Doc. No.: X3J16/95-0105
WG21/N0705
Date: May 30, 1995
Project: Programming Language C++
Reply To: Mats Henricson
Ellemtel Telecom Systems Labs
mats.henricson@eua.ericsson.se

Clause 18 (Language support library) Issues List
Version 2

Revision History

Version 1 - February 1, 1995: Distributed in pre-Austin mailing.
Version 2 - May 30, 1995: Distributed in pre-Monterey mailing.

Introduction

This document is a summary of the issues identified in Clause 18. For each issue the status, a short description, and pointers to relevant reflector messages and papers are given. This evolving document will serve as a basis of discussion and historical for Language support library issues and as a foundation of proposals for resolving specific issues.

Issues

Work Group: Library Clause 18
Issue Number: 18-001
Title: Typedef typedef void fvoid_t(); not used anywhere
Sections: 18.1.2 [lib.stddef.types]
Status: closed

Description:
The first box in this chapter claims that this typedef is not used anywhere:

```
typedef void fvoid_t();
```

I think this is correct. I have done a global search through the previous CD and this is the only place where fvoid_t is mentioned (except for in tables 28 and 40 where it is just listed). This means that the first paragraph under 18.1.2 should be removed together with its code example, and the box itself.

Proposed Resolution:
Remove the typedef
Resolution: Typedef removed (Austin resolution)
Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:
However, Andy pointed out this problem to me:

> Removing the typedef would not be editorial, because it would change the meaning of a program that uses it.

Is the use of this typedef widespread?

Work Group: Library Clause 18
Issue Number: 18-002
Title: Redundant typedefs
Sections: 18.1.2 [lib.stddef.types]
Status: closed

Description:
The second box in this chapter claims that this typedef is redundant with what you find in <cstddef>:

```
typedef T ptrdiff_t;
```

The third box in this chapter claims that this typedef is redundant with what you find in <ctime>, <cstddef>, <cstdio>, and <cstring>:

```
typedef T size_t;
```

The fourth box in this chapter claims that this typedef is redundant with what you find in <cwchar> and <cwctype>:

```
typedef T wint_t;
```

This seems to be correct, so the involved paragraphs under 18.1.2 should be removed together with code examples and the boxes themselves.

Proposed Resolution:
Remove them.
Resolution: Removed (Austin resolution)

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:
Andy pointed out this problem to me:

> If ptrdiff_t is defined in two places, removing one of them is not editorial because a program could presumably include one and not the other.

The same problem holds for all the typedefs above.

Work Group: Library Clause 18
Issue Number: 18-003
Title: Call to set_new_handler() with null pointer
Sections: 18.4.1.1.1 operator new [lib.op.new]
Status: closed

Description:
I think it is still an issue what happens if the last call to set_new_handler() was a null pointer, i.e. should the infamous footnote be there or not?

Proposed Resolution:
Resolution: According to ANSI X3J16/95-0047, ISO WG21/N0647

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se

Emails: ?
Papers: ?
Discussion:
Andy agrees, and adds:
A programmer must either be entitled to assume that "new" never returns 0 or must not be entitled to assume it. I can see no middle ground possible.

I am not up to date with the discussion on this.

Work Group: Library Clause 18
Issue Number: 18-004
Title: Inherited members explicitly mentioned
Sections: 18.5.2.1 Class bad_cast [lib.bad.cast]
18.5.2.1.3 bad_cast::what [lib.bad.cast::what]
Status: closed

Description:
Why is bad_cast explicitly noted to have an inherited member function:

```
// virtual string what() const;
```

while the class bad_typeid is not, even though they inherit from the same base class logic_error? I think the comment noted above should be removed. It does not serve any purpose. The fact that public base class member functions are inherited should be known by all readers of the document.

Proposed Resolution:
Remove the comment
Resolution: The comment is removed (Austin resolution)

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:
I think these comments are quite widespread. If we decide to remove this comment, then I think we should do it throughout the document.

Work Group: Library Clause 18
Issue Number: 18-005
Title: Call to set_terminate() or set_unexpected() with null pointer
Sections: 18.6.1.2 set_terminate [lib.set.terminate]
18.6.2.2 set_unexpected [lib.set.unexpected]
Status: closed

Description:
What happens if set_terminate() or set_unexpected() is called with a null pointer as argument? "18.6.1.2 [lib.set.terminate]" claims:

```
f shall not be a null pointer.
```

What does that mean? The compiler cannot know beforehand, so is it implementation defined what happens? Or undefined?

Proposed Resolution:
Resolution: This is now stated explicitly as a "Requirement", which is defined as a precondition in 17.2.1.3.

