Doc. No.:	X3J16 / 96-0166
	WG21 / N0984
Date:	July 23, 1996
Project:	Programming Language C++
Reply To:	Richard K. Wilhelm
	Strategic Technology Resources
	rwilhelm@str.com

Clause 21 (Strings Library) Issues List
Revision 18

Revision History

Version 1 - January 30, 1995: Distributed in pre-Austin mailing.
Version 2 - March 6, 1995: Distributed at Austin meeting.
Version 3 - March 24, 1995: Distributed in post-Austin mailing. Several issues added. Several issues updated to reflect decisions at Austin meeting.
Version 4 - May 19, 1995: Distributed in pre-Monterey mailing.
Version 5 - July 9, 1995: Distributed at the Monterey meeting. Includes many issues added from public comments.
Version 6 - July 11, 1995: Distributed at the Monterey meeting. Added no new issues from previous version. Included issues prepared for formal vote. Added solutions for issues 8, 21,31, 38, 69, 71. Made only changes to reflect the decisions of the string sub-group, correct working paper text and to correct typographical errors.
Version 7 - July 27, 1995: Distributed in the post-Monterey mailing. Reflects the resolutions and discussions of the Monterey meeting.
Version 8 - September 24, 1995: Distributed in the pre-Tokyo mailing. Some new issues added.
Version 9 - November 2, 1995: Distributed at the Tokyo meeting. Added issue 79. Added solutions for issues: 29, 30, 61, 62, and 63.
Version 10 - November 8, 1995: Distributed at the Tokyo meeting. Contains resolutions for issues to be closed by a vote.
Version 11 - December 2, 1995: Distributed in the post-Tokyo mailing. Updated issues closed in Tokyo. Added several new issues
Version 12 - January 29, 1996: Distributed in the pre-Santa Cruz mailing.
Version 13 - March 10, 1996: Distributed at the Santa Cruz meeting.
Version 14 - March 13, 1996: Distributed at the Santa Cruz meeting. Reflects changes to resolutions make by the library group.
Version 15 - March 28, 1996: Distributed in the post-Santa Cruz mailing. Updated issues closed in Santa Cruz.
Version 16 - May 28, 1996: Distributed in the pre-Stockholm mailing.
Version 17 - July 9, 1996: Distributed at the Stockholm meeting.
Version 18 - July 23, 1996: Distributed in the post-Stockholm mailing. Contains revisions made a the Stockholm meeting and voted in as N0948R1 / 96-0130R1.

Introduction

This document is a summary of the issues identified in Clause 21. For each issue the status, a short description, and pointers to relevant reflector messages and papers are given. This evolving document will serve as a basis of discussion and historical record for Strings issues and as a foundation of proposals for resolving specific issues.

For clarity, active issues are separated from issues recently closed. Closed issues are retained for one revision of the paper to serve as a record of recent resolutions. Subsequently, they will be removed from the paper for brevity. Any issue which has been removed will include the document number of the final paper in which it was included.

Active Issues

Issue Number: 21-090
Title: operator>> consuming whitespace
Section: 21.1.1.10.8 [lib.string.io]
Status: active
Description:

From a public comment:
"It seems to me that, to be useful, operator>>() must eat zero or more delimiters specified by basic_string<...>::traits::is_del() prior to reading each string. This should be specifed in the standard, to prevent varying implementations. If that is not the committee's intent, it should be explicitly stated in the standard what the intent is."

Judy Ward (j_ward@decc.enet.dec.com) commented that operator>> should call is.ipfx() not is.ipfx(true); calling ipfx(true) does not skip white space.

Proposed Resolution:

No change, close the issue. Issues made moot by acceptance of 96-0147 in Stockholm.

Originally proposed solution was:
In 21.1.1.10.8 [lib.string.io], change the call to `is.ipfx(true)` to `is.ipfx()`.

Requester: John Mulhern (jmulhern@empros.com).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-095
Title: basic_string::getline() cannot set the count in basic_istream
Section: 21.2.1.8.9 [lib.string.io]
Status: active
Description:

The description of the getline() member states:

> The function ends by storing the count in `is`…

There is no basic_istream member which would allow this to happen.

