Data Structures & Algorithms

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

facebook.com/cs.MohamedAyman

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 6

Deque

Array Based

Course Roadmap

Part 1: Linear Data Structures

Lecture 1: Complexity Analysis & Recursion

Lecture 2: Arrays

Lecture 3: Linked List

Lecture 4: Stack

Lecture 5: Queue

Lecture 6: Deque

Lecture 7: STL in C++ (Linear Data Structures)

Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to Deque

2- Insertion Operation

3- Deletion Operation

4- Front & Back Operations

5- Traverse Operation

6- Time Complexity & Space Complexity


Let's
STARTUP

Lecture Agenda

Section 1: Introduction to Deque

Section 2: Insertion Operation

Section 3: Deletion Operation

Section 4: Front & Back Operations

Section 5: Traverse Operation

Section 6: Time Complexity & Space Complexity

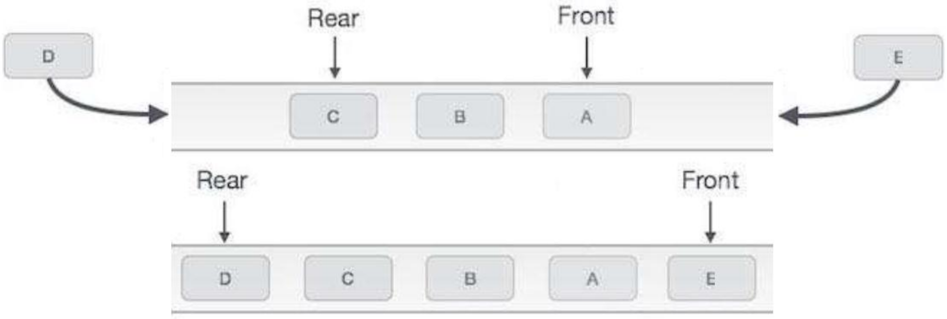
Introduction to Deque

- A double-ended queue (abbreviated to deque) 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 often called a head-tail linked list.
- A deque, also known as a double-ended queue, is an ordered collection of items similar to the queue. It has two ends, a front and a rear, and the items remain positioned in the collection. What makes a deque different is the unrestrictive nature of adding and removing items. New items can be added at either the front or the rear. Likewise, existing items can be removed from either end. In a sense, this hybrid linear structure provides all the capabilities of stacks and queues in a single data structure.
- A deque, also known as a double-ended queue where by it has two ends, a front and a rear, and the items remain positioned in the collection. New items can be added or removed at either the front or the rear. It does not require the LIFO and FIFO orderings that are enforced by those data structures. It is up to you to make consistent use of the addition and removal operations.

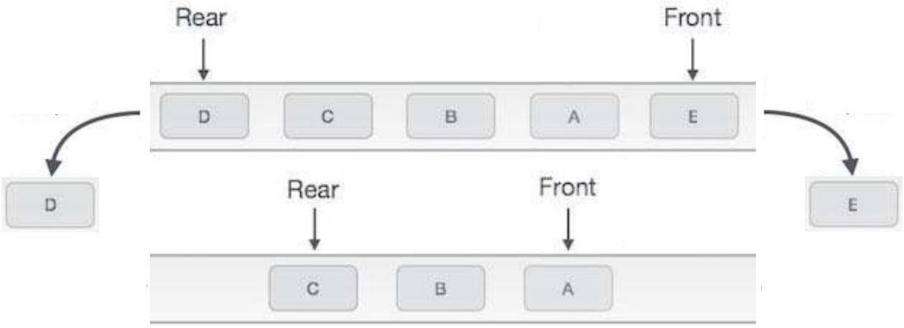
Introduction to Deque

- Deque operations may involve initializing or defining the deque, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with deques

- insert begin() - adds a new item to the front of the deque.
- insert end() - adds a new item to the rear of the deque.
- delete begin() - removes the front item from the deque.
- delete end() - removes the rear item from the deque.



Deque Push



Deque Pop

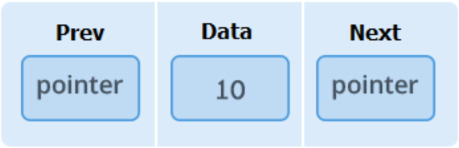
Introduction to Deque

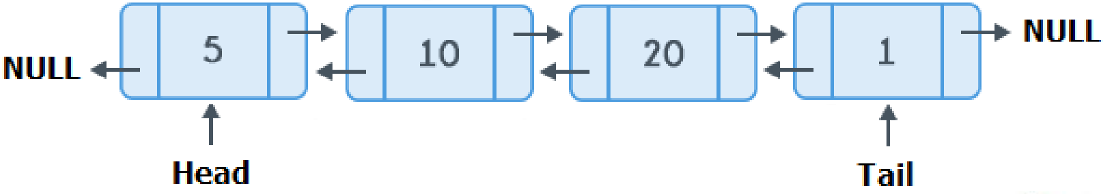
- Following are the basic operations supported by a deque.

- Insert (begin & end) - adds a new item to the front & rear of the deque.
- Delete (begin & end) - removes the front & rear item from the deque.
- Front & Back: which gets the first element and the last element in the queue.

