N0892 = 96-0074

Working Paper changes for Small Template Issues

Sean A Corfield, ocs@corf.demon.co.uk
Bill Gibbons, bill@gibbons.org
John Spicer, jhs@edg.com

13 March, 1996 - Santa Cruz

This document shows the proposed WP changes for resolutions to issues raised in N0841 = 96-0023 Template Issues and Proposed Resolutions - Revision 14.

2.28

Replace 7.1.5.3 [dcl.type.elab] paragraph 2 with:

> If an *elaborated-type-specifier* is the sole constituent of a declaration, the declaration is ill-formed unless it is a specialization (_temp.spec_), an explicit instantiation (_temp.explicit_) or it has one of the following forms:
>
> *class-key identifier*;

2.29 (different resolution to N0841 = 96-0023)

No Working Paper changes.

2.30

In 14.5 [temp.spec] paragraph 1, replace the first sentence of paragraph 1 with:

> An explicit specialization of any of the following:
>
> function template
> class template
> member function of a class template
> static data member of a class template
> member class of a class template
> member class template of a class template
> member function template of a class template
>
> can be declared by a declaration introduced by `template <>`; that is:

And insert before "A specialization of a static data member...":

> Member function templates and member class templates of nontemplate classes and class template specializations may be specialized in the same manner as function templates and class templates.

A specialization of a member function template or member class template of a nonspecialized class template is itself a template.

3.28 (different resolution to N0841 = 96-0023)

Add to 7.1.5.3 [dcl.type.elab], paragraph 5 after "resolves to a *typedef-name*" insert "or a template *type-parameter*"

Remove 14.2.1 [temp.local] paragraphs 2 and 3.

3.29 part 1

Replace 14.10.2 [temp.deduct] paragraph 2 with (changes indicated in boldface):

> Type deduction is done for each parameter of a function template that contains a reference to a template parameter that is not explicitly specified. The type of the parameter of the function template (call it *P*) is compared to the type of the corresponding argument of the call (call it *A*), and an attempt is made to find types for the template type arguments, **templates for the template template arguments**, and values for the template non-type arguments, that will make *P* after substitution of the deduced values and explicitly-specified values (call that the deduced *P*) compatible with the call argument. Type deduction is done independently for each parameter/argument pair, and the deduced template argument types, **templates** and values are then combined. If type deduction cannot be done for any parameter/argument pair, or if for any parameter/argument pair the deduction leads to more than one possible set of deduced values, or if different parameter/argument pairs yield different deduced values for a given template argument, or if any template argument remains neither deduced nor explicitly specified, template argument deduction fails.

Replace 14.10.2 [temp.deduct] paragraph 5 first sentence with:

> A template type argument *T*, **a template template argument *TT*** or a template non-type argument *i* can be deduced if *P* and *A* have one of the following forms:

Add to 14.10.2 [temp.deduct] paragraph 5 the following forms:

```
TT<T>

TT<i>

TT<>
```

Replace 14.10.2 [temp.deduct] paragraph 6, sentence 2 with:

> Similarly, *<T>* represents template argument lists where at least one argument contains a *T*, *<i>* represents template argument lists where at least one argument contains an *i* **and <> represents template argument lists where no argument contains a *T* or an *i*.**

Remove ediorial box 32.

6.30

Replace 14.4 [temp.explicit] paragraph 2 with:

> The syntax for explicit instantiation is:
>
> ```
> explicit-instantiation:
>
>    template declaration
> ```
>
> Where the *unqualified-id* in the *declaration* shall be **either** a *template-id* **or, where all template parameters can be deduced, a *template-name*.** *[Example:*
>
> ```
> template class Array<char>;
>
> template void sort(Array<char>&);
> ```
>
> *-- end example]*

Remove editorial box 29.

6.31

Add after14 [temp] paragraph 2:

> In a function template declaration, the *declarator-id* shall be a *template-name* (i.e., not a *template-id*).

6.32

Add to 14.4 [temp.explicit] paragraph 3:

> If the *declaration* names a compiler-generated function, the program is ill-formed.

In 14.5 [temp.spec] paragraph 1, insert before “A specialization of a static data member…”:

If the *declaration* names a compiler-generated function, the program is ill-formed.

6.33 and 6.38

In 8.3 [dcl.meaning], replace "the definition of a function, variable, or class member of a namespace outside of its namespace" with "the definition or explicit instantiation of a function, variable, or class member of a namespace outside of its namespace, or the definition of a previously declared explicit specialization outside of its namespace".

6.34

Add to 14.4 [temp.explicit] paragraph 5:

A member class of a template class may be explicitly instantiated.

6.35 option 2

Replace 7.3.3 [namespace.udecl] paragraph 1, syntax with:

using-declaration:

using typename$_{opt}$ ::$_{opt}$ *nested-name-specifier unqualified-id* ;

using :: *unqualified-id* ;

Add to 14.7 [temp.param] paragraph 1:

typename followed by an *unqualified-id* names a template type parameter. **typename** followed by a *qualified-name* denotes the type in a nontype *parameter-declaration.*

6.36

Delete 14.2 [temp.res] paragraph 3, last sentence "The *qualified-name* shall include...".

6.37

Add to 14.2 [temp.res] paragraph 3:

The usual qualified name lookup (_basic.lookup.qual_) is used to find the *qualified-name* even in the presence of **typename**.

6.39 (different resolution to N0841 = 96-0023)

Replace 14.7 [temp.param] paragraph 2 with:

Default template arguments shall not be specified in a declaration or a definition of a function template. Default function arguments shall not be specified in the declaration or definition of an explicit specialization.

6.40

Add after 14.4 [temp.explicit] paragraph 5:

The usual access checking rules do not apply to explicit instantations. In particular, the template arguments and names used in the function declarator (e.g., including parameter types, return types, and exception specifications) may be private types or objects which would normally not be accessible and the template may be a member template or member function which would not normally be accessible.

6.41 (different resolution to N0841 = 96-0023)

Replace 14 [temp] paragraph 4 with:

A template, explicit specialization (14.5 [temp.spec]), or guiding declaration (14.10.5 [temp.over.spec]) shall not have C linkage. If the linkage of one of these is something other than C or C++, the behavior is implementation-defined.