```
Accredited Standards Committee X3       Doc No: X3J16/96-0122   WG21/N0940
Information Processing Systems          Date:   May 28th, 1996
Operating under the procedures of       Project: Programming Language C++
American National Standards Institute   Ref Doc:
                                        Reply to: Josee Lajoie
                                                  (josee@vnet.ibm.com)
+-----------------------------------------------+
| Name Look Up Issues and Proposed Resolutions |
+-----------------------------------------------+
```

```
654 - Qualified look up for names after global scope ::
======

  The description in 3.4.2.2[namespace.qual] indicates that for
    A::m
  the name m is looked up in the scope of A and, if not found in A, in
  the scopes named by using directives in A, and if not found in these
  scopes, in the scopes named by using directives in these scopes, ...

  This subclause omits to mention what happens if the name is qualified
  by the :: global scope resolution operator.

  Possible Solutions:
  -------------------
  1) such a name is looked up just as unqualified-ids are, in which case
     a transitive closure of all active using directives in global scope
     is used.

  2) such a name is looked up just as qualified-ids are, in which case
     the global scope is searched first, and if the name is not found in
     global scope, the scopes named by using directives in the global
     scope are searched, ...

  Proposed Resolution:
  --------------------
    Adopt option 2.
    This makes qualified name look up more consistent and reinforces
    the notion that global scope is just another namespace that behaves
    like any other namespace scope.

    Proposed new words:
      3.4.2[basic.lookup.qual] para 4 should refer to the subclause on
      qualified namespace member look up (3.4.2.2[namespace.qual]) for
      the description of the look up of names following the :: unary
      operator.

      3.4.2.2[namespace.qual] should be reworked to make sure the rules
      cover the look up of names following the :: unary operator.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

646 - Can a using declaration refer to a hidden base class member?
======

  Given:
      struct A {
          typedef int T;
      };
      struct B : A {
      protected:
          typedef double T;
      };
      struct C : B {
          using A::T;
      };
  Is the using declaration above well-formed?
  7.3.3[namespace.udecl] para 4 says:
    "A using-declaration used as a member-declaration shall refer to a
     member of a base class of the class being defined."

  This doesn’t seem to prohibit applying a using-declaration to a
  hidden base class member.

  Proposed Resolution:
  --------------------
    Make it clear that a using-declaration may refer to a base class
    member that is hidden in the class being defined.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

636 - Can a typedef-name be used to declare an operator function?
======

  Given:
      typedef int I;
      struct S {
          operator I(); // Is this allowed?
      };

  Proposed Resolution:
  --------------------
    I believe typedefs should not be used as the identifier in a
    declarator for a constructor, destructor or operator function.
    This makes things simpler.

    12.3.2 [class.conv.fct] should say:
      "A typedef-name shall not be used as the identifier representing
       the type of a conversion function."

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

446 - Can explicit qualification be used for base class navigation?

  Bill Gibbons asks the following:
    > Can explicit qualification be used for base class sublattice
    > navigation?
    >
    >   class A {
    >   public:
    >     int i;
    >   };
    >   class B : public A { };
    >   class C : public B { };
    >   class D {
    >   public:
    >     int i;
    >   };
    >   class E : public D { };
    >   class F : public E { };
    >   class Z : public C, public F { };
    >   Z z;
    >   ... z.F::E::D::i; // is qualification allowed here to navigate the
    >                     // base class sublattice?

  Proposed Resolution:
  --------------------
    I don’t believe this is really necessary.
    I would like to close this issue without any further action.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```