Proposing `std::is_specialization_of`

Document #: WG21 P2098R0
Date: 2020–02–13
Audience: ~~LEWG-I~~ done ⇒ LEWG ⇒ LWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>
Bob Steagall <bob.steagall.cpp@gmail.com>

Contents

Abstract

[P2078R0] “proposes the addition [to the standard library] of a new unary type traits class template, **`is_complex<T>`**.” Recognizing that there are numerous similar traits templates that could profitably be added to the standard library, this paper proposes to add instead a single, more general trait that can be straightforwardly customized via a simple alias, to obtain any such trait as needed.

To think is to forget a difference, to generalize, to abstract.
— Jorge Luis Borges

Generalizations, like brooms, ought not to stand in a corner forever; they ought to sweep as a matter of course.
— John Lukacs

1 Introduction

[P2078R0] “proposes the addition [to the standard library] of a new unary type traits class template, **`is_complex<T>`**.” Such a trait can be straightforwardly implemented as follows:

```
template< class T >
struct
  is_complex : std::false_type;

template< class U >
struct
  is_complex<std::complex<U>> : std::true_type;

template< class T >
inline constexpr bool
  is_complex_v = is_complex<T>::value;
```

However, there are numerous very similar trait templates that could profitably be added to the standard library.

For example, each of the following exemplifies a useful trait much like the above **`is_complex<T>`**. Further, the authors know that these and many others have already been implemented (albeit under an uglified or slightly different name, in some cases) using essentially the same technique as was used above to implement **`is_complex<T>`**.

- **`is_reference_wrapper<T>`**,
- **`is_string<T>`**,
- **`is_pair<T>`**,
- **`is_tuple<T>`**,
- **`is_vector<T>`**,
- and many more!

Therefore, this paper proposes to **add instead a single, more general trait** that can be straightforwardly customized via a simple alias, to obtain any such trait as needed.

2 Proposal

As motivated above, we propose to add a more general **`is_specialization_of`** trait to the standard library. Such a trait can be defined along the following lines such that it corresponds to **`true_type`** when a given type is a specialization of a given template; otherwise, it corresponds to **`false_type`** when the given type is not a specialization of that given template:

```
template< class T
        , template<class...> class Primary >
struct
  is_specialization_of : false_type;

template< template<class...> class Primary
        , class... Args >
struct
  is_specialization_of< Primary<Args...>, Primary> : true_type;

template< class T
        , template<class...> class Primary >
inline constexpr bool
  is_specialization_of_v = is_specialization_of<T, Primary>::value;
```

Given the above primitive, we can produce [P2078R0]'s proposed **`is_complex<T>`** as follows:

```
template< class T >
using
  is_complex = is_specialization_of<T, std::complex>;

template< class T >
inline constexpr bool
  is_complex_v = is_specialization_of_v<T, std::complex>;
```

Each of the other examples listed in §1 can be produced in like manner. This frees programmers from the need to comprehend the intricacies of template template parameters (a topic often not well-understood by non-experts). Moreover, the nature of the proposed trait is such that it can be used in a very wide variety of contexts, not just within namespace **`std`**.

Finally, investigation has revealed that there is prior art for this proposal, as MSVC's standard library implementation provides this exact trait[1], and uses it in the manner proposed herein.

The proposed name, **`is_specialization_of`**, is admittedly slighly longer than some consider convenient. While shorter names[2] are of course possible, we prefer the proposed longer name in the interest of clarity of purpose: we believe that the longer name succinctly and correctly captures the mission of the trait and follows the naming precedent of the long-standing **`is_base_of`** trait.

For all the reasons above, we believe the **`is_specialization_of`** trait to be a worthy candidate for C++23 standardization.

3 Proposed wording[3]

3.1 After adjusting *yyyymm*, below, so as to denote this proposal's month of adoption, insert the following line among the similar directives following [version.syn]/2:

```
#define __cpp_lib_is_specialization_of yyyymmL  // also in <type_traits>
```

3.2 Augment [meta.type.synop] as shown:

```
namespace std {
  ⋮
  template<class Base, class Derived>
    struct is_pointer_interconvertible_base_of;
  template<class T, template<class...> Primary>
    struct is_specialization_of;
  ⋮
  template<class Base, class Derived>
    inline constexpr bool is_pointer_interconvertible_base_of_v
      = is_pointer_interconvertible_base_of<Base, Derived>::value;
  template<class T, template<class...> Primary>
    inline constexpr bool is_specialization_of_v
      = is_specialization_of<T,Primary>::value;
  ⋮
}
```

3.3 Augment Table [tab:meta.rel] (Type relationship predicates) as shown:

[1]Their trait is found in their **`<type_traits>`** header under the uglified name **`_Is_specialization`**, with an equivalent implementation to that shown above.

[2]Examples of such shorter names include **`is_specialization`**, **`specializes`**, **`is_spec_of`**, and the like.

[3]Proposed additions (there are no ~~deletions~~) are based on [N4849]. Editorial instructions and drafting notes look like this .

Primary	**Condition**	**Comments**
⋮		
`template<class Base, class Derived> struct is_pointer_interconvertible_base_of;`	`Derived` is unambiguously . . .	If `Base` and `Derived` are . . .
`template<class T, class<...> Primary> struct is_specialization_of;`	`T` is a specialization ([temp.spec]) of `Primary`	
⋮		

4 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments.

5 Bibliography

[N4849] Richard Smith: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N4849 (pre-Prague mailing), 2020–01–14. https://wg21.link/n4849.

[P2078R0] Bob Steagall: "Add new traits type **`std::is_complex<T>`**." ISO/IEC JTC1/SC22/WG21 document P2078R0 (pre-Prague mailing), 2020–01–13. https://wg21.link/P2078R0.

6 Document history

Rev.	**Date**	**Changes**
0	2020–02–13	• Published as P2098R0, post-Prague mailing, incorporating minor guidance from very favorable (10 | 7 | 0 | 0 | 0) LEWG-I review of a draft of this paper.