- Examples

In this lesson, we'll discuss the examples of final and override.

```
WE'LL COVER THE FOLLOWING ^
Example 1: Override final
Explanation
Example 2: Override
Explanation
Example 3: final
Explanation
```

Example 1: Override final

```
#include <iostream>
class Sort{
public:
  virtual void processData(){
    readData();
    sortData();
    writeData();
private:
  virtual void readData(){}
 virtual void sortData() = 0;
 virtual void writeData(){}
};
class QuickSort: public Sort{
private:
  void readData(){
    std::cout << "readData" << std::endl;</pre>
  void sortData(){
    std::cout << "sortData" << std::endl;</pre>
  void writeData(){
    std::cout << "writeData" << std::endl;</pre>
```

```
int main(){
    std::cout << std::endl;
    Sort* sort = new QuickSort;
    sort->processData();
    std::cout << std::endl;
}</pre>
```

Explanation

- We have implemented two classes named **Sort** and **QuickSort**.
- We have created three private virtual methods and a public virtual method processData in the Sort class which calls the three private methods.
- The QuickSort class publicly inherits from the Sort class.
- We have overridden the methods of the **Sort** class in **QuickSort**.
- By using a pointer to the Base class, we can access the overridden methods of the derived class.

Example 2: Override

```
class Base {
  void func1();
  virtual void func2(float);
  virtual void func3() const;
  virtual long func4(int);

  virtual void f();

};

class Derived: public Base {
  // ill-formed; no virtual method func1 exists
  virtual void fun1() override;

// ill-formed: had type
```

```
virtual void func2(double) override;

// ill-formed: const missing
  virtual void func3() override;

// ill-formed: wrong return type
  virtual int func4(int) override;

// well-formed: f override Base::f
  virtual void f() override;

};

int main(){
  Base base;
  Derived derived;
};
```







[]

Explanation

When we compile the program, the compiler will complain a lot. The error message is very specific.

The compiler complains in line 15 that the method <code>func1</code> is not overriding a method. The same holds true for <code>func2</code>. It has the wrong parameter type. Continuing with the method <code>func3</code>, it complains that <code>func3</code> has no <code>const</code> qualifier. <code>func4</code> has the wrong return type. Only the method <code>f</code> in line 27 correctly overrides the method <code>f</code> of its base class.

final is the right tool for the job if a virtual method should not be overridden.

Example 3: final

The given function causes a compilation error because the base method is declared final.

```
class Base {
  virtual void h(int) final;

  virtual void g(long int);
};
```

```
class Derived: public Base {
  // ill-formed: base method declared final
  virtual void h(int);
   // well-formed: a new virtual function
  virtual void h(double);
  virtual void g(long int) final;
};
class DerivedDerived: public Derived {
 virtual void g(long int);
};
struct FinalClass final { };
struct DerivedClass: FinalClass { };
int main(){
  Base base;
 Derived derived;
 DerivedDerived derivedDerived;
  FinalClass finalClass;
  DerivedClass derivedClass;
```







Explanation

The compiler performs its job neatly. It complains that the method h in the class Base (line 2) is overridden by the method in class Derived (line 10). Of course, it's okay that the method h (line 13) in class Derived overloads f for the parameter type double. This method g (line 15) in the class Derived is quite interesting. The method overrides the method g (line 4) of the class Base and declares the method final. Therefore, g cannot be overridden in DerivedDerived (line 19).

DerivedClass cannot be derived from FinalClass, because the BaseClass is final.

In the next lesson, we'll solve an exercise related to the example given in this lesson.