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

Document #:	P2870R2
Date:	2023-09-14
Project:	Programming Language C++
Audience:	Library Evolution

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

R2: September 2023 (midterm mailing)

— Removed revision history's redundant subsection numbering
— Recorded initial LEWG reflector review

— Wording updates
  — Affirmed wording against latest working draft, [N4958]
  — Updated stable label cross-reference to C++23

R1: August 2023 (midterm mailing)

Updates following LEWG online review, before submitting to electronic poll.

— Fixed assorted typos and markdown issues
— Moved to LEWG from LEWGI
— Add a second paragraph to Analysis highlighting the risk of using the deprecated function across different versions of the Standard
— Added a subsection below Analysis for migrating old code
— Corrected test results to show when MSVC started warning with the /W3 switch
— Strengthened recommendation to remove in C++26 to help correct old code that still expects the `shrink_to_fit` behavior
— Confirmed that no changes are expected for 16.4.5.3.2 [zombie.names]
— Provided Annex C wording
— Added acknowledgements for all the help

R0: May 2023 (pre-Varna)

— Initial draft of this 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, [P2863R1], 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.24 [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.

Note that retaining this overload leads to a behavior change between code written against C++11 – C++17 vs. code written against C++20 or later. To maintain consistent behavior across all C++ dialects from C++11 onward, the call should be changed to `shrink_to_fit()` instead.

4.1 Updating deprecated code

Code that is still using the deprecated function today can be updated in one of three ways.

1. Simply remove the function call, as it does nothing.
2. Call `reserve(0)` to retain the call if desired, but it still does nothing.
3. Call `shrink_to_fit()` instead to preserve the pre-C++20 semantic.

All three updates will compile with tool chains back to C++11, and all but option 3 will compile (with a different meaning) back to C++98.

4.2 Deprecation experience

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

```
#include <string>

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

— libc++ 12.0
— libstdc++ 11.1
— Microsoft 19.25 (requires /W3 in MSVC)

4.3 C++23 review

The LEWG telecon on 2020/07/13 saw general agreement that this member is a holdover from days past and whose replacement has been in place for some time. The consensus was 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.

5 Recommendation for C++26

Following up from the C++23 recommendation for removal, we note that the change of behavior between C++17 and C++20 was silent, so users may be unaware that the good style of switching to `shrink_to_fit` in C++11 is now essential to retain that behavior. Hence, we strongly encourage removing this deprecated feature from C++26.

5.1 Initial review by LEWG reflector 2023/06/26 – 2023/07/02

A motion to advance the paper to electronic polling passed with 18 votes and no objection to moving to EP and a strong (but not unanimous) sentiment in favor of this paper.

An observation was made that MSVC emits a warning under the IDE's default setting of `/W3`, and this paper was updated accordingly.

One concern raised was that retaining the old function did no harm, so this paper was updated to highlight that with the silent behavior change to this API in C++20, retaining this API does raise concerns best handled by updating the deprecated code.

An observation was made that being deprecated for two Standard cycles was probably a reasonable notice before removing a deprecated facility, but two cycles is likely the minimum notice we would want to give unless a feature is doing active harm. This point was not discussed, but no objections to the sentiment were raised.

6 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4958], the latest draft at the time of writing.

6.1 Zombie names

The only function being removed is a single overload of a member function that remains. Hence, there is nothing to add to 16.4.5.3.2 [zombie.names].

6.2 Update Annex C:

C.1.X Annex D: compatibility features [diff.cpp23.depr]

1 **Change:** Remove `basic_string::reserve()` with no arguments.

Rationale: An earlier Standard changed this function from behaving like a call to `shrink_to_fit` to instead having no effect. Using old code written to the old idiom across different implementations of this Standard is misleading.

Effect on original feature: A valid C++ 2023 program that calls `reserve()` on a `basic_string` object may fail to compile. The old functionality can be achieved by calling `shrink_to_fit()` instead, or the function call can be safely eliminated with no side effects.

6.3 Strike all of D.24 [depr.string.capacity] Deprecated `basic_string` capacity

D.24 [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*]

6.4 Update cross-reference for stable labels for C++23

Cross-references from ISO C++ 2023

All clause and subclause labels from ISO C++ 2023 (ISO/IEC 14882:2023, *Programming Languages — C++*) are present in this document, with the exceptions described below.

container.gen.reqmts *see*
container.requirements.general

depr.res.on.required *removed*
depr.string.capacity *removed*

7 Acknowledgements

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

Thanks to Matt Godbolt for Compiler Explorer, still the best tool for testing code against a variety of compilers implementing multiple editions of the C++ Standard.

Thanks to Detlef Vollmann for highlighting the need to document a migration strategy.

Thanks to Tim Song for pointing out that MSVC actually does give a deprecation warning when the correct (default) warning flags are used.

Thanks to Lori Hughes for reviewing this paper and providing editorial feedback.

8 References

[N4958] Thomas Köppe. 2023-08-14. Working Draft, Programming Languages -- C++.
https://wg21.link/n4958

[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

[P2863R1] Alisdair Meredith. 2023-08-16. Review Annex D for C++26.
https://wg21.link/p2863r1