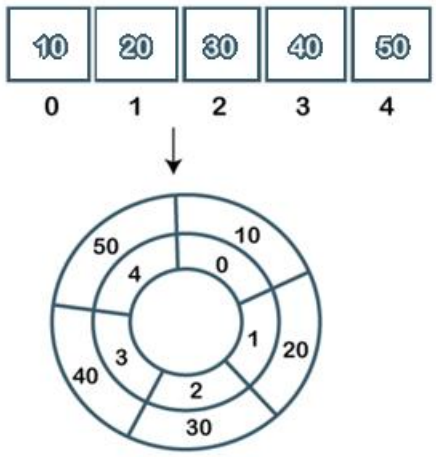
a Circular Queue

A circular queue is also a linear data structure like a normal queue that follows the FIFO principle but it does not end the queue; it connects the last position of the queue to the first position of the queue. If we want to insert new elements at the beginning of the queue, we can insert it using the circular queue data structure



In a **normal Queue**, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.

In the **circular queue**, when the rear reaches the end of the queue, then rear is reset to zero. It helps in refilling all the free spaces. The problem of managing the circular queue is overcome if the first position of the queue comes after the last position of the queue.

Conditions for the queue to be a circular queue

- Front ==0 and rear=n-1

If the above conditions is satisfied means that the queue is a circular queue.

Operations on Circular Queue

The following are the two operations that can be performed on a circular queue are:

- **Enqueue:** It inserts an element in a queue.
- **Dequeue:** It performs a deletion operation in the Queue.

```
#include < iostream >
using namespace std;
const int size =4;
#include<conio.h>
template<class T>

class CircularQ
{
private:
T q[size];
int front,rear;
int count;
public:
CircularQ()
{
front = 0;
rear = size - 1;
count = 0;
}
bool isFull()
{
return(count==size);
}
```

```
bool isEmpty()
{
return (count==0);
}
void enqueue(T item)
{
if(isFull())
{
cout<<"\nfull \n";
}
else
{
rear = (rear + 1) % size;
q[rear] = item;
count++;
}
}
int dequeue()
{
if(isEmpty())
{
cout<<"empty";
return 0;
}
else
{
int x = q[front];
front = (front + 1) % size;
count--;
return x;
}
}
int getSize()
{
return count;

}
void print_queue()
{
      for (int i = front; i <= rear; i++)
cout << "Position : " << i << " , Value  : " << q[i]<<endl;
}

};
```

```
int main()
{
CircularQ <int> qq;
cout<<qq.dequeue();
qq.enqueue(6);
qq.enqueue(2);
qq.enqueue(8);
qq.enqueue(1);
qq.enqueue(5);
qq.enqueue(7);
qq.print_queue();
while (qq.getSize())
cout << qq.dequeue() << endl;
qq.enqueue(10);

qq.print_queue();
getch();
return 0;
}
```

```
empty0
full

full
Position : 0 , Value  : 6
Position : 1 , Value  : 2
Position : 2 , Value  : 8
Position : 3 , Value  : 1
6
2
8
1
Position : 0 , Value  : 10
```