CodInClub

Powered By BridgeLabz

C++ Reference Questions

1. Introduction to OOP Concepts

Theory Questions

Easy

1. What is Object-Oriented Programming (OOP)?
2. Name the four main features of OOP.
3. What is an object in C++?
4. Define class in simple terms.
5. What is the main difference between a class and an object?

Medium

1. Explain encapsulation with a real-life example.
2. How does inheritance promote code reusability?
3. What is polymorphism? Give an example.
4. Why is abstraction important in OOP?
5. Compare procedural programming and OOP in one sentence.

Hard

1. How does OOP help in managing large software projects?
2. Explain how data hiding enhances security in OOP.
3. Can we achieve OOP concepts in C? Justify your answer.
4. Discuss the role of abstraction and encapsulation in building secure software.
5. Why is OOP considered more maintainable than procedural programming?

Programming Questions

Easy

1. Write a simple class `Student` with two data members: `name` and `age`.
2. Create an object of the `Student` class and initialize it.
3. Add a member function to display the student's details.
4. Modify the class to make data members private and use public functions to access them.
5. Write a `main()` function to create and display a student.

Medium

1. Create a class `Rectangle` with length and breadth. Include a method to calculate area.
2. Make the data members private and use public getter/setter methods.
3. Add a constructor to initialize the dimensions.

4. Create two objects and compare their areas.
5. Demonstrate encapsulation by validating input in setter functions (e.g., no negative values).

Hard

1. Design a class `BankAccount` with balance, account number, and name. Include deposit, withdraw, and display functions.
2. Implement data hiding and input validation (e.g., no negative deposit).
3. Use a static member to count the number of accounts created.
4. Add a friend function to transfer money between two accounts.
5. Simulate 3 accounts, perform transactions, and show the final state.

---

2. C++ Programming Statements, Variables, Data Types, and Scopes

Theory Questions

Easy

1. List five basic data types in C++.
2. What is a variable?
3. What is the difference between `int` and `float`?
4. What is the scope of a variable?
5. What is a global variable?

Medium

1. Explain the difference between local and global scope.
2. What is the lifetime of a static variable?
3. What happens if you use a variable before declaring it?
4. What is the difference between `auto` and explicit type declaration?

Hard

1. Explain block scope with nested blocks.
2. What are the consequences of variable shadowing?
3. Why is it recommended to declare variables close to their usage?
4. How does `const` affect variable scope and lifetime?

Programming Questions

Easy

1. Declare variables of `int`, `float`, `double`, `char`, and `bool`.
2. Print the size of each data type using `sizeof()`.
3. Declare a global and a local variable with the same name and print both.
4. Use `auto` to declare a variable and let the compiler deduce its type.
5. Initialize a `const` variable and try to modify it (observe the error).

Medium

1. Write a program with two functions: one using a local static variable to count calls.
2. Demonstrate variable shadowing in nested blocks.
3. Declare variables inside `if` and `for` blocks and access them outside (observe scope error).
4. Use `const` with pointers: `const int*`, `int* const`, and `const int* const`.

Hard

1. Write a program where a global variable is hidden by a local one, and access both using scope resolution.
2. Simulate a counter using `static` variable in a function across recursive calls.
3. Use `constexpr` to define a compile-time constant and use it in array size.
4. Create a namespace and define variables with overlapping names; solve using scope.

---

3. Basic Operations – Arithmetic, Logical, Bitwise Operators

Theory Questions

Easy

1. What is the result of `5 / 2` in C++?
2. What does the `&&` operator do?
3. What is the difference between `&` and `&&`?
4. What is the output of `5 << 1`?
5. What does the `%` operator do?

Medium

1. Explain the difference between pre-increment and post-increment.

2. What is operator precedence? Give an example.
3. How does the ternary operator work?
4. What is the result of `~5`? Show the bit-level calculation.
5. Why are bitwise operators useful in embedded systems?

Hard

1. Explain how to swap two numbers using XOR without a temporary variable.
2. How can you check if a number is even using bitwise operators?
3. What is the difference between logical and bitwise OR in conditional statements?
4. How can you extract a specific bit from an integer?
5. Why does `!0` return `true` but `~0` returns `-1`?

