Spondon Ganguli
An Educator, Artist & Author

Data Structure – Part 2

By Spondon Ganguli

Spondon Ganguli
An Educator, Artist & Author

Chapter: Simple Data Structure (QUEUE contd.)

- Variations In Queue
  - Circular queue
    - Advantages of Circular queue
  - Deque (Double-ended queue)
    - Advantages of Deque
    - Two special types of deque
  - Priority Queue
- Applications Of Queue

Variations in Queue

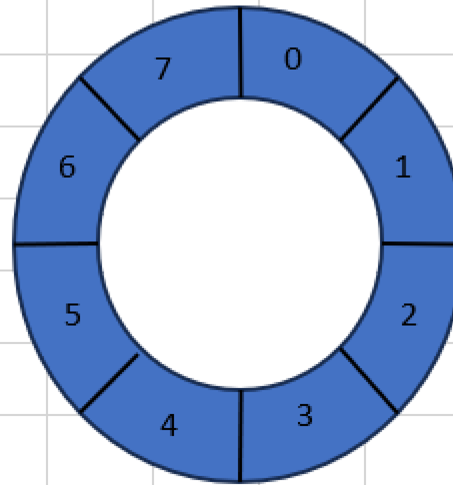
6.1 Circular queue

In a circular queue, all the elements are arranged in a circle, instead of a row or line. In the circular queue, a pointer may be used to ensure the maximum size is not exceeded. The pointer is reset to the initial value of 1, as and when the number of elements in the circular queue reaches the optimum value.

6.1.1 Advantages of Circular Queue

1. Very often, in an array queue, empty cells are generated at the front after deletion operations. Although there are many vacant cells, the queue overflow signal is given, preventing new entries. This deficiency is overcome in the Circular Queue.
2. If both rear and front reach MAX position, then insertion cannot be possible until both are again re-initialized to 0 in the case of Linear Queue whereas this re-initialization is not required in the case of Circular Queue.
3. Also, in the case of an array with a larger size where insertion and deletion take place with very few entries, in a simple queue most of the cells remain unused resulting in a wastage of memory. But in the circular queue, as shifting of rear and front to 0 do not required, this never happens. As all the cells will be used at some point in time.

Circular Queue through illustration



Pictorial representation of a Circular Queue

0	1	2	3	4	5	6	7	size=8
								QUEUE IS EMPTY
F=R=0								
0	1	2	3	4	5	6	7	
10	11	12	13	14	15	16		QUEUE IF FULL
F							R	
0	1	**2**	3	4	5	6	7	
X	X	**12**	13	14	15	16		2 ELEMENTS DELETED
		F					R	
0	1	**2**	3	4	5	6	7	
18		**12**	13	14	15	16	17	QUEUE IS FULL
	R	F						

Spondon Ganguli
An Educator, Artist & Author

Another example

0	1	2	3	4	5	6	7	INITIAL VALUES: F=0, R=0
10	12	14	15					TOTAL SIZE: 8
F				R				
10	12	14	15	17	20	22		QUEUE IS FULL
F							R	

INCREMENTING F & R	Index Pos
(0+1)%8	1
(1+1)%8	2
(2+1)%8	3
(3+1)%8	4
(4+1)%8	5
(5+1)%8	6
(6+1)%8	7
(7+1)%8	0

AFTER DELITING 3 ELEMENTS FROM THE QUEUE, FRONT=3

0	1	2	3	4	5	6	7
			15	17	20	22	
			F				R

ADDING ONE ELEMENT, REAR SHIFTED TO 0 AND FRONT=3

0	1	2	3	4	5	6	7
			15	17	20	22	31
R			F				

AFTER ADDING THREE MORE ELEMENTS, REAR=2 AND FRONT=3

0	1	2	3	4	5	6	7	
32	33		15	17	20	22	31	QUEUE IS FULL
		R	F					

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

Write a program in Java to implement the Circular Queue concept. The class structure is given as below:

Class name: **CircularQueue**

Data members/Instance variables:

- int CQue[ ]
- int F, R, capacity

Member functions/Methods:

- CircularQueue(int n) – parameterised constructor, setting F and R at 0 and capacity = n
- void pushAtRear(int s) – store one element (s) in the queue
- int removeFront( ) – delete one element from the queue at the front and return the same. If no more elements to delete, it will return -999
- void display( ) – to display all the elements present in the queue at any point in time.

```java
class CircularQueue
{
 int CQue[];//array to be used as a Queue
 int capacity;
 int F,R;//pointer to front and rear
 public CircularQueue(int n)
 {
     capacity=Math.abs(n);
     F=R=0;
     CQue=new int[capacity];
 }
 public void pushAtRear(int num)
 {
     if((R+1)%capacity==F)
     {
         System.out.println("Queue is full now");
         return;
     }
     CQue[R]=num;
     R=(R+1)%capacity;//rear here points to the next empty cell after each entry
     System.out.println("FRONT:"+F+" & REAR:"+R);
 }
```