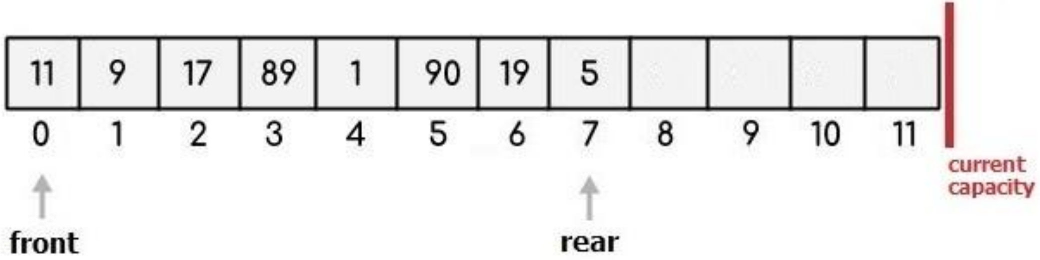
- Deque Types

1. Deque (Linked List Based)

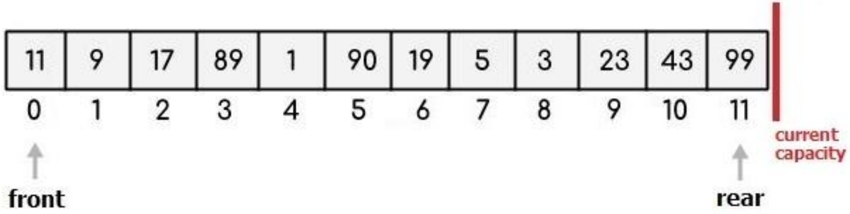




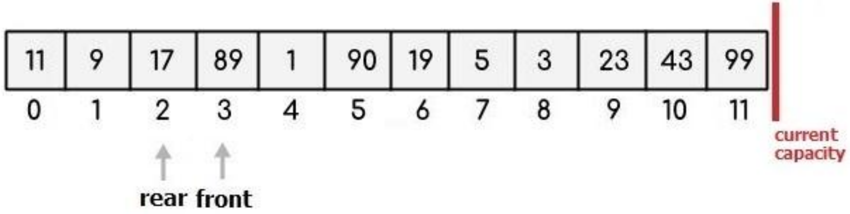
2. Deque (Array Based)



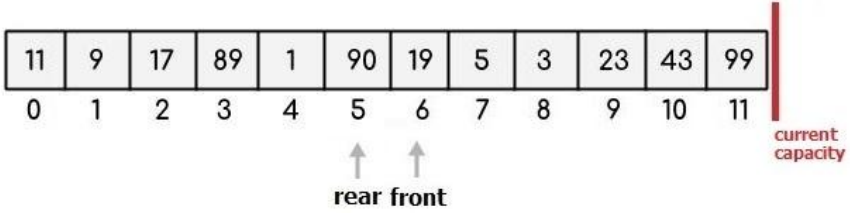
Deque (Array Based)



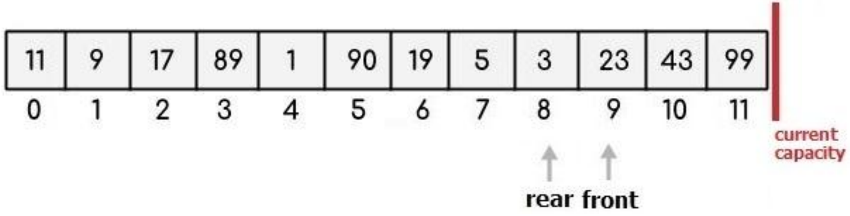
11, 9, 17, 89, 1, 90, 19, 5, 3, 23, 43, 99



89, 1, 90, 19, 5, 3, 23, 43, 99, 11, 9, 17

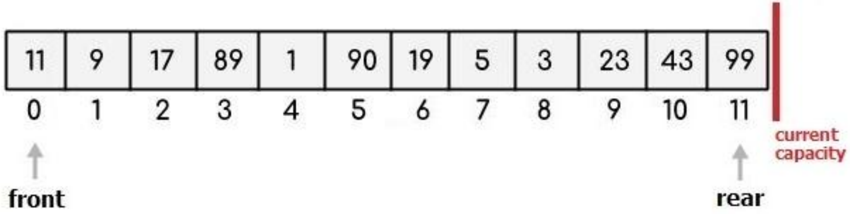


19, 5, 3, 23, 43, 99, 11, 9, 17, 89, 1, 90

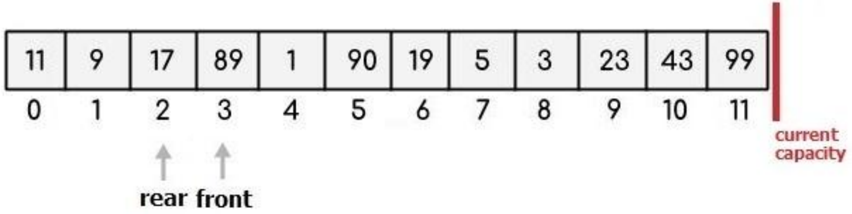


23, 43, 99, 11, 9, 17, 89, 1, 90, 19, 5, 3

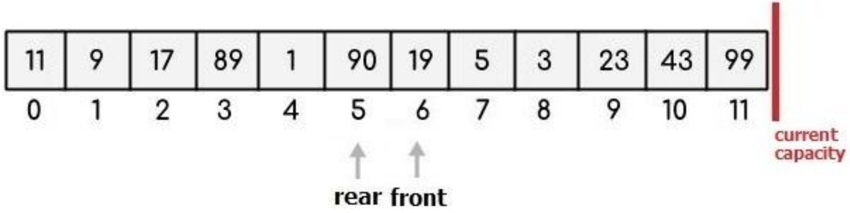
Deque (Array Based)



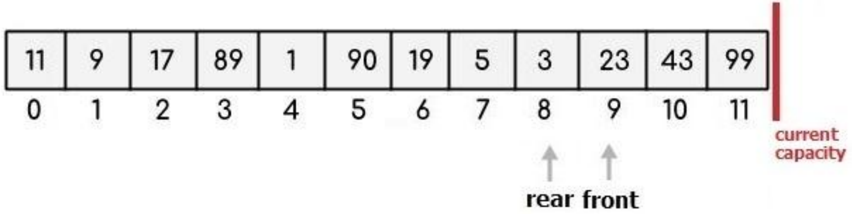
$$\text{Length} = (\text{rear} - \text{front} + \text{capacity}) \,\%\, \text{capacity} + 1$$
$$= (11 - 0 + 12) \,\%\, 12 + 1 = 12$$



$$\text{Length} = (\text{rear} - \text{front} + \text{capacity}) \,\%\, \text{capacity} + 1$$
$$= (2 - 3 + 12) \,\%\, 12 + 1 = 12$$

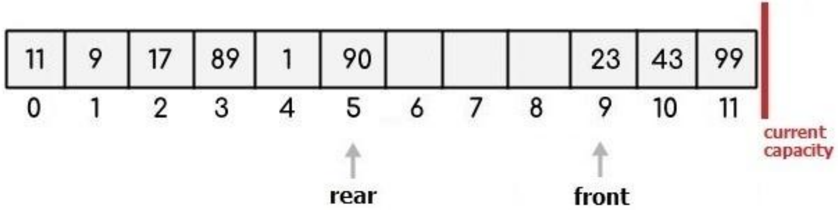


$$\text{Length} = (\text{rear} - \text{front} + \text{capacity}) \,\%\, \text{capacity} + 1$$
$$= (5 - 6 + 12) \,\%\, 12 + 1 = 12$$

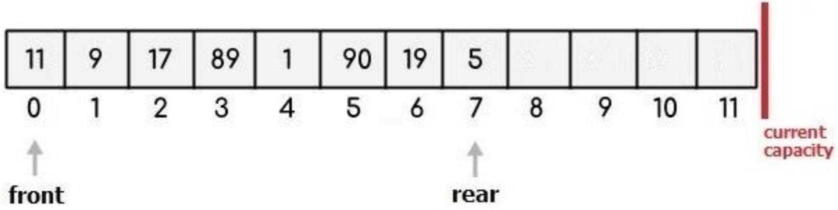


$$\text{Length} = (\text{rear} - \text{front} + \text{capacity}) \,\%\, \text{capacity} + 1$$
$$= (8 - 9 + 12) \,\%\, 12 + 1 = 12$$

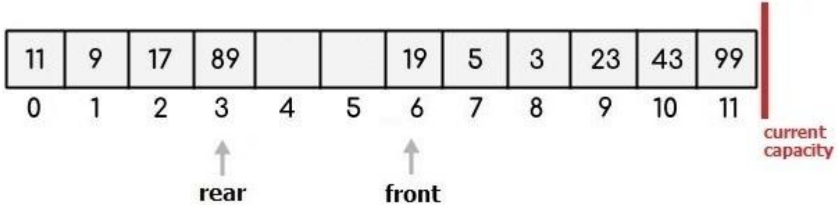
Deque (Array Based)



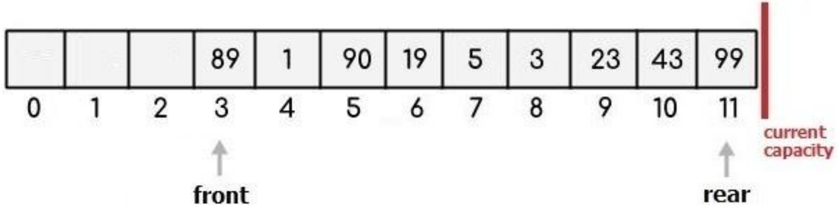
23, 43, 99, 11, 9, 17, 89, 1, 90



11, 9, 17, 89, 1, 90, 19, 5

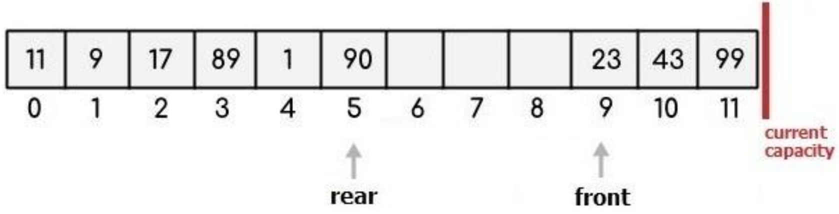


19, 5, 3, 23, 43, 99, 11, 9, 17, 89

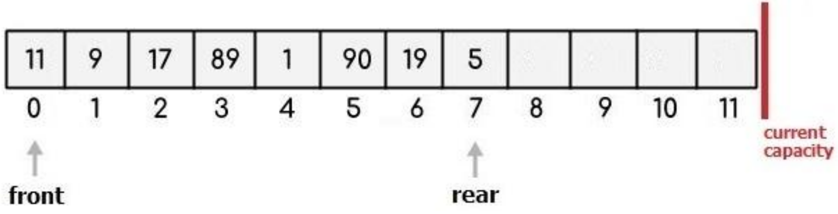


89, 1, 90, 19, 5, 3, 23, 43, 99

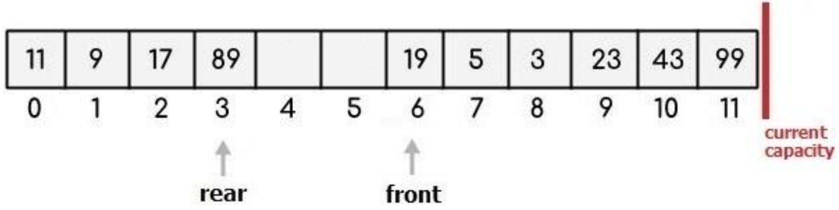
Deque (Array Based)



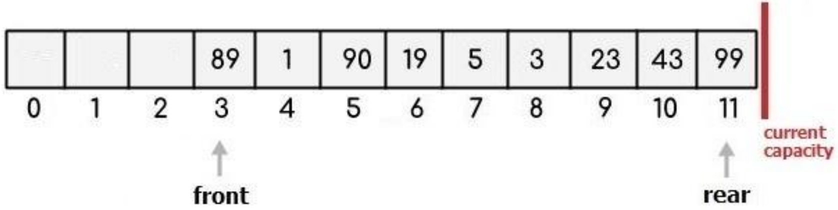
Length = (rear - front + capacity) % capacity +1
= (5 - 9 + 12) % 12 + 1 = 9



Length = (rear - front + capacity) % capacity +1
= (7 - 0 + 12) % 12 + 1 = 8



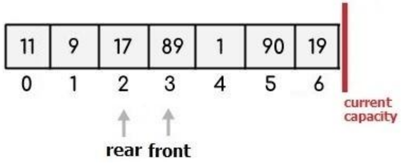
Length = (rear - front + capacity) % capacity +1
= (3 - 6 + 12) % 12 + 1 = 10

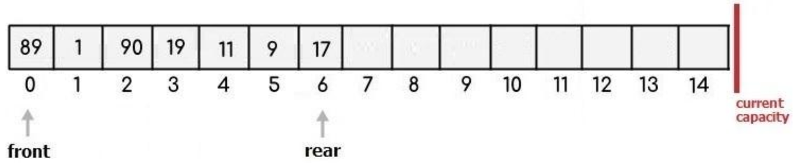


Length = (rear - front + capacity) % capacity +1
= (11 - 3 + 12) % 12 + 1 = 9

Reserve Method - Deque (Array Based)

```cpp
// This function updates the capacity of the deque
void reserve(int new_capacity) {
    // Initialize a new deque with the new capacity
    int* temp = new int[new_capacity];
    // copy the elements in the current deque to the new deque
    for (int i = 0 ; i < n ; i++)
        temp[i] = arr[(front_idx + i) % capacity];
    // delete the old deque
    delete[] arr;
    // set the temp deque with new capacity to be the deque
    arr = temp;
    // reset front_idx and rear_idx
    front_idx = 0;
    rear_idx = n - 1;
    // set the current capacity of the deque to be the new capacity
    capacity = new_capacity;
}
```





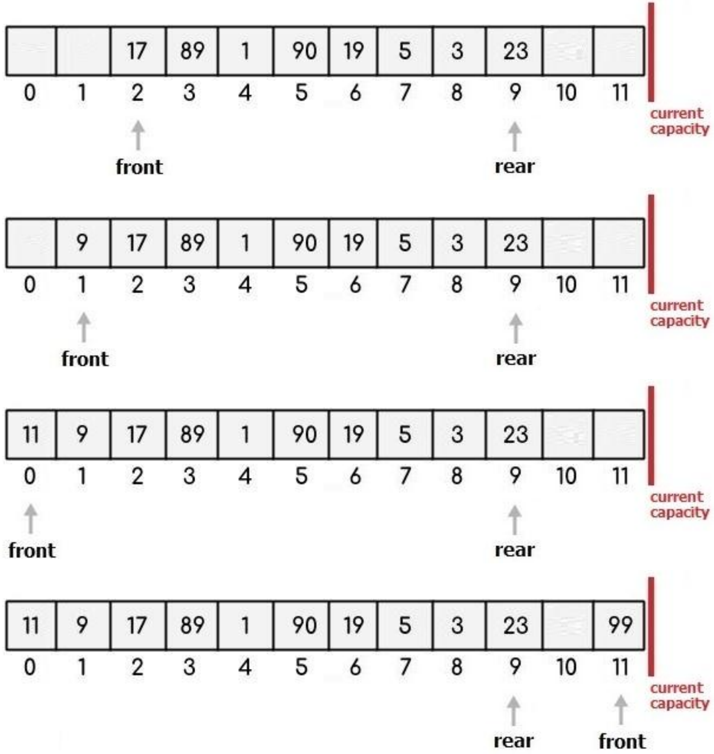
Deque-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- Section 2: Insertion Operation
- Section 3: Deletion Operation
- Section 4: Front & Back Operations
- Section 5: Traverse Operation
- Section 6: Time Complexity & Space Complexity

Insertion Operation - Deque (Array Based)

- Insert Operation is to add (store) an item to the deque.
- Insert at begin 9, 11, 99



- ***Insertion Algorithm***:

If the array is full increase the capacity

1. $\boldsymbol{if\ n == 0\ then\ front = rear = 0}$
2. $\boldsymbol{otherwise\ front = (front - 1 + capacity)\ \%\ capacity}$
3. $\boldsymbol{arr[front] = data}$
4. $\boldsymbol{n = n + 1}$

Insertion Operation - Deque (Array Based)

```
// This function inserts an element at the begin of the deque
void insert_begin(int new_data) {
    // check if we need to update the capacity of the deque
    if (n == capacity)
        reserve(2 * capacity + 1);
    // check if the deque is empty
    if (n == 0)
        front_idx = rear_idx = 0;
    // update the front_idx of the deque
    else
        front_idx = (front_idx - 1 + capacity) % capacity;
    // insert current element into Deque
    arr[front_idx] = new_data;
    // update the size of the deque
    n = n + 1;
}
```

Deque-Array-Based.cpp

Insertion Operation - Deque (Array Based)

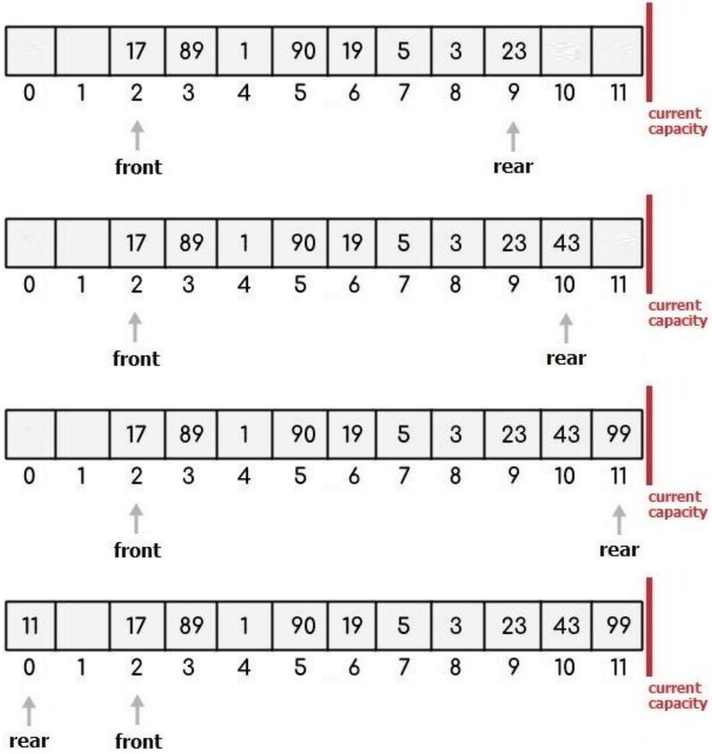
- Insert Operation is to add (store) an item to the deque.

- $\boldsymbol{Insertion\ Algorithm}$:

If the array is full increase the capacity

1. $\boldsymbol{if\ n == 0\ then\ front = rear = 0}$
2. $\boldsymbol{otherwise\ rear}$
$\boldsymbol{= (rear + 1)\ \%\ capacity}$
3. $\boldsymbol{arr[rear] = data}$
4. $\boldsymbol{n = n + 1}$

- Insert at end 43, 99, 11



Insertion Operation - Deque (Array Based)

```
// This function inserts an element at the end of the deque
void insert_end(int new_data) {
    // check if we need to update the capacity of the deque
    if (n == capacity)
        reserve(2 * capacity + 1);
    // check if the deque is empty
    if (n == 0)
        front_idx = rear_idx = 0;
    // update the rear_idx of the deque
    else
        rear_idx = (rear_idx + 1) % capacity;
    // insert the new element
    arr[rear_idx] = new_data;
    // update the size of the deque
    n = n + 1;
}
```

Deque-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- ✓ Section 2: Insertion Operation
- **Section 3: Deletion Operation**
- Section 4: Front & Back Operations
- Section 5: Traverse Operation
- Section 6: Time Complexity & Space Complexity

Deletion Operation - Deque (Array Based)

- Delete Operation remove (access) an item from the deque.

- $\boldsymbol{Deletion\ Algorithm}$:

1. $\boldsymbol{if\ front == rear\ then}$

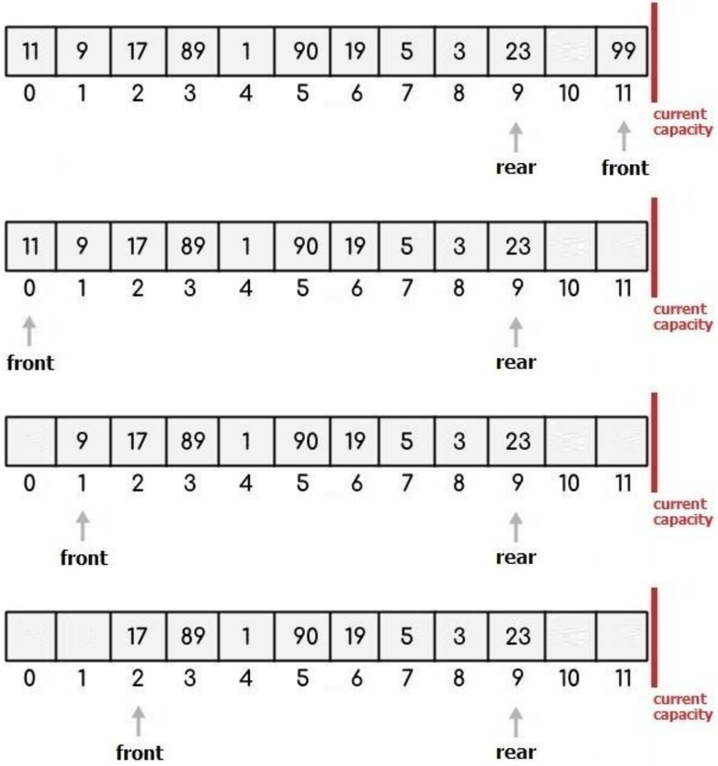
$$\boldsymbol{front = rear = -1}$$

2. $\boldsymbol{otherwise\ front}$
$\boldsymbol{= (front + 1)\ \%\ capacity}$

3. $\boldsymbol{n = n - 1}$

If the array is half full decrease the capacity

- Delete at begin 99, 11, 9



Deletion Operation - Deque (Array Based)

```
// This function deletes the first element in the deque
void delete_begin() {
    // check if the deque is empty
    if (n == 0)
        return;
    // check if the deque has only one element
    if (front_idx == rear_idx)
        front_idx = rear_idx = -1;
    // otherwise update the front_idx of the queue
    else
        front_idx = (front_idx + 1) % capacity;
    // update the size of the deque
    n = n - 1;
    // check if we need to update the capacity of the deque
    if (n < capacity / 2)
        reserve(capacity / 2);
}
```

Deque-Array-Based.cpp

Deletion Operation - Deque (Array Based)

- Delete Operation remove (access) an item from the deque.

- $\boldsymbol{Deletion\ Algorithm}$:

1. $\boldsymbol{if\ front == rear\ then}$

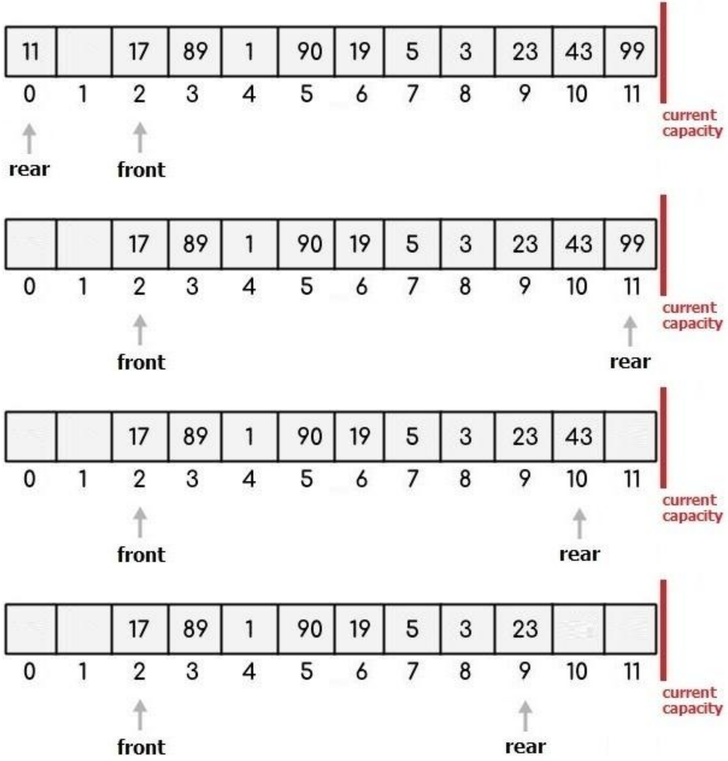
$$\boldsymbol{front = rear = -1}$$

2. $\boldsymbol{otherwise\ rear = (rear - 1 + capacity)\ \%\ capacity}$

3. $\boldsymbol{n = n - 1}$

If the array is half full decrease the capacity

- Delete at end 11, 99, 43



Deletion Operation - Deque (Array Based)

```
// This function deletes the last element in the deque
void delete_end() {
    // check if the deque is empty
    if (n == 0)
        return;
    // check if the deque has only one element
    if (front_idx == rear_idx)
        front_idx = rear_idx = -1;
    // update the rear_idx of the deque
    else
        rear_idx = (rear_idx - 1 + capacity) % capacity;
    // update the size of the deque
    n = n - 1;
    // check if we need to update the capacity of the deque
    if (n < capacity / 2)
        reserve(capacity / 2);
}
```

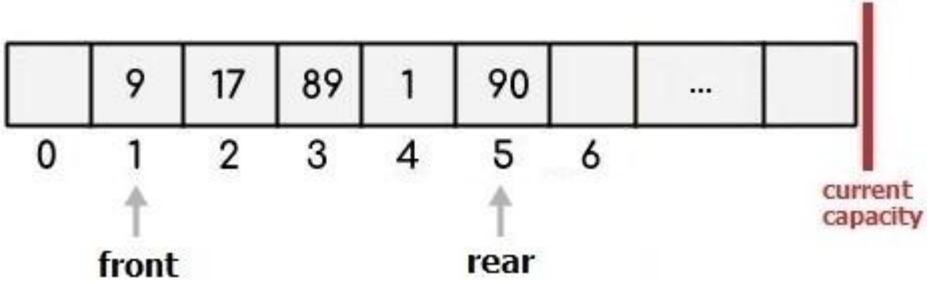
Deque-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- **Section 4: Front & Back Operations**
- Section 5: Traverse Operation
- Section 6: Time Complexity & Space Complexity

Front & Back Operations - Deque (Array Based)

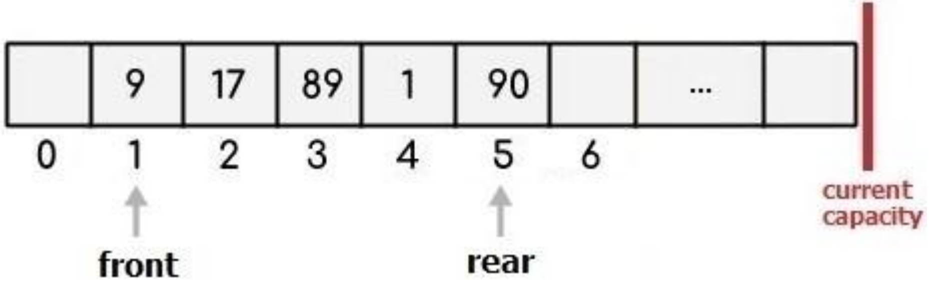
- Front Operation gets the first item in the queue.
- Front 9



- $\boldsymbol{Front\ Algorithm}$:

$\mathbf{1.}\ \boldsymbol{return\ arr[front]}$

- Back Operation gets the last item in the queue.
- Back 90



- $\boldsymbol{Back\ Algorithm}$:

$\mathbf{1.}\ \boldsymbol{return\ arr[rear]}$

Front & Back Operations - Deque (Array Based)

```
// This function returns the value of the first element in the deque
int front() {
    // check if the deque is empty
    // to return the biggest integer value as an invalid value
    if (n == 0)
        return INT_MAX;
    // otherwise return the real value
    else
        return arr[front_idx];
}
// This function returns the value of the last element in the deque
int back() {
    // check if the deque is empty
    // to return the biggest integer value as an invalid value
    if (n == 0)
        return INT_MAX;
    // otherwise return the real value
    else
        return arr[rear_idx];
}
```

Deque-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations
- Section 5: Traverse Operation
- Section 6: Time Complexity & Space Complexity

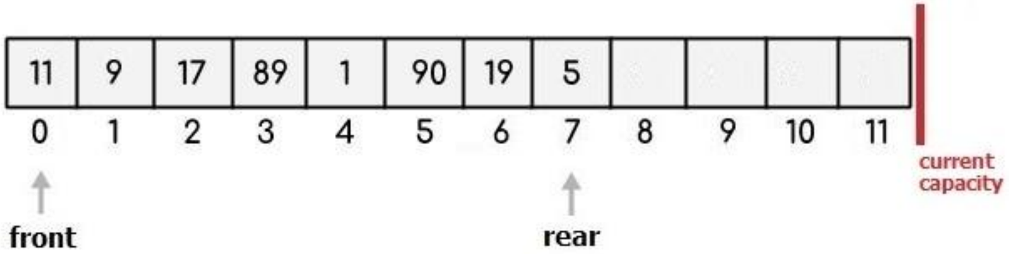
Traverse Operation - Deque (Array Based)

