suggest approval

Forward Declaration of Nested Classes

Doc: X3J16/91-0118==WG21/0051

Tony L. Hansen
AT&T Bell Laboratories
tony@attmail.com
hansen@pegasus.att.com

1. Introduction

There is a proposal on the agenda of the Extensions Working Group for X3J16 concerning whether it is allowable to "forward declare" a nested class. That is, is it permissible to define a nested class outside the class in which it was declared? For example, consider the following declarations:

```
class outer {
    class inner;
    . . .
};

class outer::inner {
    . . .
};
```

In this example, the nested class named "inner" has its declaration presented outside of the bounds of the class named "outer".[1] The separate definition of `class outer::inner` is exactly analogous to the way member functions may be separately defined.

2. Pros

There were many arguments presented on the mail reflector in favor of this proposal.

2.1 Modules

Consider a module defined by `class X` which is defined in a main header file `X.h`. Now consider the desire to separate the implementation of that class into an implementation class. The module class usually is implementated using a pointer to the implementation class. Right now you have two choices: use a pointer to a globally-known class, or use a nested class whose definition is nested within `X` and shown in the `X.h`. It would be preferable to have a pointer to a nested class whose definition is expanded only in a private header file. That is, `class X` would have a pointer to `X::Y` within it and `class X::Y` would

1. Many of the examples and arguments used in this memo are derived directly from the conversations found on the mail reflectors. In particular, please refer to posts on the subject by Peter Chapin, Ron Guilmette, Tony Hansen, Andrew Koenig, Glen McCluskey, William M. Miller, and Martin O'Riordan. See x3j16-core-130, x3j16-core-131, x3j16-core-133, x3j16-core-142, x3j16-core-218, x3j16-core-222, x3j16-core-228, x3j16-core-601, x3j16-core-615, x3j16-core-616, x3j16-core-618 and x3j16-core-619. My thanks to all who participated in the discussion of this feature.

be defined in the private header file.

2.2 Clarity

Consider the use of related classes, such as a reference-counted `class String` and the class for its internal representation `String::Rep`. Typically both class definitions are complicated. If you try defining them together within String, you get a major maintenance catastrophy. For an example, look at `class srep` found in *The C++ Answer Book*, pages 240-241, and imagine moving its definition within `class string`, pages 241-243. Now consider adding additional classes such as `String::Iterator` and `String::Substring`, and the problem keeps growing. It would be a boon to the maintenance of nested classes to be able to separate the definitions of the nested and the external classes.

2.3 Orthogonality

Member functions can be declared within the class and defined elswhere. Why not nested classes?

2.4 Usefulness

More importantly, forward class declarations are a useful tool for structuring header files. In many environments, there are several hundred headers and a class is defined as a forward typically 5 times before its actual definition. Presumably the same would be true for nested classes as well.

2.5 Consistency

The users of C++ will expect the capability to exist since it clearly exists for member functions.

3. Cons

There were few objections raised against the proposal, each with a valid rejoinder.

3.1 Does the One Definition Rule (ODR) Apply?

One objection went like this:

> The notion that one can define a class (such as `X`) and then later augment its definition (by separately defining `X::Y`) is incredibly dangerous, because it implicitly allows `X::Y` to be different in different translation units.

In response,

> Class `X` *is* completely defined in the sense that all of its nested classes are declared within the definition of `class X`. It's not like the example is trying to forceably introduce a new member in `class X` without `class X`'s consent.
>
> If we make it clear that the nested classes of an externally linked class are themselves externally linked, then there can be only one definition of `class X::Y` even if it is defined outside `class X`'s definition. This is due to the One Definition Rule. Indeed, the objection Andy raised is not an issue for out-of-line member functions because the ODR is enforced for such functions by most implementations.
>
> In fact, defining `class X::Y` inside `class X` does not force the definition of `class X::Y` to be the same in all translation units. For example, the definition of `class X::Y` might use typedef names that were declared differently above `class X`. Alternatively the definition of `class X` might contain conditional compliation directives which cause `class X::Y` to be defined inconsistently in different translation units.

In summary, no consistency problems are introduced by this proposal which violate the ODR.

3.2 Definition Within the Same or Containing Scope

Another objection went like this:

> I do think we need to be careful to specify exactly what we want to allow, though. For example, is there any reason to allow or forbid something like the following:
>
> ```
> class A {
>       class B {
>             class C {
>                   class D;
>             };
>       };
>       class B::C::D { };
> };
> ```
>
> In other words, should the declared class be permitted to be declared in the "same or containing scope," or should we say it has to be at the same scope or outer containing scope and nothing in-between?

In response, whatever we decide on for member functions should be looked to for guidance for where nested classes definitions can be placed.[2] If a member function declared within `class C` could be defined at the same place that `B::C::D` was placed above, then `B::C::D` should be allowed there as well. I feel that using the "same or containing scope" as the rule would be simple to follow and sufficient. This also eliminates various problems with trying to declare the nested class in another type of scope, such as

```
class A {
      class B;
};

void foo()
{
      class A::B { . . . };
}
```

With the "same or containing scope" as the rule, this example would be illegal.

4. Changes to the Working Draft

There is really nothing within the current draft which explicitly prohibits this proposal, but there is certainly nothing which says that it is permitted. I suggest that the following wording be added to the of §9.7:

---

2. Note that member function definitions are permitted only in very restricted contexts, whereas class definitions are currently legal just about anywhere.

Like a member function, a nested class may be declared within the enclosing class and defined elsewhere in the same or containing scope, using the scoped name for the class definition:

```
class enclose {
      class inner;
      . . .
};

class enclose::inner {
      . . .
};
```

In §17.5, the grammar for *class-head* becomes

class-head:
class-key identifier$_{opt}$ *base-clause*$_{opt}$
class-key nested-class-specifier base-clause$_{opt}$

5. Summary

In summary, this proposal provides definite benefits in modularity, clarity, orthogonality, usefulness, and consistency. It has received few objections, none of which were insoluble. In addition, the description is simple to add to the working draft.