about const member function [duplicate]

This question already has an answer here:
What is the meaning of a const at end of a member function? *3 answers*

I met two explanation of const member function

```
class A{
  public:
  ...
  void f() const {}
  ...
}
```

1. it means it could only access constant members;
2. it means it does not modify any members;

I think the second one is right. But why does the first one come out? Is there anything to be clarify?

Thanks!

c++ const const-correctness

edited Dec 27 '09 at 18:37 James Thompson 19.7k 8 47 75

asked Dec 27 '09 at 16:29 skydoor 6,995 19 90 148

marked as duplicate by Praetorian c++ Jun 24 '14 at 4:21

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

1 What does your C++ book say? – Alok Singhal Dec 27 '09 at 16:42

4 This site **is** my C++ book! – James Thompson Dec 27 '09 at 17:42

They are both correct. To a degree (see below). – Loki Astari Dec 27 '09 at 19:36

Another dupe: stackoverflow.com/questions/751681/… – Praetorian Jun 24 '14 at 4:22

4 Answers

You can examine all class member values in a const member function, and in some cases you can even change the value of member variables. The first explanation is incorrect, I don't know where it comes from. The second explanation is correct, but with a few exceptions.

There are some exceptions to this rule. You can also change mutable variables in a const member function, for example a member variable declared like this:

```
mutable float my_rank;
```

You can also break const-correctness in a class by const_cast'ing a reference to yourself like this:

```
Class* self = const_cast<Class*> (this);
```

While technically allowed in C++, this is usually considered poor form because it throws away all of the const modifiers of your design. Don't do this unless you actually have to, and if you find yourself having to do this quite a lot that suggests a problem with your design. The C++ FAQ covers this very well.

Here are two references in case you want to do more reading:

- Const-correctness (cprogramming.com)
- Const correctness (C++ FAQ Lite)

edited Dec 27 '09 at 17:41

answered Dec 27 '09 at 16:30

James Thompson
19.7k 8 47 75

They are both correct.

A const member function can not alter the **state** of the object.

- This means it can read (but not modify) all member variables.
- This also means it can only call other const member functions
  Other methods that guarantee not to change the state of the object.

Above James also mentions mutable members.
So I should also cover those here.

A mutable member variable is a variable that is not part of the state of the object (the compiler does not consider it part of the objects state). You should also treat it this way. Any member variable that holds state information about the object should **NOT** be marked mutable. You should only use it to hold temporary information that can be re-constructed from the objects state.

A simple example is a date-time object. Where the object has a method that converts the data/time into a readable string format. This string may be cached in the object in a mutable member for efficiency (so that you don't need to repeatedly build the string). But the string is not part of the object state (because it can be built from other members).

Also James mentions above casting away constness using **const_cast**.

Except under very special situations where you know the object **CAN NEVER BE** const doing this is considered universally a bad idea. As it leads directly to undefined behavior. If you find yourself needing to cast away constness then something very wrong in the design has happened in your program.

In fact I can only think of one situation where it happens normally. And then I am unwilling to commit it to code without first going and doing research to make sure I don't look silly.

edited Dec 27 '09 at 19:48

answered Dec 27 '09 at 19:43

Loki Astari
139k 38 198 366

I think that you read the first question differently than me. The proposed definition was "it could only access constant members." A const member function can access non-constant members so long as it doesn't change modify their state, so that definition is incorrect. – James Thompson Dec 28 '09 at 1:49

@James. Please make your statement clearer. A const method can read any member variable. But can only call const member methods. I think it is clear that the original question had just got minor details mixed up. – Loki Astari Dec 28 '09 at 4:19

No worries, although 2 concrete examples were given.. Will take out all the comments and start supporting the media channels demonstrating trivial observation. Who knows, perhaps I will start seeing a dedicated lane for each car, instead of a single road. – rama-jka toti Dec 29 '09 at 11:34

@rama: What are you talking about? – Loki Astari Dec 29 '09 at 15:56

In a simple meaning , in const function you can't change the state of the object.

In const function this pointer behaves as **const pointer to const data** , where as in **non-const function** it behaves like **const pointer to data**.

```
void foo() const  --> const ClassName * const this (so you can't alter data)

void foo() --> ClassName * const this (so you can alter data)
```

As far as const data member is concern , you can access (read) it from any member function whether its const or not.

As James Thompson has shown you can even change state of object by removing constness if you want like this.

```
class Bar
{
    int bar;
    public:
     void foo() const
     {
        this->bar = 0; //flashes error

        Bar * const thisClass = const_cast<Bar * const>(this);
        thisClass->bar = 0;
     }
};
```

Also Mutable data members can be changed in const function.

edited Dec 15 '13 at 11:59

Benjamin
9,652 14 77 147

answered Dec 27 '09 at 17:29

Ashish
2,891 4 28 63

I think the case 1 after some clarification may concern the situation when you have a const object of type A. In such a case you can only call its member functions declared as const like f() in this case. So according to your post you must assume that 'it' is the caller of the member functions on an object of type const A. Maybe you should review the definition you found having in mind this assumption.

answered Dec 27 '09 at 17:41

jszpilewski
930 1 9 11
