FACULTY NAME- PRAVIN KUMAR PANDEY
DEPARTMENT NAME- COMPUTER SCIENCE AND ENGINEERING
SUBJECT- DATA STRUCTURE USING- C
SEMESTER- 3
YEAR- 2

UNIT-2

Queue:

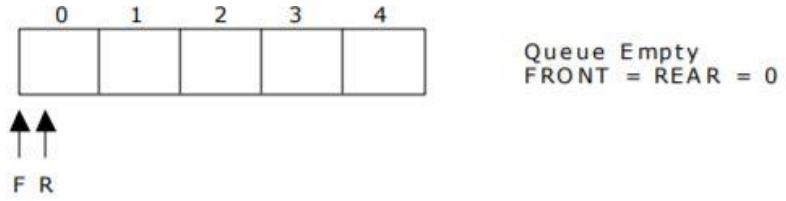
A queue is a data structure that is best described as "**first in, first out (FIFO)**". A queue is another special kind of list, where items are inserted at one end called the **rear** and deleted at the other end called the **front**. A real-world example of a queue is people waiting in line at the bank. As each person enters the bank, he or she is "enqueued" at the back of the line. When a teller becomes available, they are "dequeued" at the front of the line.

The operations for a queue are analogues to those for a stack, the difference is that the insertions go at the end of the list, rather than the beginning. We shall use the following operations on queues:

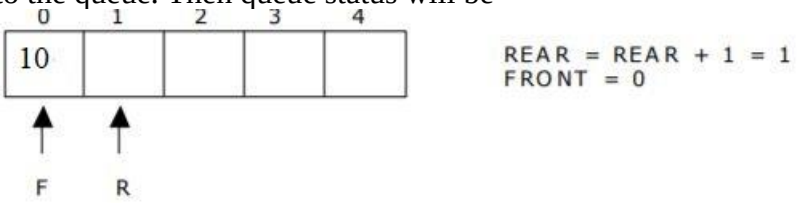
- **Enqueue:** which inserts an element at the end of the queue.
- **Dequeue:** which deletes an element at the start of the queue.

Representation of Queue:

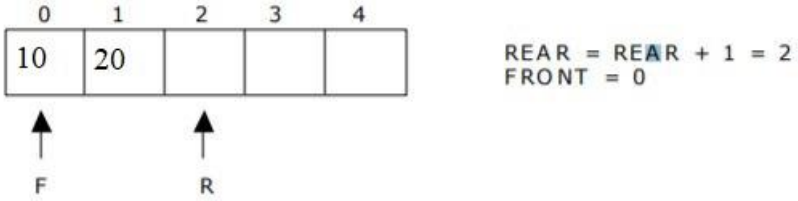
Let us consider a queue, which can hold maximum of five elements. Initially the queue is empty.



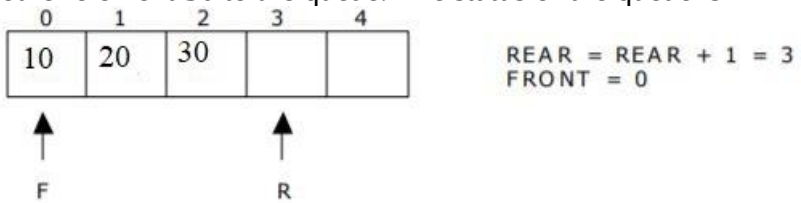
Now, insert 10 to the queue. Then queue status will be-



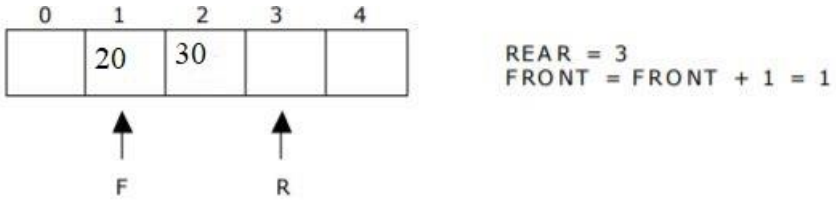
Next, insert 20 to the queue. Then the queue status is-



