WG14/N443
X3J11/95-044

Date: 5 July 1995

From: ISO/IEC JTC1/SC22/WG14 (programming language C)

To: ISO/IEC JTC1/SC22/WG21 (programming language C++)

Subject: Review of C++ draft presented for CD balloting

INTRODUCTION

WG14 continues to provide useful feedback to WG21 on the draft C++ Standard submitted for balloting as a Committee Draft. As with our review during the CD registration ballot (1 February 1995), it was our hope and expectation that we could supply at this stage a cogent list of issues whose resolution would ensure maximum compatibility between our two closely related languages. Given the current state of the C++ draft, however, that important goal remains elusive:

* Substantial features still have no accompanying semantic description. The discussion of locales (clause 22), for example, is of particular interest to the C community and remains sorely lacking in explanatory detail.

* All too many substantive changes have been made that are not reflected in the resolutions published with the minutes of WG21 meetings. Change bars are too numerous to provide any guidance to areas that have suffered real change. It is thus hard to have any faith that portions of the document that have been nominally stable are truly left unchanged.

* The review period is woefully short. Many members of WG14 had only a few weeks to review a document with numerous changes since the last review.

* Many statements obviously intended as normative are in Notes subclauses, which are said to be non-normative. Conversely, quite a bit of commentary still masquerades as normative text, albeit largely toothless.

* The number of typographical errors and lurches in style continue to show that the document is nowhere near ready for the precise review required to determine whether compatibility between C and C++ has been adequately safeguarded.

As with our previous review, we supply here a simple compendium of comments made by various members of WG14. If the editing process continues past the July 1995 meeting -- as we fully expect -- WG14 will endeavor to supply additional comments as time permits. And as always, we stand ready to supply additional guidance and eview, to ensure that C and C++ remain ``as close as possible, but no closer."

UK Comments on C++ CD for Public Review

(I am afraid we have hardly scratched the surface.)

Clause 1.1

Paragraph 2, last sentence. Delete this sentence and Annex C.1.2. This is the first standard for C++, what happened prior to 1985 is not relevant to this document.

Clause 1.2

Paragraph 1, change "ISO/IEC 9899:1990, C Standard" to "ISO/IEC 9899:1990 Programming Languages -- C"

Paragraph 1, change "ISO/IEC 9899:1990/DAM 1, Amendment to C Standard" to "ISO/IEC:1990 Programming languages -- C AMENDMENT 1: C Integrity"

Add year of current publication of ISO/IEC 2382
Clause 1.3

Paragraph 1, multibyte character. Last sentence. What is the basic character set? Is it the basic source character set or basic execution character set (see clause 5.2.1 of ISO 9899)? There is an index refence for basic execution character set to this clause.

Also need to add definitions of the basic execution and basic source character set. See ISO 9899, Clause 5.2.1.

Paragraph 1, undefined behaviour. ISO 9899 states that "Undefined behaviouris otherwise indicated in this International Standard by the words "undefined behaviour" or by the omission of any explicit definition of behaviour".

The C++ standard should also adopt the rule that omission of explicit defintion of behaviour results in undefined behaviour.

Paragraph 1, well-formed program. Other standards use the term Conforming to describe this concept. The C++ standard should follow this precedent. It should also introduce the concept of Strict Conformance, that is a program that contains no undefined, implementation defined or unspecified behaviours.

Clause 1.5, paragraph 1, second sentence. Contains a use of the term "basic execution character set". See previous discussion.

Clause 1.8, paragraph 4. Need to include text stating that the standard imposes no requirements on the behaviour of programs that contain undefined behaviour.

Clause 1.8, paragraph 9, second sentence. What is a "needed side-effect"? This paragraph, along with footnote 3 appears to be a definition of the C standard "as-if" rule. This rule should be defined as such.

Clause 2.1, phase 8, first sentence. Change "The translation units that will form a program are combined." to "The translation units

are combined to form a program."

Clause 2.2, paragraph 1. Delete and replace with wording from C standard. "All occurrences in a source file of the following sequences of three characters (called trigraph sequences) are replaced with the corresponding single character. No other trigraph sequence exists. Each ? that does not begin one of the above trigraphs listed above is not changed."

Clause 2.3, paragraph 3, first sentence. Change "... lexically analsysed ..."
to "... parsed ...". To agree with wording in C standard.

Clause 2.3, paragraph 3, last sentence. Delete ", even if that would cause further lexical analysis to fail". To agree with existing, clear wording in C standard.

Clause 2.4. This is a gratuatous difference from the Addendum to the C standard with no technical merit. It should be deleted and replaced by the text from the Addendum.

Clause 2.8, paragraph 3. Reserving identifiers containing a double underscore is overly restrictive. Identifiers starting with double underscore should be reserved.

Clause 2.9.1, paragraph 1. This is a clumsy rewrite of the description in Clause 6.1.3.2 of the C standard. Replace by the text contained in the two paragraphs of the Description in Clause 6.1.3.2.

Clause 2.9.1, paragraph 2. This is a clumsy rewrite of the semantics in Clause 6.1.3.2 of the C standard. Replace by the text contained in the two paragraphs of the Semantics in Clause 6.1.3.2.

Clause 2.9.2, paragraph 1, second sentence. What is "the machine's character set"? Is this the basic source character set that we have forgotten to define? Suggest that the wording from C standard, Clause 6.1.3.4, Semantics, first paragraph be used (it contains the important concept of mapping).

Clause 2.9.2, paragraph 2. Suggest that C standard, Clause 6.1.3.4, Semantics, second paragraph be used as the basis of a rewrite of this paragraph.

Clause 2.9.2, paragraph 3. Suggest that C standard, Clause 6.1.3.4, Description, paragraph 2, 3, 4, and 5 be used as the basis of a rewrite of this paragraph.

Clause 2.9.2, paragraph 4. Ditto comment on paragraph 3.

Clause 2.9, paragraph 1. Suggest that this be replaced by C standard Clause 6.1.3.1, Description, paragraph 1. Otherwise the term "missing" should be replaced by "ommitted".

Clause 2.9.4. Suggest that paragraph 1, 2 and 3 be replaced by C standard, Clause 6.1.4, all paragraphs in Description and Semantics.

Clause 2.9.4, paragraph 4. Delete. The size of a string is not equal to the number of characters it contains. The \" rule is already covered by the text from the C standard. The first paragraph belongs in an introductory text to the language.

Clause 5.16, syntax rule. Change "assignment-expression" to "conditional-expression" to agree with the C standard, ISO 9899 Clause 6.3.15

Page 32 Para 9

This states :
Types bool, char, wchar_t, and the signed and unsigned integer types are collectively called integral types. 27) A synonym for integral type is integer type.

ISO 9899 does not include wchar_t as a member of the integral types, this should at least be noted in Annex C, and does raise a number of compatability issues

Page 84 Para 5

The underlying type of an enumeration is an integral type, not gratuitously
larger than int

Is this meant to be a requirement on an implementation ?

if so then the requirement should be stated positively.
i.e. an enumeration is an integral type that can represent all enumerator values .... otherwise remove the not gratuitously ...

1.7 Processor compliance para 2
typo -diagnosable errors repeated

Page 6 para 18
the word builtin needsd a hypen i.e built-in

Paragraph 3.3.4 Page 20
Scope

File 1
// First file
// declare i in global namespace as per page 20 of draft
// and has external linkage

```
int i=5;
```

File 2
//Second file

```
static int i = 10 ;  // declare i in global namespace with internal
linkage
int y = ::i ;   // What is the value of y
          // does :: resolve linkage to external or internal ??
```

```
void f(void)
{
      int i =6;
      int j =::i; // Global namespace i internal or external

}
```

If an implementation is required to accept both

```
int main(){}
```

and

```
int main(int argc, char * argv[]){}
```

Is it permitted to have a prototype of both forms visible ?

```
int main();
int main(int, char **);
```

If not is a disgnostic required nn this case.

Page 77

The following two statements appear to contradict each other

The inline specifier is a hint to the implementation that inline substitution of the function body is to be preferred to the usual function call implementation. The hint can be ignored.

The above statement clearly indicates that inline can be ignored however the draft goes on to state:

A function (8.3.5, 9.4, 11.4) defined within the class definition "is" inline.

Is an implementation free to ignore the inline within a class definition ?

Page 45 para 7 [expr.call]

This section describes the promotions prior to a function call and refers to section 4.5 (integral promotions), however section 4.5 refers to promotion of wchar_t and bool, paragraph 7 remains silent on wchar_t and bool leaving a question over whether promotion of these takes place prior to the function call.

The following are points directly relating to C.

Clause 1.1

Paragraph 2, last sentence. Delete this sentence and Annex C.1.2. This is the first standard for C++, what happened prior to 1985 is not relevant to this document.

Clause 1.2

Paragraph 1, change "ISO/IEC 9899:1990, C Standard" to "ISO/IEC 9899:1990 Programming Languages -- C"

Paragraph 1, change "ISO/IEC 9899:1990/DAM 1, Amendment to C Standard" to "ISO/IEC:1990 Programming languages -- C AMENDMENT 1: C Integrity"

Clause 1.3

Paragraph 1, multibyte character. Last sentence. What is the basic character set? Is it the basic source character set or basic execution character set (see clause 5.2.1 of ISO 9899)? There is an index refence for basic execution character set to this clause.

Also need to add definitions of the basic execution and basic source character set. See ISO 9899, Clause 5.2.1.

Paragraph 1, undefined behaviour. ISO 9899 states that "Undefined behaviour is otherwise indicated in this International Standard by the words "undefined behaviour" or by the omission of any explicit definition of behaviour". The C++ standard should also adopt the rule that omission of explicit defintion of behaviour results in undefined behaviour.

Paragraph 1, well-formed program. Other standards use the term Conforming to describe this concept. The C++ standard should follow this precedent. It should also introduce the concept of Strict Conformance, that is a program that contains no undefined, implementation defined or unspecified behaviours.

Clause 1.5, paragraph 1, second sentence. Contains a use of the term "basic execution character set". See previous discussion.

Clause 1.8, paragraph 9, second sentence. What is a "needed side-effect"? This paragraph, along with footnote 3 appears to be a definition of the C standard "as-if" rule. This rule should be defined as such.

Clause 2.3, paragraph 3, first sentence. Change "... lexically analsysed ..." to "... parsed ...". To agree with wording in C standard.

Clause 2.3, paragraph 3, last sentence. Delete ", even if that would cause further lexical analysis to fail". To agree with existing, clear wording in C standard.

Clause 2.4. This is a gratuatous difference from the Addendum to the C standard with no technical merit. It should be deleted and replaced by the text from the Addendum.

