**override Keyword in C++**

[Function overriding](https://www.geeksforgeeks.org/function-overloading-vs-function-overriding-in-cpp/) is redefinition of base class function in its derived class with same signature i.e return type and parameters.  
But there may be situations when a programmer makes a mistake while overriding that function. So, to keep track of such an error, C++11 has come up with the keyword override. It will make the compiler to check the base class to see if there is a virtual function with this exact signature. And if there is not, the compiler will show an error.

This will be more clear from the following example:

|  |
| --- |
| // A CPP program without override keyword. Here  // programmer makes a mistake and it is not caught.  #include <iostream>  using namespace std;    class Base {  public:        // user wants to override this in      // the derived class      virtual void func() {          cout << "I am in base" << endl;      }  };    class derived : public Base {  public:      // did a silly mistake by putting      // an argument "int a"      void func(int a) {          cout << "I am in derived class" << endl;      }  };    // Driver code  int main()  {      Base b;      derived d;      cout << "Compiled successfully" << endl;      return 0;  } |

Output:

Compiled successfully

**Explanation**: Here the user intended to override the function func() in the derived class but did a silly mistake and redefined the function with different signature. Which was not detected by the compiler. However, the program is not actually what the user wanted. So, to get rid of such silly mistake to be in safe side, override keyword can be used.  
Below is a C++ example to show the use of override keyword in C++.

|  |
| --- |
| // A CPP program that uses override keyword so  // that any difference in function signature is  // caught during compilation.  #include <iostream>  using namespace std;    class Base {  public:        // user wants to override this in      // the derived class      virtual void func()      {          cout << "I am in base" << endl;      }  };    class derived : public Base {  public:        // did a silly mistake by putting      // an argument "int a"      void func(int a) override      {          cout << "I am in derived class" << endl;      }  };    int main()  {      Base b;      derived d;      cout << "Compiled successfully" << endl;      return 0;  } |

Output:

prog.cpp:17:7: error: 'void derived::func(int)'

marked 'override', but does not override

void func(int a) override

^

In short, it serves the following functions. It helps to check if :

* There is a method with the same name in the parent class.
* The method in the parent class is declared as “virtual” which means it was intended to be rewritten.
* The method in the parent class has the same signature as the method in the subclass.

#include <iostream>

using namespace std;

class Base {

public:

// user wants to override this in

// the derived class

virtual const int func() const {

cout << "I am in base" << endl;

return 1;

}

};

class derived : public Base {

public:

// did a silly mistake by putting

// an argument "int a"

const int func() const override {

cout << "I am in derived class" << endl;

return 1;

}

};

// Driver code

int main()

{

Base b;

derived d;

cout << "Compiled successfully" << endl;

return 0;

}

|  |  |  |
| --- | --- | --- |
| Base | Derived | Error? |
| virtual int func() | int func(int) override | Error |
| virtual int func() | int func() override | No Error |
| virtual const int func() const | int func() const override | Error |
| virtual int func() | int func() const override | Error |
| virtual const int func() | int func() override | Error |

Signature for base class and derived class function muct be same for overriding the function.

Override keyword (C++11) checks this.