Constexpr in `std::pointer_traits`

Document #:	D1006R0
Date:	2018-03-27
Project:	Programming Language C++
Audience:	LEWG

Reply-to: Louis Dionne <ldionne.2@gmail.com>

1 Abstract

As part of the `constexpr` reflection effort, and in particular making `std::vector` `constexpr`, we need to make `std::pointer_traits` `constexpr` (it is used in the implementation).

2 Difficulties

The standard currently defines a base template `std::pointer_traits` and a specialization of it for raw pointers (`std::pointer_traits<T*>`). Marking the base template as `constexpr` would imply that all specializations of it need to be marked `constexpr` too, since specializations of templates in namespace `std` for user-defined types need to retain the same interface as the base template. Indeed, per **[namespace.std] 20.5.4.2.1/1** in [N4727]:

> The behavior of a C++ program is undefined if it adds declarations or definitions to namespace `std` or to a namespace within namespace `std` unless otherwise specified. A program may add a template specialization for any standard library template to namespace `std` only if the declaration depends on a user-defined type and the specialization meets the standard library requirements for the original template and is not explicitly prohibited.

However, forcing all specializations of `std::pointer_traits` to be marked `constexpr` will preclude useful fancy pointer implementations from using it, such as `offset_ptr`. `offset_ptr` is a pointer represented as an offset from `this`, which is used in memory mapped files and similar contexts. The problem with `offset_ptr` is that it uses a `reinterpret_cast` internally, which isn't allowed in constant expressions (and the barrier to allowing that is very high).

So marking the base template `constexpr` is not an option without changing **[namespace.std]**. The only other option is to mark the specialization of `std::pointer_traits` for raw pointers (`std::pointer_traits<T*>`) as `constexpr`, which does not seem to validate **[namespace.std]** because it is not a user-provided specialization.

Also note that in practice, we don't expect (and have no use for) `std::vector` being `constexpr`-friendly for allocators other than the default allocator, which means that we don't really care about making more than `std::pointer_traits<T*>` `constexpr`. This is the direction this paper takes.

3 Proposed wording

This wording is based on the working draft [N4727]. Change in **[pointer.traits] 23.10.3/1**:

```
namespace std {
  template<class Ptr> struct pointer_traits {
    using pointer         = Ptr;
    using element_type    = see below;
    using difference_type = see below;

    template<class U> using rebind = see below;

    static pointer pointer_to(see below r);
  };

  template<class T> struct pointer_traits<T*> {
    using pointer         = T*;
    using element_type    = T;
    using difference_type = ptrdiff_t;

    template<class U> using rebind = U*;

    static constexpr pointer pointer_to(see below r) noexcept;
  };
}
```

4 Acknowledgements

Thanks to Ion Gaztañaga for discussing the troubles of `offset_ptr` and **`constexpr`** with me.

5 References

[N4727] Richard Smith, *Working Draft, Standard for Programming Language C++*
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4727.pdf