Clause 2.9.1, paragraph 1. This is a clumsy rewrite of the description in Clause 6.1.3.2 of the C standard. Replace by the text contained in the two paragraphs of the Description in Clause 6.1.3.2.

Clause 2.9.1, paragraph 2. This is a clumsy rewrite of the semantics in Clause 6.1.3.2 of the C standard. Replace by the text contained in the two paragraphs of the Semantics in Clause 6.1.3.2.

Clause 2.9.1, footnote 16. This statement of a well know fact is not need for the historical education of users of K&R C compilers.

Clause 2.9.2, paragraph 1, second sentence. What is "the machine's character set"? Is this the basic source character set that we have forgotten to define? Suggest that the wording from C standard, Clause 6.1.3.4, Semantics, first paragraph be used (it contains the important concept of mapping).

Clause 2.9.2, paragraph 2. Suggest that C standard, Clause 6.1.3.4, Semantics, second paragraph be used as the basis of a rewrite of this paragraph.

Clause 2.9.2, paragraph 3. Suggest that C standard, Clause 6.1.3.4, Description, paragraph 2, 3, 4, and 5 be used as the basis of a rewrite of this paragraph.

Clause 2.9.2, paragraph 4. Ditto comment on paragraph 3.

Clause 2.9, paragraph 1. Suggest that this be replaced by C standard Clause 6.1.3.1, Description, paragraph 1. Otherwise the term "missing" should be replaced by "ommitted".

Clause 2.9.4. Suggest that paragraph 1, 2 and 3 be replaced by C standard, Clause 6.1.4, all paragraphs in Description and Semantics.

Clause 2.9.4, paragraph 4. Delete. The size of a string is not equal to the number of characters it contains. The \" rule is already covered by the text from the C standard. The first paragraph belongs in an introductory text to teh language.

Clause 3.9, paragraph 6, last sentence. In ISO 9899 an incomplete type is not an object type (Clause 6.1.2.5, first paragraph). Defining an "incompletely-defined object type" is a needless incompatibility with ISO 9899. Use another term.

Clause 3.9, paragraph 7, last sentence. ISO 9899 allows a typedef declaration of an array of unknown size to be later completed for a specific object (Clause 6.5.7, example 6). C++ should also allow such a usage. Disallowing this construct is a needless incompatibility.

!!! indicates meatier comments.

3.6.2. The latitude with which static initialization might occur is problematic for use of the floating-point environment, viz. the floating-point exception flags and rounding direction modes required by IEC559. The sequence { clear-overflow-flag, compute, test-overflow-flag } would be defeated if the implementation chose to execute some overflowing static initializations between the clear and test. The sequence { set-special-rounding, compute, restore-usual-rounding } could affect the results of static initializations the implementation chose to execute between the set and restore. In order to support the floating-point environment, some implementations, depending on

their initialization model, might need to insulate static initialization with say { save-FP-environment, set-default-FP-environment, execute-initializations, restore-FP-environment }. A note to this effect would be helpful.

3.9.1, P10, Box 21. Yes, say "at least as much range and precision". Both are desired, and one doesn't imply the other.

5, P4. The first sentence may not be clear. I assume "where the operators really are" means the rearrangement in question would not change values. Better would be to disallow rearrangement (except by the as-if rule). "Rearrangement" is better than "regrouping", as the distributive law is problematic too.

!!! 5, P12. There's no mention of license for wide evaluation of floating expressions, as in 3.2.1.5 of the C standard. Wide evaluation is needed by the host of systems based on wide registers.

17.3.1.1, P10, Table 15. Typo: unititialized_fill

17.3.3.1.2, P1. This seems to say that a header can optionally declare or define any names it wishes. This statement may have been taken out of context from the C standard, where, I thought, the optional reserved names were confined to those in the subsequent bullets.

17.3.3.2, P1. Sentence is difficult to parse.

17.3.4.2, P1. Footnote says masking macros are disallowed. Why disallow them?

!!! 17. Assuming wide expression evaluation is allowed, math functions should be able to have return types appropriate to the implementation's expression evaluation method. E.g. if the minimum evaluation format is double, then cos should have the prototypes

```
double cos(float);
double cos(double);
long double cos(long double);
```

(Note this doesn't affect signatures.)

17.3.4.8, P3, Box 70. I think it's right to not require C functions to throw exceptions, but why prohibit it?

18.2.1.1. Is tinyness_before actually useful for any programming task? Being in the interface makes the diligent programmer worry about whether she needs to consider it. The IEEE 754 (IEC 559) standardization group regarded it as an implementation option that didn't matter to the user.

18.2.1.2, P23, 27. Footnote says these are equivalent to xxx_MIN_EXP and xxx_MAX_EXP, but their definitions don't imply that. Better to use the same wording as in the C standard.

18.2.1.2, P23, 25, 27, 29. These refer to "range", which is intended to imply normalized. "Range of normalized floating-point numbers", as in the C standard, would avoid the ambiguity.

18.2.1.2, P61. round_style would be more useful if its value reflected the

current execution-time rounding style, which can be changed dynamically on most systems, including all IEC559 ones.

18.2.1.4, P2. Example is inconsistent in that is_iec559 is true but has_denorm is false -- IEC559 requires denorms.

19.1. The hierarchy of exceptions is confusing. (1) What are the differences between domain_error, invalid_argument, and out_of_range? (2) out_of_range and range_error sound like the same thing but aren't. (3) In mathematics (though not the C standard), domain refers to argument values and range to return values, but here out_of_range refers to argument values. (4) How do they map to the IEC559 exceptions (invalid, overflow, underflow, div-by-zero, and inexact)?

19.1. I believe (and hope) there's not a requirement that builtin operators on builtin types or standard math functions throw any of these exceptions, but a reader might leap to the conclusion that they do.

!!! 26.2. The complex library provides a subset of the capabilities one might expect from builtin complex types. A description of what capabilities are and are not supported would be very helpful. What conversions? Which among complex<int>, complex<long>, complex<float>, and complex<double> have implicit conversions? What (mixed mode) operations? Do integer and complex operands mix (e.g. complex_z * 2)? Is double_complex_z * 2.0L OK? Without this description the reader must infer from the overloading rules. (It appears there are no implicit conversions from complex to real nor from wider to narrower among complex<long double>, complex<double>, and complex<float>, which presumably allows for automatic "promotions" from real to complex and from narrower to wider complex types. Saying so much -- whatever is correct -- would be helpful.)

!!! 26.2 In reviewing the complex library I'm further confounded by not being able to try it. It uses member templates, which aren't implemented in either of the two compilers I have access to. Are there enough implementations of this?

26.2 (and elsewhere). The lack of rationale makes review more difficult.

26.2, P1. Typo in the second divide operator.

26.2.1. What are the requirements for type X?

!!! 26.2.2. Compound assignments should be overloaded for real operands. This is CRITICAL for consistency with IEC559 and for efficiency (see section 2.3.6 of "Complex C Extensions", Chapter 6 of X3J11's TR on Numerical C Extensions), particularly since the binary operators are defined in terms of the compound assignments. complex_z *= 2.0 must not entail a conversion of 2.0 to complex.

26.2.2. Why initialize re and im to 0?.

26.2.3. How do the default arguments like T re = T() apply to builtin types like int?

26.2.4. The class declarations for the compound assignments use member templates, but they don't show up here. Likewise the complex(const

complex<X>&) constructor is missing.

!!! 26.2.5. Definitions for binary operators refer to compound assignments, but compound assignments aren't declared for complex<T> op= T. This is a deficiency in the compound assignments (see above). Also the semantics are wrong for T op complex<T>, as they entail a conversion of T to complex<T> (see above).

26.2.5. For ==, typo: lhsP.real

26.2.5. For ==, the Returns and Notes parts are awkward.

26.2.5. For !=, typo in Returns part.

26.2.6. abs is missing.

26.2.6. Can't review the two TBS.

26.2.6. I believe the term "norm" commonly refers to the square root of the squared magnitude (i.e. abs), and not the squared magnitude. Is a function for the squared magnitude needed? Note that the squared magnitude can be computed from abs with only deserved over/underflow, but not vise versa.

26.2.6. Typos in argument list for polar.

26.2.7. I don't think atan2 should be overloaded for complex arguments? How would it be defined?

26.2.7. log10(z) is easily computed as log(z)/log(10.0), so isn't really necessary.

!!! 26.2.7. Branch cuts and ranges need to be specified for functions. See section 3 of "Complex C Extensions", Chapter 6 of X3J11's TR on Numerical C Extensions.

26.5. There's no long double version of ldexp.

26.5. The float version of modf is out of alphabetical order.

26.5. pow doesn't accommodate mixed mode calls. E.g. pow(2.0f, 3.0) is ambiguous, matching both pow(float,float) and pow(float,int). pow(2.0, 3L) is ambiguous too. A description (clearer than the overloading rules) would be helpful. Maybe more overloads are desirable.

26.5. New overloads make math functions ambiguous for integer arguments, e.g. atan(1) would be ambiguous. C++ would be more restrictive than C in this respect. Of course, more overloads could solve the problem.

!!! 26.5. The functions in <fp.h> and <fenv.h>, specified in "Floating-Point C Extensions", Chapter 5 of X3J11's TR on Numerical C Extensions, support a substantially broader spectrum of numerical programming.

17.3.1.3:
A freestanding implementation doesn't include <stdexcept>,

which defines class exception, needed by <exception>.
Should probably move class exception to <exception>.

17.3.3.1:
A C++ program must be allowed to extend the namespace std if only
to specialize class numeric_limits.

17.3.4.1:
Paragraph 4 is a repeat.

18.2.1:
float_rounds_style should be float_round_style (correct once).

18.2.1.1:
Paragraph 2 is subsumed by the descriptions of radix, epsilon(),
and round_error(). Should be removed here.

18.2.1.1:
Paragraph 3 is repeated as 18.2.1.2, paragraph 50, where it belongs.
Should be removed here.

18.2.1.1:
Should say that numeric_limits<T> must be able to return T(0).
Should say that round_style defaults to round_indeterminate,
not round_toward_zero.

18.2.1.2:
denorm_min() does *not* return the minimum positive normalized value.
Should strike the mention of this function in paragraph 2.

18.2.1.2:
Paragraph 22 must supply a more precise definition of ``rounding error."

18.2.1.2:
Paragraph 23 must replace ``is in range" with
``is a normalized value".

18.2.1.2:
Paragraph 25 must replace ``is in range" with
``is a normalized value".

18.2.1.2:
Paragraph 27 must replace ``is in range" with
``is a finite value".

18.2.1.2:
Paragraph 29 must replace ``is in range" with
``is a finite value".

18.2.1.2:
In paragraph 41, ``flotaing" should be ``floating".

18.2.1.3:
Semantics must be specified for enum float_round_style.

18.5.1:

type_info::operator!=(const type_info&) is ambiguous
in the presence of the template operators in <utility>, and it is
unnecessary. It should be removed.

18.6.1.1:
Paragraph 1 incorrectly states that bad_exception is thrown by the
implementation to report a violation of an exception-specification.
Such a throw is merely a permissible option.

18.7:
There are five Table 28s.

19.1.1:
exception(const exception&) should not be declared with the
return type exception&. (Error repeated in semantic description.)

20.1:
Allocators are described in terms of ``memory models" which is an
undefined concept in Standard C++. The term should be *defined* here
as the collection of related types, sizes, etc. in Table 33 that
characterize how to allocate, deallocate, and access objects of
som managed type.

20.1:
Paragraph 3 talks about ``amortized constant time" for allocator
operations, but gives no hint about what parameter it should be
constant with respect to.

20.1:
a.max_size() is *not* ``the largest positive value of X::difference_type."
It is the largest valid argument to a.allocate(n).

20.1:
Table 33 bears little resemblance to the currently accepted version
of class allocator (though it should, if various bugs are fixed, as
described later.) Essentially *every* item in the `expression' column
is wrong, as well as all the X:: references elsewhere in the table.

20.3:
binder1st is a struct in the synopsis, a class later.
Should be a class uniformly, like binder2nd.

20.3.5:
class unary_negate cannot return anything. Should say that its
operator() returns !pred(x).

20.3.6.1:
binder1st::value should have type Operation::first_argument_type,
not argument_type.

20.3.6.3:
binder2nd::value should have type Operation::second_argument_type,
not argument_type.

20.3.7:
``Shall" is inappropriate in a footnote, within a comment, that

refers to multiple memory models not even recognized by the Standard.

20.4:
return_temporary_buffer shouldn't have a second (T*) parameter.
It's not in STL, it was not in the proposal to add it, and
it does nothing.

20.4.1:
allocator::types<T> shows all typedefs as private.
They must be declared public to be usable.

20.4.1:
It is not clear from Clause 14 whether explicit template member
class specializations can be first declared outside the containing
class. Hence, class allocator::types<void> should probably be declared
inside class allocator.

20.4.1:

The explicit specialization allocator::types<void> should include:
```
  typedef const void* const_pointer;
```
It is demonstrably needed from time to time.

20.4.1:
Footnote 169 should read ``An implementation,''
not ``In implementation.''

20.4.1.1:
allocator::allocate(size_type, types<U>::const_pointer) has no
semantics for the second (hint) parameter.

20.4.1.1:
allocator::allocate(size_type, types<U>::const_pointer) requires
that all existing calls of the form A::allocate(n) be rewritten
as al.allocate<value_type, char>(n, 0) -- a high notational
price to pay for rarely used flexibility. If the non-template form
of class allocator is retained, an unhinted form should
be supplied, so one can write al.allocate<value_type>(n).

20.4.1.1:
allocator::allocate(size_type, types<U>::const_pointer) should
return neither new T nor new T[n], both of which call the default
constructor for T one or more times. Note that deallocate, which
follows, calls operator delete(void *), which calls no destructors.
Should say it returns operator new((size_type)(n * sizeof (T))).

20.4.1.1:
allocator::max_size() has no semantics, and for good reason. For
allocator<T>, it knew to return (size_t)(-1) / sizeof (T) --
the largest sensible repetition count for an array of T. But the
class is no longer a template class, so there is no longer a T to
consult. Barring a general cleanup of class allocator, at the least
max_size() must be changed to a template function, callable as
either max_size<T>() or max_size(T *).

20.4.1.1:

A general cleanup of class allocator can be easily achieved by making it a template class once again:

```
template<class T> class allocator {
public:
   typedef size_t    size_type;
   typedef ptrdiff_t difference_type;
   typedef T*        pointer;
   typedef const T*  const_pointer;
   typedef T&        reference;
   typedef const T&  const_reference;
   typedef T         value_type;
   pointer address(reference x) const;
   const_pointer address(const_reference x) const;
   pointer allocate(size_type n);
   void deallocate(pointer p);
   size_type init_page_size() const;
   size_type max_size() const;
   };
```

The default allocator object for a container of type T would then be allocator<T>(). All of the capabilities added with the Nov. '94 changes would still be possible, and users could write replacement allocators with a *much* cleaner interface.

20.4.1.2:
operator new(size_t N, allocator& a) can't possibly return a.allocate<char, void>(N, 0). It would attempt to cast the second parameter to allocator::types<void>::const_pointer, which is undefined in the specialization allocator::types<void>. If related problems aren't fixed, the second template argument should be changed from void to char, at the very least.

20.4.1.2:
If allocator is made a template class once again, this version

of operator new becomes:

```
template<class T>
   void *operator new(size_t, allocator<T>& a);
```

20.4.1.3:
The example class runtime_allocator supplies a public member allocate(size_t) obvoously intended to mask the eponymous function in the base class allocator. The signature must be allocate<T, U>(size_t, types<U>::const_pointer) for that to happen, however. The example illustrates how easy it is to botch designing a replacement for class allocator, given its current complex interface. (The example works as is with the revised template class allocator described earlier.)

20.4.2:
raw_storage_iterator<OI, T>::operator*() doesn't return ``a reference to the value to which the iterator points.'' It returns *this.

20.4.3.1:
Template function allocate doesn't say how it should ``obtain a typed pointer to an uninitialized memory buffer of a given size.'' Should say that it calls operator new(size_t).

20.4.3.2:
Template function deallocate has no semantics. Should say that it calls operator delete(buffer).

20.4.3.5:
get_temporary_buffer fails to make clear where it ``finds the largest buffer not greater than ..." Do two calls in a row ``find" the same buffer? Should say that the template function allocates the buffer from an unspecified pool of storage (which may be the standard heap). Should also say that the function can fail to allocate any storage at all, in which case the `first' component of the return value is a null pointer.

20.4.3.5:
Strike second parameter to return_temporary_buffer, as before.
Should say that a null pointer is valid and does nothing.
Should also say that the template function renders indeterminate the value stored in p and makes the returned storage available for future calls to get_temporary_buffer.

20.4.4:
Footnote 171 talks about ``huge pointers" and type ``long long." Neither concept is defined in the Standard (nor should it be). This and similar comments desperately need rewording.

20.4.4.3:
Header should be ``uninitialized_fill_n", not ``uninitialized_fill."

20.4.5:
When template class auto_ptr ``holds onto" a pointer, is that the same as storing its value in a member object? If not, what can it possibly mean?

20.4.5:
auto_ptr(auto_ptr&) is supposed to be a template member function.

20.4.5:
auto_ptr(auto_ptr&) is supposed to be a template member function.

20.4.5:
auto_ptr<T>::operator= should return auto_ptr<T>&, not void, according to the accepted proposal.

20.4.5.1:
Need to say that auto_ptr<T>::operator= returns *this.

20.4.5.2:
auto_ptr<T>::operator->() doesn't return get()->m -- there is no m. Should probably say that ap->m returns get()->m, for an object ap of class auto_ptr<T>.

20.4.5.2:
auto_ptr<T>::release() doesn't say what it returns. Should say it returns the previous value of get().

20.4.5.2:
auto_ptr<T>::reset(X*) doesn't say what it returns, or that it deletes its current pointer. Should say it executes ``delete get()'' and returns its argument.

20.5:
The summary of <ctime> excludes the function clock() and the types clock_t and time_t. Is this intentional?

21.1:
template function operator+(const basic_string<T,tr,A> lhs, const_pointer rhs) should have a second argument of type const T *rhs.

21.1:
Paragraph 1 begins, ``In this subclause, we call...'' All first person constructs should be removed.

21.1.1.1:
string_char_traits::ne is hardly needed, given the member eq. It should be removed.

21.1.1.1:
string_char_traits::char_in is neither necessary nor sufficient. It simply calls is.get(), but it insists on using the basic_istream with the default ios_traits. operator>> for basic_string still has to call is.putback(charT) directly, to put back the delimiter that terminates the input sequence. char_in should be eliminated.

21.1.1.1:
string_char_traits::char_out isn't really necessary. It simply calls os.put(), but it insists on using the basic_ostream with the default ios_traits. char_out should be eliminated.

21.1.1.1:
string_char_traits::is_del has no provision for specifying a locale, even though isspace, which it is supposed to call, is notoriously locale dependent. is_del should be eliminated, and operator>> for strings should stop on isspace, using the istream locale, as does the null-terminated string extractor in basic_istream.

21.1.1.1:
string_char_traits is missing three important speed-up functions, the generalizations of memchr, memmove, and memset. Nearly all the mutator functions in basic_string can be expressed as calls to these three primitives, to good advantage.

21.1.1.2:
No explanation is given for why the descriptions of the members of template class string_char_traits are ``default definitions.'' If it is meant to suggest that the program can supply an explicit specialization, provided the specialization satisfies the semantics of the class, then the text should say so (here and several other places as well).

21.1.1.2:

string_char_traits::eos should not be required to return the result of the default constructor char_type() (when specialized). Either the specific requirement should be relaxed or the function should be eliminated.

21.1.1.2:
string_char_traits::char_in, if retained, should not be required to return is >> a, since this skips arbitrary whitespace. The proper return value is is.get().

21.1.1.2:
string_char_traits::is_del, if retained, needs to specify the locale in effect when it calls isspace(a).

21.1.1.3:
Paragraph 1 doesn't say enough about the properties of a ``char-like object." It should say that it doesn't need to be constructed or destroyed (otherwise, the primitives in string_char_traits are woefully inadequate). string_char_traits::assign (and copy) must suffice either to copy or initialize a char_like element.
The definition should also say that an allocator must have the same definitions for the types size_type, difference_type, pointer, const_pointer, reference, and const_reference as class allocator::types<charT> (again because string_char_traits has no provision for funny address types).

21.1.1.4:
The copy constructor for basic_string should be replaced by two

constructors:
basic_string(const basic_string& str);
basic_string(const_basic_string& str, size_type pos,
  size_type n = npos, Allocator& = Allocator());
The copy constructor should copy the allocator object, unless explicitly stated otherwise.

21.1.1.4:
basic_string(const charT*, size_type n, Allocator&) should be

required to throw length_error if n > max_size(). Should say:
Requires: s shall not be a null pointer
        n <= max_size()
Throws: length_error if n > max_size().

21.1.1.4:
basic_string(size_type n, charT, Allocator&) is required to throw

length_error if n == npos. Should say:
Requires: n <= max_size()
Throws: length_error if n > max_size().

