Document number: P4216R0
Date: 2026-05-07
Project: Programming Language C++
Audience: LEWG
Reply-to: Michael Florian Hava[1] <mfh.cpp@gmail.com>

Comparisons for span

Abstract
This paper proposes comparison operators for `span`, mirroring the design of `string_view`, `optional<T&>`, and `reference_wrapper`.

Tony Table

Before	Proposed

`reference_wrapper<int> r0 = …, r1 = …;`
`optional<int&> o0 = …, o1 = …;`
`string_view s0 = …, s1 = …;`
`span<T> p0 = …, p1 …;`
	`reference_wrapper<int> r0 = …, r1 = …;`
	`optional<int&> o0 = …, o1 = …;`
	`string_view s0 = …, s1 = …;`
	`span<T> p0 = …, p1 …;`

`//equality comparisons:`
`//NOTE: all of these do „deep value comparisons“`
`r0 == r1;`
`o0 == o1;`
`s0 == s1;`
 	
`ranges::equal(p0, p1);`
	`//equality comparisons:`
	`//NOTE: all of these do „deep value comparisons“`
	`r0 == r1;`
	`o0 == o1;`
	`s0 == s1;`
	
	`p0 == p1;`

`//three-way comparisons:`
`//NOTE: all of these do „deep value comparisons“`
`r0 <=> r1;`
`o0 <=> o1;`
`s0 <=> s1;`
 	
`lexicographical_compare_three_way(`
`  p0.begin(), p0.end(),`
`  p1.begin(), p1.end(),`
`  synth-three-way //but that's exposition-only 😀`
`);`
	`//three-way comparisons:`
	`//NOTE: all of these do „deep value comparisons“`
	`r0 <=> r1;`
	`o0 <=> o1;`
	`s0 <=> s1;`
	
	`p0 <=> p1;`

Revisions
R0: Initial version

Motivation
The standard library has several types that represent non-owning references to data: `span`, `string_view`, `reference_wrapper`, `optional<T&>`. All but the first of these support comparisons (both equality and three-way). This paper aims to fix this inconsistency by adding comparisons to `span`.

Design Space
Deep value comparisons
All existing *reference types* use the same comparison model: they compare the referenced values (if any), not their identity. This model has always been in place for `string_view` and was adopted

[1] RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg, Austria, michael.hava@risc-software.at

for `optional<T&>` and `reference_wrapper` in C++26 with [P2988] and [P2944] respectively. We propose to adopt this model for `span`.

Why `span` isn't already comparable?

The original proposal for `span` ([P0122]) included comparisons, modelling deep value comparisons. It was adopted in Jacksonville in March 2018. In October of the same year they were removed in the San Diego meeting by [P1085].

The concerns of P1085 can be summarised as: `span` …

- … has „reference semantics", therefore it's value can change transparently.
- … does not enforce deep const (note, that there is no discussion on `span<const T>`).
- … rebinds on assignment.

All these concerns are real. What P1085 doesn't acknowledge is that all but the second concern equally apply to `string_view`, neither does it make a convincing case for why these concerns are in any way alleviated by removing comparisons!

Furthermore, all these concerns are the correct semantics for a *reference type*! It took us close to a decade to come to the conclusion that rebinding assignment is exactly the right thing for `optional<T&>` to do. The same design was chosen for `reference_wrapper` and `function_ref`. We therefore see no point in not providing comparisons for `span`.

Impact on the Standard

This proposal is a pure library addition. Existing standard library classes are modified in a non-ABI-breaking way.

Proposed Wording

Wording is relative to [N5032]. Additions are presented like this, removals like ~~this~~ and drafting notes like **this**.

[version.syn]

```
#define __cpp_lib_span_comparison YYYYMML //freestanding, also in <span>
```

[DRAFTING NOTE: Adjust the placeholder value as needed to denote the proposal's date of adoption.]

[span.syn]

??.?.?.? Header synopsis **[span.syn]**

```
// mostly freestanding
namespace std {
  …
  // [views.span], class template span
  template<class ElementType, size_t Extent = dynamic_extent>
    class span;

  template<class ElementType, size_t Extent>
    constexpr bool operator==(span<ElementType, Extent> x, span<ElementType, Extent> y);
  template<class ElementType, size_t Extent>
    constexpr synth-three-way-result<ElementType>
      operator<=>(span<ElementType, Extent> x, span<ElementType, Extent> y);

  template<class ElementType, size_t Extent>
    constexpr bool ranges::enable_view<span<ElementType, Extent>> = true;
  …
}
```

[DRAFTING NOTE: We assume that these changes are enough to „trigger" [container.reqmts-44] and [container.opt.reqmts-4]]

Acknowledgements

Thanks to RISC Software GmbH for supporting this work.