**Q. 1. What is the purpose of including #include<bits/stdc++.h> in a C++ program?**

A) It is used to include all standard C++ library headers.  
B) It is used to include only specific standard C++ headers required for a program.  
C) It is a user-defined header for convenience.  
D) It is used to import only the most commonly used functions in C++.

================================================================

**Q. 2. When should you prefer using macros over inline functions in C++?**

A) When you need type safety and debugging support.  
B) When you need to define platform-dependent code or use preprocessing directives.  
C) When you want to limit the scope of a function.  
D) When you need to ensure no side effects in the code.

==============================================================

**Q. 3. Inheritance and Method Overriding**

You have a base class Shape with a virtual method draw() and a derived class Circle that overrides draw(). You create an object of Circle but store it in a pointer of type Shape\*. What happens when you call draw() using the Shape\* pointer?

A) It will call the draw() method of the Shape class.  
B) It will call the draw() method of the Circle class.  
C) It will not compile due to a method signature mismatch.  
D) It will result in a runtime error.

==================================================================

**Q. 4. Abstract Classes**

Consider the following abstract class in C++:

class Shape {

virtual void draw() = 0;

};

What will happen if you try to instantiate the Shape class directly?

A) It will compile but cannot be instantiated.  
B) It will result in a compilation error.  
C) It will run successfully, but will not call draw().  
D) It will give an error at runtime.

==================================================================

**Q. 5. Constructor Initialization**

Consider the following C++ code:

class MyClass {

int x;

public:

MyClass(int i) : x(i) {}

};

What is the purpose of the constructor initialization list (: x(i))?

A) It initializes the value of x with i.  
B) It calls the default constructor for x.  
C) It initializes x after the constructor body is executed.  
D) It does not do anything, and x gets assigned inside the constructor body.

==================================================================

**Q. 6. Multiple Inheritance**

You have two classes ClassA and ClassB, both of which inherit from a base class Base. If ClassA and ClassB both define a method doSomething(), and you create an object of ClassA, which method will be called when you invoke doSomething()?

A) doSomething() from ClassA.  
B) doSomething() from ClassB.  
C) A compile-time error occurs due to ambiguity.  
D) The method will be inherited from Base.

==================================================================

**Q. 7. Polymorphism with Virtual Methods**

Given a base class Animal with a virtual method speak() and derived class Dog that overrides speak(), what is the behavior when a pointer to Animal that points to a Dog object calls speak()?

A) It will always call the speak() method of Animal.  
B) It will call the speak() method of Dog, due to dynamic binding.  
C) It will result in undefined behavior.  
D) It will compile but not run.

==================================================================

**Q. 8. Static Variables in a Class**

Consider a class Counter with a static member variable count that is incremented in the constructor. What happens when you create multiple instances of Counter?

A) Each instance will have its own count value.  
B) All instances will share the same count value.  
C) count will cause a compilation error because it’s static.  
D) The constructor will not execute.

==================================================================

**Q. 9. Copy Constructor**

What will happen if you try to copy an object that contains dynamically allocated memory without providing a custom copy constructor?

A) The program will crash due to a shallow copy.  
B) A deep copy of the object will be made automatically.  
C) It will result in undefined behavior.  
D) A compile-time error will occur.

==================================================================

**Q. 10. Operator Overloading**

What is the correct reason to overload the assignment operator in a class?

A) To allow the object to be compared with other objects.  
B) To allocate dynamic memory for each new instance.  
C) To define custom behavior when an object is assigned another object of the same class.  
D) To allow the class to handle multiple data types.

==================================================================

**Q. 11. Friend Functions**

In C++, friend functions can:

A) Access private and protected members of a class.  
B) Only access public members of a class.  
C) Only access static members of a class.  
D) Not be declared within a class.

==================================================================

**Q. 12. Memory Management and Destructor**

What happens if you fail to explicitly define a destructor in a class that dynamically allocates memory?

A) The compiler will automatically generate a destructor to handle the deallocation.  
B) It will result in a memory leak.  
C) The destructor will be inherited from the base class.  
D) The object will be deleted without proper deallocation.

==================================================================

**Q. 13. Run-time Polymorphism**

Which of the following is true about run-time polymorphism in C++?

A) It can be achieved only through function overloading.  
B) It requires the use of static type checking.  
C) It requires the use of virtual functions in base classes.  
D) It can be achieved only through operator overloading.

.

==================================================================

**Q. 14. Virtual Functions and Pure Virtual Functions**

What happens when a class contains a pure virtual function?

A) The class cannot be instantiated.  
B) The class will act as a concrete class.  
C) The class can be instantiated but will not have any concrete methods.  
D) The pure virtual function must be defined in the derived class.

==================================================================

**Q. 15. Dynamic Cast**

Which of the following is correct regarding the dynamic\_cast operator in C++?

A) It is used for converting base class pointers to derived class pointers during compile-time.  
B) It throws an exception if the cast is not possible.  
C) It can only be used with objects of class type.  
D) It is used for converting any pointer or reference type to another, regardless of inheritance.

==================================================================

**Q. 16. Abstract Class**

Which of the following statements is true about abstract classes in C++?

A) An abstract class cannot have any member functions.  
B) An abstract class can only have pure virtual functions.  
C) An abstract class can have both pure virtual and concrete functions.  
D) You can create instances of an abstract class directly.

==================================================================

**Q. 17. Exception Handling**

In C++, which of the following operators we can’t overload?

A) +  
B) -  
C) .  
D) %

==================================================================

**Q. 18. Custom Exception Class**

What is the purpose of creating a custom exception class in C++?

A) To define custom error messages and provide more specific exception handling.  
B) To override the default behavior of the std::exception class.  
C) To avoid the use of try and catch blocks.  
D) To automatically handle all types of exceptions.

==================================================================

**Q. 19. Formatted I/O Operations**

Which of the following is a C++ stream manipulator used to set the width of the output?

A) setw  
B) setprecision  
C) fixed  
D) endl

==================================================================

**Q. 20. File Handling**

Which function is used to open a file in C++?

A) open()  
B) ifstream::open()  
C) file.open()  
D) fstream::open()

==================================================================

**Q. 21. Templates**

Which of the following is the correct way to define a template function in C++?

A) template <class T> void func(T arg)  
B) template <T> void func(T arg)  
C) void template <class T> func(T arg)  
D) void func<T>(T arg)

==================================================================

**Q. 21. STL: Stack**

Which of the following C++ Standard Library containers follows the Last In First Out (LIFO) principle?

A) queue  
B) vector  
C) map  
D) stack

===========================================================================

**Q. 22.** Which of the following is **true** about destructors in C++?

A) A destructor can be overloaded.

B) A destructor is invoked explicitly.

C) A destructor has the same name as the class with a tilde (~) before it.

D) A destructor does not release memory used by objects

==================================================================

**Q. 23.** Given the following code:

class A {

public:

A() { cout << "Constructor A\n"; }

~A() { cout << "Destructor A\n"; }

};

class B : public A {

public:

B() { cout << "Constructor B\n"; }

~B() { cout << "Destructor B\n"; }

};

int main() {

B obj;

}

What will be the output?

1. Constructor A Constructor B Destructor B Destructor A
2. Constructor B Constructor A Destructor B Destructor A
3. Constructor A Destructor A Constructor B Destructor B
4. d) Constructor B Destructor B Constructor A Destructor A

==================================================================

**Q. 24.** What happens when a class inherits from another class with a private constructor?

A) The derived class can call the private constructor of the base class.

B) The derived class can’t call the private constructor of the base class directly.

C) The derived class can’t be instantiated.

D) The base class constructor becomes public.

**==========================================================**

**Q. 25.** Consider the following code:

class A {

int a;

public:

A(int x) : a(x) { }

void display() { cout << "A: " << a << endl; }

};

class B : public A {

int b;

public:

B(int x, int y) : A(x), b(y) { }

void display() { A::display(); cout << "B: " << b << endl; }

};

int main() {

B obj(10, 20);

obj.display();

}

What will be the output of this code?

A) A: 10 B: 20

B) A: 0 B: 20

C) A: 10 B: 0

D) A: 10 B: 10

**==========================================================**

**Q. 26.** Which of the following is true about a **pure virtual function** in C++?

A) It can have a definition in the base class.

B) It must be implemented in the derived class.

C) It has no declaration in the base class.

D) It allows objects of the base class to be instantiated.

**==========================================================**

**Q. 27.** Which of the following access specifiers ensures that a derived class has access to the base class’s members but not the outside world?

