1. What is **operator overloading** in C++?

a) Creating new operators
b) Defining multiple operators with the same name
c) Overloading existing operators with custom behavior
d) Using operators for conditional statements

2. Which of the following **operators cannot be overloaded**?

+ :: [ ] ? :

3. What is the purpose of the **friend keyword** in operator overloading?

a) To declare a function that can access private members of a class
b) To indicate a function that overloads the assignment operator
c) To specify a friend class for operator overloading
d) To define a global function for operator overloading

4. Which operator is used for overloading the **stream insertion** operator in C++?

< >> << ~

5. What is the correct syntax for **overloading the unary minus (-) operator** in a class?

void operator-() operator-() const void operator-() const operator-()

6. In C++, which operator overloading is done by a member function?

Binary Unary Ternary Assignment

7. When overloading the subscript operator [], what parameter type does the overloaded function take?

Int double char Any data type

8. What is the role of the **const keyword** in an operator overloading function?

a) It indicates that the function is constant
b) It specifies that the object is constant within the function
c) It signals that the function does not modify the object's state
d) It allows overloading operators for const and non-const objects

9. Which of the following operators **cannot be overloaded** using a member function?

= . ?: ->

10. What is the return type of the overloaded [] operator for array subscripting?

a) Int void Any data type The type of the array elements

11. In C++, which operator cannot be overloaded as a global function?

a) == = -> [ ]

12. What is the purpose of the operator new and operator delete functions in C++ operator overloading?

a) Overloading assignment operator
b) Overloading memory allocation and deallocation operators
c) Overloading stream insertion and extraction operators
d) Overloading arithmetic operators

13. When overloading the equality (==) operator, which of the following is the correct signature for a member function?

a) bool operator==(const MyClass& other)
b) bool operator==(MyClass& other)
c) MyClass operator==(const MyClass& other)
d) void operator==(const MyClass& other)

14. Which of the following is an example of a unary operator?

a) + += * /

15. What is the correct syntax for **overloading the addition (+) operator** as a global function?

a) MyClass operator+(const MyClass& other)
b) MyClass operator+(MyClass& other)
c) void operator+(const MyClass& other)
d) operator+(const MyClass& other)

16. In C++, what is the purpose of overloading the stream extraction (>>) operator for a user-defined type?

a) To perform bitwise extraction

b) To read data from a stream into an object of the user-defined type
c) To concatenate two strings
d) To display the object's content to the console

17. Which of the following is a correct way to overload the post-increment (++)operator for a user-defined class named MyClass?

a) MyClass operator++(int)
b) void operator++(int)
c) MyClass& operator++(int)
d) int operator++(MyClass&)

18. When overloading the assignment (=) operator as a member function, what is thereturn type?

a) Void MyClass& MyClass bool

19. In C++, what does the **const member function qualifier** imply in the context ofoperator overloading?

a) The operator cannot be overloaded
b) The object cannot be modified within the operator overloading function
c) The operator can only be overloaded as a global function
d) The operator can only be overloaded for constant objects

20. What is the purpose of the **operator -> overloading** in C++?

a) To define a member function for class pointer dereferencing
b) To create a custom arrow operator
c) To allow using an object like a pointer in a class
d) To overload the conditional operator

21. When overloading the stream insertion (<<) operator as a global function, whatparameter type is typically used for the right-hand side operand?

Int ostream istream MyClass

22. In C++, what is the purpose of **overloading the subscript ([]) operator** for auser-defined class?

a) To define array multiplication
b) To allow accessing class elements using array notation
c) To perform array division
d) To concatenate two arrays

23. Which of the following **operators cannot be overloaded** for a user-defined class?

a) &&(logical AND) =(assignment) .(member access) #(preprocessor)

24. When overloading the less than (<) operator as a member function, what is acommon return type?

a) Int void bool MyClass

25. In C++, what is the purpose of overloading the unary plus (+) operator for auser-defined class?

a) To perform unary addition
b) To allow using the class with arithmetic expressions

c) To define custom behavior when using the unary plus
d) To concatenate two objects

26. When overloading the equality (==) operator as a global function, which of the following is the correct parameter type for the left-hand side operand?

Int MyClass const MyClass& bool

27. What is the purpose of overloading the post-decrement (--) operator for a user-defined class?

a) To decrease the object's value by 1
b) To define custom behavior after decrementing the object
c) To increment the object's value by 1
d) To allow using the object in a loop

28. When overloading the function call () operator, what is the primary purpose?

a) To define custom behavior when the object is called like a function
b) To perform function composition
c) To invoke a member function of the object
d) To enable the object to be used in a loop

29. In C++, what is the **role of the friend keyword** in a function declaration for operator overloading?

a) To indicate a function that overloads the assignment operator
b) To declare a function that can access private members of a class
c) To specify a friend class for operator overloading
d) To allow overloading operators for constant and non-constant objects

30. When overloading the bitwise NOT (~) operator, what is the common return type?

Bool int char void

Coding type MCQs:

Question 1:

```
#include <iostream>
Using namespace std ;
class MyClass
{
private:
 int value;

public:
 MyClass(int v) : value(v) {}

 // Missing operator overloading code
};

int main() {
 MyClass
 obj1(5);
 MyClass
 obj2(3);

 MyClass result = obj1 /* ??? */ obj2;

 cout << "Result: " << result.getValue() << endl;

 return 0;
}
```

What should replace /* ??? */to overload the subtraction operator?

