Remove `basic_string::reserve()` From C++26

Document #:	P2870R0
Date:	2023-05-15
Project:	Programming Language C++
Audience:	Library Evolution Incubator
Revises:	N/A

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net>

Contents

1 Abstract

The `basic_string::reserve()` function overload taking no arguments was deprecated for C++20 as a poor substitute for `basic_string::shrink_to_fit`. This paper proposes removing that overload from the C++ Standard Library.

2 Revision history

2.1 R0: Varna 2023

Initial draft of the paper.

3 Introduction

At the start of the C++23 cycle, [P2139R2] tried to review each deprecated feature of C++ to see which we would benefit from actively removing and which might now be better undeprecated. Consolidating all this analysis into one place was intended to ease the (L)EWG review process but in return gave the author so much feedback that the next revision of the paper was not completed.

For the C++26 cycle, a much shorter paper, [P2863R0], will track the overall analysis, but for features that the author wants to actively progress, a distinct paper will decouple progress from the larger paper so that the delays on a single feature do not hold up progress on all.

This paper takes up the deprecated `basic_string::reserve()` function overload, D.25 [depr.string.capacity].

4 Analysis

The `basic_string::reserve()` function taking no arguments was deprecated for C++20 by the paper [P0966R1]. This deprecation was a consequence of cleaning up the behavior of the `reserve` function to no longer optionally reallocate on a request to shrink. The original C++98 specification for `basic_string` supplied a default argument of 0 for `reserve`, turning a call to `reserve()` into a non-binding `shrink_to_fit` request. Note that `shrink_to_fit` was added in C++11 to better support this use case. With the removal of the potentially reallocating behavior, `reserve()` is now a redundant function overload that is guaranteed to do nothing. Hence it was deprecated in C++20, with a view to removing it entirely in a later standard to eliminate one more legacy source of confusion from the standard.

4.1 C++23 Review {#analysis.C++23}

At the LEWG telecon on 2020/07/13, there was general agreement that this member is a holdover from another time, whose replacement has been in place for some time. There was consensus to remove this member from C++23, assuming the subsequent research does not reveal major concerns before the main LEWG review that is to follow.

4.2 Deprecation Experience

The following program was tested on Godbolt compiler explorer to determine whether current library implementations report depration warnings in their C++20 build mode, and if so, from which release:

```
#include <string>

int main() {
    std::string s;
    s.reserve();    // Should be deprated
}
```

- libc++ 12.0
- libstdc++ 11.1
- Microsoft No warning

5 C++26 Recommendation

While there is no pressing need to remove this member, the sentiment was for removal in the previous standard, so three years later we make the same (weak) recommendation to remove from C++26.

6 Wording

All wording is relative to [N4944], the latest working draft at the time of writing.

D.25 [depr.string.capacity] Deprecated basic_string capacity

1 The following member is declared in addition to those members specified in 23.4.3.5 [string.capacity]:

```
namespace std {
  template<class charT, class traits = char_traits<charT>,
           class Allocator = allocator<charT>>
```

```
    class basic_string {
    public:
      void reserve();
  };
}
```

```
void reserve();
```

2 *Effects:* After this call, `capacity()` has an unspecified value greater than or equal to `size()`.

[*Note 1:* This is a non-binding shrink to fit request. —*end note*]

7 Acknowledgements

Thanks to Michael Parks for the pandoc-based framework used to transform this document's source from Markdown.

8 References

[N4944] Thomas Köppe. 2023-03-22. Working Draft, Standard for Programming Language C++.
https://wg21.link/n4944

[P0966R1] Mark Zeren, Andrew Luo. 2018-02-08. string::reserve Should Not Shrink.
https://wg21.link/p0966r1

[P2139R2] Alisdair Meredith. 2020-07-15. Reviewing Deprecated Facilities of C++20 for C++23.
https://wg21.link/p2139r2