Add new traits type `std::is_complex<T>`

Abstract

This paper proposes the addition of a new unary type traits class template, `is_complex<T>`, to the standard C++ library. The facility described herein is a pure addition, requiring no changes to existing code.

1 Motivation

There is not yet a standard method for detecting at compile time whether or not a given type is a complex number represented by `std::complex<T>`. This paper proposes that a new traits class template, `is_complex<T>`, be added to namespace `std` in order to remedy this deficit.

The reasons for adding this new traits facility are twofold: first, to fill a small gap in the existing catalogue of type traits; second, and more specifically, to facilitate the compile-time determination of the conjugate transpose type of a matrix (also called the Hermitian transpose or simply, the Hermitian).

As motivating example, it is envisioned that `is_complex<T>` might be used in the following way (adapted from [P1385R4]):

```cpp
template<class ET, class OT>
class matrix
{
public:
  // Types
  using engine_type     = ET;
  using element_type    = typename engine_type::element_type;
  ...
  using transpose_type  = matrix<...>;
  using hermitian_type  = conditional_t<is_complex<element_type>, matrix, transpose_type>;
  ...

  constexpr hermitian_type h() const;
  ...
};
```

The idea here is straightforward: if `element_type` is a type alias of `std::complex<U>` for some scalar type `U`, then the type of the Hermitian transpose is the same as that of the matrix type itself, and the `matrix` object returned by member function `h()` would contain the transposed and conjugated elements of the target object.

Otherwise, `element_type` is presumed to represent a scalar and the type of the Hermitian transpose is the same as that of the ordinary transpose. The transpose type may be a different type than `matrix`, such as a non-owning “view” type, but has the same `element_type` as `matrix`. 
2 Proposed Wording

In section [meta.type.synop]

```cpp
// 20.15.4.2, composite type categories
template<class T> struct is_reference;
template<class T> struct is_arithmetic;
+ template<class T> struct is_complex;
template<class T> struct is_fundamental;
template<class T> struct is_object;
```

Also in section [meta.type.synop]:

```cpp
// 20.15.4.2, composite type categories
template<class T>
inline constexpr bool is_reference_v = is_reference<T>::value;
template<class T>
inline constexpr bool is_arithmetic_v = is_arithmetic<T>::value;
+ template<class T>
+ inline constexpr bool is_complex_v = is_complex<T>::value;
template<class T>
inline constexpr bool is_fundamental_v = is_fundamental<T>::value;
template<class T>
inline constexpr bool is_object_v = is_object<T>::value;
```

In section [tab:meta.unary.comp]:

<table>
<thead>
<tr>
<th>Template</th>
<th>Condition</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>template&lt;class T&gt;</td>
<td>'T' is an lvalue reference or</td>
<td>is_reference</td>
</tr>
<tr>
<td>is_reference</td>
<td>an rvalue reference</td>
<td></td>
</tr>
<tr>
<td>template&lt;class T&gt;</td>
<td>'T' is an arithmetic</td>
<td>is_arithmetic</td>
</tr>
<tr>
<td>is_arithmetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>template&lt;class T&gt;</td>
<td>'T' is equal to 'complex&lt;U&gt;' for</td>
<td>is_complex</td>
</tr>
<tr>
<td>is_complex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>template&lt;class T&gt;</td>
<td>'T' is an object type (6.8)</td>
<td>is_object</td>
</tr>
<tr>
<td>is_object</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Revision history

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>Initial version for pre-Prague mailing.</td>
</tr>
</tbody>
</table>

3 References

https://wg21.link/p1385r4