Proposed Resolution:

No change, close the issue. Issues made moot by acceptance of 96-0147 in Stockholm.

Requester: Judy Ward (j_ward@decc.enet.dec.com).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-111
Title: operator>>() using non-existent traits member
Section: 21.2.1.8.9 [lib.string.io]

Status:	active
Description:	

The description of the operator>>() member states:

IS_traits::is_whitespace(c,ctype) is true for the next available input character c, where ctype is acquired by calling use_facet<ctype<charT> >(is.getloc()).

The is_whitespace member does not exist.

Proposed Resolution:

No change, close the issue. Issues made moot by acceptance of 96-0147 in Stockholm.

Requester:	Rick Wilhelm <rwilhelm@str.com>
Owner:	
Emails:	(none)
Papers:	(none)

Closed Issues

Issues which have been recently closed are included in their entirety. Issues which have appeared in a previous version of the issues list as "closed" have the bulk of their content deleted for brevity. The document number of the paper in which they last appeared is included in parentheses for reference.

21-001 Should basic_string have a getline() function? (N0721=95-0121)
21-002 Are string_traits members char_in() and char_out() necessary? (N0815=95-0215)
21-003 Character-oriented assign function has incorrect signature (N0721=95-0121)
21-004 Character-oriented replace function has incorrect signature (N0759=95-0159)
21-005 How come the string class does not have a prepend() function? (N0759=95-0159)
21-006 Should the Allocator be the last template argument to basic_string? (N0721=95-0121)
21-007 Should the string_char_traits speed-up functions be specified as inline? (N0759=95-0159)
21-008 Should an iostream inserter and extractor be specified for basic_string? (N0759=95-0159)
21-009 Why are character parameters passed as "const charT"? (N0721=95-0121)
21-010 Should member parameters passed as "const_pointer"? (N0721=95-0121)
21-011 Why are character parameters to the string traits functions passed by reference? (N0721=95-0121)
21-012 Why are character parameters to the string functions passed by value? (N0800=95-0200)
21-013 There is no provision for errors caused by implementation limits. (N0815=95-0215)
21-014 Argument order for copy() is incorrect. (N0899=96-0081)
21-015 The copy() member should be const. (N0759=95-0159)
21-016 The error conditions are not well-specified for the find() and rfind() functions. (N0759=95-0159)
21-017 Can `reserve()` cause construction of characters? (N0815=95-0215)
21-018 Specification of traits class is constraining. (N0815=95-0215)
21-019 The `Allocator` template parameter is not reflected in a member typedef. (N0759=95-0159)
21-020 Header for Table 42 is incorrect. (N0759=95-0159)
21-021 `compare()` has unexpected results (N0759=95-0159)
21-022 s.append('c') appends 99 nulls. (N0759=95-0159)
21-023 Non-conforming default `Allocator` arguments (N0759=95-0159)
21-024 Name of traits delimiter function is confusing (N0815=95-0215)
21-025 Does `string_char_traits` need a locale? (N0815=95-0215)
21-026 Description of `string_char_traits::compare()` is expressed in code. (N0815=95-0215)

21-027 Description of `string_char_traits::compare()` overspecifies return value. (N0815=95-0215)
21-028 Description of `string_char_traits::length()` is expressed in code. (N0815=95-0215)
21-029 Description of `string_char_traits::copy()` is overconstraining. (N0815=95-0215)
21-030 Description of `string_char_traits::copy()` is silent on overlapping strings. (N0815=95-0215)
21-031 Copy constructor takes extra argument to switch allocator but does not allow allocator to remain the same. (N0815=95-0215)
21-032 Description for operator+() is incorrect (N0759=95-0159)
21-033 Requirements for `const charT*` arguments not specified (N0759=95-0159)
21-034 Inconsistency in requirements statements involving npos (N0815=95-0215)
21-034a Expand ability to throw length_error (N0815=95-0215)
21-035 Character replacement does not change length. (N0759=95-0159)
21-036 Character case disregarded during common operations. (N0759=95-0159)
21-037 Traits needs a move() for overlapping copies. (N0815=95-0215)
21-038 Operator < clashes cause ambiguity (N0759=95-0159)
21-039 Iterator parameters can get confused with size_type parameters. (N0759=95-0159)
21-040 Repetition parameter non-intuitive (N0759=95-0159)
21-041 Assignment operator defined in terms of itself (N0759=95-0159)
21-042 Character assignment defined in terms of non-existent constructor (N0759=95-0159)
21-043 Character append operator defined in terms of non-existent constructor (N0759=95-0159)
21-044 Character modifiers defined in terms of non-existent constructor (N0759=95-0159)
21-045 Iterator typenames overspecified (N0759=95-0159)
21-046 basic_string type syntactically incorrect in some descriptions (N0759=95-0159)
21-047 Error in description of replace() member (N0759=95-0159)
21-048 Inconsistency in const-ness of compare() declarations (N0759=95-0159)
21-049 Inconsistency constructor effects and semantics of data() (N0759=95-0159)
21-050 Incorrect semantics for operator+() (N0759=95-0159)
21-051 Incorrect return type for insert() member (N0759=95-0159)
21-052 Unconstrained position arguments for find members. (N0759=95-0159)
21-053 Semantics of size() prevents null characters in string (N0759=95-0159)
21-054 Change the semantics of length() (N0759=95-0159)
21-055 append(), assign() have incorrect requirements (N0759=95-0159)
21-056 Requirements for insert() are too weak. (N0759=95-0159)
21-057 replace has incorrect requirements (N0759=95-0159)
21-058 Description of data() is over-constraining. (N0759=95-0159)
21-059 String traits have no relationship to iostream traits. (N0899=96-0081)
21-060 string_char_traits::ne not needed (N0815=95-0215)
21-061 Missing explanation of traits specialization (N0815=95-0215)

Issue Number: 21-062

Title:	Missing explanation of requirements on `charT`.
Section:	21.1.1 [lib.char.traits.defs]
Status:	closed

Description:

A public comment noted:
Paragraph 1 doesn't say enough about the properties of a "char-like object." It should say that it doesn't need to be constructed or destroyed (otherwise, the primitives in string_char_traits are woefully inadequate).
string_char_traits::assign (and copy) must suffice either to copy or initialize a char-like element. The definition should also say than an allocator must have the same definitions for the types size_type, difference_type, pointer, const_pointer,

reference, and const_reference as class allocator::types<charT> (again because string_char_traits has no provision for funny address types).

In all-1437, Jack Reeves <76217.2354@CompuServe.COM> commented:

Now, I have to say that when I started writing my string implementation, I added these functions to class "string_char_traits" and tried to use them. I finally decided that I did not like the resulting code, and changed my mind about the desirability of allowing basic_string<> to be instantiated with any class type. Instead, I chose the following:

2. Restrict the range of user defined types that can be used to instantiate basic_string<> as follows:
The class must have a copy constructor equivalent to a "trivial" copy constructor, an assignment operator equivalent to the "trivial" assignment operator, and a destructor equivalent to a "trivial" destructor. In other words, no virtual functions and it must be possible to copy construct (or assign) an object of the type by a simple "memcpy" from an existing object, and it must be possible to 'destroy' an object of the type by simply deleting its memory.

Recommendation:
I am sure that everyone would like for basic_string<> to be truly general purpose, my gut feeling is that it really has to work best with objects that meet the second suggestion above. Without this restriction, basic_string<> is going to be more complicated, even with the speed-up functions I outlined in suggestion 1. This is because of the need to determine at all steps whether the char-like objects can be copied, or must be copy constructed. While library vendors would probably provide specialization's for basic_string<char> and basic_string<wchar_t> that were more efficient than the instantiated versions, I doubt if users would be very happy to discover that they were not able to get similar performance out of basic_string<unsigned char> and similar simple character types.

I recommend that the committee refine the meaning of char-like and restrict it to user defined types that behave similarly to the built-in types with regards to copy construction and destruction. Users can always use vector<> for manipulating more complicated objects.

