# Assignment 3: C++ OOP Theory Questions (1-53)

**1. What is an object in C++?**

An object is an instance of a class. It represents a real-world entity and can access the class's data and functions.

**2. What is a class in C++ and how does it differ from an object?**

A class is a blueprint or template that defines the structure and behavior (data and functions) of objects. A class is like a plan; an object is something built using that plan.

**3. Explain the concept of encapsulation with an example.**

Encapsulation is the process of keeping data and functions that operate on that data together in one unit (class), and restricting access to internal details.

Example:

```
class Bank {
private:
    int balance;
public:
    void setBalance(int b) { balance = b; }
    int getBalance() { return balance; }
};
```

**4. How do you define a class in C++?**

```
class ClassName {
    // data members
    // member functions
};
```

**5. Describe the syntax for creating an object of a class.**

```
ClassName obj; // obj is an object of ClassName
```

**6. What are private members in a class and how are they accessed?**

Private members are accessible only inside the class. They cannot be accessed directly from outside the class.

**7. What are public members in a class and how are they accessed?**

Public members can be accessed from outside the class using an object.

Example: obj.publicFunction();

**8. Explain the significance of access specifiers in a class.**

Access specifiers (private, public, protected) control how members of a class can be accessed.

**9. Provide an example of a class with both private and public members.**

```
class Student {
private:
    int roll;
public:
    void setRoll(int r) { roll = r; }
    int getRoll() { return roll; }
};
```

**10. How does data hiding work in C++?**

Data hiding means keeping class details private to protect object data from direct access and misuse. It is done using private or protected access specifiers.

**11. What is a static data member in C++?**

A static data member belongs to the class, not to any specific object. All objects share the same copy of the static data member.

**12. How do you declare and initialize a static data member?**

Declare inside the class: static int count;

Define outside the class: int ClassName::count = 0;

**13. What is a static function member in C++?**

A static function member can be called without creating an object and can only access static members.

**14. How do static function members differ from regular function members?**

Static function members do not depend on any object and can only access other static members. Regular functions can access all members.

**15. Provide an example of a class with static data and function members.**

```
class MyClass {
public:
    static int count;
    static void showCount() {
        cout << count;
    }
};
int MyClass::count = 0;
```

**16. What is a constructor in C++ and why is it important?**

A constructor is a special function that is automatically called when an object is created. It is used to initialize objects.

**17. Explain the different types of constructors in C++.**

Types include: Default constructor, Parameterized constructor, Copy constructor, and Constructor with initializer list.

**18. What is a default constructor and when is it used?**

A default constructor takes no arguments and is used to initialize objects with default values.

**19. How do parameterized constructors work?**

They accept arguments to initialize data members at the time of object creation.

**20. What is a copy constructor and what is its purpose?**

A copy constructor creates a new object by copying data from an existing object. Used in pass-by-value or object returning.

**21. Explain the concept of constructor overloading.**

Constructor overloading means having multiple constructors with different parameter lists in the same class.

**22. How does a constructor initializer list work?**

It initializes data members directly before entering the constructor body.

Example: MyClass(int a) : x(a) {}

**23. What is a destructor in C++ and what is its purpose?**

A destructor is a special function that is called when an object goes out of scope. It releases resources.

**24. How is a destructor declared and defined?**

Declared using a tilde (~) followed by class name: ~ClassName(); Defined like a regular function.

**25. What happens if a destructor is not explicitly defined in a class?**

The compiler provides a default destructor that handles basic cleanup.

**26. Explain the concept of automatic and dynamic storage duration in relation to destructors.**

Automatic objects are destroyed when they go out of scope. Dynamic objects must be deleted manually using delete to call destructor.

**27. How do destructors differ from constructors?**

Constructors initialize objects; destructors clean up. Constructors can be overloaded, destructors cannot.

**28. What is operator overloading in C++ and why is it useful?**

It allows you to redefine the behavior of operators for user-defined types.

**29. Describe the syntax for overloading an operator.**

ReturnType operator+(const ClassName& obj) { /* logic */ }

**30. Which operators can and cannot be overloaded in C++?**

Most operators can be overloaded. Cannot overload: ::, ., .*, sizeof, typeid, ?:.

**31. Provide an example of overloading the '+' operator for a custom class.**

```
class Complex {
int a, b;
public:
    Complex operator+(const Complex& c) {
        return Complex(a + c.a, b + c.b);
    }
};
```

**32. Explain the concept of friend functions in the context of operator overloading.**

A friend function can access private members and is used for overloading operators like << and >>.

**33. What is a friend function in C++ and how is it declared?**

It is declared with the keyword 'friend' inside a class. It can access private/protected members.

**34. How do friend functions differ from member functions?**

Friend functions are not members of the class, but they can access private data. They are called without an object.

**35. Explain the benefits and potential drawbacks of using friend functions.**

Benefits: useful for operator overloading. Drawbacks: breaks encapsulation.

**36. What is inheritance in C++ and why is it important?**

Inheritance lets a class (derived) use properties and behaviors of another class (base). It promotes code reuse.

**37. Explain the different types of inheritance in C++.**

Types: Single, Multiple, Multilevel, Hierarchical, Hybrid.

**38. How do you implement single inheritance in C++?**

class B : public A { };

**39. What is multiple inheritance and how does it differ from single inheritance?**

Multiple inheritance allows a class to inherit from more than one class. Single from only one.

**40. Describe hierarchical inheritance with an example.**

class A {}; class B : public A {}; class C : public A {}; // A is base for B and C

**41. What is multilevel inheritance and how is it implemented in C++?**

class A {}; class B : public A {}; class C : public B {}; // A -> B -> C

**42. Explain the concept of hybrid inheritance.**

It is a mix of multiple and multilevel inheritance. It can lead to diamond problem.

**43. What are access modifiers in C++ and what are the different types?**

Access modifiers: public, private, protected. They control visibility of class members.

**44. How do public, private, and protected access modifiers affect inheritance?**

Public: keeps same visibility. Protected: base's public/protected become protected. Private: base's public/protected become private.

**45. Explain how access modifiers control member accessibility in derived classes.**

Public members: accessible everywhere

Protected: accessible in derived classes

Private: not accessible in derived

**46. What is function overriding in the context of inheritance?**

When a derived class defines a function with same signature as base class to provide specific behavior.

**47. How do you override a base class function in a derived class?**

Declare same function in derived class with same name and parameters.

**48. Explain the use of the "virtual" keyword in function overriding.**

Makes base class function virtual to support runtime polymorphism.

**49. What is the significance of the "override" specifier in C++11 and later?**

Ensures the function is actually overriding a virtual function from base class. Helps catch errors.

**50. What is a virtual base class in C++ and why is it used?**

Used to avoid multiple copies of base class when using multiple inheritance. Solves diamond problem.

**51. How do you declare and implement a virtual base class?**

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

**52. Explain the role of virtual base classes in resolving ambiguity in multiple inheritance.**

Ensures only one shared instance of the base class is present in the derived class.

**53. Provide an example of using a virtual base class to avoid the diamond problem in inheritance.**

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