1. Problem Statement: Single Inheritance

Problem Title: Vehicle Efficiency Calculator

Description: Design a class hierarchy to manage vehicle efficiency. Your task is to create a base class `Vehicle` and a derived class `Car`, demonstrating single inheritance. The system should be able to calculate and report fuel efficiency, but without using any virtual functions.

- **Base Class:** `Vehicle`
  - **Protected Members:**
    - `string make`: The vehicle's manufacturer.
    - `string model`: The vehicle's model name.
    - `int year`: The manufacturing year.
    - `double baseEfficiency`: A default efficiency value (e.g., in miles per gallon).
  - **Public Members:**
    - A constructor to initialize the members.
    - A `double calculateEfficiency() const;` method that simply returns the `baseEfficiency`.
    - A `displayInfo()` method that prints the vehicle's make, model, and year.
- **Derived Class:** `Car`
  - **Public Inheritance:** `class Car : public Vehicle`
  - **Private Members:**
    - `bool isElectric`: True if the car is electric.
    - `bool hasTurbo`: True if the car has a turbocharger.
  - **Public Members:**
    - A constructor to initialize all members, including those from the base class.
    - An overridden `calculateEfficiency()` method. This method should implement the following logic:
      - If `isElectric` is `true`, return a constant `baseEfficiency` value that represents the efficiency in a different unit (e.g., kWh/100km).
      - If `isElectric` is `false` and `hasTurbo` is `true`, calculate the efficiency as `baseEfficiency - (baseEfficiency * 0.25)`.
      - If `isElectric` is `false` and `hasTurbo` is `false`, return the `baseEfficiency` value directly.

Challenge: Write a `main` function that creates a `Car` object and calls its `calculateEfficiency()` method directly. Create a second `Car` object with different parameters and show how the `calculateEfficiency()` method produces a different result for each.

---

2. Problem Statement: Multilevel Inheritance

Problem Title: Smart Home Appliance Management System

Description: Develop a three-level class hierarchy to simulate a basic smart home appliance system. This problem will challenge you to manage data and functionality across multiple levels of inheritance, with each

level building upon the previous one.

- **Base Class:** `Appliance`
  - **Protected Members:**
    - `string name`: The name of the appliance (e.g., "Television").
    - `bool isOn`: The power status of the appliance.
    - `double powerConsumption_W`: The wattage of the appliance.
  - **Public Members:**
    - A constructor to initialize `name` and `powerConsumption_W`.
    - A `turnOn()` method that sets `isOn` to `true` and prints a confirmation.
    - A `turnOff()` method that sets `isOn` to `false` and prints a confirmation.
- **Intermediate Class:** `SmartAppliance`
  - **Public Inheritance:** `class SmartAppliance : public Appliance`
  - **Protected Members:**
    - `string ipAddress`: The IP address of the appliance.
    - `bool isConnected`: The Wi-Fi connection status.
  - **Public Members:**
    - A constructor that calls the base class constructor and initializes its own members.
    - A `connectToWiFi(string ip)` method that sets `ipAddress` and `isConnected` to `true`. This method should only be callable if the appliance is currently `isOn`.
- **Derived Class:** `SmartTV`
  - **Public Inheritance:** `class SmartTV : public SmartAppliance`
  - **Private Members:**
    - `int currentChannel`: The currently displayed channel number.
  - **Public Members:**
    - A constructor that initializes all members from its parent and grandparent classes.
    - A `displayChannel(int channel)` method. This method should only function if the appliance is both `isOn` and `isConnected`. If these conditions are met, it should set `currentChannel` and print a message. Otherwise, it should print an error message.

Challenge: In your `main` function, create a `SmartTV` object. Write a sequence of method calls to first turn it on, then attempt to display a channel (which should fail), then connect it to Wi-Fi, and finally successfully display a channel. This sequence will demonstrate the cascading dependencies and state management across the inheritance levels.