Resolution:

Change the first sentence of 21 [lib.strings] from:

This clause describes components for manipulating sequences of “characters” where characters may be of type char, wchar_t, or of a type defined in a C++ program.

To:

This clause describes components for manipulating sequences of “characters” where characters may be of any POD ([class]) type.

Requester: Public comment T21 (p. 108).
Owner:

Emails:	all-1437
Papers:	(none)

21-063 No constraints on constructor parameter. (N0815=95-0215)
21-064 Miscellaneous errors in resize(size_type n) (N0759=95-0159)
21-065 Incorrect return value for insert() (N0759=95-0159)
21-066 Description of remove() is over-specific (N0759=95-0159)
21-067 Traits specializations are over-constrained for `eos()` member (N0815=95-0215)
21-068 What is the proper role of the "Notes" section in Clause 21. (N0815=95-0215)
21-069 Swap complexity underspecified. (N0759=95-0159)
21-070 operator>= described incorrectly (N0759=95-0159)
21-071 Does getline() have the correct semantics? (N0759=95-0159)
21-072 Incorrect use of size_type in third table in section (N0759=95-0159)
21-073 Add overloads to functions that take default character object. (N0759=95-0159)
21-074 Should `basic_string` have a member semantically equivalent to `strlen()` (N0815=95-0215)
21-075 Incomplete specification for assignment operator (N0800=95-0200)
21-076 Inconsistent pattern of arguments in basic_string overloads (N0815=95-0215)
21-077 basic_string not identified as a Sequence. (N0815=95-0215)
21-078 Possible problem with reference counting and strings. (N0815=95-0215)
21-079 Possible problem with `operator<<()` (N0815=95-0215)
21-080 Allow template specialization for `basic_string` and `string_char_traits`?
21-082 Typedef for reverse_iterator is incorrect. (N0899=96-0081)
21-083 Traits member eos() is not forced to return the same value every time. (N0899=96-0081)
21-084 Specialize swap() algorithm for basic_string. (N0899=96-0081)

Issue Number: 21-085

Title:	Awkward argument order for basic_string traits.
Section:	21.1.2 [lib.char.traits.require]
Status:	closed

Description:

Two string_char_traits members have the following signatures:
```
static const char_type*
find(const char_type* s, size_t n, const char_type& a)

static char_type*
assign(char_type* s, size_t n, const char_type& a)
```
The semantics of these members emulate memchr() and memset(). However, the argument order is slightly different. In the interest of consistency, the order of these arguments should be corrected.

Proposed Resolution:

This resolution was rejected and no action taken.

In section 21.1.2 [lib.char.traits.require] change the signatures of find() and assign() as follows:
```
static const char_type*
find(const char_type* s, const char_type& a, size_t n)

static char_type*
assign(char_type* s, const char_type& a, size_t n)
```

Requester:	LWG
Owner:	
Emails:	(none)
Papers:	(none)

21-086 New type added to table (N0899=96-0081)
21-087 Different return values for index operations (N0899=96-0081)
21-088 Slight glitch in return value for find() (N0899=96-0081)
21-089 Should basic_string have a release() member. (N0899=96-0081)
21-091 More specific description for capacity() and reserve() (N0899=96-0081)

Issue Number: 21-092
Title: Incorrect description for traits::find()
Section: 21.1.2 [lib.char.traits.require]
Status: closed
Description:
At the end of the second sentence of the member's description, the description of the range for `i` is incorrectly stated as `[0, n)`.
Resolution:
Close this issue. This problem was corrected in the process of adopting N0854R1 in Santa Cruz.
Requester: Judy Ward (j_ward@decc.enet.dec.com).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-093
Title: Clarify return value for traits::find()
Section: 21.1.2 [lib.char.traits.require]
Status: closed
Description:
The description of the function does not define what should be returned if the character cannot be found

Resolution:
Add the following to the end of the Returns section:
…, `zero otherwise`
Requester: Judy Ward (j_ward@decc.enet.dec.com).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-094
Title: Description for operator>> does not cleanse string
Section: 21.2.1.8.9 [lib.string.io]
Status: closed
Description:
The description of the stream extraction operator states:
The function extracts characters and appends them to *`str`* as if by calling *`str`*`.append(1,` *`c`*`)`.
This incorrectly implies that extracting into a string which already contains data would append the data to the string

Resolution:
In the description for operator>>() in 21.1.1.10.8 [lib.string.io] change:
The function extracts characters and appends them to *`str`* as if by calling *`str`*`.append(1,` *`c`*`)`.

to:

> The string is initially made empty by calling str.erase(). Then the function extracts characters and appends them to *`str`* as if by calling *`str`*`.append(1, `*`c`*`)`.

