Document Number: WG21/N0905
X3J16/96-0087
Date: 28 May 1996
Project: Programming Language C++
Reply to: William M. Miller
wmm@world.std.com

QUALIFIED NAME LOOKUP IN USING DECLARATIONS

I. SUMMARY OF THE ISSUE

In reflector message core-6506, Mike Anderson pointed out the following example from 7.3.3 [namespace.udecl], paragraph 11:

```
namespace B {
  struct g { };
  void g(char);       // OK: hides struct g
}
void func() {
  using B::g;
  g('a');             // calls B::g(char)
  struct g g1;        // g1 has class type B::g
}
```

The issue is whether the declaration of g1 is well-formed. If it is, then the lookup in the using declaration is required to find both declarations of g in namespace B and introduce both into the function's scope. If, on the other hand, references in using declarations are subject to the normal hiding rules (in which the declaration of struct g can only be referred to in an elaborated type specifier), the declaration of g1 is ill-formed.

II. REFLECTOR REFERENCES

The following messages on the core reflector dealt with this topic:

6508-6510; 6513-6520; 6522-6524; 6526-6528; 6580-6585, 6684, 6688.

III. ANALYSIS

Paragraph 2 of 3.4.2.2 [namespace.qual] describes the lookup of a namespace-qualified name as resulting in a set of declarations (as opposed to entities). In a case such as the example cited above, the set will contain _both_ the declarations of g from namespace B:

Given X::m, where X is a namespace, if m is declared directly in X, let S be the set of all such declarations of m.

The question then becomes what to do with the declarations so found. Later in the cited paragraph, the following restriction is made:

If S has exactly one member then X::m refers to that member. Otherwise if the use of m is not one that allows a unique declaration to be chosen from S, the program is ill-formed. [Note: the choice could be made by overload resolution (_over.match_) or resolution between class names and non-class names (_class.name_).

It is clear that this wording is deficient with respect to using declarations; taken literally, it would prohibit using declarations that refer to overloaded functions. A better formulation might be:

> If S has exactly one member then X::m refers to that member. Otherwise, if the use of m is a using-declaration, then X::m refers to all the declarations in S. Otherwise, if the use of m is not one that allows a unique declaration to be chosen from S, the program is ill-formed.

Given this rewording, which is necessary to allow for using declarations that refer to a set of overloaded functions, the question is not whether the lookup finds or does not find a hidden class name, as was supposed in the early part of the reflector discussion, but only whether a hidden class name found by the lookup is introduced into the scope of the using declaration as a hidden class name or not.

IV. ALTERNATIVES

Option 1:

All declarations found by the lookup are introduced into the scope of the using declaration, hence the example cited above is correct.

This would seem to be the most consistent approach -- since multiple declarations (of overloaded functions) are already being "cloned" into the scope of the using declaration, it would seem strange to exclude the declaration of the class name, since it also is a member of the set resulting from the lookup.

Bjarne Stroustrup indicated (in reflector message core-6513) that he believes that the example "reflected the intent of the extensions group at the time," and Tom Wilcox opined (in reflector message core-6508) that, as a user, he would expect the example to work.

Small changes to the current wording of 7.3.3 [namespace.udecl] would be required to implement this resolution, mainly to pluralize references to the "entity" whose declaration is introduced by the using declaration. (This is primarily due to the current text’s definition in paragraph 3 of clause 3 [basic] that a set of overloaded functions constitute a single "entity" and the use of that concept rather than the perhaps more natural concept of "declaration" in the description of using declarations.) These references are:

> paragraph 1: "That name is a synonym for the name of _one or more entities_ declared elsewhere."

> paragraph 8: "The _entities_ declared by a using-declaration shall be known in the context using _them_ according to _their definitions_ at the point of the using-declaration."

No additional normative changes are required to support this option; in particular, paragraph 9 already refers in the plural to the referents of a using declaration ("A name defined by a using-declaration is an alias for its original declarations..."), and paragraph 10 handles the case in which a class name and non-type entities are both declared. It would probably be helpful to the reader to add a footnote something like the following to paragraph 2 or paragraph 9:

> Class names hidden by non-type names declared in the same namespace are found by namespace qualified lookup [namespace.qual] and thus are declared as hidden names in the using declarative region, as well.

Option 2:

If a class name is hidden by a non-type name in the scope in which it is declared, only the non-class names from the result of the lookup are introduced into the scope of the using declaration, hence the example above is incorrect.

This approach would be more consistent with the statement in paragraph 2 of 9.1 [class.name] that class names hidden by non-type names can only be referred to by an elaborated type specifier. Those advocating this position in the reflector discussion (John Spicer, core-6515, and Bill Gibbons, core-6520) expressed support for allowing an elaborated type specifier in the using-declaration syntax, thus allowing either the class name or the non-type name to be used.

The changes required to the current wording to support this option would include:

paragraph 1: change the syntax to read as follows:

```
    using-declaration:
        using class-keyopt ::opt nested-name-specifier
                        unqualified-id ;
        using class-keyopt :: unqualified-id ;
```

paragraph 2: "The member names specified in a using-declaration are declared in the declarative region in which the using-declaration appears, _except that if a class-name is hidden by a non-class name in its scope [class.name] and class-key is omitted from the using-declaration, the hidden class-name is not declared in the using declarative region_."

paragraph 11: change the example to read:

```
    struct g g1;  // error: struct g not declared
```

In message core-6684, Bill Gibbons offered the following points in support of this option:

1. Access declarations are now defined as exactly equivalent to using declarations, so the behavior of existing implementations with respect to access declarations should be considered in resolving this question. Of the three compilers Bill surveyed, two handled using declarations in a manner consistent with this option.

2. All other lookups in the language except for those in elaborated type specifiers ignore hidden class names, so using declarations should do so for consistency.

3. This option allows a finer-grained control over which names are made visible.

V. RECOMMENDATIONS

1. Make the change to 3.4.2.2 [namespace.qual] described in III above (as noted, this is mandatory to allow using declarations to refer to sets of overloaded functions).

2. Option 1 of IV. This option appears to be what was originally intended; several committee members, speaking as users, expressed their expectation that using declarations would work this way; and this approach seems most consistent with the idea of importing a set of declarations of overloaded functions with a single using declaration.