Dependent Names

In this lesson, we'll study dependent names.

WE'LL COVER THE FOLLOWING

Dependent Names

A dependent name is essentially a name that depends on a template parameter. A dependent name can be a type, a non-type, or a template-template parameter.

If you use a dependent name in a template declaration or template definition, the compiler has no idea, whether this name refers to a type, a non-type, or a template parameter. In this case, the compiler assumes that the dependent name refers to a non-type, which may be wrong.

Let’s have a look at the example of dependent names:

```
template<typename T>
struct X : B<T>   // "B<T>" is dependent on T
{
    typename T::A* pa;   // "T::A" is dependent on T
    void f(B<T>* pb) {
        static int i = B<T>::i;    // "B<T>::i" is dependent on T
        pb->j++;    // "pb->j" is dependent on T
    }
};
```

`T` is the template parameter. The names `B<T>`, `T::A`, `B<T>`, `B<T>::i`, and `pb->j` are dependent.

Two-phase name lookup

- Dependent names are resolved during template instantiation.
- Non-dependent names are resolved during template definition.

`A` from a template parameter `T` is dependent, qualified name `T::A` can be a

- Type
- Non-type
- Template

> The compiler assumes by default that `T::A` is a non-type.

The compiler has to be convinced that `T::A` is a type or a template.

The Dependent Name is a Type `typename`

```
template <typename T> void test(){
  std::vector<T>::const_iterator* p1;          // ERROR
  typename std::vector<T>::const_iterator* p2; //OK
}
```

Without `typename` like in line 3, the expression in line 2 would be interpreted as multiplication.

The Dependent Name is a Template `.template`

```
template<typename T>
struct S{
  template <typename U> void func(){}
}

template<typename T>
void func2(){
  S<T> s;
  s.func<T>();             // ERROR
  s.template func<T>();    // OK
}
```

Compare lines 9 and 10. When the compiler reads the name `s.func` (line 9), it decides to interpret it as non-type. This means, the `<` sign stands in this case for the comparison operator but not opening square bracket of the template

argument of the generic method `func`. To help the parser, you have to specify that `s.func` is a template like in line 10: `s.template func`.

To learn more about dependent names, click here.

In the next lesson, we'll look at an example of dependent names.