Doc. no. J16/01-0032 = WG21 N1318

Date: 11 Sep 2001

Project: Programming Language C++

Reply to: Matt Austern <austern@research.att.com>

C++ Standard Library Defect Report List (Revision 19)

Reference ISO/IEC IS 14882:1998(E)

Also see:

- Table of Contents for all library issues.
- Index by Section for all library issues.
- Index by Status for all library issues.
- Library Active Issues List
- Library Closed Issues List

This document contains only library issues which have been closed by the Library Working Group (LWG) after being found to be defects in the standard. That is, issues which have a status of DR, TC, or RR. See the Library Closed Issues List for issues closed as non-defects. See the Library Active Issues List for active issues and more information. The introductory material in that document also applies to this document.

Revision History

- R19: Pre-Redmond mailing. Added new issues 323-335.
- R18: Post-Copenhagen mailing; reflects actions taken in Copenhagen. Added new issues 312-317, and discussed new issues 271-314. Changed status of issues 103 118 136 153 165 171 183 184 185 186 214 221 234 237 243 248 251 252 256 260 261 262 263 265 268 to DR. Changed status of issues 49 109 117 182 228 230 232 235 238 241 242 250 259 264 266 267 271 272 273 275 281 284 285 286 288 292 295 297 298 301 303 306 307 308 312 to Ready. Closed issues 111 277 279 287 289 293 302 313 314 as NAD.
- R17: Pre-Copenhagen mailing. Converted issues list to XML. Added proposed resolutions for issues 49, 76, 91, 235, 250, 267. Added new issues 278-311.
- R16: post-Toronto mailing; reflects actions taken in Toronto. Added new issues 265-277. Changed status of issues 3, 8, 9, 19, 26, 31, 61, 63, 86, 108, 112, 114, 115, 122, 127, 129, 134, 137, 142, 144, 146, 147, 159, 164, 170, 181, 199, 208, 209, 210, 211, 212, 217, 220, 222, 223, 224, 227 to "DR". Reopened issue 23. Reopened issue 187. Changed issues 2 and 4 to NAD. Fixed a typo in issue 17. Fixed issue 70: signature should be changed both places it appears. Fixed issue 160: previous version didn’t fix the bug in enough places.
- R15: pre-Toronto mailing. Added issues 233-264. Some small HTML formatting changes so that we pass Weblint tests.
- R14: post-Tokyo II mailing; reflects committee actions taken in Tokyo. Added issues 228 to 232. (00-0019R1/N1242)
- R13: pre-Tokyo II updated: Added issues 212 to 227.
- R12: pre-Tokyo II mailing: Added issues 199 to 211. Added "and paragraph 5" to the proposed resolution of issue 29. Add further rationale to issue 178.
- R11: post-Kona mailing: Updated to reflect LWG and full committee actions in Kona (99-0048/N1224). Note changed resolution of issues 4 and 38. Added issues 196 to 198. Closed issues list split into "defects" and "closed" documents. Changed the proposed resolution of issue 4 to NAD, and changed the wording of proposed resolution of issue 38.
- R10: pre-Kona updated. Added proposed resolutions 83, 86, 91, 92, 109. Added issues 190 to 195. (99-0033/D1209, 14 Oct 99)

- R9: pre-Kona mailing. Added issues 140 to 189. Issues list split into separate "active" and "closed" documents. (99-0030/N1206, 25 Aug 99)
- R8: post-Dublin mailing. Updated to reflect LWG and full committee actions in Dublin. (99-0016/N1193, 21 Apr 99)
- R7: pre-Dublin updated: Added issues 130, 131, 132, 133, 134, 135, 136, 137, 138, 139 (31 Mar 99)
- R6: pre-Dublin mailing. Added issues 127, 128, and 129. (99-0007/N1194, 22 Feb 99)
- R5: update issues 103, 112; added issues 114 to 126. Format revisions to prepare for making list public. (30 Dec 98)
- R4: post-Santa Cruz II updated: Issues 110, 111, 112, 113 added, several issues corrected. (22 Oct 98)
- R3: post-Santa Cruz II: Issues 94 to 109 added, many issues updated to reflect LWG consensus (12 Oct 98)
- R2: pre-Santa Cruz II: Issues 73 to 93 added, issue 17 updated. (29 Sep 98)
- R1: Correction to issue 55 resolution, 60 code format, 64 title. (17 Sep 98)

Defect Reports

1. C library linkage editing oversight

Section: 17.4.2.2 [lib.using.linkage] **Status:** DR **Submitter:** Beman Dawes **Date:** 16 Nov 1997

The change specified in the proposed resolution below did not make it into the Standard. This change was accepted in principle at the London meeting, and the exact wording below was accepted at the Morristown meeting.

Proposed resolution:

Change 17.4.2.2 paragraph 2 from:

> It is unspecified whether a name from the Standard C library declared with external linkage has either extern "C" or extern "C++" linkage.

to:

> Whether a name from the Standard C library declared with external linkage has extern "C" or extern "C++" linkage is implementation defined. It is recommended that an implementation use extern "C++" linkage for this purpose.

---

3. Atexit registration during atexit() call is not described

Section: 18.3 [lib.support.start.term] **Status:** DR **Submitter:** Steve Clamage **Date:** 12 Dec 1997

We appear not to have covered all the possibilities of exit processing with respect to atexit registration.
Example 1: (C and C++)

```
#include <stdlib.h>
void f1() { }
void f2() { atexit(f1); }

int main()
{
    atexit(f2); // the only use of f2
    return 0; // for C compatibility
}
```

At program exit, f2 gets called due to its registration in main. Running f2 causes f1 to be newly registered during the exit processing. Is this a valid program? If so, what are its semantics?

Interestingly, neither the C standard, nor the C++ draft standard nor the forthcoming C9X Committee Draft says directly whether you can register a function with atexit during exit processing.

All 3 standards say that functions are run in reverse order of their registration. Since f1 is registered last, it ought to be run first, but by the time it is registered, it is too late to be first.

If the program is valid, the standards are self-contradictory about its semantics.

Example 2: (C++ only)

```
void F() { static T t; } // type T has a destructor

int main()
{
    atexit(F); // the only use of F
}
```

Function F registered with atexit has a local static variable t, and F is called for the first time during exit processing. A local static object is initialized the first time control flow passes through its definition, and all static objects are destroyed during exit processing. Is the code valid? If so, what are its semantics?

Section 18.3 "Start and termination" says that if a function F is registered with atexit before a static object t is initialized, F will not be called until after t's destructor completes.

In example 2, function F is registered with atexit before its local static object O could possibly be initialized. On that basis, it must not be called by exit processing until after O's destructor completes. But the destructor cannot be run until after F is called, since otherwise the object could not be constructed in the first place.

If the program is valid, the standard is self-contradictory about its semantics.

I plan to submit Example 1 as a public comment on the C9X CD, with a recommendation that the results be undefined. (Alternative: make it unspecified. I don't think it is worthwhile to specify the case where f1 itself registers additional functions, each of which registers still more functions.)

I think we should resolve the situation in the whatever way the C committee decides.

For Example 2, I recommend we declare the results undefined.

[See reflector message lib-6500 for further discussion.]

Proposed resolution:

Change section 18.3/8 from:

> First, objects with static storage duration are destroyed and functions registered by calling atexit are called. Objects with static storage duration are destroyed in the reverse order of the completion of their constructor. (Automatic objects are not destroyed as a result of calling exit().) Functions registered with atexit are called in the reverse order of their registration. A function registered with atexit before an object obj1 of static storage duration is initialized will not be called until obj1's destruction has completed. A function registered with atexit after an object obj2 of static storage duration is initialized will be called before obj2's destruction starts.

to:

First, objects with static storage duration are destroyed and functions registered by calling atexit are called. Non-local objects with static storage duration are destroyed in the reverse order of the completion of their constructor. (Automatic objects are not destroyed as a result of calling exit().) Functions registered with atexit are called in the reverse order of their registration, except that a function is called after any previously registered functions that had already been called at the time it was registered. A function registered with atexit before a non-local object obj1 of static storage duration is initialized will not be called until obj1's destruction has completed. A function registered with atexit after a non-local object obj2 of static storage duration is initialized will be called before obj2's destruction starts. A local static object obj3 is destroyed at the same time it would be if a function calling the obj3 destructor were registered with atexit at the completion of the obj3 constructor.

Rationale:

See 99-0039/N1215, October 22, 1999, by Stephen D. Clamage for the analysis supporting to the proposed resolution.

---

5. String::compare specification questionable

Section: 21.3.6.8 [lib.string::compare] **Status:** DR **Submitter:** Jack Reeves **Date:** 11 Dec 1997

At the very end of the basic_string class definition is the signature: int compare(size_type pos1, size_type n1, const charT* s, size_type n2 = npos) const; In the following text this is defined as: returns basic_string<charT,traits,Allocator>(*this,pos1,n1).compare(basic_string<charT,traits,Allocator>(s,n2);

Since the constructor basic_string(const charT* s, size_type n, const Allocator& a = Allocator()) clearly requires that s != NULL and n < npos and further states that it throws length_error if n == npos, it appears the compare() signature above should always throw length error if invoked like so: str.compare(1, str.size()-1, s); where 's' is some null terminated character array.

This appears to be a typo since the obvious intent is to allow either the call above or something like: str.compare(1, str.size()-1, s, strlen(s)-1);

This would imply that what was really intended was two signatures int compare(size_type pos1, size_type n1, const charT* s) const int compare(size_type pos1, size_type n1, const charT* s, size_type n2) const; each defined in terms of the corresponding constructor.

Proposed resolution:

Replace the compare signature in 21.3 (at the very end of the basic_string synopsis) which reads:

```
int compare(size_type pos1, size_type n1,
            const charT* s, size_type n2 = npos) const;
```

with:

```
int compare(size_type pos1, size_type n1,
            const charT* s) const;
int compare(size_type pos1, size_type n1,
            const charT* s, size_type n2) const;
```

Replace the portion of 21.3.6.8 paragraphs 5 and 6 which read:

```
int compare(size_type pos, size_type n1,
            charT * s, size_type n2 = npos) const;
```

Returns:

```
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s, n2))
```

with:

```
int compare(size_type pos, size_type n1,
            const charT * s) const;
```
Returns:
```
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s ))
int compare(size_type pos, size_type n1,
            const charT * s, size_type n2) const;
```
Returns:
```
basic_string<charT,traits,Allocator>(*this, pos, n1).compare(
             basic_string<charT,traits,Allocator>( s, n2))
```

Editors please note that in addition to splitting the signature, the third argument becomes const, matching the existing synopsis.

Rationale:

While the LWG dislikes adding signatures, this is a clear defect in the Standard which must be fixed. The same problem was also identified in issues 7 (item 5) and 87.

---

7. String clause minor problems

Section: 21 [lib.strings] **Status:** DR **Submitter:** Matt Austern **Date:** 15 Dec 1997

(1) In 21.3.5.4 , the description of template <class InputIterator> insert(iterator, InputIterator, InputIterator) makes no sense. It refers to a member function that doesn't exist. It also talks about the return value of a void function.

(2) Several versions of basic_string::replace don't appear in the class synopsis.

(3) basic_string::push_back appears in the synopsis, but is never described elsewhere. In the synopsis its argument is const charT, which doesn't makes much sense; it should probably be charT, or possible const charT&.

(4) basic_string::pop_back is missing.

(5) int compare(size_type pos, size_type n1, charT* s, size_type n2 = npos) make no sense. First, it's const charT* in the synopsis and charT* in the description. Second, given what it says in RETURNS, leaving out the final argument will always result in an exception getting thrown. This is paragraphs 5 and 6 of 21.3.6.8

(6) In table 37, in section 21.1.1 , there's a note for X::move(s, p, n). It says "Copies correctly even where p is in [s, s+n)". This is correct as far as it goes, but it doesn't go far enough; it should also guarantee that the copy is correct even where s in in [p, p+n). These are two orthogonal guarantees, and neither one follows from the other. Both guarantees are necessary if X::move is supposed to have the same sort of semantics as memmove (which was clearly the intent), and both guarantees are necessary if X::move is actually supposed to be useful.

Proposed resolution:

ITEM 1: In 21.3.5.4 [lib.string::insert], change paragraph 16 to
EFFECTS: Equivalent to insert(p - begin(), basic_string(first, last)).
ITEM 2: Not a defect; the Standard is clear.. There are ten versions of replace() in the synopsis, and ten versions in 21.3.5.6 [lib.string::replace].
ITEM 3: Change the declaration of push_back in the string synopsis (21.3, [lib.basic.string]) from:

```
    void push_back(const charT)
```
to
```
    void push_back(charT)
```
Add the following text immediately after 21.3.5.2 [lib.string::append], paragraph 10.

void basic_string::push_back(charT c);
EFFECTS: Equivalent to append(static_cast<size_type>(1), c);
ITEM 4: Not a defect. The omission appears to have been deliberate.
ITEM 5: Duplicate; see issue 5 (and 87).
ITEM 6: In table 37, Replace:
"Copies correctly even where p is in [s, s+n)."
with:
"Copies correctly even where the ranges [p, p+n) and [s, s+n) overlap."

---

8. Locale::global lacks guarantee

Section: 22.1.1.5 [lib.locale.statics] **Status:** DR **Submitter:** Matt Austern **Date:** 24 Dec 1997

It appears there's an important guarantee missing from clause 22. We're told that invoking locale::global(L) sets the C locale if L has a name. However, we're not told whether or not invoking setlocale(s) sets the global C++ locale.

The intent, I think, is that it should not, but I can't find any such words anywhere.

Proposed resolution:

Add a sentence at the end of 22.1.1.5 , paragraph 2:

> No library function other than `locale::global()` shall affect the value returned by `locale()`.

---

9. Operator new(0) calls should not yield the same pointer

Section: 18.4.1 [lib.new.delete] **Status:** DR **Submitter:** Steve Clamage **Date:** 4 Jan 1998

Scott Meyers, in a comp.std.c++ posting: I just noticed that section 3.7.3.1 of CD2 seems to allow for the possibility that all calls to operator new(0) yield the same pointer, an implementation technique specifically prohibited by ARM 5.3.3.Was this prohibition really lifted? Does the FDIS agree with CD2 in the regard? [Issues list maintainer's note: the IS is the same.]

Proposed resolution:

Change the last paragraph of 3.7.3 from:

> Any allocation and/or deallocation functions defined in a C++ program shall conform to the semantics specified in 3.7.3.1 and 3.7.3.2.

to:

> Any allocation and/or deallocation functions defined in a C++ program, including the default versions in the library, shall conform to the semantics specified in 3.7.3.1 and 3.7.3.2.

Change 3.7.3.1/2, next-to-last sentence, from :

> If the size of the space requested is zero, the value returned shall not be a null pointer value (4.10).

to:

> Even if the size of the space requested is zero, the request can fail. If the request succeeds, the value returned shall be a non-null pointer value (4.10) p0 different from any previously returned value p1, unless that value p1 was since passed to an operator delete.

5.3.4/7 currently reads:

> When the value of the expression in a direct-new-declarator is zero, the allocation function is called to allocate an array with no elements. The pointer returned by the new-expression is non-null. [Note: If the library allocation function is called, the pointer returned is distinct from the pointer to any other object.]

Retain the first sentence, and delete the remainder.

18.4.1 currently has no text. Add the following:

> Except where otherwise specified, the provisions of 3.7.3 apply to the library versions of operator new and operator delete.

To 18.4.1.3, add the following text:

> The provisions of 3.7.3 do not apply to these reserved placement forms of operator new and operator delete.

Rationale:

See 99-0040/N1216, October 22, 1999, by Stephen D. Clamage for the analysis supporting to the proposed resolution.

---

11. Bitset minor problems

Section: 23.3.5 [lib.template.bitset] **Status:** DR **Submitter:** Matt Austern **Date:** 22 Jan 1998

(1) bitset<>::operator[] is mentioned in the class synopsis (23.3.5), but it is not documented in 23.3.5.2.

(2) The class synopsis only gives a single signature for bitset<>::operator[], reference operator[](size_t pos). This doesn't make much sense. It ought to be overloaded on const. reference operator[](size_t pos); bool operator[](size_t pos) const.

(3) Bitset's stream input function (23.3.5.3) ought to skip all whitespace before trying to extract 0s and 1s. The standard doesn't explicitly say that, though. This should go in the Effects clause.

Proposed resolution:

ITEMS 1 AND 2:
In the bitset synopsis (23.3.5), replace the member function

```
reference operator[](size_t pos);
```

with the two member functions

```
bool operator[](size_t pos) const;
reference operator[](size_t pos);
```

Add the following text at the end of 23.3.5.2 , immediately after paragraph 45:

```
bool operator[](size_t pos) const;
```

Requires: pos is valid
Throws: nothing
Returns: `test(pos)`

```
bitset<N>::reference operator[](size_t pos);
```

Requires: pos is valid
Throws: nothing
Returns: An object of type `bitset<N>::reference` such that `(*this)[pos] == this->test(pos)`, and such that `(*this)[pos] = val` is equivalent to `this->set(pos, val);`

Rationale:

The LWG believes Item 3 is not a defect. "Formatted input" implies the desired semantics. See 27.6.1.2 .

---

13. Eos refuses to die

Section: 27.6.1.2.3 [lib.istream::extractors] **Status:** DR **Submitter:** William M. Miller **Date:** 3 Mar 1998

In 27.6.1.2.3, there is a reference to "eos", which is the only one in the whole draft (at least using Acrobat search), so it’s undefined.

Proposed resolution:

In 27.6.1.2.3 , replace "eos" with "charT()"

---

14. Locale::combine should be const

Section: 22.1.1.3 [lib.locale.members] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

locale::combine is the only member function of locale (other than constructors and destructor) that is not const. There is no reason for it not to be const, and good reasons why it should have been const. Furthermore, leaving it non-const conflicts with 22.1.1 paragraph 6: "An instance of a locale is immutable."

History: this member function originally was a constructor. it happened that the interface it specified had no corresponding language syntax, so it was changed to a member function. As constructors are never const, there was no "const" in the interface which was transformed into member "combine". It should have been added at that time, but the omission was not noticed.

Proposed resolution:

In 22.1.1 and also in 22.1.1.3 , add "const" to the declaration of member combine:

```
template <class Facet> locale combine(const locale& other) const;
```

---

15. Locale::name requirement inconsistent

Section: 22.1.1.3 [lib.locale.members] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

locale::name() is described as returning a string that can be passed to a locale constructor, but there is no matching constructor.

Proposed resolution:

In 22.1.1.3 , paragraph 5, replace "`locale(name())`" with "`locale(name().c_str())`".

---

16. Bad ctype_byname<char> decl

Section: 22.2.1.4 [lib.locale.ctype.byname.special] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The new virtual members ctype_byname<char>::do_widen and do_narrow did not get edited in properly. Instead, the member do_widen appears four times, with wrong argument lists.

Proposed resolution:

The correct declarations for the overloaded members `do_narrow` and `do_widen` should be copied from 22.2.1.3 .

---

17. Bad bool parsing

Section: 22.2.2.1.2 [lib.facet.num.get.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

This section describes the process of parsing a text boolean value from the input stream. It does not say it recognizes either of the sequences "true" or "false" and returns the corresponding bool value; instead, it says it recognizes only one of those sequences, and chooses which according to the received value of a reference argument intended for returning the result, and reports an error if the other sequence is found. (!) Furthermore, it claims to get the names from the ctype<> facet rather than the numpunct<> facet, and it examines the "boolalpha" flag wrongly; it doesn't define the value "loc"; and finally, it computes wrongly whether to use numeric or "alpha" parsing.
I believe the correct algorithm is "as if":

```
// in, err, val, and str are arguments.
err = 0;
const numpunct<charT>& np = use_facet<numpunct<charT> >(str.getloc());
const string_type t = np.truename(), f = np.falsename();
bool tm = true, fm = true;
size_t pos = 0;
while (tm && pos < t.size() || fm && pos < f.size()) {
  if (in == end) { err = str.eofbit; }
  bool matched = false;
  if (tm && pos < t.size()) {
    if (!err && t[pos] == *in) matched = true;
    else tm = false;
  }
  if (fm && pos < f.size()) {
    if (!err && f[pos] == *in) matched = true;
    else fm = false;
  }
  if (matched) { ++in; ++pos; }
  if (pos > t.size()) tm = false;
  if (pos > f.size()) fm = false;
}
if (tm == fm || pos == 0) { err |= str.failbit; }
else                      { val = tm; }
return in;
```

Notice this works reasonably when the candidate strings are both empty, or equal, or when one is a substring of the other. The proposed text below captures the logic of the code above.