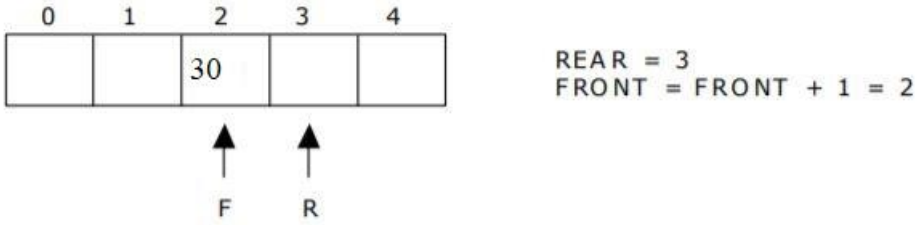
Again, insert another element 30 to the queue. The status of the queue is-



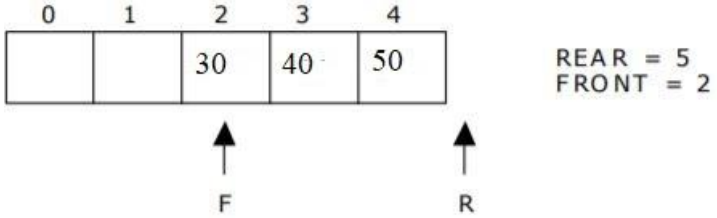
Now, delete an element. The element deleted is the element at the front of the queue. So, the status of the queue is-



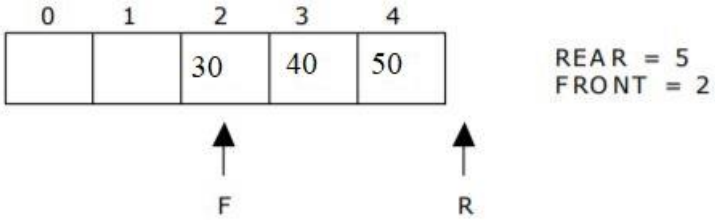
Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 20 is deleted. The queue status is as follows-



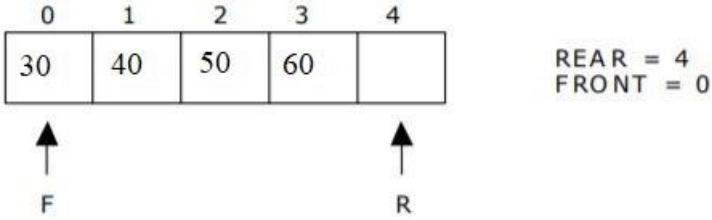
Now, insert new elements 40 and 50 into the queue. The queue status is:



Next insert another element, say 60 to the queue. We cannot insert 60 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows-



Now it is not possible to insert an element 60 even though there are two vacant positions in the linear queue. To overcome this problem the elements of the queue are to be shifted towards the beginning of the queue so that it creates vacant position at the rear end. Then the FRONT and REAR are to be adjusted properly. The element 60 can be inserted at the rear end. After this operation, the queue status is as follows-



This difficulty can overcome if we treat queue position with index 0 as a position that comes after position with index 4 i.e., we treat the queue as a circular queue.

Procedure and code for Queue operations using array

In order to create a queue we require a one dimensional array Q (1: n) and two variables front and rear. The conventions we shall adopt for these two variables are that front is always 1 less than the actual front of the queue and rear always points to the last element in the queue. Thus, front = rear if and only if there are no elements in the queue. The initial condition then is front = rear = 0.

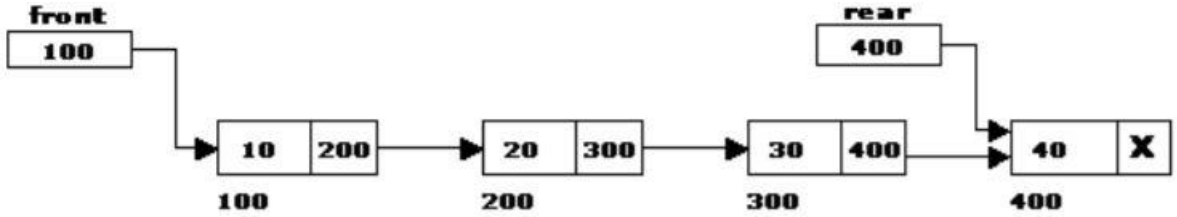
The various queue operations to perform creation, deletion and display the elements in a queue are as follows-

