

# C++ forwarding reference and r-value reference

Asked 4 years, 7 months ago   Modified 1 year ago   Viewed 6k times

I understand that a forwarding reference is "an rvalue reference to a cv-unqualified template parameter", such as in

```
template <class T> void foo(T&& );
```

which means the above function can take both l-value and r-value reference.

There's something I don't understand, e.g.

```
template <class T>
class A
{
    template <class U>
    void foo(T&& t, U&& u)
    {
        T t2( std::forward(t) ); // or should it be std::move(t)? is T&& forwarding or
r-value reference
        U u2( std::forward(u) ); // or should it be std::move(u)? I believe U&& is
forwarding reference
    }
};
```

in the above code, **are both T&& and U&& forwarding references?**

I wrote some code to test (VS2015 compiler):

```
class A
{
public:
    A(){};
    A(const A& rhs)
    {
        std::cout << "calling 'const A&' l-value" << std::endl;
    }

    A(A&& rhs)
    {
        std::cout << "calling ' A&&' r-value" << std::endl;
    }
};

template <class T>
class Test
{
public:
    void test1(T&& t)
    {
        T t2(std::forward<T>(t));
    }

    template <typename X>
    void test2(X&& x)
    {
        T t2( std::forward<T>( x ) );
    }
};

void main()
{
    A a;
    Test<A> test;
    test.test1(A());
    test.test1(std::move(a));
    //test.test1(a); // this doesn't compile. error: cannot convert argument 1 from 'A'
to 'A &&', You cannot bind an lvalue to an rvalue reference

    test.test2<A>(A());
    test.test2<A>( std::move( a ) );

    //test.test2<A>( a ); // this doesn't compile. error: cannot convert argument 1
from 'A' to 'A &&', You cannot bind an lvalue to an rvalue reference
}
```

I was expecting that test.test1(a); and test.test2(a) should both compile if they are forwarding references, but neither does.

Could someone explain this to me? Thanks!

**Edit** -----thanks, guys----- Richard and Artemy are correct.

c++   rvalue-reference   forwarding-reference

Share   Edit   Follow   Flag

edited Sep 1, 2017 at 15:13

asked Sep 1, 2017 at 8:33

 **Dave**  
169 ● 1 ● 6

3   Unrelated, I don't know whether its sad or surreal that, in this day and age, the VS2015 toolchain accepts `void main()` . Really ? – [WhozCraig](#) Sep 1, 2017 at 8:50

1   In your last closed case U is not deduced - so your case will not compile. To make it deducible call it as test.test2( a ); and inside test 2 change forward<T> to forward<X> – [Artemy Vysotsky](#) Sep 1, 2017 at 9:05

## 3 Answers

Sorted by: Highest score (default)

It's a great question which foxes almost everyone in the beginning.

```
template <class T>
class A
{
    template <class U>
    void foo(T&& t, U&& u);
};
```

In this example, **T** is **not deduced** (you explicitly define it when you instanciate the template).

**u** is deduced because it's deduced from the argument **u**.

Therefore, in almost all cases it would be:

```
std::move(t);
std::forward<U>(u);
```

Share   Edit   Follow   Flag

answered Sep 1, 2017 at 8:39

 **Richard Hodges**  
66.1k ● 6 ● 85 ● 131

1   BTW `std::forward<T>(t);` would be equivalent `std::move(t);` is that case (but less explicit). – [Jarod42](#) Sep 1, 2017 at 8:57

are both T&& and U&& forwarding references?

No, only **u&&** is a **forwarding reference**, because **u** is the only **template argument** that's **being deduced**. **T** was already "chosen" when instantiating **A**.

只能靠选择得知的类型: 右值引用


可以通过推导得到的模板类型参数: 转发引用

In addition to what Richard and Artemy pointed out, when you specified `test.test2<A>( a )`, the type X is already explicitly defined to be A.

When you change it to `test.test2( a )`, then the type X should be deduced and it should compile.

Share   Edit   Follow   Flag

answered Apr 3, 2021 at 21:25

 **Johanna Ye**  
1