# Deque

- At the end of this topic, you will be able to perform the following with deque containers
  - Know when to use a deque
  - Perfom on a deque the same operations as a vector
  - Perform push_front and pop_front

- Class `deque` provides many of the benefits of a `vector` and a `list` in one container
- The term `deque` is short for "double-ended queue"
- Implemented to provide efficient indexed access (using subscripting) for reading and modifying its elements, much like a `vector`
- Also implemented for efficient insertion and deletion operations at its front and back, much like a `list` (although a `list` is also capable of efficient insertions and deletions in the middle of the `list`)
- Provides support for random-access iterators
  - So `deque`s can be used with all Standard Library algorithms

## Performance Tip 15.8

In general, `deque` has higher overhead than `vector`.

## Performance Tip 15.9

Insertions and deletions in the middle of a `deque` are optimized to minimize the number of elements copied, so it's more efficient than a `vector` but less efficient than a `list` for this kind of modification.

- One of the most common uses of a `deque` is to maintain a first-in, first-out queue of elements.
  - A `deque` is the default underlying implementation for the `queue` adaptor
- Additional storage for a `deque` can be allocated at either end of the `deque` in blocks of memory
  - Typically maintained as a built-in array of pointers to those blocks
- Due to the *noncontiguous memory layout* of a `deque`
  - A `deque` iterator must be more "intelligent" than the pointers that are used to iterate through `vector`s, `array`s or built-in arrays

- Class `deque` provides the same basic operations as class `vector`
  - But like `list` adds member functions `push_front` and `pop_front` to allow insertion and deletion at the beginning of the `deque`

- Figure 15.14 demonstrates features of class `deque`

- Header `<deque>` must be included to use class `deque`

- `push_front` and `push_back`
  - Insert elements at the beginning and end of the `deque`

- Using function `size`
  - Ensures that we do not attempt to access an element *outside* the bounds of the `deque`

- Function `pop_front`
  - Removes the first element of a `deque`
- Deques support random access so the subscript operator obtains an *lvalue*
  - Enabling values to be assigned directly to any element of the `deque`

- This topic taught and demonstrated how to perform the following with deque containers
  - Know when to use a deque
  - Perfom on a deque the same operations as a vector
  - Perform push_front and pop_front