+ - * /

Question 2:

```
#include <iostream>

class Vector {
private:
  double x, y;

public:
  Vector(double x_val, double y_val) : x(x_val), y(y_val) {}

  // Missing operator overloading code
};

int main() {
  Vector v1(1.0, 2.0);
  Vector v2(3.0, 4.0);

  Vector result = v1 /* ??? */ v2;

  std::cout << "Result: (" << result.getX() << ", " << result.getY() << ")" <<
std::endl;

  return 0;
}
```

What should replace /* ??? */to overload the addition operator?

+ - * /

Question 3:

```
#include <iostream>
Using namespace std;

class Counter {
private:
  int count;

public:
  Counter(int initial) : count(initial) {}

  // Missing operator overloading code
};
```

```
int main() {
```

```
 Counter
 c1(5);Counter
 c2(3);

 Counter result = c1 /* ??? */ c2;

 cout << "Result: " << result.getCount() << endl;

 return 0;
}
```

What should replace /* ??? */to overload the greater than (>) operator?

> < == !=

Question 4:

```
#include <iostream>

class Fraction {
private:
 int numerator, denominator;

public:
 Fraction(int num, int den) : numerator(num), denominator(den) {}

 // Missing operator overloading code
};

int main() {
 Fraction frac1(1, 2);
 Fraction frac2(2, 3);

 Fraction result = frac1 /* ??? */ frac2;

 std::cout << "Result: " << result.getNumerator() << "/" <<
result.getDenominator() << std::endl;

 return 0;
}
```

What should replace /* ??? */to overload the multiplication (*) operator?

a) +
b) -
c) *
d) /

Answer: c) *

Question 5:

```
#include <iostream>

class Point {
private:
  int x, y;

public:
  Point(int x_val, int y_val) : x(x_val), y(y_val) {}

  // Missing operator overloading code
};

int main() {
  Point p1(1, 2);
  Point p2(3, 4);

  Point result = p1 /* ??? */ p2;

  std::cout << "Result: (" << result.getX() << ", " << result.getY() << ")" <<
std::endl;

  return 0;
}
```

What should replace /* ??? */to overload the equality (==) operator?

a) ==
b) !=
c) <
d) >

Answer: a) ==

Question:

```
#include <iostream>

class Complex
{private:
 double real, imaginary;

public:
 Complex(double r, double i) : real(r), imaginary(i) {}

 // Missing operator overloading code

 void display() const {
 std::cout << real << " + " << imaginary << "i";
 }
};

int main() {
 Complex c1(2.5, 3.0);
 Complex c2(1.5, 2.0);

 Complex result = c1 /* ??? */ c2;

 std::cout << "Result: ";
 result.display();

 return 0;
}
```

What should replace /* ??? */to overload the addition (+) operator using a friendfunction?

a) +
b) -
c) *
d) /

Answer: a) +

Explanation: When overloading the binary addition operator as a friend function, youneed to declare the friend function in the class and define it outside the class. The correct syntax is:

```
friend Complex operator+(const Complex& lhs, const Complex& rhs);
```

You would then define the function outside the class:

```
Complex operator+(const Complex& lhs, const Complex& rhs) {
  return Complex(lhs.real + rhs.real, lhs.imaginary + rhs.imaginary);
}
```

This allows the addition operator to be used with Complex objects as shown in the main function.

Que: What is the primary purpose of a constructor in C++?

To create objects

To allocate memory

To initialize object properties

To perform object destruction

Que: When is a default constructor called in C++?

When an object is declared

When an object is created using the new keyword

When an object is passed as a function argument

When an object goes out of scope

Que: What is the purpose of a destructor in C++?

To create objects

To initialize object properties

To deallocate memory and perform cleanup

To copy object properties

Que: In C++, can a class have multiple constructors?

No, a class can have only one constructor

Yes, but only if they have different return types

Yes, through constructor overloading

Yes, but only if the constructors have different access specifiers

Que: What is constructor overloading in C++?

Defining multiple constructors in a class

Overriding a constructor in a derived class

Creating a constructor with a single argument

Using the override keyword in a constructor

Que: **Which of the following is a valid example of constructor overloading?**

Circle(int radius); Circle(float diameter);

Circle(); All of the above

Que: What is the purpose of a copy constructor in C++?

To create a duplicate object To copy the contents of one object to another

To initialize object properties To perform object destruction

Que: How is a copy constructor typically defined in C++?