- InsertQ(): inserts an element at the end of queue Q.
- DeleteQ(): deletes the first element of Q.
- DisplayQ(): displays the elements in the queue.

```
# include
# define MAX 6
int Q[MAX];
int front, rear;
void insertQ()
{
	int data;
	if(rear == MAX)
	{
		printf("\n Linear Queue is full");
		return;
	}
	else
	{
		printf("\n Enter data: ");
		scanf("%d", &data);
		Q[rear] = data;
		rear++; printf("\n Data Inserted in the Queue ");
	}
}

void deleteQ()
{
	if(rear == front)
	{
		printf("\n\n Queue is Empty..");
		return;
	}
	else
	{
		printf("\n Deleted element from Queue is %d", Q[front]);
		 front++;
	}
}

void displayQ()
{
	int i;
	if(front == rear)
	{
		printf("\n\n\t Queue is Empty"); r
		eturn;
	}
	else
```

```
	{
		printf("\n Elements in Queue are: ");
		for(i = front; i < rear; i++)
		{
			printf("%d\t", Q[i]);
		}
	}
}
int menu()
{
	int ch;
	clrscr();
	printf("\n \tQueue operations using ARRAY..");
	printf("\n ----------**********----------\n");
	printf("\n 1. Insert ");
	printf("\n 2. Delete ");
	printf("\n 3. Display");
	printf("\n 4. Quit ");
	printf("\n Enter your choice: ");
	scanf("%d", &ch);
	return ch;
}
void main()
{
	int ch;
	do
	{
		ch = menu();
		switch(ch)
		{
			case 1:
				insertQ();
				break;
			case 2:
				deleteQ();
				break;
			case 3:
				displayQ();
				break;
			case 4:
				return;
		}
		getch();
	}while(1);
}
```

Linked List Implementation of Queue: We can represent a queue as a linked list. In a queue data is deleted from the front end and inserted at the rear end. We can perform similar operations on the two ends of a list. We use two pointers front and rear for our linked queue implementation.



Source code for queue operations using linked list:

```
# include
# include
struct queue
{
        int data;
        struct queue *next;
};
typedef struct queue node;
node *front = NULL;
node *rear = NULL;

node* getnode()
{
        node *temp;
        temp = (node *) malloc(sizeof(node)) ;
        printf("\n Enter data ");
        scanf("%d", &temp -> data);
        temp -> next = NULL;
        return temp;
}
void insertQ()
{
        node *newnode;
        newnode = getnode();
        if(newnode == NULL)
        {
                printf("\n Queue Full");
                return;
        }
        if(front == NULL)
        {
                front = newnode;
                rear = newnode;
        }
        else
        {
                rear -> next = newnode;
                rear = newnode;
        }
        printf("\n\n\t Data Inserted into the Queue..");
}
void deleteQ()
{
        node *temp;
        if(front == NULL)
        {
                printf("\n\n\t Empty Queue..");
```

```
		return;
	}
	temp = front;
	front = front -> next;
	printf("\n\n\t Deleted element from queue is %d ", temp -> data);
	free(temp);
}
void displayQ()
{
	node *temp;
	if(front == NULL)
	{
		printf("\n\n\t\t Empty Queue ");
	}
	else
	{
		temp = front;
		printf("\n\n\n\t\t Elements in the Queue are: ");
		while(temp != NULL )
		{
			printf("%5d ", temp -> data);
			temp = temp -> next;
		}
	}
}
char menu()
{
	char ch;
	clrscr();
	printf("\n \t..Queue operations using pointers.. ");
	printf("\n\t ----------**********----------\n");
	printf("\n 1. Insert ");
	printf("\n 2. Delete ");
	printf("\n 3. Display");
	printf("\n 4. Quit ");
	printf("\n Enter your choice: ");
	ch = getche();
	return ch;
}
void main()
{
	char ch;
	do
	{
		ch = menu();
		switch(ch)
		{
			case '1' :
				insertQ();
				break;
			case '2' :
				deleteQ();
				break;
			case '3' :
				displayQ();
```

```
                    break;
                case '4':
                    return;
            }getch();
        } while(ch != '4');

}
```

Applications of Queues:

- It is used to schedule the jobs to be processed by the CPU.
- When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO) basis.
- Breadth first search uses a queue data structure to find an element from a graph.

Disadvantages of Linear Queue:

- **Time consuming:** linear time to be spent in shifting the elements to the beginning of the queue.
- **Signaling queue full:** even if the queue is having vacant position.

Circular Queue:

Circular queue is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle.

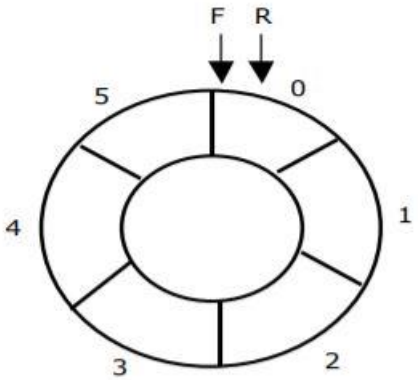
- Circular linked list fallow the First In First Out principle.
- Elements are added at the rear end and the elements are deleted at front end of the queue.
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as “Ring buffer”.

