C++ Constructors have no return type. Just exactly why?

I've Googled this and read many posts, but there are so many different answers that all make logical sense that I was wondering if an expert on the topic could demystify this question.

Some say that there is no return because there is no way to return - the syntax prohibits it - yes, this makes sense, but I believe that all functions have to return something, no? Others say that the constructor sort of returns the newly created object itself, which seems to make sense since the assignment operator is used on the constructor. Still others have other interesting explanations.

c++ | compiler-construction | constructor | return

asked Apr 27 '12 at 18:42

wonton
707 1 8 23

Out of curiosity, when would it be useful or even make sense to have the constructor return anything other than the object it is constructing? – kevin628 Apr 27 '12 at 18:45

AFAIK it returns the newly created object. – Tamer Shlash Apr 27 '12 at 18:45

4 Who says a constructor is a function? (Aside from the fact that it's described in the Standard section on "Special Member Functions") – Ben Voigt Apr 27 '12 at 18:50

add a comment

8 Answers

Constructors aren't called like other functions, so they don't return like other functions. They execute as a side-effect of certain constructs (cast, `new` , variable definition, *ctor-initializer-list*, pass-by-value, return-by-value).

answered Apr 27 '12 at 18:49

Ben Voigt
165k 14 174 336

add a comment

The constructor does not specify a return type because it would be redundant: there is no type other than the one being constructed that a constructor could potentially "return". I put "return" in quotes because technically constructors do not return anything: when they are invoked in a static context, they initialize an instance in place; when they are invoked in a dynamic context, it is the operator `new` that returns something, not a constructor.

edited Apr 27 '12 at 21:11

andand
7,944 7 20 52

answered Apr 27 '12 at 18:48

dasblinkenlight
303k 26 258 497

2 and the thing that *looks* like a constructor call is actually a cast. – Ben Voigt Apr 27 '12 at 18:53

2 @downvoter please explain your vote – dasblinkenlight Apr 28 '12 at 12:50

add a comment

The constructor constructs in-place, it has no need to return anything.

I believe that all functions have to return something

`void` ?

answered Apr 27 '12 at 18:44

Puppy
94.2k 8 116 264

Constructors don't even return `void` (if that makes sense). They have no return type. – David Hammen Apr 27 '12 at 18:49

1 @DavidHammen That's only a difference in the syntax. Declaring a function's return type to be `void` has the same meaning. – bames53 Apr 27 '12 at 18:55

Is it only a difference in syntax? A constructor is a void function in the sense that a return statement must be `return;` in a constructor. However, by making the constructor uncallable and by leaving the return type unspecified, the standard appears to allow a compiler vendor to have a constructor return something so long as it acts as-if it is returning void. – David Hammen Apr 27 '12 at 19:06

@DavidHammen The fact that constructors are uncallable and that they're specified not to return anything isn't what gives implementations that latitude. The as-if rule does that. Implementations can cause callable, value returning functions to 'return' other things so long as they appear to behave according to the standard. – bames53 Apr 28 '12 at 17:42

add a comment

Assuming constructors could return something, then this has problematic implications for the following function call:

```
class Object{
    public:
        bool Object(){ ... };
        ...
}

SomeFun(Object obj){ ... }

SomeFun(Object());
// Ha! SomeFun jokes on an error about a non-Object argument(of type bool), returned
// by the anonymous temporary Object()
```

Preventing constructor functions from returning, facilitates the use of anonymous temporaries, along with many more C++ features. Without such a rule, innocuous statements can become ambiguous:

```
Object obj(Object());
// How does the variable obj decide which constructor to call,
// based on its argument type?
//
// Does it call: bool Object::Object(Object&) or
//                   Object::Object(bool)(if it exists)?
```

The ability for constructors to return a value, complicates the creation of objects - having a single unambiguous type, the class name, in the absence of arbitrary return values, avoids such problems.

