C++ Viva Questions & Answers - College Level

Section 1: Basic Concepts

Q1. What is C++?

Answer: C++ is an object-oriented programming language developed by Bjarne Stroustrup in 1979. It is an extension of C language with features like classes, objects, inheritance, polymorphism, and encapsulation. It supports both procedural and object-oriented programming paradigms.

Q2. What is the difference between C and C++?

Answer:

- C is procedural; C++ is both procedural and object-oriented
- C has no concept of classes/objects; C++ supports OOP
- C uses scanf/printf; C++ uses cin/cout
- C has no function overloading; C++ supports it
- C has no reference variables; C++ has references

Q3. What are the main features of C++?

Answer:

- Object-Oriented Programming (Encapsulation, Inheritance, Polymorphism)
- Rich library support (STL)
- Operator and function overloading
- Exception handling
- Templates for generic programming
- Memory management features

Q4. What is the difference between struct and class?

Answer:

- In struct, members are public by default; in class, they are private by default
- Both can have constructors, destructors, and member functions
- Structs are traditionally used for passive data; classes for active objects

Q5. What are access specifiers in C++?

Answer:

- **public:** Members accessible from anywhere
- **private:** Members accessible only within the class
- **protected:** Members accessible within the class and derived classes

---

Section 2: Input/Output

Q6. What is the difference between cout and printf?

Answer:

- cout is a stream object (type-safe); printf is a function
- cout automatically identifies data type; printf needs format specifiers
- cout is part of iostream; printf is from stdio.h
- cout supports operator overloading; printf doesn't

Q7. Write a simple program to read and display a number.

Answer:

```cpp
#include <iostream>
using namespace std;

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    cout << "You entered: " << num;
    return 0;
}
```

Section 3: Functions

Q8. What is function overloading?

Answer: Function overloading allows multiple functions with the same name but different parameters (number, type, or order). The compiler differentiates them based on the function signature.

Example:

```cpp
```

```cpp
int add(int a, int b) { return a + b; }
double add(double a, double b) { return a + b; }
int add(int a, int b, int c) { return a + b + c; }
```

Q9. What is the difference between call by value and call by reference?

Answer:

- **Call by value:** Copy of variable is passed; original remains unchanged
- **Call by reference:** Address is passed; original variable can be modified

Example:

```cpp
void byValue(int x) { x = 10; } // Original unchanged
void byReference(int &x) { x = 10; } // Original modified
```

Q10. What is an inline function?

Answer: Inline function is a function whose code is inserted at the point of call rather than being called. It's defined using the `inline` keyword. It reduces function call overhead but increases code size.

```cpp
inline int square(int x) {
    return x * x;
}
```

Section 4: Object-Oriented Programming

Q11. What is a class and object?

Answer:

- **Class:** Blueprint or template that defines properties and behaviors
- **Object:** Instance of a class that occupies memory

Example:

```cpp
```

```
class Student {
   int rollNo;
   string name;
};

Student s1;  // s1 is an object
```

Q12. What is encapsulation?

Answer: Encapsulation is wrapping data (variables) and code (functions) together as a single unit (class). It hides internal implementation and protects data using access specifiers.

Q13. What is abstraction?

Answer: Abstraction means showing only essential features and hiding implementation details. It's achieved through abstract classes and interfaces. Example: You know a car has brakes, but you don't need to know how the braking system works internally.

Q14. What is inheritance?

Answer: Inheritance is a mechanism where one class acquires properties and behaviors of another class. It promotes code reusability.

Types: Single, Multiple, Multilevel, Hierarchical, Hybrid

Example:

```
cpp

class Animal {  // Base class
public:
   void eat() { cout << "Eating..."; }
};

class Dog : public Animal {  // Derived class
public:
   void bark() { cout << "Barking..."; }
};
```

Q15. What is polymorphism?