Circular Queue can be created in three ways they are-

- Using single linked list.
- Using double linked list.
- Using arrays.

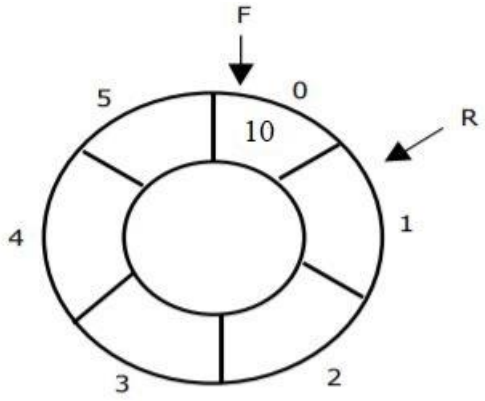
Representation of Circular Queue:

Let us consider a circular queue, which can hold maximum (MAX) of six elements. Initially the queue is empty-



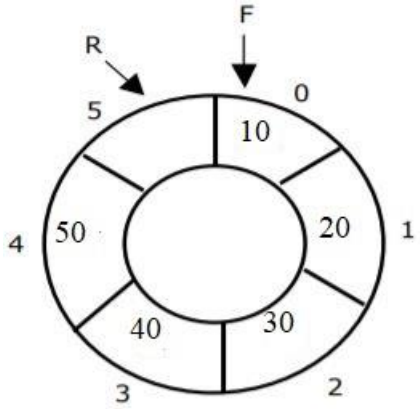
Queue Empty
MAX = 6
FRONT = REAR = 0
COUNT = 0

Now, insert 10 to the circular queue. Then circular queue status will be-



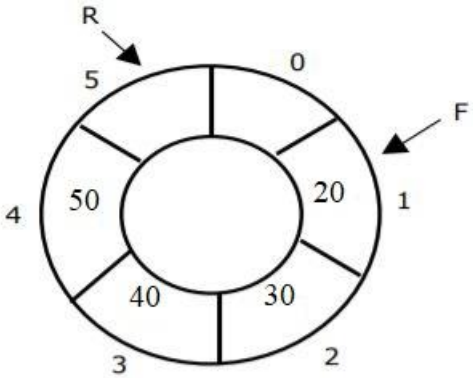

```
FRONT = 0
REAR = (REAR + 1) % 6 = 1
COUNT = 1
```

Similar Insert new elements 20, 30, 40 and 50 into the circular queue. The circular queue status is-



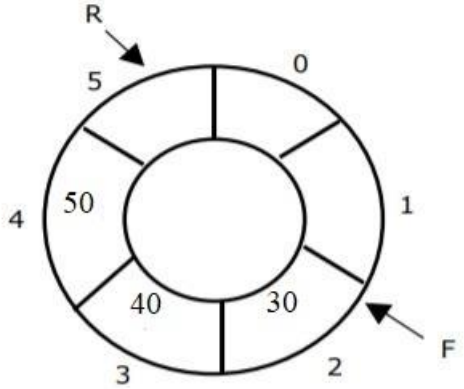

```
FRONT = 0, REAR = 5
REAR = REAR % 6 = 5
COUNT = 5
```

Now, delete an element. The element deleted is the element at the front of the circular queue. So, 10 is deleted. The circular queue status is as follows-



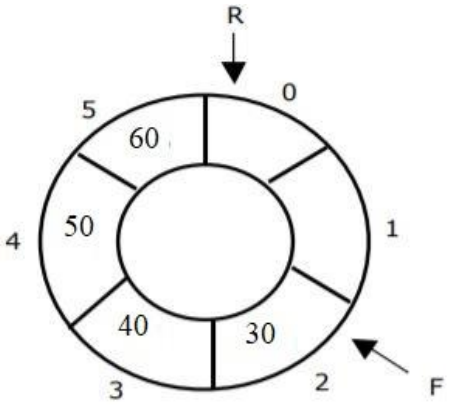

```
FRONT = (FRONT + 1) % 6 = 1
REAR = 5
COUNT = COUNT - 1 = 4
```

Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 20 is deleted. The circular queue status is as follows-



