Deque (or double-ended queue)

The deque stands for Double Ended Queue. Deque is a linear data structure where the insertion and deletion operations are performed from both ends. We can say that deque is a generalized version of the queue.

Though the insertion and deletion in a deque can be performed on both ends, it does not follow the FIFO rule. The representation of a deque is given as follows -

Representation of deque

Types of deque

There are two types of deque -

- Input restricted queue
- Output restricted queue

Input restricted Queue

In input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.

input restricted double ended queue

Output restricted Queue

In output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.

Output restricted double ended queue

Operations performed on deque

There are the following operations that can be applied on a deque -

- Insertion at front
- Insertion at rear
- Deletion at front
- Deletion at rear

We can also perform peek operations in the deque along with the operations listed above. Through peek operation, we can get the deque's front and rear elements of the deque. So, in addition to the above operations, following operations are also supported in deque -

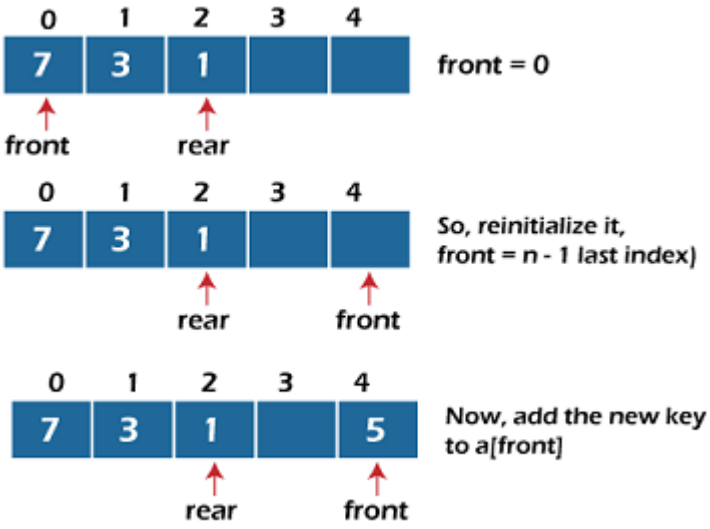
- Get the front item from the deque
- Get the rear item from the deque
- Check whether the deque is full or not
- Checks whether the deque is empty or not

Now, let's understand the operation performed on deque using an example.

Insertion at the front end

In this operation, the element is inserted from the front end of the queue. Before implementing the operation, we first have to check whether the queue is full or not. If the queue is not full, then the element can be inserted from the front end by using the below conditions -

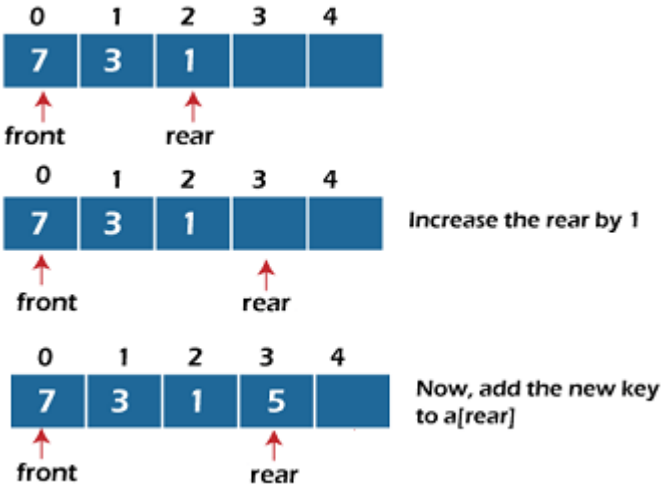
- If the queue is empty, both rear and front are initialized with 0. Now, both will point to the first element.
- Otherwise, check the position of the front if the front is less than 1 (front < 1), then reinitialize it by **front = n - 1**, i.e., the last index of the array.



Insertion at the rear end

In this operation, the element is inserted from the rear end of the queue. Before implementing the operation, we first have to check again whether the queue is full or not. If the queue is not full, then the element can be inserted from the rear end by using the below conditions -

- If the queue is empty, both rear and front are initialized with 0. Now, both will point to the first element.
- Otherwise, increment the rear by 1. If the rear is at last index (or size - 1), then instead of increasing it by 1, we have to make it equal to 0.



