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

Answer: c) Overloading existing operators with custom behavior

2. Which of the following operators cannot be overloaded?

a) `+`
b) `::`
c) `[]`
d) `? :`

Answer: b) `::`

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

Answer: a) To declare a function that can access private members of a class

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

a) <
b) >>
c) <<
d) ~

Answer: c) <<

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

a) `void operator-()`
b) `operator-() const`
c) `void operator-() const`
d) `operator-()`

Answer: c) `void operator-() const`

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

a) Binary operators
b) Unary operators
c) Ternary operators
d) Assignment operators

Answer: a) Binary operators

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

a) int
b) double
c) char
d) Any data type

Answer: a) int

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 constant and non-constant objects

Answer: c) It signals that the function does not modify the object's state

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

a) `=`
b) `.`
c) `?:`
d) `->`

Answer: d) `->`

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

a) int
b) void
c) Any data type
d) The type of the array elements

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

a) `==`
b) `=`
c) `->`
d) `[]`

Answer: c) `->`

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

Answer: b) Overloading memory allocation and deallocation 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)`

Answer: a) `bool operator==(const MyClass& other)`

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

a) `+`
b) `+=`
c) `*`
d) `/`

Answer: 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)`

Answer: a) `MyClass 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

Answer: b) To read data from a stream into an object of the user-defined type

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&)`

Answer: c) `MyClass& operator++(int)`

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

a) `void`
b) `MyClass&`
c) `MyClass`
d) `bool`

Answer: b) `MyClass&`

19. In C++, what does the `const` member function qualifier imply in the context of operator 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

Answer: b) The object cannot be modified within the operator overloading function

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

Answer: c) To allow using an object like a pointer in a class

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

a) `int`
b) `ostream`
c) `istream`
d) `MyClass`

Answer: b) `ostream`

22. In C++, what is the purpose of overloading the subscript (`[]`) operator for a user-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

Answer: b) To allow accessing class elements using array notation

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

a) `&&` (logical AND)
b) `=` (assignment)
c) `.` (member access)
d) `#` (preprocessor)

Answer: c) `.` (member access)

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

a) `int`
b) `void`
c) `bool`
d) `MyClass`

Answer: c) `bool`

25. In C++, what is the purpose of overloading the unary plus (+) operator for a user-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

Answer: c) To define custom behavior when using the unary plus

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?

a) `int`
b) `MyClass`
c) `const MyClass&`
d) `bool`

Answer: c) `const MyClass&`

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

Answer: b) To define custom behavior after decrementing the object

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

Answer: a) To define custom behavior when the object is called like a function

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

Answer: b) To declare a function that can access private members of a class

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

a) `bool`
b) `int`
c) `char`
d) `void`

<u>Coding type MCQs:</u>

Question 1:

```
#include <iostream>

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;

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

 return 0;
}
```

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

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

Answer: b) `-`

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?

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

Answer: a) `+`

Question 3:

```
#include <iostream>

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;

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

 return 0;
}
```

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

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

Answer: a) `>`

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 friend function?

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

Answer: a) `+`

Explanation: When overloading the binary addition operator as a friend function, you need 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.

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

a. To create objects
b. To allocate memory
c. To initialize object properties
d. To perform object destruction

Answer: c. To initialize object properties

When is a default constructor called in C++?

a. When an object is declared
b. When an object is created using the `new` keyword
c. When an object is passed as a function argument
d. When an object goes out of scope

Answer: a. When an object is declared

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

a. To create objects
b. To initialize object properties
c. To deallocate memory and perform cleanup
d. To copy object properties

Answer: c. To deallocate memory and perform cleanup

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

a. No, a class can have only one constructor
b. Yes, but only if they have different return types
c. Yes, through constructor overloading
d. Yes, but only if the constructors have different access specifiers

Answer: c. Yes, through constructor overloading

What is constructor overloading in C++?

a. Defining multiple constructors in a class
b. Overriding a constructor in a derived class
c. Creating a constructor with a single argument
d. Using the `override` keyword in a constructor

Answer: a. Defining multiple constructors in a class

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

a. `Circle(int radius);`
b. `Circle(float diameter);`
c. `Circle();`
d. All of the above