Proposed resolution:

In 22.2.2.1.2 , in the first line of paragraph 14, change "&&" to "&".

Then, replace paragraphs 15 and 16 as follows:

> Otherwise target sequences are determined "as if" by calling the members `falsename()` and `truename()` of the facet obtained by `use_facet<numpunct<charT> >(str.getloc())`. Successive characters in the range `[in,end)` (see [lib.sequence.reqmts]) are obtained and matched against corresponding positions in the target sequences only as necessary to identify a unique match. The input iterator `in` is compared to `end` only when necessary to obtain a character. If and only if a target sequence is uniquely matched, `val` is set to the corresponding value.

The `in` iterator is always left pointing one position beyond the last character successfully matched. If `val` is set, then err is set to `str.goodbit`; or to `str.eofbit` if, when seeking another character to match, it is found that `(in==end)`. If `val` is not set, then *err* is set to `str.failbit`; or to `(str.failbit|str.eofbit)`if the reason for the failure was that `(in==end)`. [Example: for targets `true`:"a" and `false`:"abb", the input sequence "a" yields `val==true` and `err==str.eofbit`; the input sequence "abc" yields `err=str.failbit`, with `in` ending at the 'c' element. For targets `true`:"1" and `false`:"0", the input sequence "1" yields `val==true` and `err=str.goodbit`. For empty targets (""), any input sequence yields `err==str.failbit`. --end example]

---

18. Get(...bool&) omitted

Section: 22.2.2.1.1 [lib.facet.num.get.members] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the list of num_get<> non-virtual members on page 22-23, the member that parses bool values was omitted from the list of definitions of non-virtual members, though it is listed in the class definition and the corresponding virtual is listed everywhere appropriate.

Proposed resolution:

Add at the beginning of 22.2.2.1.1 another get member for bool&, copied from the entry in 22.2.2.1 .

---

19. "Noconv" definition too vague

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the definitions of codecvt<>::do_out and do_in, they are specified to return noconv if "no conversion is needed". This definition is too vague, and does not say normatively what is done with the buffers.

Proposed resolution:

Change the entry for noconv in the table under paragraph 4 in section 22.2.1.5.2 to read:

`noconv`: `internT` and `externT` are the same type, and input sequence is identical to converted sequence.

Change the Note in paragraph 2 to normative text as follows:

If returns `noconv`, `internT` and `externT` are the same type and the converted sequence is identical to the input sequence `[from,from_next)`. `to_next` is set equal to `to`, the value of `state` is unchanged, and there are no changes to the values in `[to, to_limit)`.

---

20. Thousands_sep returns wrong type

Section: 22.2.3.1.2 [lib.facet.numpunct.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The synopsis for numpunct<>::do_thousands_sep, and the definition of numpunct<>::thousands_sep which calls it, specify that it returns a value of type char_type. Here it is erroneously described as returning a "string_type".

Proposed resolution:

In 22.2.3.1.2 , above paragraph 2, change "string_type" to "char_type".

---

21. Codecvt_byname<> instantiations

Section: 22.1.1.1.1 [lib.locale.category] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the second table in the section, captioned "Required instantiations", the instantiations for codecvt_byname<> have been omitted. These are necessary to allow users to construct a locale by name from facets.

Proposed resolution:

Add in 22.1.1.1.1 to the table captioned "Required instantiations", in the category "ctype" the lines

```
codecvt_byname<char,char,mbstate_t>,
codecvt_byname<wchar_t,char,mbstate_t>
```

---

22. Member open vs. flags

Section: 27.8.1.7 [lib.ifstream.members] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The description of basic_istream<>::open leaves unanswered questions about how it responds to or changes flags in the error status for the stream. A strict reading indicates that it ignores the bits and does not change them, which confuses users who do not expect eofbit and failbit to remain set after a successful open. There are three reasonable resolutions: 1) status quo 2) fail if fail(), ignore eofbit 3) clear failbit and eofbit on call to open().

Proposed resolution:

In 27.8.1.7 paragraph 3, *and* in 27.8.1.10 paragraph 3, under open() effects, add a footnote:

> A successful open does not change the error state.

---

24. "do_convert" doesn't exist

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The description of codecvt<>::do_out and do_in mentions a symbol "do_convert" which is not defined in the standard. This is a leftover from an edit, and should be "do_in and do_out".

Proposed resolution:

In 22.2.1.5 , paragraph 3, change "do_convert" to "do_in or do_out". Also, in 22.2.1.5.2 , change "do_convert()" to "do_in or do_out".

---

25. String operator<< uses width() value wrong

Section: 21.3.7.9 [lib.string.io] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the description of operator<< applied to strings, the standard says that uses the smaller of os.width() and str.size(), to pad "as described in stage 3" elsewhere; but this is inconsistent, as this allows no possibility of space for padding.

Proposed resolution:

Change 21.3.7.9 paragraph 4 from:
"... where `n` is the smaller of `os.width()` and `str.size()`; ..."
to:
"... where `n` is the larger of `os.width()` and `str.size()`; ..."

26. Bad sentry example

Section: 27.6.1.1.2 [lib.istream::sentry] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In paragraph 6, the code in the example:

```
template <class charT, class traits = char_traits<charT> >
basic_istream<charT,traits>::sentry(
         basic_istream<charT,traits>& is, bool noskipws = false) {
    ...
    int_type c;
    typedef ctype<charT> ctype_type;
    const ctype_type& ctype = use_facet<ctype_type>(is.getloc());
    while ((c = is.rdbuf()->snextc()) != traits::eof()) {
      if (ctype.is(ctype.space,c)==0) {
        is.rdbuf()->sputbackc (c);
        break;
      }
    }
    ...
 }
```

fails to demonstrate correct use of the facilities described. In particular, it fails to use traits operators, and specifies incorrect semantics. (E.g. it specifies skipping over the first character in the sequence without examining it.)

Proposed resolution:

Remove the example above from 27.6.1.1.2 paragraph 6.

Rationale:

The originally proposed replacement code for the example was not correct. The LWG tried in Kona and again in Tokyo to correct it without success. In Tokyo, an implementor reported that actual working code ran over one page in length and was quite complicated. The LWG decided that it would be counter-productive to include such a lengthy example, which might well still contain errors.

---

27. String::erase(range) yields wrong iterator

Section: 21.3.5.5 [lib.string::erase] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The string::erase(iterator first, iterator last) is specified to return an element one place beyond the next element after the last one erased. E.g. for the string "abcde", erasing the range [’b’..’d’) would yield an iterator for element ’e’, while ’d’ has not been erased.

Proposed resolution:

In 21.3.5.5 , paragraph 10, change:

> Returns: an iterator which points to the element immediately following _last_ prior to the element being erased.

to read

> Returns: an iterator which points to the element pointed to by _last_ prior to the other elements being erased.

28. Ctype<char>is ambiguous

Section: 22.2.1.3.2 [lib.facet.ctype.char.members] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The description of the vector form of ctype<char>::is can be interpreted to mean something very different from what was intended. Paragraph 4 says

> Effects: The second form, for all *p in the range [low, high), assigns vec[p-low] to table()[(unsigned char)*p].

This is intended to copy the value indexed from table()[] into the place identified in vec[].

Proposed resolution:

Change 22.2.1.3.2 , paragraph 4, to read

> Effects: The second form, for all *p in the range [low, high), assigns into vec[p-low] the value table()[(unsigned char)*p].

---

29. Ios_base::init doesn't exist

Section: 27.3.1 [lib.narrow.stream.objects] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

Sections 27.3.1 and 27.3.2 mention a function ios_base::init, which is not defined. Probably they mean basic_ios<>::init, defined in 27.4.4.1 , paragraph 3.

Proposed resolution:

[R12: modified to include paragraph 5.]

In 27.3.1 paragraph 2 and 5, change

> ios_base::init

to

> basic_ios<char>::init

Also, make a similar change in 27.3.2 except it should read

> basic_ios<wchar_t>::init

---

30. Wrong header for LC_*

Section: 22.1.1.1.1 [lib.locale.category] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

Paragraph 2 implies that the C macros LC_CTYPE etc. are defined in <cctype>, where they are in fact defined elsewhere to appear in <clocale>.

Proposed resolution:

In 22.1.1.1.1 , paragraph 2, change "<cctype>" to read "<clocale>".

---

31. Immutable locale values

Section: 22.1.1 [lib.locale] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

Paragraph 6, says "An instance of `locale` is *immutable*; once a facet reference is obtained from it, ...". This has caused some confusion, because locale variables are manifestly assignable.

Proposed resolution:

In 22.1.1 replace paragraph 6

> An instance of `locale` is immutable; once a facet reference is obtained from it, that reference remains usable as long as the locale value itself exists.

with

> Once a facet reference is obtained from a locale object by calling use_facet<>, that reference remains usable, and the results from member functions of it may be cached and re-used, as long as some locale object refers to that facet.

---

32. Pbackfail description inconsistent

Section: 27.5.2.4.4 [lib.streambuf.virt.pback] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The description of the required state before calling virtual member basic_streambuf<>::pbackfail requirements is inconsistent with the conditions described in 27.5.2.2.4 [lib.streambuf.pub.pback] where member sputbackc calls it. Specifically, the latter says it calls pbackfail if:

> traits::eq(c,gptr()[-1]) is false

where pbackfail claims to require:

> traits::eq(*gptr(),traits::to_char_type(c)) returns false

It appears that the pbackfail description is wrong.

Proposed resolution:

In 27.5.2.4.4 , paragraph 1, change:

```
"traits::eq(*gptr(),traits::to_char_type( c))"
```

to

```
"traits::eq(traits::to_char_type(c),gptr()[-1])"
```

Rationale:

Note deliberate reordering of arguments for clarity in addition to the correction of the argument value.

---

33. Codecvt<> mentions from_type

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the table defining the results from do_out and do_in, the specification for the result *error* says

encountered a from_type character it could not convert

but from_type is not defined. This clearly is intended to be an externT for do_in, or an internT for do_out.

Proposed resolution:

In 22.2.1.5.2 paragraph 4, replace the definition in the table for the case of _error_ with

encountered a character in `[from,from_end)` that it could not convert.

---

34. True/falsename() not in ctype<>

Section: 22.2.2.2.2 [lib.facet.num.put.virtuals] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In paragraph 19, Effects:, members truename() and falsename are used from facet ctype<charT>, but it has no such members. Note that this is also a problem in 22.2.2.1.2, addressed in (4).

Proposed resolution:

In 22.2.2.2.2 , paragraph 19, in the Effects: clause for member put(...., bool), replace the initialization of the string_type value s as follows:

```
const numpunct& np = use_facet<numpunct<charT> >(loc);
string_type s = val ? np.truename() : np.falsename();
```

---

35. No manipulator unitbuf in synopsis

Section: 27.4 [lib.iostreams.base] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In 27.4.5.1 , we have a definition for a manipulator named "unitbuf". Unlike other manipulators, it’s not listed in synopsis. Similarly for "nounitbuf".

Proposed resolution:

Add to the synopsis for <ios> in 27.4 , after the entry for "nouppercase", the prototypes:

```
ios_base& unitbuf(ios_base& str);
ios_base& nounitbuf(ios_base& str);
```

---

36. Iword & pword storage lifetime omitted

Section: 27.4.2.5 [lib.ios.base.storage] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the definitions for ios_base::iword and pword, the lifetime of the storage is specified badly, so that an implementation which only keeps the last value stored appears to conform. In particular, it says:

The reference returned may become invalid after another call to the object’s iword member with a different index ...

This is not idle speculation; at least one implementation was done this way.

Proposed resolution:

Add in 27.4.2.5 , in both paragraph 2 and also in paragraph 4, replace the sentence:

> The reference returned may become invalid after another call to the object's iword [pword] member with a different index, after a call to its copyfmt member, or when the object is destroyed.

with:

> The reference returned is invalid after any other operations on the object. However, the value of the storage referred to is retained, so that until the next call to copyfmt, calling iword [pword] with the same index yields another reference to the same value.

substituting "iword" or "pword" as appropriate.

---

37. Leftover "global" reference

Section: 22.1.1 [lib.locale] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

In the overview of locale semantics, paragraph 4, is the sentence

> If Facet is not present in a locale (or, failing that, in the global locale), it throws the standard exception bad_cast.

This is not supported by the definition of use_facet<>, and represents semantics from an old draft.

Proposed resolution:

In 22.1.1 , paragraph 4, delete the parenthesized expression

> (or, failing that, in the global locale)

---

38. Facet definition incomplete

Section: 22.1.2 [lib.locale.global.templates] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

It has been noticed by Esa Pulkkinen that the definition of "facet" is incomplete. In particular, a class derived from another facet, but which does not define a member *id*, cannot safely serve as the argument *F* to use_facet<F>(loc), because there is no guarantee that a reference to the facet instance stored in *loc* is safely convertible to *F*.

Proposed resolution:

In the definition of std::use_facet<>(), replace the text in paragraph 1 which reads:

> Get a reference to a facet of a locale.

with:

> Requires: `Facet` is a facet class whose definition contains the public static member `id` as defined in 22.1.1.1.2 .

[Kona: strike as overspecification the text "(not inherits)" from the original resolution, which read "... whose definition contains (not inherits) the public static member `id`..."]

---

39. istreambuf_iterator<>::operator++(int) definition garbled

Section: 24.5.3.4 [lib.istreambuf.iterator::op++] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

Following the definition of istreambuf_iterator<>::operator++(int) in paragraph 3, the standard contains three lines of garbage text left over from a previous edit.

```
istreambuf_iterator<charT,traits> tmp = *this;
sbuf_->sbumpc();
return(tmp);
```

Proposed resolution:

In 24.5.3.4 , delete the three lines of code at the end of paragraph 3.

---

40. Meaningless normative paragraph in examples

Section: 22.2.8 [lib.facets.examples] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

Paragraph 3 of the locale examples is a description of part of an implementation technique that has lost its referent, and doesn't mean anything.

Proposed resolution:

Delete 22.2.8 paragraph 3 which begins "This initialization/identification system depends...", or (at the editor's option) replace it with a place-holder to keep the paragraph numbering the same.

---

41. Ios_base needs clear(), exceptions()

Section: 27.4.2 [lib.ios.base] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The description of ios_base::iword() and pword() in 27.4.2.4 , say that if they fail, they "set badbit, which may throw an exception". However, ios_base offers no interface to set or to test badbit; those interfaces are defined in basic_ios<>.

Proposed resolution:

Change the description in 27.4.2.5 in paragraph 2, and also in paragraph 4, as follows. Replace

> If the function fails it sets badbit, which may throw an exception.

with

> If the function fails, and `*this` is a base sub-object of a `basic_ios<>` object or sub-object, the effect is equivalent to calling `basic_ios<>::setstate(badbit)` on the derived object (which may throw `failure`).

[Kona: LWG reviewed wording; setstate(failbit) changed to setstate(badbit).]

---

42. String ctors specify wrong default allocator

Section: 21.3 [lib.basic.string] **Status:** DR **Submitter:** Nathan Myers **Date:** 6 Aug 1998

The basic_string<> copy constructor:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos, const Allocator& a = Allocator());
```

specifies an Allocator argument default value that is counter-intuitive. The natural choice for a the allocator to copy from is str.get_allocator(). Though this cannot be expressed in default-argument notation, overloading suffices.

Alternatively, the other containers in Clause 23 (deque, list, vector) do not have this form of constructor, so it is inconsistent, and an evident source of confusion, for basic_string<> to have it, so it might better be removed.

Proposed resolution:

In 21.3 , replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str);
basic_string(const basic_string& str, size_type pos, size_type n = npos,
             const Allocator& a = Allocator());
```

In 21.3.1 , replace the copy constructor declaration as above. Add to paragraph 5, Effects:

> In the first form, the Allocator value used is copied from `str.get_allocator()`.

Rationale:

The LWG believes the constructor is actually broken, rather than just an unfortunate design choice.

The LWG considered two other possible resolutions:

A. In 21.3 , replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos);
basic_string(const basic_string& str, size_type pos,
             size_type n, const Allocator& a);
```

In 21.3.1 , replace the copy constructor declaration as above. Add to paragraph 5, Effects:

> When no `Allocator` argument is provided, the string is constructed using the value `str.get_allocator()`.

B. In 21.3 , and also in 21.3.1 , replace the declaration of the copy constructor as follows:

```
basic_string(const basic_string& str, size_type pos = 0,
             size_type n = npos);
```

The proposed resolution reflects the original intent of the LWG. It was also noted by Pete Becker that this fix "will cause a small amount of existing code to now work correctly."

[Kona: issue editing snafu fixed - the proposed resolution now correctly reflects the LWG consensus.]

---

46. Minor Annex D errors

Section: D.7 [depr.str.strstreams] **Status:** DR **Submitter:** Brendan Kehoe **Date:** 1 Jun 1998

See lib-6522 and edit-814.

Proposed resolution:

Change D.7.1 (since streambuf is a typedef of basic_streambuf<char>) from:

```
        virtual streambuf<char>* setbuf(char* s, streamsize n);
```

to:

```
        virtual streambuf* setbuf(char* s, streamsize n);
```

In D.7.4 insert the semicolon now missing after int_type:

```
    namespace std {
      class strstream
        : public basic_iostream<char> {
      public:
        // Types
        typedef char                                  char_type;
        typedef typename char_traits<char>::int_type int_type
        typedef typename char_traits<char>::pos_type pos_type;
```

---

47. Imbue() and getloc() Returns clauses swapped

Section: 27.4.2.3 [lib.ios.base.locales] **Status:** DR **Submitter:** Matt Austern **Date:** 21 Jun 1998

Section 27.4.2.3 specifies how imbue() and getloc() work. That section has two RETURNS clauses, and they make no sense as stated. They make perfect sense, though, if you swap them. Am I correct in thinking that paragraphs 2 and 4 just got mixed up by accident?

Proposed resolution:

In 27.4.2.3 swap paragraphs 2 and 4.

---

48. Use of non-existent exception constructor

Section: 27.4.2.1.1 [lib.ios::failure] **Status:** DR **Submitter:** Matt Austern **Date:** 21 Jun 1998

27.4.2.1.1, paragraph 2, says that class failure initializes the base class, exception, with exception(msg). Class exception (see 18.6.1) has no such constructor.

Proposed resolution:

Replace 27.4.2.1.1 , paragraph 2, with

EFFECTS: Constructs an object of class `failure`.

---

50. Copy constructor and assignment operator of ios_base

Section: 27.4.2 [lib.ios.base] **Status:** DR **Submitter:** Matt Austern **Date:** 21 Jun 1998

As written, ios_base has a copy constructor and an assignment operator. (Nothing in the standard says it doesn't have one, and all classes have copy constructors and assignment operators unless you take specific steps to avoid them.) However, nothing in 27.4.2 says what the copy constructor and assignment operator do.

My guess is that this was an oversight, that ios_base is, like basic_ios, not supposed to have a copy constructor or an assignment operator.

Jerry Schwarz comments: Yes, its an oversight, but in the opposite sense to what you're suggesting. At one point there was a definite intention that you could copy ios_base. It's an easy way to save the entire state of a stream for future use. As you note, to carry out that intention would have required a explicit description of the semantics (e.g. what happens to the iarray and parray stuff).

Proposed resolution:

In 27.4.2 , class ios_base, specify the copy constructor and operator= members as being private.

Rationale:

The LWG believes the difficulty of specifying correct semantics outweighs any benefit of allowing ios_base objects to be copyable.

---

51. Requirement to not invalidate iterators missing

Section: 23.1 [lib.container.requirements] **Status:** DR **Submitter:** David Vandevoorde **Date:** 23 Jun 1998

The std::sort algorithm can in general only sort a given sequence by moving around values. The list<>::sort() member on the other hand could move around values or just update internal pointers. Either method can leave iterators into the list<> dereferencable, but they would point to different things.

Does the FDIS mandate anywhere which method should be used for list<>::sort()?

Matt Austern comments:

I think you've found an omission in the standard.

The library working group discussed this point, and there was supposed to be a general requirement saying that list, set, map, multiset, and multimap may not invalidate iterators, or change the values that iterators point to, except when an operation does it explicitly. So, for example, insert() doesn't invalidate any iterators and erase() and remove() only invalidate iterators pointing to the elements that are being erased.

I looked for that general requirement in the FDIS, and, while I found a limited form of it for the sorted associative containers, I didn't find it for list. It looks like it just got omitted.

The intention, though, is that list<>::sort does not invalidate any iterators and does not change the values that any iterator points to. There would be no reason to have the member function otherwise.

Proposed resolution:

Add a new paragraph at the end of 23.1:

> Unless otherwise specified (either explicitly or by defining a function in terms of other functions), invoking a container member function or passing a container as an argument to a library function shall not invalidate iterators to, or change the values of, objects within that container.

Rationale:

This was US issue CD2-23-011; it was accepted in London but the change was not made due to an editing oversight. The wording in the proposed resolution below is somewhat updated from CD2-23-011, particularly the addition of the phrase "or change the values of"

---

52. Small I/O problems

Section: 27.4.3.2 [lib.fpos.operations] **Status:** DR **Submitter:** Matt Austern **Date:** 23 Jun 1998

First, 27.4.4.1 , table 89. This is pretty obvious: it should be titled "basic_ios<>() effects", not "ios_base() effects".

[The second item is a duplicate; see issue 6 for resolution.]

Second, 27.4.3.2 table 88 . There are a couple different things wrong with it, some of which I've already discussed with Jerry, but the most obvious mechanical sort of error is that it uses expressions like P(i) and p(i), without ever defining what sort of thing "i" is.

(The other problem is that it requires support for streampos arithmetic. This is impossible on some systems, i.e. ones where file position is a complicated structure rather than just a number. Jerry tells me that the intention was to require syntactic support for streampos arithmetic, but that it wasn't actually supposed to do anything meaningful except on platforms, like Unix, where genuine arithmetic is possible.)

Proposed resolution:

Change 27.4.4.1 table 89 title from "ios_base() effects" to "basic_ios<>() effects".

---

53. Basic_ios destructor unspecified

Section: 27.4.4.1 [lib.basic.ios.cons] **Status:** DR **Submitter:** Matt Austern **Date:** 23 Jun 1998

There's nothing in 27.4.4 saying what basic_ios's destructor does. The important question is whether basic_ios::~basic_ios() destroys rdbuf().

Proposed resolution:

Add after 27.4.4.1 paragraph 2:

```
virtual ~basic_ios();
```

Notes: The destructor does not destroy `rdbuf()`.

Rationale:

The LWG reviewed the additional question of whether or not `rdbuf(0)` may set `badbit`. The answer is clearly yes; it may be set via `clear()`. See 27.4.4.2 , paragraph 6. This issue was reviewed at length by the LWG, which removed from the original proposed resolution a footnote which incorrectly said "`rdbuf(0)` does not set `badbit`".

---

54. Basic_streambuf's destructor

Section: 27.5.2.1 [lib.streambuf.cons] **Status:** DR **Submitter:** Matt Austern **Date:** 25 Jun 1998

The class synopsis for basic_streambuf shows a (virtual) destructor, but the standard doesn't say what that destructor does. My assumption is that it does nothing, but the standard should say so explicitly.

Proposed resolution:

Add after 27.5.2.1 paragraph 2:

```
virtual  ~basic_streambuf();
```

Effects: None.

---

55. Invalid stream position is undefined

Section: 27 [lib.input.output] **Status:** DR **Submitter:** Matt Austern **Date:** 26 Jun 1998

Several member functions in clause 27 are defined in certain circumstances to return an "invalid stream position", a term that is defined nowhere in the standard. Two places (27.5.2.4.2, paragraph 4, and 27.8.1.4, paragraph 15) contain a cross-reference to a definition in _lib.iostreams.definitions_, a nonexistent section.

I suspect that the invalid stream position is just supposed to be pos_type(-1). Probably best to say explicitly in (for example) 27.5.2.4.2 that the return value is pos_type(-1), rather than to use the term "invalid stream position", define that term somewhere, and then put in a cross-reference.

The phrase "invalid stream position" appears ten times in the C++ Standard. In seven places it refers to a return value, and it should be changed. In three places it refers to an argument, and it should not be changed. Here are the three places where "invalid stream position" should not be changed:

27.7.1.3 , paragraph 14
27.8.1.4 , paragraph 14
D.7.1.3 , paragraph 17 [lib.stringbuf.virtuals]

Proposed resolution:

In 27.5.2.4.2 , paragraph 4, change "Returns an object of class pos_type that stores an invalid stream position (_lib.iostreams.definitions_)" to "Returns `pos_type(off_type(-1))`".

In 27.5.2.4.2 , paragraph 6, change "Returns an object of class pos_type that stores an invalid stream position" to "Returns `pos_type(off_type(-1))`".

In 27.7.1.3 , paragraph 13, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`".

In 27.8.1.4 , paragraph 13, change "returns an invalid stream position (27.4.3)" to "returns `pos_type(off_type(-1))`"

In 27.8.1.4 , paragraph 15, change "Otherwise returns an invalid stream position (_lib.iostreams.definitions_)" to "Otherwise returns `pos_type(off_type(-1))`"

In D.7.1.3 , paragraph 15, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`"

In D.7.1.3 , paragraph 18, change "the object stores an invalid stream position" to "the return value is `pos_type(off_type(-1))`"

---

56. Showmanyc's return type

Section: 27.5.2 [lib.streambuf] **Status:** DR **Submitter:** Matt Austern **Date:** 29 Jun 1998

The class summary for basic_streambuf<>, in 27.5.2, says that showmanyc has return type int. However, 27.5.2.4.3 says that its return type is streamsize.

Proposed resolution:

Change `showmanyc`'s return type in the 27.5.2 class summary to `streamsize`.

---

57. Mistake in char_traits

Section: 21.1.3.2 [lib.char.traits.specializations.wchar.t] **Status:** DR **Submitter:** Matt Austern **Date:** 1 Jul 1998

21.1.3.2, paragraph 3, says "The types streampos and wstreampos may be different if the implementation supports no shift encoding in narrow-oriented iostreams but supports one or more shift encodings in wide-oriented streams".

That's wrong: the two are the same type. The <iosfwd> summary in 27.2 says that streampos and wstreampos are, respectively, synonyms for fpos<char_traits<char>::state_type> and fpos<char_traits<wchar_t>::state_type>, and, flipping back to clause 21, we see in 21.1.3.1 and 21.1.3.2 that char_traits<char>::state_type and char_traits<wchar_t>::state_type must both be mbstate_t.

Proposed resolution:

Remove the sentence in 21.1.3.2 paragraph 3 which begins "The types streampos and wstreampos may be different..." .

---

59. Ambiguity in specification of gbump

Section: 27.5.2.3.1 [lib.streambuf.get.area] **Status:** DR **Submitter:** Matt Austern **Date:** 28 Jul 1998

27.5.2.3.1 says that basic_streambuf::gbump() "Advances the next pointer for the input sequence by n."

The straightforward interpretation is that it is just gptr() += n. An alternative interpretation, though, is that it behaves as if it calls sbumpc n times. (The issue, of course, is whether it might ever call underflow.) There is a similar ambiguity in the case of pbump.

(The "classic" AT&T implementation used the former interpretation.)

Proposed resolution:

Change 27.5.2.3.1 paragraph 4 gbump effects from:

> Effects: Advances the next pointer for the input sequence by n.

to:

> Effects: Adds `n` to the next pointer for the input sequence.

Make the same change to 27.5.2.3.2 paragraph 4 pbump effects.

---

60. What is a formatted input function?

Section: 27.6.1.2.1 [lib.istream.formatted.reqmts] **Status:** DR **Submitter:** Matt Austern **Date:** 3 Aug 1998

Paragraph 1 of 27.6.1.2.1 contains general requirements for all formatted input functions. Some of the functions defined in section 27.6.1.2 explicitly say that those requirements apply ("Behaves like a formatted input member (as described in 27.6.1.2.1)"), but others don't. The question: is 27.6.1.2.1 supposed to apply to everything in 27.6.1.2, or only to those member functions that explicitly say "behaves like a formatted input member"? Or to put it differently: are we to assume that everything that appears in a section called "Formatted input functions" really is a formatted input function? I assume that 27.6.1.2.1 is intended to apply to the arithmetic extractors (27.6.1.2.2), but I assume that it is not intended to apply to extractors like

```
basic_istream& operator>>(basic_istream& (*pf)(basic_istream&));
```

and

```
basic_istream& operator>>(basic_streammbuf*);
```

There is a similar ambiguity for unformatted input, formatted output, and unformatted output.

Comments from Judy Ward: It seems like the problem is that the basic_istream and basic_ostream operator <<()'s that are used for the manipulators and streambuf* are in the wrong section and should have their own separate section or be modified to make it clear that the "Common requirements" listed in section 27.6.1.2.1 (for basic_istream) and section 27.6.2.5.1 (for basic_ostream) do not apply to them.

Additional comments from Dietmar Kühl: It appears to be somewhat nonsensical to consider the functions defined in 27.6.1.2.3 paragraphs 1 to 5 to be "Formatted input function" but since these functions are defined in a section labeled "Formatted input functions" it is unclear to me whether these operators are considered formatted input functions which have to conform to the "common requirements" from 27.6.1.2.1 : If this is the case, all manipulators, not just `ws`, would skip whitespace unless `noskipws` is set (... but setting `noskipws` using the manipulator syntax would also skip whitespace :-)

It is not clear which functions are to be considered unformatted input functions. As written, it seems that all functions in 27.6.1.3 are unformatted input functions. However, it does not really make much sense to construct a sentry object for `gcount()`, `sync()`, ... Also it is unclear what happens to the `gcount()` if eg. `gcount()`, `putback()`, `unget()`, or `sync()` is called: These functions don't extract characters, some of them even "unextract" a character. Should this still be reflected in `gcount()`? Of course, it could be read as if after a call to `gcount()` `gcount()` return `0` (the last unformatted input function, `gcount()`, didn't extract any character) and after a call to `putback()` `gcount()` returns `-1` (the last unformatted input function `putback()` did "extract" back into the stream). Correspondingly for `unget()`. Is this what is intended? If so, this should be clarified. Otherwise, a corresponding clarification should be used.

Proposed resolution:

In 27.6.1.2.2 [lib.istream.formatted.arithmetic], paragraph 1. Change the beginning of the second sentence from "The conversion occurs" to "These extractors behave as formatted input functions (as described in 27.6.1.2.1). After a sentry object is constructed, the conversion occurs"

In 27.6.1.2.3, [lib.istream::extractors], before paragraph 1. Add an effects clause. "Effects: None. This extractor does not behave as a formatted input function (as described in 27.6.1.2.1).

In 27.6.1.2.3, [lib.istream::extractors], paragraph 2. Change the effects clause to "Effects: Calls pf(*this). This extractor does not behave as a formatted input function (as described in 27.6.1.2.1).

In 27.6.1.2.3, [lib.istream::extractors], paragraph 4. Change the effects clause to "Effects: Calls pf(*this). This extractor does not behave as a formatted input function (as described in 27.6.1.2.1).

In 27.6.1.2.3, [lib.istream::extractors], paragraph 12. Change the first two sentences from "If sb is null, calls setstate(failbit), which may throw ios_base::failure (27.4.4.3). Extracts characters from *this..." to "Behaves as a formatted input function (as described in 27.6.1.2.1). If sb is null, calls setstate(failbit), which may throw ios_base::failure (27.4.4.3). After a sentry object is constructed, extracts characters from *this...".

In 27.6.1.3, [lib.istream.unformatted], before paragraph 2. Add an effects clause. "Effects: none. This member function does not behave as an unformatted input function (as described in 27.6.1.3, paragraph 1)."

In 27.6.1.3, [lib.istream.unformatted], paragraph 3. Change the beginning of the first sentence of the effects clause from "Extracts a character" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts a character"

In 27.6.1.3, [lib.istream.unformatted], paragraph 5. Change the beginning of the first sentence of the effects clause from "Extracts a character" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts a character"

In 27.6.1.3, [lib.istream.unformatted], paragraph 5. Change the beginning of the first sentence of the effects clause from "Extracts characters" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts characters"

[No change needed in paragraph 10, because it refers to paragraph 7.]

In 27.6.1.3, [lib.istream.unformatted], paragraph 12. Change the beginning of the first sentence of the effects clause from "Extracts characters" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts characters"

[No change needed in paragraph 15.]

In 27.6.1.3, [lib.istream.unformatted], paragraph 17. Change the beginning of the first sentence of the effects clause from "Extracts characters" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts characters"

[No change needed in paragraph 23.]

In 27.6.1.3, [lib.istream.unformatted], paragraph 24. Change the beginning of the first sentence of the effects clause from "Extracts characters" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, extracts characters"

In 27.6.1.3, [lib.istream.unformatted], before paragraph 27. Add an Effects clause: "Effects: Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, reads but does not extract the current input character."

In 27.6.1.3, [lib.istream.unformatted], paragraph 28. Change the first sentence of the Effects clause from "If !good() calls" to Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, if !good() calls"

In 27.6.1.3, [lib.istream.unformatted], paragraph 30. Change the first sentence of the Effects clause from "If !good() calls" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, if !good() calls"

In 27.6.1.3, [lib.istream.unformatted], paragraph 32. Change the first sentence of the Effects clause from "If !good() calls..." to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, if !good() calls..." Add a new sentence to the end of the Effects clause: "[Note: this function extracts no characters, so the value returned by the next call to gcount() is 0.]"

In 27.6.1.3, [lib.istream.unformatted], paragraph 34. Change the first sentence of the Effects clause from "If !good() calls" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1). After constructing a sentry object, if !good() calls". Add a new sentence to the end of the Effects clause: "[Note: this function extracts no

characters, so the value returned by the next call to gcount() is 0.]"

In 27.6.1.3, [lib.istream.unformatted], paragraph 36. Change the first sentence of the Effects clause from "If !rdbuf() is" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to gcount(). After constructing a sentry object, if rdbuf() is"

In 27.6.1.3, [lib.istream.unformatted], before paragraph 37. Add an Effects clause: "Effects: Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to gcount()." Change the first sentence of paragraph 37 from "if fail()" to "after constructing a sentry object, if fail()".

In 27.6.1.3, [lib.istream.unformatted], paragraph 38. Change the first sentence of the Effects clause from "If fail()" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to gcount(). After constructing a sentry object, if fail()

In 27.6.1.3, [lib.istream.unformatted], paragraph 40. Change the first sentence of the Effects clause from "If fail()" to "Behaves as an unformatted input function (as described in 27.6.1.3, paragraph 1), except that it does not count the number of characters extracted and does not affect the value returned by subsequent calls to gcount(). After constructing a sentry object, if fail()

In 27.6.2.5.2 [lib.ostream.inserters.arithmetic], paragraph 1. Change the beginning of the third sentence from "The formatting conversion" to "These extractors behave as formatted output functions (as described in 27.6.2.5.1). After the sentry object is constructed, the conversion occurs".

In 27.6.2.5.3 [lib.ostream.inserters], before paragraph 1. Add an effects clause: "Effects: None. Does not behave as a formatted output function (as described in 27.6.2.5.1).".

In 27.6.2.5.3 [lib.ostream.inserters], paragraph 2. Change the effects clause to "Effects: calls pf(*this). This extractor does not behave as a formatted output function (as described in 27.6.2.5.1).".

In 27.6.2.5.3 [lib.ostream.inserters], paragraph 4. Change the effects clause to "Effects: calls pf(*this). This extractor does not behave as a formatted output function (as described in 27.6.2.5.1).".

In 27.6.2.5.3 [lib.ostream.inserters], paragraph 6. Change the first sentence from "If sb" to "Behaves as a formatted output function (as described in 27.6.2.5.1). After the sentry object is constructed, if sb".

In 27.6.2.6 [lib.ostream.unformatted], paragraph 2. Change the first sentence from "Inserts the character" to "Behaves as an unformatted output function (as described in 27.6.2.6, paragraph 1). After constructing a sentry object, inserts the character".

In 27.6.2.6 [lib.ostream.unformatted], paragraph 5. Change the first sentence from "Obtains characters" to "Behaves as an unformatted output function (as described in 27.6.2.6, paragraph 1). After constructing a sentry object, obtains characters".

In 27.6.2.6 [lib.ostream.unformatted], paragraph 7. Add a new sentence at the end of the paragraph: "Does not behave as an unformatted output function (as described in 27.6.2.6, paragraph 1)."

Rationale:

See J16/99-0043==WG21/N1219, Proposed Resolution to Library Issue 60, by Judy Ward and Matt Austern. This proposed resolution is section VI of that paper.

---

61. Ambiguity in iostreams exception policy

Section: 27.6.1.3 [lib.istream.unformatted] **Status:** DR **Submitter:** Matt Austern **Date:** 6 Aug 1998

The introduction to the section on unformatted input (27.6.1.3) says that every unformatted input function catches all exceptions that were thrown during input, sets badbit, and then conditionally rethrows the exception. That seems clear enough. Several of the specific functions, however, such as get() and read(), are documented in some circumstances as setting eofbit and/or failbit. (The standard notes, correctly, that setting eofbit or failbit can sometimes result in an exception being thrown.) The question: if one of these functions throws an exception triggered by setting failbit, is this an exception "thrown during input" and hence covered by 27.6.1.3, or does 27.6.1.3 only refer to a limited class of exceptions? Just to make this concrete, suppose you have the following snippet.