21.1.16:
basic_string::size() Notes says the member function ``Uses traits::length(). There is no reason for this degree of overspecification. The comment should be struck.

21.1.1.6:
basic_string::resize should throw length_error for n >= max_size(), not n == npos.

21.1.1.6:
resize(size_type) should not have a Returns clause -- it's a void function. Clause should be labeled Effects.

21.1.1.6:
resize(size_type) should call resize(n, charT()), not resize(n, eos()).

21.1.16:
basic_string::resize(size_type) Notes says the member function ``Uses traits::eos(). It should actually use charT() instead. The comment should be struck.

21.1.1.6:
basic_string::reserve says in its Notes clause, ``It is guaranteed that..." A non-normative clause cannot make guarantees. Since the guarantee is important, it should be labeled differently. (This is one of many Notes clauses that make statements that should be normative, throughout the description of basic_string.)

21.1.1.8.2:
basic_string::append(size_type n, charT c) should return append(basic_string(n, c)). Arguments are reversed.

21.1.1.8.3:
basic_string::assign(size_type n, charT c) should return assign(basic_string(n, c)). Arguments are reversed.

21.1.1.8.4:
basic_string::insert(size_type n, charT c) should return insert(basic_string(n, c)). Arguments are reversed.

21.1.1.8.4:
basic_string::insert(iterator p, charT c) should not return p, which may well be invalidated by the insertion. It should return the new iterator that designates the inserted character.

21.1.1.8.4:
basic_string::insert(iterator, size_type, charT) should return void, not iterator. (There is no Returns clause, luckily.)

21.1.1.8.5:
basic_string::remove(iterator) says it ``calls the character's destructor" for the removed character. This is pure fabrication, since constructors and destructors are called nowhere else, for elements of the controlled sequence, in the management of the basic_string class. The words should be struck.

21.1.1.8.5:
basic_string::remove(iterator, iterator) says it ``calls the character's destructor" for the removed character(s). This is pure fabrication, since constructors and destructors are called nowhere else, for

elements of the controlled sequence, in the management of the basic_string class. The words should be struck.

21.1.1.8.5:
basic_string::remove(iterator, iterator) Complexity says ``the destructor is called a number of times ..." This is pure fabrication, since constructors and destructors are called nowhere else, for elements of the controlled sequence, in the management of the basic_string class. The Complexity clause should be struck.

21.1.1.8.6:
replace(size_type pos1, size_type, const basic_string&,...) Effects has the expression ``size() - &pos1." It should be ``size() - pos1."

21.1.1.8.6:
basic_string::replace(size_type, size_type n, charT c) should return replace(pos, n, basic_string(n, c)). Arguments are reversed.

21.1.1.8.8:
basic_string::swap Complexity says ``Constant time." It doesn't say with respect to what. Should probably say, ``with respect to the lengths of the two strings, assuming that their two allocator objects compare equal." (This assumes added wording describing how to compare two allocator objects for equality.)

21.1.1.9.1:
basic_string::find(const charT*, ...) Returns has a comma missing before pos argument.

21.1.1.9.8:
basic_string::compare has nonsensical semantics. Unfortunately, the last version approved, in July '94 Resolution 16, is also nonsensical in a different way. The description should be restored to the earlier version, which at least has the virtue

of capturing the intent of the original string class proposal:
1) If n is less than str.size() it is replaced by str.size().
2) Compare the smaller of n and size() - pos with traits::compare.
3) If that result is nonzero, return it.
4) Otherwise, return negative for size() - pos < n, zero for size() - pos == n, or positive for size() - pos > n.

21.1.1.10.3:
operator!=(const basic_string&, const basic_string&) is ambiguous in the presence of the template operators in <utility>, and it is unnecessary. It should be removed.

21.1.1.10.5:
operator>(const basic_string&, const basic_string&) is ambiguous in the presence of the template operators in <utility>, and it is unnecessary. It should be removed.

21.1.1.10.6:
operator<=(const basic_string&, const basic_string&) is ambiguous in the presence of the template operators in <utility>, and it is unnecessary. It should be removed.

21.1.1.10.7:
operator>=(const basic_string&, const basic_string&) is ambiguous in the presence of the template operators in <utility>, and it is unnecessary. It should be removed.

21.1.1.10.7:
operator>= with const charT* rhs should return lhs >= basic_string(rhs), not <=.

21.1.1.10.8:
Semantics of operator>> for basic_string are vacuous. Should be modeled after those for earlier string class.

21.1.1.10.8:
Semantics of operator<< for basic_string are vacuous. Should be modeled after those for earlier string class.

21.1.1.10.8:
getline for basic_string reflects none of the changes adopted by July '94 resolution 26. It should not fail if a line exactly fills, and it should set failbit if it *extracts* no characters, not if it *appends* no characters. Should be changed to match 27.6.1.3.

21.1.1.10.8:
getline for basic_string says that extraction stops when npos - 1 characters are extracted. The proper value is str.max_size() (which is less than allocator.max_size(), but shouldn't be constrained more precisely than that). Should be changed.

21.2:
There are five Table 44s.

21.2:
<cstring> doesn't define size_type. Should be size_t.

22.1:
template operator<<(basic_ostream, const locale&) as well as template operator>>(basic_ostream, const locale&) now have a second template argument (for ios traits) added without approval. While this change may be a good idea, it should be applied uniformly (which has not happened), and only after committee approval.

22.1.1:
locale::category is defined as type unsigned. For compatibility with C, it should be type int.

22.1.1:
class locale has the constructor locale::locale(const locale& other, const locale& one, category). I can find no resolution that calls for this constructor to be added.

22.1.1:
Example of use of num_put has silly arguments. First argument should be ostreambuf_iterator(s.rdbuf()).

22.1.1:
Paragraph 8 says that locale::transparent() has unspecified behavior when imbued on a stream or installed as the global locale. There is no good reason why this should be so and several reasons why the behavior should be clearly defined. The sentence should be struck.

22.1.1:
Paragraph 9 says that ``cach[e]ing results from calls to locale facet member functions during calls to iostream inserters and extractors, and in streambufs between calls to basic_streambuf::imbue, is explicitly supported." In the case of inserters and extractors, this behavior follows directly from paragraph 8. No need to say it again. For basic_streambuf, the draft can (and should) say explicitly that the stream buffer fixates on a facet at imbue time and ignores any subsequent changes that might occur in the delivered facet until the next imbue time (if then). (An adequate lifetime for the facet can be assured by having the basic_streambuf object memorize a copy of a locale object directly containing the facet, as well as a pointer to the facet, for greater lookup speed.) In any event, saying something ``is explicitly supported" doesn't make the behavior *required.* The paragraph should be struck, and words added to the description of basic_streambuf to clarify the lifetime of an imbued codecvt facet. (More words are needed here anyway, for other reasons.)

22.1.1.1.1:
Table 46 lists the ctype facets codecvt<char, wchar_t, mbstate_t> and codecvt<wchar_t, char, mbstate_t> as being essential, but what about codecvt<char, char, mbstate_t>? Should say that this facet must be present and must cause no conversion.

22.1.1.1.1:
Table 46, and paragraph 3 following, identify the facets that implement each locale category (in the C library sense). But these words offer no guidance as to what facets should be present in the default locale (locale::classic()). The template classes listed each represent an unbounded set of possible facets. Should list the following explicit instantiations of the templates as being required, along

with those explicit instantiations already listed in Table 46:
num_get<char, istreambuf_iterator<char> >
num_get<wchar_t, istreambuf_iterator<wchar_t> >
num_put<char, ostreambuf_iterator<char> >
num_put<wchar_t, ostreambuf_iterator<wchar_t> >
money_get<char, istreambuf_iterator<char> >
money_get<wchar_t, istreambuf_iterator<wchar_t> >
money_put<char, ostreambuf_iterator<char> >
money_put<wchar_t, ostreambuf_iterator<wchar_t> >
time_get<char, istreambuf_iterator<char> >
time_get<wchar_t, istreambuf_iterator<wchar_t> >
time_put<char, ostreambuf_iterator<char> >
time_put<wchar_t, ostreambuf_iterator<wchar_t> >

22.1.1.2:
As mentioned earlier, locale::locale(const locale&, const locale&, category) has been added without approval. It should be struck.

22.1.1.3:

Description of locale::use() Effects contains a nonsense statement:
``Because locale objects are immutable, subsequent calls to use<Facet>()
return the same object, regardless of changes to the global locale."
If a locale object is immutable, then changes to the global locale
should *always* shine through, for any facet that is not present
in the *this locale object. If the intent is to mandate cacheing
semantics, as sketched out in the original locales proposal, this
sentence doesn't quite succeed. Nor should it. Cacheing of facets
found in the global locale leads to horribly unpredictable behavior,
is unnecessary, and subverts practically any attempt to restoee
compatibility with past C++ practice and the current C Standard.
The sentence should be struck.

22.1.1.3:
Description of locale::use Notes uses the term ``value semantics"
and the verb ``to last." Are either of these terms defined within
the Standard? The sentence should be reworded, or struck since it's
non-normative anyway.

22.1.1.5:
locale::transparent() Notes says ``The effect of imbuing this locale
into an iostreams component is unspecified." If this is a normative
statement, it doesn't belong in a Notes clause. And if it's intended
to be normative, it should be struck. Imbuing a stream with
locale::transparent() is the *only* way to restore the behavior
of iostreams to that in effect for *every* C++ programming running
today. It is also essential in providing compatible behavior with
the C Standard. The sentence should be struck.

22.2.1.3.3:
ctype<char> describes this subclause as ``overridden virtual functions,"
but they're not. A template specialization has nothing to do with any
virtuals declared in the template. Should be renamed.

22.2.1.4:
Description of codecvt, paragraph 3, fails to make clear how an
implementation can ``provide instantiations for <char, wchar_t,
mbstate_t> and <wchar_t, char, mbstate_t>." Must specializations
be written for the template? If so, must they also have virtuals
that do the actual work? Or can the implementation add to the
default locale facets derived from the template, overriding the
virtual do_convert? Needs to be clarified.

22.2.1.4:
Implementations should also be required to provide an instantiation
of codecvt<char, char, mbstate_t> which transforms characters one
for one (preferably by returning noconv). It is needed for
the very common case, basic_filebuf<char>.

22.2.1.4.2:
codecvt::do_convert uses pointer triples (from, from_next, from_end)
and (to, to_next, to_end) where only pairs are needed. Since the
function ``always leaves the from_next and to_next pointers pointing

one beyond the last character successfully converted," the function must be sure to copy from to from_next and to to to_next early on. A better interface would eliminate the from and to pointers.