Programming Questions

Easy

1. Write a program to add, subtract, multiply, and divide two numbers.
2. Check if a number is even or odd using `%`.
3. Use `&&` and `||` to check if a number is between 10 and 20.
4. Use `++` and `--` operators on a variable and print before and after.
5. Compute `a = 5 << 2` and print the result.

Medium

1. Write a program to toggle the 3rd bit of a number using XOR.
2. Check if the 5th bit is set using bitwise AND.
3. Swap two numbers using XOR.
4. Use the ternary operator to find the maximum of two numbers.
5. Evaluate an expression like `a = 5 + 3 * 2 > 10 ? 1 : 0` and explain.

Hard

1. Write a function to count the number of set bits in an integer.

2. Reverse the bits of a 32-bit integer.

3. Write a macro using bitwise operators to set, clear, and toggle a bit.

4. Implement a function to check if a number is a power of two using bitwise ops.

5. Use bitwise operators to multiply a number by 8 without `*`.

---

4. Control Flow – Condition and Loops

Theory Questions

Easy

1. What is the difference between `if` and `if-else`?
2. What is the purpose of the `break` statement?
3. What is the difference between `while` and `do-while`?
4. When is the `continue` statement used?
5. How many times does a `for` loop execute if the condition is false initially?

Medium

1. Explain the difference between entry-controlled and exit-controlled loops.
2. Can we have multiple `else if` blocks? Give syntax.
3. What happens if you forget `break` in a `switch` case?
4. How can you simulate a `for` loop using `while`?
5. When should you prefer `switch` over `if-else`?

Hard

1. Explain nested loops with an example of pattern printing.
2. How can infinite loops be useful? Give an example.
3. Discuss time complexity of nested loops with examples.
4. How can you optimize a loop that checks for prime numbers?

Programming Questions

Easy

1. Write a program to check if a number is positive, negative, or zero.
2. Print numbers from 1 to 10 using a `for` loop.
3. Print even numbers from 1 to 20 using `while`.
4. Use `switch` to print the day of the week based on number (1-7).
5. Print a triangle of stars with 5 rows using nested loops.

Medium

1. Check if a number is prime using a loop.
2. Print the Fibonacci series up to `n` terms.
3. Find the factorial of a number using a loop.
4. Print a multiplication table using nested loops.
5. Reverse a number using a `while` loop.

Hard

1. Print a pyramid pattern using nested loops (e.g., 1, 121, 12321).
2. Find all Armstrong numbers between 1 and 1000.
3. Simulate a menu-driven calculator using `switch` and loop.
4. Print all prime numbers between 1 and 100.
5. Implement a number guessing game with limited attempts.

---

5. Pointers

Theory Questions

Easy

1. What is a pointer?
2. How do you get the address of a variable?
3. What is dereferencing?
4. What is a null pointer?
5. What is the size of a pointer on a 64-bit system?

Medium

1. What are the drawbacks of raw pointers?
2. What is a dangling pointer?
3. Why should we initialize pointers?
4. What is pointer arithmetic?
5. How do pointers relate to arrays?

Hard

1. Explain the drawbacks of between raw and smart pointers.
2. What are the risks of using raw pointers?
3. How does `std::unique_ptr` prevent memory leaks?
4. Can a pointer point to a function? Explain.
5. Why are smart pointers preferred in modern C++?

Programming Questions

Easy

1. Declare a pointer to an `int` and assign it the address of a variable.
2. Dereference the pointer and print the value.
3. Make the pointer point to `nullptr`.
4. Print the size of an `int*`.
5. Use a pointer to modify the value of a variable.

Medium

1. Use pointer arithmetic to traverse an array.

2. Swap two numbers using pointers.
3. Pass an array to a function using a pointer and print its elements.
4. Dynamically allocate memory for an `int` using `new`.
5. Write a function that returns a pointer to a local static variable.

Hard

1. Implement a dynamic array using `new` and `delete`.
2. Use `std::unique_ptr` to manage an object.
3. Create a pointer to a function and call it.
4. Write a function that returns a `unique_ptr`.

