1. Introduction to Inheritance

Inheritance is one of the most important features of **Object-Oriented Programming (OOP)**. It allows a new class to **acquire the properties and behaviors** of an existing class. The existing class is called the **base class (or parent class)**, and the new class is called the **derived class (or child class)**.

Inheritance helps in **code reusability**, **better organization**, and **reduced redundancy**.

---

2. Meaning of Inheritance

The word *inheritance* means receiving properties from ancestors. In programming, it means a class can reuse the data members and member functions of another class.

For example, a `Dog` class can inherit properties like `eat()` and `sleep()` from an `Animal` class.

---

3. Need for Inheritance

Inheritance is needed because:

- It avoids code duplication
- It improves program readability
- It simplifies maintenance
- It supports hierarchical classification

Without inheritance, large programs become difficult to manage and update.

---

4. Inheritance in Real Life

Real-life examples of inheritance include:

- A child inherits traits from parents
- A car model inherits features from a base model
- A smartphone inherits features from previous versions

These examples show how inheritance helps reuse existing features.

---

5. Inheritance in C++

In C++, inheritance is implemented using the **colon (:)** symbol.

Syntax

```
class Derived : access_specifier Base {
  // members
};

The access specifier can be public, protected, or private.
```

---

6. Base Class and Derived Class

Base Class

- The class whose properties are inherited
- Also called parent or super class

Derived Class

- The class that inherits properties
- Also called child or sub class

---

7. Types of Inheritance in C++

C++ supports five types of inheritance:

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

---

8. Single Inheritance

In single inheritance, one derived class inherits from one base class.

Example

```
class Animal {
  public:
    void eat() {}
};

class Dog : public Animal {
};
```

---

9. Multiple Inheritance

In multiple inheritance, one derived class inherits from more than one base class.

Example

```
class A {
};

class B {
};

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

---

10. Multilevel Inheritance

In multilevel inheritance, a class is derived from another derived class.

Example

```
class A {
};

class B : public A {
};

class C : public B {
};
```

---

11. Hierarchical Inheritance

In hierarchical inheritance, multiple derived classes inherit from a single base class.

Example

```
class Animal {
};

class Dog : public Animal {
};

class Cat : public Animal {
};
```

---

12. Hybrid Inheritance

Hybrid inheritance is a combination of two or more types of inheritance.

It may cause ambiguity, which is handled using **virtual base classes**.

---

13. Access Specifiers in Inheritance

Access specifiers determine how base class members are inherited.

Base Member	Public Inheritance	Protected Inheritance	Private Inheritance
public	public	protected	private
protected	protected	protected	private
private	Not accessible	Not accessible	Not accessible

14. Visibility of Base Class Members

- Private members of base class cannot be accessed directly in derived class
- Public and protected members can be accessed depending on inheritance type

15. Constructors and Inheritance

- Base class constructor is called before derived class constructor
- Destructor of derived class is called before base class destructor

This ensures proper object initialization and cleanup.

16. Function Overriding

Function overriding occurs when a derived class provides its own implementation of a base class function.

Example

```
class Base {
  public:
    void show() {}
};

class Derived : public Base {
  public:
    void show() {}
};
```

17. Virtual Functions and Inheritance

Virtual functions support **runtime polymorphism**. They ensure that the correct function is called based on the object type.

Example

```
class Base {
  public:
    virtual void display() {}
};
```

---

18. Advantages of Inheritance

- Code reusability
- Reduced redundancy
- Easy maintenance
- Supports extensibility
- Improves code structure

---

19. Limitations of Inheritance

- Increases complexity
- Tight coupling between classes
- Changes in base class may affect derived classes
- Not suitable for all problems

---

20. Applications of Inheritance

Inheritance is widely used in:

- GUI frameworks
- Game development
- Banking systems
- Software libraries
- Operating systems

---

21. Common Mistakes in Inheritance

- Using inheritance unnecessarily
- Improper access specifiers
- Deep inheritance hierarchies
- Ignoring virtual destructors

---

22. Best Practices for Inheritance

- Use inheritance only when “is-a” relationship exists
- Keep base classes simple
- Prefer composition when suitable
- Use virtual functions carefully

---

23. Inheritance vs Composition

Inheritance represents an “is-a” relationship, while composition represents a “has-a” relationship.

Choosing the correct approach improves program design.

---

24. Conclusion

Inheritance is a powerful feature of C++ that allows reuse of existing code and creation of hierarchical class structures. When used properly, inheritance improves code readability, maintainability, and scalability. Understanding inheritance is essential for mastering Object-Oriented Programming in C++.