Document number:	**N2909=09-0099**
Date:	2009-06-12
Project:	Programming Language C++
Reference:	N2857=09-0047
Reply to:	**Alan Talbot**
	cpp@alantalbot.com

Specifying the complexity of size()

Abstract

The current language describing size() in the various places it appears is either silent or (deliberately) fuzzy on the subject of complexity. This allows implementations a degree of flexibility, but it is confusing to programmers, and makes it hard to use containers in situations where testing their size is a frequent operation. This paper proposes wording changes to specify the complexity of size() everywhere it appears. It addresses National Body comment US 2 (address all open issues) with regard to LWG issue 632 (Time complexity of size() for std::set).

List

List is a special case because of the problem with splice. The range version of splice() cannot be O(1) if size() is also O(1). I believe that this tradeoff is best taken by specifying size() to be O(1). Splicing is useful in some situations, but size() is much more likely to be heavily used, and the average programmer is more likely to expect size() to take constant time than splice().

An even better argument is that splicing is most important when the contained element types are expensive or impossible to copy. The biggest advantage of splice is that it does not copy elements. Traversing a range is O(n) but is very fast (compared to allocating and copy-constructing new objects) so the overall performance of splice() is likely to be acceptable even if the implementation does a count to maintain the cached size.

String

String (basic_string) is also of possible concern, but I believe that most or all current implementations store the size, so I recommend that basic_string::size() be specified to be O(1).

Other size() functions

There are several other places where size() appears. With one exception (where it's pretty obvious that it's a compile-time constant) I have provided language specifying the size() function's complexity.

Proposed Wording

Highlighted text indicates places where I am looking for guidance.

Note to the Editor: In all cases I have assumed that the paragraph numbering will be updated below the insertions as needed.

18.9.2 Initializer list access [support.initlist.access]

`size_t size() const;`

5 *Returns:* the number of elements in the array.

6 *Throws:* nothing.

7 *Complexity:* constant time.

20.3.6.2 bitset members [bitset.members]

Since bitset is parameterized on a fixed size N, and size() is specified to return N, I did not think it was necessary to specify that size() has compile-time complexity. I can add this if others think it is needed.

21.4.4 basic_string capacity [string.capacity]

`size_type size() const;`

1 *Returns:* a count of the number of char-like objects currently in the string.

2 *Complexity:* constant time.

`size_type length() const;`

3 *Returns:* size().

`size_type max_size() const;`

4 *Returns:* The ~~maximum~~ size of the largest possible string.

5 *Complexity:* constant time.

~~4 Remark: See Container requirements table (23.2).~~

I removed ¶ 4 (the reference to 23.2) because it did not seem to add anything other than the complexity requirement. If someone else thinks there are other requirements implied here, please let me know.

23.2.1 General container requirements [container.requirements.general]

In ¶ 4, changes to Table 80:

a.size()	size_type	a.end() – a.begin()	~~(Note A)~~constant
a.max_size()	size_type	Size() of the largest possible container	~~(Note A)~~constant

Note to the Editor: Typo in ¶ 5:

5 The member function size() returns the number of elements in the container. Its semantics ~~is~~are defined by the rules of constructors, inserts, and erases.

26.5.7.1 Class seed_seq [rand.util.seedseq]

`size_t size() const;`

7 *Returns:* The number of 32-bit units that would be ~~be~~ returned by a call to param().

8 *Complexity:* constant time.

26.6.2.7 valarray member functions [valarray.members]

```
size_t size() const;
```

4 ~~This function returns~~*Returns:* ~~t~~The number of elements in the array.

5 *Complexity:* constant time.

Note to the Editor: There are a number of other functions in this section that do not conform to the normal function definition style.

26.6.4.2 slice access functions [slice.access]

```
size_t start() const;
size_t size() const;
size_t stride() const;
```

1 ~~These functions return~~*Returns:* ~~t~~The start, length, or stride specified by a slice object.

2 *Complexity:* constant time.

I am not sure that the complexity of these is constant. I assume so from the semantics.

26.6.6.2 gslice access functions [gslice.access]

```
size_t start() const;
valarray<size_t> size() const;
valarray<size_t> stride() const;
```

1 ~~These access functions return~~*Returns:* ~~t~~The representation of the start, lengths, or strides specified for the gslice.

2 *Complexity:* linear in the number of strides.

Again, I am assuming linear complexity from the semantics.

Note to the Editor: The language used in ¶ 1 is not quite the same as used in § 26.6.4.2 ¶ 1 above.

28.11.2 match_results size [re.results.size]

The complexity of match_results::size() will now be specified to be constant because match_results satisfies the requirements of a constant sequence container. If this is a problem for match_results, some additional language is needed here.

Index – p. 1326

Note to the Editor: Both size and size() appear in the index. This seems like a typo.

Acknowledgements

Thanks to Howard Hinnant for thoughts on list::size() and size() complexity in general. Thanks to Walter Brown for random help.