C++ Basics Practice Questions

September 15, 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: Fleet Vehicle Monitoring System

Scenario:

A logistics company wants to build a system to monitor its fleet of delivery vehicles. Each vehicle has various attributes and behaviors. You are required to design a C++ program that models this system using object-oriented principles.

Class: FleetVehicle

Private Data Members:

- int vehicleID
- float fuelLevel
- double distanceTravelled
- char status ('A' for Active, 'I' for Inactive)
- bool isAvailable
- std::string driverName

Constructors:

1. **Default Constructor** Initializes members with default values:
    - vehicleID = 0;
    - fuelLevel = 50.0;
    - distanceTravelled = 0.0;
    - status = 'A';
    - isAvailable = true;
    - driverName = "Unassigned";
2. **Parameterized Constructor** Accepts values for all members and initializes them.

Destructor:

Prints a message like: "Destructor called for vehicle ID: <vehicleID>;"

Member Functions:

- Getters and Setters for each data member.
- void updateStatus()\ Uses **if-else**:
- if (fuelLevel < 10.0 || !isAvailable)
- status = 'I';
- else
- status = 'A';
- void displayInfo()\ Prints all vehicle details.

Global Functions:

1. void assignDriver(FleetVehicle &vehicle, std::string name)
    - Uses setter to assign driver name.
    - Uses getter to confirm assignment.
2. void refuelVehicle(FleetVehicle &vehicle, float fuelAmount)
    - Uses setter to increase fuel level.
    - Validates fuel amount using **if-else**.
    - Throws an exception if fuelAmount <= 0.

Main Function Requirements:

- Create an array of 3 FleetVehicle objects using either static or dynamic allocation.
- Use a **switch-case** menu to:
    1. Add vehicle details using parameterized constructor
    2. Assign a driver
    3. Refuel a vehicle
    4. Update status
    5. Display vehicle info
    6. Exit
- The menu should run in a loop until the user selects the Exit option.
- Ensure all output is clearly labeled and formatted for readability.

Sample Data for Testing:

```
FleetVehicle v1(101, 45.5, 120.0, 'A', true, "Raj");
FleetVehicle v2(102, 8.0, 300.0, 'A', true, "Priya");
FleetVehicle v3; // Uses default constructor
```

Question 2: Library Book Management System

Scenario:

A public library wants to automate its book tracking system. Each book has various attributes and behaviors. You are required to design a C++ program that models this system using object-oriented principles and demonstrates good programming practices, including the use of the const keyword and enumerations.

Class: Book

Enum: Genre { FICTION, NONFICTION, SCIENCE, HISTORY, TECHNOLOGY}

Private Data Members:

- int bookID
- float rating
- double price
- char availabilityStatus ('A' for Available, 'B' for Borrowed)
- bool isReferenceOnly
- std::string title
- Genre genre

Constructors:

1. **Default Constructor** Initializes members with default values:
2. bookID = 0;
3. rating = 0.0;
4. price = 0.0;
5. availabilityStatus = 'A';
6. isReferenceOnly = false;
7. title = "Untitled";
8. genre = FICTION;
9. **Parameterized Constructor** Accepts values for all members and initializes them.

Destructor:

Prints a message like: "Book object with ID <bookID> destroyed."

Member Functions:

- Getters and Setters for each data member.
- void updateAvailability(bool isBorrowed) Uses **if-else**: availabilityStatus = isBorrowed ? 'B' : 'A';
- void displayDetails() const : Prints all book details including genre as a string. Uses const to ensure no modification of object state.

Global Functions:

1. void markAsReference(Book &book)
    - Uses setter to mark the book as reference-only.
    - Uses getter to confirm and print a message.
2. bool isAffordable(const Book &book, const double budget)
    - Uses getter to compare book price with budget.
    - Returns true if book price ≤ budget, else false.
3. void printFormattedBookList(const Book books[], const int size)
    - Uses **const** parameters.
    - Iterates through the array and prints each book in a formatted table:
      ID | Title | Price | Rating | Status | Reference |

Main Function Requirements:

- Create an array of 3 Book objects.
- Implement a **menu-driven loop** using while or do-while and **if-else** statements.
- Prompt the user to choose actions:
    - Add book details using parameterized constructor
    - Mark a book as reference-only

    - Check if a book is affordable
    - Update availability
    - Display book details
    - Display all books in formatted list
    - Exit
- Use **input validation** for:
    - Rating range (0.0–5.0)
    - Price must be positive
    - Non-empty title
    - Valid genre selection

Sample Data for Testing:

Book book1(101, 4.5, 299.99, 'A', false, "C++ Primer", TECHNOLOGY);

Book book2(102, 3.8, 150.00, 'B', true, "Design Patterns", SCIENCE);

Book book3; // Uses default constructor

Question 3: Aerospace Component Monitoring System

Scenario:

An aerospace company needs a system to monitor various components used in aircraft. Each component has specific attributes such as type, status, and performance metrics. You are required to design a C++ program that models this system using object-oriented principles and demonstrates good programming practices, including MISRA C++ naming conventions and the use of the const keyword.