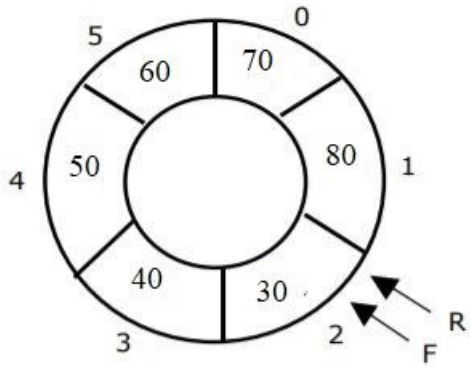

```
FRONT = (FRONT + 1) % 6 = 2
REAR = 5
COUNT = COUNT - 1 = 3
```

Again, insert another element 60 to the circular queue. The status of the circular queue is-



FRONT = 2
REAR = (REAR + 1) % 6 = 0
COUNT = COUNT + 1 = 4

Now, insert new elements 70 and 80 into the circular queue. The circular queue status is-



FRONT = 2, REAR = 2
REAR = REAR % 6 = 2
COUNT = 6

Now, if we insert an element to the circular queue, as COUNT = MAX we cannot add the element to circular queue. So, the circular queue is full.

Code for Circular Queue Implementation

```
# include
# include
# define MAX 6
int CQ[MAX];
int front = 0;
int rear = 0;
int count = 0;

void insertCQ()
{
	int data;
	if(count == MAX)
	{
		printf("\n Circular Queue is Full");
	}
	else
	{
		printf("\n Enter data: ");
		scanf("%d", &data);
		CQ[rear] = data;
		rear = (rear + 1) % MAX;
		count ++;
		printf("\n Data Inserted in the Circular Queue ");
	}
}
```

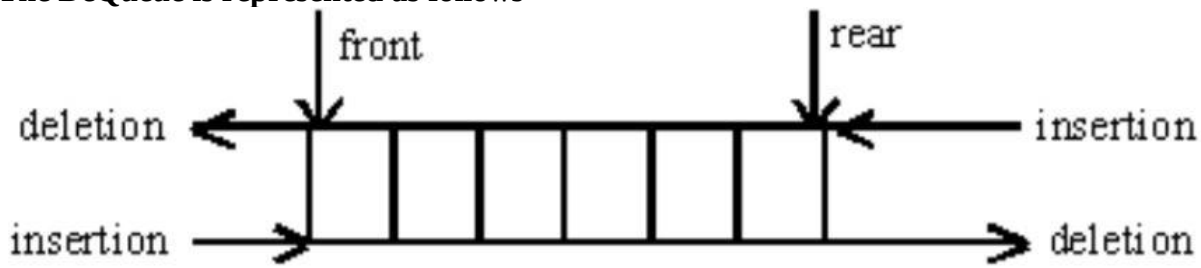
```
void deleteCQ()
{
	if(count == 0)
	{
		printf("\n\nCircular Queue is Empty..");
	}
	else
	{
		printf("\n Deleted element from Circular Queue is %d ", CQ[front]);
		front = (front + 1) % MAX;
		count --;
	}
}

void displayCQ()
{
	int i, j;
	if(count == 0)
	{
		printf("\n\n\t Circular Queue is Empty ");
	}
	else
	{
		printf("\n Elements in Circular Queue are: ");
		j = count;
		for(i = front; j != 0; j--)
		{
			printf("%d\t", CQ[i]);
			i = (i + 1) % MAX;
		}
	}
}

int menu()
{
	int ch;
	clrscr();
	printf("\n \t Circular Queue Operations using ARRAY..");
	printf("\n -----------**********-------------\n");
	printf("\n 1. Insert ");
	printf("\n 2. Delete ");
	printf("\n 3. Display");
	printf("\n 4. Quit ");
	printf("\n Enter Your Choice: ");
	scanf("%d", &ch);
	return ch;
}

void main()
{
	int ch;
	do
	{
		ch = menu();
```

```
                switch(ch)
                {
                        case 1:
                                insertCQ();
                                break;
                        case 2:
                                deleteCQ();
                                break;
                        case 3:
                                displayCQ();
                                break;
                        case 4:
                                return;
                        default:
                                printf("\n Invalid Choice ");
                }
                getch();
        } while(1);
}
```

DEQUE(Double Ended Queue):

A double-ended queue (dequeue, often abbreviated to deque, pronounced deck) 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. Like an ordinary queue, a double-ended queue is a data structure it supports the following operations: enq_front, enq_back, deq_front, deq_back, and empty. **Dequeue can be behave like a queue by using only enq_front and deq_front , and behaves like a stack by using only enq_front and deq_rear.**

