

Making `std::queue` constexpr

Document #: P0000
Date: 2019-09-23
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::queue` is not currently **constexpr** friendly. With the loosening of requirements on **constexpr** in [P0784R1] and related papers, we can now make `std::queue` **constexpr**, and we should in order to support the **constexpr** reflection effort (and other evident use cases).

3 Motivation

`std::queue` is not so widely-used standard container as `std::vector` or `std::string`. But there is no reason to keep `std::queue` 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::queue` available in compile-time.

4 Proposed wording

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

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

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

Change in [queue.syn] 22.6.2:

```

#include <initializer_list>

namespace std {
    template<class T, class Container = deque<T>> class queue;

    template<class T, class Container>
        constexpr bool operator==(const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, class Container>
        constexpr bool operator!=(const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, class Container>
        constexpr bool operator< (const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, class Container>
        constexpr bool operator> (const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, class Container>
        constexpr bool operator<=(const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, class Container>
        constexpr bool operator>=(const queue<T, Container>& x, const queue<T, Container>& y);
    template<class T, three_way_comparable Container>
        constexpr compare_three_way_result_t<Container>
            operator<=>(const queue<T, Container>& x, const queue<T, Container>& y);

    template<class T, class Container>
        constexpr 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>;
}

[...]
```

Change in [queue.defn] 22.6.4.1:

```

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:
        constexpr queue() : queue(Container()) {}
        constexpr explicit queue(const Container&);
        constexpr explicit queue(Container&&);
        template<class Alloc> constexpr explicit queue(const Alloc&);
        template<class Alloc> constexpr queue(const Container&, const Alloc&);
        template<class Alloc> constexpr queue(Container&&, const Alloc&);
```

```

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

[[nodiscard]] constexpr bool empty() const { return c.empty(); }
constexpr size_type size() const { return c.size(); }
constexpr reference front() { return c.front(); }
constexpr const_reference front() const { return c.front(); }
constexpr reference back() { return c.back(); }
constexpr const_reference back() const { return c.back(); }
constexpr void push(const value_type& x) { c.push_back(x); }
constexpr void push(value_type&& x) { c.push_back(std::move(x)); }
template<class... Args>
    constexpr decltype(auto) emplace(Args&&... args)
    { return c.emplace_back(std::forward<Args>(args)...); }
constexpr void pop() { c.pop_front(); }
constexpr void swap(queue& q) noexcept(is_nothrow_swappable_v<Container>)
    { using std::swap; swap(c, q.c); }
};

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

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

template<class T, class Container>
    constexpr 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 { };
}

```

Change in [queue.cons] 22.6.4.2:

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

[...]

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

Change in [queue.cons.alloc] 22.6.4.3:

```
template<class Alloc> constexpr explicit queue(const Alloc& a);
```

[...]

```
template<class Alloc> constexpr queue(const container_type& cont, const Alloc& a);
```

[...]

```
template<class Alloc> constexpr queue(container_type&& cont, const Alloc& a);
```

[...]

```
template<class Alloc> constexpr queue(const queue& q, const Alloc& a);
```

[...]

```
template<class Alloc> constexpr queue(queue&& q, const Alloc& a);
```

[...]

Change in [queue.ops] 22.6.4.4:

```
template<class T, class Container>
constexpr bool operator==(const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

```
template<class T, class Container>
constexpr bool operator!=(const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

```
template<class T, class Container>
constexpr bool operator< (const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

```
template<class T, class Container>
constexpr bool operator> (const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

```
template<class T, class Container>
constexpr bool operator<=(const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

```
template<class T, class Container>
constexpr bool operator>=(const queue<T, Container>& x,
                          const queue<T, Container>& y);
```

[...]

```
template<class T, three_way_comparable Container>
constexpr compare_three_way_result_t<Container>
operator<=>(const queue<T, Container>& x, const queue<T, Container>& y);
```

[...]

Change in [queue.special] 22.6.4.5:

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

5 Implementation

Possible implementation can be found here: [LLVM fork](#). Notice that when proposal was written constexpr destructors were not supported in Clang.

6 References

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