# deque Class

Arranges elements of a given type in a linear arrangement and, like a vector, enables fast random access to any element, and efficient insertion and deletion at the back of the container. However, unlike a vector, the **deque** class also supports efficient insertion and deletion at the front of the container.

template <  
 class Type,   
 class Allocator=allocator<Type>   
>  
 class deque

## Parameters

*Type*

The element data type to be stored in the deque.

*Allocator*

The type that represents the stored allocator object that encapsulates details about the deque's allocation and deallocation of memory. This argument is optional, and the default value is **allocator<Type>***.*

## Remarks

The choice of container type should be based in general on the type of searching and inserting required by the application. Vectors should be the preferred container for managing a sequence when random access to any element is at a premium and insertions or deletions of elements are only required at the end of a sequence. The performance of the [list](http://msdn.microsoft.com/en-us/library/802d66bt.aspx) container is superior when efficient insertions and deletions (in constant time) at any location within the sequence is at a premium. Such operations in the middle of the sequence require element copies and assignments proportional to the number of elements in the sequence (linear time).

Deque reallocation occurs when a member function must insert or erase elements of the sequence:

* If an element is inserted into an empty sequence, or if an element is erased to leave an empty sequence, then iterators earlier returned by [begin](http://msdn.microsoft.com/en-us/library/z50625f7.aspx) and [end](http://msdn.microsoft.com/en-us/library/zz7bd328.aspx) become invalid.
* If an element is inserted at the first position of the deque, then all iterators, but no references, that designate existing elements become invalid.
* If an element is inserted at the end of the deque, then [end](http://msdn.microsoft.com/en-us/library/zz7bd328.aspx) and all iterators, but no references, that designate existing elements become invalid.
* If an element is erased at the front of the deque, only that iterator and references to the erased element become invalid.
* If the last element is erased from the end of the deque, only that iterator to the final element and references to the erased element become invalid.

Otherwise, inserting or erasing an element invalidates all iterators and references.

### Constructors

|  |  |
| --- | --- |
| deque | Constructs a **deque.** Several constructors are provided to set up the contents of the new **deque** in different ways: empty; loaded with a specified number of empty elements; contents moved or copied from another **deque**; contents copied or moved by using an iterator; and one element copied into the **deque***\_Count* times. Some of the constructors enable using a custom **allocator** to create elements. |

### Typedefs

|  |  |
| --- | --- |
| allocator\_type | A type that represents the **allocator** class for the **deque** object. |
| const\_iterator | A type that provides a random-access iterator that can access and read elements in the **deque** as const |
| const\_pointer | A type that provides a pointer to an element in a **deque** as a const. |
| const\_reference | A type that provides a reference to an element in a **deque** for reading and other operations as a const. |
| const\_reverse\_iterator | A type that provides a random-access iterator that can access and read elements in the **deque** as const. The deque is viewed in reverse. For more information, see reverse\_iterator Class |
| difference\_type | A type that provides the difference between two random-access iterators that refer to elements in the same **deque**. |
| iterator | A type that provides a random-access iterator that can read or modify any element in a **deque**. |
| pointer | A type that provides a pointer to an element in a **deque**. |
| reference | A type that provides a reference to an element stored in a **deque**. |
| reverse\_iterator | A type that provides a random-access iterator that can read or modify an element in a **deque**. The deque is viewed in reverse order. |
| size\_type | A type that counts the number of elements in a **deque**. |
| value\_type | A type that represents the data type stored in a **deque**. |

### Member Functions

|  |  |
| --- | --- |
| assign | Erases elements from a **deque** and copies a new sequence of elements to the target **deque**. |
| at | Returns a reference to the element at a specified location in the **deque**. |
| back | Returns a reference to the last element of the **deque**. |
| begin | Returns a random-access iterator addressing the first element in the **deque**. |
| deque::cbegin | Returns a const iterator to the first element in the **deque**. |
| deque::cend | Returns a random-access const iterator that points just beyond the end of the **deque**. |
| clear | Erases all the elements of a **deque**. |
| deque::crbegin | Returns a random-access const iterator to the first element in a **deque** viewed in reverse order. |
| deque::crend | Returns a random-access const iterator to the first element in a **deque** viewed in reverse order. |
| deque::emplace | Inserts an element constructed in place into the **deque** at a specified position. |
| deque::emplace\_back | Adds an element constructed in place to the end of the **deque**. |
| deque::emplace\_front | Adds an element constructed in place to the start of the **deque**. |
| empty | Returns true if the **deque** contains zero elements, and false if it contains one or more elements. |
| end | Returns a random-access iterator that points just beyond the end of the **deque**. |
| erase | Removes an element or a range of elements in a **deque** from specified positions. |
| front | Returns a reference to the first element in a **deque**. |
| get\_allocator | Returns a copy of the **allocator** object that is used to construct the **deque**. |
| insert | Inserts an element, several elements, or a range of elements into the **deque** at a specified position. |
| max\_size | Returns the maximum possible length of the **deque**. |
| pop\_back | Erases the element at the end of the **deque**. |
| pop\_front | Erases the element at the start of the **deque**. |
| push\_back | Adds an element to the end of the **deque**. |
| push\_front | Adds an element to the start of the **deque**. |
| rbegin | Returns a random-access iterator to the first element in a reversed **deque**. |
| rend | Returns a random-access iterator that points just beyond the last element in a reversed **deque**. |
| resize | Specifies a new size for a **deque**. |
| deque::shrink\_to\_fit | Discards excess capacity. |
| size | Returns the number of elements in the **deque**. |
| swap | Exchanges the elements of two **deque**s. |

### Operators

|  |  |
| --- | --- |
| operator[] | Returns a reference to the **deque** element at a specified position. |
| deque::operator= | Replaces the elements of the **deque** with a copy of another **deque**. |

## Requirements

**Header**: <deque>