CopyConstructor(int val) { /* constructor code */ }

void CopyConstructor(const CopyConstructor& obj) { /*constructor code */ }

CopyConstructor(const CopyConstructor& obj) { /* constructorcode */ }

CopyConstructor(CopyConstructor obj) { /* constructor code */ }

Que: What is a dynamic constructor in C++?

A constructor that allocates memory using malloc

A constructor that is dynamically created at runtime

A constructor that initializes dynamic variables

A constructor with a variable number of arguments

Que: How are static members of a class typically initialized in C++?

Inside the constructor Automatically initialized by the compiler

Using the static keyword in the class declaration Inside the destructor

Que: Which of the following statements about constructor overloading is correct?

Constructors cannot be overloaded

Overloaded constructors must have the same return type

Overloaded constructors must have the same name

Overloaded constructors must have different parameter lists

Consider the following C++ code:

```
class Rectangle {
public:
  Rectangle (int length, int width);
  Rectangle (int side);
};
```

Que: What concept is demonstrated in the code?

Constructor overloading Constructor overriding

Copy constructor Dynamic constructor

Que: When is the copy constructor called in C++?

When a new object is declared When an object is deleted

When an object is passed by reference to a function

When an object is assigned the value of another object

Consider the following C++ code:

```
class MyClass {
public:
  MyClass(const MyClass& obj);
};
```

Que: What is the **purpose of the constructor** in the code?

Dynamic memory allocation Copy constructor

Constructor overloading Default constructor

Que: What is a dynamic constructor in C++?

A constructor that allocates memory dynamically using new

A constructor that has dynamic parameters A constructor that is defined at runtime

A constructor with variable-length parameter lists

Consider the following C++ code:

```
class DynamicExample{
public:
  DynamicExample (int size);
  ~DynamicExample ();
};
```

Que: What is the likely purpose of this class?

To demonstrate constructor overloading To manage dynamic memory allocation

To illustrate copy constructors To showcase static members

Que: What is a static member variable in C++?

A variable that can be accessed only within the same function

A variable that is automatically initialized by the compiler

A variable that can be accessed only within the same class

A variable that is shared among all instances of the class

Consider the following C++ code:

```
class Example {
public:
  static int count;
  Example();
  ~Example();
};
int Example::count = 0;
```

Que: What does the static member variable count represent?

The total number of instances of the class

The number of instances currently in scope

The count of dynamic memory allocations The count of static members

Que: How is a static member function typically defined in C++?

static void myFunction(); void static myFunction();

void myFunction() static; void myFunction();

Que: What is the primary benefit of using static members in a class?

They allow for dynamic memory allocation

They can be accessed without creating an instance of the class

They automatically initialize themselves They are automatically deallocated

Que: Which of the following is a valid example of constructor overloading?

Rectangle(); Rectangle(int length);

Rectangle(int width); All of the above

Consider the following C++ code:

```
class Point {
 public:
   Point();
   Point(int x, int y);
 };
```

Que: What concept is demonstrated in the code?

Constructor overloading Constructor overriding

Copy constructor Dynamic constructor

Que: What is the correct **signature for a copy constructor** in C++?

CopyConstructor();

void CopyConstructor(CopyConstructor obj);

CopyConstructor(CopyConstructor& obj);

CopyConstructor(const CopyConstructor& obj);

Consider the following C++ code:

```
class Employee {
public:
  Employee(const Employee& emp);
};
```

Que: What is the **purpose of the constructor** in the code?

To create a new employee To copy the contents of one employee to another

To initialize employee properties To perform employee termination

Que: What is the primary difference between a dynamic constructor and a regular constructor in C++?

Dynamic constructors allocate memory using malloc

Dynamic constructors have variable-length parameter lists

Dynamic constructors can be called explicitly at runtime

Dynamic constructors allocate memory using new

Consider the following C++ code:

```
class DynamicArray {

public:
  DynamicArray(int size);
  ~DynamicArray();
};
```

Que: What is the **likely purpose** of this class?

To demonstrate constructor overloading To manage a dynamically allocated array

To illustrate copy constructors To showcase static members

Consider the following C++ code:

```
class Counter {

public:
  static int count;
  Counter();
  ~Counter();
};
int Counter::count = 0;
```

Que: What does the static member variable count represent?

The total number of instances of the class

The number of instances currently in scope

The count of dynamic memory allocations The count of static members

Que: What is the purpose of a static member function in C++?

To create dynamic instances of a class

To access dynamic memory allocation functions

To perform operations that do not depend on a specific instance

To call other member functions dynamically.

Que: In C++, how is a static member variable typically accessed?

object.staticMember Class::staticMember

staticMember(Class) object->staticMember

Que: What happens to a static member variable when the last instance of the class isdestroyed in C++?

It is automatically deallocated It retains its value until the program ends

It is automatically set to zero It becomes inaccessible

Que: Which of the following statements about constructor overloading is correct?

Overloaded constructors must have the same number of parameters

Overloaded constructors can have different return types

Overloaded constructors must have the same access specifiers

Overloaded constructors must have the same names

Consider the following C++ code:

```
class Car {
public:
  Car();
  Car(int year, const string& model);
};
```

Que: What concept is demonstrated in the code?