```
char buffer[N];
istream is;
...
is.exceptions(istream::failbit); // Throw on failbit but not on badbit.
is.read(buffer, N);
```

Now suppose we reach EOF before we've read N characters. What iostate bits can we expect to be set, and what exception (if any) will be thrown?

Proposed resolution:

In 27.6.1.3, paragraph 1, after the sentence that begins "If an exception is thrown...", add the following parenthetical comment: "(Exceptions thrown from `basic_ios<>::clear()` are not caught or rethrown.)"

Rationale:

The LWG looked to two alternative wordings, and choose the proposed resolution as better standardese.

---

62. Sync's return value

Section: 27.6.1.3 [lib.istream.unformatted] **Status:** DR **Submitter:** Matt Austern **Date:** 6 Aug 1998

The Effects clause for sync() (27.6.1.3, paragraph 36) says that it "calls rdbuf()->pubsync() and, if that function returns -1 ... returns traits::eof()."

That looks suspicious, because traits::eof() is of type traits::int_type while the return type of sync() is int.

Proposed resolution:

In 27.6.1.3 , paragraph 36, change "returns `traits::eof()`" to "returns `-1`".

---

63. Exception-handling policy for unformatted output

Section: 27.6.2.6 [lib.ostream.unformatted] **Status:** DR **Submitter:** Matt Austern **Date:** 11 Aug 1998

Clause 27 details an exception-handling policy for formatted input, unformatted input, and formatted output. It says nothing for unformatted output (27.6.2.6). 27.6.2.6 should either include the same kind of exception-handling policy as in the other three places, or else it should have a footnote saying that the omission is deliberate.

Proposed resolution:

In 27.6.2.6, paragraph 1, replace the last sentence ("In any case, the unformatted output function ends by destroying the sentry object, then returning the value specified for the formatted output function.") with the following text:

> If an exception is thrown during output, then `ios::badbit` is turned on [Footnote: without causing an `ios::failure` to be thrown.] in `*this`'s error state. If `(exceptions() & badbit) != 0` then the exception is rethrown. In any case, the unformatted output function ends by destroying the sentry object, then, if no exception was thrown, returning the value specified for the formatted output function.

Rationale:

This exception-handling policy is consistent with that of formatted input, unformatted input, and formatted output.

---

64. Exception handling in `basic_istream::operator>>(basic_streambuf*)`

Section: 27.6.1.2.3 [lib.istream::extractors] **Status:** DR **Submitter:** Matt Austern **Date:** 11 Aug 1998

27.6.1.2.3, paragraph 13, is ambiguous. It can be interpreted two different ways, depending on whether the second sentence is read as an elaboration of the first.

Proposed resolution:

Replace 27.6.1.2.3 , paragraph 13, which begins "If the function inserts no characters ..." with:

> If the function inserts no characters, it calls `setstate(failbit)`, which may throw `ios_base::failure` (27.4.4.3). If it inserted no characters because it caught an exception thrown while extracting characters from `sb` and `failbit` is on in `exceptions()` (27.4.4.3), then the caught exception is rethrown.

---

66. Strstreambuf::setbuf

Section: D.7.1.3 [depr.strstreambuf.virtuals] **Status:** DR **Submitter:** Matt Austern **Date:** 18 Aug 1998

D.7.1.3, paragraph 19, says that strstreambuf::setbuf "Performs an operation that is defined separately for each class derived from strstreambuf". This is obviously an incorrect cut-and-paste from basic_streambuf. There are no classes derived from strstreambuf.

Proposed resolution:

D.7.1.3 , paragraph 19, replace the setbuf effects clause which currently says "Performs an operation that is defined separately for each class derived from strstreambuf" with: [depr.strstreambuf.virtuals]

> **Effects**: implementation defined, except that `setbuf(0,0)` has no effect.

---

68. Extractors for char* should store null at end

Section: 27.6.1.2.3 [lib.istream::extractors] **Status:** DR **Submitter:** Angelika Langer **Date:** 14 Jul 1998

Extractors for char* (27.6.1.2.3) do not store a null character after the extracted character sequence whereas the unformatted functions like get() do. Why is this?

Comment from Jerry Schwarz: There is apparently an editing glitch. You'll notice that the last item of the list of what stops extraction doesn't make any sense. It was supposed to be the line that said a null is stored.

Proposed resolution:

27.6.1.2.3 , paragraph 7, change the last list item from: [lib.istream::extractors]

> A null byte (`charT()`) in the next position, which may be the first position if no characters were extracted.

to become a new paragraph which reads:

> Operator>> then stores a null byte (`charT()`) in the next position, which may be the first position if no characters were extracted.

---

69. Must elements of a vector be contiguous?

Section: 23.2.4 [lib.vector] **Status:** DR **Submitter:** Andrew Koenig **Date:** 29 Jul 1998

The issue is this: Must the elements of a vector be in contiguous memory?

(Please note that this is entirely separate from the question of whether a vector iterator is required to be a pointer; the answer to that question is clearly "no," as it would rule out debugging implementations)

Proposed resolution:

Add the following text to the end of 23.2.4 , paragraph 1.

> The elements of a vector are stored contiguously, meaning that if v is a `vector<T, Allocator>` where T is some type other than `bool`, then it obeys the identity `&v[n] == &v[0] + n` for all `0 <= n < v.size()`.

Rationale:

The LWG feels that as a practical matter the answer is clearly "yes". There was considerable discussion as to the best way to express the concept of "contiguous", which is not directly defined in the standard. Discussion included:

- An operational definition similar to the above proposed resolution is already used for valarray (26.3.2.3).
- There is no need to explicitly consider a user-defined operator& because elements must be copyconstructible (23.1 para 3) and copyconstructible (20.1.3) specifies requirements for operator&.
- There is no issue of one-past-the-end because of language rules.

---

70. Uncaught_exception() missing throw() specification

Section: 18.6 [lib.support.exception], 18.6.4 [lib.uncaught] **Status:** DR **Submitter:** Steve Clamage
Date: Unknown

In article 3E04@pratique.fr, Valentin Bonnard writes:

uncaught_exception() doesn't have a throw specification.

It is intentional ? Does it means that one should be prepared to handle exceptions thrown from uncaught_exception() ?

uncaught_exception() is called in exception handling contexts where exception safety is very important.

Proposed resolution:

In 15.5.3 , paragraph 1, 18.6 , and 18.6.4 , add "throw()" to uncaught_exception().

---

71. Do_get_monthname synopsis missing argument

Section: 22.2.5.1 [lib.locale.time.get] **Status:** DR **Submitter:** Nathan Myers **Date:** 13 Aug 1998

The locale facet member `time_get<>::do_get_monthname` is described in 22.2.5.1.2 with five arguments, consistent with do_get_weekday and with its specified use by member get_monthname. However, in the synopsis, it is specified instead with four arguments. The missing argument is the "end" iterator value.

Proposed resolution:

In 22.2.5.1 , add an "end" argument to the declaration of member do_monthname as follows:

```
virtual iter_type do_get_monthname(iter_type s, iter_type end, ios_base&,
                                   ios_base::iostate& err, tm* t) const;
```

---

74. Garbled text for `codecvt::do_max_length`

Section: 22.2.1.5.2 [lib.locale.codecvt.virtuals] **Status:** DR **Submitter:** Matt Austern **Date:** 8 Sep 1998

The text of `codecvt::do_max_length`'s "Returns" clause (22.2.1.5.2, paragraph 11) is garbled. It has unbalanced parentheses and a spurious **n**.

Proposed resolution:

Replace 22.2.1.5.2 paragraph 11 with the following:

> **Returns**: The maximum value that `do_length(state, from, from_end, 1)` can return for any valid range `[from, from_end)` and `stateT` value `state`. The specialization `codecvt<char, char, mbstate_t>::do_max_length()` returns 1.

---

75. Contradiction in `codecvt::length`'s argument types

Section: 22.2.1.5 [lib.locale.codecvt] **Status:** DR **Submitter:** Matt Austern **Date:** 18 Sep 1998

The class synopses for classes `codecvt<>` (22.2.1.5) and `codecvt_byname<>` (22.2.1.6) say that the first parameter of the member functions `length` and `do_length` is of type `const stateT&`. The member function descriptions, however (22.2.1.5.1, paragraph 6; 22.2.1.5.2, paragraph 9) say that the type is `stateT&`. Either the synopsis or the summary must be changed.

If (as I believe) the member function descriptions are correct, then we must also add text saying how `do_length` changes its `stateT` argument.

Proposed resolution:

In 22.2.1.5 , and also in 22.2.1.6 , change the `stateT` argument type on both member `length()` and member `do_length()` from

```
const stateT&
```

to

```
stateT&
```

In 22.2.1.5.2 , add to the definition for member `do_length` a paragraph:

Effects: The effect on the `state` argument is ‘‘as if’’ it called `do_in(state, from, from_end, from, to, to+max, to)` for `to` pointing to a buffer of at least `max` elements.

---

78. Typo: event_call_back

Section: 27.4.2 [lib.ios.base] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

typo: event_call_back should be event_callback

Proposed resolution:

In the 27.4.2 synopsis change "event_call_back" to "event_callback".

---

79. Inconsistent declaration of polar()

Section: 26.2.1 [lib.complex.synopsis], 26.2.7 [lib.complex.value.ops] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

In 26.2.1 polar is declared as follows:

```
template<class T> complex<T> polar(const T&, const T&);
```

In 26.2.7 it is declared as follows:

```
template<class T> complex<T> polar(const T& rho, const T& theta = 0);
```

Thus whether the second parameter is optional is not clear.

Proposed resolution:

In 26.2.1 change:

```
template<class T> complex<T> polar(const T&, const T&);
```

to:

```
template<class T> complex<T> polar(const T& rho, const T& theta = 0);
```

---

80. Global Operators of complex declared twice

Section: 26.2.1 [lib.complex.synopsis], 26.2.2 [lib.complex] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

Both 26.2.1 and 26.2.2 contain declarations of global operators for class complex. This redundancy should be removed.

Proposed resolution:

Reduce redundancy according to the general style of the standard.

---

83. String::npos vs. string::max_size()

Section: 21.3 [lib.basic.string] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

Many string member functions throw if size is getting or exceeding npos. However, I wonder why they don't throw if size is getting or exceeding max_size() instead of npos. May be npos is known at compile time, while max_size() is known at runtime. However, what happens if size exceeds max_size() but not npos, then? It seems the standard lacks some clarifications here.

Proposed resolution:

After 21.3 paragraph 4 ("The functions described in this clause...") add a new paragraph:

> For any string operation, if as a result of the operation, `size()` would exceed `max_size()` then the operation throws `length_error`.

Rationale:

The LWG believes length_error is the correct exception to throw.

---

86. String constructors don't describe exceptions

Section: 21.3.1 [lib.string.cons] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

The constructor from a range:

```
template<class InputIterator>
         basic_string(InputIterator begin, InputIterator end,
                      const Allocator& a = Allocator());
```

lacks a throws clause. However, I would expect that it throws according to the other constructors if the numbers of characters in the range equals npos (or exceeds max_size(), see above).

Proposed resolution:

In 21.3.1 , Strike throws paragraphs for constructors which say "Throws: length_error if n == npos."

Rationale:

Throws clauses for length_error if n == npos are no longer needed because they are subsumed by the general wording added by the resolution for issue 83.

---

90. Incorrect description of operator >> for strings

Section: 21.3.7.9 [lib.string.io] **Status:** DR **Submitter:** Nico Josuttis **Date:** 29 Sep 1998

The effect of operator >> for strings contain the following item:

> `isspace(c,getloc())` is true for the next available input character c.

Here `getloc()` has to be replaced by `is.getloc()`.

Proposed resolution:

In 21.3.7.9 paragraph 1 Effects clause replace:

> `isspace(c,getloc())` is true for the next available input character c.

with:

> `isspace(c,is.getloc())` is true for the next available input character c.

---

103. set::iterator is required to be modifiable, but this allows modification of keys

Section: 23.1.2 [lib.associative.reqmts] **Status:** DR **Submitter:** AFNOR **Date:** 7 Oct 1998

Set::iterator is described as implementation-defined with a reference to the container requirement; the container requirement says that const_iterator is an iterator pointing to const T and iterator an iterator pointing to T.

23.1.2 paragraph 2 implies that the keys should not be modified to break the ordering of elements. But that is not clearly specified. Especially considering that the current standard requires that iterator for associative containers be different from const_iterator. Set, for example, has the following:

```
typedef implementation defined iterator;
       // See _lib.container.requirements_
```

23.1 actually requires that iterator type pointing to T (table 65). Disallowing user modification of keys by changing the standard to require an iterator for associative container to be the same as const_iterator would be overkill since that will unnecessarily significantly restrict the usage of associative container. A class to be used as elements of set, for example, can no longer be modified easily without either redesigning the class (using mutable on fields that have nothing to do with ordering), or using const_cast, which defeats requiring iterator to be const_iterator. The proposed solution goes in line with trusting user knows what he is doing. [lib.container.requirements]

Other Options Evaluated:

Option A. In 23.1.2 , paragraph 2, after first sentence, and before "In addition,...", add one line:

> Modification of keys shall not change their strict weak ordering.

Option B. Add three new sentences to 23.1.2 :

> At the end of paragraph 5: "Keys in an associative container are immutable." At the end of paragraph 6: "For associative containers where the value type is the same as the key type, both `iterator` and `const_iterator` are constant iterators. It is unspecified whether or not `iterator` and `const_iterator` are the same type."

Option C. To 23.1.2 , paragraph 3, which currently reads:

> The phrase ‘‘equivalence of keys’’ means the equivalence relation imposed by the comparison and not the operator== on keys. That is, two keys k1 and k2 in the same container are considered to be equivalent if for the comparison object comp, comp(k1, k2) == false && comp(k2, k1) == false.

add the following:

> For any two keys k1 and k2 in the same container, comp(k1, k2) shall return the same value whenever it is evaluated. [Note: If k2 is removed from the container and later reinserted, comp(k1, k2) must still return a consistent value but this value may be different than it was the first time k1 and k2 were in the same container. This is intended to allow usage like a string key that contains a filename, where comp compares file contents; if k2 is removed, the file is changed, and the same k2 (filename) is reinserted, comp(k1, k2) must again return a consistent value but this value may be different than it was the previous time k2 was in the container.]

Proposed resolution:

Add the following to 23.1.2 at the indicated location:

> At the end of paragraph 3: "For any two keys k1 and k2 in the same container, calling comp(k1, k2) shall always return the same value."
>
> At the end of paragraph 5: "Keys in an associative container are immutable."
>
> At the end of paragraph 6: "For associative containers where the value type is the same as the key type, both `iterator` and `const_iterator` are constant iterators. It is unspecified whether or not `iterator` and `const_iterator` are the same type."

Rationale:

Several arguments were advanced for and against allowing set elements to be mutable as long as the ordering was not effected. The argument which swayed the LWG was one of safety; if elements were mutable, there would be no compile-time way to detect of a simple user oversight which caused ordering to be modified. There was a report that this had actually happened in practice, and had been painful to diagnose. If users need to modify elements, it is possible to use mutable members or const_cast.

Simply requiring that keys be immutable is not sufficient, because the comparison object may indirectly (via pointers) operate on values outside of the keys.

The types `iterator` and `const_iterator` are permitted to be different types to allow for potential future work in which some member functions might be overloaded between the two types. No such member functions exist now, and the LWG believes that user functionality will not be impaired by permitting the two types to be the same. A function that operates on both iterator types can be defined for `const_iterator` alone, and can rely on the automatic conversion from `iterator` to `const_iterator`.

[Tokyo: The LWG crafted the proposed resolution and rationale.]

---

106. Numeric library private members are implementation defined

Section: 26.3.5 [lib.template.slice.array] **Status:** DR **Submitter:** AFNOR **Date:** 7 Oct 1998

This is the only place in the whole standard where the implementation has to document something private.

Proposed resolution:

Remove the comment which says "// remainder implementation defined" from:

- 26.3.5 [lib.template.slice.array]
- 26.3.7 [lib.template.gslice.array]
- 26.3.8 [lib.template.mask.array]
- 26.3.9 [lib.template.indirect.array]

108. Lifetime of exception::what() return unspecified

Section: 18.6.1 [lib.exception] **Status:** DR **Submitter:** AFNOR **Date:** 7 Oct 1998

In 18.6.1, paragraphs 8-9, the lifetime of the return value of exception::what() is left unspecified. This issue has implications with exception safety of exception handling: some exceptions should not throw bad_alloc.

Proposed resolution:

Add to 18.6.1 paragraph 9 (exception::what notes clause) the sentence:

> The return value remains valid until the exception object from which it is obtained is destroyed or a non-const member function of the exception object is called.

Rationale:

If an exception object has non-const members, they may be used to set internal state that should affect the contents of the string returned by `what()`.

---

110. istreambuf_iterator::equal not const

Section: 24.5.3 [lib.istreambuf.iterator], 24.5.3.5 [lib.istreambuf.iterator::equal] **Status:** DR **Submitter:** Nathan Myers **Date:** 15 Oct 1998

Member istreambuf_iterator<>::equal is not declared "const", yet 24.5.3.6 says that operator==, which is const, calls it. This is contradictory.

Proposed resolution:

In 24.5.3 and also in 24.5.3.5 , replace:

```
bool equal(istreambuf_iterator& b);
```

with:

```
bool equal(const istreambuf_iterator& b) const;
```

---

112. Minor typo in `ostreambuf_iterator` constructor

Section: 24.5.4.1 [lib.ostreambuf.iter.cons] **Status:** DR **Submitter:** Matt Austern **Date:** 20 Oct 1998

The **requires** clause for `ostreambuf_iterator`'s constructor from an `ostream_type` (24.5.4.1, paragraph 1) reads "*s* is not null". However, *s* is a reference, and references can't be null.

Proposed resolution:

In 24.5.4.1 :

Move the current paragraph 1, which reads "Requires: s is not null.", from the first constructor to the second constructor.

Insert a new paragraph 1 Requires clause for the first constructor reading:

Requires: `s.rdbuf()` is not null.

---

114. Placement forms example in error twice

Section: 18.4.1.3 [lib.new.delete.placement] **Status:** DR **Submitter:** Steve Clamage **Date:** 28 Oct 1998

Section 18.4.1.3 contains the following example:

```
[Example: This can be useful for constructing an object at a known address:
        char place[sizeof(Something)];
        Something* p = new (place) Something();
 -end example]
```

First code line: "place" need not have any special alignment, and the following constructor could fail due to misaligned data.

Second code line: Aren't the parens on Something() incorrect? [Dublin: the LWG believes the () are correct.]

Examples are not normative, but nevertheless should not show code that is invalid or likely to fail.

Proposed resolution:

Replace the first line of code in the example in 18.4.1.3 with:

```
void* place = operator new(sizeof(Something));
```

---

115. Typo in strstream constructors

Section: D.7.4.1 [depr.strstream.cons] **Status:** DR **Submitter:** Steve Clamage **Date:** 2 Nov 1998

D.7.4.1 strstream constructors paragraph 2 says:

> Effects: Constructs an object of class strstream, initializing the base class with iostream(& sb) and initializing sb with one of the two constructors:
>
> - If mode&app==0, then s shall designate the first element of an array of n elements. The constructor is strstreambuf(s, n, s).
>
> - If mode&app==0, then s shall designate the first element of an array of n elements that contains an NTBS whose first element is designated by s. The constructor is strstreambuf(s, n, s+std::strlen(s)).

Notice the second condition is the same as the first. I think the second condition should be "If mode&app==app", or "mode&app!=0", meaning that the append bit is set.

Proposed resolution:

In D.7.3.1 paragraph 2 and D.7.4.1 paragraph 2, change the first condition to `(mode&app)==0` and the second condition to `(mode&app)!=0`.

---

118. `basic_istream` uses nonexistent `num_get` member functions

Section: 27.6.1.2.2 [lib.istream.formatted.arithmetic] **Status:** DR **Submitter:** Matt Austern **Date:** 20 Nov 1998

Formatted input is defined for the types `short`, `unsigned short`, `int`, `unsigned int`, `long`, `unsigned long`, `float`, `double`, `long double`, `bool`, and `void*`. According to section 27.6.1.2.2, formatted input of a value `x` is done as if by the following code fragment:

```
typedef num_get< charT,istreambuf_iterator<charT,traits> > numget;
iostate err = 0;
use_facet< numget >(loc).get(*this, 0, *this, err, val);
setstate(err);
```

According to section 22.2.2.1.1 , however, `num_get<>::get()` is only overloaded for the types `bool`, `long`, `unsigned short`, `unsigned int`, `unsigned long`, `unsigned long`, `float`, `double`, `long double`, and `void*`. Comparing the lists from the two sections, we find that 27.6.1.2.2 is using a nonexistent function for types `short` and `int`.

Proposed resolution:

In 27.6.1.2.2 Arithmetic Extractors, remove the two lines (1st and 3rd) which read:

```
operator>>(short& val);
...
operator>>(int& val);
```

And add the following at the end of that section (27.6.1.2.2) :

```
operator>>(short& val);
```

The conversion occurs as if performed by the following code fragment (using the same notation as for the preceding code fragment):

```
typedef num_get< charT,istreambuf_iterator<charT,traits> > numget;
iostate err = 0;
long lval;
use_facet< numget >(loc).get(*this, 0, *this, err, lval);
      if (err == 0
              && (lval < numeric_limits<short>::min() || numeric_limits<short>::max() < lval))
              err = ios_base::failbit;
setstate(err);
```

```
operator>>(int& val);
```

The conversion occurs as if performed by the following code fragment (using the same notation as for the preceding code fragment):

```
typedef num_get< charT,istreambuf_iterator<charT,traits> > numget;
iostate err = 0;
long lval;
use_facet< numget >(loc).get(*this, 0, *this, err, lval);
      if (err == 0
              && (lval < numeric_limits<int>::min() || numeric_limits<int>::max() < lval))
              err = ios_base::failbit;
setstate(err);
```

[Post-Tokyo: PJP provided the above wording.]

---

119. Should virtual functions be allowed to strengthen the exception specification?

Section: 17.4.4.8 [lib.res.on.exception.handling] **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 17.4.4.8 states:

"An implementation may strengthen the exception-specification for a function by removing listed exceptions."

The problem is that if an implementation is allowed to do this for virtual functions, then a library user cannot write a class that portably derives from that class.

For example, this would not compile if ios_base::failure::~failure had an empty exception specification:

```
#include <ios>
#include <string>

class D : public std::ios_base::failure {
public:
        D(const std::string&);
        ~D(); // error - exception specification must be compatible with
              // overridden virtual function ios_base::failure::~failure()
};
```

Proposed resolution:

Change Section 17.4.4.8 from:

"may strengthen the exception-specification for a function"

to:

"may strengthen the exception-specification for a non-virtual function".

---

122. streambuf/wstreambuf description should not say they are specializations

Section: 27.5.2 [lib.streambuf] **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

Section 27.5.2 describes the streambuf classes this way:

> The class streambuf is a specialization of the template class basic_streambuf specialized for the type char.
>
> The class wstreambuf is a specialization of the template class basic_streambuf specialized for the type wchar_t.

This implies that these classes must be template specializations, not typedefs.

It doesn't seem this was intended, since Section 27.5 has them declared as typedefs.

Proposed resolution:

Remove 27.5.2 paragraphs 2 and 3 (the above two sentences).

Rationale:

The `streambuf` synopsis already has a declaration for the typedefs and that is sufficient.

---

124. ctype_byname<charT>::do_scan_is & do_scan_not return type should be const charT*

Section: 22.2.1.2 [lib.locale.ctype.byname] **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

In Section 22.2.1.2 ctype_byname<charT>::do_scan_is() and do_scan_not() are declared to return a const char* not a const charT*.

Proposed resolution:

Change Section 22.2.1.2 `do_scan_is()` and `do_scan_not()` to return a `const charT*`.

---

125. valarray<T>::operator!() return type is inconsistent

Section: 26.3.2 [lib.template.valarray] **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

In Section 26.3.2 valarray<T>::operator!() is declared to return a valarray<T>, but in Section 26.3.2.5 it is declared to return a valarray<bool>. The latter appears to be correct.

Proposed resolution:

Change in Section 26.3.2 the declaration of `operator!()` so that the return type is `valarray<bool>`.

---

126. typos in Effects clause of ctype::do_narrow()

Section: 22.2.1.1.2 [lib.locale.ctype.virtuals] **Status:** DR **Submitter:** Judy Ward **Date:** 15 Dec 1998

Typos in 22.2.1.1.2 need to be fixed.

Proposed resolution:

In Section 22.2.1.1.2 change:

```
do_widen(do_narrow(c),0) == c
```

to:

```
do_widen(do_narrow(c,0)) == c
```

and change:

```
(is(M,c) || !ctc.is(M, do_narrow(c),dfault) )
```

to:

```
(is(M,c) || !ctc.is(M, do_narrow(c,dfault)) )
```

---

127. auto_ptr<> conversion issues

Section: 20.4.5 [lib.auto.ptr] **Status:** DR **Submitter:** Greg Colvin **Date:** 17 Feb 1999

There are two problems with the current `auto_ptr` wording in the standard:

First, the `auto_ptr_ref` definition cannot be nested because `auto_ptr<Derived>::auto_ptr_ref` is unrelated to `auto_ptr<Base>::auto_ptr_ref`. *Also submitted by Nathan Myers, with the same proposed resolution.*

Second, there is no `auto_ptr` assignment operator taking an `auto_ptr_ref` argument.

I have discussed these problems with my proposal coauthor, Bill Gibbons, and with some compiler and library implementors, and we believe that these problems are not desired or desirable implications of the standard.

25 Aug 1999: The proposed resolution now reflects changes suggested by Dave Abrahams, with Greg Colvin's concurrence; 1) changed "assignment operator" to "public assignment operator", 2) changed effects to specify use of release(), 3) made the conversion to auto_ptr_ref const.

2 Feb 2000: Lisa Lippincott comments: [The resolution of] this issue states that the conversion from auto_ptr to auto_ptr_ref should be const. This is not acceptable, because it would allow initialization and assignment from _any_ const auto_ptr! It also introduces an implementation difficulty in writing this conversion function -- namely, somewhere along the line, a const_cast will be necessary to remove that const so that release() may be called. This may result in undefined behavior [7.1.5.1/4]. The conversion operator does not have to be const, because a non-const implicit object parameter may be bound to an rvalue [13.3.3.1.4/3] [13.3.1/5].

Tokyo: The LWG removed the following from the proposed resolution:

In 20.4.5 , paragraph 2, and 20.4.5.3 , paragraph 2, make the conversion to auto_ptr_ref const:

```
template<class Y> operator auto_ptr_ref<Y>() const throw();
```

Proposed resolution:

In 20.4.5 , paragraph 2, move the `auto_ptr_ref` definition to namespace scope.

In 20.4.5 , paragraph 2, add a public assignment operator to the `auto_ptr` definition:

```
auto_ptr& operator=(auto_ptr_ref<X> r) throw();
```

Also add the assignment operator to 20.4.5.3 :

```
auto_ptr& operator=(auto_ptr_ref<X> r) throw()
```

Effects: Calls `reset(p.release())` for the `auto_ptr p` that `r` holds a reference to.
Returns: `*this`.

---

129. Need error indication from seekp() and seekg()

Section: 27.6.1.3 [lib.istream.unformatted], 27.6.2.4 [lib.ostream.seeks] **Status:** DR **Submitter:** Angelika Langer **Date:** 22 Feb 1999

Currently, the standard does not specify how seekg() and seekp() indicate failure. They are not required to set failbit, and they can't return an error indication because they must return *this, i.e. the stream. Hence, it is undefined what happens if they fail. And they *can* fail, for instance, when a file stream is disconnected from the underlying file (is_open()==false) or when a wide character file stream must perform a state-dependent code conversion, etc.

The stream functions seekg() and seekp() should set failbit in the stream state in case of failure.

Proposed resolution:

Add to the Effects: clause of seekg() in 27.6.1.3 and to the Effects: clause of seekp() in 27.6.2.4 :

In case of failure, the function calls `setstate(failbit)` (which may throw `ios_base::failure`).

Rationale:

Setting failbit is the usual error reporting mechanism for streams

---

132. list::resize description uses random access iterators

Section: 23.2.2.2 [lib.list.capacity] **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 1999

The description reads:

-1- Effects:

```
if (sz > size())
  insert(end(), sz-size(), c);
else if (sz < size())
  erase(begin()+sz, end());
else
  ;                           //  do nothing
```

Obviously list::resize should not be specified in terms of random access iterators.

Proposed resolution:

Change 23.2.2.2 paragraph 1 to:

Effects:

```
if (sz > size())
  insert(end(), sz-size(), c);
else if (sz < size())
{
  iterator i = begin();
  advance(i, sz);
  erase(i, end());
}
```

[Dublin: The LWG asked Howard to discuss exception safety offline with David Abrahams. They had a discussion and believe there is no issue of exception safety with the proposed resolution.]

---

133. map missing get_allocator()

Section: 23.3.1 [lib.map] **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 1999

The title says it all.

Proposed resolution:

Insert in 23.3.1 , paragraph 2, after operator= in the map declaration:

```
allocator_type get_allocator() const;
```

---

134. vector constructors over specified

Section: 23.2.4.1 [lib.vector.cons] **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 1999

The complexity description says: "It does at most 2N calls to the copy constructor of T and logN reallocations if they are just input iterators ...".

This appears to be overly restrictive, dictating the precise memory/performance tradeoff for the implementor.

Proposed resolution:

Change 23.2.4.1 , paragraph 1 to:

-1- Complexity: The constructor template <class InputIterator> vector(InputIterator first, InputIterator last) makes only N calls to the copy constructor of T (where N is the distance between first and last) and no reallocations if iterators first and last are of forward, bidirectional, or random access categories. It makes order N calls to the copy constructor of T and order logN reallocations if they are just input iterators.

Rationale:

"at most 2N calls" is correct only if the growth factor is greater than or equal to 2.

---

136. seekp, seekg setting wrong streams?

Section: 27.6.1.3 [lib.istream.unformatted] **Status:** DR **Submitter:** Howard Hinnant **Date:** 6 Mar 1999

I may be misunderstanding the intent, but should not seekg set only the input stream and seekp set only the output stream? The description seems to say that each should set both input and output streams. If that's really the intent, I withdraw this proposal.

Proposed resolution:

In section 27.6.1.3 change:

```
basic_istream<charT,traits>& seekg(pos_type pos);
Effects: If fail() != true, executes rdbuf()->pubseekpos(pos).
```

To:

```
basic_istream<charT,traits>& seekg(pos_type pos);
Effects: If fail() != true, executes rdbuf()->pubseekpos(pos, ios_base::in).
```

In section 27.6.1.3 change:

```
basic_istream<charT,traits>& seekg(off_type& off, ios_base::seekdir dir);
Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir).
```

To:

```
basic_istream<charT,traits>& seekg(off_type& off, ios_base::seekdir dir);
Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir, ios_base::in).
```

In section 27.6.2.4, paragraph 2 change:

```
-2- Effects: If fail() != true, executes rdbuf()->pubseekpos(pos).
```

To:

```
-2- Effects: If fail() != true, executes rdbuf()->pubseekpos(pos, ios_base::out).
```

In section 27.6.2.4, paragraph 4 change:

```
-4- Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir).
```

To:

```
-4- Effects: If fail() != true, executes rdbuf()->pubseekoff(off, dir, ios_base::out).
```

[Dublin: Dietmar Kühl thinks this is probably correct, but would like the opinion of more iostream experts before taking action.]

[Tokyo: Reviewed by the LWG. PJP noted that although his docs are incorrect, his implementation already implements the Proposed Resolution.]

[Post-Tokyo: Matt Austern comments:
Is it a problem with basic_istream and basic_ostream, or is it a problem with basic_stringbuf? We could resolve the issue either by changing basic_istream and basic_ostream, or by changing basic_stringbuf. I prefer the latter change (or maybe both changes): I don't see any reason for the standard to require that std::stringbuf s(std::string("foo"), std::ios_base::in); s.pubseekoff(0, std::ios_base::beg); must fail.
This requirement is a bit weird. There's no similar requirement for basic_streambuf<>::seekpos, or for basic_filebuf<>::seekoff or basic_filebuf<>::seekpos.]

---

137. Do use_facet and has_facet look in the global locale?

Section: 22.1.1 [lib.locale] **Status:** DR **Submitter:** Angelika Langer **Date:** 17 Mar 1999

Section 22.1.1 says:

-4- In the call to use_facet<Facet>(loc), the type argument chooses a facet, making available all members of the named type. If Facet is not present in a locale (or, failing that, in the global locale), it throws the standard exception bad_cast. A C++ program can check if a locale implements a particular facet with the template function has_facet<Facet>().

This contradicts the specification given in section 22.1.2 :
template <class Facet> const Facet& use_facet(const locale& loc);
-1- Get a reference to a facet of a locale.
-2- Returns: a reference to the corresponding facet of loc, if present.
-3- Throws: bad_cast if has_facet<Facet>(loc) is false.
-4- Notes: The reference returned remains valid at least as long as any copy of loc exists

Proposed resolution:

Remove the phrase "(or, failing that, in the global locale)" from section 22.1.1.

Rationale:

Needed for consistency with the way locales are handled elsewhere in the standard.

---

139. Optional sequence operation table description unclear

Section: 23.1.1 [lib.sequence.reqmts] **Status:** DR **Submitter:** Andrew Koenig **Date:** 30 Mar 1999

The sentence introducing the Optional sequence operation table (23.1.1 paragraph 12) has two problems:

A. It says ‘‘The operations in table 68 are provided only for the containers for which they take constant time.’’ That could be interpreted in two ways, one of them being ‘‘Even though table 68 shows particular operations as being provided, implementations are free to omit them if they cannot implement them in constant time.’’
B. That paragraph says nothing about amortized constant time, and it should.

Proposed resolution:

Replace the wording in 23.1.1 paragraph 12 which begins ‘‘The operations in table 68 are provided only..." with:

> Table 68 lists sequence operations that are provided for some types of sequential containers but not others. An implementation shall provide these operations for all container types shown in the ‘‘container’’ column, and shall implement them so as to take amortized constant time.

---

141. basic_string::find_last_of, find_last_not_of say pos instead of xpos

Section: 21.3.6.4 [lib.string::find.last.of], 21.3.6.6 [lib.string::find.last.not.of] **Status:** DR **Submitter:** Arch Robison **Date:** 28 Apr 1999

Sections 21.3.6.4 paragraph 1 and 21.3.6.6 paragraph 1 surely have misprints where they say:
‹ `xpos <= pos` and `pos < size();`

Surely the document meant to say ‘‘`xpos < size()`’’ in both places.

[Judy Ward also sent in this issue for 21.3.6.4 with the same proposed resolution.]

Proposed resolution:

Change Sections 21.3.6.4 paragraph 1 and 21.3.6.6 paragraph 1, the line which says:
‹ `xpos <= pos` and `pos < size();`
to:
‹ `xpos <= pos` and `xpos < size();`

---

142. lexicographical_compare complexity wrong

Section: 25.3.8 [lib.alg.lex.comparison] **Status:** DR **Submitter:** Howard Hinnant **Date:** 20 Jun 1999

The lexicographical_compare complexity is specified as:
"At most min((last1 - first1), (last2 - first2)) applications of the corresponding comparison."
The best I can do is twice that expensive.

Nicolai Josuttis comments in lib-6862: You mean, to check for equality you have to check both < and >? Yes, IMO you are right! (and Matt states this complexity in his book)

Proposed resolution:

Change 25.3.8 complexity to:

At most `2*min((last1 - first1), (last2 - first2))` applications of the corresponding comparison.

Change the example at the end of paragraph 3 to read:

[Example:

```
for ( ; first1 != last1 && first2 != last2 ; ++first1, ++first2) {
  if (*first1 < *first2) return true;
  if (*first2 < *first1) return false;
}
return first1 == last1 && first2 != last2;
```

--end example]

---

144. Deque constructor complexity wrong

Section: 23.2.1.1 [lib.deque.cons] **Status:** DR **Submitter:** Herb Sutter **Date:** 9 May 1999

In 23.2.1.1 paragraph 6, the deque ctor that takes an iterator range appears to have complexity requirements which are incorrect, and which contradict the complexity requirements for insert(). I suspect that the text in question, below, was taken from vector:

> Complexity: If the iterators first and last are forward iterators, bidirectional iterators, or random access iterators the constructor makes only N calls to the copy constructor, and performs no reallocations, where N is last - first.

The word "reallocations" does not really apply to deque. Further, all of the following appears to be spurious:

> It makes at most 2N calls to the copy constructor of T and log N reallocations if they are input iterators.1)
>
> 1) The complexity is greater in the case of input iterators because each element must be added individually: it is impossible to determine the distance between first abd last before doing the copying.

This makes perfect sense for vector, but not for deque. Why should deque gain an efficiency advantage from knowing in advance the number of elements to insert?

Proposed resolution:

In 23.2.1.1 paragraph 6, replace the Complexity description, including the footnote, with the following text (which also corrects the "abd" typo):

> Complexity: Makes last - first calls to the copy constructor of T.

---

146. complex<T> Inserter and Extractor need sentries

Section: 26.2.6 [lib.complex.ops] **Status:** DR **Submitter:** Angelika Langer **Date:** 12 May 1999

The extractor for complex numbers is specified as:

```
template<class T, class charT, class traits>
basic_istream<charT, traits>&
operator>>(basic_istream<charT, traits>& is, complex<T>& x);
```

> Effects: Extracts a complex number x of the form: u, (u), or (u,v), where u is the real part and v is the imaginary part (lib.istream.formatted).

> Requires: The input values be convertible to T. If bad input is encountered, calls is.setstate(ios::failbit) (which may throw ios::failure (lib.iostate.flags).
> Returns: is .

Is it intended that the extractor for complex numbers does not skip whitespace, unlike all other extractors in the standard library do? Shouldn't a sentry be used?
The inserter for complex numbers is specified as:

```
template<class T, class charT, class traits>
basic_ostream<charT, traits>&
operator<<(basic_ostream<charT, traits>& o, const complex<T>& x);
Effects: inserts the complex number x onto the stream o as if it were implemented as follows:
template<class T, class charT, class traits>
basic_ostream<charT, traits>&
operator<<(basic_ostream<charT, traits>& o, const complex<T>& x)
{
basic_ostringstream<charT, traits> s;
s.flags(o.flags());
s.imbue(o.getloc());
s.precision(o.precision());
s << '(' << x.real() << "," << x.imag() << ')';
return o << s.str();
}
```

Is it intended that the inserter for complex numbers ignores the field width and does not do any padding? If, with the suggested implementation above, the field width were set in the stream then the opening parentheses would be adjusted, but the rest not, because the field width is reset to zero after each insertion.

I think that both operations should use sentries, for sake of consistency with the other inserters and extractors in the library. Regarding the issue of padding in the inserter, I don't know what the intent was.

Proposed resolution:

After 26.2.6 paragraph 14 (operator>>), add a Notes clause:

> Notes: This extraction is performed as a series of simpler extractions. Therefore, the skipping of whitespace is specified to be the same for each of the simpler extractions.

Rationale:

For extractors, the note is added to make it clear that skipping whitespace follows an "all-or-none" rule.

For inserters, the LWG believes there is no defect; the standard is correct as written.

---

147. Library Intro refers to global functions that aren't global

Section: 17.4.4.3 [lib.global.functions] **Status:** DR **Submitter:** Lois Goldthwaite **Date:** 4 Jun 1999

The library had many global functions until 17.4.1.1 [lib.contents] paragraph 2 was added:

> All library entities except macros, operator new and operator delete are defined within the namespace std or namespaces nested within namespace std.

It appears "global function" was never updated in the following:

17.4.4.3 - Global functions [lib.global.functions]
-1- It is unspecified whether any global functions in the C++ Standard Library are defined as inline (dcl.fct.spec).
-2- A call to a global function signature described in Clauses lib.language.support through lib.input.output behaves the same as if the implementation declares no additional global function signatures.*
[Footnote: A valid C++ program always calls the expected library global function. An implementation may also define additional global functions that would otherwise not be called by a valid C++ program. --- end footnote]
-3- A global function cannot be declared by the implementation as taking additional default arguments.
17.4.4.4 - Member functions [lib.member.functions]
-2- An implementation can declare additional non-virtual member function signatures within a class:

-- by adding arguments with default values to a member function signature; The same latitude does not extend to the implementation of virtual or global functions, however.

Proposed resolution:

Change "global" to "global or non-member" in:

17.4.4.3 [lib.global.functions] section title,
17.4.4.3 [lib.global.functions] para 1,
17.4.4.3 [lib.global.functions] para 2 in 2 places plus 2 places in the footnote,
17.4.4.3 [lib.global.functions] para 3,
17.4.4.4 [lib.member.functions] para 2

Rationale:

Because operator new and delete are global, the proposed resolution was changed from "non-member" to "global or non-member.

---

148. Functions in the example facet BoolNames should be const

Section: 22.2.8 [lib.facets.examples] **Status:** DR **Submitter:** Jeremy Siek **Date:** 3 Jun 1999

In 22.2.8 paragraph 13, the do_truename() and do_falsename() functions in the example facet BoolNames should be const. The functions they are overriding in numpunct_byname<char> are const.

Proposed resolution:

In 22.2.8 paragraph 13, insert "const" in two places:

```
string do_truename() const { return "Oui Oui!"; }
string do_falsename() const { return "Mais Non!"; }
```

---

150. Find_first_of says integer instead of iterator

Section: 25.1.4 [lib.alg.find.first.of] **Status:** DR **Submitter:** Matt McClure **Date:** 30 Jun 1999

Proposed resolution:

Change 25.1.4 paragraph 2 from:

Returns: The first iterator i in the range [first1, last1) such that for some integer j in the range [first2, last2) ...

to:

Returns: The first iterator i in the range [first1, last1) such that for some iterator j in the range [first2, last2) ...

---

151. Can't currently clear() empty container

Section: 23.1.1 [lib.sequence.reqmts] **Status:** DR **Submitter:** Ed Brey **Date:** 21 Jun 1999

For both sequences and associative containers, a.clear() has the semantics of erase(a.begin(),a.end()), which is undefined for an empty container since erase(q1,q2) requires that q1 be dereferenceable (23.1.1,3 and 23.1.2,7). When the container is empty, a.begin() is not dereferenceable.
The requirement that q1 be unconditionally dereferenceable causes many operations to be intuitively undefined, of which clearing an empty container is probably the most dire.
Since q1 and q2 are only referenced in the range [q1, q2), and [q1, q2) is required to be a valid range, stating that q1 and q2 must be iterators or certain kinds of iterators is unnecessary.

Proposed resolution:

In 23.1.1, paragraph 3, change:

p and q2 denote valid iterators to a, q and q1 denote valid dereferenceable iterators to a, [q1, q2) denotes a valid range

to:

p denotes a valid iterator to a, q denotes a valid dereferenceable iterator to a, [q1, q2) denotes a valid range in a

In 23.1.2, paragraph 7, change:

p and q2 are valid iterators to a, q and q1 are valid dereferenceable iterators to a, [q1, q2) is a valid range

to

p is a valid iterator to a, q is a valid dereferenceable iterator to a, [q1, q2) is a valid range into a

---

152. Typo in `scan_is()` semantics

Section: 22.2.1.1.2 [lib.locale.ctype.virtuals] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The semantics of `scan_is()` (paragraphs 4 and 6) is not exactly described because there is no function `is()` which only takes a character as argument. Also, in the effects clause (paragraph 3), the semantic is also kept vague.

Proposed resolution:

In 22.2.1.1.2 paragraphs 4 and 6, change the returns clause from:

"... such that `is(*p)` would..."

to: "... such that `is(m, *p)` would...."

---

153. Typo in `narrow()` semantics

Section: 22.2.1.3.2 [lib.facet.ctype.char.members] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The description of the array version of `narrow()` (in paragraph 11) is flawed: There is no member `do_narrow()` which takes only three arguments because in addition to the range a default character is needed.

Additionally, for both `widen` and `narrow` we have two signatures followed by a **Returns** clause that only addresses one of them.

Proposed resolution:

Change the returns clause in 22.2.1.3.2 paragraph 10 from:

> Returns: do_widen(low, high, to).

to:

> Returns: do_widen(c) or do_widen(low, high, to), respectively.

Change 22.2.1.3.2 paragraph 10 and 11 from:

```
char        narrow(char c, char /*dfault*/) const;
const char* narrow(const char* low, const char* high,
                   char /*dfault*/, char* to) const;

Returns: do_narrow(low, high, to).
```

to:

```
char        narrow(char c, char dfault) const;
const char* narrow(const char* low, const char* high,
                   char dfault, char* to) const;

Returns: do_narrow(c, dfault) or
         do_narrow(low, high, dfault, to), respectively.
```

[Kona: 1) the problem occurs in additional places, 2) a user defined version could be different.]

[Post-Tokyo: Dietmar provided the above wording at the request of the LWG. He could find no other places the problem occurred. He asks for clarification of the Kona "a user defined version..." comment above. Perhaps it was a circuitous way of saying "dfault" needed to be uncommented?]

[Post-Toronto: the issues list maintainer has merged in the proposed resolution from issue 207, which addresses the same paragraphs.]

---

154. Missing `double` specifier for `do_get()`

Section: 22.2.2.1.2 [lib.facet.num.get.virtuals] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The table in paragraph 7 for the length modifier does not list the length modifier `l` to be applied if the type is `double`. Thus, the standard asks the implementation to do undefined things when using `scanf()` (the missing length modifier for `scanf()` when scanning `double`s is actually a problem I found quite often in production code, too).

Proposed resolution:

In 22.2.2.1.2 , paragraph 7, add a row in the Length Modifier table to say that for `double` a length modifier `l` is to be used.

Rationale:

The standard makes an embarrassing beginner's mistake.

---

155. Typo in naming the class defining the class `Init`

Section: 27.3 [lib.iostream.objects] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

There are conflicting statements about where the class `Init` is defined. According to 27.3 paragraph 2 it is defined as `basic_ios::Init`, according to 27.4.2 it is defined as `ios_base::Init`.

Proposed resolution:

Change 27.3 paragraph 2 from "`basic_ios::Init`" to "`ios_base::Init`".

Rationale:

Although not strictly wrong, the standard was misleading enough to warrant the change.

---

156. Typo in `imbue()` description

Section: 27.4.2.3 [lib.ios.base.locales] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

There is a small discrepancy between the declarations of `imbue()`: in 27.4.2 the argument is passed as `locale const&` (correct), in 27.4.2.3 it is passed as `locale const` (wrong).

Proposed resolution:

In 27.4.2.3 change the `imbue` argument from "`locale const" to "locale const&".`

---

158. Underspecified semantics for `setbuf()`

Section: 27.5.2.4.2 [lib.streambuf.virt.buffer] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The default behavior of `setbuf()` is described only for the situation that `gptr() != 0 && gptr() != egptr()`: namely to do nothing. What has to be done in other situations is not described although there is actually only one reasonable approach, namely to do nothing, too.

Since changing the buffer would almost certainly mess up most buffer management of derived classes unless these classes do it themselves, the default behavior of `setbuf()` should always be to do nothing.

Proposed resolution:

Change 27.5.2.4.2 , paragraph 3, Default behavior, to: "Default behavior: Does nothing. Returns this."

---

159. Strange use of `underflow()`

Section: 27.5.2.4.3 [lib.streambuf.virt.get] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The description of the meaning of the result of `showmanyc()` seems to be rather strange: It uses calls to `underflow()`. Using `underflow()` is strange because this function only reads the current character but does not extract it, `uflow()` would extract the current character. This should be fixed to use `sbumpc()` instead.

Proposed resolution:

Change 27.5.2.4.3 paragraph 1, `showmanyc()`returns clause, by replacing the word "supplied" with the words "extracted from the stream".

---

160. Typo: Use of non-existing function `exception()`

Section: 27.6.1.1 [lib.istream] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The paragraph 4 refers to the function `exception()` which is not defined. Probably, the referred function is `basic_ios<>::exceptions()`.

Proposed resolution:

In 27.6.1.1 , 27.6.1.3 , paragraph 1, 27.6.2.1 , paragraph 3, and 27.6.2.5.1 , paragraph 1, change "`exception()`" to "`exceptions()`".

[Note to Editor: "exceptions" with an "s" is the correct spelling.]

---

161. Typo: `istream_iterator` vs. `istreambuf_iterator`

Section: 27.6.1.2.2 [lib.istream.formatted.arithmetic] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The note in the second paragraph pretends that the first argument is an object of type `istream_iterator`. This is wrong: It is an object of type `istreambuf_iterator`.

Proposed resolution:

Change 27.6.1.2.2 from:

> The first argument provides an object of the istream_iterator class...

to

> The first argument provides an object of the istreambuf_iterator class...

---

164. do_put() has apparently unused fill argument

Section: 22.2.5.3.2 [lib.locale.time.put.virtuals] **Status:** DR **Submitter:** Angelika Langer **Date:** 23 Jul 1999

In 22.2.5.3.2 the do_put() function is specified as taking a fill character as an argument, but the description of the function does not say whether the character is used at all and, if so, in which way. The same holds for any format control parameters that are accessible through the ios_base& argument, such as the adjustment or the field width. Is strftime() supposed to use the fill character in any way? In any case, the specification of time_put.do_put() looks inconsistent to me.
Is the signature of do_put() wrong, or is the effects clause incomplete?

Proposed resolution:

Add the following note after 22.2.5.3.2 paragraph 2:

> [Note: the `fill` argument may be used in the implementation-defined formats, or by derivations. A space character is a reasonable default for this argument. --end Note]

Rationale:

The LWG felt that while the normative text was correct, users need some guidance on what to pass for the `fill` argument since the standard doesn't say how it's used.

---

165. `xsputn()`, `pubsync()` never called by `basic_ostream` members?

Section: 27.6.2.1 [lib.ostream] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

Paragraph 2 explicitly states that none of the `basic_ostream` functions falling into one of the groups "formatted output functions" and "unformatted output functions" calls any stream buffer function which might call a virtual function other than `overflow()`. Basically this is fine but this implies that `sputn()` (this function would call the virtual function `xsputn()`) is never called by any of the standard output functions. Is this really intended? At minimum it would be convenient to call `xsputn()` for strings... Also, the statement that `overflow()` is the only virtual member of `basic_streambuf` called is in conflict with the definition of `flush()` which calls `rdbuf()->pubsync()` and thereby the virtual function `sync()` (`flush()` is listed under "unformatted output functions").

In addition, I guess that the sentence starting with "They may use other public members of `basic_ostream` ..." probably was intended to start with "They may use other public members of `basic_streamuf`..." although the problem with the virtual members exists in both cases.

I see two obvious resolutions:

1. state in a footnote that this means that `xsputn()` will never be called by any ostream member and that this is intended.
2. relax the restriction and allow calling `overflow()` and `xsputn()`. Of course, the problem with `flush()` has to be resolved in some way.

Proposed resolution:

Change the last sentence of 27.6.2.1 (lib.ostream) paragraph 2 from:

> They may use other public members of basic_ostream except that they do not invoke any virtual members of rdbuf() except overflow().

to:

> They may use other public members of basic_ostream except that they shall not invoke any virtual members of rdbuf() except overflow(), xsputn(), and sync().

[Kona: the LWG believes this is a problem. Wish to ask Jerry or PJP why the standard is written this way.]

[Post-Tokyo: Dietmar supplied wording at the request of the LWG. He comments: The rules can be made a little bit more specific if necessary be explicitly spelling out what virtuals are allowed to be called from what functions and eg to state specifically that flush() is allowed to call sync() while other functions are not.]

168. Typo: formatted vs. unformatted

Section: 27.6.2.6 [lib.ostream.unformatted] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The first paragraph begins with a descriptions what has to be done in *formatted* output functions. Probably this is a typo and the paragraph really want to describe unformatted output functions...

Proposed resolution:

In 27.6.2.6 paragraph 1, the first and last sentences, change the word "formatted" to "unformatted":

> "Each **unformatted** output function begins ..."
> "... value specified for the **unformatted** output function."

---

169. Bad efficiency of `overflow()` mandated

Section: 27.7.1.3 [lib.stringbuf.virtuals] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

Paragraph 8, Notes, of this section seems to mandate an extremely inefficient way of buffer handling for `basic_stringbuf`, especially in view of the restriction that `basic_ostream` member functions are not allowed to use `xsputn()` (see 27.6.2.1): For each character to be inserted, a new buffer is to be created.

Of course, the resolution below requires some handling of simultaneous input and output since it is no longer possible to update `egptr()` whenever `epptr()` is changed. A possible solution is to handle this in `underflow()`.

Proposed resolution:

In 27.7.1.3 paragraph 8, Notes, insert the words "at least" as in the following:

> To make a write position available, the function reallocates (or initially allocates) an array object with a sufficient number of elements to hold the current array object (if any), plus **at least** one additional write position.

---

170. Inconsistent definition of `traits_type`

Section: 27.7.4 [lib.stringstream] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

The classes `basic_stringstream` (27.7.4), `basic_istringstream` (27.7.2), and `basic_ostringstream` (27.7.3) are inconsistent in their definition of the type `traits_type`: For `istringstream`, this type is defined, for the other two it is not. This should be consistent.

Proposed resolution:

Proposed resolution:

To the declarations of `basic_ostringstream` (27.7.3) and `basic_stringstream` (27.7.4) add:

```
typedef traits traits_type;
```

---

171. Strange `seekpos()` semantics due to joint position

Section: 27.8.1.4 [lib.filebuf.virtuals] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 20 Jul 1999

Overridden virtual functions, seekpos()

In 27.8.1.1 paragraph 3, it is stated that a joint input and output position is maintained by `basic_filebuf`. Still, the description of `seekpos()` seems to talk about different file positions. In particular, it is unclear (at least to me) what is supposed to happen to the output buffer (if there is one) if only the input position is changed. The standard seems to mandate that the output buffer is kept and processed as if there was no positioning of the output position (by changing the input position). Of course, this can be exactly what you want if the flag `ios_base::ate` is set. However, I think, the standard should say something like this:

- If `(which & mode) == 0` neither read nor write position is changed and the call fails. Otherwise, the joint read and write position is altered to correspond to `sp`.
- If there is an output buffer, the output sequences is updated and any unshift sequence is written before the position is altered.
- If there is an input buffer, the input sequence is updated after the position is altered.

Plus the appropriate error handling, that is...

Proposed resolution:

Change the unnumbered paragraph in 27.8.1.4 (lib.filebuf.virtuals) before paragraph 14 from:

> pos_type seekpos(pos_type sp, ios_base::openmode = ios_base::in | ios_base::out);
>
> Alters the file position, if possible, to correspond to the position stored in sp (as described below).
>
> - if (which&ios_base::in)!=0, set the file position to sp, then update the input sequence
>
> - if (which&ios_base::out)!=0, then update the output sequence, write any unshift sequence, and set the file position to sp.

to:

> pos_type seekpos(pos_type sp, ios_base::openmode = ios_base::in | ios_base::out);
>
> Alters the file position, if possible, to correspond to the position stored in sp (as described below). Altering the file position performs as follows:
>
> 1. if (om & ios_base::out)!=0, then update the output sequence and write any unshift sequence;
>
> 2. set the file position to sp;
>
> 3. if (om & ios_base::in)!=0, then update the input sequence;
>
> where om is the open mode passed to the last call to open(). The operation fails if is_open() returns false.

[Kona: Dietmar is working on a proposed resolution.]

[Post-Tokyo: Dietmar supplied the above wording.]

---

172. Inconsistent types for `basic_istream::ignore()`

Section: 27.6.1.3 [lib.istream.unformatted] **Status:** DR **Submitter:** Greg Comeau, Dietmar Kühl **Date:** 23 Jul 1999

In 27.6.1.1 the function `ignore()` gets an object of type `streamsize` as first argument. However, in 27.6.1.3 paragraph 23 the first argument is of type `int`.

As far as I can see this is not really a contradiction because everything is consistent if `streamsize` is typedef to be `int`. However, this is almost certainly not what was intended. The same thing happened to `basic_filebuf::setbuf()`, as described in issue 173.

Darin Adler also submitted this issue, commenting: Either 27.6.1.1 should be modified to show a first parameter of type int, or 27.6.1.3 should be modified to show a first parameter of type streamsize and use numeric_limits<streamsize>::max.

Proposed resolution:

In 27.6.1.3 paragraph 23 and 24, change both uses of `int` in the description of `ignore()` to `streamsize`.

---

173. Inconsistent types for `basic_filebuf::setbuf()`

Section: 27.8.1.4 [lib.filebuf.virtuals] **Status:** DR **Submitter:** Greg Comeau, Dietmar Kühl **Date:** 23 Jul 1999

In 27.8.1.1 the function `setbuf()` gets an object of type `streamsize` as second argument. However, in 27.8.1.4 paragraph 9 the second argument is of type `int`.

As far as I can see this is not really a contradiction because everything is consistent if `streamsize` is typedef to be `int`. However, this is almost certainly not what was intended. The same thing happened to `basic_istream::ignore()`, as described in issue 172.

Proposed resolution:

In 27.8.1.4 paragraph 9, change all uses of `int` in the description of `setbuf()` to `streamsize`.

---

174. Typo: `OFF_T` vs. `POS_T`

Section: D.6 [depr.ios.members] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 23 Jul 1999

According to paragraph 1 of this section, `streampos` is the type `OFF_T`, the same type as `streamoff`. However, in paragraph 6 the `streampos` gets the type `POS_T`

Proposed resolution:

Change D.6 paragraph 1 from "`typedef OFF_T streampos;`" to "`typedef POS_T streampos;`"

---

175. Ambiguity for `basic_streambuf::pubseekpos()` and a few other functions.

Section: D.6 [depr.ios.members] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 23 Jul 1999

According to paragraph 8 of this section, the methods `basic_streambuf::pubseekpos()`, `basic_ifstream::open()`, and `basic_ofstream::open` "may" be overloaded by a version of this function taking the type `ios_base::open_mode` as last argument argument instead of `ios_base::openmode` (`ios_base::open_mode` is defined in this section to be an alias for one of the integral types). The clause specifies, that the last argument has a default argument in three cases. However, this generates an ambiguity with the overloaded version because now the arguments are absolutely identical if the last argument is not specified.

Proposed resolution:

In D.6 paragraph 8, remove the default arguments for `basic_streambuf::pubseekpos()`, `basic_ifstream::open()`, and `basic_ofstream::open()`.

---

176. `exceptions()` in `ios_base`...?

Section: D.6 [depr.ios.members] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 23 Jul 1999

The "overload" for the function `exceptions()` in paragraph 8 gives the impression that there is another function of this function defined in class `ios_base`. However, this is not the case. Thus, it is hard to tell how the semantics (paragraph 9) can be implemented: "Call the corresponding member function specified in clause 27 ."

Proposed resolution:

In D.6 paragraph 8, move the declaration of the function `exceptions()`into class `basic_ios`.

---

181. make_pair() unintended behavior

Section: 20.2.2 [lib.pairs] **Status:** DR **Submitter:** Andrew Koenig **Date:** 3 Aug 1999

The claim has surfaced in Usenet that expressions such as

```
make_pair("abc", 3)
```

are illegal, notwithstanding their use in examples, because template instantiation tries to bind the first template parameter to `const char (&)[4]`, which type is uncopyable.
I doubt anyone intended that behavior...

Proposed resolution:

In 20.2 , paragraph 1 change the following declaration of make_pair():

```
template <class T1, class T2> pair<T1,T2> make_pair(const T1&, const T2&);
```

to:

```
template <class T1, class T2> pair<T1,T2> make_pair(T1, T2);
```

In 20.2.2 paragraph 7 and the line before, change:

```
template <class T1, class T2>
pair<T1, T2> make_pair(const T1& x, const T2& y);
```

to:

```
template <class T1, class T2>
pair<T1, T2> make_pair(T1 x, T2 y);
```

and add the following footnote to the effects clause:

> According to 12.8 [class.copy], an implementation is permitted to not perform a copy of an argument, thus avoiding unnecessary copies.

Rationale:

Two potential fixes were suggested by Matt Austern and Dietmar Kühl, respectively, 1) overloading with array arguments, and 2) use of a reference_traits class with a specialization for arrays. Andy Koenig suggested changing to pass by value. In discussion, it appeared that this was a much smaller change to the standard that the other two suggestions, and any efficiency concerns were more than offset by the advantages of the solution. Two implementors reported that the proposed resolution passed their test suites.

---

183. I/O stream manipulators don’t work for wide character streams

Section: 27.6.3 [lib.std.manip] **Status:** DR **Submitter:** Andy Sawyer **Date:** 7 Jul 1999

27.6.3 paragraph 3 says (clause numbering added for exposition): [lib.std.manip]

> Returns: An object s of unspecified type such that if [1] out is an (instance of) basic_ostream then the expression out<<s behaves as if f(s) were called, and if [2] in is an (instance of) basic_istream then the expression in>>s behaves as if f(s) were called. Where f can be defined as: ios_base& f(ios_base& str, ios_base::fmtflags mask) { // reset specified flags str.setf(ios_base::fmtflags(0), mask); return str; } [3] The expression out<<s has type ostream& and value out. [4] The expression in>>s has type istream& and value in.

Given the definitions [1] and [2] for out and in, surely [3] should read: "The expression out << s has type basic_ostream& ..." and [4] should read: "The expression in >> s has type basic_istream& ..."

If the wording in the standard is correct, I can see no way of implementing any of the manipulators so that they will work with wide character streams.

e.g. wcout << setbase(16);

Must have value ’wcout’ (which makes sense) and type ’ostream&’ (which doesn’t).

The same "cut’n’paste" type also seems to occur in Paras 4,5,7 and 8. In addition, Para 6 [setfill] has a similar error, but relates only to ostreams.

I’d be happier if there was a better way of saying this, to make it clear that the value of the expression is "the same specialization of basic_ostream as out"&

Proposed resolution:

Replace section 27.6.3 except paragraph 1 with the following:

> 2- The type designated smanip in each of the following function descriptions is implementation-specified and may be different for each function.
> `smanip resetiosflags(ios_base::fmtflags mask);`
> -3- Returns: An object s of unspecified type such that if out is an instance of basic_ostream<charT,traits> then the expression out<<s behaves as if f(s, mask) were called, or if in is an instance of basic_istream<charT,traits> then the expression in>>s behaves as if f(s, mask) were called. The function f can be defined as:*
> [Footnote: The expression cin >> resetiosflags(ios_base::skipws) clears ios_base::skipws in the format flags stored in the basic_istream<charT,traits> object cin (the same as cin >> noskipws), and the expression cout << resetiosflags(ios_base::showbase) clears ios_base::showbase in the format flags stored in the basic_ostream<charT,traits> object cout (the same as cout << noshowbase). --- end footnote]
>
> ```
> ios_base& f(ios_base& str, ios_base::fmtflags mask)
>  {
> ```

```
   // reset specified flags
   str.setf(ios_base::fmtflags(0), mask);
   return str;
   }
```

The expression out<<s has type basic_ostream<charT,traits>& and value out. The expression in>>s has type basic_istream<charT,traits>& and value in.

