Circular implementation of Deque

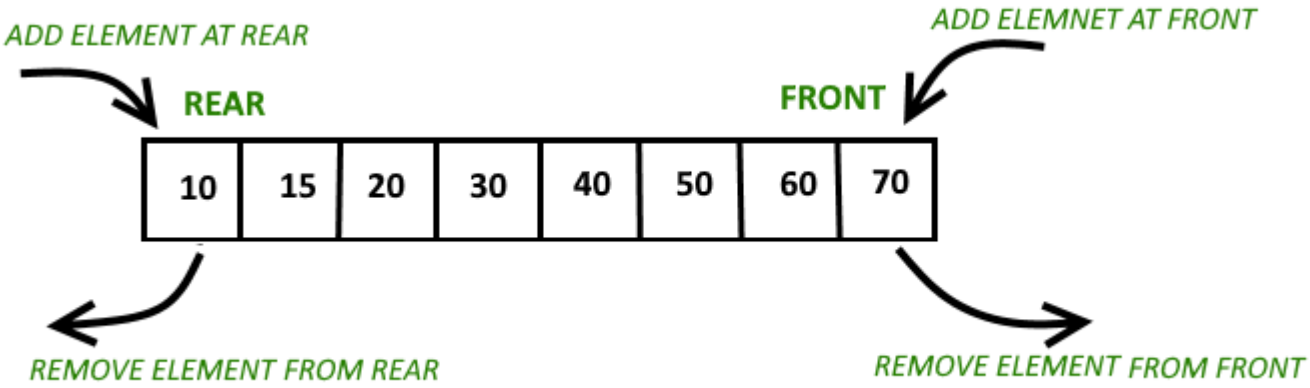
Implement a deque Using a circular array:

> Deque or Double Ended Queue is a generalized version of the Queue data structure that allows insert and delete at both ends

Operations on Deque:

> Mainly the following four basic operations are performed on queue:
>
> - **insertFront()**: Adds an item at the front of Deque.
> - **insertRear()**: Adds an item at the rear of Deque.
> - **deleteFront()**: Deletes an item from front of Deque.
> - **deleteRear()**: Deletes an item from rear of Deque.
>
> In addition to above operations, following operations are also supported
>
> - **getFront()**: Gets the front item from queue.
> - **getRear()**: Gets the last item from queue.
> - **isEmpty()**: Checks whether Deque is empty or not.

- **isFull()**: Checks whether Deque is full or not.

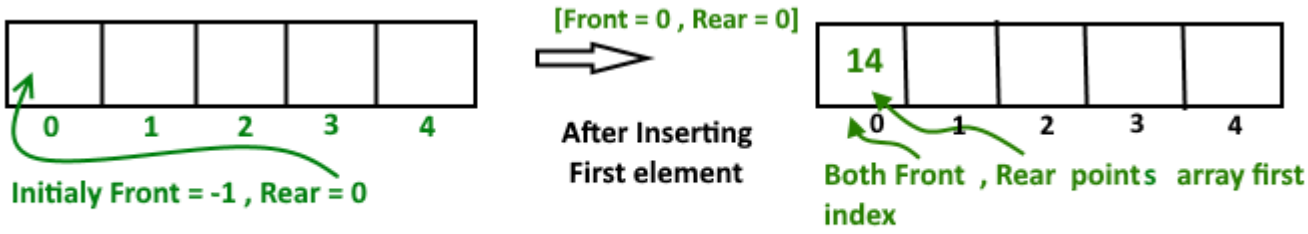


Circular array implementation of Deque:

For implementing deque, we need to keep track of two indices, front, and rear. We enqueue(push) an item at the rear or the front end of the dequeue and dequeue(pop) an item from both the rear and the front end.

Working:

Create an empty array 'arr' of size **N**
initialize **front = -1** , **rear = 0**
Inserting the First element in the deque, at either front or rear will lead to the same result:



Dash
All
Articles
Videos
Problems
Quiz
Contest

Note: After inserting **Front** Points at 0 and **Rear** points at 0

Insert Elements at the Rear end of Deque:

a). First we check deque if Full or Not

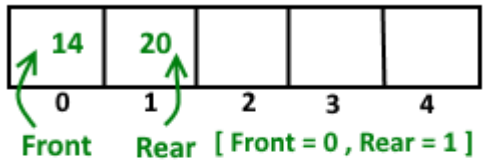
b). IF Rear == Size-1
then reinitialize Rear = 0 ;
Else increment Rear by '1'
and push current key into Arr[rear] = key
Front remain same.

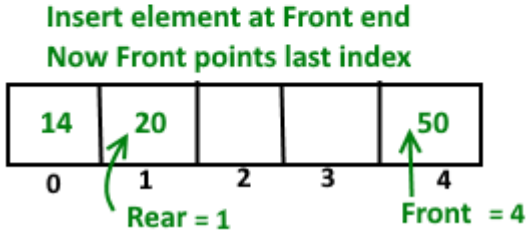
Insert Elements at the Front end of Deque:

a). First we check deque if Full or Not

b). IF Front == 0 || initial position, move Front
to points last index of array
front = size - 1
Else decremented front by '1' and push

current key into Arr[Front] = key
Rear remain same





Delete Element From Rear end of Deque:

First Check deque is Empty or Not

If deque has only one element
front = -1 ; rear =-1;

Else IF Rear points to the first index of array
it's means we have to move rear to points
last index [now first inserted element at
front end become rear end]

rear = size-1;
Else || decrease rear by '1' rear = rear-1;

