Document: N0668/95-0068
Authors: John H. Spicer, Bill Gibbons
Date: March 8, 1995

Partial Ordering of Function Templates and Class Template Specialization

1. Partial Ordering of Function Templates

New subclause to follow 14.9.4 [temp.over.spec]

Given two function templates, whether one is more specific than another can be determined as follows:

The ordering of a pair of function templates, if any may be determined as follows: First transform the first template in the following way:

For each formal type parameter, substitute a unique type each place that parameter appears in the function parameter list.

For each nontype template parameter, substitute a unique value of the appropriate type each place it appears in the function parameter list.

Using the resulting types from the first function parameter list, perform argument deduction against the second function template as described in 14.9.2 [temp.deduct]. The first template is at least as specific as the second if, and only if, the deduction succeeds.

A template is more specific than another if, and only if, it is at least as specific as the other template and the other template is not at least as specific as the first.

Modifications to 13.2.3 [over.match.best]

After the second bullet of paragraph 1 add:

- F1 and F2 are template functions with the same signature, and the function template for F1 is more specialized than the function template for F2 according to the partial ordering rules described in x.y.z [xxx.yyy], or, if not that,

2. Class Template Specializations

New subclause to follow 14.5 [temp.spec]:

A primary class template declaration is one in which the class template name is an identifier. A template declaration in which the class template name is a template-id is a partial specialization of the class template named in the template-id. Specializations must be declared after the primary template has been declared.

For example:

```
template <class T1, class T2, int I> class A {};  // #1
template <class T, int I> class A<T, T*, I> {};  // #2
template <class T, int I> class A<T*, T, I> {};  // #3
template <class T> class A<int, T*, 5> {};  // #4
template <class T1, class T2, int I> class A<T1, T2*, I> {} // #5
```

The first declaration declares the primary (unspecialized) class template. The second declaration declares a specialization of the primary template.

The template parameters are specified in the angle bracket enclosed list that immediately follows the template keyword. A template also has a template argument list. For specializations this list is explicitly written immediately following the class template name. For primary templates, this list is implicitly described by the template parameter list (i.e., the template parameter names in the sequence in which they appear in the template parameter list).

For example, the template argument list for the primary template in the example above is "<T1, T2, I>".

A nontype argument is nonspecialized if it is the name of a nontype parameter. All other nontype arguments are specialized.

Within the argument list of a class template specialization, the following restrictions apply:

- a specialized nontype argument expression shall not involve a template parameter of the specialization
- the type of a specialized nontype argument shall not depend on a type parameter of the specialization
- the argument list of the specialization must not be identical to the implicit argument list of the primary template

Matching of Class Template Specializations

When a template class is used in a context that requires a complete instantiation of the class, it must be determined whether the instantiation is to be generated using the primary template or one of the partial specializations. This is done by matching the template arguments of the template class being used with the template argument lists of the partial specializations. If no matches are found, the instantiation is generated from the primary template. If exactly one matching specialization is found, the instantiation is generated from that specialization. If more than one specialization is found, one of them shall be more specific than any other matching specialization; the instantiation is generated from that specialization Otherwise, there is an ambiguity error.

the partial order rules are used to determine whether one of the specializations is "more specialized" than the others. If one of the specializations is not more specialized than all of the other matching specializations, then the template class reference is ambiguous and the program is ill-formed.

A specialization matches a given actual template argument list if the template arguments of the specialization can be deduced from the actual template argument list. The deduction of the specialization template parameters works as described in 14.9.2 [temp.deduct]. A nontype parameter can also be deduced from the value of an actual argument of a nontype parameter of the primary template.

```
A<int, int, 1> a1;   // Uses #1
A<int, int*, 1> a2;  // Uses #2, T=int, I=1
A<int, char*, 5> a3; // Uses #4, T=int
A<int, char*, 1> a4;  // Uses #5, T1=int, T2=char, I=1
```

```
A<int*, int*, 2> a5;  // Ambiguous – matches #2 and #3
```

Partial Ordering of Class Template Specializations

Given two class template partial specializations, whether one is more specific that another is can be determined as follows:

One template is at least as specialized as another if

- the type arguments of the first template's argument list are at least as specialized as those of the second template's argument list using the ordering rules for function templates, and
- each nontype argument of the first template's argument list is at least as specialized as that of the second template's argument list. A nontype argument is at least as specialized as another nontype argument if both are formal arguments, or the first is a value and the second is a formal argument, or both are the same value.

If only one of the two templates is at least as specialized as the other, that template is more specific than the other. Otherwise, the templates are considered unordered.