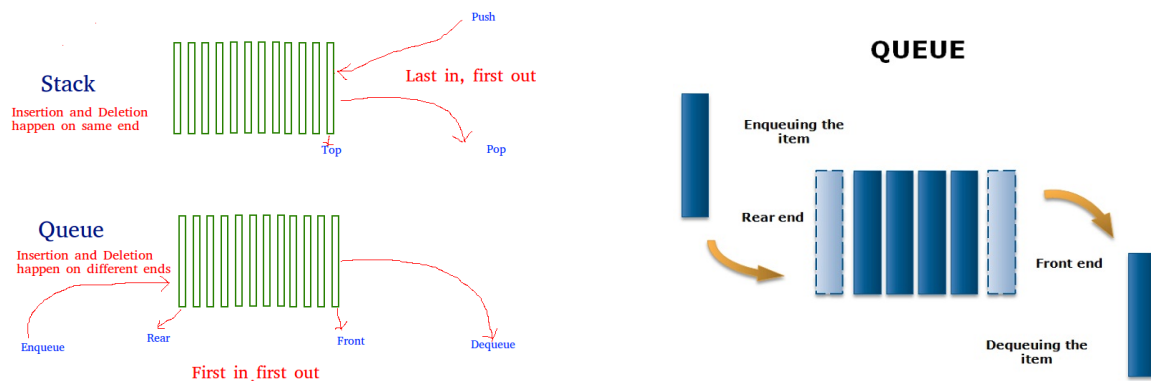


## Objective:

- Intro to data structure, queue.
- Applications of Queue.
- An optimized way of resizing the queue.

## What is Queue?

A data structure which exhibits FIFO/FCFS (first in first out / first come first serve) behaviour. It means the element added first will be removed first. In queue, the basic insertion and removal operations are named as enqueue and dequeue respectively. Enqueue operation adds an element on the rear/Tail end of the stack and dequeue operation removes an element from front/head end of the queue.



In stack, insertion is done on one end of List but in queue, insertion is done on one end and removal is done on other end of list. What is List? We shall talk in session!

## Where do you see such behavior in daily life?

- Queue/line of students in admission office or a line of persons in bank for bill payment etc.
- Printer Queue: jobs/docs submitted to a printer are serviced in order of arrival i.e. FIFO.
- Call center phone systems will use a queue to hold people in line until a service representative is free.



## Where do you see such behaviour/application in CS?

- In serving requests for single shared resource (printer, disk, CPU), generally, the most unbiased and appropriate method will be to process request to shared resource in FCFS.
- Buffers on mp3 players and portable cd players, ipod playlist. Playlist for jukebox - add songs to the end, play from the front of the list.
- When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
  - Your keyboard buffer as we discussed during File stream session.
- When programming a real-time system that can be interrupted (eg., By a mouse click or wireless connection), it is necessary to attend to the interrupts immediately, before proceeding with the current activity. If the interrupts should be handled in the same order they arrive, then a FIFO queue is the appropriate data structure.

There is another concept of queue other than FCFS and that is Priority Queue. We shall discuss about it in our upcoming lecture, actually, at the time of studying tree data structure. Priority Queue has its own amazing application. 😊



## Queue Implementation:

Two Ways:

- Linear Buffer
- Circular Buffer

We should always go for circular buffer implementation considering the time complexity of enqueue/dequeue operations.

### Implementing Queue by perceiving array as a linear buffer.

#### Version-1

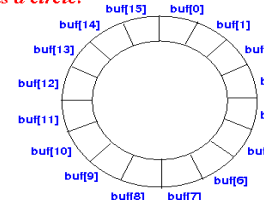
Queue initial status: r=rear=Tail rear point to location where next insertion will be done front is fixed: we shall always remove value at index 0.	<div style="text-align: center;"> <math>r</math>            Array index    0 1 2 3 4 5 6 7 8 9            Array values   <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>23</b>	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>89</b>	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>10</b>	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>19</b>	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 19 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>30</b>	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 19 30 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Remove/dequeue, it should be 23 Bad! Isn't it? We have to shift back values on every removal. If will take $O(N)$ time, where $N$ is the values count stored in queue If we want to improve this case, we have to introduce moving front end like rear instead of fixing it at 0. Let's see the second version which accommodate this improvement.	<div style="text-align: center;"> <math>r</math>            0 1 2 3 4 5 6 7 8 9            89 10 19 30 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>