Deletion at the front end

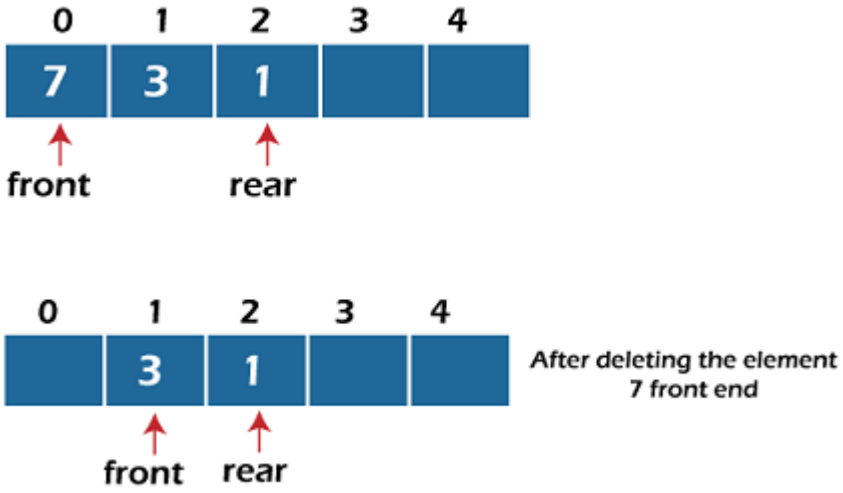
In this operation, the element is deleted from the front end of the queue. Before implementing the operation, we first have to check whether the queue is empty or not.

If the queue is empty, i.e., front = -1, it is the underflow condition, and we cannot perform the deletion. If the queue is not full, then the element can be inserted from the front end by using the below conditions -

If the deque has only one element, set rear = -1 and front = -1.

Else if front is at end (that means front = size - 1), set front = 0.

Else increment the front by 1, (i.e., front = front + 1).



Deletion at the rear end

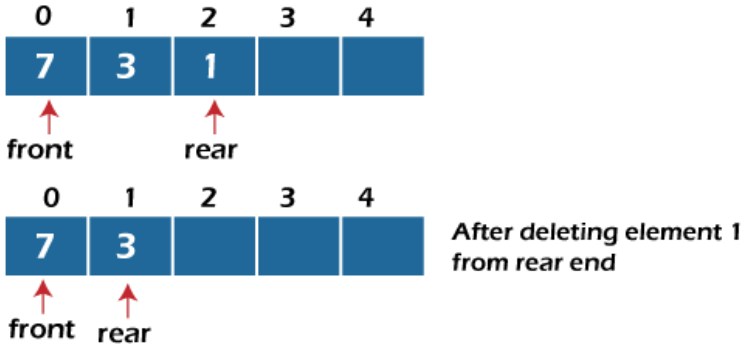
In this operation, the element is deleted from the rear end of the queue. Before implementing the operation, we first have to check whether the queue is empty or not.

If the queue is empty, i.e., front = -1, it is the underflow condition, and we cannot perform the deletion.

If the deque has only one element, set rear = -1 and front = -1.

If rear = 0 (rear is at front), then set rear = n - 1.

Else, decrement the rear by 1 (or, rear = rear -1).



Check empty

This operation is performed to check whether the deque is empty or not. If front = -1, it means that the deque is empty.

Check full

This operation is performed to check whether the deque is full or not. If front = rear + 1, or front = 0 and rear = n - 1 it means that the deque is full.

The time complexity of all of the above operations of the deque is O(1), i.e., constant.

Applications of deque

- Deque can be used as both stack and queue, as it supports both operations.
- Deque can be used as a palindrome checker means that if we read the string from both ends, the string would be the same.