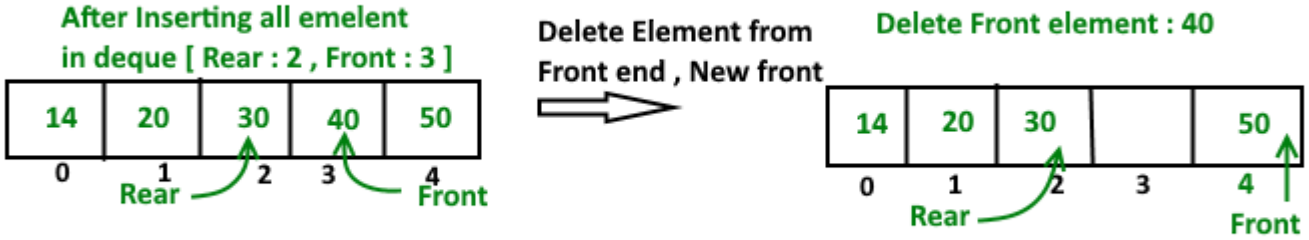
Delete Element From the Front end of Deque:

a). first Check deque is Empty or Not

b). If deque has only one element
front = -1 ; rear =-1 ;

Else IF front points to the last index of the array
it's means we have no more elements in array so
we move front to points first index of array
front = 0 ;

Else || increment Front by '1'
front = front+1;



Below is the implementation of the above methods:

```
// C++ implementation of De-queue using circular
// array
#include <iostream>
using namespace std;

// Maximum size of array or Dequeue
#define MAX 100

// A structure to represent a Deque
class Deque {
    int arr[MAX];
    int front;
    int rear;
    int size;

public:
    Deque(int size)
    {
        front = -1;
        rear = 0;
        this->size = size;
    }

    // Operations on Deque:
    void insertfront(int key);
    void insertrear(int key);
    void deletefront();
```

```
    void deleterear();
    bool isFull();
    bool isEmpty();
    int getFront();
    int getRear();
};

// Checks whether Deque is full or not.
bool Deque::isFull()
{
    return ((front == 0 && rear == size - 1)
            || front == rear + 1);
}

// Checks whether Deque is empty or not.
bool Deque::isEmpty() { return (front == -1); }

// Inserts an element at front
void Deque::insertfront(int key)
{
    // check whether Deque if  full or not
    if (isFull()) {
        cout << "Overflow\n" << endl;
        return;
    }

    // If queue is initially empty
    if (front == -1) {
        front = 0;
```

```
        rear = 0;
    }

    // front is at first position of queue
    else if (front == 0)
        front = size - 1;

    else // decrement front end by '1'
        front = front - 1;

    // insert current element into Deque
    arr[front] = key;
}

// function to inset element at rear end
// of Deque.
void Deque ::insertrear(int key)
{
    if (isFull()) {
        cout << " Overflow\n " << endl;
        return;
    }

    // If queue is initially empty
    if (front == -1) {
        front = 0;
        rear = 0;
    }
```

```
    // rear is at last position of queue
    else if (rear == size - 1)
        rear = 0;

    // increment rear end by '1'
    else
        rear = rear + 1;

    // insert current element into Deque
    arr[rear] = key;
}

// Deletes element at front end of Deque
void Deque ::deletefront()
{
    // check whether Deque is empty or not
    if (isEmpty()) {
        cout << "Queue Underflow\n" << endl;
        return;
    }

    // Deque has only one element
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else
        // back to initial position
        if (front == size - 1)
```

```
        front = 0;

    else // increment front by '1' to remove current
        // front value from Deque
        front = front + 1;
}

// Delete element at rear end of Deque
void Deque::deleterear()
{
    if (isEmpty()) {
        cout << " Underflow\n" << endl;
        return;
    }

    // Deque has only one element
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else if (rear == 0)
        rear = size - 1;
    else
        rear = rear - 1;
}

// Returns front element of Deque
int Deque::getFront()
{
```

```
    // check whether Deque is empty or not
    if (isEmpty()) {
        cout << " Underflow\n" << endl;
        return -1;
    }
    return arr[front];
}

// function return rear element of Deque
int Deque::getRear()
{
    // check whether Deque is empty or not
    if (isEmpty() || rear < 0) {
        cout << " Underflow\n" << endl;
        return -1;
    }
    return arr[rear];
}

// Driver code
int main()
{
    Deque dq(5);

      // Function calls
    cout << "Insert element at rear end  : 5 \n";
    dq.insertrear(5);

    cout << "insert element at rear end : 10 \n";
```

```
    dq.insertrear(10);

    cout << "get rear element "
         << " " << dq.getRear() << endl;

    dq.deleterear();
    cout << "After delete rear element new rear"
         << " become " << dq.getRear() << endl;

    cout << "inserting element at front end \n";
    dq.insertfront(15);

    cout << "get front element "
         << " " << dq.getFront() << endl;

    dq.deletefront();

    cout << "After delete front element new "
         << "front become " << dq.getFront() << endl;
    return 0;
}
```

Output

```
Insert element at rear end  : 5
insert element at rear end : 10
get rear element  10
After delete rear element new rear become 5
inserting element at front end
```

```
get front element  15
After delete front element new front become 5
```

Time Complexity: O(N)
Auxiliary Space: O(N)

Dash
All
Articles
Videos
Problems
Quiz
Contest

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.