#### Version-2

Queue initial status: f= front, r=rear f=-1, r=-1 rear/front, point to one location behind at which next insertion/deletion will be done	<div style="text-align: center;"> <math>f</math> <math>r</math>            Array index    0 1 2 3 4 5 6 7 8 9            Array values   <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>23</b>	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>89</b>	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>10</b>	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>19</b>	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 19 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Insert/enqueue <b>30</b>	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 19 30 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>
Remove/dequeue, it should be 23 Better than previous ☺ Removal in constant time but always? Let see! After removal of 23, There are total 4 values i.e. from $f+1 \sim r$	<div style="text-align: center;"> <math>f</math> <math>r</math>            0 1 2 3 4 5 6 7 8 9            23 89 10 19 30 <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> <span style="border: 1px solid black; padding: 2px;"> </span> </div>

Array:  
 buf[0] buf[1] ... buf[15]

Pretend array is a circle:



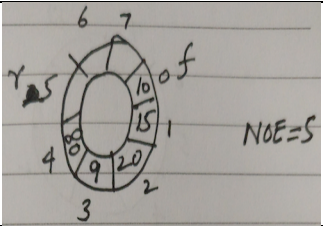
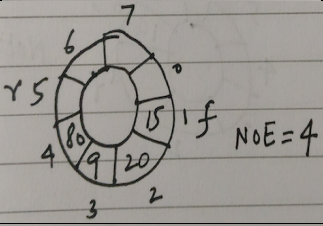
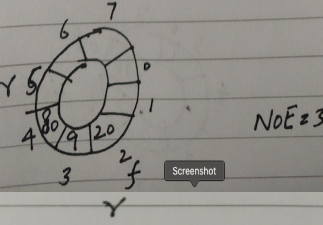
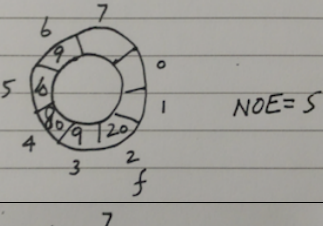
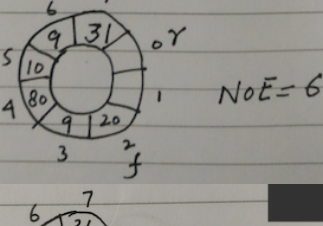
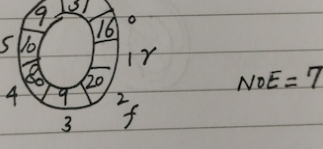


Remove/dequeue, it should be 89 Constant time again ☺	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td></td><td></td><td></td><td></td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30					
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30																											
Insert/enqueue <b>1</b>	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td>1</td><td></td><td></td><td></td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30	1				
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30	1																										
Insert/enqueue <b>2</b>	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td>1</td><td>2</td><td></td><td></td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30	1	2			
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30	1	2																									
Insert/enqueue <b>3</b>	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td>1</td><td>2</td><td>3</td><td></td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30	1	2	3		
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30	1	2	3																								
Insert/enqueue <b>4</b>	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td>1</td><td>2</td><td>3</td><td>4</td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30	1	2	3	4	
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30	1	2	3	4																							
Insert/enqueue <b>5</b>	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>23</td><td>89</td><td>10</td><td>19</td><td>30</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	23	89	10	19	30	1	2	3	4	5
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
23	89	10	19	30	1	2	3	4	5																						
Insert/enqueue <b>99</b> We can't insert. Why? We can't move 'r' further as it is at the last index. Why? Is the queue full? No! The array capacity is 10 and we have only 7 values in queue i.e. $f+1 \sim r$ . there are 3 slots, which are free ( $0 \sim f$ ) but our logic is unable to tackle them. In this case, we shall shift the elements from $f+1 \sim r$ to starting locations starting at 0.  This process will make the enqueue worst time bound $O(N)$ This version is better than the previous version practically but in terms of big $O$ , both linear buffer versions makes one of the operations enqueue/dequeue $O(N)$ .  But this case gives us idea to see this array as a circular buffer. What if we start moving the front/f and read/r in circular way? See the next implementation.	<table><tr><td colspan="2"></td><td colspan="2">f</td><td colspan="2"></td><td colspan="2">r</td><td colspan="2"></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>19</td><td>30</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>99</td><td></td><td></td></tr></table>			f				r				0	1	2	3	4	5	6	7	8	9	19	30	1	2	3	4	5	99		
		f				r																									
0	1	2	3	4	5	6	7	8	9																						
19	30	1	2	3	4	5	99																								

