X3J16/95-0015
WG21/N0615

Nested Class Access to Enclosing Classes

J. Stephen Adamczyk
Edison Design Group, Inc.
jsa@edg.com

January 26, 1995

The WP (9.8, class.nest) says "Member functions of a nested class have no special access to members of an enclosing class." Does this say the right thing? Does it say enough? Consider

```
class A {
  typedef int I;
  class B {
    I x;   // Okay? (1)
    B *p;  // Okay? (2)
    void f() {
      B *q;  // Error? (3)
    }
  };
  class C : public B {
    B *r;  // Okay? (4)
  };
  class D {
    B *s;  // Okay? (5)
  };
};
```

The comments above indicate my best guesses at answers given the current WP wording.

Note that the WP says *member functions* have no special access, which leaves open the possibility that things in a nested class that aren't member functions do have special access. That is why I made cases 1, 2, 4, and 5 valid. But maybe the restriction to member functions was not intended (e.g., it dates from a time when compilers did not check access on references outside of member functions), and *all* members of a nested class have no special access.

Is there a problem here? I think so. I've gotten quite a few bug reports on this issue, and the bug programs are trying to do reasonable things, in my opinion. The status quo is stricter than programmers want or expect.

This topic was discussed somewhat on the core reflector at the end of June 1994. No one wanted to keep the current rule or make it stricter, and the two proposals for change were

- Give class members access to the name of their own class, even if the name is a private member of a surrounding class. That would make 2 and 3 well-formed. One could go further and additionally give class members access to the names of base classes of their own class, making 4 well-formed.

- Eliminate the WP sentence quoted, and put in the opposite: members of a nested class have access to the surrounding class. In effect, the nested class is a friend of the surrounding class. That makes all the examples above well-formed.

The second proposal seemed to be the favorite, and it's my favorite too, so let me make the case for it.

A nested class is a member of the surrounding class. As such, it should have member access to the surrounding class. More precisely, everything within the nested class should have that member access, by analogy with member functions—everything within a member function has member access to the class, even member functions of local classes within the member function.

This gives consistent handling for some cases that look as if they should work the same way:

```
class A {
  typedef int I;
  class B {
    I x;                // Okay
    class N {
      void f() { I y; } // Okay (not okay with present WP)
    };
  };
  void af() {
    I x;                // Okay
    class N {
      void f() { I y; } // Okay
    };
  }
};
```

The final reference in the above also illustrates one of the strange effects of the current rules: a local class in a member function, which is in effect a nested class, has special access, but other nested classes do not.

Does the change break encapsulation? I don't consider myself enough of an OO expert to make a pronouncement, but as a practical programmer I note that putting one class inside another is a choice made by the programmer. It doesn't happen accidentally when you pull together classes from many different sources, or when you derive from them. Clearly, in putting one class inside another, a programmer is indicating a close relationship between the classes. It seems reasonable that the nested class has some special powers.

The change also eliminates the need for another special case. Consider

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

We know that this is valid. But how do we know that the reference to `B` (a private type) in `A::B::f` is valid? The working paper doesn't say this, but it seems reasonable that one should be able to refer to the name of a member (including all the classes involved in the qualified name) in order to define it. If we make nested classes friends of the surrounding classes, no special rule regarding definitions is required.

Recommended WP changes:

In 9.8 (class.nest), replace “Member functions of a nested class have no special access to members of an enclosing class.” by “Nested classes have special access to members of an enclosing class; a nested class is implicitly a friend (_class.friend_) of each enclosing class.” In the associated example, replace the comment “error: E::x is private” by “okay: I is a friend of E”.