22.2.1.4.2:
codecvt::do_convert says ``If no translation is needed (returns noconv), sets to_next equal to argument to." The previous paragraph strongly suggests that the function should also set from_next to from. Presumably, the program will call do_convert once, with nothing to convert. If it returns noconv, the program will omit future calls to do_convert. If that is the intended usage, then it should be permissible to call any instance of do_convert with (mostly) null pointers, to simplify such enquiries -- and the wording should make clear how to make such a test call.

22.2.1.4.2:
codecvt::do_convert Notes says that the function ``Does not write into *to_limit." Since there is no requirement that converted characters be written into sequentially increasing locations starting at to, this is largely toothless. Effects clause should be written more precisely.

22.2.1.4.2:
codecvt::do_convert Returns says that the function returns partial if it ``ran out of space in the destination." But the function mbrtowc, for example, can consume the remaining source characters, beyond the last delivered character, and absorb them into the state. It would be a pity to require do_convert to undo this work. Should say that partial can also mean the function ran out of source characters partway through a conversion. Then clarify that, after a return of partial, the next call to do_convert should begin with any characters between from_next and from_end, of which there might be none.

22.2.2.1:
Template class num_get defines the type ios as basic_ios<charT>, which it then uses widely to characterize parameters. Thus, this facet can be used *only* with iostreams classes that use the default traits ios_traits<charT>. Since the use of num_get is mandated for *all* basic_istream classes, this restriction rules out essentially any substitution of traits. Best fix is to make the ios parameter an ios_base parameter on all the do_get calls, then change ios accordingly. This is sufficient if setstate is moved to ios_base as proposed elsewhere. But it requires further fiddling for num_put if fill is moved *out* of ios_base, as also proposed. Must be fixed, one way or another.

22.2.2.2:
Template class num_put defines the type ios as basic_ios<charT>, which it then uses widely to characterize parameters. Thus, this facet can be used *only* with iostreams classes that use the default traits ios_traits<charT>. Since the use of num_put is mandated for *all* basic_ostream classes, this restriction rules out essentially any substitution of traits. Best fix is to make the ios parameter an ios_base parameter on all the do_put calls, then change ios accordingly. This is sufficient if

setstate is moved to ios_base as proposed elsewhere. But it requires further fiddling for num_put if fill is moved *out* of ios_base, as also proposed. Must be fixed, one way or another.

22.2.3.1:
The syntax specified for numeric values is out of place in numpunct.

22.2.3.1:
Description of numpunct says, ``For parsing, if the digits portion contains no thousands-separators, no grouping constraint is applied." This suggests that thousands-separators are permitted in an input sequence, and that the grouping constraint is applied, but it is profoundly unclear on how this might be done. Allowing thousands-separators at all in input is risky -- requiring that grouping constraints be checked is an outrageous burden on implementors, for a payoff of questionable utility and desirability. Should remove any requirement for recognizing thoudands-separators and grouping on input. And the effect on output needs considerable clarification.

22.2.4:
Template classes collate, time_get, time_put, money_get, money_put, money_punct, messages, and their support classes still have only sketchy semantics -- over a year after they were originally accepted into the draft. They are based on little or no prior art, and they present specification problems that can be addressed properly only with detailed descriptions, which do not seem to be forthcoming. Even if adequate wording were to magically appear on short notice, the public still deserves the courtesy of a proper review. For all these reasons, and more, the remainder of clause 22 from this point on should be struck.

22.2.7.1:
messages_base::THE_POSIX_CATALOG_IDENTIFIER_TYPE is not a defined type.

27.1.1:
The definition of ``character" is inadequate. It should say that it is a type that doesn't need to be constructed or destroyed, and that a bitwise copy of it preserves its value and semantics. It should also say that it can't be any of the builtin types for which conflicting inserters are defined in ostream or extractors are defined in istream.

27.1.2.4:
Description of type POS_T contains many awkward phrases. Needs rewriting for clarity.

27.1.2.4:
Paragraph 2 has ``alg" instead of ``all."

27.1.2.4:
Footnote 207 should say ``for one of" instead of ``for one if."
Also, it should``whose representation has at least" instead of ``whose representation at least."

27.2:

Forward declarations for template classes basic_ios, basic_istream, and basic_ostream should have two class parameters, not one. It is equally dicey to define ios, istream, etc. by writing just one parameter for the defining classes. All should have the second parameter supplied, which suggests the need for a forward reference to template class ios_char_traits as well, or at least the two usual specializations of that class.

27.3:
<iostream> is required to include <fstream>, but it contains no overt references to that header.

27.3.1:
cin.tie() returns &cout, not cout.

27.3.2:
win.tie() returns &cout, not cout.

27.4:
streamsize is shown as having type INT_T, but subclause 27.1.2.2 says this is the integer form of a character (such as int/wint_t). streamsize really must be a synonym for int or long, to satisfy all constraints imposed on it. (See Footnote 211.)

27.4:
Synopsis of <ios> is missing streampos and wstreampos. (They appear in later detailed semantics.) Should be added.

27.4:

Synopsis of <ios> has the declaration:
template <class charT> struct ios_traits<charT>;
The trailing <charT> should be struck.

27.4.1:
Type wstreamoff seems to have no specific use. It should be struck.

27.4.1:
Type wstreampos seems to have no specific use. It should be struck.

27.4.2:
ios_traits::state_type is listed as ``to be specified."
It needs to be specified.

27.4.2:
Definition of ios_traits lists arguments backwards for is_whitespace. Should have const ctype<char_type>& second, as in later description. (Also, first argument should be int_type, as discussed in 27.4.2.3.)

27.4.2:
ios_traits description should make clear whether user specialization is permitted. If it isn't, then various operations in <locale> and string_char_traits are rather restrictive. If it is, then the draft should be clear that ios_traits<char> and ios_traits<wchar_t> cannot be displaced by a user definition.

27.4.2:
The draft now says ``an implementation shall provide" instantiations of ios_traits<char> and ios_traits<wchar_t>. It was changed without approval from ``an implementation may provide." This change directly contradicts Nov 94 Resolution 23. The proper wording should be restored.

27.4.2.2:
ios_traits::not_eof should take an argument of int_type, not char_type.

27.4.2.2:
ios_traits::not_eof says nothing about the use made of its argument c. Should say that it returns c unless it can be mistaken for an eof().

27.4.2.2:
ios_traits::not_eof has two Returns clauses. The second is an overspecification and should be struck.

27.4.2.2:
ios_traits::length has an Effects clause but no Returns clause. The Effects clause should be reworded as a Returns clause.

27.4.2.3:
First argument to is_whitespace has been changed from int_type to char_type with no enabling resolution. It is also a bad idea. Should be restored to int_type.

27.4.2.3:
is_whitespace supposedly behaves ``as if it returns ctype.isspace(c)," but that function doesn't exist. Should say ``as if it returns ctype.is(ctype_base::space, c)."

27.4.2.3:
The draft now says that ios_traits functions to_char_type, to_int_type, and copy are ``provided from the base struct string_char_traits<CHAR-T>." This is a substantive change made without approval. It is also nonsensical, since there is no such ``base struct." The wording should be struck.

27.4.2.4:
ios_traits::to_char_type has an Effects clause which should be reworded as a Returns clause.

27.4.2.4:
ios_traits::to_int_type has an Effects clause which should be reworded as a Returns clause.

27.4.2.4:
ios_traits::copy has an Effects clause which should be reworded as a Returns clause. (It returns src.)

27.4.2.4:
ios_traits::get_state should be specified to do more than return zero. Semantics are inadequate. A pos_type conceptually has three components: an off_type (streamsize), an fpos_t, and a state_type (mbstate_t, which may be part of fpos_t). It must be possible

to compose a pos_type from these elements, in various combinations, and to decompose them into their three parts.

27.4.2.4:
ios_traits::get_pos should be specified to do more than return pos_type(pos). Semantics are inadequate. See comments on get_state. above.

27.4.3:
ios_base::fill() cannot return int_type because it's not defined. Should be int if fill() is left in ios_base.

27.4.3:
ios_base::precision() and width() should deal in streamsize arguments and return values, not int. (Even more precisely, they should be moved to basic_ios and have all their types changed to traits::streamoff.)

27.4.3.1.6:
~Init() should call flush() for wout, werr, and wlog, not just for cout, cerr, and clog.

27.4.3.2:
ios_base::fill(int_type) cannot receive or return int_type because it's not defined. Both should be int if fill(int_type) is left in ios_base.

27.4.3.4:
ios_base::iword allocates an array of long, not of int.

27.4.3.4:
ios_base::iword Notes describes a normative limitation on the lifetime of a returned reference. It should not be in a Notes clause. It should also say that the reference becomes invalid after a copyfmt, or when the ios_base object is destroyed.

27.4.3.4:
ios_base::pword Notes describes a normative limitation on the lifetime of a returned reference. It should not be in a Notes clause. It should also say that the reference becomes invalid after a copyfmt, or when the ios_base object is destroyed.

27.4.3.5:
Protected constructor ios_base::ios_base() must *not* assign initial values to its member objects as indicated in Table 72. That operation must be deferred until basic_ios::init is called. Should say here that it does no initialization, then move Table 72 to description of basic_ios::init (27.4.4.1). Also should emphasize that the object *must* be initialized before it is destroyed (thanks to reference counting of locale objects).

27.4.3.5:
Table 72 shows result of rdstate() for a newly constructed ios_base object, but that object defines no such member function. (Will be fixed if table is moved to basic_ios, as proposed.)

27.4.4.1:
basic_ios::basic_ios() has next to no semantics. Needs to be

specified:
Effects: Constructs an object of class basic_ios, leaving its member objects uninitialized. The object *must* be initialized by calling init(basic_streambuf *sb) before it is destroyed.

27.4.4.1:
basic_ios::init(basic_streambuf *sb) has no semantics. Needs to be specified:
Postconditions: rdbuf() == sb, tie() == 0, ios_base initialized according to Table 72 (currently in 27.4.3.5).

27.4.4.2:
basic_ios::tie is not necessarily synchronized with an *input* sequence. Can also be used with an output sequence.

27.4.4.2:
basic_ios::imbue(const locale&) should call rdbuf()->pubimbue(loc) only if rdbuf() is not a null pointer. Even better, it should not call rdbuf()->pubimbue(loc) at all. Changing the locale that controls stream conversions is best decoupled from changing the locale that affects numeric formatting, etc. Anyone who knows how to imbue a proper pair of codecvt facets in a streambuf won't mind having to make an explicit call.

27.4.4.2:
basic_ios::imbue(const locale&) doesn't specify what value it returns. Should say it returns whatever ios_base::imbue(loc) returns.

