Document Number: P4343R0

Date: 2026-08-11

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

- **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

```
while (!pq.empty()) {
    pq.pop();
}
```

For a std::priority_queue, this is highly inefficient. It forces the container to rebuild the heap structure on every single pop, resulting in a time complexity of O(N log N).

Workaround 2: Reassignment

```
pq = std::priority_queue<T>();
```

While this operation clears the elements quickly, it completely destroys the underlying container and its allocated memory. Subsequent insertions will trigger expensive dynamic memory reallocations, violating the core C++ principle of zero-cost abstractions.

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();
    }
  };
}
```