override

In C++03, it is possible to accidentally create a new virtual function, when one intended to override a base class function. For example:

// C++03

class Base

{

virtual void f(int);

};

class Derived : public Base

{

virtual void f(float);

};

The intention was to override **Base::f()** and to use **Derived::f()**. But instead, because it has a different signature, it creates a second virtual function.

// C++11

class Base

{

virtual void f(int);

};

class Derived : public Base

{

virtual void f(float) override; // Error

};

The override special identifier means that the compiler will check the base class(es) to see if there is a virtual function with this exact signature. And if there is not, the compiler will indicate an error.

final

C++11 also adds the ability to prevent inheriting from classes or simply preventing overriding methods in derived classes. This is done with the special identifier final. For example:

// no class can be derived from class A

class A final

{

virtual void f(int);

};

class B

{

// no class can override f()

virtual void f() final;

};

So, if we do the following, the compiler gives us errors:

class C : public A {}; // Error

class D : public B // Error

{

virtual void f();

};

Note that **override** and **final** are **not language keywords**. They are technically **identifiers** for declarator attributes.