SC22/WG21/N1198
J16/99-0021
31 August, 1999
Thomas R Wilcox
trw@rational.com

Issues Relating to Accessibility

This document discusses problems with the specification of the concept of *accessibility* in the standard and proposes a possible resolution. The discussion is motivated by four items on the core issues list:

1. Issue 9: Clarification of access to base class members
2. Issue 16: Access to members of indirect private base classes
3. Issue 17: Footnote 98 should discuss the naming class when describing members that can be accessed from friends
4. Issue 19: Clarify protected member access

Issues 9 and 16: Clarification of Access to Base Class Members

Committee document SC22/WG21/N1179 J16/99-0002 proposed extensive changes to clause 11 to resolve the logical circularity in the definition of accessible base types that exists in the standard and to generally improve the readability and robustness of that clause. After discussion at the Dublin meeting in April, 1999, it was decided to make only the changes necessary to resolve the issues exposed by the above defects.

There are two changes. The first change is to reword the definition of accessible base class at the start of paragraph 4 so that it parallels the definition of an accessible member given at the end of paragraph 4. The second change is to add a constraint to the fourth bullet at the end of paragraph 4 to prevent infinite recursion of the definition.

After these changes, clause 11.2 paragraph 4 reads as follows (additions shown in bold italic (and in red)):

-4-
A base class ***B of class N****is said to be accessible* ***at a point of reference***if an invented public member of <u>the base class</u>***B*** is accessible ***when named in N***. *If a base class is accessible, one can implicitly convert a pointer to a derived class to a pointer to that base class (conv.ptr, conv.mem). [Note: it follows that members and friends of a class* `X` *can implicitly convert an* `X*` *to a pointer to a private or protected immediate base class of* `X`*.] The access to a member 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. If both a class member access operator and a qualified-id are used to name the member (as in* `p->T::m`*), the class naming the member is the class named by the nested-name-specifier of the qualified-id (that is,* `T`*). If the member* `m` *is accessible when named in the naming class according to the rules below, the access to* `m` *is nonetheless ill-formed if the type of* `p` *cannot be implicitly converted to type* `T` *(for example, if* `T` *is an inaccessible base class of* `p`*'s class).] A 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, and the reference occurs in a member or friend of class *N*, or

- m*as a member of N is protected, and the reference occurs in a member or friend of class N, or in a member or friend of a class P derived from N, where m as a member of P is private or protected, or*
- there exists a base class *B* of *N* ***other than the class that declares m,*** suchthat ***B****is accessible at the point of 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;                    //
```

OK: `B*`*can be implicitly cast to* `A*`*,* `//`*and* `f`*has access to* `i`*in* `A` `}`

--- end example]

Discussion
of issues....

```
Issue 9 Example:
    class D;

    class B
    {
     protected:
       int b1;

       friend void foo( D* pd );
    };

    class D : protected B { };

    void foo( D* pd )
    {
       if ( pd->b1 > 0 ); // Is 'b1' accessible?
    }
```

Analysis: Is b1 accessible in foo?

- b1 is named in D because the Class D pointer pd is used to address it.
- In D, b1 is protected, so the first and second bullets of the definition of an accessible member do not apply.
- The third bullet is not satisfied since foo is not a member or friend of D or any of its subclasses.
- The fourth bullet is not satisfied since there is no base of D that satisfies the conditions. Class B is the only base class of D but it is also the class that defines b1.

```
Issue 1 Example:
  class D;

  class B {
  private:
      int i;
      friend class D;
  };

  class C : private B { };
```

```
class D : private C {
    void f() {
        B::i; //1: well-formed?
        i;    //2: well-formed?
    }
};
```

(A) Is B::I well-formed?

- Since i is a non-static data member, the expression is "this -> B::I", so (1) member i must be accessible within f when named in B and (2) B must be an accessible base class of D within f.

  1. Member i is accessible within f when named in B because i is a private member of B and f is a friend of B (second bullet).
  2. Is B an accessible base class of D within f? That is, if $Bpub were a public member of B, would it be accessible in f when named in D?

  ◊ Because private inheritance is used in both the definition of C and D, $Bpub is neither a public, protected or private member of D. So, if $Bpub is accessible, the fourth bullet needs to be satisfied.
  ◊ Since B declares $Bpub, C is the only base class of D that can be considered in the fourth bullet.
    · C is accessible to D within f since f is a member of D and a public member of C would be private in D (second bullet).
    · But $Bpub is not accessible in f when named in C.
      - $Bpub is private in C and f is neither a friend nor a member of C (second bullet) and
      - There are no base classes between C and B (fourth bullet).

(B) Is i accessible in f when named in D?

- Member i is neither public, protected, or private in D, so if it is accessible, there must be an accessible base class of D where it is accessible.
- C is an accessible base class of D within f because f is a member of D and a public member of C would be a private member of D.
- Member i is neither a public, protected, or private member of C (because it is a private member of B), so if it is accessible within f, there must be an base class of C accessible within f where it is accessible.
- B is not an accessible base class of C because (1) although a public member of B would be a private member of C, f is not a member or friend of C and (2) although f is a member of the subclass D, a public member of B would not be a protected or private member of D.

So, in both these cases, friendship with the base class does not grant access to the private member because the base class is not accessible through two levels of private base classes. If, however, B were merely a protected base of C rather than a private base of C, then i would be accessible within f because B would be an accessible base of D within f (f is a member of D and a public member of B would be a private member of D) and it has already been established that i is accessible within f when named in B. The B:: qualification is redundant in this case.

Issue 19: Clarifying Protected Member Access

In the discussion of issue 19 in Dublin it was decided that 11.5 was fine as it stood without reference to the naming class concept of 11.2 since the additional requirement defined in 11.5 is orthogonal to the naming class. The naming class issues are already dealt with in clause 11.2, to which this clause already refers.

Issue 17: Footnote in 11.2 Should Discuss the Naming Class

The footnote in paragraph 1 of clause 11.2 currently reads

[Footnote: As specified previously in clause
class.access, private members of a base class remain inaccessible even to derived classes unless `friend`declarations within the base class declaration are used to grant access explicitly. --- end footnote]

11

It would appear that the sole intent of this footnote is to point out that the partitioning of members into public, protected, and private members in the first paragraph is not complete. And that indeed there are members of a class that are neither public, protected, or private. The issue is discussed more completely in SC22/WG21/N1179 J16/99-0002.

Although the wording of the footnote could be strengthened, it is correct as it stands and, therefore, should not be considered a defect. The use of friend clauses is necessary for access to private members outside the base class, although it may not be sufficient in some naming contexts when private inheritance is used for derived classes. In fact, these additional restrictions are the subject of the paragraphs that follow this footnote in the clause, and the footnote really can’t say more.

For these reasons, I recommend that this issue not be considered a defect.