```
smanip setiosflags(ios_base::fmtflags mask);
```

-4- Returns: An object s of unspecified type such that if out is an instance of basic_ostream<charT,traits> then the expression out<<s behaves as if f(s, mask) were called, or if in is an instance of basic_istream<charT,traits> then the expression in>>s behaves as if f(s, mask) were called. The function f can be defined as:

```
  ios_base& f(ios_base& str, ios_base::fmtflags mask)
   {
   // set specified flags
   str.setf(mask);
   return str;
   }
```

The expression out<<s has type basic_ostream<charT,traits>& and value out. The expression in>>s has type basic_istream<charT,traits>& and value in.

```
smanip setbase(int base);
```

-5- Returns: An object s of unspecified type such that if out is an instance of basic_ostream<charT,traits> then the expression out<<s behaves as if f(s, base) were called, or if in is an instance of basic_istream<charT,traits> then the expression in>>s behaves as if f(s, base) were called. The function f can be defined as:

```
  ios_base& f(ios_base& str, int base)
   {
   // set basefield
   str.setf(base == 8 ? ios_base::oct :
   base == 10 ? ios_base::dec :
   base == 16 ? ios_base::hex :
   ios_base::fmtflags(0), ios_base::basefield);
   return str;
   }
```

The expression out<<s has type basic_ostream<charT,traits>& and value out. The expression in>>s has type basic_istream<charT,traits>& and value in.

```
smanip setfill(char_type c);
```

-6- Returns: An object s of unspecified type such that if out is (or is derived from) basic_ostream<charT,traits> and c has type charT then the expression out<<s behaves as if f(s, c) were called. The function f can be defined as:

```
   template<class charT, class traits>
   basic_ios<charT,traits>& f(basic_ios<charT,traits>& str, charT c)
   {
   // set fill character
   str.fill(c);
   return str;
   }
```

The expression out<<s has type basic_ostream<charT,traits>& and value out.

```
smanip setprecision(int n);
```

-7- Returns: An object s of unspecified type such that if out is an instance of basic_ostream<charT,traits> then the expression out<<s behaves as if f(s, n) were called, or if in is an instance of basic_istream<charT,traits> then the expression in>>s behaves as if f(s, n) were called. The function f can be defined as:

```
   ios_base& f(ios_base& str, int n)
   {
   // set precision
   str.precision(n);
   return str;
   }
```

The expression out<<s has type basic_ostream<charT,traits>& and value out. The expression in>>s has type basic_istream<charT,traits>& and value in

.

```
smanip setw(int n);
```

-8- Returns: An object s of unspecified type such that if out is an instance of basic_ostream<charT,traits> then the expression out<<s behaves as if f(s, n) were called, or if in is an instance of basic_istream<charT,traits> then the expression in>>s behaves as if f(s, n) were called. The function f can be defined as:

```
ios_base& f(ios_base& str, int n)
{
// set width
str.width(n);
return str;
}
```

The expression out<<s has type basic_ostream<charT,traits>& and value out. The expression in>>s has type basic_istream<charT,traits>& and value in.

[Kona: Andy Sawyer and Beman Dawes will work to improve the wording of the proposed resolution.]

[Tokyo - The LWG noted that issue 216 involves the same paragraphs.]

[Post-Tokyo: The issues list maintainer combined the proposed resolution of this issue with the proposed resolution for issue 216 as they both involved the same paragraphs, and were so intertwined that dealing with them separately appear fraught with error. The full text was supplied by Bill Plauger; it was cross checked against changes supplied by Andy Sawyer. It should be further checked by the LWG.]

---

184. numeric_limits<bool> wording problems

Section: 18.2.1.5 [lib.numeric.special] **Status:** DR **Submitter:** Gabriel Dos Reis **Date:** 21 Jul 1999

bools are defined by the standard to be of integer types, as per 3.9.1 paragraph 7. However "integer types" seems to have a special meaning for the author of 18.2. The net effect is an unclear and confusing specification for numeric_limits<bool> as evidenced below.

18.2.1.2/7 says numeric_limits<>::digits is, for built-in integer types, the number of non-sign bits in the representation.

4.5/4 states that a bool promotes to int ; whereas 4.12/1 says any non zero arithmetical value converts to true.

I don’t think it makes sense at all to require numeric_limits<bool>::digits and numeric_limits<bool>::digits10 to be meaningful.

The standard defines what constitutes a signed (resp. unsigned) integer types. It doesn’t categorize bool as being signed or unsigned. And the set of values of bool type has only two elements.

I don’t think it makes sense to require numeric_limits<bool>::is_signed to be meaningful.

18.2.1.2/18 for numeric_limits<integer_type>::radix says:

> For integer types, specifies the base of the representation.186)

This disposition is at best misleading and confusing for the standard requires a "pure binary numeration system" for integer types as per 3.9.1/7

The footnote 186) says: "Distinguishes types with base other than 2 (e.g BCD)." This also erroneous as the standard never defines any integer types with base representation other than 2.

Furthermore, numeric_limits<bool>::is_modulo and numeric_limits<bool>::is_signed have similar problems.

Proposed resolution:

Append to the end of 18.2.1.5 :

The specialization for bool shall be provided as follows:

```
namespace std {
   template<> class numeric_limits<bool> {
   public:
     static const bool is_specialized = true;
     static bool min() throw() { return false; }
     static bool max() throw() { return true; }

     static const int  digits = 1;
     static const int  digits10 = 0;
     static const bool is_signed = false;
     static const bool is_integer = true;
     static const bool is_exact = true;
     static const int  radix = 2;
     static bool epsilon() throw() { return 0; }
     static bool round_error() throw() { return 0; }

     static const int  min_exponent = 0;
     static const int  min_exponent10 = 0;
     static const int  max_exponent = 0;
     static const int  max_exponent10 = 0;

     static const bool has_infinity = false;
     static const bool has_quiet_NaN = false;
     static const bool has_signaling_NaN = false;
     static const float_denorm_style has_denorm = denorm_absent;
     static const bool has_denorm_loss = false;
     static bool infinity() throw() { return 0; }
     static bool quiet_NaN() throw() { return 0; }
     static bool signaling_NaN() throw() { return 0; }
     static bool denorm_min() throw() { return 0; }

     static const bool is_iec559 = false;
     static const bool is_bounded = true;
     static const bool is_modulo = false;

     static const bool traps = false;
     static const bool tinyness_before = false;
     static const float_round_style round_style = round_toward_zero;
   };
 }
```

[Tokyo: The LWG desires wording that specifies exact values rather than more general wording in the original proposed resolution.]

[Post-Tokyo: At the request of the LWG in Tokyo, Nico Josuttis provided the above wording.]

---

185. Questionable use of term "inline"

Section: 20.3 [lib.function.objects] **Status:** DR **Submitter:** UK Panel **Date:** 26 Jul 1999

Paragraph 4 of 20.3 says:

> [Example: To negate every element of a: transform(a.begin(), a.end(), a.begin(), negate<double>()); The corresponding functions will inline the addition and the negation. end example]

(Note: The "addition" referred to in the above is in para 3) we can find no other wording, except this (non-normative) example which suggests that any "inlining" will take place in this case.

Indeed both:

> 17.4.4.3 Global Functions [lib.global.functions] 1 It is unspecified whether any global functions in the C++ Standard Library are defined as inline (7.1.2).

and

> 17.4.4.4 Member Functions [lib.member.functions] 1 It is unspecified whether any member functions in the C++ Standard Library are defined as inline (7.1.2).

take care to state that this may indeed NOT be the case.

Thus the example "mandates" behavior that is explicitly not required elsewhere.

Proposed resolution:

In 20.3 paragraph 1, remove the sentence:

> They are important for the effective use of the library.

Remove 20.3 paragraph 2, which reads:

> Using function objects together with function templates increases the expressive power of the library as well as making the resulting code much more efficient.

In 20.3 paragraph 4, remove the sentence:

> The corresponding functions will inline the addition and the negation.

[Kona: The LWG agreed there was a defect.]

[Tokyo: The LWG crafted the proposed resolution.]

---

186. bitset::set() second parameter should be bool

Section: 23.3.5.2 [lib.bitset.members] **Status:** DR **Submitter:** Darin Adler **Date:** 13 Aug 1999

In section 23.3.5.2 , paragraph 13 defines the bitset::set operation to take a second parameter of type int. The function tests whether this value is non-zero to determine whether to set the bit to true or false. The type of this second parameter should be bool. For one thing, the intent is to specify a Boolean value. For another, the result type from test() is bool. In addition, it’s possible to slice an integer that’s larger than an int. This can’t happen with bool, since conversion to bool has the semantic of translating 0 to false and any non-zero value to true.

Proposed resolution:

In 23.3.5 Para 1 Replace:

```
bitset<N>& set(size_t pos, int val = true );
```

With:

```
bitset<N>& set(size_t pos, bool val = true );
```

In 23.3.5.2 Para 12(.5) Replace:

```
bitset<N>& set(size_t pos, int val = 1 );
```

With:

```
bitset<N>& set(size_t pos, bool val = true );
```

[Kona: The LWG agrees with the description. Andy Sawyer will work on better P/R wording.]

[Post-Tokyo: Andy provided the above wording.]

Rationale:

`bool` is a better choice. It is believed that binary compatibility is not an issue, because this member function is usually implemented as `inline`, and because it is already the case that users cannot rely on the type of a pointer to a nonvirtual member of a standard library class.

---

189. setprecision() not specified correctly

Section: 27.4.2.2 [lib.fmtflags.state] **Status:** DR **Submitter:** Andrew Koenig **Date:** 25 Aug 1999

27.4.2.2 paragraph 9 claims that setprecision() sets the precision, and includes a parenthetical note saying that it is the number of digits after the decimal point.
This claim is not strictly correct. For example, in the default floating-point output format, setprecision sets the number of significant digits printed, not the number of digits after the decimal point.
I would like the committee to look at the definition carefully and correct the statement in 27.4.2.2

Proposed resolution:

Remove from 27.4.2.2 , paragraph 9, the text "(number of digits after the decimal point)".

---

193. Heap operations description incorrect

Section: 25.3.6 [lib.alg.heap.operations] **Status:** DR **Submitter:** Markus Mauhart **Date:** 24 Sep 1999

25.3.6 [lib.alg.heap.operations] states two key properties of a heap [a,b), the first of them is
‘"(1) *a is the largest element"
I think this is incorrect and should be changed to the wording in the proposed resolution.

Actually there are two independent changes:

A-"part of largest equivalence class" instead of "largest", cause 25.3 [lib.alg.sorting] asserts "strict weak ordering" for all its sub clauses.

B-Take 'an oldest' from that equivalence class, otherwise the heap functions could not be used for a priority queue as explained in 23.2.3.2.2 [lib.priqueue.members] (where I assume that a "priority queue" respects priority AND time).

Proposed resolution:

Change 25.3.6 property (1) from:

(1) *a is the largest element

to:

(1) There is no element greater than `*a`

---

195. Should `basic_istream::sentry`'s constructor ever set eofbit?

Section: 27.6.1.1.2 [lib.istream::sentry] **Status:** DR **Submitter:** Matt Austern **Date:** 13 Oct 1999

Suppose that `is.flags() & ios_base::skipws` is nonzero. What should `basic_istream<>::sentry`'s constructor do if it reaches eof while skipping whitespace? 27.6.1.1.2/5 suggests it should set failbit. Should it set eofbit as well? The standard doesn't seem to answer that question.

On the one hand, nothing in 27.6.1.1.2 says that `basic_istream<>::sentry` should ever set eofbit. On the other hand, 27.6.1.1 paragraph 4 says that if extraction from a `streambuf` "returns `traits::eof()`, then the input function, except as explicitly noted otherwise, completes its actions and does `setstate(eofbit)`". So the question comes down to whether `basic_istream<>::sentry`'s constructor is an input function.

Comments from Jerry Schwarz:

> It was always my intention that eofbit should be set any time that a virtual returned something to indicate eof, no matter what reason iostream code had for calling the virtual.
>
> The motivation for this is that I did not want to require streambufs to behave consistently if their virtuals are called after they have signaled eof.
>
> The classic case is a streambuf reading from a UNIX file. EOF isn't really a state for UNIX file descriptors. The convention is that a read on UNIX returns 0 bytes to indicate "EOF", but the file descriptor isn't shut down in any way and future reads do not necessarily also return 0 bytes. In particular, you can read from tty's on UNIX even after they have signaled "EOF". (It isn't always understood that a ^D on UNIX is not an EOF indicator, but an EOL indicator. By typing a "line" consisting solely of ^D you cause a read to return 0 bytes, and by convention this is interpreted as end of file.)

Proposed resolution:

Add a sentence to the end of 27.6.1.1.2 paragraph 2:

> If `is.rdbuf()->sbumpc()` or `is.rdbuf()->sgetc()` returns `traits::eof()`, the function calls `setstate(failbit | eofbit)` (which may throw `ios_base::failure`).

---

199. What does `allocate(0)` return?

Section: 20.1.5 [lib.allocator.requirements] **Status:** DR **Submitter:** Matt Austern **Date:** 19 Nov 1999

Suppose that `A` is a class that conforms to the Allocator requirements of Table 32, and `a` is an object of class `A` What should be the return value of `a.allocate(0)`? Three reasonable possibilities: forbid the argument `0`, return a null pointer, or require that the return value be a unique non-null pointer.

Proposed resolution:

Add a note to the `allocate` row of Table 32: "[*Note:* If `n == 0`, the return value is unspecified. *--end note*]"

Rationale:

A key to understanding this issue is that the ultimate use of allocate() is to construct an iterator, and that iterator for zero length sequences must be the container's past-the-end representation. Since this already implies special case code, it would be over-specification to mandate the return value.

---

208. Unnecessary restriction on past-the-end iterators

Section: 24.1 [lib.iterator.requirements] **Status:** DR **Submitter:** Stephen Cleary **Date:** 02 Feb 2000

In 24.1 paragraph 5, it is stated ". . . Dereferenceable and past-the-end values are always non-singular."

This places an unnecessary restriction on past-the-end iterators for containers with forward iterators (for example, a singly-linked list). If the past-the-end value on such a container was a well-known singular value, it would still satisfy all forward iterator requirements.

Removing this restriction would allow, for example, a singly-linked list without a "footer" node.

This would have an impact on existing code that expects past-the-end iterators obtained from different (generic) containers being not equal.

Proposed resolution:

Change 24.1 paragraph 5, the last sentence, from:

> Dereferenceable and past-the-end values are always non-singular.

to:

> Dereferenceable values are always non-singular.

Rationale:

For some kinds of containers, including singly linked lists and zero-length vectors, null pointers are perfectly reasonable past-the-end iterators. Null pointers are singular.

---

209. basic_string declarations inconsistent

Section: 21.3 [lib.basic.string] **Status:** DR **Submitter:** Igor Stauder **Date:** 11 Feb 2000

In Section 21.3 the basic_string member function declarations use a consistent style except for the following functions:

```
void push_back(const charT);
basic_string& assign(const basic_string&);
void swap(basic_string<charT,traits,Allocator>&);
```

- push_back, assign, swap: missing argument name
- push_back: use of const with charT (i.e. POD type passed by value not by reference - should be charT or const charT&)
- swap: redundant use of template parameters in argument basic_string<charT,traits,Allocator>&

Proposed resolution:

In Section 21.3 change the basic_string member function declarations push_back, assign, and swap to:

```
void push_back(charT c);

basic_string& assign(const basic_string& str);
void swap(basic_string& str);
```

Rationale:

Although the standard is in general not consistent in declaration style, the basic_string declarations are consistent other than the above. The LWG felt that this was sufficient reason to merit the change.

---

210. distance first and last confused

Section: 25 [lib.algorithms] **Status:** DR **Submitter:** Lisa Lippincott **Date:** 15 Feb 2000

In paragraph 9 of section 25 , it is written:

> In the description of the algorithms operators + and - are used for some of the iterator categories for which they do not have to be defined. In these cases the semantics of [...] a-b is the same as of
> `return distance(a, b);`

Proposed resolution:

On the last line of paragraph 9 of section 25 change `"a-b"` to `"b-a".`

Rationale:

There are two ways to fix the defect; change the description to b-a or change the return to distance(b,a). The LWG preferred the former for consistency.

---

211. operator>>(istream&, string&) doesn't set failbit

Section: 21.3.7.9 [lib.string.io] **Status:** DR **Submitter:** Scott Snyder **Date:** 4 Feb 2000

The description of the stream extraction operator for std::string (section 21.3.7.9 [lib.string.io]) does not contain a requirement that failbit be set in the case that the operator fails to extract any characters from the input stream.

This implies that the typical construction

```
std::istream is;
std::string str;
...
while (is >> str) ... ;
```

(which tests failbit) is not required to terminate at EOF.

Furthermore, this is inconsistent with other extraction operators, which do include this requirement. (See sections 27.6.1.2 and 27.6.1.3), where this requirement is present, either explicitly or implicitly, for the extraction operators. It is also present explicitly in the description of getline (istream&, string&, charT) in section 21.3.7.9 paragraph 8.)

Proposed resolution:

Insert new paragraph after paragraph 2 in section 21.3.7.9 :

> If the function extracts no characters, it calls is.setstate(ios::failbit) which may throw ios_base::failure (27.4.4.3).

---

212. Empty range behavior unclear for several algorithms

Section: 25.3.7 [lib.alg.min.max] **Status:** DR **Submitter:** Nico Josuttis **Date:** 26 Feb 2000

The standard doesn't specify what min_element() and max_element() shall return if the range is empty (first equals last). The usual implementations return last. This problem seems also apply to partition(), stable_partition(), next_permutation(), and prev_permutation().

Proposed resolution:

In 25.3.7 - Minimum and maximum, paragraphs 7 and 9, append: Returns last if first==last.

Rationale:

The LWG looked in some detail at all of the above mentioned algorithms, but believes that except for min_element() and max_element() it is already clear that last is returned if first == last.

---

214. set::find() missing const overload

Section: 23.3.3 [lib.set], 23.3.4 [lib.multiset] **Status:** DR **Submitter:** Judy Ward **Date:** 28 Feb 2000

The specification for the associative container requirements in Table 69 state that the find member function should "return iterator; const_iterator for constant a". The map and multimap container descriptions have two overloaded versions of find, but set and multiset do not, all they have is:

```
iterator find(const key_type & x) const;
```

Proposed resolution:

Change the prototypes for find(), lower_bound(), upper_bound(), and equal_range() in section 23.3.3 and section 23.3.4 to each have two overloads:

```
iterator find(const key_type & x);
const_iterator find(const key_type & x) const;

iterator lower_bound(const key_type & x);
const_iterator lower_bound(const key_type & x) const;

iterator upper_bound(const key_type & x);
const_iterator upper_bound(const key_type & x) const;

pair<iterator, iterator> equal_range(const key_type & x);
pair<const_iterator, const_iterator> equal_range(const key_type & x) const;
```

[Tokyo: At the request of the LWG, Judy Ward provided wording extending the proposed resolution to lower_bound, upper_bound, and equal_range.]

---

217. Facets example (Classifying Japanese characters) contains errors

Section: 22.2.8 [lib.facets.examples] **Status:** DR **Submitter:** Martin Sebor **Date:** 29 Feb 2000

The example in 22.2.8, paragraph 11 contains the following errors:

1) The member function ‘My::JCtype::is_kanji()’ is non-const; the function must be const in order for it to be callable on a const object (a reference to which which is what std::use_facet<>() returns).

2) In file filt.C, the definition of ‘JCtype::id’ must be qualified with the name of the namespace ‘My’.

3) In the definition of ‘loc’ and subsequently in the call to use_facet<>() in main(), the name of the facet is misspelled: it should read ‘My::JCtype’ rather than ‘My::JCType’.

Proposed resolution:

Replace the "Classifying Japanese characters" example in 22.2.8, paragraph 11 with the following:

```
#include <locale>

namespace My {
    using namespace std;
    class JCtype : public locale::facet {
    public:
        static locale::id id;     //  required for use as a new locale facet
        bool is_kanji (wchar_t c) const;
        JCtype() {}
    protected:
        ~JCtype() {}
    };
}

//  file:  filt.C
#include <iostream>
#include <locale>
#include "jctype"                 //  above
std::locale::id My::JCtype::id;   //  the static  JCtype  member
declared above.

int main()
{
    using namespace std;
    typedef ctype<wchar_t> wctype;
    locale loc(locale(""),              //  the user’s preferred locale...
               new My::JCtype);         //  and a new feature ...
    wchar_t c = use_facet<wctype>(loc).widen(’!’);
    if (!use_facet<My::JCtype>(loc).is_kanji(c))
        cout << "no it isn’t!" << endl;
    return 0;
}
```

---

220. ~ios_base() usage valid?

Section: 27.4.2.7 [lib.ios.base.cons] **Status:** DR **Submitter:** Jonathan Schilling, Howard Hinnant **Date:** 13 Mar 2000

The pre-conditions for the ios_base destructor are described in 27.4.2.7 paragraph 2:

> Effects: Destroys an object of class ios_base. Calls each registered callback pair (fn,index) (27.4.2.6) as (*fn)(erase_event,*this,index) at such time that any ios_base member function called from within fn has well defined results.

But what is not clear is: If no callback functions were ever registered, does it matter whether the ios_base members were ever initialized?

For instance, does this program have defined behavior:

```
#include <ios>

class D : public std::ios_base { };

int main() { D d; }
```

It seems that registration of a callback function would surely affect the state of an ios_base. That is, when you register a callback function with an ios_base, the ios_base must record that fact somehow.

But if after construction the ios_base is in an indeterminate state, and that state is not made determinate before the destructor is called, then how would the destructor know if any callbacks had indeed been registered? And if the number of callbacks that had been registered is indeterminate, then is not the behavior of the destructor undefined?

By comparison, the basic_ios class description in 27.4.4.1 paragraph 2 makes it explicit that destruction before initialization results in undefined behavior.

Proposed resolution:

Modify 27.4.2.7 paragraph 1 from

> Effects: Each ios_base member has an indeterminate value after construction.

to

> Effects: Each ios_base member has an indeterminate value after construction. These members must be initialized by calling basic_ios::init. If an ios_base object is destroyed before these initializations have taken place, the behavior is undefined.

---

221. num_get<>::do_get stage 2 processing broken

Section: 22.2.2.1.2 [lib.facet.num.get.virtuals] **Status:** DR **Submitter:** Matt Austern **Date:** 14 Mar 2000

Stage 2 processing of numeric conversion is broken.

Table 55 in 22.2.2.1.2 says that when basefield is 0 the integral conversion specifier is %i. A %i specifier determines a number's base by its prefix (0 for octal, 0x for hex), so the intention is clearly that a 0x prefix is allowed. Paragraph 8 in the same section, however, describes very precisely how characters are processed. (It must be done "as if" by a specified code fragment.) That description does not allow a 0x prefix to be recognized.

Very roughly, stage 2 processing reads a char_type ct. It converts ct to a char, not by using narrow but by looking it up in a translation table that was created by widening the string literal "0123456789abcdefABCDEF+-". The character "x" is not found in that table, so it can't be recognized by stage 2 processing.

Proposed resolution:

In 22.2.2.1.2 paragraph 8, replace the line:

```
static const char src[] = "0123456789abcdefABCDEF+-";
```

with the line:

```
static const char src[] = "0123456789abcdefxABCDEFX+-";
```

Rationale:

If we're using the technique of widening a string literal, the string literal must contain every character we wish to recognize. This technique has the consequence that alternate representations of digits will not be recognized. This design decision was made deliberately, with full knowledge of that limitation.

---

222. Are throw clauses necessary if a throw is already implied by the effects clause?

Section: 17.3.1.3 [lib.structure.specifications] **Status:** DR **Submitter:** Judy Ward **Date:** 17 Mar 2000

Section 21.3.6.8 describes the basic_string::compare function this way:

```
21.3.6.8 - basic_string::compare [lib.string::compare]

int compare(size_type pos1, size_type n1,
               const basic_string<charT,traits,Allocator>&  str ,
               size_type  pos2 , size_type  n2 ) const;

-4- Returns:

    basic_string<charT,traits,Allocator>(*this,pos1,n1).compare(
                basic_string<charT,traits,Allocator>(str,pos2,n2)) .
```

and the constructor that's implicitly called by the above is defined to throw an out-of-range exception if pos > str.size(). See section 21.3.1 paragraph 4.

On the other hand, the compare function descriptions themselves don't have "Throws: " clauses and according to 17.3.1.3, paragraph 3, elements that do not apply to a function are omitted.

So it seems there is an inconsistency in the standard -- are the "Effects" clauses correct, or are the "Throws" clauses missing?

Proposed resolution:

In 17.3.1.3 paragraph 3, the footnote 148 attached to the sentence "Descriptions of function semantics contain the following elements (as appropriate):", insert the word "further" so that the foot note reads:

> To save space, items that do not apply to a function are omitted. For example, if a function does not specify any further preconditions, there will be no "Requires" paragraph.

Rationale:

The standard is somewhat inconsistent, but a failure to note a throw condition in a throws clause does not grant permission not to throw. The inconsistent wording is in a footnote, and thus non-normative. The proposed resolution from the LWG clarifies the footnote.

---

223. reverse algorithm should use iter_swap rather than swap

Section: 25.2.9 [lib.alg.reverse] **Status:** DR **Submitter:** Dave Abrahams **Date:** 21 Mar 2000

Shouldn't the effects say "applies iter_swap to all pairs..."?

Proposed resolution:

In 25.2.9 , replace:

> Effects: For each non-negative integer i <= (last - first)/2, applies swap to all pairs of iterators first + i, (last - i) - 1.

with:

> Effects: For each non-negative integer i <= (last - first)/2, applies iter_swap to all pairs of iterators first + i, (last - i) - 1.

---

224. clear() complexity for associative containers refers to undefined N

Section: 23.1.2 [lib.associative.reqmts] **Status:** DR **Submitter:** Ed Brey **Date:** 23 Mar 2000

In the associative container requirements table in 23.1.2 paragraph 7, a.clear() has complexity "log(size()) + N". However, the meaning of N is not defined.

Proposed resolution:

In the associative container requirements table in 23.1.2 paragraph 7, the complexity of a.clear(), change "log(size()) + N" to "linear in `size()`".

Rationale:

It's the "log(size())", not the "N", that is in error: there's no difference between *O(N)* and *O(N + log(N))*. The text in the standard is probably an incorrect cut-and-paste from the range version of `erase`.

---

227. std::swap() should require CopyConstructible or DefaultConstructible arguments

Section: 25.2.2 [lib.alg.swap] **Status:** DR **Submitter:** Dave Abrahams **Date:** 09 Apr 2000

25.2.2 reads:

```
template<class T> void swap(T& a, T& b);
```
Requires: Type T is Assignable (_lib.container.requirements_).
Effects: Exchanges values stored in two locations.

The only reasonable** generic implementation of swap requires construction of a new temporary copy of one of its arguments:

```
template<class T> void swap(T& a, T& b);
  {
      T tmp(a);
      a = b;
      b = tmp;
  }
```

But a type which is only Assignable cannot be swapped by this implementation.

**Yes, there's also an unreasonable implementation which would require T to be DefaultConstructible instead of CopyConstructible. I don't think this is worthy of consideration:

```
template<class T> void swap(T& a, T& b);
{
    T tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

Proposed resolution:

Change 25.2.2 paragraph 1 from:

> Requires: Type T is Assignable (23.1).

to:

> Requires: Type T is CopyConstructible (20.1.3) and Assignable (23.1)

---

234. Typos in allocator definition

Section: 20.4.1.1 [lib.allocator.members] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 24 Apr 2000

In paragraphs 12 and 13 the effects of `construct()` and `destruct()` are described as returns but the functions actually return `void`.

Proposed resolution:

Substitute "Returns" by "Effect".

---

237. Undefined expression in complexity specification

Section: 23.2.2.1 [lib.list.cons] **Status:** DR **Submitter:** Dietmar Kühl **Date:** 24 Apr 2000

The complexity specification in paragraph 6 says that the complexity is linear in `first - last`. Even if `operator-()` is defined on iterators this term is in general undefined because it would have to be `last - first`.

Proposed resolution:

Change paragraph 6 from

> Linear in *first - last*.

to become

> Linear in *distance(first, last)*.

---

243. `get` and `getline` when sentry reports failure

Section: 27.6.1.3 [lib.istream.unformatted] **Status:** DR **Submitter:** Martin Sebor **Date:** May 15 2000

basic_istream<>::get(), and basic_istream<>::getline(), are unclear with respect to the behavior and side-effects of the named functions in case of an error.

27.6.1.3, p1 states that "... If the sentry object returns true, when converted to a value of type bool, the function endeavors to obtain the requested input..." It is not clear from this (or the rest of the paragraph) what precisely the behavior should be when the sentry ctor exits by throwing an exception or when the sentry object returns false. In particular, what is the number of characters extracted that gcount() returns supposed to be?

27.6.1.3 p8 and p19 say about the effects of get() and getline(): "... In any case, it then stores a null character (using charT()) into the next successive location of the array." Is not clear whether this sentence applies if either of the conditions above holds (i.e., when sentry fails).

Proposed resolution:

Add to 27.6.1.3, p1 after the sentence

> "... If the sentry object returns true, when converted to a value of type bool, the function endeavors to obtain the requested input."

the following

> "Otherwise, if the sentry constructor exits by throwing an exception or if the sentry object returns false, when converted to a value of type bool, the function returns without attempting to obtain any input. In either case the number of extracted characters is set to 0; unformatted input functions taking a character array of non-zero size as an argument shall also store a null character (using charT()) in the first location of the array."

Rationale:

Although the general philosophy of the input functions is that the argument should not be modified upon failure, `getline` historically added a terminating null unconditionally. Most implementations still do that. Earlier versions of the draft standard had language that made this an unambiguous requirement; those words were moved to a place where their context made them less clear. See Jerry Schwarz's message c++std-lib-7618.

---

248. time_get fails to set eofbit

Section: 22.2.5 [lib.category.time] **Status:** DR **Submitter:** Martin Sebor **Date:** 22 June 2000

There is no requirement that any of time_get member functions set ios::eofbit when they reach the end iterator while parsing their input. Since members of both the num_get and money_get facets are required to do so (22.2.2.1.2, and 22.2.6.1.2, respectively), time_get members should follow the same requirement for consistency.

Proposed resolution:

Add paragraph 2 to section 22.2.5.1 with the following text:

> If the end iterator is reached during parsing by any of the get() member functions, the member sets ios_base::eofbit in err.

Rationale:

Two alternative resolutions were proposed. The LWG chose this one because it was more consistent with the way eof is described for other input facets.

---

251. basic_stringbuf missing allocator_type

Section: 27.7.1 [lib.stringbuf] **Status:** DR **Submitter:** Martin Sebor **Date:** 28 Jul 2000

The synopsis for the template class `basic_stringbuf` doesn't list a typedef for the template parameter `Allocator`. This makes it impossible to determine the type of the allocator at compile time. It's also inconsistent with all other template classes in the library that do provide a typedef for the `Allocator` parameter.

Proposed resolution:

Add to the synopses of the class templates basic_stringbuf (27.7.1), basic_istringstream (27.7.2), basic_ostringstream (27.7.3), and basic_stringstream (27.7.4) the typedef:

```
typedef Allocator allocator_type;
```

---

252. missing casts/C-style casts used in iostreams

Section: 27.7 [lib.string.streams] **Status:** DR **Submitter:** Martin Sebor **Date:** 28 Jul 2000

27.7.2.2, p1 uses a C-style cast rather than the more appropriate const_cast<> in the Returns clause for basic_istringstream<>::rdbuf(). The same C-style cast is being used in 27.7.3.2, p1, D.7.2.2, p1, and D.7.3.2, p1, and perhaps elsewhere. 27.7.6, p1 and D.7.2.2, p1 are missing the cast altogether.

C-style casts have not been deprecated, so the first part of this issue is stylistic rather than a matter of correctness.

Proposed resolution:

In 27.7.2.2, p1 replace

```
-1- Returns: (basic_stringbuf<charT,traits,Allocator>*)&sb.
```

with

```
-1- Returns: const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb).
```

In 27.7.3.2, p1 replace

```
-1- Returns: (basic_stringbuf<charT,traits,Allocator>*)&sb.
```

with

```
-1- Returns: const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb).
```

In 27.7.6, p1, replace

```
-1- Returns: &sb
```

with

```
-1- Returns: const_cast<basic_stringbuf<charT,traits,Allocator>*>(&sb).
```

In D.7.2.2, p1 replace

```
-2- Returns: &sb.
```

with

```
-2- Returns: const_cast<strstreambuf*>(&sb).
```

---

256. typo in 27.4.4.2, p17: copy_event does not exist

Section: 27.4.4.2 [lib.basic.ios.members] **Status:** DR **Submitter:** Martin Sebor **Date:** 21 Aug 2000

27.4.4.2, p17 says

> -17- Before copying any parts of rhs, calls each registered callback pair (fn,index) as (*fn)(erase_event,*this,index). After all parts but exceptions() have been replaced, calls each callback pair that was copied from rhs as (*fn)(copy_event,*this,index).

The name copy_event isn't defined anywhere. The intended name was copyfmt_event.

Proposed resolution:

Replace copy_event with copyfmt_event in the named paragraph.

---

260. Inconsistent return type of `istream_iterator::operator++(int)`

Section: 24.5.1.2 [lib.istream.iterator.ops] **Status:** DR **Submitter:** Martin Sebor **Date:** 27 Aug 2000

The synopsis of istream_iterator::operator++(int) in 24.5.1 shows it as returning the iterator by value. 24.5.1.2, p5 shows the same operator as returning the iterator by reference. That's incorrect given the Effects clause below (since a temporary is returned). The '&' is probably just a typo.

Proposed resolution:

Change the declaration in 24.5.1.2, p5 from

```
istream_iterator<T,charT,traits,Distance>& operator++(int);
```

to

```
istream_iterator<T,charT,traits,Distance> operator++(int);
```

(that is, remove the '&').

---

261. Missing description of `istream_iterator::operator!=`

Section: 24.5.1.2 [lib.istream.iterator.ops] **Status:** DR **Submitter:** Martin Sebor **Date:** 27 Aug 2000

24.5.1, p3 lists the synopsis for

```
template <class T, class charT, class traits, class Distance>
     bool operator!=(const istream_iterator<T,charT,traits,Distance>& x,
                     const istream_iterator<T,charT,traits,Distance>& y);
```

but there is no description of what the operator does (i.e., no Effects or Returns clause) in 24.5.1.2.

Proposed resolution:

Add paragraph 7 to the end of section 24.5.1.2 with the following text:

```
template <class T, class charT, class traits, class Distance>
     bool operator!=(const istream_iterator<T,charT,traits,Distance>& x,
                     const istream_iterator<T,charT,traits,Distance>& y);
```

-7- Returns: !(x == y).

---

262. Bitmask operator ~ specified incorrectly

Section: 17.3.2.1.2 [lib.bitmask.types] **Status:** DR **Submitter:** Beman Dawes **Date:** 03 Sep 2000

The ~ operation should be applied after the cast to int_type.

Proposed resolution:

Change 17.3.2.1.2 [lib.bitmask.types] operator~ from:

```
bitmask operator~ ( bitmask X )
  { return static_cast< bitmask>(static_cast<int_type>(~ X)); }
```

to:

```
bitmask operator~ ( bitmask X )
  { return static_cast< bitmask>(~static_cast<int_type>(X)); }
```

---

263. Severe restriction on `basic_string` reference counting

Section: 21.3 [lib.basic.string] **Status:** DR **Submitter:** Kevlin Henney **Date:** 04 Sep 2000

The note in paragraph 6 suggests that the invalidation rules for references, pointers, and iterators in paragraph 5 permit a reference- counted implementation (actually, according to paragraph 6, they permit a "reference counted implementation", but this is a minor editorial fix).

However, the last sub-bullet is so worded as to make a reference-counted implementation unviable. In the following example none of the conditions for iterator invalidation are satisfied:

```
// first example: "********************" should be printed twice
string original = "some arbitrary text", copy = original;
const string & alias = original;

string::const_iterator i = alias.begin(), e = alias.end();
for(string::iterator j = original.begin(); j != original.end(); ++j)
    *j = '*';
while(i != e)
    cout << *i++;
cout << endl;
cout << original << endl;
```

Similarly, in the following example:

```
// second example: "some arbitrary text" should be printed out
string original = "some arbitrary text", copy = original;
const string & alias = original;

string::const_iterator i = alias.begin();
original.begin();
while(i != alias.end())
    cout << *i++;
```

I have tested this on three string implementations, two of which were reference counted. The reference-counted implementations gave "surprising behavior" because they invalidated iterators on the first call to non-const begin since construction. The current wording does not permit such invalidation because it does not take into account the first call since construction, only the first call since various member and non-member function calls.

Proposed resolution:

Change the following sentence in 21.3 paragraph 5 from

> Subsequent to any of the above uses except the forms of insert() and erase() which return iterators, the first call to non-const member functions operator[](), at(), begin(), rbegin(), end(), or rend().

to

> Following construction or any of the above uses, except the forms of insert() and erase() that return iterators, the first call to non- const member functions operator[](), at(), begin(), rbegin(), end(), or rend().

---

265. std::pair::pair() effects overly restrictive

Section: 20.2.2 [lib.pairs] **Status:** DR **Submitter:** Martin Sebor **Date:** 11 Sep 2000

I don't see any requirements on the types of the elements of the std::pair container in 20.2.2. From the descriptions of the member functions it appears that they must at least satisfy the requirements of 20.1.3 [lib.copyconstructible] and 20.1.4 [lib.default.con.req], and in the case of the [in]equality operators also the requirements of 20.1.1 [lib.equalitycomparable] and 20.1.2 [lib.lessthancomparable].

I believe that the the CopyConstructible requirement is unnecessary in the case of 20.2.2, p2.

Proposed resolution:

Change the Effects clause in 20.2.2, p2 from

-2- **Effects**: Initializes its members as if implemented: `pair() : first(T1()), second(T2()) {}`

to

-2- **Effects**: Initializes its members as if implemented: `pair() : first(), second() {}`

Rationale:

The existing specification of pair's constructor appears to be a historical artifact: there was concern that pair's members be properly zero-initialized when they are built-in types. At one time there was uncertainty about whether they would be zero-initialized if the default constructor was written the obvious way. This has been clarified by core issue 178, and there is no longer any doubt that the straightforward implementation is correct.

---

268. Typo in locale synopsis

Section: 22.1.1 [lib.locale] **Status:** DR **Submitter:** Martin Sebor **Date:** 5 Oct 2000

The synopsis of the class std::locale in 22.1.1 contains two typos: the semicolons after the declarations of the default ctor locale::locale() and the copy ctor locale::locale(const locale&) are missing.

Proposed resolution:

Add the missing semicolons, i.e., change

```
    //  construct/copy/destroy:
        locale() throw()
        locale(const locale& other) throw()
```

in the synopsis in 22.1.1 to

```
    //  construct/copy/destroy:
        locale() throw();
        locale(const locale& other) throw();
```

----- End of document -----