C++ Basics Practice Questions
September 18, 2025

General Guidelines:

1. Read the question thoroughly. Understand the scenario, requirements, and expected functionality before you begin coding.
2. Use the exact variable names, function names, and class names provided in the question.
   This ensures consistency and helps in automated or manual evaluation.
3. Follow all technical requirements as specified.
4. Follow MISRA C++ naming conventions.
   - Use descriptive and consistent variable names (e.g., componentIdentifier, not id)
   - Avoid abbreviations and reserved keywords
   - Maintain proper formatting and indentation
5. Ensure your code is modular, readable, and well-commented.
   - Break down logic into functions
   - Add comments to explain key parts of your code
6. Validate all user inputs.
   - Check for valid ranges, non-empty strings, and correct enum selections
7. Test your program with the sample data provided.
   - Ensure all functionalities work as expected
8. Submit only the required .cpp and .h files.
   - Make sure it compiles and runs without errors
9. Do not modify the structure of the question.
   - Stick to the design and flow as described
10. Assume suitable data whenever required.

Question 1: ADAS Dynamic Telemetry

In an Advanced Driver Assistance System (ADAS), vehicles continuously monitor and report telemetry data such as wheel speeds and engine temperatures. These values vary depending on the number of sensors and vehicle configuration, making dynamic memory management essential.

To model this, we define a class VehicleStatus that encapsulates:

- The number of wheels and their respective speeds.
- The number of engine temperature sensors and their readings.

This class uses dynamic arrays to store the data, allowing flexibility for different vehicle setups. The implementation includes constructors, destructors, statistical analysis functions, and comparison logic to evaluate performance between vehicles.

Requirements

Define class VehicleStatus with:

- int num_wheels
- float* wheel_speed (dynamic array)
- int num_temps
- float* engine_temp (dynamic array)

Implement:

- Constructor: Allocates and initializes all arrays with given data.
- Destructor: Frees dynamic memory.
- Const function averageWheelSpeed() const: Returns average of wheel speeds.
- Const function maxEngineTemp() const: Returns maximum engine temperature.

- Function isWheelSpeedHigher(const VehicleStatus& other) const: Uses this pointer and returns true if max wheel speed of calling object is higher than other.

Global Functions:

- void printVehicleStatus(const VehicleStatus& vs) : Prints wheel speeds and engine temperatures.
- bool compareWheelSpeedGlobal(const VehicleStatus& vs1, const VehicleStatus& vs2): Calls member comparison function.

Sample Data

Vehicle 1:

num_wheels = 4

wheel_speed = [55.5f, 56.6f, 57.2f, 55.9f]

num_temps = 2

engine_temp = [90.5f, 88.9f]

Vehicle 2:

num_wheels = 4

wheel_speed = [50.0f, 51.2f, 49.8f, 50.4f]

num_temps = 2

engine_temp = [92.0f, 89.5f]

Main Task

- Create two VehicleStatus objects with sample data.
- Print their statuses using printVehicleStatus.
- Compare max wheel speeds using:
- Member function isWheelSpeedHigher()
- Global function compareWheelSpeedGlobal()
- Free dynamic memory properly using destructors.

Question 2: Lane Tracking and Sensor Confidence

In an Advanced Driver Assistance System (ADAS), the ego vehicle — the vehicle equipped with the system — constantly monitors its position within lanes and evaluates the reliability of its onboard sensors. These metrics are crucial for safe lane keeping, adaptive cruise control, and autonomous navigation.
To model this telemetry, the EgoVehicleData class is introduced. It uses dynamic arrays to store:

- **Lane positions**: Floating-point values representing the vehicle's lateral position in each detected lane.
- **Sensor confidence scores**: Floating-point values indicating the reliability of each sensor's data.

This setup allows flexible handling of vehicles with varying numbers of lanes and sensors, and supports runtime analysis and comparison.

Requirements

Define class EgoVehicleData with:

- int num_lanes
- float* lane_positions
- float* sensor_confidence

Implement:

- **Constructor**: Allocates and initializes arrays with given data.
- **Destructor**: Frees dynamic memory.
- **Const function** getAverageLanePosition() const: Computes average lane position.
- **Function** updateSensorConfidence(float factor): Multiplies all confidence scores by a given factor.