Answer: Polymorphism means "many forms" - ability to take different forms. Two types:

- **Compile-time (Static):** Function/Operator overloading
- **Run-time (Dynamic):** Virtual functions

Q16. What is the difference between function overloading and function overriding?

Answer:

- **Overloading:** Same function name, different signatures, same class (compile-time)
- **Overriding:** Same function signature in base and derived class (run-time)

Section 5: Constructors & Destructors

Q17. What is a constructor?

Answer: A constructor is a special member function with the same name as the class, automatically called when an object is created. It initializes objects. It has no return type.

Types: Default, Parameterized, Copy Constructor

Q18. What is a destructor?

Answer: A destructor is a special member function with the same name as the class prefixed with `~`. It's automatically called when an object goes out of scope. Used to release resources.

```cpp
class Demo {
public:
    Demo() { cout << "Constructor called"; }
    ~Demo() { cout << "Destructor called"; }
};
```

Q19. What is a copy constructor?

Answer: A copy constructor creates a new object as a copy of an existing object. It takes a reference to an object of the same class.

```cpp
class Box {
    int length;
public:
    Box(int l) : length(l) {}
    Box(const Box &b) {  // Copy constructor
        length = b.length;
    }
};
```

Q20. Can we overload constructors?

Answer: Yes, we can have multiple constructors with different parameters (constructor overloading).

Q21. What is a default constructor?

Answer: A constructor with no parameters. If no constructor is defined, the compiler provides a default

constructor.

Section 6: Pointers & Memory Management

Q22. What is a pointer?

Answer: A pointer is a variable that stores the memory address of another variable.

```cpp
int x = 10;
int *ptr = &x;  // ptr stores address of x
cout << *ptr;   // Prints 10 (dereferencing)
```

Q23. What is the difference between pointer and reference?

Answer:

- Pointer can be reassigned; reference cannot
- Pointer can be NULL; reference must refer to an object
- Pointer needs dereferencing (*); reference doesn't
- Pointer uses &, *; reference uses &

Q24. What is a null pointer?

Answer: A pointer that doesn't point to any valid memory location. In C++11 and later, use `nullptr` instead of NULL.

```cpp
int *ptr = nullptr;
```

Q25. What is dynamic memory allocation?

Answer: Allocating memory at runtime using `new` and deallocating using `delete`.

```cpp
int *ptr = new int;       // Single variable
int *arr = new int[10];   // Array

delete ptr;               // Free single variable
delete[] arr;             // Free array
```

Q26. What is a memory leak?

Answer: Memory leak occurs when dynamically allocated memory is not freed, causing memory

wastage. Always use `delete` to free memory allocated with `new`.

Q27. What is the this pointer?

Answer: `this` is a pointer that points to the current object. It's implicitly passed to all non-static member functions.

```cpp
class Box {
   int length;
public:
   void setLength(int length) {
      this->length = length;  // Distinguishes member from parameter
   }
};
```

Section 7: Virtual Functions & Polymorphism

Q28. What is a virtual function?

Answer: A virtual function is a member function in the base class that you expect to override in derived classes. It enables runtime polymorphism. Declared using the `virtual` keyword.

```cpp
class Base {
public:
   virtual void show() { cout << "Base"; }
};

class Derived : public Base {
public:
   void show() override { cout << "Derived"; }
};
```

Q29. What is a pure virtual function?

Answer: A virtual function with no implementation in the base class, declared by assigning 0. It makes the class abstract.

```cpp
class Shape {
public:
   virtual void draw() = 0;  // Pure virtual function
};
```

Q30. What is an abstract class?

Answer: A class that contains at least one pure virtual function. It cannot be instantiated. Used to provide a base for derived classes.

Q31. What is the difference between virtual function and pure virtual function?

Answer:

- Virtual function has implementation; pure virtual has none
- Class with virtual function can be instantiated; class with pure virtual cannot
- Pure virtual function must be overridden; virtual function is optional

---

Section 8: Operator Overloading

Q32. What is operator overloading?

Answer: Operator overloading allows operators to work with user-defined types (classes). It gives special meaning to operators for custom types.

```cpp
class Complex {
    int real, imag;
public:
    Complex operator+(Complex &c) {
        Complex temp;
        temp.real = real + c.real;
        temp.imag = imag + c.imag;
        return temp;
    }
};
```

Q33. Which operators cannot be overloaded?

Answer: `::` (scope resolution), `.` (member selection), `.*` (member pointer selector), `?:` (ternary), `sizeof`, `typeid`

Q34. Can we overload the assignment operator?

Answer: Yes, the assignment operator (=) can be overloaded. It's used to copy values from one object to another.

Section 9: Templates

Q35. What are templates?

Answer: Templates allow writing generic and reusable code. They work with any data type.

Types: Function templates and Class templates

Q36. What is a function template?

Answer: A blueprint for creating functions that work with different data types.

```cpp
template <typename T>
T maximum(T a, T b) {
    return (a > b) ? a : b;
}

// Usage
maximum(10, 20);       // int
maximum(10.5, 20.3);   // double
```

Q37. What is a class template?

Answer: A blueprint for creating classes that work with different data types.

```cpp
template <class T>
class Stack {
    T arr[100];
    int top;
public:
    void push(T x);
    T pop();
};

Stack<int> intStack;
Stack<string> strStack;
```

Section 10: Exception Handling

Q38. What is exception handling?

Answer: Exception handling is a mechanism to handle runtime errors gracefully using try, catch, and throw keywords.

```cpp
try {
    int x = 10, y = 0;
    if (y == 0)
        throw "Division by zero!";
    cout << x / y;
}
catch (const char* msg) {
    cout << "Error: " << msg;
}
```

Q39. What are the keywords used in exception handling?

Answer:

- **try:** Block containing code that might throw an exception
- **catch:** Block that handles the exception
- **throw:** Used to throw an exception

Q40. What is the difference between error and exception?

Answer:

- **Error:** Serious problems that applications shouldn't catch (e.g., out of memory)
- **Exception:** Conditions that applications can catch and handle (e.g., division by zero)

Section 11: Modern C++ Features (C++11/14/17)

Q41. What is the auto keyword?

Answer: `auto` automatically deduces the type of a variable from its initializer (C++11).

```cpp
auto x = 10;          // int
auto y = 3.14;        // double
auto str = "Hello";   // const char*
auto vec = vector<int>{1, 2, 3};
```

Q42. What is a range-based for loop?

Answer: A simpler way to iterate over containers (C++11).

```cpp
```

```
vector<int> v = {1, 2, 3, 4, 5};
for (auto x : v) {
    cout << x << " ";
}
```

Q43. What is nullptr?

Answer: `nullptr` is a keyword representing a null pointer constant (C++11). It's type-safe compared to NULL.

```cpp
int *ptr = nullptr;  // Better than NULL
```

Q44. What are smart pointers?

Answer: Smart pointers automatically manage memory, preventing memory leaks (C++11).

Types:

- **unique_ptr:** Exclusive ownership
- **shared_ptr:** Shared ownership
- **weak_ptr:** Non-owning reference

```cpp
#include <memory>

unique_ptr<int> ptr1(new int(10));
shared_ptr<int> ptr2 = make_shared<int>(20);
```

Q45. What is move semantics?

Answer: Move semantics (C++11) allows transferring resources from one object to another without copying, improving performance.

```cpp
vector<int> v1 = {1, 2, 3};
vector<int> v2 = std::move(v1);  // v1 is now empty
```

Q46. What is a lambda expression?

Answer: Anonymous functions defined inline (C++11).

```cpp
```

```cpp
auto add = [](int a, int b) { return a + b; };
cout << add(5, 3);  // Output: 8

// With capture
int x = 10;
auto multiply = [x](int a) { return a * x; };
```

Section 12: STL (Standard Template Library)

Q47. What is STL?

Answer: STL is a collection of template classes and functions providing common data structures and algorithms.

