std::fewer

Document #: P4161R0
Date: 2026-04-01
Project: Programming Language C++
Audience: LEWG, LWG
Reply-to: Jeff Snyder
<jeff-isocpp@caffeinated.me.uk>
Jonathan B. Coe
<jonathanbcoe@gmail.com>
Neelofer Banglawala
<drnbanglawala@gmail.com>

Contents

1 Abstract

The C++ standard library provides `std::less`, a comparator function object used widely in ordered containers and algorithms. However, `std::less` is grammatically incorrect when applied to integral types, which are discrete and countable quantities: the English language prescribes *fewer* for countable quantities and *less* for continuous or uncountable ones. This paper proposes `std::fewer`, an analogue of `std::less` constrained to integral types, and gives `std::less` undefined behaviour when instantiated with an integral type, bringing the C++ standard library into conformance with standard English grammar.

2 Motivation

The English language distinguishes between two comparative adjectives for describing a reduction in quantity. *Fewer* is used with discrete, countable nouns, and *less* with continuous or uncountable ones. This distinction is well-established and widely documented [Fowler]; its violation is among the most frequently cited grammatical errors in everyday English, as demonstrated by the public criticism directed at supermarkets whose express checkout signs read “10 items or less” [Telegraph].

Integers are discrete. They are countable. Nevertheless, `std::less` has since C++98 been the standard comparator for integral types, and the following code compiles without error or warning on all known implementations:

```
std::set<int, std::less<int>> s;
```

The library provides no grammatically correct alternative, and this oversight has persisted, unchallenged, for over two decades.

3 Design

3.1 `std::fewer`

We propose a new comparison function object, `std::fewer<T>`, constrained to integral types (6.9.2 [basic.fundamental]). Its semantics are identical to `std::less<T>`: `operator()` returns `x < y`. The name correctly reflects the discrete, countable nature of the types it operates on.

We do not propose `std::fewer<void>`, as the grammatical status of `void` — whether it is to be considered a countable or uncountable quantity — raises questions that fall outside the scope of the present work and which we leave for a future paper.

3.2 Changes to `std::less`

`std::less<T>` is correctly used with non-integral types, and those uses are unaffected by this proposal:

```
std::set<std::string, std::less<std::string>> s;    // "less text" - acceptable
std::set<double, std::less<double>> s;              // "less value" - fine
std::set<MyWidget*, std::less<MyWidget*>> s;        // pointers - uncountable in spirit
```

For integral types, however, we propose that `std::less` have undefined behaviour. Implementations are encouraged to diagnose this at compile time, though they are not required to do so.

3.3 Impact on existing code

All existing code using `std::less` with integral types has undefined behaviour under this proposal. We are aware that this affects the majority of ordered containers in production codebases worldwide. We consider this acceptable. The alternative — allowing grammatical incorrectness to persist in the standard library — is worse.

Migration is straightforward: `std::less<T>` should be replaced with `std::fewer<T>` wherever `T` is an integral type. We considered adding a `[[deprecated]]` attribute to the integral specialisation of `std::less` to assist tooling authors in automating this migration, but chose not to do so, as deprecated facilities are permitted to remain in the standard indefinitely, and we wish to express the committee’s firm disapproval with appropriate urgency.

3.4 Feature-test macro

We propose the macro `__cpp_lib_fewer` with value `202604L`.

4 Proposed Wording

The following changes are relative to [N5001].

4.1 Modify the `<functional>` synopsis 22.10.2 [functional.syn]

After the declaration of `std::less`, insert:

```
template<class T> struct fewer;         // freestanding
```

4.2 Modify 22.10.8.5 [comparisons.less]

At the end of 22.10.8.5 [comparisons.less], add a new paragraph:

3 *Remarks*: If `T` is an integral type (6.9.2 [basic.fundamental]), the behaviour is undefined.

4.3 Add a new subclause [comparisons.fewer]

Insert a new subclause after 22.10.8.5 [comparisons.less]:

22.10.8.6 Class template `fewer` [comparisons.fewer]

```
template<class T> requires integral<T>
struct fewer {
    constexpr bool operator()(const T& x, const T& y) const;
};
```

1 `operator()` returns `x < y`.

4.4 Modify 17.3.2 [version.syn]

Add the following macro definition:

```
#define __cpp_lib_fewer  202604L  // also in <functional>
```

5 Acknowledgments

The authors thank the WG21 Study Group on Grammatical Correctness (SGxx), which does not yet exist but which we hope this paper will motivate the committee to establish.

6 References

[Fowler] H.W. Fowler. 1926. A Dictionary of Modern English Usage.

[N5001] Thomas Köppe. 2024-12-17. Working Draft, Programming Languages — C++.
https://wg21.link/n5001

[Telegraph] The Daily Telegraph. 2008-09. Tesco to ditch “ten items or less” sign after good grammar campaign.
https://www.telegraph.co.uk/news/uknews/2659948/Tesco-to-ditch-ten-items-or-less-sign-after-good-grammar-campaign.html