LEARNIFY
VADODARA , GUJARAT
Learnify887@gmail.com

Question C++

Part A: Theory Questions (50 Total)

Easy Level (Fundamentals & Basics - 20 Qs)

1. What is the function of the #include <iostream> directive?
2. What is the purpose of the return 0; statement in the main() function?
3. What is the difference between a variable declaration and a variable definition?
4. Explain the purpose of the unsigned keyword.
5. What is the value of 5 / 2 when both operands are integers?
6. What is the primary difference between a for loop and a while loop?
7. What is a function prototype?
8. What is a C-style string?
9. What is a scope resolution operator (::)?
10. How is a variable passed by value to a function?
11. What is the fundamental difference between a pointer and a reference?
12. What are the two access specifiers introduced in Lesson 15 (Classes and Objects)?
13. What is the primary role of a constructor?
14. What is the syntax for a class destructor?
15. Which character is used to dereference a pointer?

16. What does std::endl do that \n does not?
17. In a switch statement, what is the role of the break keyword?
18. What are Getters and Setters used for?
19. What is a namespace?
20. How is an element in an array accessed?

Intermediate Level (OOP & Data Structures - 20 Qs)

21. Define Encapsulation and state its goal in C++.
22. How is a member function declared as const? What does this imply?
23. What is the difference between a struct and a class in C++?
24. Explain the concept of Function Overloading.
25. In the context of inheritance, explain the "is-a" relationship.
26. What is the significance of declaring a base class destructor as virtual?
27. Define Polymorphism. Name its two main types in C++.
28. What is a static data member? Where is it defined/initialized?
29. Why must the stream insertion operator (<<) often be overloaded as a friend function?
30. Explain the concept of Pass-by-Reference using the & operator in function parameters.
31. What is the fundamental difference in access between private and protected members in an inheritance hierarchy?
32. What is the purpose of the try, throw, and catch keywords?
33. What is an Abstract Base Class (ABC)? How is it defined?
34. Name two types of standard STL sequential containers.
35. What is the average time complexity for searching a key in an std::unordered_map? Why?

36. What is the primary difference between std::vector and std::list in terms of insertion/deletion performance?
37. When using file I/O, what is the role of the std::ios::app flag?
38. What is a std::stack? What is the principle it enforces?
39. What is a shallow copy? Why can it be dangerous?
40. What is a Lambda Expression? What is the purpose of its capture list?

Advanced Level (Modern C++ & Concurrency - 10 Qs)

41. Define the RAII principle. Give a real-world C++ example of its use.
42. What is the difference between std::unique_ptr and std::shared_ptr regarding ownership?
43. What is the purpose of Move Semantics? What is an Rvalue reference (&&)?
44. What does std::move(some_object) actually do?
45. Why must a template function's definition typically be included entirely in a header file?
46. What is a race condition in multithreading? How is it typically prevented in C++?
47. What is the purpose of std::lock_guard? How is it related to RAII?
48. What is the utility of a non-type template parameter? Give an example.
49. What are include guards (e.g., using #ifndef)?
50. What is a virtual table (v-table)?

Part B: Code Questions (50 Total)

Easy Level (Fundamentals & Basics - 20 Qs)

1. I/O and Variables: Write a program that prompts the user for their age, reads the input into an int, and prints it back.
2. Operators: Write a program that calculates (10 * 3) + (20 % 3) and stores the result in an int. Print the result.
3. Conditionals: Write an if...else block that prints "Even" if an integer num is divisible by 2, and "Odd" otherwise.
4. for Loop: Use a for loop to print the numbers 10 down to 1.
5. while Loop: Use a while loop to print "Hello" three times.
6. Function: Write a function sum(int a, int b) that takes two integers and returns their sum. Call it in main().
7. Array Access: Initialize an integer array data with values {10, 20, 30, 40}. Change the value at index 2 to 99.
8. Pointers: Declare an int variable val = 100. Declare an int pointer ptr and make it point to val. Print the value of val using ptr.
9. References: Declare an int count = 5. Declare an int reference alias initialized to count. Modify alias to 10 and print count.
10. std::string: Declare a std::string s1 = "Hello", s2 = "World". Concatenate them into a new string s3 (with a space) and print s3.
11. do-while Loop: Write a do-while loop that asks the user for a number and repeats until the number entered is greater than 5.
12. switch: Write a switch statement that checks a char variable grade. If it is 'A', print "Excellent", otherwise print "Pass".