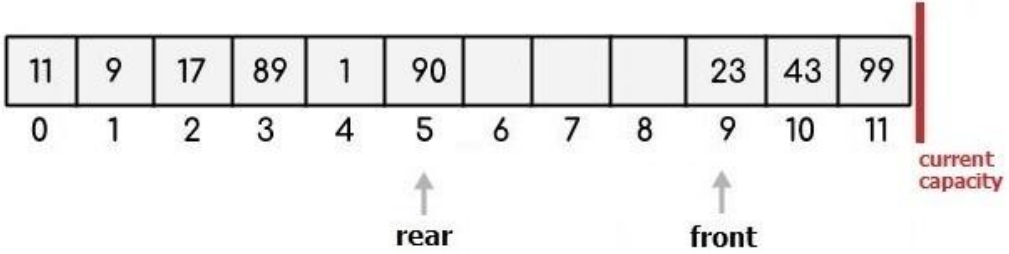
- Traverse operation is to traverse through the elements of an array.

- $\boldsymbol{Traverse\ Algorithm}$:

1. $\boldsymbol{i\ = 0}$
2. $\boldsymbol{print\ arr[(front + i)\ \%\ capacity]}$
3. $\boldsymbol{i = i + 1}$
4. $\boldsymbol{Repeat\ Step\ 3\ if\ i < n}$

- Traverse this deque





Traverse Operation - Deque (Array Based)

```cpp
// This function prints the contents of the deque
void print_deque() {
    // loop to print the elements in the deque
    for (int i = 0 ; i < n ; i++)
        cout << arr[(front_idx + i) % capacity] << ' ';
}

// This function prints the contents of the deque
void print_deque_reverse() {
    // loop to print the elements in the deque
    for (int i = n-1 ; i >= 0 ; i--)
        cout << arr[(front_idx + i) % capacity] << ' ';
}
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- Initialize a global struct

```
#include <bits/stdc++.h>
using namespace std;

// Initialize an deque with dynamic length
int front_idx = -1, rear_idx = -1;
int n;
int capacity;
int* arr;
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
cout << "adding the following elements 10 20 30 40 50\n";
insert_end(10);
insert_end(20);
insert_end(30);
insert_end(40);
insert_end(50);
cout << "the above elements have been added to the deque\n";
```

- Expected Output:

```
Deque front: 2147483647 and Deque back: 2147483647
Deque items forward:
Deque items backward:
```

```
adding the following elements 10 20 30 40 50
the above elements have been added to the deque
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
cout << "add element 60 at the end of the deque\n";
insert_end(60);

cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
Deque front: 10 and Deque back: 50
Deque items forward:  10 20 30 40 50
Deque items backward: 50 40 30 20 10
add element 60 at the end of the deque
Deque front: 10 and Deque back: 60
Deque items forward:  10 20 30 40 50 60
Deque items backward: 60 50 40 30 20 10
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "add element 20 at the begin of the deque\n";
insert_begin(20);

cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
add element 20 at the begin of the deque

Deque front: 20 and Deque back: 60
Deque items forward:  20 10 20 30 40 50 60
Deque items backward: 60 50 40 30 20 10 20
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "add element 70 at the begin of the deque\n";
insert_begin(70);
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
add element 70 at the begin of the deque

Deque front: 70 and Deque back: 60
Deque items forward:  70 20 10 20 30 40 50 60
Deque items backward: 60 50 40 30 20 10 20 70
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "delete the first element \n";
delete_begin();
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
delete the first element
Deque front: 20 and Deque back: 60
Deque items forward:  20 10 20 30 40 50 60
Deque items backward: 60 50 40 30 20 10 20
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

➢ In the Main function:

```
cout << "delete the last element \n";
delete_end();
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

➢ Expected Output:

```
delete the last element

Deque front: 20 and Deque back: 50
Deque items forward:  20 10 20 30 40 50
Deque items backward: 50 40 30 20 10 20
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "delete the first element \n";
delete_begin();
cout << "delete the last element \n";
delete_end();
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
delete the first element

delete the last element

Deque front: 10 and Deque back: 40
Deque items forward:  10 20 30 40
Deque items backward: 40 30 20 10
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "deleting the following elements 10 20 30 40\n";
delete_end();
delete_end();
delete_end();
delete_end();
cout << "the above elements have been deleted from the deque\n";
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
deleting the following elements 10 20 30 40
the above elements have been deleted from the deque

Deque front: 2147483647 and Deque back: 2147483647
Deque items forward:
Deque items backward:
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "adding the following elements 40 30 20 10\n";
insert_begin(10);
insert_begin(20);
insert_begin(30);
insert_begin(40);
cout << "the above elements have been added to the deque\n";
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
adding the following elements 40 30 20 10
the above elements have been added to the deque

Deque front: 40 and Deque back: 10
Deque items forward:  40 30 20 10
Deque items backward: 10 20 30 40
```

Deque-Array-Based.cpp

Functionality Testing - Deque (Array Based)

- In the Main function:

```
cout << "deleting the following elements 40 30 20 10\n";
delete_begin();
delete_begin();
delete_begin();
delete_begin();
cout << "the above elements have been deleted from the deque\n";
cout << "Deque front: " << front() << " and Deque back: " << back() << '\n';
cout << "Deque items forward:  ";
print_deque();
cout << '\n';
cout << "Deque items backward: ";
print_deque_reverse();
cout << '\n';
```

- Expected Output:

```
deleting the following elements 40 30 20 10
the above elements have been deleted from the deque
```

