C++ Basics Practice Questions

September 16, 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: LaneBoundary Detection in ADAS

Scenario:

In modern Advanced Driver Assistance Systems (ADAS), lane detection is a critical feature that helps vehicles maintain safe positioning on the road. The system uses camera and sensor data to identify lane markings and estimate their curvature, which indicates how sharply the lane is turning. Each detected lane is assigned a unique identifier for tracking and decision-making.

Requirements:

- Define class LaneBoundary with members:
    - float curvature
    - int lane_id
- Implement:
    - Constructor to initialize members
    - Const getter functions for members
    - Function compareCurvature(const LaneBoundary& other) const that returns true if calling object's curvature is strictly greater than other using this pointer explicitly.
    - Const function display() const to print the lane details.

Global Functions:

1. bool isCurvatureGreater(const LaneBoundary& lane1, const LaneBoundary& lane2)
   returns true if lane1 curvature > lane2 curvature.
2. void printLaneComparison(const LaneBoundary& lane1, const LaneBoundary& lane2)
   prints which lane has greater curvature or if equal.

Main Task:

- Create two LaneBoundary objects (lane1 and lane2) with sample values.
- Use both member and global functions to compare curvatures and print results.

Sample Data:

- Lane1: curvature = 0.015, lane_id = 1
- Lane2: curvature = 0.023, lane_id = 2

Question 2: Sensor Fusion for Object Distance Estimation in ADAS

Scenario:

In Advanced Driver Assistance Systems (ADAS), **sensor fusion** is a technique used to combine data from multiple sensors to improve the accuracy and reliability of object detection. Two primary sensors used for estimating the distance to objects ahead are:

- Radar: Provides robust distance measurements, especially in poor visibility conditions.
- Camera: Offers visual context and can estimate distance using image processing techniques.

Each sensor may report slightly different values due to environmental factors, sensor limitations, or object characteristics. To reconcile these differences, the system may apply calibration or adjustment logic.

The ObjectDistance class models the fused distance data from radar and camera for a detected object. It supports operations to adjust the distances (e.g., simulating calibration or sensor drift compensation) and provides utilities for dynamic memory management and data inspection.

Requirements:

- Define class ObjectDistance with members:
    - float radar_distance
    - float camera_distance
- Implement:
    - Constructor
    - Functions adjustDistancesByValue(ObjectDistance obj) and adjustDistancesByReference(ObjectDistance& obj) adding +5.0 to distances.
    - Dynamically allocate ObjectDistance object on heap.

- Global Functions:

  1. void printObjectDistance(const ObjectDistance& obj) prints radar and camera distances.
  2. ObjectDistance* createObjectDistanceOnHeap(float radar, float camera) creates and returns pointer to dynamically allocated object.

Main Task:

- Create an object on heap with radar=35.5, camera=34.8.
- Call adjust functions and print before/after using global print function.
- Delete heap object properly.

Question 3: Camera Configuration Management in ADAS

Scenario:

In Advanced Driver Assistance Systems (ADAS), multiple camera modes are used to support various driving features such as lane detection, object recognition, parking assistance, and surround view. Each mode operates at a specific resolution optimized for its task. For example:

- Mode 0: Standard forward-facing camera for lane detection.
- Mode 1: High-resolution rear camera for parking assistance.
- Mode 2: Ultra-wide camera for surround view.

The CameraConfig class is designed to manage these resolution settings dynamically. It stores the width and height of each mode in separate arrays, allowing flexible configuration and comparison of resolution quality.

Requirements:

- Define class CameraConfig with:
    - int num_modes
    - int* resolution_width (dynamic array)
    - int* resolution_height (dynamic array)
- Implement:
    - Constructor to allocate arrays dynamically for num_modes and initialize with given data.
    - Copy constructor and destructor to manage memory safely.
    - Const function printConfig() const to display mode resolutions.
    - Member function isHigherResolution(int mode1, int mode2) const using this pointer: compares (width × height) for two modes, returns true if mode1 has higher resolution.
- Global Functions:
    1. void printCameraConfig(const CameraConfig& config) prints all mode resolutions.
    2. bool globalCompareResolution(const CameraConfig& config, int m1, int m2) calls member comparison.

Sample Data:

- num_modes = 3
- resolution_width = [1920][1280][3840]
- resolution_height = [1080][2160]

Main Task:

- Create a CameraConfig object using the sample data.
- Compare resolutions of modes 0 and 2 using both member and global functions, print results.

Question 4: Radar Signal Strength Management in ADAS

Scenario

In Advanced Driver Assistance Systems (ADAS), radar sensors play a vital role in detecting objects, estimating distances, and assessing relative speeds. These sensors operate across multiple channels, each tuned to different frequencies or directions to provide comprehensive coverage around the vehicle.

Each radar channel reports a signal strength, which reflects the quality and reliability of the detection. Signal strength can vary due to environmental conditions, object reflectivity, and sensor calibration. To simulate signal enhancement or calibration, the system may apply a boost to these values.

The RadarSignal class models this multi-channel signal data and supports operations to analyze and modify signal strengths dynamically.

Requirements:

- Define class RadarSignal with:
    - int num_channels
    - float* signal_strength (dynamic array)
- Implement:
    - Constructor allocating and initializing signal strengths dynamically.
    - Destructor freeing dynamic memory.
    - Function averageSignal() const returns average signal strength.
    - Functions boostSignalByValue(RadarSignal obj) and boostSignalByReference(RadarSignal& obj) add +5.0 to each signal strength.
- Global Functions:
    1. void printRadarSignal(const RadarSignal& radar) prints all signal strengths.
    2. RadarSignal* createRadarSignalHeap(int num, float* signals) creates and returns pointer to dynamic instance.

Sample Data:

- num_channels = 4

- signal_strength = [55.5f, 48.2f, 60.1f, 52.6f]

Main Task:

- Create RadarSignal object on heap with sample data.
- Call boost functions by value and by reference, showing effects using global print function.
- Delete heap object afterwards.