Requester: Judy Ward (j_ward@decc.enet.dec.com).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-096
Title: Add several headers to basic_string
Section: 21.1 [lib.string.classes]
Status: closed
Description:

The declaration of the basic_string template does not include all headers required.

(The addition of iterator has also been recommended by the German delegation.)

Proposed Resolution:

Resolution was rejected and no action taken.

Add the following headers to the declaration of basic_string:

```
#include <stdexcept>
#include <iterator>
#include <locale>
#include <cwchar>
#include <cwctype>
```

Requester: Judy Ward (j_ward@decc.enet.dec.com).
Owner:
Emails: lib-4691
Papers: (none)

Issue Number: 21-097
Title: Remove default arguments on constructor
Section: 21.1 [lib.string.classes]
Status: closed
Description:

Jack Reeves commented:
“I consider the following constructor signature incorrect:

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

This signature permits a partial string copy to be made by writing:

```
string(str, 5);
```

After some use, I have concluded that such constructs are potentially dangerous. While the constructor is not really a problem, I chose to be consistent in my implementation, and require all substring specifications to explicitly include both the starting position and the length, and never permit the length to be defaulted.

Proposed Resolution:

Resolution was rejected and no action taken.

Replace the above signature with:

```
basic_string(const basic_string& str);
basic_string(const basic_string& str, size_type pos,
             size_type len);
```

Requester: Jack Reeves <76217.2354@CompuServe.COM>
Owner:
Emails: all-1437
Papers: (none)

Issue Number: 21-098
Title: Combine resize() signatures
Section: 21.1.1.6 [lib.string.capacity]
Status: closed
Description:

Jack Reeves commented:
After considerable thought, and some experience, I could find no reason not to combine the two resize() signatures into one as follows:

```
resize(size_type n, charT c = charT());
```

or

```
resize(size_type n, charT c = traits::eos());
```

This is a trivial point, but unless there is a reason to require both signatures, I would recommend they be combined.

Proposed Resolution:

Resolution was rejected and no action taken.

Combine the signatures for resize() into:

```
resize(size_type n, charT c = charT());
```

(This approach is consistent with deque and list.)

Requester: Jack Reeves <76217.2354@CompuServe.COM>
Owner:
Emails: all-1437
Papers: (none)

Issue Number: 21-099
Title: New return type for resize()
Section: 21.2.1.4 [lib.string.capacity]
Status: closed
Description:

Jack Reeves commented:
I changed the return type of resize() from 'void' to 'basic_string&'. I have done this in several other cases and I recommend it be the norm. 'void' should be the return type only when it does not make sense to return ANYTHING else. By returning a reference to itself, resize() can be used in situations where it is desirable to string together functions. I discovered the need for this as a result of my implementation, but I feel it is a valid principal in any case. In my implementation:

str.resize(0).append(s);

is semantically equivalent, but not functionally equivalent to:

str.assign(s);

Since the implementation does reference counting, the latter releases the internal memory of 'str' and grabs that of 's'. On the other hand, the former retains the internal memory of 'str', and does a copy when 'append()' is called. This difference is potentially important to a certain class of users. As currently defined in the DWP, the former would have to be two separate statements. Not a big deal, but a minor annoyance when almost all other functions permit the cascading.

Proposed Resolution:

Resolution was rejected and no action taken.

Change the return type of resize() to basic_string&. Add the following to the description of resize:

Returns: *this

Requester: Jack Reeves <76217.2354@CompuServe.COM>
Owner:
Emails: all-1437
Papers: (none)

Issue Number: 21-100
Title: New return type for reserve()
Section: 21.2.1.4 [lib.string.capacity]
Status: closed
Description:

Jack Reeves commented:
I changed the return type of 'reserve()' from 'void' to 'basic_string&'. The same reasons as (in 21-099), only more so. It is completely reasonable for a function to wish to allocate a string, reserve a certain amount of memory for it, assign an initial value to it, and return the result. With this change, this can all be written as:

```
return string().reserve(80).assign("initial value");
```

This supports the "return value optimization". Without this change, a named temporary would have to be created to support the call to 'reserve()'. As noted above, I recommend this change. As currently implemented, the only functions in my version of basic_string<> which return a 'void' are the two versions of insert which take an iterator and insert a range of characters. The only reason they return 'void' is that the other function which takes an iterator, returns an iterator, but there is no obvious iterator to return from these two functions.

Maintainer’s note: The return type for reserve() was modeled after the return type specified in STL. STL does not require a container to return itself due to the increased requirement

Proposed Resolution:

Resolution was rejected and no action taken.

Change the return type of reserve() to basic_string&. Add the following to the description of resize:

Returns: *this

Requester: Jack Reeves <76217.2354@CompuServe.COM>
Owner:
Emails: all-1437
Papers: (none)

Issue Number: 21-101
Title: Add signatures to mutator members
Section: 21.2.1.6 [lib.string.modifiers]
Status: closed
Description:

Jack Reeves commented:

The second highest priority issue that I would like to bring up is related to the mutator functions which take a single charT argument and a count. These are:

```
       append(size_type cnt, charT c);
       assign(size_type cnt, charT c);
       insert(size_type pos, size_type cnt, charT c);
       replace(size_type pos, size_type n1,size_type n2,charT c);
       replace(iterator i1, iterator i2, size_type n2, charT c);
```

When the string class was originally proposed, these were of the form:

```
       append(charT c, size_type cnt = 1);
       assign(charT c, size_type cnt = 1);
       insert(size_type pos, charT c, size_type cnt = 1);
               etc.
```

I am not sure why the change was made. I presume it has something to do with consistency with the other STL containers. In any case, it has created a situation where the most obvious uses are completely non-intuitive. For example, inserting an escape character in a string should be

```
       str.insert(pos, '^');
```

instead, it has to be written

```
       str.insert(pos, 1, '^');
```

(Likewise, removing the escape character should be

```
       erase(pos);
```

but it isn't - more on this below).

After considerable thought, I could not find any reason why the most obvious case should not be accommodated. Therefore I added the following signatures to my basic_string<> class:

```
       append(charT c);  // implements operator+=(charT)
       assign(charT c);  // implements operator=(charT)
       insert(size_type pos, charT c);
       replace(size_type pos, size_type len, charT c);
       replace(iterator i1, iterator i2, charT c);
```

After using this implementation for several weeks now, I am convinced that this was the right thing to do. For the types of string manipulation that I do, I find that the single character case far, far outnumbers the times when I need to add the count. I also find it is consistent with the other STL containers, which have a separate function which inserts a single element into the container. [My implementation uses a little trick to generate a compile time error if the parameters for the version using the count are written in the wrong order.]

I STRONGLY urge the committee to add these function signatures to basic_string<>.

Proposed Resolution:

Resolution was rejected and no action taken.

Add the following signatures to basic_string

```
       append(charT c);  // implements operator+=(charT)
       assign(charT c);  // implements operator=(charT)
       insert(size_type pos, charT c);
       replace(size_type pos, size_type len, charT c);
       replace(iterator i1, iterator i2, charT c);
```

Requester:	Jack Reeves <76217.2354@CompuServe.COM>
Owner:	
Emails:	all-1437
Papers:	(none)

Issue Number: 21-102

Title:	Revise signatures for erase()
Section:	21.2.1.6 [lib.string::erase]

Status: closed

Description:

Jack Reeves commented:

I consider the signature

```
       erase(size_type pos = 0, size_type len = npos);
```

to be dangerously incorrect. I mentioned this in (L) above -- the problem is that erasing a single character is likely to be a common requirement, but its form is dangerously non-intuitive. If you write

```
       erase(p);
```

to try to erase the single character at position p, in fact you have erased the entire sub-string from p to the end of the string. Erasing a single character is

```
       erase(p, 1);
```

Not only does this seem somewhat backward to the mutator forms which take the
length first, but it contradicts the form which takes an iterator:

```
       erase(it);
```

which can be used to erase a single character.
Furthermore, as I noted in (D) above, I think it is a bad idea to allow the length of a substring to default (this function is the main reason why I think it is a bad idea). Finally, I note that the other STL containers do not have an erase function which defaults both arguments -- instead they have a 'clear()' function. Now, I must admit that this does not make too much sense to me either (why not just have another signature for erase()), but for the sake of both safety and consistency, I recommend that the single erase signature described above be replaced by the following three signatures:

```
       clear(); // erases entire string
       erase(size_type pos); // erases a single character
       erase(size_type pos, size_type len); // erases substring
```

Proposed Resolution:

Resolution was rejected and no action taken.

Revise the signatures for erase() to be::

```
       erase(size_type pos); // erases a single character
       erase(size_type pos, size_type len); // erases substring
```

Add the signature:

```
       void clear()
```

Effects: erase(0, npos);

Requester: Jack Reeves <76217.2354@CompuServe.COM>

Owner:

Emails: all-1437

Papers: (none)

Issue Number: 21-0103

Title: Change signature for substr()

Section: 21.2.1.7.7 [lib.string::substr]

Status: closed

Description:

Jack Reeves commented:

In keeping with my "no default substring length" rule, I changed the signature of:

```
       substr(size_type pos = 0, size_type len = npos);
```

to

```
       substr(size_type pos, size_type len);
```

I found there was no use whatsoever for writing

```
       str.substr();
```

This is just equivalent to 'str' itself, or more correctly to:

```
       string(str);
```

and this is the preferred form. I recommend that the committee do likewise.

Proposed Resolution:

Resolution was rejected and no action taken.

Change the signature of substr() to:

```
      substr(size_type pos, size_type len);
```

Requester: Jack Reeves <76217.2354@CompuServe.COM>

Owner:

Emails: all-1437

Papers: (none)

Issue Number: 21-104

Title: Enhance signatures for compare()

Section: 21.2.1.7.8 [lib.string::compare]

Status: closed

Description:

Jack Reeves commented:

"There are currently (Jan '96 DWP) 5 compare() signatures defined. It appears that the intent is to have a full range of capability for comparing the string, or a substring thereof, to either another string, a substring, a charT* (implied length), or a charT* (explicit length). This implies 8 signatures. I was inclined to eliminate two of these, but unfortunately, one of the ones I would have eliminated was one already defined in the Jan '96 DWP, so I went ahead and defined all 8 in my implementation. This resulted in the following signatures:

```
      int compare(const basic_string& str) const;
      int compare(size_type pos, size_type len,
                  const basic_string& str) const;

      int compare(const basic_string& str,
                  size_type pos, size_type len) const;
      int compare(size_type pos, size_type len,
                   const basic_string& str,
                   size_type pos, size_type len) const;

      int compare(const charT* s) const;
      int compare(size_type pos, size_type len,
                  const charT* s) const;

      int compare(const charT* s, size_type n) const;
      int compare(size_type pos, size_type len,
                  const charT* s, size_type n) const;
```

Signatures # 3, 6, and 7 are new. The default parameter of 'npos' for the size_type argument of signature #8 (#5 in the DWP) was clearly incorrect (it would always result in a "length_error" exception) and I removed it. I recommend that the committee add the extra signatures.

Proposed Resolution:

Resolution was rejected and no action taken.

Revise the signatures to compare() as described above

Requester: Jack Reeves <76217.2354@CompuServe.COM>

Owner:

Emails: all-1437

Papers: (none)

Issue Number: 21-105

Title:	Add “throws” for constructor with charT* parameter
Section:	21.2.1.2 [lib.string.cons]
Status:	closed
Description:	

Jack Reeves commented:
The basic_string<> constructors which take an argument of type charT* state that the pointer must not be null. By extension, all functions which take an argument of type charT* have the same requirement since their behavior is always specified in terms of the constructor taking a similar argument. In my implementation, I specified that any function taking a charT* argument would throw an "invalid_argument" exception if the parameter was null, except that those functions which also take a 'size_t' argument to specify the length of the string would not check the pointer (and hence not throw an exception) if the size was specified to be 0. The committee may not wish to specify the behavior of basic_string<> to this level, but if not I suggest that they make it explicit by stating that behavior is undefined if a null charT* is passed to a function. Otherwise, I recommend my approach.

Maintainer’s note: [intro.compliance] states that if a requirement is not met by a program, the behavior of that program is undefined.

Proposed Resolution:

Resolution was rejected and no action taken.

Add the following to the description of basic_string(charT* str)
Throws: invalid_argument if str is null

Requester:	Jack Reeves <76217.2354@CompuServe.COM>
Owner:	
Emails:	all-1437
Papers:	(none)

Issue Number: 21-106

Title:	Editorial box 51: traits recast in table
Section:	21.1.2 [lib.char.traits.require]
Status:	closed
Description:	

Box 51 contains the text: “ Change: The definitions of semantics of Traits members have been recast in tabular form, consistent with requirements tables elsewhere in the draft.”

Resolution:

Remove the box.

Requester:	Rick Wilhelm <rwilhelm@str.com>
Owner:	
Emails:	(none)
Papers:	(none)

Issue Number: 21-107

Title:	Editorial box 52: missing footnote
Section:	21.1.2 [lib.char.traits.typedefs]
Status:	closed
Description:	

Box 52 contains the text: “In N0854R1, there was a reference to footnote 221. In the January 1996 DWP, footnote 221 is in the algorithms clause and is clearly irrelevant. What should the footnote be?”

This actually refers to footnote 228 in 27.1.2

Resolution:

Add the following footnote:

It is usually a synonym for one if the signed basic integral types whose representation at least as many bits as type long.

Requester: Rick Wilhelm <rwilhelm@str.com>
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-108
Title: Editorial box 53: missing footnote
Section: 21.1.2 [lib.char.traits.typedefs]
Status: closed
Description:

Box 53 contains the text: “In N0854R1 there was a reference to footnote 222. In the January 1996 wp, footnote 222 is in the algorithms clause, and is clearly irrelevant. What should the footnote text be?”

This actually refers to footnote 229 in 27.1.2

Resolution:

Add the following footnote:

An implementation may use the same type for both OFF_T and POS_T.

Requester: Rick Wilhelm <rwilhelm@str.com>
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-109
Title: Editorial box 54: fixed specilization declarations
Section: 21.1.2 [lib.char.traits.specilizations]
Status: closed
Description:

Box 54 contains the text: “The proposal in N0845R1 did not have the template<> on the above declarations, making them ill-formed. Adding it makes them declarations of explicit specializations.”

Resolution:

Remove the box.

Requester: Rick Wilhelm <rwilhelm@str.com>
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-110
Title: Semantics of type requirements remain in Clause 27
Section: 21.1.5.1 [lib.char.traits.specilizations.char]
Status: closed
Description:

The descriptions of the semantics of character traits types requirements remains in Clause 27. They are currently boxed as being ready for movement.

Resolution:

Editorial.

Requester:	Rick Wilhelm <rwilhelm@str.com>
Owner:	
Emails:	(none)
Papers:	(none)

Issue Number:	**21-112**
Title:	Box 55: unreferenced header deleted
Section:	21.3 [lib.c.strings]
Status:	closed
Description:	Box 55 contains the text: "This list formerly mentioned <ciso646> but there was no corresponding description, so I removed it. --ARK 5/96"
Resolution:	Add <ciso646> to the table of C++ headers for C Library Facilities in 17.3.1.2 [lib.headers]. Add a footnote to the same table: "The C++ version of <ciso646> is effectively empty. It is included for the sake of completeness and for C compatibility."
Requester:	Rick Wilhelm <rwilhelm@str.com>
Owner:	
Emails:	(none)
Papers:	(none)