Implementation of deque

Now, let's see the implementation of deque in C programming language.

```
1. #include <stdio.h>
2. #define size 5
3. int deque[size];
4. int f = -1, r = -1;
5. // insert_front function will insert the value from the front
6. void insert_front(int x)
7. {
8.    if((f==0 && r==size-1) || (f==r+1))
9.    {
10.      printf("Overflow");
11.   }
12.   else if((f==-1) && (r==-1))
13.   {
14.      f=r=0;
15.      deque[f]=x;
16.   }
17.   else if(f==0)
18.   {
19.      f=size-1;
```

```
20.       deque[f]=x;
21.    }
22.    else
23.    {
24.       f=f-1;
25.       deque[f]=x;
26.    }
27. }
28.
29. // insert_rear function will insert the value from the rear
30. void insert_rear(int x)
31. {
32.    if((f==0 && r==size-1) || (f==r+1))
33.    {
34.       printf("Overflow");
35.    }
36.    else if((f==-1) && (r==-1))
37.    {
38.       r=0;
39.       deque[r]=x;
40.    }
41.    else if(r==size-1)
42.    {
43.       r=0;
44.       deque[r]=x;
45.    }
46.    else
47.    {
48.       r++;
49.       deque[r]=x;
50.    }
51.
52. }
53.
54. // display function prints all the value of deque.
55. void display()
56. {
```

```
57.    int i=f;
58.    printf("\nElements in a deque are: ");
59.
60.    while(i!=r)
61.    {
62.        printf("%d ",deque[i]);
63.        i=(i+1)%size;
64.    }
65.     printf("%d",deque[r]);
66. }
67.
68. // getfront function retrieves the first value of the deque.
69. void getfront()
70. {
71.    if((f==-1) && (r==-1))
72.    {
73.        printf("Deque is empty");
74.    }
75.    else
76.    {
77.        printf("\nThe value of the element at front is: %d", deque[f]);
78.    }
79.
80. }
81.
82. // getrear function retrieves the last value of the deque.
83. void getrear()
84. {
85.    if((f==-1) && (r==-1))
86.    {
87.        printf("Deque is empty");
88.    }
89.    else
90.    {
91.        printf("\nThe value of the element at rear is %d", deque[r]);
92.    }
93.
```

```
94. }
95.
96. // delete_front() function deletes the element from the front
97. void delete_front()
98. {
99.    if((f==-1) && (r==-1))
100.            {
101.               printf("Deque is empty");
102.            }
103.            else if(f==r)
104.            {
105.               printf("\nThe deleted element is %d", deque[f]);
106.              f=-1;
107.               r=-1;
108.
109.            }
110.            else if(f==(size-1))
111.            {
112.               printf("\nThe deleted element is %d", deque[f]);
113.                f=0;
114.            }
115.            else
116.            {
117.                printf("\nThe deleted element is %d", deque[f]);
118.               f=f+1;
119.            }
120.         }
121.
122.         // delete_rear() function deletes the element from the rear
123.         void delete_rear()
124.         {
125.            if((f==-1) && (r==-1))
126.            {
127.               printf("Deque is empty");
128.            }
129.            else if(f==r)
130.            {
```

```
131.              printf("\nThe deleted element is %d", deque[r]);
132.             f=-1;
133.              r=-1;
134.
135.          }
136.          else if(r==0)
137.          {
138.             printf("\nThe deleted element is %d", deque[r]);
139.              r=size-1;
140.          }
141.          else
142.          {
143.              printf("\nThe deleted element is %d", deque[r]);
144.             r=r-1;
145.          }
146.      }
147.
148.      int main()
149.      {
150.         insert_front(20);
151.         insert_front(10);
152.         insert_rear(30);
153.         insert_rear(50);
154.         insert_rear(80);
155.         display();  // Calling the display function to retrieve the values of deque
156.         getfront();  // Retrieve the value at front-end
157.         getrear();  // Retrieve the value at rear-end
158.         delete_front();
159.         delete_rear();
160.         display(); // calling display function to retrieve values after deletion
161.         return 0;
162.      }
```

Output:

```
Elements in a deque are: 10 20 30 50 80
The value of the element at front is: 10
The value of the element at rear is 80
The deleted element is 10
The deleted element is 80
Elements in a deque are: 20 30 50
```