27.4.4.2:
basic_ios::copyfmt should say that both rdbuf() and rdstate() are left unchanged, not just the latter.

27.5.2:
basic_streambuf::sgetn should return streamsize, not int_type

27.5.2:
basic_streambuf::sungetc should return int_type, not int

27.5.2:
basic_streambuf::sputc should return int_type, not int

27.5.2:
basic_streambuf::sputn should return streamsize, not int_type

27.5.2.2.3:
In in_avail Returns: gend() should be egptr() and gnext() should be gptr().

27.5.2.2.3:
basic_streambuf::sbumpc Returns should not say the function converts *gptr() to char_type. The function returns the int_type result of the call.

27.5.2.2.3:
basic_streambuf::sgetc Returns should not say the function converts *gptr() to char_type. The function returns the int_type result of the call.

27.5.2.2.3:
basic_streambuf::sgetn should return streamsize, not int.

27.5.2.2.4:
basic_streambuf::sungetc should return int_type, not int.

27.5.2.2.4:
basic_streambuf::sputc should return int_type, not int.

27.5.2.2.5:
basic_streambuf::sputc does not return *pptr(), which points at storage with undefined content. It returns traits::to_int_type(c).

27.5.2.4.2:
basic_streambuf::sync now requires that buffered input characters ``are restored to the input sequence." This is a change made without approval. It is also difficult, or even impossible, to do so for input streams on some systems, particularly for interactive or pipelined input. The Standard C equivalent of sync leaves input alone. Posix *discards* interactive input. This added requirement is none of the above. It should be struck.

27.5.2.4.3:
basic_streambuf::showmanyc Returns has been corrupted. The function should return the number of characters that can be read with no fear of an indefinite wait while underflow obtains more characters from the input sequence. traits::eof() is only part of the story. Needs to be restored to the approved intent. (See footnote 218.)

27.5.2.4.3:
basic_streambuf::showmanyc Notes says the function uses traits::eof(). Not necessarily true.

27.5.2.4.3:
Footnote 217 is nonsense unless showmany is corrected to showmanyc.

27.5.2.4.3:
basic_streambuf::underflow has two Returns clauses. Should combine them to be comprehensive.

27.5.2.4.3:
basic_streambuf::uflow default behavior ``does" gbump(1), not gbump(-1). It also returns the value of *gptr() *before* ``doing" gbump.

27.5.2.4.3:
basic_streambuf::uflow has a nonsense Returns clause. Should be struck.

27.5.2.4.4:
basic_streambuf::pbackfail argument should be int_type, not int.

27.5.2.4.4:
basic_streambuf::pbackfail Notes begins a sentence with ``Other calls shall.'' Can't apply ``shall'' to user program behavior, by the accepted conformance model.

27.6:
<iomanip> synopsis has includes for <istream> and <ostream>, but none of the declarations appear to depend on either of these headers. They should be replaced by an include for <ios>.

27.6:
<iomanip> does *not* define a single type smanip. Rather, it defines at least two different types which depend on the type of the function argument. Should probably say that each function returns some unspecified type suitable for inserting into an arbitrary basic_ostream object or extracting from a basic_istream object.

27.6.1.1:
basic_istream::seekg(pos_type&) and basic_istream::seekg(off_type&, ios_base::seekdir) should both have const first parameters.

27.6.11:
basic_istream paragraph 2 says extractors may call rdbuf()->sbumpc(), rdbuf()->sgetc(), or ``other public members of istream except that they do not invoke any virtual members of rdbuf() except uflow().'' This is a constraint that was never approved. Besides, rdbuf()->sgetc() invokes underflow(), as does uflow() itself, and the example of ipfx in 27.6.1.1.2 uses rdbuf()->sputbackc(). The added constraint should be struck.

27.6.1.1:
basic_istream definition, paragraph 4 is confusing, particularly in the light of similar errors in 27.6.2.1 and 27.6.2.4.2 (basic_ostream). It says, ``If one of these called functions throws an exception, then unless explicitly noted otherwise the input function calls setstate(badbit) and if badbit is on in exception() rethrows the exception without completing its actions.'' But the setstate(badbit) call may well throw an exception itself, as is repeatedly pointed out throughout the draft. In that case, it will not return control to the exception handler in the input function. So it is foolish to test whether badbit is set -- it can't possibly be. Besides, I can find no committee resolution that calls for exceptions() to be queried in this event.
An alternate reading of this vague sentence implies that setstate should rethrow the exception, rather than throw ios_base::failure, as is its custom. But the interface to setstate provides no way to indicate that such a rethrow should occur, so these putative semantics cannot be implemented.
The fix is to alter the ending of the sentence to read, ``and if setstate returns, the function rethrows the exception without completing its actions.'' (It is another matter to clarify what is meant by ``completing its actions.'')

27.6.1.1.2:

basic_istream::ipfx Notes says the second argument to traits::
is_whitespace is ``const locale *''. The example that immediately
follows makes clear that it should be ``const ctype<charT>&''.

27.6.1.1.2:
Footnote 222 makes an apparently normative statement in
a non-normative context.

27.6.1.2.1:
basic_istream description is silent on how void* is converted.
Can an implementation use num_get<charT>::get for one of the
integer types? Must it *not* use this facet? Is a version of
get missing in the facet? Needs to be clarified.

27.6.1.2.1:
Example of call to num_get<charT>::get has nonsense for first two
arguments. Instead of ``(*this, 0, '' they should be
be ``(istreambuf_iterator<charT>(rdbuf()),
istreambuf_iterator<charT>(0), ''

27.6.1.2.1:
Example of numeric input conversion says ``the conversion occurs
`as if' it performed the following code fragment.'' But that
fragment contains the test ``(TYPE)tmp != tmp'' which often has
undefined behavior for any value of tmp that might yield a true
result. The test should be replaced by a metastatement such as
``<tmp can be safely converted to TYPE>''. (Then num_get needs
a version of get for extracting type float to make it possible
to write num_get in portable C++ code.)

27.6.1.2.1:
Paragraph 4, last sentence doesn't make sense. Perhaps ``since the
flexibility it has been...'' should be, ``since for flexibility
it has been...'' But I'm not certain. Subsequent sentences are
even more mysterious.

27.6.1.2.1:
Use of num_get facets to extract numeric input leaves very unclear
how streambuf exceptions are caught and properly reported. 22.2.2.1.2
makes clear that the num_get::get virtuals call setstate, and hence
can throw exceptions *that should not be caught* within any of the
input functions. (Doing so typically causes the input function to
call setstate(badbit), which is *not* called for as part of reporting
eof or scan failure. On the other hand, the num_get::get virtuals
are called with istreambuf_iterator arguments, whose very constructors
might throw exceptions that need to be caught. And the description of
the num_get::get virtuals is silent on the handling of streambuf
exceptions.
So it seems imperative that the input functions wrap each call to
a num_get::get function in a try block, but doing so will intercept
any exceptions thrown by setstate calls within the num_get::get
functions.
A related problem occurs when eofbit is on in exceptions and the
program attempts to extract a short at the very end of the file.
If num_get::get(..., long) calls setstate, the failure exception
will be thrown before the long value is converted and stored in

the short object, which is *not* the approved behavior.
The best fix I can think of is to have the num_get::get
functions return an ios_base::iostate mask which specifies what
errors the caller should report to setstate. The ios& argument
could be a copy of the actual ios for the stream, but with
exceptions cleared. The num_get::get functions can then continue
to call setstate directly with no fear of throwing an exception.
But all this is getting very messy for such a time critical
operation as numeric input.

27.6.1.2.2:
basic_istream::operator>>(char_type *) extracts an upper limit of
numeric_limits<int>::max() ``characters." This is a silly and
arbitrary number, just like its predecessor INT_MAX for
characters of type char. A more sensible value is size_t(-1) /
sizeof (char_type) - 1. Could just say ``the size of the largest
array of char_type that can also store the terminating null."
basic_istream::operator>>(bool&) has nonsense for its first
two arguments. Should be
  istreambuf_iterator<charT, traits>(rdbuf()),
  istreambuf_iterator<charT, traits>(0), etc.

27.6.1.2.2:
basic_istream::(bool& paragraph 3 describes the behavior of
num_get::get. Description belongs in clause 22.

27.6.1.2.2:
basic_istream::operator>>(unsigned short&) cannot properly check
negated inputs. The C Standard is clear that -1 is a valid field,
yielding 0xffff (for 16-bit shorts). It is equally clear that
0xffffffff is invalid. But num_get::get(... unsigned long&)
delivers the same bit pattern for both fields (for 32-bit
longs), with no way to check the origin. One fix is to have
the extractor for unsigned short (and possibly for unsigned int)
pick off any '-' flag and do the checking and negating properly,
but that precludes a user-supplied replacement for the num_get
facet from doing some other magic. Either the checking rules
must be weakened over those for Standard C, the interface to
num_get must be broadened, or the extractor must be permitted
to do its own negation.

27.6.1.2.2:
basic_istream::operator>>(basic_streambuf *sb) now says, ``If sb is
null, calls setstate(badbit)." This requirement was added without
committee approval. It is also inconsistent with the widespread
convention that badbit should report loss of integrity of the stream
proper (not some other stream). A null sb should set failbit.

27.6.1.2.2:
basic_istream::operator>>(basic_streambuf<charT,traits>* sb), last
line of Effects paragraph 4 can't happen. Previous sentence says,
``If the function inserts no characters, it calls setstate(failbit),
which may throw ios_base failure. Then the last sentence says,
``If failure was due to catching an exception thrown while extracting
characters from sb and failbit is on in exceptions(), then the caught
exception is rethrown." But in this case, setstate has already thrown

ios_base::failure. Besides, I can find no committee resolution
that calls for exceptions() to be queried in this event.
In fact, the approved behavior was simply to terminate the copy
operation if an extractor throws an exception, just as for
get(basic_streambuf&) in 27.6.1.3. Last sentence should be struck.

27.6.1.3:
basic_istream::get(basic_streambuf& sb) Effects says it inserts
characters ``in the output sequence controlled by rdbuf()."
Should be the sequence controlled by sb.

27.6.1.3:
basic_istream::readsome refers several times to in_avail(), which
is not defined in the class. All references should be to
rdbuf()->in_avail(). And the description should specify what
happens when rdbuf() is a null pointer. (Presumably sets badbit.)

27.6.1.3:
basic_istream::readsome is now defined for rdbuf()->in_avail() < 0.
The original proposal defined only the special value -1. Otherwise,
it requires that rdbuf()->in_avail >= 0. Should be restored.

27.6.1.3:
basic_istream::readsome cannot return read, as stated. That
function has the wrong return type. Should return gcount().