Constructor overloading Constructor overriding

Copy constructor Dynamic constructor

Que: What is the primary role of a copy constructor in C++?

Initializing an object with values from another object

Allocating memory for an object Deallocating memory for an object

Initializing an object with default values.

Consider the following C++ code:

```
class Book {
public:
  Book(const Book& other);
};
```

Que: What is the **purpose of the constructor** in the code?

To create a new book

To destroy a book

To copy the contents of one book to another

To allocate memory for a book

Que: What is the primary advantage of using dynamic constructors in C++?

They allow for automatic memory deallocation

They allow for dynamic memory allocation and deallocation

They can be called explicitly at runtime

They are automatically called when an object goes out of scope

Que:

```
class DynamicObject {
public:
  DynamicObject(int size);
  ~DynamicObject();
};
```

Que: What is the **likely purpose** of this class?

To demonstrate constructor overloading

To manage dynamic memory allocation

To illustrate copy constructors

To showcase static members

Constructors and Destructors with Static Members:

Que: What is a static member variable in C++?

A variable that can only be accessed within the same function

A variable that is automatically initialized by the compiler

A variable that can be accessed only within the same class

A variable that is shared among all instances of the class

Consider the following C++ code:

```
class Counter {
public:
  static int count;
  Counter();
  ~Counter();
};
int Counter::count = 0;
```

Que: What is a primary use case for a static member function in C++?

To modify the values of non-static members

To perform operations that do not depend on a specific instance

To access dynamic memory allocation functions

To create dynamic instances of a class

Que: How is a static member variable typically declared in a C++ class?

int static myVar; myVar(int) static;

static int myVar; static myVar int;

Que: What is inheritance in C++?

A way to achieve encapsulation

A way to create a new class using the properties of an existing class

A process of converting a class to an interface.

A way to hide the implementation details of a class.

Que: Which keyword is used to indicate inheritance in C++?

extend inherit derives : (colon)

Que: What is the purpose of the virtualkeyword in C++ when used with a function in a base class?

It indicates that the function is static It makes the function constant

It specifies that the function cannot be overridden

It signals that the function may be overridden in derived classes

Que: In C++, what is the syntax for inheriting publicly?

class Derived: public Base class Derived extends Base

class Derived - public Base class Derived(public) : Base

Que: In C++, what is the significance of the protectedaccess specifier in a derived class?

Members are accessible only within the same class

Members are accessible only within the same file

Members are accessible in derived classes and within the sameclass

Members are not accessible in any derived class.

Que: Which keyword is used to call the constructor of the base class from the derived class in C++?

base this super :: (scope resolution)

Que: What is the purpose of the override keyword in C++?

It indicates that a function is virtual It specifies that a function is static

It declares the intention to override a virtual function in a derived class

It is used to override the access specifier of a base class

Que: In C++, which type of inheritance allows a class to inherit from more than one base class?

Single Multiple Hierarchical Multilevel

Que: What is the **purpose of pure virtual functions** in C++?

They cannot be overridden in derived classes

They provide a default implementation in the base class

They force derived classes to provide their own implementation

They are static methods in a class.

Que: In C++, what is the **role of the virtual keyword when used with a destructor in the base class?**

It makes the destructor private

It allows the destructor to be overridden in derived classes

It prevents the destructor from being called

It is not allowed to be used with destructors.

Que: What does the term **"upcasting"** refer to in C++?

Converting a derived class object to a base class object

Converting a base class object to a derived class object

Converting a derived class to an abstract class

Converting a base class to a friend class.

Que: In C++, what is the purpose of the **dynamic_cast operator**?

To cast a variable to a different data type To allocate memory dynamically

To perform type checking during runtime for polymorphic classes

To convert a derived class object to a base class object

Que: What is the **significance of the protected keyword** in a derived class in C++?

It makes the members accessible only within the same class

It makes the members accessible in any class within the same file

It makes the members accessible only in derived classes

It makes the members accessible to any class.

Que: In C++, what is the purpose of the **friend keyword** in the context ofinheritance?

To indicate that a class is derived To define a class as a base class

To specify that a function is a friend of a derived class

To allow private members of a class to be accessible in a derivedclass

Consider the following C++ code:

```
class Shape {
protected:
  int width, height;
public:
  Shape(int w, int h) : width(w), height(h) {}virtual
  int area() { return 0; }
};

class Rectangle : public Shape {
public:
  Rectangle(int w, int h) : Shape(w, h) {}
  // What is missing in this class to correctly calculate the area?
  // a. int area() { return width * height; }
  // b. int area() override { return width * height; }
  // c. int calculateArea() { return width * height; }
  // d. int calculateArea() override { return width * height; }
};
```

Que: Which option correctly completes the Rectangle class to calculate the area?

a. int area() { return width * height; }
b. int area() override { return width * height; }
c. int calculateArea() { return width * height; }
d. int calculateArea() override { return width * height; }