Can readers suggest further examples where C++ idioms are hindered by the absence of such a rule?

edited Apr 27 '12 at 20:43

answered Apr 27 '12 at 20:08

damienh
139 2 7

this argument would have us believe that a constructor returning arbitrary values would be desirable if it did not rub other established language features the wrong way... to the contrary, a constructor is special purpose function used to initialize objects of its class... revisit your answer in that light... – user1055604 Apr 28 '12 at 6:25

add a comment

---

but I believe that all functions have to return something, no?

No. What does it mean to return a value from a function? Well, the ABI for an implementation will specify that for some return type a function will set certain registers, or some memory at some offset from the stack pointer, to the value that is 'returned' by the function.

Obviously registers and memory exist and contain data after a constructor or void function is called, but the language says those functions don't return anything, and so the ABI doesn't specify which registers or memory to look in to find a return value. The code can't set a return value and calling code can get any return value.

So no, functions don't have to return anything.

edited Apr 27 '12 at 18:57

answered Apr 27 '12 at 18:50

bames53
42.9k 4 57 109

add a comment

~~That constructors lack a return type — not even `void` — means that you can't call a constructor from your C++ code. And that's the point. Calling a constructor doesn't make sense. Making it illegal to call a constructor eliminates the very possibility of this type of user error.~~

Edit

barnes is right in that the ultimate reason you can't call constructors is that they have no name. (And even this ignores that you can call a constructor indirectly via placement new.)

Trying again:

You can't call constructors from your C++ code because constructors don't have a name. Making constructors not have a return type emphasizes to the programmer that constructors are a very different kind of function than other functions. What a constructor actually does return, if it returns anything, is up to the vendor.

edited Apr 27 '12 at 18:59

answered Apr 27 '12 at 18:47

David Hammen
18.4k 4 16 48

The fact that constructors don't return anything has nothing to do with whether or not you can call a constructor. You can't call a constructor because constructors don't have a name, and there's no syntax for calling it. (Only a workaround; placement new). – bames53 Apr 27 '12 at 18:49

add a comment

Constructors implicitly return an instance of the class itself. It will be contradicting if it were designed to return something and the programmer returned something completely different other than the class itself. Basically the syntax would be confusing.

answered Apr 27 '12 at 21:29

fastcodejava
14.2k 12 73 118

add a comment

Constructors are perhaps a bad name for it. It is practically an initializer for pre-constructed objects. Easiest way to illustrate this is a pseudo-example for the equivalent construct without using the C++ constructors.

With constructors

```
struct X {
    int a;
    X() {
      a = -5;
    }
};

int main() {
    X* x1 = new X(); // X created as a "reference object".
    X  x2;           // X created as a "value object" on the stack.
    return x1->a - x2.a;
}
```

Without constructors

```
struct X {
    int a;
    void initialize() {
        a = -5;
    }
    static X* create() {
        X* res = (X*)malloc(sizeof(X));
        res->initialize();
        return res;
    }
};

int main() {
    X* x1 = X::create(); // New constructed heap-allocated X
    X  x2;               // Stack-allocated X
    x2.initialize();     // Manually initialized
    return x1->a - x2.a;
}
```

Now, if you imagine that `X::initialize` in the second example were to be made to return, say a `bool` indicating success or failure, you would have a problem. In `main()`, the inattentive programmer might end up with an `X` that isn't properly initialized, leading to undefined behavior (usually a hard-to debug crash that might not be discovered before production).

This is one of the core reasons why constructors are made special. The only two ways to exit a constructor, is through normal completion, or through an exception, which *must* then be handled by the caller (or passed on up the stack). In any case, you prevent ending up with uninitialized objects.

As a side-note, uninitialized objects are one of the more common causes of bugs in C, which does not have anything to help the programmer like this.

edited Dec 18 '12 at 12:33

answered Apr 28 '12 at 6:57

Rawler

597 3 15

add a comment

Not the answer you're looking for? Browse other questions tagged c++ compiler-construction constructor return **or ask your own question.**