27.6.1.3:
basic_istream::putback does *not* call ``rdbuf->sputbackc(c)".
It calls ``rdbuf()->sputbackc(c)" and then only if rdbuf()
is not null.

27.6.1.3:
basic_istream::unget does *not* call ``rdbuf->sungetc(c)".
It calls ``rdbuf()->sungetc(c)" and then only if rdbuf()
is not null.

27.6.1.3:
basic_istream::sync describes what happens when rdbuf()->pubsync()
returns traits::eof(), but that can't happen in general because
pubsync returns an int, not an int_type. This is an unauthorized,
and ill-advised, change from the original EOF. Return value should
also be EOF.

27.6.1.3:
basic_istream::sync Notes says the function uses traits::eof().
Obviously it doesn't, as described above. Clause should be struck.

27.6.2.1:
basic_ostream::seekp(pos_type&) and basic_ostream::seekp(off_type&,
ios_base::seekdir) should both have const first parameters.

27.6.2.1:
basic_ostream definition, last line of paragraph 2 can't happen.
It says, ``If the called
function throws an exception, the output function calls
setstate(badbit), which may throw ios_base::failure, and if badbit

is on in exceptions() rethrows the exception." But in this case, setstate has already thrown ios_base::failure. Besides, I can find no committee resolution that calls for exceptions() to be queried in this event. Last sentence should end with, ``and if setstate returns, the function rethrows the exception."

27.6.1.2.1:
Use of num_put facets to insert numeric output leaves very unclear how output failure is reported. Only the ostreambuf_iterator knows when such a failure occurs. If it throws an exception, the calling code in basic_ostreambuf is obliged to call setstate(badbit) and rethrow the exception, which is *not* the approved behavior for failure of a streambuf primitive.
Possible fixes are: have ostreambuf_iterator report a specific type of exception, have ostreambuf_iterator remember a failure for later testing, or give up on restoring past behavior. Something *must* be done in this area, however.

e7.6.2.4.1:
Tabie 76 is mistitled. It is not just about floating-point conversions.

27.6.2.4.1:
Table 77 has an unauthorized change of rules for determining fill padding. It gives the three defined states of flags() & adjustfield as left, internal, and otherwise. It should be right, internal, and otherwise. Needs to be restored to the earlier (approved) logic.

27.6.2.4.2:
basic_ostream<<operator<<(bool) should use ostreambuf_iterator, not istreambuf_iterator. The first argument is also wrong in the call to num_put::put.

27.6.2.4.2:
basic_ostream::operator<<(basic_streambuf *sb) says nothing about sb being null, unlike the corresponding extractor (27.6.1.2.2). Should either leave both undefined or say both set failbit.

27.6.2.4:
basic_ostream::operator<<(streambuf *) says nothing about the failure indication when ``inserting in the output sequence fails". Should say the function sets badbit.

27.6.2.4.2:
basic_ostream::operator<<(basic_streambuf<charT,traits>* sb), last line of Effects paragraph 2 can't happen. Previous sentence says that if ``an exception was thrown while extracting a character, it calls setstate(failbit) (which may throw ios_base::failure)." Then the last sentence says, ``If an exception was thrown while extracting a character and failbit is on in exceptions() the caught exception is rethrown." But in this case, setstate has already thrown ios_base::failure. Besides, I can find no committee resolution that calls for exceptions() to be queried in this event. And an earlier sentence says unconditionally that the exception is rethrown. Last sentence should be struck.

27.6.2.5:

basic_ostream::flush can't test for a return of traits::eof() from basic_streambuf::pubsync. It tests for EOF.

27.6.3:
``headir" should be ``header."

27.6.3:
<iomanip> does *not* define a single type smanip. Rather, it defines at least two different types which depend on the type of the function argument. Should probably say that each function returns some unspecified type suitable for inserting into an arbitrary basic_ostream object or extracting from a basic_istream object.

27.7:
<sstream> synopsis refers to the nonsense class int_charT_traits. It should be ios_traits.

27.7:
Table 77 (<cstdlib> synopsis) is out of place in the middle of the presentation of <sstream>.

27.7.1:
basic_stringbuf::basic_stringbuf(basic_string, openmode) Effects repeats the phrase ``initializing the base class with basic_streambuf()." Strike the repetition.

27.7.1:
basic_stringbuf::basic_stringbuf(basic_string, openmode) Postconditions requires that str() == str. This is true only if which has in set. Condition should be restated.

27.7.1:
Table 78 describes calls to setg and setp with string arguments, for which no signature exists. Needs to be recast.

27.7.1:
basic_stringbuf::str(basic_string s) Postconditions requires that str() == s. This is true only if which had in set at construction time. Condition should be restated.

27.7.1.2:
Table 80 describes calls to setg and setp with string arguments, for which no signature exists. Needs to be recast.

27.7.1.3:
basic_stringbuf::underflow Returns should return int_type(*gptr()), not char_type(*gptr()).

27.7.1.3:
basic_stringbuf::pbackfail returns c (which is int_type) in first case, char_type(c) in second case. Both cases should be c.

27.7.1.3:
basic_stringbuf::pbackfail supposedly returns c when c == eof(). Should return traits::not_eof(c).

27.7.1.3:
basic_stringbuf::seekpos paragraph 4 has ``positionedif" run together.

27.8.1.1:
basic_filebuf paragraph 3 talks about a file being ``open for reading or for update," and later ``open for writing or for update." But ``open for update" is not a defined term. Should be struck in both cases.

27.8.1.3:
basic_filebuf::is_open allegedly tests whether ``the associated file is available and open." No definition exists for ``available." The term should be struck.

27.8.1.3:
basic_filebuf::open Effects says the function fails if is_open() is initially false. Should be if initially true.

27.8.1.3:
basic_filebuf::open Effects says the function calls the default constructor for basic_streambuf. This is nonsense. Should say, at most, that it initializes the basic_filebuf as needed, and then only after it succeeds in opening the file.

27.8.1.3:
Table 83 has a duplicate entry for file open mode ``in | out".

27.8.1.4:
filebuf::showmanyc (and several overriden virtual functions that follow) have a Requires clause that says ``is_open == true." The behavior of all these functions should be well defined in that event, however. Typically, the functions all fail. The Requires clause should be struck in all cases.

27.8.1.4:
filebuf::showmanyc Effects says the function ``behaves the same as basic_streambuf::showmanyc." The description adds nothing and should be struck.

27.8.1.4:
basic_filebuf::underflow effects shows arguments to convert as ``st,from_buf,from_buf+FSIZE,from_end,to_buf, to_buf+to_size, to_end". st should be declared as an object of type state_type, and n should be defined as the number of characters read into from_buf. Then the arguments should be ``st, from_buf, from_buf + n, from_end, to_buf, to_buf + TSIZE, to_end". Also, template parameter should be ``traits::state_type," not ``ios_traits::state_type."

27.8.1.4:
basic_filebuf::underflow is defined unequivocally as the function that calls codecvt, but there are performance advantages to having this conversion actually performed in uflow. If the specification cannot be broadened sufficiently to allow either function to do the translation, then uflow loses its last rationale for being added in the first place. Either the extra latitude should be

granted implementors or uflow should be removed from basic_streambuf and all its derivatives.

27.8.1.4:
basic_filebuf::pbackfail(traits::eof()) used to return a value other than eof() if the function succeeded in backing up the input. Now the relevant Returns clause says the function returns the metacharacter c, which is indistinguishable from a failure return. This is an unapproved change. Should probably say the function returns traits::not_eof(c).

27.8.1.4:
basic_filebuf::pbackfail Notes now says ``if is_open() is false, the function always fails." This is an unapproved change. The older wording should be restored.

27.8.1.4:
basic_filebuf::pbackfail Notes now says ``the function does not put back a character directly to the input sequence." This is an unapproved change and not in keeping with widespread practice. The older wording should be restored.

27.8.1.4:
basic_filebuf::pbackfail has a Default behavior clause. Should be struck.

27.8.1.4:
basic_filebuf::overflow effects shows arguments to convert as ``st,b(),p(),end,buf,buf+BSIZE,ebuf". st should be declared as an object of type state_type. Then the arguments should be ``st, b, p, end, buf, buf + BSIZE, ebuf". Also, template parameter should be ``traits::state_type," not ``ios_traits::state_type."

27.8.1.4:
basic_filebuf::overflow doesn't say what it returns on success. Should say it returns c.

27.8.1.4:
basic_filebuf::setbuf has no semantics. Needs to be supplied.

27.8.1.4:
basic_filebuf::seekoff Effects is an interesting exercise in creative writing. It should simply state that if the stream is opened as a text file or has state-dependent conversions, the only permissible seeks are with zero offset relative to the beginning or current position of the file. (How to determine that predicate is another matter -- should state for codecvt that even a request to convert zero characters will return noconv.) Otherwise, behavior is largely the same as for basic_stringstream, from whence the words should be cribbed. The problem of saving the stream state in a traits::pos_type object remains unsolved. The primitives described for ios_traits are inadequate.

27.8.1.4:
basic_filebuf::seekpos has no semantics. Needs to be supplied.

27.8.1.4:
basic_filebuf::sync has no semantics. Needs to be supplied.

27.8.1.4:
basic_filebuf::imbue has silly semantics. Whether or not sync() succeeds has little bearing on whether you can safely change the working codecvt facet. The most sensible thing is to establish this facet at construction. (Then pubimbue and imbue can be scrubbed completely.) Next best is while is_open() is false. (Then imbue can be scrubbed, since it has nothing to do.) Next best is to permit any imbue that doesn't change the facet or is at beginning of file. Next best is to permit change of facet any time provided either the current or new facet does not mandate state-dependent conversions. (See comments under seekoff.)

27.8.1.7:
basic_filebuf::rdbuf should not have explicit qualifier.

27.8.1.9:
basic_ofstream::basic_ofstream(const char *s, openmode mode = out) has wrong default second argument. It should be `out | trunc', the same as for basic_ofstream::open (in the definition at least).

27.8.1.10:
basic_ofstream::open(const char *s, openmode mode = out) has wrong default second argument. It should be `out | trunc', the same as for basic_ofstream::open in the definition.

27.8.2:
<cstdio> synopsis has two copies of tmpfile and vprintf, no vfprintf or putchar.

27.8.2:
<cwchar> summary should also list the type wchar_t. Aside from the addition of the (incomplete) type struct tm, this table 84 is identical to table 44 in 21.2. It is not clear what purpose either table serves; it is less clear what purpose is served by repeating the table.

27.8.2:
See Also reference for <wchar> should be 7.13.2, not 4.6.2.

D.2:
Functions overloaded on io_state, open_mode, and seek_dir ``call the corresponding member function." But no hint is given as to what constitutes ``correspondence."