Components: Containers, Iterators, Algorithms, Functors

Q48. What are the types of containers in STL?

Answer:

- **Sequence:** vector, list, deque, array
- **Associative:** set, map, multiset, multimap
- **Unordered:** unordered_set, unordered_map
- **Adapters:** stack, queue, priority_queue

Q49. What is a vector?

Answer: Vector is a dynamic array that can resize automatically. It provides random access and fast insertion at the end.

```cpp
#include <vector>
vector<int> v = {1, 2, 3};
v.push_back(4);
cout << v[0];  // Access element
```

Q50. What is the difference between vector and array?

Answer:

- Vector is dynamic; array is static
- Vector can resize; array has fixed size
- Vector is slower due to dynamic allocation

- Vector provides more member functions

Section 13: Code Output Questions

Q51. What is the output?

```cpp
int x = 10;
int &ref = x;
ref = 20;
cout << x;
```

Answer: 20 (reference modifies the original variable)

Q52. What is the output?

```cpp
class A {
public:
    A() { cout << "A "; }
    ~A() { cout << "~A "; }
};

int main() {
    A obj;
    return 0;
}
```

Answer: A ~A (Constructor called, then destructor when object goes out of scope)

Q53. What is the output?

```cpp
int x = 5;
int *ptr = &x;
cout << *ptr << " " << ptr;
```

Answer: 5 [memory address] (Value and address of x)

Q54. What is the output?

```cpp
```

```cpp
class Base {
public:
    virtual void show() { cout << "Base "; }
};

class Derived : public Base {
public:
    void show() { cout << "Derived "; }
};

int main() {
    Base *ptr;
    Derived d;
    ptr = &d;
    ptr->show();
}
```

Answer: Derived (Virtual function enables runtime polymorphism)

Q55. What is the output?

```cpp
int arr[] = {1, 2, 3, 4, 5};
cout << arr[2] << " " << *(arr + 2);
```

Answer: 3 3 (Both access the third element)

Section 14: Scenario-Based Questions

Q56. When should you use a virtual destructor?

Answer: Always use virtual destructor in a base class when you have virtual functions or when the class is meant to be inherited. This ensures proper cleanup of derived class objects when deleted through a base class pointer.

Q57. What happens if you don't delete dynamically allocated memory?

Answer: Memory leak occurs. The allocated memory remains occupied but inaccessible, wasting system resources. In long-running programs, this can exhaust available memory.

Q58. Explain the diamond problem in multiple inheritance.

Answer: Diamond problem occurs when a class inherits from two classes that both inherit from a common base class, causing ambiguity. Solved using virtual inheritance.

```cpp
```

```
class A { };
class B : virtual public A { };
class C : virtual public A { };
class D : public B, public C { };  // No ambiguity
```

Q59. Why is it recommended to use smart pointers over raw pointers?

Answer:

- Automatic memory management (no memory leaks)
- Exception-safe (memory freed even if exception occurs)
- Clear ownership semantics
- Prevents dangling pointers

Q60. When would you use a const member function?

Answer: Use const member function when the function doesn't modify any member variables. It allows the function to be called on const objects.

```cpp
class Box {
   int length;
public:
   int getLength() const { return length; }  // Doesn't modify members
};
```

Section 15: Advanced Concepts

Q61. What is RAII?

Answer: Resource Acquisition Is Initialization. Resources (memory, files) are acquired in the constructor and released in the destructor. This ensures proper cleanup even during exceptions.

Q62. What is the difference between deep copy and shallow copy?

Answer:

- **Shallow copy:** Copies pointer address (both objects share same memory)
- **Deep copy:** Creates a new copy of the pointed data

Q63. What are static members?

Answer: Static members belong to the class rather than any object. Only one copy exists regardless of the number of objects created.

```cpp
class Counter {
    static int count;
public:
    Counter() { count++; }
    static int getCount() { return count; }
};
int Counter::count = 0;
```

Q64. What is a friend function?

