c++ - <unresolved overloaded function type>

In my class called `Mat`, I want to have a function which takes another function as a parameter. Right now I have the 4 functions below, but I get an error in when calling print(). The second line gives me an error, but I don't understand why, since the first one works. The only difference is function `f` is not a member of the class `Mat`, but `f2` is. The failure is: `error: no matching function for call to Mat::test( < unresolved overloaded function type>, int)'`

```
template <typename F>
int Mat::test(F f, int v){
    return f(v);
}

int Mat::f2(int x){
    return x*x;
}

int f(int x){
    return x*x;
}

void Mat::print(){
    printf("%d\n",test(f ,5));    // works
    printf("%d\n",test(f2 ,5));    // does not work
}
```

Why does this happen?

c++ function templates

edited Jan 31 at 19:18

FluxLemur
114 1 8

asked Apr 5 '13 at 18:49

Mordrag
81 1 1 4

1 Is `f2` static or not? – Joseph Mansfield Apr 5 '13 at 18:52

Try changing the printf call to printf("%d\n", test(Mat::f2, 5)); – 2to1mux Apr 5 '13 at 18:57

Do you have more than one overload of `f2` ? – Andy Prowl Apr 5 '13 at 18:58

1 @2to1mux. parashift.com/c++-faq-lite/addr-of-memfn.html. Who knew? – user995502 Apr 5 '13 at 19:22

1 @stardust_ thanks! That makes so much sense now! – 2to1mux Apr 5 '13 at 19:59

2 Answers

The type of `pointer-to-member-function` is different from `pointer-to-function`.

The type of a function is different depending on whether it is an ordinary function or a **non-static member function** of some class:

```
int f(int x);
the type is "int (*)(int)" // since it is an ordinary function
```

And

```
int Mat::f2(int x);
the type is "int (Mat::*)(int)" // since it is a non-static member function of
class Mat
```

Note: if it's a static member function of class Fred, its type is the same as if it were an ordinary function: `"int (*)(char,float)"`

> In C++, member functions have an implicit parameter which points to the object (the this pointer inside the member function). Normal C functions can be thought of as having a different calling convention from member functions, **so the types of their pointers (pointer-to-member-function vs pointer-to-function) are different and incompatible.** C++ introduces a new type of pointer, **called a pointer-to-member, which can be invoked only by providing an object**.
>
> NOTE: do not attempt to "cast" a pointer-to-member-function into a pointer-to-function; the result is undefined and probably disastrous. E.g., **a pointer-to-member-function is not**

More on this Here and here.

edited Apr 5 '13 at 19:59

answered Apr 5 '13 at 19:18
user995502

But this doesn't explain why the template can't resolve it. Or am I missing something? – Luchian Grigore Apr 5 '13 at 20:36

@LuchianGrigore This is how I understood it. The compiler expects a type that provides a `call operator` or sees if it can find a function that matches `int f(int)` reachable from Mat::Test. And according the the description above `pointer-to-member-function vs pointer-to-function are different and incompatible` . So it will not match `this->f2` as a function. This is just my understanding. I didn't even know about this before today :) – user995502 Apr 5 '13 at 20:50

Links seem to be dead. – derM Sep 20 '17 at 12:39

The problem here is that `f2` is a method on `Mat` , while `f` is just a free function. You can't call `f2` by itself, it needs an instance of `Mat` to call it on. The easiest way around this might be:

```
printf("%d\n", test([=](int v){return this->f2(v);}, 5));
```

The `=` there will capture `this` , which is what you need to call `f2` .

answered Apr 5 '13 at 19:01

Barry
149k 15 247 455

Since f2 and print are both member functions of Mat, isn't print allowed to call f2 without referencing a Mat object? – 2to1mux Apr 5 '13 at 19:04

Wow, this works (with a few tweaks from the original code). +1 from me – Luchian Grigore Apr 5 '13 at 19:04

-1 . the error message says that the error happens because of an overloaded function, and not because the function template's function call syntax is wrong (which is what you say). – Johannes Schaub - litb Apr 5 '13 at 19:06

@JohannesSchaub-litb That isn't what he is saying. He's saying that f2 can only be called via an instance of a Mat object. This solution solves the issue (Although, I actually think it solves the original compiler error for a different reason than the one stated by Barry). – 2to1mux Apr 5 '13 at 19:09

I think in this case the compiler error is not the best. `f2` just isn't a function, `&Mat::f2` is. If you tried to pass in that instead, you'd get a compile error about invalid call syntax. – Barry Apr 5 '13 at 19:10