Class: AerospaceComponent

Enum Types:

```
enum ComponentType
{
    COMPONENT_TYPE_ENGINE,
    COMPONENT_TYPE_AVIONICS,
    COMPONENT_TYPE_LANDING_GEAR,
    COMPONENT_TYPE_FUEL_SYSTEM
};

enum ComponentStatus
{
    COMPONENT_STATUS_OPERATIONAL,
    COMPONENT_STATUS_MAINTENANCE_REQUIRED,
    COMPONENT_STATUS_FAILED
};
```

Private Data Members:

- int componentIdentifier
- double componentEfficiency
- std::string componentManufacturer

- ComponentType componentType
- ComponentStatus componentStatus

Constructors:

1. **Default Constructor** Initializes members with default values:

   componentIdentifier = 0;

   componentEfficiency = 100.0;

   componentManufacturer = "Unknown";

   componentType = COMPONENT_TYPE_ENGINE;

   componentStatus = COMPONENT_STATUS_OPERATIONAL;

2. **Parameterized Constructor**

   Accepts values for all members and initializes them.

Destructor: "Component <componentIdentifier> destroyed."

Member Functions

1. **void UpdateStatus()**
   - Uses **if-else** logic:
     - If componentEfficiency < 50.0, set componentStatus = COMPONENT_STATUS_FAILED
     - If componentEfficiency < 80.0, set componentStatus = COMPONENT_STATUS_MAINTENANCE_REQUIRED
     - Else, set componentStatus = COMPONENT_STATUS_OPERATIONAL
2. **void SimulateUsage(int usageHours)**
   - Uses a **for loop**:
     - For each hour, reduce efficiency by 0.5%
     - Ensure efficiency does not drop below 0
3. **void PerformMaintenanceCheck()**
   - Uses a **switch-case** on componentStatus:

        - Print appropriate maintenance message
4. **void BoostEfficiency()**
    - Uses a **while loop**:
        - Increment efficiency by 1.0 until it reaches 100.0
5. **void DisplayDetails() const**
    - Uses const to ensure no modification
    - Prints all component details

Global Functions:

1. **void AssignManufacturer(AerospaceComponent &component, const std::string &manufacturerName)**
    - Uses setter to assign manufacturer name
2. **bool IsEfficient(const AerospaceComponent &component)**
    - Returns true if componentEfficiency > 85.0
3. **bool IsSameType(const AerospaceComponent &componentA, const AerospaceComponent &componentB)**
    - Compares componentType
4. **bool IsSameStatus(const AerospaceComponent &componentA, const AerospaceComponent &componentB)**
    - Compares componentStatus
5. **void PrintFormattedComponentList(const AerospaceComponent componentList[], const int listSize)**
    - Displays all components in a formatted table
6. **int SearchComponentByIdentifier(const AerospaceComponent componentList[], const int listSize, const int searchIdentifier)**
    - Searches for a component by componentIdentifier
    - Returns index if found, -1 otherwise

Main Function Requirements:

- Create an array of 3 AerospaceComponent objects
- Implement a **menu-driven loop** using while and if-else

- Prompt the user to choose actions:
    - Add component details
    - Assign manufacturer
    - Simulate usage
    - Boost efficiency
    - Check maintenance status
    - Compare components
    - Search by component ID
    - Display all components
    - Exit
- Use **input validation** for:
    - Efficiency range (0.0–100.0)
    - Non-empty manufacturer name
    - Valid enum selection

Sample Data for Testing:

- AerospaceComponent componentOne(101, 92.5, "GE Aviation", COMPONENT_TYPE_ENGINE, COMPONENT_STATUS_OPERATIONAL);
- AerospaceComponent componentTwo(102, 45.0, "Honeywell", COMPONENT_TYPE_AVIONICS, COMPONENT_STATUS_FAILED);
- AerospaceComponent componentThree; *// Uses default constructor*