Does Overloading Operator<< works inside the class?

Asked 10 years ago Active 3 years, 11 months ago Viewed 17k times

15

I mean, I was trying to overload the operator<< inside the class

like this

```
class A {
     public:
          ostream &operator<<(ostream &os);// which doesnt work

     private:
          friend ostream &operator<<(ostream &os, const A& a); //Works
          int i;

};

 Definition
         ostream &operator<<(ostream &os, const A& a) {
             os<<a.i;
             return os;
         }
```

why can't I overload the operator inside the class specific to class? or Am I missing something? or Am I stupid to even think in such a way? Please advise.

c++ operator-overloading ostream

Share Edit Follow Flag

edited Feb 19, 2012 at 17:17

asked Feb 19, 2012 at 17:02
howtechstuffworks
1,674 ● 4 ● 25 ● 46

2 possible duplicate of Operator overloading (see especially this answer) – Ben Voigt Feb 19, 2012 at 17:06

3 Answers

Active | Oldest | Votes

32

The problem is that your `operator<<` would take `ostream` as a second parameter, not a first one. That way, you could do `myObject << std::cout`, but it would look unintuitive and you would not be able to chain calls due to `operator<<` being left-associative.

Another benefit of declaring the operator as a friend as opposed to a member function is that automatic conversions can occur. That means that if you have a class `B` that doesn't derive from `A` but does have a `B(A const&)` constructor, you will still be able to do `std::cout << my_b;` and have it converted to an `A` and then printed.

Fortunately, defining `operator<<` as a friend can be done within the class if you prefer to. Your code could be written as

```
class A {
    int i;
    friend std::ostream& operator<<(std::ostream& o, A const& a) {
        o << a.i;
        return o;
    }
};
```

Why doesn't the standard allow for you to specify which side the argument should go on? Let's pretend it did, and add the `left_of` and `right_of` keywords to specify:

```
struct B;

struct A {
     A left_of operator+(B const&) {
         return *this;
     }
};

struct B {
     B right_of operator+(A const&) {
         return *this;
     }
};
```

What happens now when we do `A a; B b; f(a + b);`? Each class has an operator that handles this case, which means we can't decide. Seeing as many operators should be friends due to the conversion possibilities anyway, not allowing this kind of thing is not that big a problem, and prevents these kind of ambiguities. (Of course, you could also define a member operator and a free one, which would cause a very similar problem.)

By the way, as things are, the definition of your `friend operator<<` isn't returning anything, which will mess up chaining.

Share Edit Follow Flag

edited Feb 19, 2012 at 17:16

answered Feb 19, 2012 at 17:03
Anton Golov
3,344 ● 1 ● 20 ● 40

^ got u..... But the first one doesn't make sense at all..... was it done, because to maintain consistency among all operators, where all the things appear right to the operator, is taken as a parameter? – howtechstuffworks Feb 19, 2012 at 17:10

^ thanks anton..... I understood lot better now..... Btw, I agree ur argument for + operator... I am not arguing, that we should have to specify which way it is to be done... I agree that all the operators by default take the right value, but what happens if we decide to do it the other way? default by left, for operators in need? like <<.... << operator always takes the left as argument and bind to the class specified in right..... Or was it overlooked, because of the two disadv, yout mentioned at the top of the comment...???? – howtechstuffworks Feb 19, 2012 at 17:22

@howtechstuffworks: While using `operator<<` for output is common, it is not the only use, and making some operators take the value to the right and the others the value to the left would be even more of a headache. – Anton Golov Feb 19, 2012 at 17:24

27

The member function:

```
ostream &operator<<(ostream &os);
```

does work, but not for the situation you want. It will be called when you do something like:

```
A a;
a << std::cout;
```

i.e. where the object is the left-hand side of the operator.

Share Edit Follow Flag

answered Feb 19, 2012 at 17:04
Oliver Charlesworth
259k ● 30 ● 545 ● 665

2

Remember that for a non static member function of a class first argument is this pointer. So the signature of your the "operator<<" function, after compilation will become:

```
ostream &operator<<(A *this, ostream &os);
```

This will work when you do something like:

```
A obj;
obj << std::cout;
```

Think of "obj" as first argument and "std::cout" as 2nd argument to the operator "<<".

Better way to overload "<<" is by using friend function because in this way you can make your overloaded operator associative.

Share Edit Follow Flag

answered Apr 3, 2018 at 12:08
Ankit Agrawal
51 ● 1 ● 5

Really Helpful and Insightful! – Savannah Madison May 14, 2021 at 11:02