Default parameters with C++ constructors [closed]

Asked 16 years, 2 months ago   Modified 6 years, 10 months ago   Viewed 329k times

147

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 8 years ago.

Improve this question

Is it good practice to have a class constructor that uses default parameters, or should I use separate overloaded constructors? For example:

```
// Use this...
class foo
{
private:
    std::string name_;
    unsigned int age_;
public:
    foo(const std::string& name = "", const unsigned int age = 0) :
        name_(name),
        age_(age)
    {
        ...
    }
};

// Or this?
class foo
{
private:
    std::string name_;
    unsigned int age_;
public:
    foo() :
    name_(""),
    age_(0)
{
}

foo(const std::string& name, const unsigned int age) :
        name_(name),
        age_(age)
    {
        ...
    }
};
```

Either version seems to work, e.g.:

```
foo f1;
foo f2("Name", 30);
```

Which style do you prefer or recommend and why?

c++ constructor coding-style overloading

Share

Improve this question

Follow

edited Oct 9, 2008 at 15:02

Matt Price
45.3k ● 9 ● 39 ● 44

asked Oct 9, 2008 at 14:58

Rob
78.5k ● 57 ● 161 ● 199

1 I personally have made extensive use of it in many projects. As to whether it's "best", I can't speak, but it's certainly been extremely helpful to me in the past. – warren Oct 9, 2008 at 15:00

1 Not a big deal either way. However the one warning is that any constructor that only takes one argument should be explicit to avoid nasty implicit conversions. – Matt Price Oct 9, 2008 at 15:01

1 Both approaches are OK, as long as you don't mix them - otherwise you may get some nasty surprises. – Nemanja Trifunovic Oct 9, 2008 at 15:49

Does your 2nd option works ??? It seems to me it is wrong. You are overloading the constructor, so you should declare both functions foo() and foo(string, unsigned)... then you would have your 1st option. – Thomio Feb 6, 2017 at 12:37

3 C++ guidelines are clear on the subject: none of the two options, but second one is better. Make several overloaded constructors and prefer in-class initializers. – Bentoy13 May 23, 2018 at 8:43

12 Answers

Sorted by: Highest score (default)

104

Definitely a matter of style. I prefer constructors with default parameters, so long as the parameters make sense. Classes in the standard use them as well, which speaks in their favor.

One thing to watch out for is if you have defaults for all but one parameter, your class can be implicitly converted from that parameter type. Check out this thread for more info.

edited May 23, 2017 at 10:31

answered Oct 9, 2008 at 14:59

Share

Improve this answer

Follow

Community Bot
1 ● 1

luke
37.4k ● 8 ● 61 ● 81

49

I'd go with the default arguments, especially since C++ doesn't let you chain constructors (so you end up having to duplicate the initialiser list, and possibly more, for each overload).

That said, there are some gotchas with default arguments, including the fact that constants may be inlined (and thereby become part of your class' binary interface). Another to watch out for is that adding default arguments can turn an explicit multi-argument constructor into an implicit one-argument constructor:

```
class Vehicle {
public:
  Vehicle(int wheels, std::string name = "Mini");
};

Vehicle x = 5;  // this compiles just fine... did you really want it to?
```

Share

Improve this answer

Follow

edited Oct 9, 2008 at 23:12

answered Oct 9, 2008 at 15:07

Sam Stokes
14.8k ● 7 ● 38 ● 34

26 In that case, add the "explicit" keyword to the constructor. It should prevent `Vehicle x = 5;` from compiling. – Denilson Sá Maia Dec 28, 2011 at 21:34

14 And now, 5 years later, C++11 has added delegating constructors and default member initialization in the class definition, making the default parameter workaround unnecessary: en.wikipedia.org/wiki/C%2B%2B11#Object_construction_improvement – mskfisher Jun 4, 2013 at 18:01

20

This discussion apply both to constructors, but also methods and functions.

Using default parameters?

The good thing is that you won't need to overload constructors/methods/functions for each case:

```
// Header
void doSomething(int i = 25) ;

// Source
void doSomething(int i)
{
   // Do something with i
}
```

The bad thing is that you must declare your default in the header, so you have an hidden dependancy: Like when you change the code of an inlined function, if you change the default value in your header, you'll need to recompile all sources using this header to be sure they will use the new default.

If you don't, the sources will still use the old default value.

using overloaded constructors/methods/functions?

The good thing is that if your functions are not inlined, you then control the default value in the source by choosing how one function will behave. For example:

```
// Header
void doSomething() ;
void doSomething(int i) ;

// Source

void doSomething()
{
   doSomething(25) ;
}

void doSomething(int i)
{
   // Do something with i
}
```

The problem is that you have to maintain multiple constructors/methods/functions, and their forwardings.

Share
Improve this answer
Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ● 1

answered Oct 9, 2008 at 16:14
paercebal
83.2k ● 38 ● 134 ● 160

14

In my experience, default parameters seem cool at the time and make my laziness factor happy, but then down the road I'm using the class and I am surprised when the default kicks in. So I don't really think it's a good idea; better to have a className::className() and then a className::init(*arglist*). Just for that maintainability edge.

Share Improve this answer Follow

answered Oct 9, 2008 at 15:04
Paul Nathan

40.2k ● 30 ● 120 ● 215

8

Sam's answer gives the reason that default arguments are preferable for constructors rather than overloading. I just want to add that C++-0x will allow delegation from one constructor to another, thereby removing the need for defaults.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:26

Community Bot

1 ● 1

answered Oct 9, 2008 at 17:01

Richard Corden

21.7k ● 9 ● 61 ● 87

5

Either approach works. But if you have a long list of optional parameters make a default constructor and then have your set function return a reference to this. Then chain the settors.

```
class Thingy2
{
public:
    enum Color{red,gree,blue};
    Thingy2();

    Thingy2 & color(Color);
    Color color()const;

    Thingy2 & length(double);
    double length()const;
    Thingy2 & width(double);
    double width()const;
    Thingy2 & height(double);
    double height()const;

    Thingy2 & rotationX(double);
    double rotationX()const;
    Thingy2 & rotatationY(double);
    double rotatationY()const;
    Thingy2 & rotationZ(double);
    double rotationZ()const;
}

main()
{
    // gets default rotations
    Thingy2 * foo=new Thingy2().color(ret)
        .length(1).width(4).height(9)
    // gets default color and sizes
    Thingy2 * bar=new Thingy2()
        .rotationX(0.0).rotationY(PI),rotationZ(0.5*PI);
    // everything specified.
    Thingy2 * thing=new Thingy2().color(ret)
        .length(1).width(4).height(9)
        .rotationX(0.0).rotationY(PI),rotationZ(0.5*PI);
}
```

Now when constructing the objects you can pick an choose which properties to override and which ones you have set are explicitly named. Much more readable :)

Also, you no longer have to remember the order of the arguments to the constructor.

Share
Improve this answer
Follow

edited Oct 9, 2008 at 16:28

answered Oct 9, 2008 at 16:18
Rodney Schuler
2,148 ● 4 ● 23 ● 34

I like this ability to chain settors; whether at instantiation or for later updates. Have you ever tried accepting a struct and picking the attributes out of that into the object - or just using the struct as the data retainer within the object? – WillC Nov 16, 2018 at 16:17

4

One more thing to consider is whether or not the class could be used in an array:

```
foo bar[400];
```

In this scenario, there is no advantage to using the default parameter.

This would certainly NOT work:

```
foo bar("david", 34)[400]; // NOPE
```

Share Improve this answer Follow

answered Feb 8, 2010 at 5:16
peter
6,137 ● 2 ● 35 ● 45

3

Mostly personal choice. However, overload can do anything default parameter can do, but **not** vice versa.

Example:

You can use overload to write A(int x, foo& a) and A(int x), but you cannot use default parameter to write A(int x, foo& = null).

The general rule is to use whatever makes sense and makes the code more readable.

Share Improve this answer Follow

answered Oct 10, 2008 at 0:55
Yang Lu
56 ● 1

3

If creating constructors with arguments is bad (as many would argue), then making them with default arguments is even worse. I've recently started to come around to the opinion that ctor arguments are bad, because your ctor logic should *be as minimal as possible*. How do you deal with error handling in the ctor, should somebody pass in an argument that doesn't make any sense? You can either throw an exception, which is bad news unless all of your callers are prepared to wrap any "new" calls inside of try blocks, or setting some "is-initialized" member variable, which is kind of a dirty hack.

Therefore, the only way to make sure that the arguments passed into the initialization stage of your object is to set up a separate initialize() method where you can check the return code.

The use of default arguments is bad for two reasons; first of all, if you want to add another argument to the ctor, then you are stuck putting it at the beginning and changing the entire API. Furthermore, most programmers are accustomed to figuring out an API by the way that it's used in practice -- this is *especially* true for non-public API's used inside of an organization where formal documentation may not exist. When other programmers see that the majority of the calls don't contain any arguments, they will do the same, remaining blissfully unaware of the default behavior your default arguments impose on them.

Also, it's worth noting that the google C++ style guide shuns both ctor arguments (unless absolutely necessary), and default arguments to functions or methods.

Share
Improve this answer
Follow

edited Feb 6, 2018 at 14:31
Metaxal
1,103 ● 8 ● 10

answered Oct 9, 2008 at 15:57
Nik Reiman
40.3k ● 29 ● 107 ● 161

2

I would go with the default parameters, for this reason: Your example assumes that ctor parameters directly correspond to member variables. But what if that is not the case, and you have to process the parameters before the object is initialize. Having one common ctor would be the best way to go.

Share Improve this answer Follow

answered Oct 9, 2008 at 15:09
James Curran
103k ● 37 ● 185 ● 262

2

One thing bothering me with default parameters is that you can't specify the last parameters but use the default values for the first ones. For example, in your code, you can't create a Foo with no name but a given age (however, if I remember correctly, this will be possible in C++0x, with the unified constructing syntax). Sometimes, this makes sense, but it can also be really awkward.

In my opinion, there is no rule of thumb. Personnaly, I tend to use multiple overloaded constructors (or methods), except if only the last argument needs a default value.

Share Improve this answer Follow

answered Oct 9, 2008 at 15:10
Luc Touraille
81.9k ● 16 ● 99 ● 139

0

Matter of style, but as Matt said, definitely consider marking constructors with default arguments which would allow implicit conversion as 'explicit' to avoid unintended automatic conversion. It's not a requirement (and may not be preferable if you're making a wrapper class which you want to implicitly convert to), but it can prevent errors.

I personally like defaults when appropriate, because I dislike repeated code. YMMV.

Share Improve this answer Follow

answered Oct 9, 2008 at 16:18
Nick
6,846 ● 1 ● 24 ● 34