Answer: d. All of the above

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

a. To create a duplicate object
b. To copy the contents of one object to another
c. To initialize object properties
d. To perform object destruction

Answer: b. To copy the contents of one object to another

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

a. **`CopyConstructor(int val) { /* constructor code */ }`**
b. **`void CopyConstructor(const CopyConstructor& obj) { /* constructor code */ }`**
c. **`CopyConstructor(const CopyConstructor& obj) { /* constructor code */ }`**
d. **`CopyConstructor(CopyConstructor obj) { /* constructor code */ }`**

Answer: c. **`CopyConstructor(const CopyConstructor& obj) { /* constructor code */ }`**

What is a dynamic constructor in C++?

a. A constructor that allocates memory using `malloc`
b. A constructor that is dynamically created at runtime
c. A constructor that initializes dynamic variables
d. A constructor with a variable number of arguments

Answer: c. A constructor that initializes dynamic variables

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

a. Inside the constructor
b. Automatically initialized by the compiler
c. Using the `static` keyword in the class declaration
d. Inside the destructor

Answer: c. Using the `static` keyword in the class declaration

Which of the following statements about constructor overloading is correct?

a. Constructors cannot be overloaded
b. Overloaded constructors must have the same return type
c. Overloaded constructors must have the same name
d. Overloaded constructors must have different parameter lists
Answer: d. Overloaded constructors must have different parameter lists

Consider the following C++ code:

```
class Rectangle {


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

What concept is demonstrated in the code?

a. Constructor overloading
b. Constructor overriding
c. Copy constructor
d. Dynamic constructor
Answer: a. Constructor overloading

When is the copy constructor called in C++?

a. When a new object is declared
b. When an object is passed by reference to a function
c. When an object is assigned the value of another object
d. When an object is deleted
Answer: c. When an object is assigned the value of another object
Consider the following C++ code:

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

What is the purpose of the constructor in the code?

a. Dynamic memory allocation
b. Copy constructor
c. Constructor overloading
d. Default constructor
Answer: b. Copy constructor

What is a dynamic constructor in C++?

a. A constructor that allocates memory dynamically using `new`
b. A constructor that has dynamic parameters
c. A constructor that is defined at runtime
d. A constructor with variable-length parameter lists
Answer: a. A constructor that allocates memory dynamically using `new`
Consider the following C++ code:

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

What is the likely purpose of this class?

a. To demonstrate constructor overloading
b. To manage dynamic memory allocation
c. To illustrate copy constructors
d. To showcase static members
Answer: b. To manage dynamic memory allocation

What is a static member variable in C++?

a. A variable that can be accessed only within the same function
b. A variable that is automatically initialized by the compiler
c. A variable that can be accessed only within the same class
d. A variable that is shared among all instances of the class
Answer: d. 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;
```

What does the static member variable `count` represent?

a. The total number of instances of the class
b. The number of instances currently in scope
c. The count of dynamic memory allocations
d. The count of static members

Answer: a. The total number of instances of the class

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

a. `static void myFunction();`
b. `void static myFunction();`
c. `void myFunction() static;`
d. `void myFunction();`

Answer: a. `static void myFunction();`

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

a. They allow for dynamic memory allocation
b. They can be accessed without creating an instance of the class
c. They automatically initialize themselves
d. They are automatically deallocated

Answer: b. They can be accessed without creating an instance of the class

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

a. `Rectangle();`
b. `Rectangle(int length);`
c. `Rectangle(int width);`
d. All of the above
Answer: d. All of the above

Consider the following C++ code:

```
class Point {


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

What concept is demonstrated in the code?

a. Constructor overloading
b. Constructor overriding
c. Copy constructor
d. Dynamic constructor
Answer: a. Constructor overloading

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

a. `CopyConstructor();`
b. `void CopyConstructor(CopyConstructor obj);`
c. `CopyConstructor(CopyConstructor& obj);`
d. `CopyConstructor(const CopyConstructor& obj);`
Answer: d. `CopyConstructor(const CopyConstructor& obj);`

Consider the following C++ code:

```
class Employee {

public:

 Employee(const Employee& emp);
};
```

What is the purpose of the constructor in the code?

a. To create a new employee
b. To copy the contents of one employee to another
c. To initialize employee properties
d. To perform employee termination

Answer: b. To copy the contents of one employee to another

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

a. Dynamic constructors allocate memory using `malloc`
b. Dynamic constructors have variable-length parameter lists
c. Dynamic constructors can be called explicitly at runtime
d. Dynamic constructors allocate memory using `new`
Answer: d. Dynamic constructors allocate memory using `new`

Consider the following C++ code:

```
class DynamicArray {


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

What is the likely purpose of this class?

a. To demonstrate constructor overloading
b. To manage a dynamically allocated array
c. To illustrate copy constructors
d. To showcase static members
Answer: b. To manage a dynamically allocated array

Consider the following C++ code:

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

What does the static member variable `count` represent?

a. The total number of instances of the class
b. The number of instances currently in scope
c. The count of dynamic memory allocations
d. The count of static members
Answer: a. The total number of instances of the class

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

a. To create dynamic instances of a class
b. To access dynamic memory allocation functions
c. To perform operations that do not depend on a specific instance
d. To call other member functions dynamically
Answer: c. To perform operations that do not depend on a specific instance

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

a. `object.staticMember`
b. `Class::staticMember`
c. `staticMember(Class)`
d. `object->staticMember`
Answer: b. `Class::staticMember`

What happens to a static member variable when the last instance of the class is destroyed in C++?

a. It is automatically deallocated
b. It retains its value until the program ends
c. It is automatically set to zero
d. It becomes inaccessible
Answer: b. It retains its value until the program ends

Which of the following statements about constructor overloading is correct?

a. Overloaded constructors must have the same number of parameters
b. Overloaded constructors can have different return types
c. Overloaded constructors must have the same access specifiers
d. Overloaded constructors must have the same names
Answer: b. Overloaded constructors can have different return types

Consider the following C++ code:

```
class Car {


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

What concept is demonstrated in the code?

a. Constructor overloading
b. Constructor overriding
c. Copy constructor
d. Dynamic constructor
Answer: a. Constructor overloading

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

a. Initializing an object with values from another object
b. Allocating memory for an object
c. Deallocating memory for an object
d. Initializing an object with default values
Answer: a. Initializing an object with values from another object

Consider the following C++ code:

```
class Book {


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

What is the purpose of the constructor in the code?

a. To create a new book
b. To destroy a book
c. To copy the contents of one book to another
d. To allocate memory for a book
Answer: c. To copy the contents of one book to another

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

a. They allow for automatic memory deallocation
b. They allow for dynamic memory allocation and deallocation
c. They can be called explicitly at runtime
d. They are automatically called when an object goes out of scope
Answer: b. They allow for dynamic memory allocation and deallocation

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

What is the likely purpose of this class?

a. To demonstrate constructor overloading
b. To manage dynamic memory allocation
c. To illustrate copy constructors
d. To showcase static members
Answer: b. To manage dynamic memory allocation

Constructors and Destructors with Static Members:

What is a static member variable in C++?