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:

Work Group: Library Clause 18
Issue Number: 18-006
Title: <stdarg.h> and references
Sections: 18.7 Other runtime support [lib.support.runtime]
Status: active

Description:
It seems like we need a statement in the standard which describes the behavior of 'stdarg' when varargs are C++ types. This involves references, pointer to member, classes which are no PODS, (maybe others). The behavior can be chosen of ill-formed, undefined or defined behavior.

Part of the behavior is already given in [expr.call] 5.2.2. Here it is described that passing a non-POD class has undefined behavior.

Proposed Resolution:
Undefined behavior?
Resolution:

Requestor: Erwin Unruh, Erwin.Unruh@mch.sni.de
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: lib-848-855, 3725, 3728, 3729
Papers: None.
Discussion:

This is lib-3725:

Erwin have found out that we need some kind of wording in chapter 18 whether or not references should work with <cstdarg>. This was previously discussed in September 1993 (lib-848-855), but there was no conclusion, and also nothing mentioned in the CD. Erwin has included below the main arguments from 1993.

I think I would like to see some kind of wording along the line of what Jerry say in 851 below:

> My attitude is that to apply the stdargs macros all types and the type of parmN must be a "C type". That means a primitive type that is also in C or be a PODS.

Erwin today would like:

> I could live with an error or the requirement to the compiler to make it work. I would prefer the compiler doing it right.

While Jerry, at that time, preferred everything else to be undefined.

Comments? I will later put this on the Chapter 18 issues list.

Mats

********** Message c++std-lib-848

Subject: VA with a ref parmN

When parmN in a variable argumented function is a reference, some compilers, if not all, may do something confusing, though I believe one of the valid possibilities. That is, pick up the arguments from some area from the caller (say its stack, just after the object being referenced) rather than the space reserved for the ...'d arguments.

Since varargs requires more programmer intervention than I personally care for (from a user of the language point of view), and parmN is special, but references are not know in ANSI C, what is supposed to be the behavior, or is it purposely left unspecified (I've looked in what I believe to be the last working library spec and didn't see anything on it)?

********** Message c++std-lib-849

Some compilers warn about this problem already:

```
#include <stdarg.h>

void foo( C &r, ... )      // va_start will not work as expected
{
   int x;
   va_list ap;

   va_start( ap, r );      // address of 'r' is not necessarily
   for(;;) {               // related to the "..." parms
      x = va_arg( ap, int );
      if( x == 0 ) break;
      cout << x << endl;
   }
   va_end( ap );
}
```

It would be a good thing if we made this a "Common Warning" in something similar to Appendix E of the ISO/ANSI C standard since it almost always prevents a problem.

********** Message c++std-lib-850

I agree that we need to add words to the library draft in this area. The stdarg.h macros are too delicate to expose lightly to the extra semantics of C++. Either we ban reference parmNs (my preference) or we say what they mean.

I see a similar accident waiting to happen with the va_list argument to the various macros.

********** Message c++std-lib-851

My attitude is that to apply the stdargs macros all types and the type of parmN must be a "C type". That means a primitive type that is also in C or be a PODS.

I have a vague recollection that the library working group discussed this issue and reached that conclusion, but I don't know that it was ever explicitly incorporated in the working paper.

Everything else (including parmN being a reference type) is undefined.

********** Message c++std-lib-853

Is this going too far? Should passing pointer-to-member via the ellipsis be well-defined, too?

********** Message c++std-lib-854

How about implementation defined? If I know how my implementation works, then there should be nothing to prevent me from using the stdargs macros in an appropriate way for accessing references, pointers to members, etc.

********** Message c++std-lib-855

Why restricting the usage of variadic functions? The problems are in the C *implementation* of the stdarg macros. We can postulate that the Macros can be used with any C++-Type as long as the correct header was included.

It is then up to the implementation to supply correct macros for C++.

Are there any problems which cannot be solved?

Work Group: Library Clause 18
Issue Number: 18-007
Title: denormal_loss member
Sections: 18.2.1 Numeric limits [lib.limits]
Status: active

Description:
I would like another member added to the numeric_limits class (18.2.1.1):
static const bool denormal_loss= implementation-defined

with the following definition in 18.2.1.2:
true if loss of accuracy is detected as a denormalization loss (rather than as an inexact result). Need a footnote to refer to IEC 559.

Proposed Resolution:
Resolution:

Requestor: Fred Tydeman, tydeman@netcom.com
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion: From an email from Fred Tydeman:

Better support for ANSI/IEEE Std 754-1985, ANSI/IEEE Std 854-1987, and ISO/IEC 559.

...

