**Container Adapters in Detail**

The following subsections describe the members and operations of the container adapters stack<>, queue<>, and priority\_queue<> in detail.

# Type Definitions

## contadapt::value\_type

The type of the elements.

It is equivalent to container::value\_type.

## contadapt::reference

The type of element references.

It is equivalent to container::reference.

Available since C++11.

## contadapt::const\_reference

The type of read-only element references.

It is equivalent to container::const\_reference.

Available since C++11.

## contadapt::size\_type

The unsigned integral type for size values.

It is equivalent to container::size\_type.

## contadapt::container\_type

The type of the container.

# Constructors

## Default

**contadapt::contadapt ()**

The default constructor. Creates an empty stack or (priority) queue.

## Copy and move

**explicit contadapt::contadapt (const Container& cont)**

**explicit contadapt::contadapt (Container&& cont)**

Creates a stack or queue that is initialized by the elements of cont, which has to be an object of the container type of the container adapter.

* With the first form, all elements of cont are copied.
* With the second form, all elements of cont are moved if the passed container provides move semantics; otherwise, they are copied (available since C++11).
* Both forms are not provided for priority\_queue<>.

Since C++11, all constructors allow you to pass an allocator as additional argument, which is used to initialize the allocator of the internal container.

# Supplementary Constructors for Priority Queues

## explicit priority\_queue::priority\_queue (const CompFunc& op)

Creates an empty priority queue with op used as the sorting criterion.

Check examples that demonstrate how to pass a sorting criterion as a constructor argument.

## priority\_queue::priority\_queue (const CompFunc& op, const Container& cont)

Creates a priority queue that is initialized by the elements of cont and that uses op as the sorting criterion.

All elements of cont are copied.

## priority\_queue::priority\_queue (InputIterator beg, InputIterator end)

Creates a priority queue that is initialized by all elements of the range [beg,end).

This function is a member template, so the elements of the source range might have any type that is convertible into the element type of the container.

## priority\_queue::priority\_queue (InputIterator beg, InputIterator end, const CompFunc& op)

Creates a priority queue that is initialized by all elements of the range [beg,end) and that uses op as the sorting criterion.

This function is a member template, so the elements of the source range might have any type that is convertible into the element type of the container.

Check examples that demonstrate how to pass a sorting criterion as a constructor argument.

## priority\_queue::priority\_queue (InputIterator beg, InputIterator end, const CompFunc& op, const Container& cont)

Creates a priority queue that is initialized by all elements of the container cont plus all elements of the range [beg,end) and that uses op as the sorting criterion.

This function is a member template, so the elements of the source range might have any type that is convertible into the element type of the container.

Since C++11, all constructors allow you to pass an allocator as additional argument, which is used to initialize the allocator of the internal container.

# Operations

## Size and empty

**bool contadapt::empty () const**

Returns whether the container adapter is empty (contains no elements).

It is equivalent to contadapt::size()==0 but might be faster.

**size\_type contadapt::size () const**

Returns the current number of elements.

To check whether the container adapter is empty (contains no elements), use empty() because it might be faster.

## Push

**void contadapt::push (const value\_type& elem)**

**void contadapt::push (value\_type&& elem)**

The first form inserts a copy of elem.

The second form moves elem if move semantics are provided; otherwise, it copies elem (available since C++11).

**void contadapt::emplace (args)**

Inserts a new element, which is initialized by the argument list args.

Available since C++11.

## Top and front

**reference contadapt::top ()**

**const\_reference contadapt::top () const**

**reference contadapt::front () // for queue only**

**const\_reference contadapt::front () const // for queue only**

All forms, if provided, return the next element.

1. For a stack, both forms of top() are provided, which return the element that was inserted last.
2. For a queue, both forms of front() are provided, which return the element that was inserted first.
3. For a priority queue, only the second form of top() is provided, which yields the element with the maximum value. If more than one element has the maximum value, it is undefined which element it returns.

The caller has to ensure that the container adapter contains an element (size()>0); otherwise, if size is 0 the behavior is undefined.

The forms that return a nonconstant reference allow you to modify the next element while it is in the stack/queue. It is up to you to decide whether this is good style.

Before C++11, the return type was (const) value\_type&, which usually should be the same.

## Pop

**void contadapt::pop ()**

Removes the next element from the container adapter.

1. For a stack, the next element is the one that was inserted last.
2. For a queue, the next element is the one that was inserted first.
3. For a priority queue, the next element is the one with the maximum value.

If more than one element has the maximum value, it is undefined which element it removes.

This function has no return value. To process this next element, you must call top() or front() first.

The caller must ensure that the container adapter contains an element (size()>0); otherwise, if size is 0 the behavior is undefined.

## Back

**reference queue::back ()**

**const\_reference queue::back () const**

Provided for queue<> only.

Both forms return the last element of a queue. The last element is the one that was inserted after all other elements in the queue.

The caller must ensure that the queue contains an element (size()>0); otherwise, the behavior is undefined.

The first form for nonconstant queues returns a reference. Thus, you could modify the last element while it is in the queue.

Before C++11, the return type was (const) value\_type&, which usually should be the same.

## Comparison

**bool comparison (const contadapt& stack1, const contadapt& stack2)**

Returns the result of the comparison of two stacks or queues of the same type.

comparison might be any of the following operators:

operators == and !=

operators <, >, <=, and >=

Two stacks or queues are equal if they have the same number of elements and contain the same elements in the same order (all comparisons of two corresponding elements must yield true).

The container adapters are compared lexicographically.

Not provided for priority\_queue<>.

## Swap

**void contadapt::swap (contadapt& c) // memberfunction**

**void swap (contadapt& c1, contadapt& c2) // algorithm**

Swaps the contents of \*this with c or c1 with c2, respectively.

For priority queues, it also swaps the sorting criterion.

Calls swap() for the corresponding container.

Available since C++11.

# END