Doc. No.: X3J16/96-0081
WG21/ N0899
Date: March 28, 1996
Project: Programming Language C++
Reply To: Richard K. Wilhelm
Strategic Technology Resources
rwilhelm@str.com

Clause 21 (Strings Library) Issues List
Revision 15

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-Monetery 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.

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: 21-062
Title: Missing explanation of requirements on charT.
Section: 21.1.1.3 [lib.basic.string]
Status: active
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).

Proposed Resolution:

Add the following text after paragraph 1 in 21.1.1.3 [lib.basic.string]

> A "char-like type" does not need to be constructed or destroyed. A string's allocator shall have the same definitions for the types size_type, difference_type, pointer, const_pointer, reference, const_reference as class allocator::types<charT>.

In private email, P.J. Plauger wrote:
"In reviewing my code, I realize that I overstated the case here. It is more accurate to say that the basic_string class presumes that charT has a default constructor (and a destructor), which the class uses to construct (and destroy) all elements of the controlled sequence. Whenever the class is asked to copy out elements, as with the copy member function, it assumes that it need only assign to previously constructed elements.

"A better design of string_char_traits would probably include uninitialized_copy and uninitialized_fill members, but I feel it's way too late to propose such additions."

Requester: Public comment T21 (p. 108).
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-085
Title: Awkward argument order for basic_string traits.
Section: 21.1.1.2 [lib.string.char.traits.members]
Status: active
Description:

Two string_char_traits members have the following signatures:

```
static const char_type*
find(const char_type* s, int 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.

Additionally, change the type of the `find()` member's '`n`' argument to `size_t`

Proposed Resolution:

In section 21.1.1.2 [lib.string.char.traits.members] 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)

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."

Proposed Resolution:

None yet.

Requester: John Mulhern (jmulhern@empros.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)

Issue Number: 21-014
Title: Argument order for copy() is incorrect.
Section: 21.1.1.8.7 [lib.string::copy]
Status: closed
Description:

In private email, John Dlugosz wrote:
"In copy() the arguments are in a different order than on other functions. I suppose this was to provide for a default on pos. However, if someone does specify both he will be likely to get them backwards and the compiler will not catch this. I feel it is a point of usability that is not worth the default argument. Provide two forms of copy() instead:

```
copy (dest, pos, len);
copy (dest,len);
```

Note: The current interface to copy is:

```
size_type copy(charT* s, size_type n, size_type pos=0);
```

Proposed Resolution:

Provide two forms of copy():

```
size_type copy(charT* s, size_type pos, size_type n);
```

This function differs from the current copy only in the order of its last two arguments and the lack of a default argument.

```
size_type copy(charT* s, size_type n);
```

Returns:

```
copy(s, 0, n);
```

Resolution:

Closed with no action taken. Insufficient support for the change.

Requester: John Dlugosz: jdlugosz@objectspace.com
Owner:
Emails: (none)
Papers: (none)

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)

Issue Number: 21-059
Title: String traits have no relationship to iostream traits.
Section: 21.1.1.1 [lib.string.char.traits]
Status: closed
Description:

I would like to propose (whether officially or not) to modify the current CD:

```
template <class charT> struct ios_traits {};
```

to

```
template <class charT> struct ios_traits :
    public string_char_traits<charT> {};
```

in order to integrate the closely related traits, 'ios_traits' and 'string_char_traits'.

We can expect the integration of the common features, such as 'eq', 'eos', 'length', and 'copy' which is now inappropriately separated with no explicit reasons.

In lib-3832, Nathan Myers wrote:
"I have been careful to avoid getting too involved with Clause 21, thus far, because I have been quite busy with other chapters. However, it would be my recommendation to eliminate most of the string character traits: eq(), ne(), lt(), assign(), char_in(), char_out(), and is_del(). Also, I would either add a few "speed-up functions" needed to efficently implement strings without specialization, such as a move() member, or eliminate them all, and let the implementation specialize speedups for types known to it."

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

See also issue 21-018.

Discussion at the Tokyo meeting found merit in the idea of integrating string_char_traits and ios_char_traits. However, no action was taken pending further investigation.

A cursory review of string and iostream character traits shows that the signatures are basically compatible except for the string_char_traits::eq() and ios_char_traits::eq_char_type().

Resolution:

Some traits issues are addressed in issue 21-002, 21-018, 21-024, and 21-060. This issue remains open as a discussion of the possible integration of iostream traits and string character traits.

Norihiro Kumakai's paper N0854=95-0036 contains the resolution for this issue.

The Committee accepted an amended version of this paper at the Santa Cruz meeting. The details were handled by the iostreams subgroup.

Requester: Norihiro Kumagai: kuma@slab.tnr.sharp.co.jp.
See also Public Comment T21 (p. 108).

Owner:
Emails: lib-3832, lib-4351
Papers: N0810R1=95-0210R1, N0854=95-0036

21-060 string_char_traits::ne not needed (N0815=95-0215)
21-061 Missing explanation of traits specialization (N0815=95-0215)
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)

Issue Number: 21-080
Title: Allow template specialization for `basic_string` and `string_char_traits`?
Section: 21.1.1.3 [lib.template.string]
Status: closed
Description:

Discussion of a general library issue in Tokyo arrived at the conclusion that template specialization would require the templates to be placed in the std namespace. Since there is currently a general prohibition on extending the std namespace [lib.reserved.names] "unless otherwise specified", `basic_string` and `string_char_traits` must be explicitly exempted from this prohibition if they can be specified.

Resolution:

With the acceptance of the resolution for library issue 17-005, any template classes in the Standard Library can be specialized, provided they fulfill the templates requirements.

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

Issue Number: 21-081
Title: Portions of Clause 21 are redundant with portions of Clause 23.
Section: 21.1.1.3 [lib.template.string]
Status: closed
Description:

Since basic_string is a Sequence (as defined in Clause 23) portions of the description for basic_string are redundant. In particular, the parts that describe members which fulfill Sequence requirements.

In Tokyo, the issue of clarity and maintainability was raised. If portions of the basic_string description are removed, the clause becomes easier to maintain because it can rely on Clause 23 for all Sequence requirements. However, this removal may impact the clarity of Clause 21.

Resolution:

Retain the current organization. Given the committee's current deadline constraints, such a large reorganization would introduce a great deal of instability into both clauses.

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

Issue Number: 21-082
Title: Typedef for reverse_iterator is incorrect.

Section: 21.1.1.3 [lib.template.string]
Status: closed
Description:

In 24.3.1.3 [lib.reverse.iterator], the class reverse_iterator has the following template arguments:

```
template <class RandomAccessIterator, class T,
          class Reference = T&, class Pointer = T*,
          class Distance = ptrdiff_t>
       class reverse_iterator
```

The fifth template argument was added recently. The reverse_iterator typedef in basic_string does not reflect this change.

Resolution:

Change the typedefs for for basic_string's reverse_iterator and const_reverse_iterator to:

```
typedef
reverse_iterator<iterator, value_type,
       reference, pointer, difference_type> reverse_iterator;
typedef
reverse_iterator<const_iterator, value_type,
       const_reference, const_pointer, difference_type>
       const_reverse_iterator;
```

Requester: Larry Podmolik (podmolik@str.com)
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-083
Title: Traits member eos() is not forced to return the same value every time.
Section: 21.1.1.2 [lib.string.char.traits.members]
Status: closed
Description:

With the resolution of issue 21-067, the traits member eos() is not required to return the value char_type(). However, this desirable freedom might be construed to allow an implementation to return a different value for eos() on subsequent calls.

Proposed Resolution:

Add the following text to the portion of 21.1.1.2 [lib.string.char.traits.members] which describes eos():

> Subsequent calls to this member will return an equivalent object.

Resolution:

Take no action. With the integration of string_char_traits and ios_char_traits, this issue is addressed elsewhere.

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

Issue Number: 21-084
Title: Specialize swap() algorithm for basic_string.
Section: 21.1.1.10.8 [lib.string.special]
Status: closed
Description:

From Box 1 in Clause 23: "Change: Issue 23-031 in N0781R2=95-0181R2, approved in Tokyo, approved the addition of swap specializations for all containers except basic_string. It only mentioned the problem in this class. In the interest of stability and correctness, it has been added and an issue opened to formalize the change."

Resolution:

No change. Remove the box from section 21.1.1.10.8 [lib.string.special]

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

Issue Number: 21-086
Title: New type added to table
Section: 21.2 [lib.c.strings]
Status: closed
Description:

An editorial box has the content: "Change: added wchar_t to the above table because wcsmemchr uses it."

Resolution:

No change. The editors change is correct. Remove the editorial box.

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

Issue Number: 21-087
Title: Different return values for index operations
Section: 21.1.1.7 [lib.string.access]
Status: closed
Description:

Although the following accessors are semantically equivalent, the return values are different:

```
charT   operator[](size_type pos) const;
const_reference at(size_type pos) const;
```

Resolution:

Change the return value of the operator[]member as follows:

```
const_reference operator[](size_type pos) const;
```

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

Issue Number: 21-088
Title: Slight glitch in return value for find()
Section: 21.1.1.9.1 [lib.string::find]
Status: closed
Description:

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

Resolution:

Change the returns description of:
`size_type find(const charT* s, size_type pos, size_type n) const;`
as follows:
Returns: `find(basic_string<charT,traits,Allocator>( s, n), pos).`

Requester: P.J. Plauger.
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-089
Title: Should basic_string have a release() member.
Section: 21.1.1.6 [lib.string.capacity]
Status: closed
Description:

Jack Reeves wrote:

5. I have already suggested the following, but will suggest it again, as I consider it important. Class basic_string has a reserve() function, but no release() function. It really needs a release() (or shrink_to_fit()) function. Partly this is just good design (pardon my arrogance) -- the reserve() function is used to indicated an anticipated increase in the size of the string, and the release() function is its opposite and is used to indicate that no more changes are anticipated and the excess reserved memory can be given back to the system. Partly, reserve() and release() can be used with a special allocator that deals with relocatable memory such as the original Macintosh or Windows -- reserve() would do a lock and release() could unlock (as well as shrink). I note two aspects about release(). The first is that it could interact somewhat poorly with c_str().

```
  void f(string s) {
    s.release();                  // shrink to fit
    cout << s.c_str() << endl;    // trying to re-alloc the string
                                  // to size()+1 might cause it
                                  // to have quite a bit of slop
  }
```

I would consider this annoying, but something that could be lived with. However, an alternative provides a solution to my desire for a release() function and this problem -- redefine the semantics of reserve() to allow it to function as a release() function also. Thusly -

after reserve(size_type n) ::=
  if (n < size()) then capacity is set to size()
  otherwise capacity() will equal n.

Frankly, this would be my preference. Thus the example above would become

```
     void f(string s) {
          s.reserve(s.size()+1);
          cout << s.c_str() << endl;
     }
```

with the assurance that the actual memory used is the minimum necessary. The reserve() function could be prototyped as
`void reserve(size_type res_arg = 0)`
where the default argument would allow the use of s.reserve() to be semantically equivalent to shrink-to-fit.

Proposed Resolution:

Changing the semantics of reserve() would both overconstrain implemtations and break with existing practice. If this change is to be made, it should be done with a new member.

Add the member:
`void release(size_type res_arg = 0)`
as follows:

> The member function release() is a directive that attempts to force an upper bound on a string object's storage.
> Effects: if size() < res_arg < capacity(), then reallocation happens and, subsequently capacity() == res_arg. Otherwise, there is no effect. Reallocation invalidates all the references, pointers, and iterators referring to the elements in the sequence. It is guaranteed that no reallocation takes place during the insertions that happen after release() takes place until the time when the size of the string reaches the size specified by release().
> Complexity: It does not change the size of the sequence and takes at most linear time in the size of the sequence.

Resolution:

No change. Discussion in the LWG achieved the consensus that such a member would expose too much information about memory management and thus overconstrain the implementation.

Requester: Jack Reeves. (jack@fx.com)
Owner:
Emails: (none)
Papers: (none)

Issue Number: 21-091
Title: More specific description for capacity() and reserve()
Section: 21.1.1.6 [lib.string.capacity]
Status: closed
Description:

From a comment by the German delegation:
In the C++ library exist two "container" which have the member functions capacity() and reserve(), namely string and vector. Their meaning is unclear described or even not sensibly described:

- capacity() returns "the size of allocated storage"

- reserve() enlarges capacity if necessary and ensures that:

> "It is guaranteed that no reallocation takes place during the insertion that happend after reserve() til the time when the size of the string/vector reaches the size specified by reserve".

The meaning seems not to be quite clear:

- What does the return value of capacity() and the parameter for reserve() mean ? Is it the size of the storage or the logical number of elements/chars ?

- If it is the size of the storage, does it include eos for strings ?

- What does "reaches the size" mean? This seems not to be exact, because it has to become greater than the specified size for reserve().

Resolution:

Change 21.1.1.6 [lib.string.capacity] as follows:

Add a default = 0 to the argument of reserve().

Delete all the text from "if reallocation happens …" onwards in the reserve() Effects paragraph.

Delete the text from "It is guaranteed…" onwards in the reserve() Notes paragraph.

Requester: Nicolai Josuttis (nico@bredex.de) .
Owner:
Emails: (none)
Papers: lib-4496