Consider the following C++ code:

```
class Animal {
public:
  virtual void speak() { cout << "Animal speaks" << endl; }
};

class Dog : public Animal {
public:
  // What is the correct way to override the speak function in Dogclass?
  // a. void speak() { cout << "Dog barks" << endl; }
  // b. void speak() override { cout << "Dog barks" << endl; }
  // c. void speak() override final { cout << "Dog barks" << endl; }
```

```
  // d. final void speak() { cout << "Dog barks" << endl; }
};
```

Which option correctly overrides the speak function in the Dog class?

a. void speak() { cout << "Dog barks" << endl; }

b. void speak() override { cout << "Dog barks" << endl; }

c. void speak() override final { cout << "Dog barks" << endl; }

d. final void speak() { cout << "Dog barks" << endl; }

Consider the following C++ code:

```
class A {
public:
  virtual void print() const { cout << "A"; }
};

class B : public A {
public:
  void print() const override { cout << "B"; }
};

void display(const A& obj) {
  obj.print();
}
int main() {B
  bObj;
  display(bObj);
  return 0;
}
```

What will be the output of the main function?

a. A B AB Compiler error

Consider the following C++ code:

```
class Base {
public:
  virtual void display() const { cout << "Base"; }
};

class Derived : public Base
{
```

```
public:
  void display() const override
 {
  cout << "Derived";
  }
};

int main( )
{
  Base* ptr = new Derived;
  ptr->display();
  delete ptr;
  return 0;
}
```

What will be the output of the main function?

a. Base Derived Compiler error Undefined behavior

Consider the following C++ code:

```
class Vehicle {
public:
  virtual void start() { cout << "Vehicle started"; }
};

class Car : public Vehicle {
public:
  // What is the correct way to override the start function in Car class?
  // a. void start() { cout << "Car started"; }
  // b. void start() const override { cout << "Car started"; }
  // c. const void start() override { cout << "Car started"; }
  // d. void start() override final { cout << "Car started"; }
};
```

Que: Which option correctly overrides the start function in the Car class?

void start() { cout << "Car started"; }

void start() const override { cout << "Car started"; }

const void start() override { cout << "Car started"; }

```
void start() override final { cout << "Car started"; }
```

Consider the following C++ code:

```
class Animal {
public:
  virtual void sound() const { cout << "Animal sound" << endl; }
};
class Cat : public Animal {
public:
  // How should the sound function be overridden in the Cat class?
  // a. void sound() override { cout << "Meow" << endl; }
  // b. void sound() final { cout << "Meow" << endl; }
  // c. final void sound() { cout << "Meow" << endl; }
  // d. void sound() { cout << "Meow" << endl; }
};
```

Que: Which option correctly **overrides the sound function in the Cat class**?

```
void sound() override { cout << "Meow" << endl; }
```

```
void sound() final { cout << "Meow" << endl; }
```

```
final void sound() { cout << "Meow" << endl; }
```

```
void sound() { cout << "Meow" << endl; }
```

Consider the following C++ code:

```
class Shape {
public:
  virtual void draw() const { cout << "Drawing shape" << endl; }
};

class Circle : public Shape {
public:
  // What is the correct way to override the draw function in the Circle class?
  // a. void draw() const override { cout << "Drawing circle" << endl; }
  // b. void draw() const { cout << "Drawing circle" << endl; }
  // c. const void draw() override { cout << "Drawing circle" << endl; }
  // d. void draw() override final { cout << "Drawing circle" << endl; }
};
```

Que: Which option correctly **overrides the draw function in the Circle class?**

void draw() const override { cout << "Drawing circle" << endl; }

void draw() const { cout << "Drawing circle" << endl; }

const void draw() override { cout << "Drawing circle" << endl; }

void draw() override final { cout << "Drawing circle" << endl; }

Consider the following C++ code:

```
class A {
public:
  virtual void print() const { cout << "A"; }
};

class B : public A {
public:
  void print() const override { cout << "B"; }
};

void display(const A& obj)
{
obj.print();
}

int main() {
B bObj;
  display(bObj);
  return 0;
}
```

Que: What will be the output of the main function?

A B AB Compiler error

Consider the following C++ code:

```
class Base {
public:
  virtual void display() const { cout << "Base"; }
};
```

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

int main()
{
  Base* ptr = new Derived;
  ptr->display();
  delete ptr;
  return 0;
}
```

Que: What will be the output of the main function?

Base Derived Compiler error Undefined behavior

Consider the following C++ code:

```
class Vehicle {
public:
  virtual void start() { cout << "Vehicle started"; }
};

class Car : public Vehicle {
public:
  // What is the correct way to override the start function in Car class?
  // a. void start() { cout << "Car started"; }
  // b. void start() const override { cout << "Car started"; }
  // c. const void start() override { cout << "Car started"; }
  // d. void start() override final { cout << "Car started"; }
};
```

Que: Which option correctly overrides the start function in the Car class?

void start() { cout << "Car started"; }

void start() const override { cout << "Car started"; }

const void start() override { cout << "Car started"; }

void start() override final { cout << "Car started"; }

Que: What is the purpose of a virtual base class in C++?

To enable multiple inheritance without creating ambiguous paths

To prevent inheritance To enforce encapsulation To make a class abstract

Que: In C++, how is a class designated as a virtual base class?

a. Using the virtual keyword before the class name
b. Using the base keyword before the class name
c. Using the vb keyword before the class name
d. There is no specific keyword; it is determined by the derived class.

Que: What is the significance of a virtual base class in terms of member variables?

Virtual base classes cannot have member variables

Each derived class has its own copy of member variables from the virtual base class

All derived classes share a single copy of member variables from the virtual base class

Member variables from the virtual base class are inherited as private in all derived classes.

Que: In C++, what problem does the use of a virtual base class help to solve in thecontext of multiple inheritance?

Diamond problem Encapsulation problem

Polymorphism problem Ambiguity problem

Consider the following C++ code

```
class A {
public:
  int data;
};
class B : virtual public A {}; class C :
virtual public A {}; class D : public
B, public C {};
```

Que: What will be the size of an object of class D?

4 bytes 8 bytes 12 bytes 16 bytes

Que: In C++, what happens if a virtual base class is not initialized in the memberinitializer list of a derived class constructor?

Compiler error Undefined behavior

Default values are assigned automatically

No impact, the compiler handles it implicitly.

Que: Which keyword is used to access a virtual base class member in a derived class inC++?

a. base virtual vb super

Que: What is the order of constructor invocation in a hierarchy involving virtual baseclasses in C++?

From the most derived class to the base class

From the base class to the most derived class

In random order The order is not defined

Que: In C++, what is the purpose of using a virtual destructor in a virtual base class?

To ensure proper destruction of derived class objects

To enable polymorphism To avoid object slicing To prevent memory leaks

Que: In a hierarchy involving virtual base classes, which type of inheritance relationshipis established among the classes?

Public Protected Private Virtual

Que: What is an abstract class in C++?

A class with no member variables

A class that cannot be instantiated and may have pure virtual functions

A class with only private members A class with static member functions

Que: In C++, how is a pure virtual function declared in an abstract class?

virtual void func() = 0; void virtual func() = 0;

pure virtual void func(); void func() pure virtual;

Que: Can an abstract class have a constructor in C++?

Yes, always Yes, but only default constructor

No, abstract classes cannot have constructors

No, abstract classes can only have a destructor

Que: In C++, can a concrete (non-abstract) class inherit from an abstract class withoutimplementing its pure virtual functions?

Yes, but only if the abstract class has a default constructor

Yes, but only if the concrete class is also marked as abstract

No, it will result in a compilation error No, it will result in a runtime error

Que: In C++, what is a virtual function?

A function that is always static

A function declared with the virtual keyword in the base class

A function that cannot be overridden in derived classes

A function that is always pure virtual

Que: What is the purpose of a virtual function in C++?

To enable polymorphism and late binding

To make a function constant

To make a function private To make a function static

Que: In C++, what happens if a virtual function is not overridden in a derived class?

It results in a compilation error It results in a runtime error

The base class function is called

It automatically becomes a pure virtual function in the derived class.

Que: Which C++ keyword is used to override a virtual function in a derived class?

override extend implement inherit.

Que: What is the concept of "late binding" in the context of virtual functions in C++?

Resolving function calls at compile-time Resolving function calls at link-time

Resolving function calls at runtime Resolving function calls at class definition

Que: In C++, can a virtual function be private in the base class?

Yes, always

Yes, but only if it is also declared as static

No, it must be at least protected

No, it must be public.

Que: What is the role of a constructor in a derived class in C++?

To initialize only the base class members

To initialize only the derived class members

To initialize both the base and derived class members

Constructors are not allowed in derived classes.

Que: In C++, how is a base class constructor called explicitly from a derived classconstructor?

Using the basekeyword

Using the superkeyword

Using the thiskeyword

Using the ::(scope resolution) operator

Que: What is the order of execution of constructors in a multiple inheritance scenario inC++?

Random order

From the most derived class to the base class

From the base class to the most derived class

Constructors are not executed in multiple inheritance

Que: In C++, what happens if a derived class does not provide a constructor?

The base class constructor is called automatically

It results in a compilation error

The program runs without any issue

The derived class inherits the base class constructor.

Que: Which keyword is used to indicate that a member function in a derived class isintended to override a virtual function in the base class?

virtual override inherit extend

Que: In C++, what happens if a virtual function in a derived class does not have the overridekeyword?

It results in a compilation error The program runs without any issue

The virtual function is treated as a regular function

It is automatically treated as an override

Consider the following C++ code:

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

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

class C : public B {
public:
  C() { cout << "C"; }
  ~C() { cout << "~C"; }
};

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

What will be the output of the main function?

ABCCBA CBAABC ACBCBA CABABC

Que: In C++, what happens if a derived class provides its own implementation of a non-virtual function from the base class?

It results in a compilation error

The derived class implementation is ignored

Both the base and derived class implementations are called

The base class implementation is overridden.

Que: In C++, what is the **purpose of a destructor in a base class in the context of inheritance?**

To prevent memory leaks To ensure proper destruction of derived class objects

To enable polymorphism Destructors are not allowed in base classes

Que: What is the **significance of the virtual keyword in the base class** destructor inC++?

It indicates that the destructor is static

It allows the destructor to be overridden in derived classes

It prevents the destructor from being called

It makes the destructor constant

Que: What is **Hybrid Inheritance** in C++?

Inheriting from multiple classes using a common base class

Inheriting from multiple classes with no common base class

Inheriting from a single class

Inheriting from classes with virtual functions only

Consider the following C++ code:

cpp

Copy code

