Problem Statement: Vehicle Simulation System with Single Inheritance

Objective:

Design and implement a simulation system for different types of vehicles using single inheritance in C++. The system should allow creation of various vehicle types like cars and motorcycles, with the ability to handle shared behaviors and specialized actions for each vehicle type.

Problem Requirements:

1. **Base Class - Vehicle**:
    - The `Vehicle` class should serve as the parent class for all vehicle types and should contain the following properties:
        - `brand` (type: `string`) - the vehicle's brand.
        - `year` (type: `int`) - the manufacturing year of the vehicle.
    - The class should include:
        - A constructor to initialize the `brand` and `year` of the vehicle.
        - A `virtual` method called `startEngine()` that prints a generic message for starting a vehicle's engine (this method should be overridden in derived classes).
        - A `virtual` destructor that cleans up the base class resources.
2. **Derived Class - Car**:
    - The `Car` class should inherit from `Vehicle` and represent a specific type of vehicle. This class should include:
        - An additional property: `doors` (type: `int`) that represents the number of doors on the car.
        - A constructor to initialize `brand`, `year`, and `doors`.
        - An overridden `startEngine()` method that prints a car-specific message for starting its engine.
        - A destructor that properly cleans up the `Car` object.
3. **Derived Class - Motorcycle**:
    - The `Motorcycle` class should also inherit from `Vehicle` and represent another specific type of vehicle. This class should include:
        - An additional property: `hasSideCar` (type: `bool`) that indicates whether the motorcycle has a sidecar.
        - A constructor to initialize `brand`, `year`, and `hasSideCar`.
        - An overridden `startEngine()` method that prints a motorcycle-specific message for starting its engine.
        - A destructor that properly cleans up the `Motorcycle` object.

4. **Simulation**:
   - In the `main()` function, create instances of `Car` and `Motorcycle` using pointers to the base class `Vehicle`. This demonstrates **polymorphism**, where the correct `startEngine()` method is called based on the object type.
   - After using the objects, ensure proper cleanup of resources by deleting the objects created with `new` to avoid memory leaks.
5. **Expected Output**:
   - The program should print messages when objects are created and destroyed, and the correct `startEngine()` method should be called based on the type of vehicle (Car or Motorcycle).

Example Output:

```
Vehicle constructed: Toyota (2021)
Car constructed with 4 doors.
Starting the car's engine with a roar!
Vehicle destructed: Toyota
Car destructed: Toyota
Vehicle constructed: Harley (2020)
Motorcycle constructed with sidecar: Yes
Starting the motorcycle's engine with a rev!
Vehicle destructed: Harley
Motorcycle destructed: Harley
```

Additional Considerations:

- Ensure that destructors are virtual to prevent resource leakage when objects are deleted through base class pointers.
- Properly manage memory allocation and deallocation to avoid memory leaks.
- Consider the use of `new` and `delete` operators for dynamic object creation.

Challenges:

- Managing complex initialization and inheritance relationships while keeping track of constructor chaining.
- Correctly overriding virtual methods to ensure polymorphic behavior.
- Ensuring proper destruction of derived class objects when deleted through base class pointers.