Meaning of 'const' last in a function declaration of a class?

Asked 12 years, 11 months ago Active 2 months ago Viewed 304k times

817

What is the meaning of `const` in declarations like these? The `const` confuses me.

```
class foobar
{
  public:
     operator int () const;
     const char* foo() const;
};
```

c++ constants declaration c++-faq

Share Edit Follow Flag
edited Jun 3, 2018 at 13:20 Rakete1111 44.2k ●11 ●115 ●153
asked Apr 15, 2009 at 13:27 user91111

Please mark an answer. Thank you – Jesse Roper Mar 2 at 20:04

Add a comment
Start a bounty

12 Answers

Active Oldest Votes

1066

When you add the `const` keyword to a method the `this` pointer will essentially become a pointer to `const` object, and you cannot therefore change any member data. (Unless you use `mutable`, more on that later).

The `const` keyword is part of the functions signature which means that you can implement two similar methods, one which is called when the object is `const`, and one that isn't.

```
#include <iostream>

class MyClass
{
private:
    int counter;
public:
    void Foo()
    {
        std::cout << "Foo" << std::endl;
    }

    void Foo() const
    {
        std::cout << "Foo const" << std::endl;
    }

};

int main()
{
    MyClass cc;
    const MyClass& ccc = cc;
    cc.Foo();
    ccc.Foo();
}
```

This will output

```
Foo
Foo const
```

In the non-const method you can change the instance members, which you cannot do in the `const` version. If you change the method declaration in the above example to the code below you will get some errors.

```
    void Foo()
    {
        counter++; //this works
        std::cout << "Foo" << std::endl;
    }

    void Foo() const
    {
        counter++; //this will not compile
        std::cout << "Foo const" << std::endl;
    }
```

This is not completely true, because you can mark a member as `mutable` and a `const` method can then change it. It's mostly used for internal counters and stuff. The solution for that would be the below code.

```
#include <iostream>

class MyClass
{
private:
    mutable int counter;
public:

    MyClass() : counter(0) {}

    void Foo()
    {
        counter++;
        std::cout << "Foo" << std::endl;
    }

    void Foo() const
    {
        counter++;    // This works because counter is `mutable`
        std::cout << "Foo const" << std::endl;
    }

    int GetInvocations() const
    {
        return counter;
    }
};

int main(void)
{
    MyClass cc;
    const MyClass& ccc = cc;
    cc.Foo();
    ccc.Foo();
```

which would output

```
Foo
Foo const
Foo has been invoked 2 times
```

Share Edit Follow Flag
edited May 28, 2020 at 6:14 stuckexchange 15 ●5
answered Apr 15, 2009 at 13:49 Mats Fredriksson 18.9k ●6 ●35 ●56

3 What if I just make a const method but without a normal method, and then I call the method using a non-const object, my code usually runs fine. Is it wrong or harmful or something? – KhiemGOM Aug 27, 2021 at 1:55

2 @KhiemGOM That's completely fine, and is pretty normal pattern for read only members. – Mats Fredriksson Nov 15, 2021 at 19:11

Add a comment

201

The const means that the method promises not to alter any members of the class. You'd be able to execute the object's members that are so marked, even if the object itself were marked `const`:

```
const foobar fb;
fb.foo();
```

would be legal.

See [How many and which are the uses of "const" in C++?](#) for more information.

Share Edit Follow Flag
edited Apr 15, 2009 at 13:42 gnud 75.3k ●5 ●62 ●77
answered Apr 15, 2009 at 13:29 Blair Conrad 216k ●25 ●128 ●110

Add a comment

51

The `const` qualifier means that the methods can be called on any value of `foobar`. The difference comes when you consider calling a non-const method on a const object. Consider if your `foobar` type had the following extra method declaration:

```
class foobar {
  ...
  const char* bar();
}
```

The method `bar()` is non-const and can only be accessed from non-const values.

```
void func1(const foobar& fb1, foobar& fb2) {
  const char* v1 = fb1.bar();  // won't compile
  const char* v2 = fb2.bar();  // works
}
```

The idea behind `const` though is to mark methods which will not alter the internal state of the class. This is a powerful concept but is not actually enforceable in C++. It's more of a promise than a guarantee. And one that is often broken and easily broken.

```
foobar& fbNonConst = const_cast<foobar&>(fb1);
```