Answer: A non-member function that has access to private and protected members of a class. Declared using the `friend` keyword.

```cpp
class Box {
    int length;
    friend void printLength(Box b);
};

void printLength(Box b) {
    cout << b.length;  // Can access private member
}
```

Q65. What is multiple inheritance?

Answer: A class inheriting from more than one base class. Can lead to ambiguity and the diamond problem.

```cpp
class A { };
class B { };
class C : public A, public B { };  // Multiple inheritance
```

Section 16: Compilation & Linking

Q66. What is the difference between compilation and linking?

Answer:

- **Compilation:** Converts source code to object code (.cpp to .o/.obj)
- **Linking:** Combines object files and libraries into executable

Q67. What are header guards?

Answer: Mechanism to prevent multiple inclusion of the same header file.

```cpp
#ifndef MYHEADER_H
#define MYHEADER_H
// Header content
#endif
```

Q68. What is the difference between #include <> and #include ""?

Answer:

- <>: Searches in system directories (for standard libraries)
- "": Searches in current directory first (for user-defined headers)

Section 17: Miscellaneous

Q69. What is the size of an empty class?

Answer: 1 byte (to ensure each object has a unique address)

Q70. What is function hiding in C++?

Answer: When a derived class defines a function with the same name as base class function, it hides all overloaded versions of that function from the base class.

Q71. Can main() function be overloaded?

Answer: No, main() cannot be overloaded in C++.

Q72. What is the difference between new/delete and malloc/free?

Answer:

- new/delete are operators; malloc/free are functions
- new calls constructor; malloc doesn't
- delete calls destructor; free doesn't
- new is type-safe; malloc returns void*

Q73. What are the differences between struct and class in C++?

Answer: Only difference is default access: struct members are public by default, class members are private. Both support all OOP features.

Q74. What is namespace?

Answer: Namespace is a declarative region that provides scope to identifiers. It prevents naming conflicts.

```cpp
namespace MySpace {
    int value = 10;
}

cout << MySpace::value;
```

Q75. What is the purpose of the explicit keyword?

Answer: Prevents implicit type conversions for constructors with single parameters.

```cpp
class MyClass {
public:
    explicit MyClass(int x) { }
};

MyClass obj = 10;  // Error: implicit conversion prevented
MyClass obj(10);   // OK
```

Real-World Application Questions

Q76. Where is polymorphism used in real applications?

Answer:

- GUI frameworks (button clicks, event handling)
- Game development (different character behaviors)
- Database drivers (same interface, different databases)
- Plugin systems

Q77. Why is exception handling important in production code?

Answer:

- Prevents program crashes
- Provides graceful error recovery
- Separates error handling from normal code

- Improves debugging and maintenance

Q78. How do smart pointers help in large-scale applications?

Answer:

- Prevent memory leaks automatically
- Make code exception-safe
- Clarify ownership semantics
- Reduce debugging time
- Improve code reliability

Q79. When would you choose inheritance over composition?

Answer:

- Use inheritance for "is-a" relationships (Dog is an Animal)
- Use composition for "has-a" relationships (Car has an Engine)
- Prefer composition for flexibility and loose coupling

Q80. Why use const correctness in professional C++ code?

Answer:

- Prevents accidental modifications
- Enables compiler optimizations
- Makes code intent clear
- Allows functions to work with const objects
- Improves code safety and maintainability

Quick Tips for Viva Success

1. **Understand, don't memorize:** Focus on concepts, not just syntax
2. **Practice code writing:** Be ready to write code on paper
3. **Explain with examples:** Always support answers with simple examples
4. **Know your practical code:** Be thoroughly familiar with programs you've written
5. **Stay calm:** If you don't know an answer, say so politely and try to relate it to what you know
6. **Use proper terminology:** Use correct technical terms (e.g., "member function" not just "function")
7. **Be concise:** Give clear, direct answers; elaborate only if asked

Good luck with your viva!