Doc. no. P0289R0
Date: 2016-02-16
Project: Programming Language C++
Reply to: Matt Austern <austern@google.com>

Forward declarations of nested classes

History

This general idea was briefly discussed in EWG as a "tiny" proposal a few years ago, perhaps in 2013. There were no fundamental objections to it, but the general sense of the room was that this wasn't really tiny and that it couldn't be discussed without a paper. This is that long-delayed paper.

Motivation

Forward declarations are useful for declaring classes that refer to each other (the standard gives the example of `Matrix` and `Vector` classes each of which needs an `operator*(const Matrix&, const Vector&)`), and for hiding information (allowing clients to pass around pointers to a class that serves as an opaque token), and for reducing compilation time by reducing the number of header files that need to be included to use an interface.

A nested class `X::A` is useful for expressing the fact that `A` is part of `X`'s interface. Examples include `vector<T>::iterator` and nested messages in protocol buffers. The same considerations that make forward declaration of namespace scope classes also apply to nested classes.

There is one sense in which we already allow forward declaration of nested classes: it's legal to declare `X::A` in `X`'s class definition,

```
struct X {
  ...
  class A;
};
```

where `X` is a complete type and `A` is an incomplete type. This is common when `X` is defined in `x.h` and `A` is a private implementation detail that's defined in `x.cc`. What we don't currently have is any mechanism for referring to `X::A` when `X` itself is an incomplete type. In particular, the way that programs are usually organized today, there is no way to refer to `X::A` without having included `x.h`. Sometimes that's annoying, and it introduces an otherwise unnecessary dependency on everything in `x.h` when all that was needed was a single incomplete type to form pointers or references in function declarations.

This is a proposal to allow the forward declaration of a class nested within a type whose class definition hasn't yet been seen. The premise behind this proposal isn't that this is a high priority change, but rather that it's a small and simple change that removes an unnecessary restriction and that makes nested classes more like namespace-scope classes. This proposal doesn't attempt to provide precise wording changes; that can come later, if EWG thinks this would be a useful direction.

What would have to change in the standard

The changes in the standard would be reasonably localized. Forward declarations of classes are covered in 9.1 [class.name]/2: "A *declaration* consisting solely of *class-key identifier;* is either a redeclaration of the name in the current scope or a forward declaration of the identifier as a class name. It introduces the class name into the current scope." It's almost enough to just replace class-key with *class-key nested-name-specifier*$_{opt}$ and then adjust the wording slightly to say which scope the name is introduced into.

That change wouldn't distinguish between declaring `A` as a name nested within some class `X` whose definition hasn't been seen, and declaring `A` as a name nested within some namespace `X` whose definition hasn't been seen. Even without that distinction it would probably be implementable; if a compiler knows that `X::A` is to be treated as the name of a class, and knows how to mangle function declarations using that class, then that's enough to be able to make sense of declarations using the name. Currently `X::A` always refers to a pre-existing entity, and it's possible that some compilers use that fact when lexing, but it shouldn't pose fundamental difficulties.

From the perspective of error reporting and spelling correction, however, this minimal change would be insufficient without another restriction: if there's a forward declaration of the form `class X::A;` then we must already have seen a declaration (not necessarily a definition) of `X`, so that we know whether `X` refers to a class or a namespace. So it would be legal to write

```
    class X;
    class X::A;
```

but not simply

```
    class X::A;
```

with no preceding declaration of `X`. This also guards against the possibility that some compilers might mangle `X::A` differently depending on whether `X` is a class or a namespace.

Finally, we would have to decide whether only a declaration of `X::A` is permitted when `X` is an incomplete type, or also a definition. Both options have disadvantages. Allowing only declarations is nonuniform and would require extra text elsewhere in the standard to express that restriction. Allowing the definition of `X::A` when `X` hasn't been defined, however, would raise other questions. First, name lookup in a nested class includes name lookup into its surrounding class, which doesn't make sense if the surrounding class is incomplete. (We probably don't want to make the definition of `X::A` legal regardless of whether `X` has been defined but have different rules for name lookup in the two cases — that would be too error prone.) Second, if we were to allow such class definitions then we'd also have to decide whether to allow a class to inherit from one of its own nested classes, e.g.

```
    class X;
    class X::A { … };
    class X : public X::A { … };
```

So despite the nonuniformity, I propose that if a class definition for `X` has not been seen then it is legal to declare `X::A` as an incomplete type but not to define it.

Implications and unanswered questions

The usual assumption about an incomplete type is that it will eventually be completed. This introduces some new issues, because we no longer care just about whether a type is completed in some other translation unit but also about whether and how its enclosing type is completed. In particular:

- What if we declare and use `X::A` but `X`'s full class definition never appears in any other translation unit of the program?
- What if `X`'s class definition does appear in some other translation unit but it does not include a nested class `A`?
- What if `X`'s class definition does appear in some other translation unit and does include a nested class `A`, but it's a private member?

Presumably the answer to all three questions would be something along the lines of "ill formed, no diagnostic required."

Additionally, allowing these sorts of forward declarations would imply giving up some properties that declarations currently have: in particular that a name qualified with a nested-name-specifier always refers to a pre-existing entity, rather than introducing something new (as discussed above), and that there is no lookup into an incomplete class type.

(Thanks to Richard Smith for pointing out some of these issues.)