X3J16/96-0034
WG21/N0852

Clarifying Access Control

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

January 30, 1996

Introduction

The description of access control in 11 [access] fails to mention one rule that seems to be universally implemented. Perhaps one can intuit the rule from the things that are described, but probably only if one already knows how access control is "supposed" to work. This paper proposes to add the missing rule.

The problem

The following program works on everyone's C++ compiler:

```
class B;
class A {
private:
  int i;
  friend void f(B *);
};
class B : public A {};
void f(B *p) {
  p->i = 1;
}
```

It's not clear whether this program is well-formed according to the WP, however. The issue is that the member `i` is inaccessible in the class `B` (it's private in the base class `A` and therefore inaccessible in the derived class `B`), so at first glance `p->i` must be an access violation. However, function `f` is a friend of class `A`, so it has permission to implicitly cast a pointer to `B` to a pointer to `A`. In `A`, `i` is private, but since function `f` is a friend, it can access a private member. Therefore `f` would be allowed access in this case if the code were written

```
static_cast<A*>(p)->i = 1;
```

This formulation does not do any cheating that defeats access control (such as an old-style explicit cast to the base class), so it seems clear that `f` does have the access needed to refer to `i` in some way. Therefore, it seems reasonable to allow `f` to have access with the original formulation also.

Perhaps this seems obvious, and it's certainly existing practice, but I don't think there's anything in [access] that permits it.

I recommend that we modify the WP to say that a nonstatic member can be accessible either because it's accessible in the class in which it's named, or because it's possible to implicitly cast the pointer to the object you have to a base class where you do have access. The same applies for references using the "`.`" operator, where an implicit cast to a reference to a base class can be used.

Another reason to care

The namespaces changes added all kinds of interesting new access cases that couldn't come up previously, and here's one where the fact that using-declarations can do things that access-declarations formerly couldn't brings up this issue even without a friend declaration:

```
class A {
public:
  int i;
};
class B : public A {
private:
  using A::i;
};
void f(B *p) {
  p->i = 1;
}
```

`i` is private in the derived class, but it's public in the base class; the derivation is public, so anyone can do an implicit cast to the base class. Therefore by the reasoning above this example should be well-formed.

Working Paper changes

Add at the end of 11.2 [class.access.base]:

> For nonstatic members, the access is affected by the class in which the member is named. This naming class is the class in which the member name was looked up and found. [Note: this class can be explicit, e.g., when a *qualified-id* is used, or implicit, e.g., when a class member access operator (_expr.ref_) is used (including cases where an implicit "`this->`" is added).] A nonstatic member m is accessible when named in class N if
>
> - m as a member of N is public, or
> - m as a member of N is private or protected, and the reference occurs in a member or friend of class N, or
> - there exists a base class B of N that is accessible at the point of the reference, and m is accessible when named in class B.
>
> [Example:
>
> ```
> class B;
> class A {
> private:
>   int i;
>   friend void f(B *);
> ```

```
};
class B : public A {
};
void f(B *p) {
  p->i = 1;  // Okay: B* can be implicitly cast to A*, and f has
             // access to i in A
}
```

–end example]