Iterators pair constructors for stack and queue

Document #:	P1425R1
Date:	2020-02-25
Project:	Programming Language C++
Audience:	LEWG, LWG

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

This paper proposes to add iterators-pair constructors to `std::stack` and `std::queue`

Tony tables

Before	After

`std::vector<int> v(42);`
`std::queue<int> q({v.begin(), v.end()});`
`std::stack<int> s({v.begin(), v.end()});`
	`std::vector<int> v(42);`
	`std::queue q(v.begin(), v.end());`
	`std::stack s(v.begin(), v.end());`

Motivation

`std::stack` and `std::queue` do not provide iterators based constructors which is inconsistent. This paper is an offshoot of [P1206], for which I conducted a review of existing containers and containers adapters constructors.

The lack of these constructors forces the implementation of `ranges::to` to special case container-adapters or to not support them. Their absence make it also impossible to deduce their type using CTAD.

While this is a a small change, we believe its impact on the standard is low and consistent designs are less surprising and therefore easier to use: with this change, all container-like types, whether they are *Containers* or container adapters, can be constructed from an iterators pair, making them more compatible with `ranges`.

Implementation

This proposal has been Implemented in libc++

Proposed Wording

🔲 Definition [queue.defn]

```
namespace std {
        template<class T, class Container = deque<T>>
        class queue {
                public:
                using value_type      = typename Container::value_type;
                using reference       = typename Container::reference;
                using const_reference = typename Container::const_reference;
                using size_type       = typename Container::size_type;
                using container_type  =          Container;

                protected:
                Container c;

                public:
                queue() : queue(Container()) {}
                explicit queue(const Container&);
                explicit queue(Container&&);

                template<class InputIterator>
                queue(InputIterator first, InputIterator last, const Container&);

                template<class InputIterator>
                queue(InputIterator first, InputIterator last, Container&& = Container());

                template<class Alloc> explicit queue(const Alloc&);
                template<class Alloc> queue(const Container&, const Alloc&);
                template<class Alloc> queue(Container&&, const Alloc&);
                template<class Alloc> queue(const queue&, const Alloc&);
                template<class Alloc> queue(queue&&, const Alloc&);

                //...
        };

        template<class Container>
        queue(Container) -> queue<typename Container::value_type, Container>;

        template<class InputIterator,
         class Container = deque<typename iterator_traits<InputIterator>::value_type>>
```

```
        queue(InputIterator, InputIterator, Container c = Container())
        -> queue<typename iterator_traits<InputIterator>::value_type, Container>;

        template<class Container, class Allocator>
        queue(Container, Allocator) -> queue<typename Container::value_type, Container>;

        template<class T, class Container>
        void swap(queue<T, Container>& x, queue<T, Container>& y) noexcept(noexcept(x.swap(y)));

        template<class T, class Container, class Alloc>
        struct uses_allocator<queue<T, Container>, Alloc>
        : uses_allocator<Container, Alloc>::type { };
}
```

🔲 Constructors [queue.cons]

```
explicit queue(const Container& cont);
```

Effects: Initializes `c` with `cont`.

```
explicit queue(Container&& cont);
```

Effects: Initializes `c` with `std::move(cont)`.

```
template<class InputIterator>
queue(InputIterator first, InputIterator last, const Container & cont );

template<class InputIterator>
queue(InputIterator first, InputIterator last, Container && cont );
```

Effects: Initializes `c` from `cont` (copy constructing or move constructing as appropriate); and calls `c.insert(c.end(), first, last);`

🔲 Definition [stack.defn]

```
namespace std {
        template<class T, class Container = deque<T>>
        class stack {
                public:
                using value_type      = typename Container::value_type;
                using reference       = typename Container::reference;
                using const_reference = typename Container::const_reference;
                using size_type       = typename Container::size_type;
                using container_type  = Container;

                protected:
                Container c;
```

```
        public:
        stack() : stack(Container()) {}
        explicit stack(const Container&);
        explicit stack(Container&&);

        template<class InputIterator>
        stack(InputIterator first, InputIterator last, const Container&);

        template<class InputIterator>
        stack(InputIterator first, InputIterator last, Container&& = Container());

        template<class Alloc> explicit stack(const Alloc&);
        template<class Alloc> stack(const Container&, const Alloc&);
        template<class Alloc> stack(Container&&, const Alloc&);
        template<class Alloc> stack(const stack&, const Alloc&);
        template<class Alloc> stack(stack&&, const Alloc&);

        //...
    };

    template<class Container>
    stack(Container) -> stack<typename Container::value_type, Container>;

    template<class InputIterator,
     class Container = deque<typename iterator_traits<InputIterator>::value_type>>
    stack(InputIterator, InputIterator, Container c = Container())
    -> stack<typename iterator_traits<InputIterator>::value_type, Container>;

    template<class Container, class Allocator>
    stack(Container, Allocator) -> stack<typename Container::value_type, Container>;

    template<class T, class Container, class Alloc>
    struct uses_allocator<stack<T, Container>, Alloc>
    : uses_allocator<Container, Alloc>::type { };
}
```

🔲 Constructors [stack.cons]

```
explicit stack(const Container& cont);
```

Effects: Initializes `c` with `cont`.

```
explicit stack(Container&& cont);
```

Effects: Initializes `c` with `std::move(cont)`.

```
template<class InputIterator>
stack(InputIterator first, InputIterator last, const Container & cont );

template<class InputIterator>
stack(InputIterator first, InputIterator last, Container && cont );
```

Effects: Initializes `c` from `cont` (copy constructing or move constructing as appropriate); and calls `c.insert(c.end(), first, last);`

Acknowledgment

Thanks to Eric Niebler who reviewed the wording

References

[P1206] Corentin Jabot *A function to convert any range to a container*
https://wg21.link/P1206