```
class A {

public:

  virtual void display() { cout << "A"; }

};
```

class B : virtual public A {};class

C : public A {};

class D : public B, public C {};

Que: What is the type of inheritance represented by class D?

Hybrid inheritance Hierarchical inheritance

Multiple inheritance Multipath inheritance

Que: In Hybrid Inheritance, what issue does the virtual keyword in the base class solve?

Ambiguity in multiple inheritance Ambiguity in multipath inheritance

Ambiguity in hierarchical inheritance Ambiguity in constructor calls

Que: What is **Hierarchical Inheritance** in C++?

Inheriting from multiple classes using a common base class

Inheriting from a single class with multiple derived classes

Inheriting from multiple classes with no common base class

Inheriting from abstract classes only

Consider the following C++ code:

```
class Shape {

public:

  virtual void draw() { cout << "Drawing shape"; }

};

class Circle : public Shape {}; class

Rectangle : public Shape {};class

Triangle : public Shape {};
```

Que: What type of inheritance is represented by the classes Circle, Rectangle, and Triangle?

Hybrid inheritance

Hierarchical inheritance

Multiple inheritance

Single inheritance

Que: What is the Diamond Problem in C++?

The challenge of defining multiple constructors in a class

The ambiguity that arises in multiple inheritance with a common base class

The difficulty in using the virtual keyword

The limitation of using polymorphism.

Consider the following C++ code:

```
class A {

public:

void display() { cout << "A"; }

 };

 class B : public A {};

 class C : public A {};

 class D : public B, public C {};
```

Que: What is the issue with the class D in terms of inheritance?

Multiple inheritance Multipath inheritance

Diamond problem Hybrid inheritance

Que: How can the Diamond Problem in C++ be resolved?

By avoiding multiple inheritance By using the virtual keyword in the base class

By making all functions static By using the final keyword in the derived classes

Consider the following C++ code:

```
class A {

public:

  virtual void display() { cout << "A"; }

};

class B : virtual public A {}; class C :

