C++ Basics Practice Questions
September 30, 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.
11. If you are not aware of few functionalities then those implementations can be ignored.
12. Provide Appropriate File Name:
    YourName_MonthDate_Task1_main.cpp
    YourName_MonthDate_Task1_className.cpp
    YourName_MonthDate_Task1_className.h

    Ex
    John_Oct1_Task2_main.cpp
    John_Oct1_Task2_myClass.cpp
    John_Oct1_Task2_myClass.h

Task 1: SmartFleet — Hybrid Vehicle Analytics System

You're building a system for a fleet management company that tracks hybrid vehicles. Each vehicle logs electric and gasoline usage, and the system must support advanced analytics and intuitive operations using operator overloading.

Class: HybridVehicle

Attributes:

- string modelName
- float electricKm
- float gasolineKm
- float gasolineUsed
- int trips[] — array of trip distances (max 10)

Operators to Overload

Operator	Type	Purpose
+	Binary arithmetic	Combine two vehicles' usage
==	Binary comparison	Compare efficiency
=	Assignment	Deep copy of trips and usage
++	Unary (prefix)	Increment trip count and add dummy trip
[]	Subscript	Access trip distance by index
()	Function call	Return total distance
<<	Stream insertion	Print vehicle details
float()	Type conversion	Convert to fuel efficiency (km/L)

Tasks in main()

1. Create two HybridVehicle objects with sample data.
2. Display both using <<.
3. Use + to combine them and display the result.
4. Use == to compare efficiency.
5. Use ++ to simulate a new trip.
6. Access a trip using [].
7. Use () to get total distance.
8. Use float() to get efficiency.
9. Use = to copy one vehicle to another and verify deep copy.

Sample Data

```
HybridVehicle v1("Prius", 120.5, 80.0, 4.5);
HybridVehicle v2("Volt", 100.0, 90.0, 5.0);
v1[0] = 50; v1[1] = 60;
v2[0] = 70; v2[1] = 40;
```

Task 2: Battery Health Monitor with Dynamic Trip Logging

Scenario: Battery Health Monitor with Dynamic Trip Logging

A startup is building a system to monitor the battery health of electric scooters. Each scooter logs battery usage, charging cycles, and degradation metrics. The system must support intuitive comparisons and updates using operator overloading, and must dynamically manage trip data.

Class: BatteryMonitor

Attributes:

- std::string scooterID
- int chargeCycles
- float batteryCapacity — current capacity in kWh
- float originalCapacity — original capacity in kWh
- bool isActive
- int* tripDistances — dynamically allocated array of trip distances
- int tripCount — number of trips

Memory Management Requirements:

- Use dynamic memory (new and delete) to manage tripDistances
- Implement:
    - **Constructor**: allocates memory for trips
    - **Copy constructor**: deep copy
    - **Assignment operator (=)**: deep copy with self-assignment check
    - **Destructor**: deallocates memory

Operators to Overload

Operator	Type	Purpose
> / <	Relational	Compare battery health (capacity ratio)
+=	Compound assignment	Add charge cycles and simulate degradation
[]	Subscript	Access trip distance by index
!	Unary logical	Check if scooter is inactive
<<	Stream insertion	Print scooter status and health %

Tasks in main()

1. Create two BatteryMonitor objects with dynamic trip arrays.
2. Populate trips using [].
3. Display both using <<.
4. Use > and < to compare battery health.
5. Use += to simulate 50 charge cycles and reduce capacity.
6. Use ! to check inactive status.
7. Use = to copy one scooter to another and verify deep copy.
8. Demonstrate memory cleanup via destructor.

Sample Data

```
BatteryMonitor s1("SCT101", 300, 4.0, 5.0, true, 3);  // 80% health
BatteryMonitor s2("SCT202", 150, 3.5, 5.0, true, 2);  // 70% health

s1[0] = 12; s1[1] = 18; s1[2] = 25;
s2[0] = 10; s2[1] = 15;
```

Task 3: Geometry Toolkit for Shape Classification

You're building a geometry toolkit for a CAD (Computer-Aided Design) application. The system must model various types of quadrilaterals and support classification, area computation, and hierarchy traversal. Your task is to design a deep inheritance hierarchy that reflects geometric relationships and constraints.

Class Hierarchy Design

Base Class: Quadrilateral

- Represents any four-sided polygon.
- Attributes:
  - float sides[4] — lengths of the four sides
  - float angles[4] — internal angles in degrees
- Methods:
  - virtual bool isValid() — checks if sum of angles is 360°
  - virtual float area() — returns 0 by default

Derived Classes (Deepest Possible Hierarchy)

```
Quadrilateral
├──── Trapezoid
│     └──── Parallelogram
│           └──── Rectangle
│                 └──── Square
```

- **Trapezoid**: At least one pair of parallel sides
- **Parallelogram**: Two pairs of parallel sides
- **Rectangle**: All angles are 90°
- **Square**: All sides equal and all angles are 90°

Each derived class should override isValid() and area() appropriately.