Share Edit Follow Flag
edited Dec 18, 2016 at 14:11 GntS 607 ●1 ●11 ●32
answered Apr 15, 2009 at 13:30 JaredPar 697k ●141 ●1206 ●1435

3 I thought the answer is about other const methods and not about const objects. – Mykola Golubyev Apr 15, 2009 at 14:14

Thanks for the "The idea behind `const` though is to mark methods which will not alter the internal state of the class". That's really what I was looking for. – kovac Mar 17, 2019 at 9:01

1 @JaredPar does this mean that any member function representing a read-only operation should be marked as `const`? – kovac Mar 17, 2019 at 9:03

Add a comment

30

These const mean that compiler will Error if the method 'with const' changes internal data.

```
class A
{
public:
    A():member_()
    {
    }

    int hashGetter() const
    {
        state_ = 1;
        return member_;
    }
    int goodGetter() const
    {
        return member_;
    }
    int getter() const
    {
        //member_ = 2; // error
        return member_;
    }
    int badGetter()
    {
        return member_;
    }
private:
    mutable int state_;
    int member_;
};
```

The test

```
int main()
{
    const A a1;
    a1.badGetter(); // doesn't work
    a1.goodGetter(); // works
    a1.hashGetter(); // works

    A a2;
    a2.badGetter(); // works
    a2.goodGetter(); // works
    a2.hashGetter(); // works
}
```

Read [this](#) for more information

Share Edit Follow Flag
edited Apr 15, 2009 at 13:39
answered Apr 15, 2009 at 13:30 Mykola Golubyev 54.6k ●14 ●84 ●101

2 A question on `const` member functions that doesn't mention [mutable](#) is incomplete at best. – Ilnspectable Dec 20, 2015 at 4:21

Add a comment

13

Blair's answer is on the mark.

However note that there is a `mutable` qualifier which may be added to a class's data members. Any member so marked *can* be modified in a `const` method without violating the `const` contract.

You might want to use this (for example) if you want an object to remember how many times a particular method is called, whilst not affecting the "logical" constness of that method.

Share Edit Follow Flag
answered Apr 15, 2009 at 13:37 Alnitak 323k ●70 ●394 ●480

Add a comment

12

Meaning of a Const Member Function in [C++ Common Knowledge: Essential Intermediate Programming](#) gives a clear explanation:

> The type of the this pointer in a non-const member function of a class X is X * const. That is, it's a constant pointer to a non-constant X (see Const Pointers and Pointers to Const [7, 21]). Because the object to which this refers is not const, it can be modified. The type of this in a const member function of a class X is const X * const. That is, it's a constant pointer to a constant X. Because the object to which this refers is const, it cannot be modified. That's the difference between const and non-const member functions.

So in your code:

```
class foobar
{
  public:
     operator int () const;
     const char* foo() const;
};
```

You can think it as this:

```
class foobar
{
  public:
     operator int (const foobar * const this) const;
     const char* foo(const foobar * const this) const;
};
```

Share Edit Follow Flag
edited Jun 4, 2018 at 3:04
answered Mar 14, 2017 at 5:48 Nan Xiao 15.4k ●15 ●80 ●145

`this` is not `const`. The reason why it can't be modified is that it's a prvalue. – Brian Bi Apr 12, 2019 at 16:57

Add a comment

8

I would like to add the following point.

You can also make it a `const &` *and* `const &&`

So,

```
struct s{
    void val1() const {
     // *this is const here. Hence this function cannot modify any member of *thi
    }
    void val2() const & {
    // *this is const& here
    }
    void val3() const && {
    // The object calling this function should be const rvalue only.
    }
    void val4() && {
    // The object calling this function should be rvalue reference only.
    }

};

int main(){
  s a;
  a.val1(); //okay
  a.val2(); //okay
  // a.val3() not okay, a is not rvalue will be okay if called like
  std::move(a).val3(); // okay, move makes it a rvalue
}
```

Feel free to improve the answer. I am no expert

Share Edit Follow Flag
edited Mar 26, 2019 at 21:24
answered Mar 26, 2019 at 17:24 coder3101 3,430 ●2 ●21 ●25

1 `*this` is always an lvalue, even if the member function is rvalue-ref-qualified and is called on an rvalue. [Example](#). – HolyBlackCat Mar 26, 2019 at 17:30

Updated. Is that okay? – coder3101 Mar 26, 2019 at 18:05

Add a comment

7

when you use `const` in the method signature (like your said: `const char* foo() const;`) you are telling the compiler that memory pointed to by `this` can't be changed by this method (which is `foo` here).

