What is Polymorphism ?

Need of Polymorphism ?

-Function / Operator Overloading

-Function Overriding

Function / Operator Overloading

Compile time Polymorphism

Functions cannot be overloaded in C++

Operators that cannot be overloaded in C++

Function Overriding

Run Time Polymorphism

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

#include<iostream>

using namespace std;

class A {

public:

void sayHello() {

cout << "Hello Love Babbar" << endl;

}

int sayHello(char name) {

cout << "Hello Love Babbar" << endl;

return 1;

}

void sayHello(string name) {

cout << "Hello " << name << endl;

}

};

class B {

public:

int a;

int b;

public:

int add() {

return a+b;

}

void operator+ (B &obj) {

/\* int value1 = this -> a;

int value2 = obj.a;

cout << "output " << value2 - value1 << endl;

\*/

cout << "Hello Babbar" << endl;

}

void operator() () {

cout << "main Bracket hu " << this->a << endl;

}

};

class Animal {

public:

void speak() {

cout << "Speaking "<< endl;

}

};

class Dog: public Animal {

public:

void speak() {

cout << "Barking " << endl;

}

};

int main() {

Dog obj;

obj.speak();

/\*

B obj1, obj2;

obj1.a = 4;

obj2.a = 7;

obj1 + obj2;

obj1();\*/

/\*

A obj;

obj.sayHello();

\*/

return 0;

}

new vs override

In C++ Object-Oriented Programming (OOP), both new and override are related to polymorphism but serve different purposes. Here's a detailed explanation of each:

**1. new (in context of polymorphism)**

* **Purpose**: The new keyword can be used in two contexts in C++. In the context of OOP, it's related to the new operator for dynamic memory allocation and to method hiding in inheritance.
* **Dynamic Memory Allocation**: new is primarily used to allocate memory dynamically for objects or primitive types at runtime.  
    
  cpp  
  Copy code  
  int\* ptr = new int; // Dynamically allocates an integer
* **Method Hiding**: When a derived class defines a function with the same name as a base class function but not necessarily with the same signature (parameters), the base class function is hidden (not overridden). To explicitly indicate this intention, the new keyword is used in some other languages like C# but not in C++.  
  In C++, method hiding happens automatically without a keyword:  
    
  cpp  
  Copy code  
  class Base {  
  public:  
   void show() {  
   std::cout << "Base class" << std::endl;  
   }  
  };  
    
  class Derived : public Base {  
  public:  
   void show() { // This hides the Base class function  
   std::cout << "Derived class" << std::endl;  
   }  
  };

**2. override (in context of virtual functions)**

* **Purpose**: The override keyword in C++ is used to indicate that a virtual function in a derived class is meant to override a base class function. It helps catch errors at compile time by ensuring that a function is actually overriding a virtual function from the base class.
* **When to use**: When you have a base class function marked virtual, and you want to override it in the derived class, you use override to ensure it's an actual override.
* **Example**:  
    
  cpp  
  Copy code  
  class Base {  
  public:  
   virtual void show() {  
   std::cout << "Base class" << std::endl;  
   }  
  };  
    
  class Derived : public Base {  
  public:  
   void show() override { // Correct override  
   std::cout << "Derived class" << std::endl;  
   }  
  };
* **What happens if override is missing**: If you accidentally mismatch the function signature, the function won't actually override the base class function, and the override keyword helps catch that mistake.  
    
  cpp  
  Copy code  
  class Derived : public Base {  
  public:  
   void show(int x) override { // ERROR: Doesn't match base class function signature  
   std::cout << "Derived class" << std::endl;  
   }  
  };

**Summary:**

* **new**: Used for dynamic memory allocation and related to method hiding (not overriding).
* **override**: Ensures that a derived class method is overriding a virtual function from the base class and helps catch potential mistakes during compile-time.