Operator Overloads

Operator	Purpose
==	Compare area of two shapes
<<	Print shape type and dimensions
()	Return perimeter
float()	Convert to area

Memory Management Requirement

- Use dynamic memory allocation for sides and angles arrays.
- Implement:
    - Constructor with new
    - Copy constructor
    - Assignment operator
    - Destructor with delete[]

Tasks in main()

1. Create one object of each class: Quadrilateral, Trapezoid, Parallelogram, Rectangle, Square.
2. Store them in a Quadrilateral* array and demonstrate polymorphism.
3. Call isValid() and area() for each.
4. Use << to print details.
5. Use == to compare area of Rectangle and Square.
6. Use float() to convert a shape to its area.
7. Use () to get perimeter.

Sample Data

Square sq({5,5,5,5}, {90,90,90,90});
Rectangle rect({6,4,6,4}, {90,90,90,90});
Parallelogram para({6,4,6,4}, {110,70,110,70});
Trapezoid trap({6,5,4,3}, {100,80,100,80});
Quadrilateral quad({3,4,5,6}, {90,90,90,90});

Task 4: University Student Information System

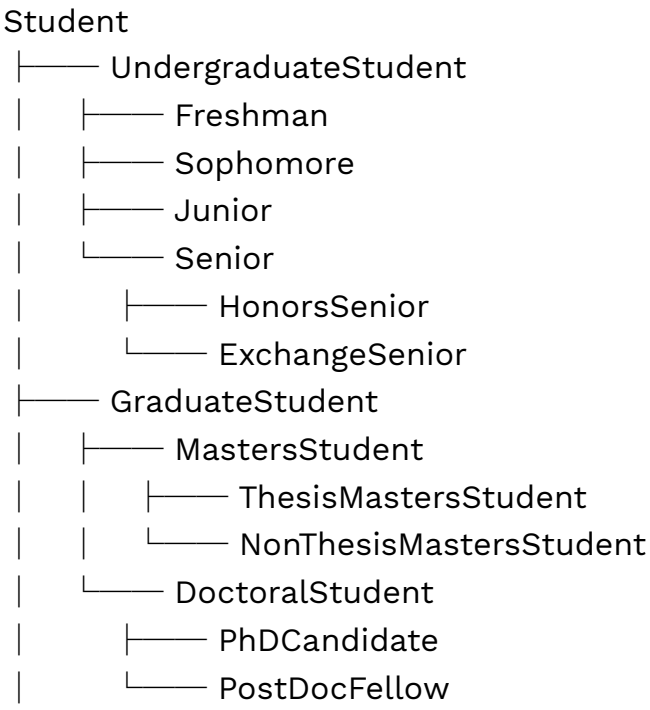
A university wants to build a student information system that models the academic progression of its students. The system must support detailed classification of students based on their level (undergraduate or graduate), year of study, and program type. The design must reflect real-world academic relationships using a deep inheritance hierarchy.

Requirements

Base Class: Student

- Attributes:
  - std::string name
  - std::string studentID
  - std::string department
- Methods:
  - virtual void displayInfo()
  - virtual std::string getLevel() — returns "Student" by default

Derived Classes



Class-Specific Attributes

- UndergraduateStudent: int year, int creditsEarned
- HonorsSenior: std::string thesisTitle
- ExchangeSenior: std::string homeUniversity
- GraduateStudent: std::string advisor
- ThesisMastersStudent: std::string thesisTitle
- NonThesisMastersStudent: std::string projectTitle
- DoctoralStudent: std::string dissertationTitle, int yearsInProgram
- PostDocFellow: std::string researchGrant

Virtual Method Overrides

Each derived class must override:

- displayInfo() to show class-specific data
- getLevel() to return appropriate academic level (e.g., "Freshman", "PhDCandidate")

Tasks to Perform in main()

1. Create one object of each class using sample data.
2. Store all objects in a Student* array or vector.
3. Use a loop to call displayInfo() and getLevel() polymorphically.
4. Demonstrate dynamic binding by printing student levels.
5. Optionally, use typeid or dynamic_cast to identify specific types.

Sample Data

Freshman f("Amit", "UG101", "Computer Science", 1, 15);

HonorsSenior hs("Neha", "UG401", "Physics", 4, 120, "Quantum Entanglement Thesis");

ExchangeSenior es("Luca", "UG402", "Mechanical", 4, 110, "Politecnico di Milano");

ThesisMastersStudent tm("Priya", "GR201", "Electrical", "Dr. Rao", "Smart Grid Optimization");

NonThesisMastersStudent ntm("Raj", "GR202", "Civil", "Dr. Mehta", "Bridge Load Simulation");

PhDCandidate phd("Sneha", "GR301", "Biotech", "Dr. Kapoor", "Gene Editing Ethics", 3);
PostDocFellow pdf("Dr. Tanmay", "GR401", "AI Research", "Dr. Bose", "DARPA Grant");