```
Deque front: 2147483647 and Deque back: 2147483647
Deque items forward:
Deque items backward:
```

Deque-Array-Based.cpp

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations
- ✓ Section 5: Traverse Operation

Section 6: Time Complexity & Space Complexity

Time Complexity & Space Complexity

- Time Analysis

	Worst Case	Average Case
• Insert at begin	$\Theta(n)$	$\theta(1)$
• Insert at end	$\Theta(n)$	$\theta(1)$
• Delete at begin	$\Theta(n)$	$\theta(1)$
• Delete at end	$\Theta(n)$	$\theta(1)$
• Front	$\Theta(1)$	$\theta(1)$
• Back	$\Theta(1)$	$\theta(1)$
• Traverse	$\Theta(n)$	$\theta(n)$

Lecture Agenda

- ✓ Section 1: Introduction to Deque
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations
- ✓ Section 5: Traverse Operation
- ✓ Section 6: Time Complexity & Space Complexity

Practice

Practice

1- Reverse string using stack
2- Check string is palindrome or not
3- Convert Infix Expression to Postfix Expression
4- Convert Infix Expression to Prefix Expression
5- Convert Postfix Expression to Infix Expression
6- Convert Prefix Expression to Infix Expression
7- Convert Postfix Expression to Prefix Expression
8- Convert Prefix Expression to Postfix Expression
9- Evaluation of Postfix Expression
10- Reverse a stack using recursion
11- Check for balanced parentheses in an expression
12- Length of the longest valid substring
13- Minimum number of bracket reversals needed to make an expression balanced
14- Next Greater Element
15- Delete middle element of a stack

Practice

16- Reverse individual words
17- Largest Rectangular Area in a Histogram
18- Find maximum depth of nested parenthesis in a string
19- Expression contains redundant bracket or not
20- Check if two expressions with brackets are same
21- Delete consecutive same words in a sequence
22- Remove brackets from an algebraic string
23- Range Queries for Longest Correct Bracket Subsequence
24- Check if stack elements are pairwise consecutive
25- Reverse a number using stack
26- Tracking current Maximum Element in a Stack
27- Decode a string recursively encoded as count followed by substring
28- Find maximum difference between nearest left and right smaller elements
29- Find if an expression has duplicate parenthesis or not
30- Find index of closing bracket for a given opening bracket in an expression

Practice

31- Sliding Window Maximum summation of all sub-arrays of size k
32- Check string is palindrome or not
33- Generate Binary Numbers from 1 to n
34- Reversing a queue using recursion
35- Reversing the first K elements of a Queue
36- Find the largest multiple of 3
37- Smallest multiple of a given number made of digits 0 and 9 only
38- Delete all elements in the deque
39- Sum of minimum and maximum elements of all subarrays of size k
40- First negative integer in every window of size k

Assignment

Implement STL Deque

- Deque (usually pronounced like "deck") is an irregular acronym of double-ended queue. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back).
- Specific libraries may implement deques in different ways, generally as some form of dynamic array. But in any case, they allow for the individual elements to be accessed directly through random access iterators, with storage handled automatically by expanding and contracting the container as needed.
- Therefore, they provide a functionality similar to vectors, but with efficient insertion and deletion of elements also at the beginning of the sequence, and not only at its end. But, unlike vectors, deques are not guaranteed to store all its elements in contiguous storage locations: accessing elements in a deque by offsetting a pointer to another element causes undefined behavior.
- Double ended queues are sequence containers with the feature of expansion and contraction on both the ends. They are similar to vectors, but are more efficient in case of insertion and deletion of elements. Unlike vectors, contiguous storage allocation may not be guaranteed.

Implement STL Deque

- Both vectors and deques provide a very similar interface and can be used for similar purposes, but internally both work in quite different ways: While vectors use a single array that needs to be occasionally reallocated for growth, the elements of a deque can be scattered in different chunks of storage, with the container keeping the necessary information internally to provide direct access to any of its elements in constant time and with a uniform sequential interface (through iterators). Therefore, deques are a little more complex internally than vectors, but this allows them to grow more efficiently under certain circumstances, especially with very long sequences, where reallocations become more expensive.
- Double Ended Queues are basically an implementation of the data structure double ended queue. A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back. Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends. The functions for deque are same as vector, with an addition of push and pop operations for both front and back.

More Info: cplusplus.com/reference/deque/deque/
More Info: en.cppreference.com/w/cpp/container/deque
More Info: geeksforgeeks.org/deque-cpp-stl/

Implement STL Deque

- Member functions:
  - (constructor) Construct vector (public member function)
  - (destructor) Vector destructor (public member function)
  - (operator=) Assign content (public member function)
- Iterators:
  - (begin) Return iterator to beginning (public member function)
  - (end) Return iterator to end (public member function)
  - (rbegin) Return reverse iterator to reverse beginning (public member function)
  - (rend) Return reverse iterator to reverse end (public member function)
  - (cbegin) Return const_iterator to beginning (public member function)
  - (cend) Return const_iterator to end (public member function)
  - (crbegin) Return const_reverse_iterator to reverse beginning (public member function)
  - (crend) Return const_reverse_iterator to reverse end (public member function)

Implement STL Deque

- Capacity:
  - (size) Return size (public member function)
  - (max_size) Return maximum size (public member function)
  - (resize) Change size (public member function)
  - (empty) Test whether vector is empty (public member function)
  - (shrink_to_fit) Shrink to fit (public member function)
- Element access:
  - (operator[]) Access element (public member function)
  - (at) Access element (public member function)
  - (front) Access first element (public member function)
  - (back) Access last element (public member function)

Implement STL Deque

- Modifiers:

(assign) Assign vector content (public member function)
(push_back) Add element at the end (public member function)
(push_front) Insert element at beginning (public member function)
(pop_back) Delete last element (public member function)
(pop_front) Delete first element (public member function)
(insert) Insert elements (public member function)
(erase) Erase elements (public member function)
(swap) Swap content (public member function)
(clear) Clear content (public member function)

More Info: cplusplus.com/reference/deque/deque/
More Info: en.cppreference.com/w/cpp/container/deque


DO
MORE.