---

6. Functions in C++

Theory Questions

Easy

1. What is a function?
2. What is function overloading?
3. What is the difference between call by value and call by reference?
4. What is a return type?
5. What is recursion?

Medium

1. Can we overload functions based on return type? Why or why not?
2. What is a lambda function?
3. When is call by reference more efficient than call by value?

4. What are default arguments in functions?

Hard

1. Explain how function overloading works with type promotion.
2. How do lambda functions capture variables by value and by reference?
3. Can a recursive function be optimized by the compiler?

Programming Questions

Easy

1. Write a function to add two numbers and return the result.
2. Write a function that swaps two numbers by reference.
3. Write a recursive function to calculate factorial.
4. Write a lambda that adds two numbers and call it.

Medium

1. Overload a function to handle \`int\`, \`float\`, and \`double\`.
2. Use a lambda with capture to modify a local variable.
3. Write a function that returns a reference to a static variable.
4. Write a recursive function to compute Fibonacci.
5. Use \`std::function\` to store a lambda.

Hard

1. Write a function template with overloading for different types.
2. Implement a recursive function to reverse a string.
3. Use a lambda in \`std::sort\` to sort a vector in descending order.
4. Write a function that takes a function pointer as a parameter.
5. Create a functor class and use it like a function.

7. Classes and Structures

Theory Questions

Easy

1. What is a class?
2. What is the default access specifier in a class?
3. What is a constructor?
4. What is a destructor?
5. What is a friend function?

Medium

1. Explain encapsulation using access specifiers.
2. Why are constructors useful?
3. What is a static member?
4. Can a constructor be private? When?
5. What is a constant member function?

Hard

1. Explain how friend functions violate encapsulation but are still useful.
2. Why can't we have static constructors?
3. How does `const` object restrict member function calls?
4. What is the order of constructor and destructor calls in objects?
5. Can a destructor be overloaded? Why?

Programming Questions

Easy

1. Define a class `Circle` with radius and a method to compute area.

2. Add a constructor to initialize radius.
3. Add a destructor that prints a message.

4. Make the radius private and provide public accessors.
5. Create an object and call area function.

Medium

1. Add a static member to count the number of `Circle` objects.
2. Add a constant member function to get radius.
3. Create a friend function to compare two circles by area.
4. Create a constant object and call const member functions.
5. Use initializer list in constructor.

Hard

1. Implement a class with a private constructor and a static factory method.
2. Write a friend class that can access private members of another class.
3. Create a function and return it call by value, pointer and reference.
4. Overload `+` operator using friend function.
5. Implement a singleton pattern using a private constructor.

---

8. Inheritance

Theory Questions

Easy

1. What is inheritance?
2. What is a base class and derived class?
3. What is single inheritance?

4. What is the role of `protected` access specifier?
5. What is multilevel inheritance?

Medium

1. Explain multiple inheritance with syntax.
2. What is the diamond problem?
3. How does access specifier affect inheritance?
4. What is an abstract class?
5. Can a derived class access private members of the base class?

Hard

1. How does C++ solve the diamond problem with virtual inheritance?
2. Why are virtual functions needed in abstract classes?
3. Can we instantiate an abstract class?
4. Explain the difference between interface and abstract class in C++.
5. How does inheritance affect constructor calling order?

Programming Questions

Easy

1. Create a base class `Animal` and derived class `Dog`.
2. Inherit `Dog` from `Animal` and add a method `bark()`.
3. Use `protected` members in base class.
4. Create objects of both classes.
5. Call base and derived methods.

Medium

1. Implement multilevel inheritance: `Vehicle → Car → SportsCar`.
2. Use `public`, `private`, and `protected` inheritance and observe access.
3. Add constructors in each class and observe calling order.

4. Create an abstract class `Shape` with pure virtual `area()`.
5. Derive `Circle` and `Rectangle` from `Shape`.

Hard

1. Implement multiple inheritance: `Student` and `Employee` → `WorkingStudent`.
2. Solve diamond problem using virtual inheritance.
3. Use virtual destructors in base class.
4. Override virtual functions in derived classes.
5. Store derived objects in base class pointers and call virtual functions.

