Bill Gibbons, John Spicer, Erwin Unruh

1. Issues from 95-0101/N701 accepted without change

Working paper changes for issue 2.24:

Add to 14.6 [temp.class.spec] paragraph 4:

> A partial specialization must be declared before any point at which it would be used were it declared.

Working paper changes for issue 2.27:

Change 14.12 [temp.friend] paragraph 2:

> A friend template may be defined within a class.
>
> ```
> class A {
>       template <class T> friend class B { /* ... */ };  // ok
>       template <class T> friend void f(T){ /* ... */ } // ok
> };
> ```

Working paper changes for issues 3.24, 3.25, 3.26, and 3.27:

Add before 14.10.2 [temp.deduct] paragraph 7:

> Note: the template type parameter cannot be deduced from the type of a nontype template argument.

Replace paragraph 9 with:

> [Note: a major array bound is not part of a function parameter type, except for reference types, so it can't be deduced from an argument:
>
> (update example as follows:)
> ```
> template <int I> void f3(int (&a)[I]);
> ...
> ```
> (in function g)
> ```
> int n[5];
> f3(n);  // ok
> ```

Replace paragraph 10 with:

> If a nontype parameter is used in an expression in the function declaration, template argument deduction fails. The type of the function template-parameter shall match the type of the template argument type exactly, except that a template parameter deduced from an array bound may be any integral type.

Add to 14.10.3 [temp.over] paragraph 1:

> If, for a given function template, argument deduction fails, no function is added to the set of candidate functions for that template.

Working paper changes for issue 4.11:

Add 14.6.3:

> 14.6.3 Member functions of class template specializations
>
> A member function of a class template specialization has a template parameter list that matches the template parameter list of the class template specialization and a template argument list that matches the template argument list of the class template specialization. A class template specialization is a distinct template. The members of the class template specialization are unrelated to the members of the primary template. Class template specialization members that are used in a way that requires a definition must be defined; the definitions of members of the primary template will never be used to provide definitions for members of a class template specialization. An explicit specialization of a member of a class template specialization is declared in the same way as an explicit specialization of the primary template.

```
// Primary template
template <class T, int I> struct A {
      void f();
};

template <class T, int I> void A<T,I>::f(){}

// Class template specialization
template <class T> struct A<T, 2> {
      void f();
      void g();
      void h();
};

// Member of class template specialization
template <class T> void A<T,2>::g(){}
// Explicit specialization
void A<char, 2>::h(){}

int main()
{
      A<char, 0> a0;
      A<char, 2> a2;
      a0.f();   // ok
      a2.g();   // ok
      a2.h();   // ok
      a2.f();...// Error A<T,2>::f is not defined
                // A<char,2>::f cannot be instantiated
                // primary template is not used.
}
```

Wording changes for issue 7.3.2:

Replace 14.3.2 [temp.point] paragraph 8 with:

> If a template for which a definition is in scope is used in a way that involves overload resolution, conversion to a base class, or pointer to member conversion, the definition of a template specialization is generated if the template is defined.

2. Issues from 95-0101/N0701 accepted with modifications

Wording changes for issue 2.25:

Modify section 14.3.2 [temp.point] paragraph 4, last sentence to read:

> An implementation shall not instantiate a function, member function, class, or member class that does not require instantiation.

Add a new paragraph after paragraph 5:

> A member class of a class template may be defined after the class template which declares it just as with nontemplate classes ([class.nest]). Such a member class must be defined before the first use which requires generation of a specialization.

```
template <class T> struct A {
      class B;
};
A<int>::B *b1;  // requires A to be defined, but not A::B
template <class T> class A<T>::B {};
A<int>::B b2;  // requires A::B to be defined
```

Wording changes for issue 2.26:

Add to 14.12 [temp.friend] after paragraph 2:

> A member of a class template may be declared to be a friend.

```
template <class T> struct A {
      struct B {};
      void f();
};

class C {
      template <class T> friend struct A<T>::B;
      template <class T> friend void A<T>::f();
};
```

Wording changes for issue 7.2 and 7.6:

Add to 14.10.2 [temp.deduct] paragraph 2,, sentence 4:

> or if for any parameter/argument pair the deduction leads to more than one possible set of deduced values,

Wording changes for issue 7.5:

Add to 14.10.2 [temp.deduct] paragraph 7, before example:

> and if the set of overloaded functions does not contain template functions

add to example:

```
template <class T> void foo(int, T);
```

```
f(&foo);  // type deduction fails: foo is a template
```