Should I use virtual, override, or both keywords?

Asked 5 years, 8 months ago Modified 3 months ago Viewed 122k times

170

59

In the last weeks something is bugging my brain about `virtual` and `override`. I've learned that when you do inheritance with virtual function you have to add `virtual` to let the compiler know to search for the right function. Afterwards I learned also that in c++ 11 there is a new keyword - `override`. Now I'm a little confused; **Do i need to use both virtual and override keywords in my program, or it's better to use only one of them?**

To explain myself - code examples of what I mean:

```
class Base
{
public:
    virtual void print() const = 0;
    virtual void printthat() const = 0;
    virtual void printit() const = 0;
};

class inhert : public Base
{
public:
    // only virtual keyword for overriding.
    virtual void print() const {}

    // only override keyword for overriding.
    void printthat() const override {}

    // using both virtual and override keywords for overriding.
    virtual void printit() const override {}
};
```

What is the best method?

`c++` `c++11`

Share Edit Follow Flag

edited May 8, 2021 at 20:37
Pierre
1,800 ● 2 ● 23 ● 36

asked Oct 8, 2016 at 12:32
Daniel
1,693 ● 2 ● 8 ● 6

9 You should use the `override` keyword whenever you are intending to override a method. That way, if you mistype the name in the derived class, you will get an error telling you there was no method found to override – Eric Oct 8, 2016 at 12:36

3 Is the 'override' keyword just a check for a overriden virtual method? – Solarflare Oct 8, 2016 at 12:37

2 A more interesting question is "What happens if I use override without virtual" – Eric Oct 8, 2016 at 12:39

31 Why is this marked as a duplicate? It clearly states a different question, and a very good question, in my opinion. This question deals with code style (should `virtual` be left out when `override` is used?), whereas the other question deals with the mechanics of `override`. Anyway, I found the answer to my question here. – Lindydancer Apr 6, 2018 at 9:28

@Lindydancer Voted to reopen – bobobobo Oct 23, 2021 at 16:10

2 Answers

Sorted by: Highest score (default)

249

When you override a function you don't technically need to write either `virtual` or `override`.

The original base class declaration needs the keyword `virtual` to mark it as virtual.

In the derived class the function is virtual by way of having the [1]same type as the base class function.

However, an `override` can help avoid bugs by producing a compilation error when the intended override isn't technically an override. For instance, the function type isn't exactly like the base class function. Or that a maintenance of the base class changes that function's type, e.g. adding a defaulted argument.

In the same way, a `virtual` keyword in the derived class can make such a bug more subtle by ensuring that the function is still virtual in the further derived classes.

So the general advice is,

- Use `virtual` for the base class function declaration.
  This is technically necessary.
- Use `override` (only) for a derived class' override.
  This helps maintenance.

Example:

```
struct Base { virtual void foo() {} };
struct Derived: Base { void foo() override {} };
```

Notes:

[1] C++ supports covariant raw pointer and raw reference results. With covariance the type of the override isn't exactly the same. It just has a compatible type.

Share Edit Follow Flag

edited Feb 29, 2020 at 22:33
Ardent Coder
3,519 ● 9 ● 25 ● 46

answered Oct 8, 2016 at 12:55
Cheers and hth. - Alf
139k ● 15 ● 198 ● 315

25 From cppreference (en.cppreference.com/w/cpp/language/virtual): "If some member function vf is declared as virtual in a class Base, and some class Derived, which is derived, directly or indirectly, from Base, has a declaration for member function with the same name parameter type list (but not the return type) cv-qualifiers ref-qualifiers Then this function in the class Derived is also virtual (whether or not the keyword virtual is used in its declaration) and overrides Base::vf (whether or not the word override is used in its declaration)." – solstice333 Jun 23, 2017 at 2:59

2 so in summay `override` is useless – iambr Nov 20, 2019 at 10:42

22 override is optional and as this answer states helps detects coding mistakes – Superfly Jon Nov 26, 2019 at 14:02

6 @iambr no, override doesn't change the behaviour of a *correct* program, but yells loudly at *incorrect* programs. That is *far* from useless. – Caleth Feb 10 at 9:13

Would it still be virtual in the further derived classes if we don't specify the virtual? – Brunisboy Mar 22 at 0:37

20

According to the C++ Core Guidelines C.128, each virtual function declaration should specify ***exactly one*** of `virtual`, `override`, or `final`.

- `virtual`: For the "first" appearance of a function in the base class
- `override`: For overrides of that `virtual` function in a class derived from some base class providing a `virtual` function of the same (or covariant) signature
- `final`: For marking an override as ***unoverrideable***. That is, derivatives of a class with a `final` virtual function override cannot have that virtual function override overridden.

```
struct A {
    virtual void go() { puts("A"); }
};
struct B : A {
    // Overrides A::go(). No need to specify `virtual` explicitly,
    // it already implicitly is virtual and overrideable
    void go() override { puts("B"); }
};
struct C : B {
    void go() final { puts("C"); }
    //virtual void go() override final { puts("C"); } would still compile,
    // but it is considered, for lack of a better description, "extra"
};
struct D : C {
    // Would produce error C3248 : 'C::go' : function declared as 'final' cannot be
overridden by 'D::go'
    //void go() override { puts("D"); }
};
```

Share Edit Follow Flag

edited Feb 10 at 9:09
Edgar Bonet
3,306 ● 14 ● 18

answered Nov 11, 2021 at 17:46
bobobobo
61.6k ● 58 ● 247 ● 348