Discuss C++ Template Downcast

Table of Contents

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.1330078933.A.html

The problem

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

```
template <bool _Test, class _Type = void>
struct enable_if { };

template<class _Type>
struct enable_if<true, _Type> {
    typedef _Type type;
};

class A { };
class B : A { };

template <typename T>
struct traits { static int const value = false; };

template <>
struct traits<A> { static int const value = true; };

template <typename T>
void f(T, typename enable_if<traits<T>::value>::type* = 0) { }

template <>
void f<A>(A, enable_if<traits<A>::value>::type*) { }



template <typename T>
class BB {};

template <typename T>
class DD : public BB<T> {};

template <typename T> void ff(BB<T>) {};

int main(int argc, char * argv[])
{
    A a; B b;
    DD<long> dd;
    //f(b);
    ff(dd);
}
```

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

```
DD<long> dd;
```

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

```
ff(dd);
```

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

Step 1 we need to infer `T` of `ff<T>` from argument `DD#long -> 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 -> 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

```
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 -> 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.