Improving the Return Value of Erase-Like Algorithms II: Free `erase`/`erase_if`

Document #:	P1115R1
Date:	September 25, 2019
Project:	Programming Language C++ Library Working Group

Reply-to: Marc Mutz <marc.mutz@kdab.com>

Abstract

We propose to change the return type of `erase()` and `erase_if()` free functions from `void` to *`<container>`*`::size_type`, returning the number of elements removed. This restores consistency with long-established API, such as `map/set::erase(key_type)`, as well as the recent changes to `forward_/list::remove()`.

Contents

0 Change History

This is a spin-off and revision of P0646R0 at the request of LWG in Rapperswil to work around the problem of LFv3 not having opened shop in Rapperswil, yet.

0.1 Changes from P1115R0

1. Fixed an IS section reference in Section 3.
2. Rebased onto [N4830].

0.2 Changes from P0646R0

1. Removed changes to the IS draft, as these continued as P0646R1 (which has since been adopted in Rapperswil).
2. Changed the return type from `size_t` to `<container>::size_type`, as requestd by LEWG in Toronto.
3. Rebased on IS draft, as the target of this proposal has since been merged into it from the LFv2 TS.
4. Added feature test macro.

1 Motivation and Scope

This section is copied from P0646R1, so readers familiar with that paper can skip these paragraphs.

1.1 [[nodiscard]] Useful Information

Alexander Stepanov, in his A9 courses[A9], teaches us not to throw away useful information, but instead return it from the algorithm.

With that in mind, look at the following example:

```
std::forward_list<std::shared_ptr<T>> fl = ...;
erase(fl, nullptr);
```

Did `erase()` erase anything? We don't know. The only way we *can* learn whether the algorithm removed something is to check the size of the list before and after the algorithm run. For most containers, that is a valid option, and fast. All `size()` methods of STL containers are $O(1)$ these days.

But `std::forward_list` has no `size()`...

We therefore propose to make the algorithms return the number of removed elements. While it is only really necessary for `forward_list`, we believe that consistency here is more important than minimalism.

Returning the number of elements also enables convenient one-line checks:

```
if (erase(fl, nullptr)) {
    // erased some
}
```

1.2 Consistency

In Rapperswil, the committee accepted P0646R1, which changed the `list` and `forward_list` member algorithms `remove`/`_if` and `unique` to return the number of elements erased. This paper applies the same logic to the non-member versions of these algorithms.

We note that the associative containers have returned the number of erased elements from their `erase(key_type)` member functions since at least [SGI STL]. This proposal therefore also restores lost consistency with existing practice.

2 Impact on the Standard

Minimal. We propose to change the return value of library functions from `void` to `size_type`. Existing users of the LFv2 versions expecting no return value can continue to ignore it. In particular, this is one of the changes explicitly mentioned in [P0921R2].

Strictly speaking, the change is source-incompatible: Existing code which assumes that the algorithms return `void` might fail to compile. This can e.g. come up in situations where the C++ user explicitly specialized these algorithms. However, all such code will so far have used the LFv2 versions of these algorithms, which are in a different namespace.

For the same reason, there is no binary-compatibility issue here: the algorithms in LFv2 were specified in namespace `std::experimental`, while the changed algorithms will be in `std` directly.

3 Proposed Wording

The following changes are relative to [N4830]:

- In [**support.limits.general**], Table 36, adjust the listed **Value** of "`__cpp_lib_erase_if`" to match the date of application of this paper to the IS draft.
- In each of [**string.syn**], [**string.erasure**], [**deque.syn**], [**forward.list.syn**], [**list.syn**], [**vector.syn**], [**deque.erasure**], [**forward.list.erasure**], [**list.erasure**], [**vector.erasure**], [**associative.map.syn**], [**associative.set.syn**], [**unord.map.syn**], [**unord.set.syn**], [**map.erasure**], [**multimap.erasure**], [**set.erasure**], [**multiset.erasure**],

[**unord.map.erasure**], [**unord.multimap.erasure**],
[**unord.set.erasure**], [**unord.multiset.erasure**]:

For each `erase(`*`<container>`*`& c, ...)` and `erase_if(`*`<container>`*`& c, ...)` function, change the return type from `void` to `typename` *`<container>`*`::size_type`.

- In each of [**string.erasure**], [**deque.erasure**], [**vector.erasure**], change paragraphs 1 as follows:

```
- Effects: Equivalent to: c.erase(remove(c.begin(), c.end(), pred), c.end());
+ Effects: Equivalent to:
+     auto it = remove(c.begin(), c.end(), pred);
+     auto r = distance(it, c.end());
+     c.erase(it, c.end());
+     return r;
```

- In each of [**string.erasure**], [**deque.erasure**], [**vector.erasure**], change paragraphs 2 as follows:

```
- Effects: Equivalent to: c.erase(remove_if(c.begin(), c.end(), pred), c.end());
+ Effects: Equivalent to:
+     auto it = remove_if(c.begin(), c.end(), pred);
+     auto r = distance(it, c.end());
+     c.erase(it, c.end());
+     return r;
```

- In each of [**forward.list.erasure**], [**list.erasure**], in paragraphs 1 and 2, add "`return` " between "Equivalent to:" and the start of the code.

- In each of [**map.erasure**], [**multimap.erasure**], [**set.erasure**], [**multiset.erasure**], [**unord.map.erasure**], [**unord.multimap.erasure**], [**unord.map.erasure**], [**unord.multiset.erasure**]:

  Change paragraphs 1 as indicated:

```
+ typename <container>::size_type res = 0;
  for (auto i = c.begin(), last = c.end(); i != last; ) {
    if (pred(*i)) {
      i = c.erase(i);
+     ++res;
    } else {
      ++i;
    }
  }
+ return res;
```

  where *`<container>`* is as follows:

  – in [**map.erasure**]: `map<Key, T, Compare, Allocator>`

  – in [**multimap.erasure**]: `multimap<Key, T, Compare, Allocator>`

  – in [**set.erasure**]: `set<Key, Compare, Allocator>`

  – in [**multiset.erasure**]: `multiset<Key, Compare, Allocator>`

– in [**unord.map.erasure**]: `unordered_map<K, T, H, P, A>`

– in [**unord.multimap.erasure**]: `unordered_multimap<K, T, H, P, A>`

– in [**unord.set.erasure**]: `unordered_set<K, T, H, P, A>`

– in [**unord.multiset.erasure**]: `unordered_multiset<K, T, H, P, A>`

3.1 Feature Test Macro

No new macro is necessary.

4 Design Decisions

4.1 `size_t` vs. `size_type`

Should we return *`<container>`*`::size_type` or `std::size_t` from these functions? P0646R0 chose `size_t`, for brevity, but LEWG in Toronto favoured `size_type`, so this is what's proposed now.

4.2 Performance Considerations

Please refer to P0646R0 for a detailed analysis. TL;DR: We believe that returning the number of elements removed does not pessimise callers that don't need it.

5 Acknowledgements

We thank the reviewers of draft versions of the original proposal and the participants of the associated discussion on `std-proposals@isocpp.org` and LWG in Rapperswil for their input: Sean Parent, Arthur O'Dwyer, Nicol Bolas, Ville Voutilainen, Casey Carter, Milian Wolff, André Somers, Jonathan Wakely, Walter E. Brown. All remaining errors are ours.

References

[A9] Alexander Stepanov *et al.*
Four Algorithmic Journeys / Efficient Programming With Components / Programming Conversations
https://www.youtube.com/user/A9Videos/playlists?view=1

[SGI STL] Alexander Stepanov *et al.*
Associative Container
in: *Standard Template Library Programmer's Guide*
https://www.sgi.com/tech/stl/AssociativeContainer.html (accessed 2017-06-01)

[N4830] Richard Smith (editor)
Working Draft: Standard for Programming Language C++
http://wg21.link/N4830

[P0921R2] Titus Winters
Standard Library Compatibility
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0921r2.pdf