Document number: P3625R1
Date: 2026-04-07
Project: Programming Language C++
Audience: LWG
Reply-to: Michael Florian Hava[1] <mfh.cpp@gmail.com>

either neither

Abstract
This paper proposes adding two usability concepts to more concisely express (not) matching a list of types.

Tony Table

Before	Proposed

`template<typename T>`
`requires(same_as<T, char> or same_as<T, short>)`
`void func(T) { … }`
	`template<either<char, short> T>`
	`void func(T) { … }`
	
	`// or just:`
	
	`void func(either<char, short> auto) { … }`

`template<typename T>`
`requires(not same_as<T, short> and not same_as<T, int>)`
`void func(T) { … }`
	`template<neither<short, int> T>`
	`void func(T) { … }`
	
	`// or just:`
	
	`void func(neither<short, int> auto) { … }`

Revisions
R0: Initial version

R1: Updates after LEWG review in Croydon on 2026-03-27:

- Added new FTM instead of bumping existing FTM.

Motivation
We've encountered the need to express a template type parameter matching one of multiple types across several projects. Whilst we can't provide sustained data on how widespread this pattern already is, we want to point to both the documentation for `same_as` on cppreference as well as stackoverflow for prior usage/requests to express this semantics.

Design Space
Given these facilities are simple wrappers around foldings of `same_as`, the only design decisions pertain to naming. We've considered the following alternative names and discarded them for the given reasons.

- `oneof/noneof` … only partially correct as one could assume that `oneof` expresses matching exactly one given type.
- `anyof/noneof` … express the basic idea, but are unusual spellings to circumvent collisions with the respective algorithms.

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

The best names we could come up with after excluding the above have been `either`/`neither`, which admittedly look funky[2] when used directly (`either<T, U...>`) but look fine when using the "terse notation" (`template<either<U...> T>, void f(either<U...> auto)`).

Impact on the Standard

This proposal is a pure library addition.

Proposed Wording

Wording is relative to [N5001]. Additions are presented like this, removals like ~~this~~ and drafting notes like **this**.

[version.syn]

```
#define __cpp_lib_either_neither YYYYMML // freestanding, also in <concepts>
```

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

[concepts]

??.? Header <concepts> synopsis **[concepts.syn]**

```
// all freestanding
namespace std {
  // [concepts.lang], language-related concepts
  // [concept.same], concept same_as
  template<class T, class U>
    concept same_as = see below;

  template<class T, class... U>
    concept either = (same_as<T, U> || ...);

  template<class T, class... U>
    concept neither = !either<T, U...>;

  // [concept.derived], concept derived_from
}
```

Acknowledgements

Thanks to RISC Software GmbH for supporting this work. Thanks to Bernhard Manfred Gruber for giving feedback on the draft of this paper.

[2] Not more funky then already existing concepts (e.g. `derived_from<Derived, Base>`) though…