How can we call "delete this; " in a const-member function?

Ask Question

Asked 11 years ago Active 6 years, 7 months ago Viewed 1k times

6

I saw the code snippet as follows:

```
class UPNumber {
public:
  UPNumber();
  UPNumber(int initValue);
  ...

  // pseudo-destructor (a const member function, because
  // even const objects may be destroyed)
  void destroy() const { delete this; } // why this line is correct???

  ...

private:
  ~UPNumber();
};
```

First, I am sure that above class definition is correct. Here is my question, why we can define the function 'destroy' as above? The reason being asking is that why we can modify 'this' in a const-member function?

c++ delete-operator const-correctness

Share Edit Follow Flag

edited Jul 20, 2015 at 20:22 Brian Tompsett - 汤莱恩 5,420 •68 •54 •126

asked Feb 28, 2011 at 23:08 q0987 32.7k •66 •231 •368

Add a comment

Start a bounty

4 Answers

Active Oldest Votes

12

That works for the same reason that this work:

```
const int* upn = new int();
delete upn;
```

You can `delete` a pointer to a const-qualified object. The const-qualification on the member function just means that `this` has a type `const UPNumber*`.

Share Edit Follow Flag

edited Feb 28, 2011 at 23:37

answered Feb 28, 2011 at 23:09 James McNellis 336k •73 •896 •965

@James, here, we cannot call "delete upn" b/c the destructor is private. -- thx – q0987 Feb 28, 2011 at 23:34

@q0987: Class members have access to private members. The function `destroy` requires access to the destructor, because of the `delete this` expression, and has access because it's a member of the class. – GManNickG Feb 28, 2011 at 23:51

1 @GMan: I think the OP was complaining about my example, which at first used his `UPNumber` class instead of `int`. – James McNellis Feb 28, 2011 at 23:54

@James: When reading your code I thought you were trying to make a reference to the old UPN network. :) – Kredns Mar 9, 2011 at 1:19

Add a comment

6

The `const` qualifier applied to a method have the effect of making the `this` passed to it a `const` pointer; in particular, in your case it will be a `const UPNumber *`.

Still, this is not a problem for `delete`: actually you can use `delete` on a `const` pointer without having to cast anything, as specified at §5.3.5 ¶2:

> [Note: a pointer to a const type can be the operand of a delete-expression; it is not necessary to cast away the constness (5.2.11) of the pointer expression before it is used as the operand of the delete-expression.]

Notice that, before the standard was completed, there have been many discussion about whether this was or wasn't a good idea, so some pre-standard compilers will issue an error when trying to `delete` `const` pointers.

The idea behind allowing this behavior is that otherwise you would have no way to `delete` `const` objects without using a `const_cast`; see this question for more info.

Share Edit Follow Flag

edited May 23, 2017 at 11:44 Community Bot 1 •1

answered Feb 28, 2011 at 23:12 Matteo Italia 119k •17 •195 •289

Add a comment

2

Where did you get the idea that we are modifying `this`? In fact, `this` is not an lvalue. It It cannot ever be modified, regardless of whether the member function is `const` or not.

Applying *delete-expression* to a pointer (any pointer, not just `this`) is not in any way considered a modification of that pointer. Moreover, the argument of delete-expression is treated as rvalue, meaning that it cannot possibly be modified by `delete`.

So, there are no problems with that application of `delete` in your code.

Share Edit Follow Flag

answered Feb 28, 2011 at 23:22 AnT 300k •39 •505 •747

if I understood your point right, the const-member function ONLY promises NOT change the values pointed by this. --thx – q0987 Feb 28, 2011 at 23:42

4 I think this is more than needs to be said about the fact that the questioner has mistakenly written "why we can modify 'this' in a const-member function?", despite meaning something more like, "why can we modify the object *through* `this`". The unspoken assumption being that deleting an object is a form of modification (and really it is, since the destructor is free to modify members on its way out). – Steve Jessop Feb 28, 2011 at 23:55

Add a comment

0

As it is **technically possible and defined** to call `delete this` provided that you are careful, the constness of the "suicide" method is **technically pointless**, since the object must not be touched after the call.

That's why it is **semantically incorrect** to have a `const` method calling `delete this`.

An interesting point brought Steve Jessep's comment. Running the destructor doesn't keep an object unchanged, so applying the concept of *const*ness is questionable.

Share Edit Follow Flag

edited May 23, 2017 at 12:29 Community Bot 1 •1

answered Jan 6, 2015 at 14:46 Wolf 8,997 •7 •56 •98

Add a comment

Your Answer

Community wiki

Post Your Answer

Not the answer you're looking for? Browse other questions tagged c++ delete-operator const-correctness or ask your own question.

Linked

99 Deleting a pointer to const (T const*)

Related

1054 Can I call a constructor from another constructor (do constructor chaining) in C++?
1013 How to convert a std::string to const char* or char*
702 How to call a parent class function from derived class function?
2038 How can I profile C++ code running on Linux?
817 Meaning of 'const' last in a function declaration of a class?
7 How to deal with initialization of non-const reference member in const object?
15 is it good practice to add const at end of member functions - where appropriate?
1 Is the "this" pointer always const?
76 Member function with static linkage