What is Inheritance ?

Need of Inheritance

Can OOP exist without Inheritance ?

Implementation

Access Modifiers and Inheritance

Inheritance Ambiguity

Limitations of Inheritance

------------------------------------

Inheritance is one of the key features of Object-oriented programming in C++. It allows us to create a new class (derived class) from an existing class (base class).

The derived class inherits the fe atures from the base class and can have additional features of its own.

Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

class parent\_class

{

//Body of parent class

};

class child\_class: access\_modifier parent\_class

{

//Body of child class

};

Here, child\_class is the name of the subclass, access\_mode is the mode in which you want to inherit this sub-class, for example, public, private, etc., and parent\_class is the name of the superclass from which you want to inherit the subclass.

**Modes of Inheritance**

Public mode: If we derive a subclass from a public base class. Then, the base class’s public members will become public in the derived class, and protected class members will become protected in the derived class.

Protected mode: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

Private mode: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

: 忄 0 
01)C)0d 

**Access Modifiers:**

1. **public:** Members declared as public can be accessed from anywhere outside the class.
2. **protected:** Members declared as protected can only be accessed by derived classes and friends.
3. **private:** Members declared as private cannot be accessed directly from outside the class or by derived classes, but can be accessed within the class itself.

Base class 
member 
access 
s ectfier 
Public 
Protected 
Private 
Type of Inheritence 
Public 
Public 
Protected 
Not accessible 
(Hidden) 
Protected 
Protected 
Protected 
Not accessible 
(Hidden) 
Private 
private 
Not accessible 
(Hidden) 

Example: 
Suppose we have three classes with names: car, bicycle, and truck. The properties for 
each are as follows: 
Car 
• Colour 
• Max Speed 
• Number of Gears 
Bicycle 
• Colour 
• Max Speed 
• Is foldable? 
Truck 
• Colour 
• Max Speed 
• Max weight 
From above, we can see that two of the properties: Colour and MaxSpeed, are the 
same for every object. Hence, we can combine all these in one parent class and make 
three classes their subclass. This property is called Inheritance. 

From above, we can see that two of the properties: Colour and MaxSpeed, are the same for every object. Hence, we can combine all these in one parent class and make three classes their subclass. This property is called Inheritance.

Technically, inheritance is defined as the process of acquiring the features and behaviors of a class by another class. Here, the class that contains these members is called the base class, and the class that inherits these members from the base class is called the derived class of that base class.

#include <iostream>

using namespace std;

// Base class (Parent)

class Parent {

protected:

int protectedVar; // Protected: Accessible in derived classes

public:

int publicVar; // Public: Accessible anywhere

Parent() : protectedVar(0), publicVar(0) {} // Constructor to initialize variables

void setValues(int a, int b) {

protectedVar = a;

publicVar = b;

}

void showParentValues() {

cout << "Parent class - Protected Var: " << protectedVar << ", Public Var: " << publicVar << endl;

}

};

// Derived class (Child)

class Child : public Parent { // Inherits Parent publicly

public:

void showChildValues() {

// Can access protected and public members of the base class

cout << "Child class - Protected Var: " << protectedVar << ", Public Var: " << publicVar << endl;

}

void changeValues(int a, int b) {

// Directly accessing protectedVar is allowed in derived class

protectedVar = a;

publicVar = b;

}

};

int main() {

Child obj;

// Accessing public members directly

obj.publicVar = 10;

// obj.protectedVar = 5; // Error: Cannot access protected member directly

obj.setValues(5, 20); // Can modify protectedVar using public methods

obj.showParentValues(); // Display Parent class values

obj.showChildValues(); // Display Child class values

obj.changeValues(15, 30); // Modify values through Child class

obj.showParentValues(); // Updated values from Parent's view

obj.showChildValues(); // Updated values from Child's view

return 0;

}

**Explanation:**

1. **Parent class:**
   * **protectedVar is a protected member, meaning it can be accessed in derived classes (like Child), but not outside the class.**
   * **publicVar is a public member, accessible both inside and outside the class.**
2. **Child class:**
   * **Inherits Parent class publicly, so it can access protectedVar and publicVar but not private members (if there were any).**
   * **It has access to protectedVar directly because it’s declared as protectedin the Parent class.**
   * **We modify and display inherited members in the Child class using showChildValues() and changeValues().**

Output

Parent class - Protected Var: 5, Public Var: 20

Child class - Protected Var: 5, Public Var: 20

Parent class - Protected Var: 15, Public Var: 30

Child class - Protected Var: 15, Public Var: 30

@Inheritance

#include<iostream>

using namespace std;

class Human {

private:

int height;

public:

int weight;

private:

int age;

public:

int getAge() {

return this->age;

}

void setWeight(int w) {

this->weight = w;

}

};

class Male: private Human {

public:

string color;

void sleep() {

cout << "Male Sleeping" << endl;

}

int getHeight() {

return this->height;

}

};

int main() {

Male m1;

//cout << m1.height << endl;

/\*

Male object1;

cout << object1.age << endl;

cout << object1.weight << endl;

cout << object1.height << endl;

cout << object1.color << endl;

object1.setWeight(84);

cout << object1.weight << endl;

object1.sleep();

\*/

return 0;

}

@inheritanceambiguity

#include<iostream>

using namespace std;

class A {

public:

void func() {

cout << " I am A" << endl;

}

};

class B {

public:

void func() {

cout << " I am B" << endl;

}

};

class C: public A, public B {

};

int main() {

C obj;

//obj.func();

obj.A::func() ;

obj.B::func();

return 0;

}

This C++ code demonstrates the concept of ambiguity in inheritance. Let's break it down:

**Class Structure:**

1. **Class A**: Contains a public member function func() that prints "I am A".
2. **Class B**: Contains a public member function func() that prints "I am B".
3. **Class C**: Inherits from both A and B (multiple inheritance).

**Ambiguity in Multiple Inheritance:**

* When class C inherits from both A and B, it has two versions of the func() method: one from A and one from B.
* This causes ambiguity because the compiler doesn't know which version of func()to call if you try obj.func().

**Code Walkthrough:**

1. **Uncommented obj.func()**:
   * **If you were to use obj.func() (currently commented out), the compiler would throw an error because of the ambiguity: both A and B have a func() method, and the compiler cannot decide which one you want to call.  
       
     csharp  
     Copy code  
     error: request for member 'func' is ambiguous**
2. **Explicitly Calling obj.A::func() and obj.B::func()**:
   * **By specifying obj.A::func(), you're explicitly telling the compiler to use the func() from class A.**
     + **Output: "I am A"**
   * **Similarly, obj.B::func() calls the func() method from class B.**
     + **Output: "I am B"**

**Output:**

plaintext

Copy code

I am A  
I am B

**Key Takeaway:**

* In cases of multiple inheritance where two parent classes have the same function signature, you need to resolve ambiguity by explicitly specifying which parent class's method you want to call using the scope resolution operator (::).