---

9. Data Structures and Algorithms

Theory Questions

Easy

1. What is a data structure?
2. What is the difference between linear and non-linear DS?
3. What is an algorithm?
4. What is a static data structure?
5. Name two linear data structures.

Medium

1. Why are data structures important in programming?
2. Compare arrays and linked lists.
3. What is time complexity?
4. What is dynamic memory allocation used for?
5. Give a real-world example of a stack.

Hard

1. How do data structures affect algorithm efficiency?

2. Explain how a hash table works in brief.
3. Why is choosing the right data structure crucial for performance?
4. What are the trade-offs between static and dynamic structures?
5. How do trees and graphs model real-world problems?

---

10. Algorithm: Complexity Analysis, Asymptotic Notations, and Efficiency

Theory Questions

Easy

1. What is time complexity?
2. What does Big-O notation represent?
3. What is the time complexity of a single loop running `n` times?
4. What is space complexity?
5. What is the difference between worst-case and best-case complexity?

Medium

1. Explain the meaning of Ω (Omega) and θ (Theta) notations.
2. What is the difference between O(n) and o(n)?
3. Why do we ignore constants and lower-order terms in asymptotic analysis?
4. What is the time complexity of nested loops where both run `n` times?
5. How does recursion affect time and space complexity?

Hard

1. Explain the Master Theorem and when it is applicable.

2. Prove that `$5n^2 + 3n + 10$` is `$O(n^2)$` using the formal definition.
3. What is the significance of tight bounds (θ) in algorithm analysis?

4. Why is `$f(n) = n^2$` in `$\omega(n)$` but not in `$\omega(n^2)$`?
5. Discuss the trade-off between time and space with a real example (e.g., memoization).

Programming Questions

Easy

1. Write a function with a single `for` loop and determine its time complexity.
2. Write a function with two sequential loops (each `n` iterations) and find complexity.
3. Write a function that prints "Hello" 10 times — what is its time complexity?
4. Write a function to find the maximum in an array of size `n` and analyze it.
5. Write a function with constant time — `O(1)` — and explain why.

Medium

1. Write a function with two nested loops (each from 0 to `n`) and compute its time complexity.
2. Implement binary search iteratively and analyze its time complexity.
3. Write a recursive function for factorial and calculate its time and space complexity.
4. Write a function that sums all elements in a 2D matrix (`n x n`) and analyze it.
5. Given a function with three nested loops, express its complexity in Big-O.

Hard

1. Apply the Master Theorem to solve: `T(n) = 2T(n/2) + n`. Identify `a`, `b`, `f(n)` and find complexity.
2. Solve `$T(n) = 3T(n/4) + n^2$` using Master Theorem. State which case applies.
3. Write a recursive Fibonacci function (without memoization) and analyze its time complexity using a recursion tree.

4. Implement merge sort and derive its time complexity using recurrence relation and Master Theorem.
5. Write a function with the recurrence `T(n) = T(n-1) + n` and solve it using iteration method.

Bonus: Practice Problems for Complexity Computation (Theory + Programming)

Theory (Hard)

1. Compare the growth rates: `n`, `n log n`, `$n^2$`, `$2^n$`, `log n`. Arrange in increasing order.
2. Is `O(log n)` always better than `O(n)`? Justify with an example where constants matter.
3. What is the complexity of an algorithm with `T(n) = T(n/3) + 1`? Solve using iteration.
4. Why is `θ(n)` considered a "tight bound"?
5. Can an algorithm have different time and space complexity? Give an example.

Programming (Hard)

1. Write a function that checks if a number is prime using a loop up to `√n`. Analyze time complexity.
2. Implement a function that prints all pairs from an array of size `n`. What is its complexity?
3. Write a recursive function for `T(n) = 2T(n/2) + 1` and verify complexity experimentally for `n=8, 16, 32`.
4. Create a function with time complexity `O(n log n)` and justify it with operation counting.
5. Write a space-inefficient version of Fibonacci using recursion and compare space usage with iterative version.