a. A variable that can only be accessed within the same function
b. A variable that is automatically initialized by the compiler
c. A variable that can be accessed only within the same class
d. A variable that is shared among all instances of the class
Answer: d. 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;
```

What does the static member variable `count` represent?

a. The total number of instances of the class
b. The number of instances currently in scope
c. The count of dynamic memory allocations
d. The count of static members
Answer: a. The total number of instances of the class

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

a. To modify the values of non-static members
b. To perform operations that do not depend on a specific instance
c. To access dynamic memory allocation functions
d. To create dynamic instances of a class
Answer: b. To perform operations that do not depend on a specific instance

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

a. `int static myVar;`
b. `myVar(int) static;`
c. `static int myVar;`
d. `static myVar int;`
Answer: c. `static int myVar;`

What is inheritance in C++?

- a. A way to achieve encapsulation
- b. A way to create a new class using the properties of an existing class
- c. A process of converting a class to an interface
- d. A way to hide the implementation details of a class

Answer: b. A way to create a new class using the properties of an existing class

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

- a. extend
- b. inherit
- c. derives
- d. : (colon)

Answer: d. : (colon)

What is the purpose of the `virtual` keyword in C++ when used with a function in a base class?

- a. It indicates that the function is static
- b. It specifies that the function cannot be overridden
- c. It signals that the function may be overridden in derived classes
- d. It makes the function constant

Answer: c. It signals that the function may be overridden in derived classes

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

- a. class Derived: public Base
- b. class Derived extends Base
- c. class Derived - public Base
- d. class Derived(public) : Base

Answer: a. class Derived: public Base

In C++, what is the significance of the `protected` access specifier in a derived class?

- a. Members are accessible only within the same class
- b. Members are accessible only within the same file
- c. Members are accessible in derived classes and within the same class
- d. Members are not accessible in any derived class

Answer: c. Members are accessible in derived classes and within the same class

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

- a. base
- b. this
- c. super
- d. :: (scope resolution)

Answer: d. :: (scope resolution)

What is the purpose of the `override` keyword in C++?

- a. It indicates that a function is virtual
- b. It specifies that a function is static
- c. It declares the intention to override a virtual function in a derived class
- d. It is used to override the access specifier of a base class

Answer: c. It declares the intention to override a virtual function in a derived class

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

- a. Single inheritance
- b. Multiple inheritance
- c. Hierarchical inheritance
- d. Multilevel inheritance

Answer: b. Multiple inheritance

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

- a. They cannot be overridden in derived classes
- b. They provide a default implementation in the base class
- c. They force derived classes to provide their own implementation
- d. They are static methods in a class

Answer: c. They force derived classes to provide their own implementation

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

- a. It makes the destructor private
- b. It allows the destructor to be overridden in derived classes
- c. It prevents the destructor from being called
- d. It is not allowed to be used with destructors

Answer: b. It allows the destructor to be overridden in derived classes

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

- a. Converting a derived class object to a base class object
- b. Converting a base class object to a derived class object
- c. Converting a derived class to an abstract class
- d. Converting a base class to a friend class

Answer: a. Converting a derived class object to a base class object

In C++, what is the purpose of the `dynamic_cast` operator?

- a. To cast a variable to a different data type
- b. To perform type checking during runtime for polymorphic classes
- c. To convert a derived class object to a base class object
- d. To allocate memory dynamically

Answer: b. To perform type checking during runtime for polymorphic classes

What is the significance of the `protected` keyword in a derived class in C++?

- a. It makes the members accessible only within the same class
- b. It makes the members accessible in any class within the same file
- c. It makes the members accessible only in derived classes
- d. It makes the members accessible to any class

Answer: c. It makes the members accessible only in derived classes

In C++, what is the purpose of the `friend` keyword in the context of inheritance?

- a. To indicate that a class is derived
- b. To specify that a function is a friend of a derived class
- c. To allow private members of a class to be accessible in a derived class
- d. To define a class as a base class

Answer: c. To allow private members of a class to be accessible in a derived class

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; }
};
```

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; }

Answer: b. int area() 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 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; }
};
```

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; }

Answer: b. void speak() override { 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. B
c. AB
d. Compiler error

Answer: b. B

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
b. Derived
c. Compiler error
d. Undefined behavior
Answer: b. Derived

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"; }
};
```

Which option correctly overrides the `start` function in the `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"; }
Answer: a. void start() { 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; }
};
```

Which option correctly overrides the `sound` function 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; }
Answer: a. void sound() override { 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; }
};
```

Which option correctly overrides 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; }
Answer: a. void draw() const override { 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;
}
```

What will be the output of the `main` function?
a. A
b. B
c. AB
d. Compiler error
Answer: b. B

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
b. Derived
c. Compiler error
d. Undefined behavior
Answer: b. Derived

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"; }
};
```

Which option correctly overrides the `start` function in the `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"; }
Answer: a. void start() { cout << "Car started"; }

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

a. To prevent inheritance
b. To enable multiple inheritance without creating ambiguous paths
c. To enforce encapsulation
d. To make a class abstract