13. Global/Local Scope: Declare a global variable g = 5. Inside main(), declare a local variable g = 10. Print the value of the global variable using the scope resolution operator.
14. Unary Operators: Given int x = 5;, calculate and print y where y = ++x * 2;.
15. Structs: Define a struct Point { int x; int y; };. Create a Point variable p and initialize its members to 1 and 2.
16. Function Prototype: Provide the correct function prototype for a function named is_valid that takes a float and a bool and returns a bool.
17. C-Style String: Initialize a char array word with the literal "Code". Print the character at index 1.
18. Conditional Operator: Use the conditional (ternary) operator to set a string variable status to "Active" if bool is_on is true, or "Inactive" otherwise.
19. Type Casting: Given double d = 3.14;, explicitly cast it to an int variable i and print i.
20. Function Call: Write a function print_hello() that takes no arguments and returns void. Call it in main().

Intermediate Level (OOP & Data Structures - 20 Qs)

21. Basic Class & Encapsulation: Define a class Student with a private int score. Implement public methods setScore(int s) (with validation) and getScore().
22. Constructor: For the Student class above, implement a parameterized constructor Student(int s) that uses the setScore method for initialization.
23. Function Overloading: Write two overloaded functions named add. One takes two ints, and the other takes two doubles, and both return the sum.

24. Destructor: Create a class Resource that prints "Resource Acquired" in its constructor and "Resource Released" in its destructor. Show the output when an object of Resource is created in main().
25. this Pointer: In a class Item, implement a setter method void setID(int id) where the parameter name id is the same as the private member name id. Use this to resolve the ambiguity.
26. Inheritance: Create a base class Shape with a protected string name. Derive a class Circle that publicly inherits from Shape. Implement Circle's constructor, ensuring Shape's constructor is called.
27. Polymorphism (Virtual): In the Shape base class above, declare a virtual function draw(). In the Circle derived class, override draw() to print "Drawing Circle".
28. Abstract Base Class: Modify the Shape base class to be an Abstract Base Class by declaring virtual double area() = 0;.
29. Static Member: Create a class Counter with a private static int count initialized to 0. Implement a static method getCount() and a constructor that increments count.
30. Operator Overloading (+): Create a class Vector2D {int x, y;}. Overload the addition operator (+) as a member function to add two Vector2D objects.
31. Operator Overloading (<<): For the Vector2D class, overload the stream insertion operator (<<) as a friend function to print the vector as (x, y).
32. Exception Handling: Write a function safeDivide(int num, int den) that throws a C-style string "Error: Division by zero" if den is 0. Call it inside a try...catch block.
33. File I/O (Output): Use std::ofstream to write the text "Test Data" to a file named "output.txt".
34. File I/O (Input): Use std::ifstream to read an entire line from the file "output.txt" created in the previous question, and print it to the console.

35. std::vector: Create a std::vector<double>. Use push_back to add the values 1.1, 2.2, 3.3. Iterate through the vector using a range-based for loop and print the elements.
36. std::map: Create a std::map<int, std::string> to store IDs and names. Insert two entries (1 "Alpha", 2 "Beta") and print the name associated with ID 1.
37. std::set: Create a std::set<int>. Insert the numbers 5, 1, 5. Check and print whether the number 1 exists in the set.
38. std::stack: Create a std::stack<char>. Push the characters 'A' and 'B'. Pop and print the character that was pushed last.
39. std::queue: Create a std::queue<int>. Push 10 and 20. Access and print the element at the front without removing it.
40. Copy Constructor: Write a class Box with a double width. Implement a custom Copy Constructor that prints "Copy Constructor Called" and copies the width.

Advanced Level (Modern C++ & Concurrency - 10 Qs)

41. Function Template: Write a function template isEqual that takes two arguments of the same type T and returns true if they are equal.
42. Class Template: Write a class template Buffer that takes a type T and has a private member T data. Implement a constructor and a getter.
43. Smart Pointers (Unique): Create a std::unique_ptr<int> named uptr1 initialized with the value 42. Demonstrate that you cannot copy it, but you can transfer ownership to uptr2 using std::move.
44. Smart Pointers (Shared): Create two std::shared_ptr<double> objects, sptr1 and sptr2, that share ownership of a single dynamically allocated value (e.g., 99.9). Print the reference count for sptr1.

45. Lambdas (Capture): Given int offset = 5;, use std::for_each and a lambda that captures offset by value to add offset to every element in a std::vector<int> {1, 2, 3}.
46. Lambdas (Mutable): Write a lambda function that captures an integer count by value, is declared mutable, and increments and prints count every time the lambda is called.
47. Move Constructor: Write a minimal class DataWrapper with a private int* ptr. Implement the Move Constructor to steal the pointer from the source object and set the source's pointer to nullptr.
48. std::thread: Write a simple function thread_task() that prints "Task Finished". In main(), create an std::thread to execute this function and use .join() to wait for it.
49. Thread Synchronization: Given a global int total = 0; and a function increment_total(), write the code to protect the total variable using a std::lock_guard<std::mutex> inside increment_total.
50. STL Algorithm: Use the std::sort algorithm from <algorithm> to sort a std::vector<int> in descending order. (Hint: Use std::greater<int>() as the third argument or a custom lambda).