): Changes the car’s gear.
  + getGearState(): Retrieves the current gear state.
  + getSpeed(): Retrieves the car’s current speed.
  + getLocation(): Retrieves the car’s current location.
  + getDestination(): Retrieves the car’s destination.
  + setDestination(double lat, double lon): Sets the car’s destination.
  + calculateDistance(): Calculates the distance between the car's current location and destination.
  + drive(double timeInHours): Simulates driving the car for a specified time.
  + timeToDestination(): Calculates the estimated time to reach the destination.
  + hasArrived(): Checks if the car has reached the destination.
  + printStatus(): Prints the car's status, including gear, speed, and location.

**GearType Enum**

Represents the possible gear states of the car:

* PARK: The car is in park.
* REVERSE: The car is in reverse gear.
* NEUTRAL: The car is in neutral gear.
* DRIVE: The car is in drive gear.

**Location Struct**

Represents the geographical location of the car:

* latitude: Latitude of the car's location.
* longitude: Longitude of the car's location.

**3. Technical Specifications**

**Data Structures**

1. **GearType Enum**: Used to represent the gear states.

enum class GearType {

PARK, REVERSE, NEUTRAL, DRIVE

};

1. **Location Struct**: Used to store the geographical coordinates of the car.

struct Location {

double latitude;

double longitude;

};

1. **Car Class**: Handles the functionality of the car.

class Car {

private:

GearType gear;

double speed;

Location currentLocation;

Location destination;

bool destinationSet;

public:

Car();

void accelerate(double acceleration);

void brake(double brakeForce);

void shiftGear(GearType newGear);

GearType getGearState() const;

double getSpeed() const;

Location getLocation() const;

Location getDestination() const;

void setDestination(double lat, double lon);

double calculateDistance() const;

void drive(double timeInHours);

double timeToDestination() const;

bool hasArrived() const;

void printStatus() const;

};

**Distance Calculation**

The distance between the current location and destination is calculated using the **Haversine formula**:

* The formula uses the Earth's radius (R = 6371 km) and the latitudes and longitudes of the two locations.
* The resulting distance is in kilometers.

**4. User Guide**

**Getting Started**

1. Clone or download the project.
2. Build the project using CMake.
   * Ensure that you have **MinGW** or another compatible C++ compiler installed.
   * Run cmake and mingw32-make to compile the code.

**Running the System**

1. After building, run the executable car\_sim.exe from the command line.
2. The system will simulate the car's operations based on the code in main.cpp.

**User Inputs**

* Users can interact with the system by calling methods such as:
  + shiftGear()
  + accelerate()
  + brake()
  + setDestination()
  + drive()
  + timeToDestination()

The car’s current state (speed, gear, location) can be printed using printStatus().

**Example Usage**

Car myCar;

myCar.setDestination(10.0, 10.0);

myCar.shiftGear(GearType::DRIVE);

myCar.accelerate(60); // 60 km/h

myCar.drive(1.0); // Drive for 1 hour

myCar.printStatus();

std::cout << "Time to destination: " << myCar.timeToDestination() << " hours\n";

**5. Testing**

The system includes a set of test cases in the test.cpp file that:

* Test basic operations like acceleration, braking, and gear shifting.
* Verify that the car can drive and calculate its distance to the destination.
* Ensure that the system can calculate the time to the destination and determine if the car has arrived.

**6. Limitations and Future Improvements**

**Limitations**

* The system relies on some simplified assumptions, such as:
  + Constant speed during the drive() method.
  + The car only changes location based on the current speed and time.

**Future Improvements**

* We can add fuel consumption tracking.
* We can implement more complex driving behaviour (e.g., speed changes, road conditions).
* Extend the simulation to allow for more cars or multi-car interactions.

**7. Conclusion**

This Car Functionality System is the building block in simulating how a car handles, from driving movements and mechanical gear to distance covered and time until destination. Super extensible for more advanced simulations in the future.