Additional Tasks:

- Allocate a **heap array** of EgoVehicleData objects.
- Implement void findHighestConfidenceVehicle(EgoVehicleData* array, int size, const EgoVehicleData*& highest):

    - Assigns pointer to the vehicle with the highest **sum of sensor confidence scores**.

Global Functions:

1. void printEgoVehicleData(const EgoVehicleData& data)
    - Prints lane positions and confidence scores.
2. void printHighestConfidenceVehicle(const EgoVehicleData* vehicle)
    - Prints data or warns if nullptr.

Sample Data

Object 1:

- num_lanes = 3
- lane_positions = [3.2f, 3.0f, 3.4f]
- sensor_confidence = [0.95f, 0.97f, 0.93f]

Object 2:

- num_lanes = 3
- lane_positions = [2.9f, 2.8f, 3.1f]
- sensor_confidence = [0.92f, 0.90f, 0.88f]

Object 3:

- num_lanes = 3
- lane_positions = [3.4f, 3.5f, 3.6f]
- sensor_confidence = [0.99f, 0.98f, 0.97f]

Main Task

- Dynamically create **three** EgoVehicleData **objects** using the sample data.
- Adjust each object's sensor confidence scores by multiplying with **varying factors**.
- Use findHighestConfidenceVehicle() to identify the vehicle with the highest total confidence.
- Print the result using printHighestConfidenceVehicle().
- Ensure all dynamically allocated memory is properly freed.

Question 3: Satellite Telemetry System

In a space monitoring system, each satellite transmits telemetry data including its orbital altitude and signal strength from multiple antennas. The system must manage this data efficiently, compare satellites, and track how many satellites are currently active.

To enhance realism, each satellite also has a **status** represented by an enum, indicating whether it is operational, in maintenance, or decommissioned.

Requirements

Define enum SatelliteStatus: OPERATIONAL, MAINTENANCE, DECOMMISSIONED

Define class SatelliteData with:

- intsatellite_id
- intnum_antennas
- float* signal_strength (dynamic array)
- float orbital_altitude
- SatelliteStatus status
- static intactive_satellites (tracks number of active satellites)

Implement

Member Functions:

- **Constructor**: Allocates and initializes signal strength array, sets altitude and status.
- **Copy Constructor**: Performs deep copy of signal strength array and status.
- **Destructor**: Frees dynamic memory and decrements active_satellites.
- **Const function** getAverageSignalStrength() const: Returns average signal strength.
- **Function** boostSignal(float factor): Multiplies all signal strengths by a factor.
- **Function** boostSignal(float factor, float threshold) *(Overloaded)*: Boosts only signals below the threshold.
- **Static function** getActiveSatelliteCount(): Returns current count of active satellites.

- **Function** setStatus(SatelliteStatus new_status): Updates satellite status.
- **Const function** getStatusString() const: Returns string representation of current status.

Global Functions

1. void printSatelliteData(const SatelliteData& sd)
    - Prints satellite ID, altitude, signal strengths, and status.
2. bool compareAltitude(const SatelliteData& s1, const SatelliteData& s2)
    - Returns true if s1 has higher altitude than s2.
3. void cloneSatellite(const SatelliteData& source, SatelliteData*& target)
    - Uses copy constructor to clone source into target.
4. void printActiveSatelliteCount()
    - Prints current count of active satellites using static function.
5. void updateStatusIfWeak(SatelliteData& sd, float confidence_threshold)
    - If average signal strength is below threshold, set status to MAINTENANCE.

Sample Data

Create two satellites with the following data:

Satellite 1:

- satellite_id = 101
- num_antennas = 3
- signal_strength = [78.5f, 80.2f, 79.0f]
- orbital_altitude = 550.0f
- status = OPERATIONAL

Satellite 2:

- satellite_id = 102
- num_antennas = 2
- signal_strength = [75.0f, 76.5f]
- orbital_altitude = 600.0f
- status = MAINTENANCE

Main Task

- Create two SatelliteData objects with sample data.
- Boost signal strength of one satellite using both overloaded versions.
- Compare their altitudes.
- Clone one satellite using copy constructor.
- Print all satellite data including the cloned one.
- Update status based on signal strength.
- Print active satellite count.
- Free all dynamically allocated memory.