Doc No: X3J16/97-0085 WG21/N1123
Date: September 30th, 1997
Project: Programming Language C++
Ref Doc:
Reply to: Josee Lajoie
(josee@vnet.ibm.com)

How should the keyword 'template' be added to the syntax?

A) Adding 'template' to the C++ grammar

Core 3 noticed that the current C++ grammar does not support the use of the template keyword in all the places where subclause 14.2 says it is allowed. For example, the following cases are not allowed by the grammar:

In qualified-ids:

```
      A<T>::template B<X>::template C<Y>
            -------------
```

In pseudo-destructor-calls:

```
      p->A::template B<T>::~B();
            -------------
```

After discussions with Bill Gibbons, John Spicer, Anthony Scian and myself, it seems that we cannot come to an agreement as to how to fix this.

Here are the two approaches that are under consideration:

A.1) allow the template keyword in the template-name production, i.e.

```
template-name
    template(opt) identifier
```

This is a simple grammar fix but it allows the 'template' keyword in many more contexts than that currently allowed by chapter 14. The solution would be to prohibit the 'template' keyword to appear in these additional contexts by adding additional semantics rules in the WP.

A.2) apply the grammar change higher up in the grammar, to allow the keyword template only in the places that are already allowed by the text in chapter 14. This means that a greater number of grammar rules must be changed and there is the possibility that we did not cover all cases.

2.1) Add "template" to the *middle* of the grammar rule for nested-name-specifier, as in:

```
  nested-name-specifier:
     class-or-namespace-name :: nested-name-specifier opt
|    class-or-namespace-name :: template nested-name-specifier
```

so that any "template" keyword before the final (unqualified) name is not part of the "nested-name-specifier", but is associated with the unqualified name - as it is now.

This allows the following:

```
    A<T>::template B<X>::template C<Y>
          -------------
```

2.2) Change the grammar for pseudo-destructor-name to say:

```
   pseudo-destructor-name:
       ::opt nested-name-specifier opt type-name :: ~ type-name
|      ::opt nested-name-specifier template template-id :: ~ type-name
       ::opt nested-name-specifier opt ~ type-name
```

This allows:

```
    p->A::template B<T>::~B();
          -------------
```

2.3) Change the grammar for simple-type-specifier to say:

```
   simple-type-specifier:
       ::opt nested-name-specifier opt type-name
|      ::opt nested-name-specifier template template-id
```

2.4) Change the grammar for elaborated-type-specifier to say:

```
   elaborated-type-specifier:
    class-key ::opt nested-name-specifier opt identifier
    enum ::opt nested-name-specifier opt identifier
    typename ::opt nested-name-specifier identifier
|   typename ::opt nested-name-specifier template opt template-id
```

In the discussions in the small group, Bill, Josee (and I believe John as well) preferred 2). Anthony strongly disagreed with 2) and preferred 1) because of the possibility that the changes proposed by 2) are not sufficient and will need future repairs.

B) Additional clarifications needed:

B.1) We need syntax to allow:

```
friend class A<T>;
```

and if the following is well-formed:

```
class A<T>::template B<T> *ptr;
```

we need syntax to allow this as well.

Solution 1)
To allow both, the first line in the elaborated specifier rule should be changed to read:

```
class-key ::opt nested-name-specifier opt template opt identifier
```

Solution 2)
Add the following grammar rule to support "friend class A<T>;"

```
member-declaration:
  friend class-key ::opt nested-name-specifier opt identifier
                                            < template-argument-list >
  friend class-key ::opt nested-name-specifier template identifier
                                            < template-argument-list >
```

Solution 2) does not allow the declaration of "ptr" though, so, if the declaration of ptr should be allowed, Solution 1) should be preferred.

B.2) Are the keywords 'typename' and 'template' allowed in base-specifier and mem-initializer-id?

Solution 1):

```
  "template" should be prohibited in these cases, just as it is
   currently the case for "typename", and the compiler should assume
   that any name followed by a '<' is a template name.

 Solution 2):
  "template" should be allowed in these cases (though not required)
  following the nested-name-specifier.

B.3) There is a related bug in the grammar; the production for
     default arguments for template template parameters:

     type-parameter:
        ...
        template < template-parameter-list > class identifier opt =
                                                    template-name

  does not allow member templates to be default template template
  arguments.  This seems to be an oversight.  I think it should read,
  for A.1):

     type-parameter:
        ...
        template < template-parameter-list > class identifier opt
            = ::opt nested-name-specifier opt template-name

  or for A.2):

     type-parameter:
        ...
        template < template-parameter-list > class identifier opt
            = ::opt nested-name-specifier opt template-name
        template < template-parameter-list > class identifier opt
            = ::opt nested-name-specifier template template-name

  (Note that A.1 does not fix this new problem, it just makes it
  possible to fix it in a slightly simpler way.  So the existence of
  this new problem does *not* imply that Anthony's proposal might
  cover cases we had not thought of.)
```