### Implementing Queue by perceiving array as a Circular buffer.

<p>Initial Queue configuration</p> <p>Rear/Tail/r = 0</p> <p>Front/Head/f = 0</p> <p>no. of elements = NOE = 0</p> <p>capacity = 8</p>		<p>NOE = 0 r = 0 f = 0</p>	
<p>Insert/enqueue <b>10</b></p> <p>Rear/r = 1</p> <p>Front/f = 0</p> <p>no. of elements = NOE = 1</p>		<p>NOE = 1</p>	
<p>Insert/enqueue <b>15</b></p> <p>Rear/r = 2</p> <p>Front/f = 0</p> <p>no. of elements = NOE = 2</p>		<p>NOE = 2</p>	



Insert/enqueue <b>20, 9, 80</b> Rear/r = 5 Front/f = 0 no. of elements = NOE = 5		
Remove/dequeue: we shall get 10 Rear/r = 5 Front/f = 1 no. of elements = NOE = 4		
Remove/dequeue: we shall get 15 Rear/r = 5 Front/f = 2 no. of elements = NOE = 3		
Insert/enqueue <b>10, 9</b> Rear/r = 7 Front/f = 2 no. of elements = NOE = 5		
Insert/enqueue <b>31</b> Rear/r = 0 Front/f = 2 no. of elements = NOE = 6		
Insert/enqueue <b>16</b> Rear/r = 1 Front/f = 2 no. of elements = NOE = 7		

## Queue ADT

Based on the above discussion, the Queue ADT should be implemented as circular buffer:

*Queue Resizing will work same as Stack in order to make amortized time constant. We have already discussed in detail about this process in Stack handout.*

*In tasks given to you in which you need to use queue: you will assume that copy ctor and assignment optr are not available in the ADT unless specified otherwise.*

**Stop here and start your own implementation instead of rushing towards Queue implementation given below.**

■  
■  
■  
■

```
template<typename T>
class Queue
{
private:
    int rear;
    int front;
    int capacity;
    int noOfElements;
    T * data;

    void reSize(int newSize)
    {
        T * temp = new T[newSize];
        for(int j=0,i=front; j<noOfElements; j++)
        {
            temp[j] = data[i];
            i=(i+1)%capacity;
        }
        this->~Queue();
        data = temp;
        capacity = newSize;
        rear = noOfElements;
        front=0;
    }
public:
    Queue()
    {
        rear=front=noOfElements=0;
        capacity=1;
        data = new T[capacity];
    }
    ~Queue()
    {
        if (!data)
            return;
        delete [] data;
        data = nullptr;
    }
    Queue(const Queue<T> &); //Do it yourself
    Queue<T> operator = (const Queue<T> & );
    //Do it yourself
};
```

```
void enqueue(T val)
{
    if (isFull())
    {
        reSize(capacity*2);
    }
    data[rear] = val;
    rear = (rear+1)%capacity;
    noOfElements++;
}

T dequeue()
{
    if(isEmpty())
        throw exception();
    T val = data[front];
    front = (front+1)%capacity;
    noOfElements--;
    if (noOfElements > 0 && noOfElements ==
        capacity/4)
        reSize(capacity/2);
    return val;
}

T getElementAtFront()
{
    if(isEmpty())
        throw exception();
    return data[front];
}

bool isEmpty()
{
    return noOfElements==0;
}

bool isFull()
{
    return noOfElements==capacity;
}

int getNoOfElements()
{
    return noOfElements;
}

int getCapacity()
{
    return capacity;
}
};
```

### For Practice

Attempt the problems at the following link.

<https://www.hackerearth.com/practice/data-structures/queues/basics-of-queues/practice-problems/>

### And Yes! go through STL

<http://www.cplusplus.com/reference/queue/queue/>