Doc. no.: J16/97-0008
WG21/N1046
Date: 28 January 1997
Project: Programming Language C++
Reply to: Beman Dawes
beman@esva.net

Libraries Issues List for CD2 - Version 0

History:

Version 0: Distributed at the start of the Nashua meeting.

Open Issues

Clause 17 - Library Introduction
Clause 18 - Language Support

Issue: **CD2-18-001 `Offset` macro needs additional restrictions**
Section: 18.1
Status: Open
Description:

The offsetof macro (18.1) is restricted to work on POD-structs and POD-unions. So far so good. Two problems:

1. A POD-struct is allowed to have static data and non-virtual member functions. Surely they should be explicitly excluded from use with the `offsetof` macro.

2. A POD-struct is allowed to have reference members, as of the July 1996 meeting. The usual implementation of the macro is this:

```
#define offsetof(s, m)  (size_t)(&(((s *)0)->m))
```

That doesn't work if 'm' is a reference member. I don't think that one ordinary macro can work for both reference and non-reference members. That means the compiler will have to resort to something like

```
#define offsetof(s,m) __builtin_offsetof(s, m)
```

where `__builtin_offsetof` is some compiler magic that does the right thing. Do we really want to make compilers jump through those hoops? I don't think you can do much with the value of the offset of a reference member anyway, so I suggest disallowing taking its offset.

Proposed Resolution:

Modify the 18.1 section referring to `offsetof` to say:

> "The result of applying the `offsetof` macro to a field which is a static data member, a function member, or which has a reference type is undefined."

"Undefined" will allow existing implementations to continue to be valid. If we require error detection, compilers will have to jump through hoops to recognize the `offsetof` macro, since by the time the macro is expanded the fact that invalid code came from "`offsetof`" is lost.

Requester: Steve Clamage <clamage@taumet.Eng.Sun.COM>
References: lib-5249

Clause 19 - Diagnostics
Clause 20 - Utilities
Clause 21 - Strings

Issue: **CD2-21-001 basic_string element**
Section: 21.3.4
Status: Open
Description:

This clause says that the reference returned by the non-const version of operator[] is invalid after "any subsequent call to c_str(), data(), or any non-const member function for the object." This would seem to make expressions such as

```
foo(s[a], s[b])
```

invalid, where s is not const, as the second call to operator[] would invalidate the reference returned by the first call to operator[]. In general, it seems unreasonable that a call to operator[] would invalidate the reference returned by a previous call to operator[].

Andrew Koenig questions in lib-5251 whether the following might be invalid:

```
s[I] = s[j];
```

Proposed Resolution:

Matt Austern discusses several possible resolutions in lib-5250.

Requester: Kevin S. Van Horn <kevin.s.vanhorn@iname.com>
References: lib-5248, lib-5250, lib-5251, lib-5252

Issue: **CD2-21-002 `basic_string` member require non-existent `traits::eos()`**
Section: 21.3.4 [lib.string.access], 21.3.6 [lib.string.ops] (2 places), 27.6.1.2.3 [lib.istream::extractors] (2 places), 27.6.2.7 [lib.ostream.manip]
Status: Open
Description:

Several basic_string member functions are defined to require traits::eos().

Unfortunately, character traits do not have an eos() member, either in the requirements table, or in the provided specializations.

Proposed Resolution:

Nathan Myers in lib-5247: "Yes, member eos() was removed; use char_type() as end-of-string where it is needed. We need to fix the Draft where it mentions eos()."

Requester: Hans-Juergen Boehm <boehm@mti.sgi.com>
References: lib-5245, lib-5247

Clause 22 - Localization
Clause 23 - Containers

Issue: **CD2-23-001 Priority_queue<> missing typedef for compare_type**
Section: 23.2.3.2 [lib.priority.queue]
Status: Open
Description:

std::priority_queue<> takes a template parameter "Compare", a function object, and defines a protected member with it, but there is no typedef for that parameter.

Proposed Resolution:

Add to the public interface of priority_queue<> in 23.2.3.2 [lib.priority.queue] the following definition:

```
typedef Compare compare_type;
```

Requester: Nathan Myers < ncm@cantrip.org>
References: lib-5246

Issue: CD2-23-002 Gratuitous pointer and const_pointer typedefs
Section: 23, 21
Status: Open
Description:

The standard containers provide pointer and const_pointer typedefs, but these do not appear in any requirement or function signature for any container, including basic_string.

Proposed Resolution:

Remove these typedefs.

Requester: Greg Colvin <Greg@imr.imrgold.com>
References:

Clause 24 - Iterators

Issue: CD2-24-001 Undefined lifetime of references from iterators.
Section: 24
Status: Open
Description:

Chapter 24 places no requirements on the lifetime of the reference returned by *iterator. For example, given a dereferenceable input iterator p on type int, must the following assertion be true?

```
const int& r = *p;
int i = r;
p++;
assert(i == r);
```

Proposed Resolution:

The assertion should not be required to be true. The *iterator operation might return a temporary.

Requester: Greg Colvin <Greg@imr.imrgold.com>
References:

Clause 25 - Algorithms
Clause 26 - Numerics
Clause 27 - Input/Output

Issue: CD2-27-001 Incorrect post condition for ios_base::failure
Section: 27.4.2.1.1 [lib.ios::failure]
Status: Open
Description:

The problem that existed with the other exception classes still exists in ios_base::failure (Nov '96 WP [lib.ios::failure]):

```
explicit failure(const string& msg);
```

Effects: Constructs an object of class failure, initializing the base class with `exception(msg)`.

Postcondition: `what() == msg.str()`

Proposed Resolution:

The postcondition needs to be changed to:

Postcondition: `strcmp(what(), msg.c_str()) == 0`

Requester: Kevlin Henney <kevlin@two-sdg.demon.co.uk>

References: