

Discuss C++ Template Downcast

Table of Contents

- [Original Discuss](#)
- [The problem](#)
- [My answer to the problem](#)
 - [First we discuss ff](#)
 - [Then we discuss f](#)

This is a discuss in C board in bbs.sjtu.edu.cn, about type down-cast in C++ template.

[Original Discuss](#)

<http://bbs.sjtu.edu.cn/bbstcon,board,C,reid,1330078933,file,M.>

The problem

Today I read a book about we can do cast-down in template, so I write this to test:

```
1  template <bool _Test, class _Type = void>
2  struct enable_if {};
3
4  template<class _Type>
5  struct enable_if<true, _Type> {
6      typedef _Type type;
7  };
8
9  class A {};
10 class B : A {};
11
12 template <typename T>
13 struct traits { static int const value = false; };
14
15 template <>
16 struct traits<A> { static int const value = true; };
17
18 template <typename T>
19 void f(T, typename enable_if<traits<T>::value>::type* = 0) {}
20
21 template <>
22 void f<A>(A, enable_if<traits<A>::value>::type*) {}
23
24
25
26 template <typename T>
27 class BB {};
28
29 template <typename T>
```

```

30 class DD : public BB<T> {};
31
32 template <typename T> void ff(BB<T>) {};
33
34 int main(int argc, char * argv[])
35 {
36     A a; B b;
37     DD<long> dd;
38     //f(b);
39     ff(dd);
40 }

```

It is strange when f it don't allow my specified f<A>` .

But in ff it allowed ff<BB<long>>` .

Tested under VC10 and GCC3.4

My answer to the problem

Let's think ourself as compiler to see what happened there.

Define mark # : A#B is the instantiated result when we put B into the parameter T of A<T> .

First we discuss ff

```

1 DD<long> dd;

```

After this sentence, the compiler saw the instantiation of DD<long> , so it instantiate DD#long , and also BB#long .

```
1 ff(dd);
```

This sentence required the compiler to calculate set of overloading functions.

Step 1 we need to infer T of $ff<T>$ from argument $DD\#long \rightarrow BB<T>$. Based on the inference rule:

Argument with type :code:`class_template_name<T>` can be use to infer :code:`T` .

So compiler inferred T as $long$. Here if it is not BB but CC which is complete un-related, we can also infer, as long as CC is a template like $CC<T>$.

Step 2 Template Specialization Resolution. There is only one template here so we matched $ff<T>$.

Step 3 Template Instantiation

After inferred $long \rightarrow T$, compiler instantiated $ff\#long$.

Set of available overloading functions : $\{ff\#long\}$

Then overloading resolution found the only match $ff\#long$, checked its real parameter $DD\#long$ can be down-cast to formal parameter $BB\#long$.

Then we discuss f

```
1 f(b);
```

Calculate set of overloading functions.

Step 1 infer all template parameters for template f . According to inference rule:

Parameter with type T can be used to infer T 。

So $B \rightarrow T$ is inferred.

Step 2 Template Specialization Resolution.

Here B is not A so we can not apply specialization of $f<A>$, remaining $f<T>$ as the only alternative.

Step 3 Template Instantiation.

When we put B into $f<T>$ to instantiate as $f\#B$, we need to instantiate $traits\#B`$.

There is no specialization for B so we use template $traits<T>$, $traits\#B::value=false$, so $enable_if\#false$ didn't contains a type , an error occurred.

The only template is mismatch, available overloading functions is empty set. So we got an error.