

Re: Class templates and friend function templates

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2005-07/msg01172.html>

- *From:* "John Carson" <jcarson_n o sp am @xxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 27 Jul 2005 15:35:56 +1000
-

<google@xxxxxxxxxxxxxxxxxx> wrote in message
<news:1122429421.968377.141230@xx>

John Carson wrote:
[...]

Thanks for those remarks. I really appreciate you taking the time to comment.

It never occurred to me that the friend declaration could be an overload. That certainly helps make sense of why the code compiles. I still have some questions, however. The following code compiles on both VC++ 8.0 (Beta 2) and Comeau Online.

```
#include <iostream>
using namespace std;

template <class T>
class C
{
public:
    C(int x_) : x(x_) {}
private:
    int x;

    template <class Other>
    friend bool operator == (C<T>&, C<Other>&);
};

// global instance of C<int*>
C<int*> c0(7);
```

Re: Class templates and friend function templates

```
template <class U, class V>
bool operator == (C<U>& u, C<V> &v)
{
    // access private data of c0
    cout << c0.x;

    return u.x == v.x;
}

int main()
{
    C<int> c1(1);
    C<float> c2(2);
    bool b = c1 == c2;
    return 0;
}
```

My questions concerning that code are:

1. If the friend declaration declares an overload, where is that overload defined?

In your example it is not defined. However, you can defined it "inline" (i.e., as part of the friend declaration).

(Comeau Online says it doesn't link, but "will instantiate all templates hence doing a fake link" --- I am not sure exactly what that means and whether it is relevant, but VC++ certainly links).

I suspect the online version uses an EDG option to "instantiate every used template instance". However, it is not an actual link operation.

Our demo compiler in its strict mode leads to the following link error:

```
bool operator ==<T1>(C<int> &, C<T1> &) [with T1=float] t.o
ld: fatal: Symbol referencing errors. No output written to a.out
```

Re: Class templates and friend function templates

which is as expected.

Yes, that is what I would have expected based on your comments. VC++, however, doesn't give any errors, compiler or linker, and the code runs and outputs the value 7 (the x value from c0). This would appear to be a bug in VC++. If one can make any sense of the VC++ behaviour, it seems to be accepting

```
template <class Other>
friend bool operator == (C<T>&, C<Other>&);
```

on the basis that it is injecting a new entity but then accepting

```
template <class U, class V>
bool operator == (C<U>& u, C<V> &v)
{
    cout << c0.x;
    return u.x == v.x;
}
```

as the source of this new entity's definition.

As for Comeau, I have used the online version for a long time as a test of correctness, but clearly I need the full version (with actual linking) if I am to test some of the subtler template issues.

2. If I have understood you correctly, the definitions of c0, c1 and c2 should lead to the following overload friend declarations:

```
template <class Other>
friend bool operator == (C<int*>&, C<Other>&);
```

```
template <class Other>
friend bool operator == (C<int>&, C<Other>&);
```

```
template <class Other>
```

Re: Class templates and friend function templates

```
friend bool operator == (C<float>&, C<Other>&);
```

respectively.

Correct.

The line from main()

```
bool b = c1 == c2;
```

should lead to a call to

```
operator == (C<int>&, C<float>&);
```

The signature you quote there isn't detailed enough. The candidates are the three friend functions you mention above and the template defined in your example. Of those, the viable ones (i.e., the ones that could resolve the "c1 == c2" expression) are

```
template<class Other> friend  
bool operator==(C<int>&, C<Other>&);
```

and

```
template<class U, class V>  
bool operator==(C<U>&, C<V>&);
```

The deduced signatures of these two are equally good (they're in fact the same types). Therefore, the concept of "partial ordering of function templates" (section 14.5.5.2 in the standard) comes into play. In this case, the first declaration (the friend) is selected and since it doesn't have a definition, you get the linker error mentioned above.

Re: Class templates and friend function templates

(You can tell the partial ordering by noticing that the friend candidate "could be an instance" of the nonfriend candidate, but not vice versa -- I'm talking function types here.)

I can't see where the friend declarations make this operator a friend of C<int*>, yet it can access the private data of c0, which is an instance of C<int*>. (For that matter, I can't see where the friend declarations make the operator a friend of C<float>.)

Note that the nonfriend template is never instantiated and the access check (or error) is therefore never needed.

Daveed

OK. That explains why Comeau online doesn't flag an access error.

VC++, on the other hand, is doing at least two things wrong:

1. It is instantiating an instance of the operator from

```
template <class U, class V>
bool operator == (C<U>& u, C<V> &v)
{
    cout << c0.x;
    return u.x == v.x;
}
```

when it should instead be producing a link error for the more specialised form of the operator declared in the friend declaration.

2. It is granting this operator friendship rights that it shouldn't have.

Again, many thanks for your comments.

--

John Carson

Re: Class templates and friend function templates