Add std::priority queue method to move the top element from it

Document #:	
Date:	2026-01-17
Project:	Programming Language C++
Audience:	LEWG
Reply-to:	Aleksei Sidorin <alexey.v.sidorin@yandex.ru>

Contents

1 Introduction

Unlike other containers, `std::priority_queue` doesn't allow its users to move elements out of it. This asymmetry causes performance hits and limits the type usage. I propose adding a corresponding method to `std::priority_queue` backed by a set of freestanding functions in `<algorithm>` header.

2 Motivation and Scope

In order to extract the top element from `std::priority_queue`, STL users have to write code similar to the following:

```
auto x = queue.top(); // Copy constructor or copy assignment.
queue.pop();
```

Unfortunately, we cannot move a top value from the queue because `top()` is of `const_reference` type ([N4950], Section 24.6.7.4 `[priqueue.members]`).

The first problem is that this can cause a performance hit if the value type is expensive to copy (a structure containing large strings, allocating copy constructors, etc.).

The second problem is that this also makes impossible to use `std::priority_queue` with move-only types like `std::unique_ptr`.

I've made a search in chromium sources showing that almost all heap structure usage have a similar pattern:

```
std::pop_heap(queue.begin(), queue.end());
*event = std::move(queue.back());
queue.pop_back();
```

(https://github.com/search?q=repo%3Achromium%2Fchromium%20pop_heap&type=code)

I think this clearly indicates that STL can have a better interface here.

Another issue with `top()`& `pop()` approach is that `pop()` uses `std::pop_heap()` + `c.pop_back()` call pair internally. And `std::pop_heap()` is required to move the top element to the sequence end. This can introduce stores hard to elide by compilers. For example, in https://godbolt.org/z/r7KK7MYzc we can find such a store in assembly line 13 left even with `-O3`.

Therefore, I propose adding an `std::priority_queue` method behaving like pseudo-code below:

```
T extract_top() {
  T val = std::move_if_noexcept(container.front());
  pop();
  return val;
}
```

The method is named `extract_top()` in this proposal, but the exact naming can be a subject for discussion.

In addition, I propose adding freestanding `<algorithm>` functions: `std::extract_heap_top()` and `std::ranges::extract_heap_top()`. These functions will return the queue top value and remove it from the queue with restoring heap property of the data structure. Unlike `std::pop_heap()`, these functions leave the last range element in an unspecified state, so handling range length after calling them lies on users.

3 Impact On the Standard

The change is a pure library extension which can be implemented with existing language features and current C++ compilers.

4 Design decisions

A common discussion point for this proposal and for its predecessors was exception safety. This paper proposes to apply `move_if_noexcept` behaviour when extracting the top value, so the data structure will keep its consistent state even if move constructor can throw.

It is often pointed that the `priority_queue` internal bookkeeping also involves non-trivial operations, but this proposal does not affect this maintenance, so we do not discuss it.

5 Technical specification

— Add an entry into chapter 27.8.8 `[alg.heap.operations]`: `[extract_top.heap]`

```
template <class RandomAccessIterator>
  constexpr iterator_traits_t<RandomAccessIterator>::value_type
    extract_heap_top(RandomAccessIterator first, RandomAccessIterator last);

template <class RandomAccessIterator, class Compare>
  constexpr iterator_traits_t<RandomAccessIterator>::value_type
    extract_heap_top(RandomAccessIterator first, RandomAccessIterator last, Compare comp);

template<random_access_iterator I, sentinel_for<I> S, class Comp = ranges::less,
         class Proj = identity>
  requires sortable<I, Comp, Proj>
  constexpr iterator_traits<I>::value_type
    ranges::extract_heap_top(I first, S last, Comp comp = {}, Proj proj = {});

template<random_access_range R, class Comp = ranges::less, class Proj = identity>
  requires sortable<iterator_t<R>, Comp, Proj>
  constexpr iterator_traits<iterator_t<R>>::value_type
    ranges::extract_heap_top(R&& r, Comp comp = {}, Proj proj = {});
```

1. Let `comp` be `less{}` and `proj` be `identity{}` for the overloads with no parameters by those names.
2. *Preconditions:* The range `[first, last)` is a valid non-empty heap with respect to `comp` and `proj`. For the overloads in namespace `std`, `RandomAccessIterator` meets the *Cpp17ValueSwappable* requirements (16.4.4.3) and the type of `*first` meets the *Cpp17MoveConstructible* (Table 31) and *Cpp17MoveAssignable* (Table 33) requirements.
3. *Effects:* Replaces the value in the location `first` with the value in the location `last - 1` and makes `[first, last - 1)` into a heap with respect to `comp` and `proj`.
4. *Returns:* the initial value in the location `first`.
5. *Complexity:* At most 2 log(`last - first`) comparisons and twice as many projections.

— Add an entry into chapter 24.6.7.4 `[priqueue.members]`:

```
T extract_top();
```

Effects: As if by:

```
auto result = extract_heap_top(c.begin(), c.end(), comp);
c.pop_back();
return result;
```

6 Reference implementation

I made a reference implementation of the proposed additions in `libc++`. The diff is available as a Github pull request and a branch in my repository copy.

7 Acknowledgments

Thanks to:

— Anton Polukhin for initial proposal proof-read
— Arthur O'Dwyer for suggesting `move_if_noexcept`

8 Related work

https://lists.isocpp.org/std-proposals/2021/02/2390.php - a similar proposal. No actions were taken after publishing it to std-proposals.

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p3182r1.html proposes adding `pop_value` and `push_value` methods to many STL containers. Unfortunately, the author (Brian Bi) confirmed that it is stalled.

9 References

[N4950] Thomas Köppe. 2023-05-10. Working Draft, Standard for Programming Language C++.
https://wg21.link/n4950