const char * as a function parameter in C++

Asked 12 years, 6 months ago Modified 12 years, 6 months ago Viewed 34k times

8

6

NOTE: I know there are many questions that talked about that but I'm still a beginner and I couldn't understand the examples.

I got a function prototype that goes like this:

```
int someFunction(const char * sm);
```

Here, as you know, const char* means that this function can accept const or non-const pointer-to-char. I tried something like that in the function body:

```
someMemberVar = sm;
```

someMemberVar is just a pointer-to-char. The compiler gives me an error telling me: cannot convert from const char* to char*.

Here, I didn't pass a constant, so either sm or someMemberVar aren't constants. So, what constant the compiler is talking about?

c++ constants

Share Edit Follow Flag

edited Oct 30, 2009 at 20:37

asked Oct 30, 2009 at 20:30
Loai Abdelhalim
1,840 ● 4 ● 24 ● 28

1 How exactly is `someMemberVar` declared, and what is the exact error message you're getting (the one you provide doesn't make any sense, so I suspect there's a typo there). – Pavel Minaev Oct 30, 2009 at 20:35

I fixed my error. It's declared like that: char * someMemberVar; – Loai Abdelhalim Oct 30, 2009 at 20:37

1 @Alan he's mostly correct. char* can be safely cast to const char*, so you can pass a char* to that method just like you would a const char*. The compiler will just do the cast for you. – Herms Oct 30, 2009 at 21:05

1 @Alan: No, it can accept const char * or char *. It will treat either as if they were const char *. You need an explicit cast to change a const datatype to a non-const one, but not the other way. After all, just because a variable can be written to doesn't mean I have to change its value. – David Thornley Oct 30, 2009 at 21:08

While you need to understand what's happening here for other reasons, the real solution you likely want is to make `someMemberVar` a `std::string`. Especially when you're learning the basics of the language, variables which hold values (like int, string, etc.) are easier to use. After you've learned the language, they're still easier to use (because you /want/ to hold a value most of the time). – Roger Pate Oct 30, 2009 at 21:32

7 Answers

Sorted by: Highest score (default)

12

I'll try to put in simpler terms what others are saying:

The function `someFunction` takes a read-only string (for simplicity's sake, though `char *` could be used in umpteen other cases). Whether you pass in a readonly string to `someFunction` or not, the parameter is treated as read-only by the code executing in the context of this function. Within this function therefore, the compiler will try to prevent you from writing to this string as much as possible. A non-const pointer is such an attempt to disregard the read-only tag to the string and the compiler, rightly and loudly informs you of such disregard for its type system ;)

> What's the difference between: int someFunction(const char * sm) const{...} and this: int someFunction(const char * sm){...}

The first is a function which takes a readonly parameter. The second `const` written after the closing parentheses is valid only for member functions. It not only takes a read-only parameter, but also gurantees to not alter the state of the object. This is typically referred to as design level const.

Share Edit Follow Flag

edited Oct 30, 2009 at 21:34
T.E.D.
42.8k ● 9 ● 66 ● 134

answered Oct 30, 2009 at 20:42
dirkgently
105k ● 16 ● 128 ● 186

That's fine. What's the difference between: int someFunction(const char * sm) const{...} and this: int someFunction(const char * sm){...} – Loai Abdelhalim Oct 30, 2009 at 20:45

int someFunction(const char * sm) const{...} does const here mean that I won't alter any data here at all, or it refers to the parameter? – Loai Abdelhalim Oct 30, 2009 at 20:48

1 A simple way to describe "trailing `const`" is that it applies to `this`. In a non-const method of class `T`, the type of `this` is `T*` `const`. In a const method of the same class, the type of `this` is `T` `const*` `const`. – Pavel Minaev Oct 30, 2009 at 21:02

11

It is not entirely clear from your question, and I suspect the text of the error that you give is actually wrong, and actually reads:

> cannot convert from `const char*` to `char*`

Since you say that

> someMemberVar is just a pointer-to-char.

This makes sense. Keep in mind that `const char*` is actually the same as `char const*` - that is, it is a pointer to a const char, not a const pointer to char! And you cannot convert a pointer to `T const` to a pointer to `T`, because that breaks type safety. Consider:

```
const char* a = "abc";
char* b = a; // what you're trying to do
b[0] = 'x';  // if you could do it, you could change const data without casts
```

Share Edit Follow Flag

answered Oct 30, 2009 at 20:35
Pavel Minaev
97.4k ● 25 ● 216 ● 285

5

`const char*` is a pointer to constant char:

```
const char* ptr0; // ptr0 is mutable, *ptr0 is const
char* const ptr1; // ptr1 is const, *ptr1 is mutable
```

Share Edit Follow Flag

answered Oct 30, 2009 at 20:36
Nikolai Fetissov
79.7k ● 11 ● 109 ● 168

1

In a comment from one of the other answers you said:

> const char * sm mean that I can pass a const or non-const, so why C++ converts it automatically? That doesn't make sense to me.

Between your original question and that comment I think you're misunderstanding how the compiler treats the types.

You're correct that a non-const `char *` can be cast safely to a `const char *`. However, your method is explicitly taking a `const char *`. So, while you can pass a `char *` into the function, the compiler simply automatically casts the argument when making the function call. The actual code in the function doesn't know if the original argument was const or not. The compiler has to go by the actual declaration of the variable you're using, *not* some previous variable that had the same value.

If you want to be able to treat non-const `char *` variables differently (assigning them) then you'll need to define a function that takes non-const `char *` variables.

Share Edit Follow Flag

answered Oct 30, 2009 at 21:02
Herms
35.8k ● 12 ● 76 ● 101

1

If you pass a non-const pointer to `someFunction`, it is automatically converted to a const pointer. So you can't assign `sm` to `someMemberVar` because that would violate the constness of sm. You could declare `someMemberVar` as a `const char*` and your code would work, however, you could not modify what it pointed to.

Share Edit Follow Flag

answered Oct 30, 2009 at 20:35
rlbond
62.6k ● 54 ● 170 ● 222

const char * sm mean that I can pass a const or non-const, so why C++ converts it automatically? That doesn't make sense to me. – Loai Abdelhalim Oct 30, 2009 at 20:39

1 "const char* sm" means "pointer to const char". Nothing else. That is what sm will be in someFunction. You are able to pass a "char*" to someFunction because converting from non-const to const is allowed. Removing constness is not allowed, which is what you are then trying to do when you assign sm to someMemberVar. – jamessan Oct 30, 2009 at 20:47

1 Inside the function the pointer gets converted because you want it to be const inside that function - you declared it that way. It's part of the contract that the function makes with it's callers. If the function does not want to guarantee that the memory pointed by sm does not change then don't make the parameter const char*. – Catalin Iacob Oct 30, 2009 at 20:49

0

In your example sm is const char* so someFunction has a contract with it's caller that it will not modify the memory that sm points to.

But if you assign sm to someMemberVar and someMemberVar is not const char* you will then be able to modify the memory that sm points to via someMemberVar and the compiler doesn't allow you to do that.

Share Edit Follow Flag

answered Oct 30, 2009 at 20:38
Catalin Iacob
644 ● 5 ● 18

0

const char * sm is a pointer to a constant character (or array). When you try to assign, *someMemberVar*, a pointer-to-char, you are trying to point it to a set of **constant** characters. This is the cause for error.

Share Edit Follow Flag

answered Oct 30, 2009 at 20:36
user59634