> Answer: b. To enable multiple inheritance without creating ambiguous paths

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

> Answer: a. Using the `virtual` keyword before the class name

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

a. Virtual base classes cannot have member variables
b. Each derived class has its own copy of member variables from the virtual base class
c. All derived classes share a single copy of member variables from the virtual base class
d. Member variables from the virtual base class are inherited as private in all derived classes

> Answer: c. All derived classes share a single copy of member variables from the virtual base class

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

a. Diamond problem
b. Encapsulation problem
c. Polymorphism problem
d. Ambiguity problem

Answer: d. 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 {};

// What will be the size of an object of class D?
// a. 4 bytes
// b. 8 bytes
// c. 12 bytes
// d. 16 bytes
```

Answer: c. 12 bytes

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

a. Compiler error
b. Undefined behavior
c. Default values are assigned automatically
d. No impact, the compiler handles it implicitly
Answer: a. Compiler error

Which keyword is used to access a virtual base class member in a derived class in C++?

a. `base`
b. `virtual`
c. `vb`
d. `super`
Answer: a. `base`

What is the order of constructor invocation in a hierarchy involving virtual base classes in C++?

a. From the most derived class to the base class
b. From the base class to the most derived class
c. In random order
d. The order is not defined

Answer: a. From the most derived class to the base class

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

a. To prevent memory leaks
b. To ensure proper destruction of derived class objects
c. To enable polymorphism
d. To avoid object slicing

Answer: b. To ensure proper destruction of derived class objects

In a hierarchy involving virtual base classes, which type of inheritance relationship is established among the classes?

a. Public inheritance
b. Protected inheritance
c. Private inheritance
d. Virtual inheritance

Answer: d. Virtual inheritance

What is an abstract class in C++?

a. A class with no member variables
b. A class that cannot be instantiated and may have pure virtual functions
c. A class with only private members
d. A class with static member functions

Answer: b. A class that cannot be instantiated and may have pure virtual functions

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

a. `virtual void func() = 0;`
b. `void virtual func() = 0;`
c. `pure virtual void func();`
d. `void func() pure virtual;`

Answer: a. `virtual void func() = 0;`

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

a. Yes, always
b. Yes, but only default constructor
c. No, abstract classes cannot have constructors
d. No, abstract classes can only have a destructor

Answer: a. Yes, always

In C++, can a concrete (non-abstract) class inherit from an abstract class without implementing its pure virtual functions?

a. Yes, but only if the abstract class has a default constructor
b. Yes, but only if the concrete class is also marked as abstract
c. No, it will result in a compilation error
d. No, it will result in a runtime error
Answer: c. No, it will result in a compilation error

In C++, what is a virtual function?

a. A function that is always static
b. A function declared with the `virtual` keyword in the base class
c. A function that cannot be overridden in derived classes
d. A function that is always pure virtual

Answer: b. A function declared with the `virtual` keyword in the base class

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

a. To make a function constant
b. To enable polymorphism and late binding
c. To make a function private
d. To make a function static
Answer: b. To enable polymorphism and late binding

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

a. It results in a compilation error
b. It results in a runtime error
c. The base class function is called
d. It automatically becomes a pure virtual function in the derived class

Answer: c. The base class function is called

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

a. override
b. extend
c. implement
d. inherit
Answer: a. Override

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

a. Resolving function calls at compile-time
b. Resolving function calls at link-time
c. Resolving function calls at runtime
d. Resolving function calls at class definition
Answer: c. Resolving function calls at runtime

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

a. Yes, always
b. Yes, but only if it is also declared as static
c. No, it must be at least protected
d. No, it must be public
Answer: d. No, it must be public

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

a. To initialize only the base class members
b. To initialize only the derived class members
c. To initialize both the base and derived class members
d. Constructors are not allowed in derived classes
Answer: c. To initialize both the base and derived class members

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

a. Using the `base` keyword
b. Using the `super` keyword
c. Using the `this` keyword
d. Using the `::` (scope resolution) operator
Answer: d. Using the `::` (scope resolution) operator

What is the order of execution of constructors in a multiple inheritance scenario in C++?

a. Random order
b. From the most derived class to the base class
c. From the base class to the most derived class
d. Constructors are not executed in multiple inheritance
Answer: c. From the base class to the most derived class

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

a. The base class constructor is called automatically
b. It results in a compilation error
c. The derived class inherits the base class constructor
d. The program runs without any issue
Answer: a. The base class constructor is called automatically

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

a. `virtual`
b. `override`
c. `inherit`
d. `extend`
Answer: b. `override`

In C++, what happens if a virtual function in a derived class does not have the `override` keyword?

a. It results in a compilation error
b. The virtual function is treated as a regular function
c. It is automatically treated as an override
d. The program runs without any issue
Answer: c. 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?

a. ABCCBA
b. CBAABC
c. ACBCBA
d. CABABC
Answer: a. ABCCB~A

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

a. It results in a compilation error
b. The derived class implementation is ignored
c. Both the base and derived class implementations are called
d. The base class implementation is overridden

Answer: c. Both the base and derived class implementations are called

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

a. To prevent memory leaks
b. To ensure proper destruction of derived class objects
c. To enable polymorphism
d. Destructors are not allowed in base classes
Answer: b. To ensure proper destruction of derived class objects

What is the significance of the `virtual` keyword in the base class destructor in C++?

a. It indicates that the destructor is static
b. It allows the destructor to be overridden in derived classes
c. It prevents the destructor from being called
d. It makes the destructor constant
Answer: b. It allows the destructor to be overridden in derived classes

What is Hybrid Inheritance in C++?
a. Inheriting from multiple classes using a common base class
b. Inheriting from multiple classes with no common base class
c. Inheriting from a single class
d. Inheriting from classes with virtual functions only
Answer: a. Inheriting from multiple classes using a common base class
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 {};



// What is the type of inheritance represented by class D?

// a. Hybrid inheritance

// b. Hierarchical inheritance

// c. Multiple inheritance

// d. Multipath inheritance
```

