Making `std::forward_list` constexpr

Document #: P1929R0
Date: 2019-10-07
Project: Programming Language C++
Audience: LEWGI
Reply-to: Alexander Zaitsev <zamazan4ik@tut.by, zamazan4ik@gmail.com>

1 Revision history

- R0 – Initial draft

2 Abstract

`std::forward_list` is not currently `constexpr` friendly. With the loosening of requirements on `constexpr` in [P0784R1] and related papers, we can now make `std::forward_list` `constexpr`, and we should in order to support the `constexpr` reflection effort (and other evident use cases).

3 Motivation

`std::forward_list` is not so widely-used standard container as `std::vector` or `std::string`. But there is no reason to keep `std::forward_list` in non-`constexpr` state since one of the main directions of C++ evolution is compile-time programming. And we want to use in compile-time as much as possible from STL. And this paper makes `std::forward_list` available in compile-time.

4 Proposed wording

We basically mark all the member and non-member functions of `std::forward_list` `constexpr`.

Direction to the editor: please apply `constexpr` to all of `std::forward_list`, including any additions that might be missing from this paper.

In [**support.limits.general**], add the new feature test macro `__cpp_lib_constexpr_forward_list` with the corresponding value for header `<forward_list>` to Table 36 [**tab:support.ft**].

Change in [**forwardlist.syn**] **22.3.4**:

```
#include <initializer_list>

namespace std {
  // 22.3.10, class template forward_list
  template<class T, class Allocator = allocator<T>> class forward_list;

  template<class T, class Allocator>
    constexpr bool operator==(const forward_list<T, Allocator>& x, const forward_list<T, Allocat
  template<class T, class Allocator>
    constexpr synth-three-way-result<T> operator<=>(const forward_list<T, Allocator>& x, const f

  template<class T, class Allocator>
    constexpr void swap(forward_list<T, Allocator>& x, forward_list<T, Allocator>& y)
      noexcept(noexcept(x.swap(y)));

  template<class T, class Allocator, class U>
    constexpr void erase(forward_list<T, Allocator>& c, const U& value);
  template<class T, class Allocator, class Predicate>
    constexpr void erase_if(forward_list<T, Allocator>& c, Predicate pred);

  [...]
}
```

Add after [**forwardlist.overview**] **22.3.9.1/2**:

> The types `iterator` and `const_`iterator meet the constexpr iterator requirements ([iterator.requirements.general]).