A) public

B) private

C) protected

D) friend

**==========================================================**

**Q. 28.** Which of the following is a key characteristic of **multilevel inheritance** in C++?

A) A class inherits from multiple classes

B) A class is derived from a base class which itself is derived from another class

C) Multiple derived classes inherit from a single base class

D) A class is derived from more than one base class

**==========================================================**

**Q. 29.** What will be the output of the following code?

#include<iostream>

using namespace std;

class A {

int x;

public:

A() { x = 5; }

virtual void show() { cout << "Class A: " << x << endl; }

};

class B : public A {

int y;

public:

B() { y = 10; }

void show() override { cout << "Class B: " << y << endl; }

};

int main() {

A\* ptr = new B();

ptr->show();

return 0;

}

A) Class A: 5

B) Class B: 10

C) Class B: 5

D) Class A: 10

**==========================================================**

**Q. 30. What happens if the new operator fails to allocate memory in C++?**

A) It throws a std::bad\_alloc exception (if exceptions are enabled).

B) It returns a NULL pointer.

C) It causes the program to terminate immediately.

D) It allocates memory on the stack instead.

**==========================================================**

**Q. 31. Which memory section is used for dynamically allocated memory in C++?**

A) Stack

B) Heap

C) Data Segment

D) Code Segment

==================================================================

**Q. 32. Which of the following statements is true regarding dynamic memory allocation in C++?**

A) Memory allocated using new can be freed automatically by the system.

B) malloc() in C++ is used for allocating memory on the heap but does not initialize memory.

C) C++ automatically frees dynamically allocated memory when the program exits.

D) The delete operator is not necessary when using new for memory allocation.

=================================================================

**Q. 33. What is the correct way to deallocate memory for a dynamically allocated array of integers in C++?**

A) delete ptr;

B) delete[] ptr;

C) free(ptr);

D) delete ptr[];

**==========================================================**

**Q. 34. What will happen if delete[] is not used for a dynamically allocated array in C++?**

A) The program will crash.

B) Memory will leak, causing potential issues with performance.

C) Only the first element of the array will be freed.

D) The memory will be freed automatically when the program ends.

**==========================================================**

**Q. 35. Which of the following is true about the delete operator in C++?**

A) It can only be used to free memory allocated by malloc().

B) It does not call the destructor of the object it deallocates.

C) It is used to free memory allocated by new.

D) It is used to deallocate memory on the stack.

**==========================================================**

**Q. 36. Which operator is used for dynamic memory allocation in C++?**

A) malloc()

B) calloc()

C) new

D) alloc()

**==========================================================**

**Q. 37.** What will be the output of the following code

**#include<stdio.h>**

**int main()**

**{**

**int first = 10;**

**int second = 20;**

**int third = 30;**

**{**

**int third = second - first;**

**printf("%d",third);**

**}**

**printf("%d",third);**

**}**

**A) 10 3**

**B) 30 10**

**C) 10 30**

**D) none of these**

**==========================================================**

**Q. 31. What will be the output of the code**

**#include<stdio.h>**

**int main()**

**{**

**int x = printf("IET,PUNE");**

**printf("\n%d",x);**

**return 0;**

**}**

**A) IET,PUNE**

**B) IET,PUNE 7**

**C) IET,PUNE 8**

**D) none of these**

**==========================================================**

**Q. 38. What will be the output of the code**

**#include<iostream>**

**using namespace std;**

**int main()**

**{**

**int x = 4;**

**if( x == 6);**

**x = 2;**

**cout<<"x = "<<x;**

**return 0;**

**}**

**A) 4**

**B) 6**

**C) 2**

**D) none of these**

**==========================================================**

**Q. 39. When should you use delete[] instead of delete in C++?**

A) When deallocating a dynamically allocated single object.

B) When deallocating a dynamically allocated array.

C) When deallocating memory allocated using malloc().

D) When deleting an object that is referenced by a pointer in the stack.

**==========================================================**

**Q. 40.** What is the primary disadvantage of using inline functions in C++?

a) Inline functions always execute slower than regular functions.  
b) Inline functions increase the binary size because of repeated code.  
c) Inline functions are difficult to debug.  
d) Inline functions cannot be used with templates.

**==========================================================**