Document Number: P4343R1

Date: 2026-09-10

Audience: Library Evolution Working Group (LEWG)

Author: Shubham Avasthi

Reply-to: shubhamavasthi1@gmail.com

Adding clear() to Container Adaptors

1. Abstract

This paper proposes adding a clear() member function to the standard container adaptors (std::priority_queue, std::queue, and std::stack). This addition allows developers to empty the contents of an adaptor while preserving the memory capacity of its underlying container, thereby avoiding unnecessary dynamic memory reallocations. To ensure strict backward compatibility with custom underlying containers, this method is conditionally enabled via a C++20 requires clause.

2. Revision History

- **R1:** Expanded the Motivation section to highlight API symmetry with the C++23 push_range feature and standard library consistency with modern C++23 flat container adaptors. Added Acknowledgements.
- **R0:** Initial submission. Discussed on the std-proposals mailing list with positive feedback regarding zero-cost implementation and lack of edge cases.

3. Motivation

Currently, there is no standardized, zero-overhead way to clear the elements of container adaptors. When developers need to reuse a std::priority_queue or std::queue in a performance-critical loop, they are forced to use one of two suboptimal workarounds:

- **Workaround 1: Looping and Popping:** Executing a `while (!pq.empty()) pq.pop();` iteration over a std::priority_queue compels the adaptor to restore heap invariants upon each removal step, incurring an unnecessary O(N log N) runtime overhead.
- **Workaround 2: Reassignment:** Assigning a newly constructed instance via `pq = {};` discards existing items immediately, yet it deallocates the internal storage capacity entirely. Consequently, upcoming push operations must perform costly memory allocations, running counter to C++ design philosophy.
- **API Symmetry with Bulk Operations:** In C++23, WG21 adopted push_range (P1206) to eliminate the anti-pattern of writing loops to insert multiple elements into a container. By delegating bulk insertions to the underlying container's optimized methods, push_range established a clear precedent: standard adaptors should not force users to write iteration loops for bulk state changes. Adding clear() completes this paradigm by providing the logical bulk-erase counterpart.

- **Parity with Modern C++ Adaptors:** The absence of clear() in classic adaptors presents a noticeable asymmetry across the standard library. C++23 introduced flat container adaptors (std::flat_set, std::flat_multiset, std::flat_map, and std::flat_multimap), each providing a dedicated clear() member function. Equipping classic adaptors with clear() aligns them with contemporary library interface standards.

4. Design Decisions

We propose adding a clear() member function to std::priority_queue, std::queue, and std::stack.

- **Conditional Compilation:** To prevent breaking legacy code that uses custom SequenceContainers lacking a .clear() method, the proposed function is constrained using a requires clause. It will only participate in overload resolution if c.clear() is a valid expression.
- **constexpr Support:** In alignment with C++20's push to make standard containers usable at compile-time, the clear() method is marked constexpr.
- **Exception Specification:** The method unconditionally delegates to the underlying container's clear() method and perfectly propagates its noexcept specification.

5. Proposed Wording

The proposed changes are relative to the current working draft of the C++ Standard.

Modify [queue.defn] (Queue definition):

```
namespace std {
  template<class T, class Container = deque<T>>
  class queue {
  public:
    // ... existing members ...
    constexpr void pop() { c.pop_front(); }

    constexpr void clear() noexcept(noexcept(c.clear()))
      requires requires { c.clear(); }
    {
      c.clear();
    }
  };
}
```

Modify [priority.queue] (Priority queue definition):

```
namespace std {
  template<class T, class Container = vector<T>,
           class Compare = less<typename Container::value_type>>
```

```
  class priority_queue {
  public:
    // ... existing members ...
    constexpr void pop();

    constexpr void clear() noexcept(noexcept(c.clear()))
      requires requires { c.clear(); }
    {
      c.clear();
    }
  };
}
```

Modify [stack.defn] (Stack definition):

```
namespace std {
  template<class T, class Container = deque<T>>
  class stack {
  public:
    // ... existing members ...
    constexpr void pop() { c.pop_back(); }

    constexpr void clear() noexcept(noexcept(c.clear()))
      requires requires { c.clear(); }
    {
      c.clear();
    }
  };
}
```

6. Acknowledgements

I would like to thank Joe Gottman (joegottman@verizon.net) for his valuable feedback and for suggesting the consistency argument regarding C++23 flat container adaptors.