The DeQueue is represented as follows-



DEQUE can be represented in two ways they are

1) Input restricted DEQUE(IRD)
2) output restricted DEQUE(ORD)

The output restricted DEQUE allows deletions from only one end and input restricted DEQUE allow insertions at only one end. The DEQUE can be constructed in two ways they are-

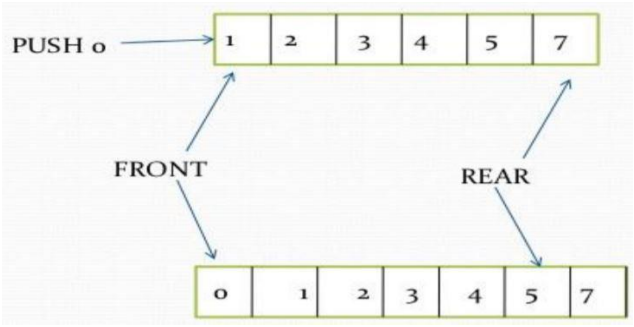
1) Using array
2) Using linked list

Operations in DEQUE

1) Insert element at front
2) Insert element at back
3) Remove element at front
4) Remove element at back

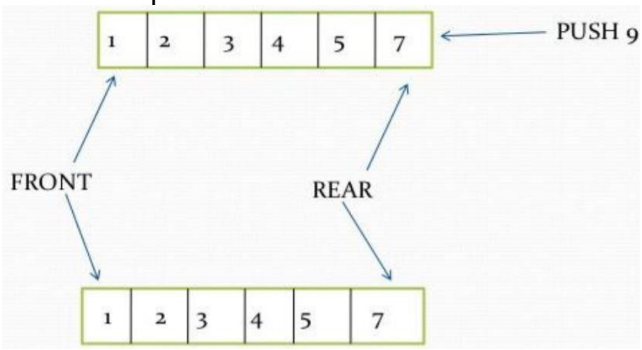
Insert element at Front

Insert front is a operation used to push an element into the front of the Deque.



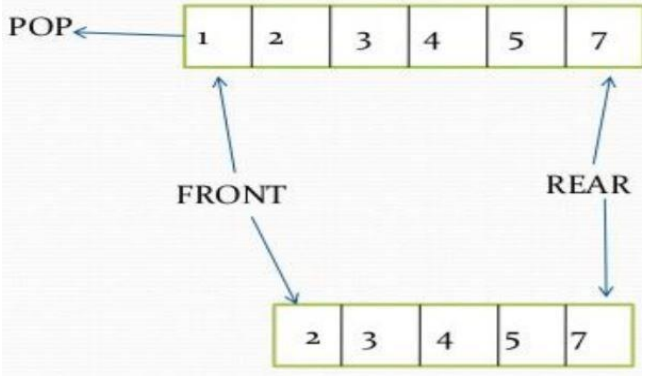
Insert element at back

Insert front is a operation used to push an element into the back of the Deque.



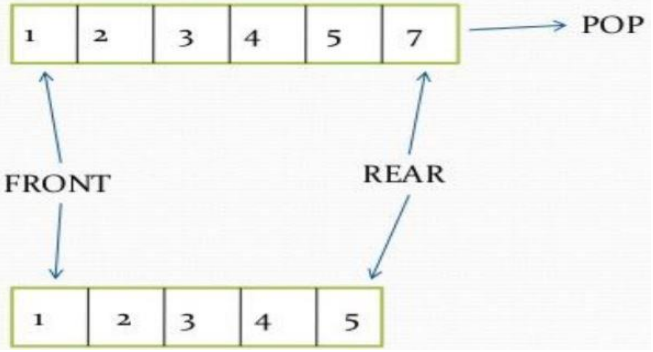
Remove element at front

Remove front is a operation used to pop an element on front of the Deque.



Remove element at back

Remove back is a operation used to pop an element on back of the Deque.



Applications of DEQUE:

- The A-Steal algorithm implements task scheduling for several processors (multiprocessor scheduling).

- The processor gets the first element from the deque.
- When one of the processor completes execution of its own threads it can steal a thread from another processor.
- It gets the last element from the deque of another processor and executes it.

References

[1] https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm
[2] https://www.edutechlearners.com/download/books/DS.pdf
[3]https://www.iare.ac.in/sites/default/files/DS_NOTES_BY_Dr_L_V_NARASIMHA_PRASAD_0.pdf
[4]http://cse.iitkgp.ac.in/~pds/notes/