Change in [**forwardlist.overview**] **22.3.9.1**:

```
namespace std {
  template<class T, class Allocator = allocator<T>>
  class forward_list {
  public:
    // types
    using value_type             = T;
    using allocator_type         = Allocator;
    using pointer                = typename allocator_traits<Allocator>::pointer;
    using const_pointer          = typename allocator_traits<Allocator>::const_pointer;
    using reference              = value_type&;
    using const_reference        = const value_type&;
    using size_type              = implementation-defined; // see 22.2
    using difference_type        = implementation-defined; // see 22.2
    using iterator               = implementation-defined; // see 22.2
    using const_iterator         = implementation-defined; // see 22.2

    // 22.3.9.2, construct/copy/destroy
    constexpr forward_list() : forward_list(Allocator()) { }
    constexpr explicit forward_list(const Allocator&);
    constexpr explicit forward_list(size_type n, const Allocator& = Allocator());
    constexpr forward_list(size_type n, const T& value, const Allocator& = Allocator());
    template<class InputIterator>
      constexpr forward_list(InputIterator first, InputIterator last, const Allocator& = Allocat
    constexpr forward_list(const forward_list& x);
    constexpr forward_list(forward_list&& x);
    constexpr forward_list(const forward_list& x, const Allocator&);
    constexpr forward_list(forward_list&& x, const Allocator&);
    constexpr forward_list(initializer_list<T>, const Allocator& = Allocator());
    constexpr ~forward_list();
    constexpr forward_list& operator=(const forward_list& x);
    constexpr forward_list& operator=(forward_list&& x)
      noexcept(allocator_traits<Allocator>::is_always_equal::value);
    constexpr forward_list& operator=(initializer_list<T>);
    template<class InputIterator>
      constexpr void assign(InputIterator first, InputIterator last);
    constexpr void assign(size_type n, const T& u);
    constexpr void assign(initializer_list<T>);
    constexpr allocator_type get_allocator() const noexcept;

    // 22.3.9.3, iterators
    constexpr iterator                before_begin() noexcept;
    constexpr const_iterator          before_begin() const noexcept;
    constexpr iterator                begin() noexcept;
    constexpr const_iterator          begin() const noexcept;
    constexpr iterator                end() noexcept;
    constexpr const_iterator          end() const noexcept;

    constexpr const_iterator          cbegin() const noexcept;
    constexpr const_iterator          cbefore_begin() const noexcept;
    constexpr const_iterator          cend() const noexcept;

    // capacity
    [[nodiscard]] constexpr bool empty() const noexcept;
    constexpr size_type max_size() const noexcept;

    // 22.3.9.4, element access
    constexpr reference       front();
    constexpr const_reference front() const;

    // 22.3.9.5, modifiers
    template<class... Args> constexpr reference emplace_front(Args&&... args);
    constexpr void push_front(const T& x);
    constexpr void push_front(T&& x);
    constexpr void pop_front();

    template<class... Args> constexpr iterator emplace_after(const_iterator position, Args&&...
    constexpr iterator insert_after(const_iterator position, const T& x);
    constexpr iterator insert_after(const_iterator position, T&& x);

    constexpr iterator insert_after(const_iterator position, size_type n, const T& x);
    template<class InputIterator>
    constexpr iterator insert_after(const_iterator position, InputIterator first, InputIterator
    constexpr iterator insert_after(const_iterator position, initializer_list<T> il);
```

```
    constexpr iterator erase_after(const_iterator position);
    constexpr iterator erase_after(const_iterator first, const_iterator last);
    constexpr void     swap(forward_list&)
      noexcept(allocator_traits<Allocator>::is_always_equal::value);

    constexpr void resize(size_type sz);
    constexpr void resize(size_type sz, const value_type& c);
    constexpr void     clear() noexcept;

    // 22.3.9.6, forward_list operations
    constexpr void splice_after(const_iterator position, forward_list& x);
    constexpr void splice_after(const_iterator position, forward_list&& x);
    constexpr void splice_after(const_iterator position, forward_list& x, const_iterator i);
    constexpr void splice_after(const_iterator position, forward_list&& x, const_iterator i);
    constexpr void splice_after(const_iterator position, forward_list& x, const_iterator first,
    constexpr void splice_after(const_iterator position, forward_list&& x, const_iterator first,

    constexpr size_type remove(const T& value);
    template<class Predicate> constexpr size_type remove_if(Predicate pred);

    constexpr size_type unique();
    template<class BinaryPredicate>
    constexpr size_type unique(BinaryPredicate binary_pred);

    constexpr void merge(forward_list& x);
    constexpr void merge(forward_list&& x);
    template<class Compare> constexpr void merge(forward_list& x, Compare comp);
    template<class Compare> constexpr void merge(forward_list&& x, Compare comp);

    constexpr void sort();
    template<class Compare> constexpr void sort(Compare comp);

    constexpr void reverse() noexcept;
  };

  template<class InputIterator,
           class Allocator = allocator<iter-value-type<InputIterator>>>
    forward_list(InputIterator, InputIterator, Allocator = Allocator())
      -> forward_list<iter-value-type<InputIterator>, Allocator>;

  // swap
  template<class T, class Allocator>
    constexpr void swap(forward_list<T, Allocator>& x, forward_list<T, Allocator>& y)
      noexcept(noexcept(x.swap(y)));
}
```

Change in [**forwardlist.cons**] **22.3.9.2**:

```
constexpr explicit forward_list(const Allocator&);
```

[...]

```
constexpr explicit forward_list(size_type n, const Allocator& = Allocator());
```

[...]

```
constexpr forward_list(size_type n, const T& value, const Allocator& = Allocator());
```

[...]

```
template<class InputIterator>
  constexpr forward_list(InputIterator first, InputIterator last,
                              const Allocator& = Allocator());
```

[...]

Change in [**forwardlist.capacity**] **22.3.9.3**:

```
constexpr iterator before_begin() noexcept;
constexpr const_iterator before_begin() const noexcept;
constexpr const_iterator cbefore_begin() const noexcept;
```

[...]

Change in [**forwardlist.access**] **22.3.9.4**:

```
constexpr reference front();
constexpr const_reference front() const;
```

[...]

Change in [**forwardlist.modifiers**] **22.3.9.5**:

```
template<class... Args> constexpr reference emplace_front(Args&&... args);

constexpr void push_front(const T& x);
constexpr void push_front(T&& x);

constexpr void pop_front();

constexpr iterator insert_after(const_iterator position, const T& x);
constexpr iterator insert_after(const_iterator position, T&& x);
constexpr iterator insert_after(const_iterator position, size_type n, const T& x);
template<class InputIterator>
  constexpr iterator insert_after(const_iterator position, InputIterator first, InputIterator last
constexpr iterator insert_after(const_iterator position, initializer_list<T>);

template<class... Args> constexpr iterator emplace_after(const_iterator position, Args&&... args);
```

[...]

```
constexpr iterator erase_after(const_iterator position);
constexpr iterator erase_after(const_iterator first, const_iterator last);

constexpr void resize(size_type sz);

constexpr void resize(size_type sz, const value_type& c);
constexpr void clear() noexcept;
```

Change in [**forwardlist.ops**] **22.3.9.6**:

```
constexpr void splice_after(const_iterator position, forward_list& x);
constexpr void splice_after(const_iterator position, forward_list&& x);
[...]
constexpr void splice_after(const_iterator position, forward_list& x, const_iterator i);
constexpr void splice_after(const_iterator position, forward_list&& x, const_iterator i);
[...]
constexpr void splice_after(const_iterator position, forward_list& x, const_iterator first,
const_iterator last);
constexpr void splice_after(const_iterator position, forward_list&& x, const_iterator first,
const_iterator last);
```

[...]

```
constexpr size_type remove(const T& value);
template<class Predicate> constexpr size_type remove_if(Predicate pred);
```

[...]

```
constexpr size_type unique();
template<class BinaryPredicate> constexpr size_type unique(BinaryPredicate binary_pred);
```

[...]

```
constexpr void merge(forward_list& x);
constexpr void merge(forward_list&& x);
template<class Compare> constexpr void merge(forward_list& x, Compare comp);
template<class Compare> constexpr void merge(forward_list&& x, Compare comp);
```

[...]

```
constexpr void sort();
template<class Compare> constexpr void sort(Compare comp);
```

[...]

```
constexpr void reverse() noexcept;
```

[...]

Change in [**forwardlist.erasure**] **22.3.9.7**:

```
template<class T, class Allocator, class U>
  constexpr void erase(forward_list<T, Allocator>& c, const U& value);

template<class T, class Allocator, class Predicate>
  constexpr void erase_if(forward_list<T, Allocator>& c, Predicate pred);
```

5 Implementation

Possible implementation can be found here: LLVM fork. Notice that when proposal was written constexpr destructors were not supported in Clang. Also in this implementation isn't used `operator<=>` - bunch of old operators used instead (just because libcxx at the moment doesn't use `operator<=>` for `std::forward_list`).

6 References

[P0784R1] Multiple authors, *Standard containers and constexpr*
`http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0784r1.html`