# Double-ended queue Code Library

## Description

Double-ended queue is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail). It is also called a head-tail linked list, though properly this refers to a specific data structure implementation. There are at least two common ways to efficiently implement a deque: with a modified dynamic array or with a double linked list.

## Deque implementation

Deques are often implemented using doubly linked lists to which they are closely related. In fact another name for a deque is a 'head-tail linked list'. The main difference between the two is that you can insert elements into or remove elements from the middle of the linked list as well as at the end points.

One List<T> represents the front and the other represents the back of the deque. Provided that there is always at least one element in both lists, then this approach is about three times faster than using a doubly linked list.

What I decided to do is to return the first element of the other list and mark it for deletion but not actually delete it. Deletion is a relatively expensive operation because all the other elements of the list need to be moved down in memory.  Also simply marking it for deletion is better than it sounds because the capacity of the list (i.e. the maximum number of elements it can accommodate) is never reduced automatically when deletions take place. So having deleted items still sitting in memory does no immediate harm.

However, it does do some longer  term harm because it reduces the time needed before the capacity of the list next needs to be increased (a very expensive operation) and, in the case of reference type elements,  increases  the time before they can be garbage collected if there are no other references to them.

Consequently, whenever a new item is added to the deque, if the deque would otherwise need resizing, any deleted items are then removed completely. This postpones the resizing operation and may even prevent it altogether. If there is more than one deleted item, it also means that the other items only need to be moved down in memory once rather than each time a deletion occurs.

## Deque methods

|  |  |
| --- | --- |
| **Method name and Signature** | **Description** |
| void PushFirst(T element) | Inserts the element at the front of Deque. |
| void PushLast(T element) | Inserts the element at the back of Deque. |
| T PopFirst() | Returns and removes the element at the front of Deque. |
| T PopLast() | Returns and removes the element at the back of Deque. |
| T PeekFirst() | Returns the element at the front of Deque without removing it. |
| T PeekLast() | Returns the element at the back of Deque without removing it. |
| void Clear() | Remove all elements from Deque. |
| bool Contains(T element) | Indicates if this element is within the Deque. |

## Deque properties

|  |  |
| --- | --- |
| **Property name and Signature** | **Description** |
| **int Count** | Gets the total number of elements in Deque |