Share Edit Follow Flag
edited May 30, 2015 at 6:09
answered May 30, 2015 at 5:54 Matrix Buster 269 ●2 ●9

Add a comment

3

Here const means that at that function any variable's value can not change

```
class Test{
private:
    int a;
public:
    void test()const{
        a = 10;
    }
};
```

And like this example, if you try to change the value of a variable in the test function you will get an error.

Share Edit Follow Flag
answered Dec 20, 2020 at 17:07 aboloo 89 ●4

This answer adds nothing to the top rated answer. – ExaltedBagel Jun 24, 2021 at 12:21

Add a comment

2

https://isocpp.org/wiki/faq/const-correctness#const-member-fns

> ## What is a "`const` member function"?
>
> A member function that inspects (rather than mutates) its object.
>
> A `const` member function is indicated by a `const` suffix just after the member function's parameter list. Member functions with a `const` suffix are called "const member functions" or "inspectors." Member functions without a `const` suffix are called "non-const member functions" or "mutators."
>
> ```
> class Fred {
> public:
>   void inspect() const;   // This member promises NOT to change *this
>   void mutate();          // This member function might change *this
> };
> void userCode(Fred& changeable, const Fred& unchangeable)
> {
>   changeable.inspect();   // Okay: doesn't change a changeable object
>   changeable.mutate();    // Okay: changes a changeable object
>   unchangeable.inspect(); // Okay: doesn't change an unchangeable object
>   unchangeable.mutate();  // ERROR: attempt to change unchangeable object
> }
> ```
>
> The attempt to call `unchangeable.mutate()` is an error caught at compile time. There is no runtime space or speed penalty for `const`, and you don't need to write test-cases to check it at runtime.
>
> The trailing `const` on `inspect()` member function should be used to mean the method won't change the object's *abstract* (client-visible) state. That is slightly different from saying the method won't change the "raw bits" of the object's struct. C++ compilers aren't allowed to take the "bitwise" interpretation unless they can solve the aliasing problem, which normally can't be solved (i.e., a non-const alias could exist which could modify the state of the object). Another (important) insight from this aliasing issue: pointing at an object with a pointer-to-const doesn't guarantee that the object won't change; it merely promises that the object won't change *via that pointer*.

Share Edit Follow Flag
edited Jun 20, 2020 at 9:12 Community Bot 1 ●1
answered Mar 20, 2020 at 19:44 qwr 7,862 ●5 ●50 ●82

Add a comment

2

The **const** keyword used with the function declaration specifies that it is a **const member function** and it will **not be able to change** the data members of the object.

Share Edit Follow Flag
answered Aug 18, 2019 at 9:05 Chandra Shekhar 553 ●1 ●6 ●23

Add a comment

0

In `const` objects only `const` methods can be called. All fielnd in such a method considered as `const` field. Last issue has curious effect:

- pointer becomes a const pointer `int* const`, which is not the same as a pointer to const `const int*`. Thus you can alter the object which pointer points to, but can't make pointer to point to another object.
- reference should become a const reference, but it is invariantly a const reference: you can not re-init it to another object. But again you can alter the object which reference refers to.

Share Edit Follow Flag
edited Dec 14, 2021 at 15:48
answered Dec 14, 2021 at 15:14 Max Popov 326 ●2 ●9

Add a comment

Linked

678 What is meant with "const" at end of function declaration?
83 What is the meaning of a const at end of a member function?
40 What does const mean following a function/method signature?
27 Using 'const' in class's functions
11 const type qualifier soon after the function name
6 why put a "const" at the end?
2 Const at the end of function declaration in C++
2 C++ Calling methods with same signature but different scope
3 what does "const" mean when overloading operator in C++
3 const parameter in function prototype
See more linked questions

Related

1568 What is the difference between const and readonly in C#?
1013 How to convert a std::string to const char* or char*
653 When can I use a forward declaration?
1502 'Static readonly' vs. 'const'
1626 What is the difference between const int*, const int * const, and int const *?
973 What is the difference between a definition and a declaration?
476 What is the meaning of prepended double colon "::"?
747 Difference between `constexpr` and `const`
1556 Replacing a 32-bit loop counter with 64-bit introduces crazy performance deviations with _mm_popcnt_u64 on Intel CPUs

Hot Network Questions