virtual public A {}; class D : public

B, public C {};
```

Que: **What type of inheritance is represented by the classes B, C, and D?**

Multiple inheritance Multipath inheritance

Diamond problem Hybrid inheritance

Que: In C++, how does the virtualkeyword address the ambiguity in multipleinheritance?

It prevents multiple inheritance It ensures that the base class is not inherited

It allows the compiler to identify the correct base class in the hierarchy

It enforces a specific order in which classes are inherited.

Question 1:

You are developing a shopping cart application in C++ for an online store. Each item in the shopping cart is represented by the Cart Item class. You need to implement operator overloading to facilitate the calculation of the total cost of items in the shopping cart.

Test Case 1:

Input:

Item 1: Name - "Laptop", Price - ₹1200.0,

Quantity - 2 Item 2: Name - "Smartphone", Price -

₹600.0, Quantity - 3Expected Output:

Total Cost: ₹4200.0

Test Case 2:

Input:

Item 1: Name - "Headphones", Price - ₹80.0,

Quantity - 1 Item 2: Name - "Smartwatch", Price -

₹150.0, Quantity - 2

Expected

Output: Total

Cost: ₹380.0

Question 2:

You are a secret agent working on a mission to build a hidden base. Design a class SecretBase with attributes for the base location and security level. Implement a default constructor that initializes the location to "Unknown" and security level to 0. Additionally, include a destructor to print a message when the base is destroyed.

Test Case1:

Input: N/A

Output: A secret base with an unknown location, security level 0, and a message when the base is destroyed.

Test Case 2: Creating a Custom Secret Base and Destroying it

Input: Custom location: "Mount Everest"
Custom security level: 5

Expected Output:

A SecretBase object is created with the location set to "Mount Everest" and the security level set to 5.
A message is displayed when the SecretBase object is destroyed, indicating thedestruction of the secret base

Question 3.

Engineers are building an advanced spaceship for interstellar travel. Create a class Spaceshipwith a dynamic array representing modules. Implement a constructor to initialize the spaceship with a specified number of modules, and a destructor to release the dynamically allocated memory when the spaceship is decommissioned.

Test Case:

Input: Number of Modules = 10

Output: A spaceship with 10 modules, and memory released when the spaceship is decommissioned.

Question 4)

In a wizardry school, students are learning to use magical wands. Design a class MagicWand with attributes like the wand's core material and length.

Overload the + operator to combine two wands, creating a more powerful wand with a longer length.

Test Case:

Input: Wand1 - Core Material = "Phoenix Feather", Length = 10

inchesWand2 - Core Material = "Dragon Heartstring", Length = 8

inches

Output: A combined wand with core material "Phoenix Feather" and length 18 inches.

Question-5)

A scientist is developing a time travel machine. Create a class TimeMachine with attributes for the year and month. Overload the ++ operator to advance the time by one month and the -- operator to go back in time by one month.

Test Case:

Input: Year = 2023, Month = May

Output: Time advanced to June 2023 using ++ operator and then back to May using -- operator.

Question-6)

In an adventurous treasure hunt game, players have a map with their current position. Design a class TreasureMap with coordinates. Overload the + operator to combine two maps, creating a larger map with combined coordinates.

Test Case:

Input: Map1 - Coordinates (x=30,

y=40)Map2 - Coordinates (x=20,

y=15)

Output: A combined map with coordinates (x=50, y=55).

Question-7:

In a zoo simulation, you are tasked with designing classes for different animals. Implement a base class Animal with attributes like name and age. Create derived classes for specific animals, such as Lion, Elephant, and Monkey, inheriting from the Animal class. Include specific behaviors for each animal.

Test Case:

Input: A Lion named "Simba" with age 5.

Output: Displaying information about Lion "Simba" including age and specific behaviors

Question-8

In a company management system, you need to represent different types of employees. Implement a base class Employee with attributes like name and salary. Create derived classes for specific employee types, such as Manager, Developer, and Intern, inheriting from the Employee class. Include methods for specific tasks related to each role.

Test Case:

Input: A Manager named "Alice" with a salary of $80,000.

Output: Displaying information about Manager "Alice," including the salary and specific managerial tasks.

Question-9

In an e-commerce application, create a class hierarchy to represent different types of products. Implement a base class Product with attributes like name and price. Create derived classes forspecific product types, such as Electronics, Clothing, and Books, inheriting from the Product class. Include methods for specific product details.

Test Case:

Input: A Book named "The Great Gatsby" with a price of $15.

Output: Displaying information about the book, including the name and price.

Question-10:

Continuing with the geometric application, you have the Shape class with a virtual function calculateArea(). Create derived classes for specific shapes, such as Circle, Rectangle, and Triangle. Override the calculateArea() function in each derived class to provide shape-specific area calculations.

Test Case:

Input: A Triangle with color "Blue" and sides of length 6, 8, and 10.

Output: Displaying information about the Blue Triangle, including its calculated area using the overridden function.

Question-11:

Continuing with the geometric application, you have the Shape class with a virtual function calculateArea(). Create derived classes for specific shapes, such as Circle, Rectangle, and Triangle. Override the calculateArea() function in each derived class to provide shape-specific area calculations.

Test Case:

Input: A Triangle with color "Blue" and sides of length 6, 8, and 10.

Output: Displaying information about the Blue Triangle, including its calculated area using the overridden function.

Question-12:

Build a system for managing university departments. Design an abstract class Department with a pure virtual function displayInfo(). Derive classes for specific departments like MathDepartment, ComputerScienceDepartment, and EnglishDepartment. Implement the displayInfo() function in each derived class to provide department-specific details.

Test Case:

Input: A Computer Science department named "CS Department" with code CS101.

Output: Displaying details about the Computer Science department "CS Department" with code CS101 using the pure virtual function.

Question-13:

Implement a system for a musical instrument shop. Create an abstract class Instrument with apure virtual function play(). Derive classes for specific instruments like Guitar, Piano, and Flute from the Instrument class. Implement the play() function in each derived class to provide instrument-specific melodies.

Test Case:

Input: A Piano named "Grand Concerto."

Output: Displaying information about Piano "Grand Concerto" and playing a melody using the pure virtual function.

Test Case:

Input:

A Flute named "Harmony Whisper."

Output:

Instrument Information:

Name: Harmony

WhisperType: Flute

Playing a Melody: [Melodic sound produced by the Flute]

Question-14:

In an online shopping application, customers can add items to their shopping cart. Implement aclass ShoppingCart with a method addItem that adds items to the cart. If the customer attemptsto add an item with a negative quantity or an invalid product code, throw appropriate exceptions (NegativeQuantityException or InvalidProductCodeException). Handle these exceptions and display error messages.

Test Case: Adding an Item with Negative

QuantityInput: Product code = "P001",

Quantity = -2

Expected Output: "Error: Cannot add item. Negative quantity not

allowed."Test Case 2: Adding an Item with Invalid Product Code

Input:

Product Code:

"" Quantity: 3

Expected

Output:

"Error: Invalid product code. Please enter a valid product code.