Constexpr in `std::pointer_traits`

Document #: P1006R1
Date: 2018-10-07
Project: Programming Language C++
Audience: LWG
Reply-to: Louis Dionne <ldionne@apple.com>

1 Revision history

- R0 – Initial draft
- R1
  - Add wording in the specification of `std::pointer_traits`, not only the synopsis.
  - Add caveat that user-provided specializations of `std::pointer_traits<T*>` now need to provide a `constexpr` `pointer_to` method.

2 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).

3 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] 15.5.4.2.1/2 in [N4762]:

> Unless explicitly prohibited, a program may add a template specialization for any standard library class template to namespace `std` provided that (a) the added declaration depends on at least one program-defined type and (b) the specialization meets the standard library requirements for the original template.

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 violate `[namespace.std]` because it is not a user-supplied 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.

However, it does mean that user-provided specializations of `std::pointer_traits<T*>`, where T is a user-defined type, need to abide by the added `constexpr` requirement.

4 Proposed wording

This wording is based on the working draft `[N4762]`. Change in `[pointer.traits] 19.10.3/1`:

```cpp
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;
    };
}
```

Change in `[pointer.traits.functions] 19.10.3.2`:

```cpp
19.10.3.2 Pointer traits member functions [pointer.traits.functions]

    static pointer pointer_traits::pointer_to(see below r);
    static constexpr pointer pointer_traits<T*>::pointer_to(see below r) noexcept;

Remarks: If `element_type` is `cv void`, the type of `r` is unspecified; otherwise, it is `element_type&`.
```
Returns: The first member function returns a pointer to \texttt{r} obtained by calling \texttt{Ptr::pointer_to(r)} through which indirection is valid; an instantiation of this function is ill-formed if \texttt{Ptr} does not have a matching \texttt{pointer_to} static member function. The second member function returns \texttt{addressof(r)}.

5 Acknowledgements

Thanks to Ion Gaztañaga for discussing the troubles of \texttt{offset_ptr} and \texttt{constexpr} with me.

6 References