Answer: a. Hybrid inheritance

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

a. Ambiguity in multiple inheritance
b. Ambiguity in multipath inheritance
c. Ambiguity in hierarchical inheritance
d. Ambiguity in constructor calls
Answer: a. Ambiguity in multiple inheritance

Hierarchical Inheritance:

What is Hierarchical Inheritance in C++?

a. Inheriting from multiple classes using a common base class
b. Inheriting from a single class with multiple derived classes
c. Inheriting from multiple classes with no common base class
d. Inheriting from abstract classes only
Answer: b. Inheriting from a single class with multiple derived classes

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 {};



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

// a. Hybrid inheritance

// b. Hierarchical inheritance

// c. Multiple inheritance

// d. Single inheritance
```

Answer: b. Hierarchical inheritance

What is the Diamond Problem in C++?

a. The challenge of defining multiple constructors in a class
b. The ambiguity that arises in multiple inheritance with a common base class
c. The difficulty in using the virtual keyword
d. The limitation of using polymorphism

Answer: b. The ambiguity that arises in multiple inheritance with a common base class

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 {};



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

// a. Multiple inheritance

// b. Multipath inheritance

// c. Diamond problem

// d. Hybrid inheritance
```

Answer: c. Diamond problem

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

a. By avoiding multiple inheritance
b. By using the virtual keyword in the base class
c. By making all functions static
d. By using the final keyword in the derived classes
Answer: b. By using the virtual keyword in the base class

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 {};



// What type of inheritance is represented by the classes B, C, and D?

// a. Multiple inheritance

// b. Multipath inheritance

// c. Diamond problem

// d. Hybrid inheritance
```

Answer: b. Multipath inheritance

In C++, how does the `virtual` keyword address the ambiguity in multiple inheritance?

a. It prevents multiple inheritance
b. It ensures that the base class is not inherited
c. It allows the compiler to identify the correct base class in the hierarchy
d. It enforces a specific order in which classes are inherited
Answer: c. It allows the compiler to identify the correct base class in the hierarchy

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 CartItem 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 - 3

Expected 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 the destruction of the secret base

Question 3.

Engineers are building an advanced spaceship for interstellar travel. Create a class Spaceship with 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 inches

Wand2 - 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 for specific 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 a pure 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 Whisper

Type: 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 a class ShoppingCart with a method addItem that adds items to the cart. If the customer attempts to 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 Quantity

Input: 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.