C++ overload resolution [duplicate]

Asked 16 years, 3 months ago Modified 7 years, 1 month ago Viewed 14k times

42

This question already has answers here:

Function with same name but different signature in derived class not found (2 answers)

Closed 10 years ago.

Given the following example, why do I have to explicitly use the statement `b->A::DoSomething()` rather than just `b->DoSomething()`?

Shouldn't the compiler's overload resolution figure out which method I'm talking about?

I'm using Microsoft VS 2005. (Note: using virtual doesn't help in this case.)

```
class A
{
  public:
    int DoSomething() {return 0;};
};

class B : public A
{
  public:
    int DoSomething(int x) {return 1;};
};

int main()
{
  B* b = new B();
  b->A::DoSomething();    //Why this?
  //b->DoSomething();    //Why not this? (Gives compiler error.)
  delete b;
  return 0;
}
```

c++ function overloading resolution

Share

Improve this question

Follow

edited Jan 23, 2010 at 18:22

Peter Mortensen

31.6k ● 22 ● 109 ● 133

asked Sep 16, 2008 at 13:09

Abe

423 ● 1 ● 5 ● 7

I am trying to call DoSomething() in class A using a pointer to class B. – Abe Sep 16, 2008 at 13:22

9 Answers

Sorted by: Highest score (default)

44

The two “overloads” aren't in the same scope. By default, the compiler only considers the smallest possible name scope until it finds a name match. Argument matching is done *afterwards*. In your case this means that the compiler sees `B::DoSomething`. It then tries to match the argument list, which fails.

One solution would be to pull down the overload from `A` into `B`'s scope:

```
class B : public A {
public:
    using A::DoSomething;
    // …
}
```

Share Improve this answer Follow

answered Sep 16, 2008 at 13:18

Konrad Rudolph
545k ● 139 ● 956 ● 1.2k

1 In fact if they are not in the same scope, they are not called overloads as well – Chubsdad Nov 4, 2010 at 4:18

4 @Chubsdad: That’s why I put the word in quotes. And once the method is pulled into the same scope, it becomes an overload. – Konrad Rudolph Nov 4, 2010 at 9:48

14

Overload resolution is one of the ugliest parts of C++

Basically the compiler finds a name match "DoSomething(int)" in the scope of B, sees the parameters don't match, and stops with an error.

It can be overcome by using the A::DoSomething in class B

```
class A
{
  public:
    int DoSomething() {return 0;}
};

class B : public A
{
  public:
    using A::DoSomething;
    int DoSomething(int x) {return 1;}
```

```
};


int main(int argc, char** argv)
{
  B* b = new B();
  // b->A::DoSomething();    // still works, but...
  b->DoSomething();    // works now too
  delete b;
  return 0;
}
```

Share Improve this answer Follow

answered Sep 16, 2008 at 13:18

Pieter

17.7k ● 8 ● 53 ● 90

6

The presence of a method in a derived class hides all methods with the same name (regardless of parameters) in base classes. This is done to avoid problems like this:

```
class A {} ;
class B :public A
{
    void DoSomething(long) {...}
}

B b;
b.DoSomething(1);     // calls B::DoSomething((long)1));
```

than later someone changes class A:

```
class A
{
    void DoSomething(int ) {...}
}
```

now suddenly:

```
B b;
b.DoSomething(1);     // calls A::DoSomething(1);
```

In other words, if it didn't work like this, a unrelated change in a class you don't control (A), could silently affect how your code works.

Share

Improve this answer

Follow

edited Nov 21, 2017 at 16:03

answered Sep 16, 2008 at 17:21

James Curran

103k ● 37 ● 185 ● 262

Very nice point, I must say. Most answers I have seen talked about why error occurs and how to avoid it but your point shows why this feature is essential and useful in real world scenario!! – Pravar Jawalekar Nov 20, 2017 at 6:59

5

No, this behaviour is present to ensure that you don't get caught out inheriting from distant base classes by mistake.

To get around it, you need to tell the compiler which method you want to call by placing a using A::DoSomething in the B class.

See this article for a quick and easy overview of this behaviour.

Share Improve this answer Follow

answered Sep 16, 2008 at 13:27

Lehane

48.6k ● 14 ● 55 ● 54

3

This has something to do with the way name resolution works. Basically, we first find the scope from which the name comes, and then we collect all overloads for that name in that scope. However, the scope in your case is class B, and in class B, B::DoSomething **hides** A::DOSomething:

3.3.7 Name hiding [basic.scope.hiding]

...[snip]...

3 In a member function definition, the declaration of a local name hides the declaration of a member of the class with the same name; see *basic.scope.class*. The declaration of a member in a derived class (*class.derived*) hides the declaration of a member of a base class of the same name; see *class.member.lookup*.

Because of name hiding, A::DoSomething is not even considered for overload resolution

Share Improve this answer Follow

answered Sep 16, 2008 at 13:30

user3458

Having the actual specification reference is good. – Sam Jan 4, 2012 at 21:32

When you define a function in a derived class then it hides all the functions with that name in the base class. If the base class function is virtual and has a compatible

2

signature then the derived class function also overrides the base class function. However, that doesn't affect the visibility.

You can make the base class function visible with a using declaration:

```
class B : public A
{
  public:
    int DoSomething(int x) {return 1;};
    using A::DoSomething;
};
```

Share Improve this answer Follow

answered Sep 16, 2008 at 13:19

Anthony Williams

68.4k ● 15 ● 137 ● 160

That's not overloading! That's HIDING!

2

Share

Improve this answer

Follow

edited Jan 23, 2010 at 18:22

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 16, 2008 at 13:57

ugasoft

3,802 ● 7 ● 29 ● 23

When searching up the inheritance tree for the function to use, C++ uses the name without arguments, once it has found any definition it stops, then examines the arguments. In the example given, it stops in class B. In order to be able to do what you are after, class B should be defined like this:

1

```
class B : public A
{
  public:
    using A::DoSomething;
    int DoSomething(int x) {return 1;};
};
```

Share Improve this answer Follow

answered Sep 16, 2008 at 13:19

Jono

101 ● 5

The function is hidden by the function with the same name in the subclass (but with a different signature). You can unhide it by using the using statement, as in using

1

```
A::DoSomething();
```

Share Improve this answer Follow

answered Sep 16, 2008 at 13:20

slicedlime

2,152 ● 1 ● 17 ● 16