J16/99-0009
WG21/N1186
February 22, 1999
J. Stephen Adamczyk
jsa@edg.com

Nested Class Access to Enclosing Classes

In J16/95-0015=WG21/N0615, I proposed changing the access rules to give nested classes access to the enclosing class. At the time, the committee had weightier issues to ponder, and nested class access was viewed as a relatively minor problem that would be largely corrected by the addition of class name injection. The committee decided against any change.

Now that the standard is out and the smoke has cleared somewhat, we can see that there are a few remaining problems with nested classes. I've been asked to bring this proposal forward again.

Problem: definitions outside the class

There is one real problem: in certain cases, it is not possible to provide a definition for a member of a nested class:

```
class A {
  class B {
    struct C {};
    static C member;
  };
};
A::B::C A::B::member;  // Error
```

The reference `A::B::C` gets an access error, because `B` is a private class. (The reference `A::B::member`, on the other hand, is allowed because there is a rule that grants special access when naming a member in the declarator of its definition.) The definition does have access to the private type `C`, but unfortunately there is no way to name that type without using the private type `B`. So there is no way to write the definition, short of adding a friend declaration.

Similar problems are possible with the return type on the definition of a member function of a nested class.

Lesser Problems

The other remaining problems are aesthetic problems.

Class name injection does allow a nested class to refer to the name of its enclosing class, but only with some forms of name:

```
class A {
  class B {
    A    *p;  // Okay
    ::A  *q;  // Okay
    B    *r;  // Okay
    A::B *s;  // Error, perhaps surprisingly
  };
};
```

At the very least, this introduces a certain amount of inconsistency that may be confusing to the programmer. One could go further and say that the access restriction seems pointless; if the name is accessible as a simple qualified name, why shouldn't it also be accessible via a qualified name?

More interesting, perhaps, are cases that go to the heart of the issue: is it desirable that a nested class have access to the members of the enclosing class? For example:

```
class A {
  class B {};
  class C {
    B *p;
  };
};
```

According to the standard, this is invalid, and yet it seems like a reasonable thing to do. The fact that the class designer has placed `B` and `C` inside `A` indicates that they are closely bound to `A`. It seems appropriate that they should have access to the members of `A` so that they can cooperate using types whose scope is limited to `A`. There is no hijacking of access here: the author of the nested classes is also the author of the enclosing class, and he or she presumably trusts the nested classes (if not, he or she could have chosen to place the nested classes outside the enclosing class).

Note that the proposal does not give `C` access to the members of `B`, only to the members of `A`. Nested classes are trusted by their parents, but not necessarily by their siblings.

Proposed change

Replace all of 11.8 [class.access.nest] by

A nested class is implicitly a friend (_class.friend_) of each enclosing class. The members of an enclosing class, on the other hand, have no special access to members of a nested class; the usual access rules (clause _class.access_) shall be obeyed.

The examples must be reworked accordingly.

Also delete 11 [class.access] paragraph 6, which is no longer needed.