Justification: The IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (ANSI/IEEE Std 854-1987) both have similar requirements on the detection of underflow. Quoting from 754:

7.4 Underflow. Two correlated events contribute to underflow. One is the creation of a tiny non-zero result between +/- 2 ** Emin which, because it is so tiny, may cause some other exception later such as overflow upon division. The other is extraordinary loss

of accuracy during the approximation of such tiny numbers by denormalized numbers. The implementor may choose how these events are detected, but shall detect these events in the same way for all operations. Tininess may be detected either

(1) After rounding -- when a nonzero result computed as though the exponent range were unbounded would lie strictly between +/- 2 ** Emin

(2) Before rounding -- when a nonzero result computed as though both the exponent range and the precision were unbounded would like strictly between +/- 2 ** Emin.

Loss of accuracy may be detected as either

(3) A denormalization loss -- when the delivered result differs from what would have been computed were exponent range unbounded.

(4) An inexact result -- when the delivered result differs from what would have been computed were both exponent range and precision unbounded

(This is the condition called inexact in 7.5).

Given the above requirement on underflow detection, C++ now does not have a means for a numeric's software writer to write software that will work on all four possible underflow implementations. C++ has tinyness_before which describes conditions (1) and (2). I am requesting that a new member (denormal_loss) be added to cover cases (3) and (4). With both members, than a writer of numeric software could tell what definition of underflow is being used on each particular hardware and taylor the numeric software accordingly.

I have been told that IEC-559 is the "same" as IEEE-754, but since I have never seen IEC-559, I do not know that for sure. This is why I quoted from IEEE-754 (of which I do have a copy).

Work Group: Library Clause 18
Issue Number: 18-008
Title: global operator new
Sections: 18.4 Dynamic memory management [lib.support.dynamic]
Status: active

Description:

The 'operator new' is now placed in the namespace std. This would prohibit using it with the syntax "int*p = ::new int;". This would call the global operator new.

Proposed Resolution:

Move the six functions with names

```
operator new              (two versions)
operator new[]            (two versions)
operator delete
operator delete[]
```

to the global namespace.

((Is this list complete???))

Resolution:

Requestor: Erwin Unruh, erwin.unruh@mch.sni.de
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: lib-3636, 3638-3643
Papers: None
Discussion:

>From lib-3636:

What is the reason for putting the operators in std. If it is just because 'everything is in there' you should re-consider it.

It has the similar problem of a user re-defining the operator. When redefining it the user would most probably use the global namespace.

I think the language support library may use the global namespace, since very few programs can live without it.

>From lib-3643:

And yes, there may be a core issue here - in a quick reading of 5.3.4 and 5.3.5 I can't find any description at all of the meaning of :: applied to new or delete. Maybe it is described elsewhere?

Work Group: Library Clause 18
Issue Number: 18-009
Title: whither exception?
Sections: 18 and 19
Status: active

Description:
Ask anybody where class exception ought to be, and they'd say, "In <exception>, of course!". There is such a header, and it contains an exception derived from exception, but not exception itself. I think the dependency should really run from <exception>, which is minimal (and part of the freestanding implementation) to <stdexcept>, which contains the library support exceptions.

Proposed Resolution:
Move class exception from <stdexcept> to <exception>, and from clause 19 to clause 18.
Resolution:

Requestor: Nathan Myers, myersn@roguewave.com
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: lib-3576
Papers: None.
Discussion:

Work Group: Library Clause 18
Issue Number: 18-010
Title: Exception specifications for class numeric_limits
Sections: 18.2.1.1, 18.2.1.2, 18.2.1.4
Status: active

Description:
I think all static member functions of class numeric_limits should have throw() as exceptions specification. This is also how I understand the editorial box at the end of 18.2.1.1 should be eliminated.

Proposed Resolution:
Add throw() to the following static member functions of numeric_limits:

```
static T min();
static T max();
static T epsilon();
static T round_error();
```

```
      static T infinity();
      static T quiet_NaN();
      static T signaling_NaN();
      static T denorm_min();
```

Also remove the editorial box at the end of 18.2.1.1.

Resolution:

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:

Work Group: Library Clause 18
Issue Number: 18-011
Title: Exception specifications for set_new_hander()
Sections: 18.4, 18.4.2.3
Status: active

Description:
I think set_new_hander() should have throw() as exception specification.

Proposed Resolution:
Add throw() as exception specification to set_new_hander().

Resolution:

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:

Work Group: Library Clause 18
Issue Number: 18-012
Title: Exception specifications for set_unexpected() and set_terminate()
Sections: 18.6, 18.6.1.3, 18.6.2.2
Status: active

Description:
I think set_unexpected() and set_terminate() should have throw() as exception specifications.

Proposed Resolution:
Add throw() as exception specification to set_unexpected() and set_terminate().

Resolution:

Requestor: Mats Henricson, mats.henricson@eua.ericsson.se
Owner: Mats Henricson, mats.henricson@eua.ericsson.se
Emails: None.
Papers: None.
Discussion:
