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 .................................................. 1
  0.1 Changes from P0646R0 ........................................ 1

1 Motivation and Scope .............................................. 2
  1.1 [[[nodiscard]]] Useful Information .......................... 2
  1.2 Consistency .................................................. 2

2 Impact on the Standard .......................................... 2

3 Proposed Wording ................................................ 3
  3.1 Feature Test Macro .......................................... 4

4 Design Decisions ................................................ 4
  4.1 `size_t` vs. `size_type` .................................... 4
  4.2 Performance Considerations ................................. 4

5 Acknowledgements ................................................. 5
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 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 requested 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:

```cpp
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:

```cpp
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 [N4810]:

- 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.erase], [deque.syn], [forward.list.syn], [list.syn], [vector.syn], [deque erase], [forward.list.erase], [list.erase], [vector.erase], [associative.map.syn], [associative.set.syn], [unord.map.syn], [unord.set.syn], [map.erase], [multimap.erase], [set.erase], [multiset.erase], [unord.map.erase], [unord.multimap.erase], [unord.map.erase]:

  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.erase], [deque.erase], [vector.erase], change paragraphs 1 as follows:

  - Effects: Equivalent to: c.erase(remove(c.begin(), c.end(), pred), c.end());
  + Effects: Equivalent to:
In each of [string.erase], [deque.erase], [vector.erase], 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.erase], [list.erase], in paragraphs 1 and 2, add “return” between “Equivalent to:” and the start of the code.

In each of [map.erase], [multimap.erase], [set.erase], [multiset.erase], [unord.map.erase], [unord.multimap.erase], [unord.map.erase], [unord.multiset.erase]:

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.erase]: map<Key, T, Compare, Allocator>
- in [multimap.erase]: multimap<Key, T, Compare, Allocator>
- in [set.erase]: set<Key, Compare, Allocator>
- in [multiset.erase]: multiset<Key, Compare, Allocator>
- in [unord.map.erase]: unordered_map<K, T, H, P, A>
- in [unord.multimap.erase]: unordered_multimap<K, T, H, P, A>
- in [unord.set.erase]: unordered_set<K, T, H, P, A>
- in [unord.multiset.erase]: 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 pessimises 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. All remaining errors are ours.

References

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

[N4810] Richard Smith (editor)
Working Draft: Standard for Programming Language C++
http://wg21.link/N4810

[P0921R2] Titus Winters
Standard Library Compatibility