D.3.1.3:
strstreambuf::overflow has numerous references to ``eof()", which no longer exists. All should be changed to EOF.

D.3.1.3:
strstreambuf::overflow says it returns ``(char)c" sometimes, but this can pun with EOF if char has a signed representation. More accurate to say it returns (unsigned char)c.

D.3.1.3:
strstreambuf::pbackfail says it returns ``(char)c" sometimes, but this can pun with EOF if char has a signed representation. More accurate to say it returns (unsigned char)c.

D.3.1.3:
strstreambuf::pbackfail says it returns ``(char)c" when c == EOF, but this can pun with EOF if char has a signed representation. More accurate to say it returns something other than EOF.

D.3.1.3:
strstreambuf::pbackfail twice says it returns EOF to indicate failure. Once is enough.

D.3.1.3:
strstreambuf::setbuf has a Default behavior clause, which is not appropriate for a derived stream buffer. It also adds nothing to the definition in the base class. The entire description should be struck.

Comments on C++ draft, dated 1995-04-28.

Terminology: A.B-C means subclause A.B, paragraph C.

3.2-8

The acronym ``ODR" has not been defined. Also, it doesn't make sense when expanded: ``one definition rule rule".

3.7.3.2-5

Footnote #20 refers to ``architectures". Other places refer to ``machines". They should all refer to ``implementations".

3.8

It is not clear what object ``use" or ``reuse" is.

3.8-2

The acronym ``POD" has not been defined. In general, each section should have ``forward references", like the C Standard.

3.8-3

Awkward wording: ``In particular, except as noted".

3.9-2

How can I tell that the ``copy operation is well-defined"? It is not clear what ``well-defined" means here or if I can test for it.

3.9-4 The ``value" of an object of type T is not necessarily based upon its bit represention, especially when the class is a handle to other data. The ``value" in this case would depend upon how the "==" operator is overloaded. Even if its ``representation value" is somehow defined, what purpose does it serve? Where else is this used in the draft?

3.9.1-1

Remove ``there are several fundamental types".

3.9.1-2

Use different wording than ``take on".

3.9.1-4

Don't refer to ``machine architecture". See C Standard wording.

3.9.1-6

Change ``laws of arithmetic modulo 2^N" to C Standard wording.

3.9.1-8

Reword ``although values of type bool generally behave ...".

3.9.1-8

Reword ``successfully be stored".

3.9.2-1

Reword ``There is a conceptually infinite ...". Remove the words ``conceptually" and ``infinite".

3.9.3-1

The definition of "volatile" is missing. It isn't in subclause 1.8 or 7.1.5.1. See the C definition: ``An object that has volatile-qualified type may be modified in ways unknown to the implementation ...".

3.9.3-5

Change ``In this document" to ``In this International Standard".

3.10-2

Footnote #30: Clarify ``... in some sense refer to an

object".

4.1-1

Reword ``necessitates ... is ill-formed" to use ``shall" or ``shall not".

4.1-1

Footnote #31. Need proper reference to Standard C.

4.3-1

Footnote #32. Reword ``there is no way ...".

4.5-1

Reword ``can" with ``shall'.

4.4-4

The sentence ``That is, the member aspect ..." should be a footnote.

4.5-2

Reword ``can" with ``shall'.

4.5-3

Reword ``can" with ``shall'.

4.5-3

Footnote #34: Reword ``If the bit-field is larger yet, ..." using ``shall" and ``shall not". If this is a constraint, it shouldn't be a footnote.

4.5-4

Reword ``can" with ``shall'.

4.7-2

What is the difference here between a note and a footnote? This should be a footnote.

5.2.2-7

A bit-field is not a type.

5.2.2-7

Change ``unsigned" to ``unsigned int" ``int, unsigned int, ...".

5.2.6-1

Reword ``... shall not cast away constness" in more precise terms. See 5.2.9-2's reference to 5.2.10.

5.2.7-1

Footnote #43: Does "*(p)" meet this requirement?

5.2.7-1

Shouldn't ``then the pointer shall either be zero" be ``then the pointer shall either be the null pointer value"?

5.2.8-1

Reword ``... shall not cast away constness" in more precise terms. See 5.2.9-2's reference to 5.2.10.

5.2.10

This section is hard to understand, especially the rules defining casting away constness.

5.2.10-4

Does ``implicit conversion" here refer to subclause 4.10, pointer conversion?

5.2.10-7

The ``[Note:" doesn't have a closing ``]". This appears to be a formatting issue throughout the document.

5.2.10-7

Where are ``multi-level' and ``mixed object" defined?

5.3.1-2

How do the ``implicit conversions" here relate to the ``implicit conversions" of 4.10 or 5.2.10? The term ``implicit conversion" should be defined explicitly.

5.3.5-2

What is ``(_class.conv,fct_)"?

5.7-6

How is C compatibility maintained if a different header is required for C++ for "ptrdiff_t"?

5.8-1

Why isn't the C wording used here, especially the semantics

for unsigned integers?

5.9-2

Change ``The usual arithmetic conversions" to ``The standard integral promotions (4.5)".

5.10-1

It is not clear ``.... have the same semantic restrictions, conversions" what this points to. The wording should be repeated or the reference to the associated text should be clearer.

5.11-1

See 5.9-2 above on ``usual arithmetic conversions".

5.12-1

See 5.9-2 above on ``usual arithmetic conversions".

5.13-1

See 5.9-2 above on ``usual arithmetic conversions".

5.16-1

What was the grammar changed from C? The expression after the colon should be ``conditional-expression".

5.16-2

If both the second and third expression are throw-expressions, then what it the type of the result? According to 15-1, the resultant type of the throw expression is "void". Thus, the resultant type of "?:" is "void". This should be made clear here.

5.17-4

Change ``the user" to ``the program". Change all other uses of ``the user" to something else in the rest of the draft.

7.1.2-2

Change ``hint" wording to use C wording similar to "register" keyword.

7.1.3-5

Reword ``... The typedef-name is still only a synonym for the dummy name and shall not be used where a true class name is required". Either ``dummy name" should be defined or removed in this paragraph (used several times). What is a

``true class name"? If the dummy name is not specified, why do I care about it for ``linkage purposes"?

7.1.5.1-3

The draft says ``CV-qualifiers are supported by the type system so that they cannot be subverted without casting", but it doesn't specify that the behavior is undefined (C says it's undefined).

7.1.5.1-7

This should not be a note, but part of the standard. The same wording should be extracted from the C Standard.

7.2-1

Reword or remove ``... not gratuitously larger than int". If it's implementation-defined, then say so.

7.2-6

The possibility that the compiler generates bit fields for enumerators means that it would not be object, i.e., not addressible. Since it is impossible to determine whether or not the address is taken (the "enum" might have its address taken in some other translation unit), having the compiler decide bit-field or not won't work. If "enum" bit fields are to be supported, they should use some *obvious* syntax. Also, implicit bit fields would be incompatible with C programs.

7.3.1.2-1

If an unnamed namespace has a unique identifier that cannot be determined and cannot be linked to (even if there is external linkage -- see footnote 54), then an unnamed namespace is equivalent to "static" at file scope. The draft should change the wording to be the equivalent of "static" at file scope (a feature all linkers can provide) rather than the requirement that a unique name be created (difficult for linkers and *very* difficult for externally developed libraries). If an implementation creates something that I cannot detect then it doesn't exist.

7.3.3-6

Remove ``... (as ever)".

7.3.4-4

What is a ``using-directive lattice"? Where is it defined?

7.5-3

If a function has more than one linkage specification (say

in different translation units) a diagnostic is required. However, the compiler and/or linker may not be able to detect this even with type-safe linking (type-safe linking doesn't imply that the function call mechanisms are the same).

7.5-6

Reword ``There is no way ...".

7.5-8

Change: ``FORTRAN" is now properly spelled ``Fortran" according to the Fortran Standard. It would be better if C++ specified that the linkage string is case insensitive and is in the ISO 646 subset. Since the linkage is all implementation-defined anyway, the linker (and the compiler) will know the true way (possibly, case-sensitive) of spelling the linkage name.

8-3

Footnote 55: Reword ``A declaration with several declarations is usually equivalent" to remove the word ``usually".

8.2-1

Reword ``In that context, it surfaces ..." to remove ``surfaces".

8.2-1

Remove ``Just as for statements". The reference to which section is unclear. The level of semantics to drag in are not specified.

8.2-2

Reword ``... can occur in many different contexts ..." to remove the word ``many".

8.2-3

Number the 4 examples.

8.3-2

Change ``inductive" to ``recursive".

8.3.1-3

Reword ``volatile specifiers are handled similarly.". Similar to what?

8.3.2-4

Reword ``In particular, null references are prohibited; no diagnostic is required.''. What does ``prohibited'' mean? Do you mean ``undefined'' here?

8.3.4-1

Typo: ``,T'' ==> ``T,'', ``.T'' ==> ``T.''

8.3.4-2

Replace with C Standard wording. The C wording is clearer and shorter: ``An array type describes a contiguously allocated nonempty set of objects with a particular member object type, called the element type.'' (there is a footnote attached that explains imcomplete types are disallowed).

8.3.4-3

Reword ``When several array of specifications are adjacent'' to remove or define the word ``adjacent''.

8.3.4-4

Reword ``... (say, N) ...'' to remove the ``say, N''. Possibly, start the sentence ``If N is the number of initial elements, ...''.

8.3.5-2

Is using the C "<cstdarg>" the same as "<stdarg.h>"? If not, then the code will be incompatible.

8.3.5-4

Remove ``Functions shall not return arrays ... [to the end of the paragraph]''. This restriction has been stated elsewhere.

8.3.5-5

Remove this paragraph. It has been stated elsewhere.

8.3.6-6

Change ``out-of-line function'' to ``non-inline function''.

8.3.6-9

Previously, the order of evaluation of function arguments was ``unspecified''. Here it's ``implementation-defined''. Which is it?

8.5-6

Need forward reference to ``POD''. It has not yet been

defined. The restriction on arrays has been stated elsewhere.

18.1-3

If the C standard header "<stddef.h>" is used, do I get the same result as including "<cstddef>"? Subclause D.1 refers to compatibility, but this isn't clear. Also, this paragraph should refer to D.1.

D.1-1

The names should be the same for C headers in C++. There should be no renaming. This breaks C code to rename them, especially when both should behave the same. Rather than the name "cstdlib", is should be "stdlib.h". Since every C compiler already supports this, C++ can't claim defective linkers, filesystems, and so on. This is a gratuitous difference that just breaks working code.

P. J. PLAUGER

WG14 CONVENOR