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

```
public int removeFront()
{
    if(F==R)//front and rear at same index position
    {
        System.out.println("Queue has been emptied");
        return -999;
    }
    int n=CQue[F];
    F=(F+1)%capacity;
    System.out.println("FRONT:"+F+" & REAR:"+R);
    return n;
}
void display()
{
    if(R==F)
    {
        System.out.println("Queue is already empty");
        return;
    }
    int i;
    for(i=F;i!=R;i=(i+1)%capacity)
     System.out.print(CQue[i]+" ");
    System.out.println();
}
}
```

```
/*
Output
Element added at rear    FRONT:0 & REAR:1
Element added at rear    FRONT:0 & REAR:2
Element added at rear    FRONT:0 & REAR:3
Element added at rear    FRONT:0 & REAR:4
   Queue is full

   10 11 12 13

Element removed from front    FRONT:1 & REAR:4
Element removed from front    FRONT:2 & REAR:4
Element removed from front    FRONT:3 & REAR:4
Element removed from front    FRONT:4 & REAR:4
   Queue is empty

   FRONT:4 & REAR:0
   FRONT:4 & REAR:1
   FRONT:4 & REAR:2
   FRONT:4 & REAR:3
   Queue is full

   20 21 22 23
*/
```

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

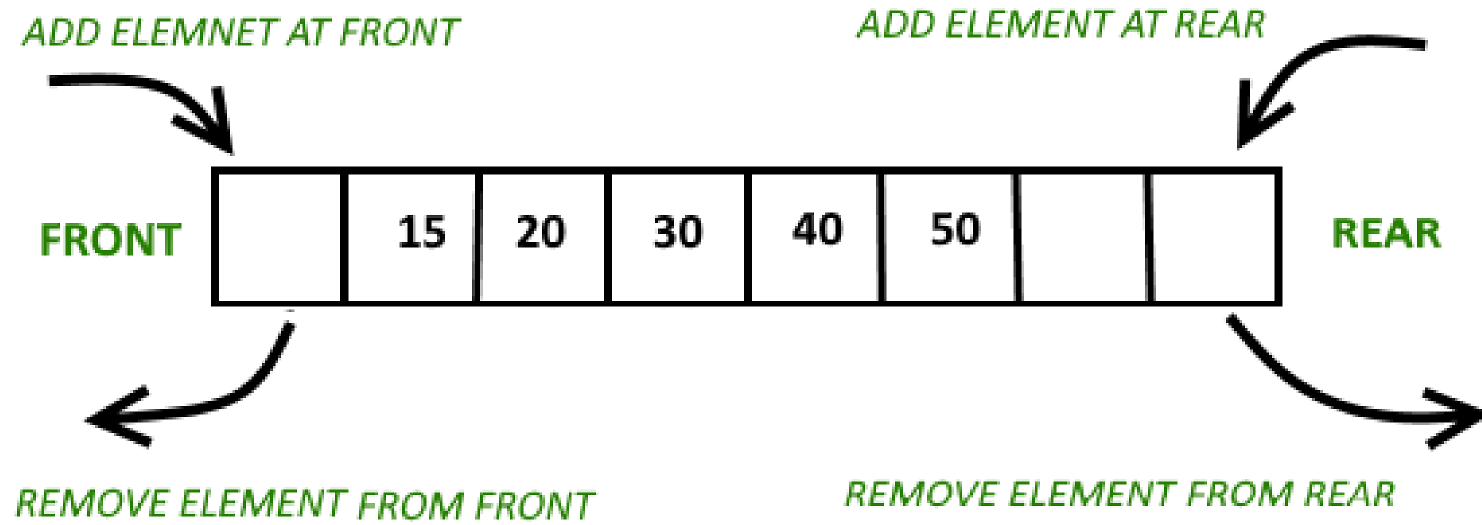
6.2 Deque (Double-ended queue)

A deque is a special form of a queue. It is a double-ended queue, in which data can be added and removed at either end. However, in a deque, neither data can be inserted nor remove data random from any other position. The word deque is pronounced either "deck" or "DQ". The deque stands for double-ended queue. A deque is an ordered collection of elements from which new elements can be added or deleted from either the first or the last position of the list but not in the middle.

6.2.1. Two special types of deque

i. **Input Restricted Deque** – a deque in which items may be deleted at either end, but the insertion of items is restricted at only one end, say rear, of the queue.
ii. **Output Restricted Deque** – a deque in which items may be inserted at either end, but deletion of items is restricted at only one end, say front, of the deque.

Spondon Ganguli
An Educator, Artist & Author



Spondon Ganguli
An Educator, Artist & Author

TYPES OF DEQUE

Input restricted Deque

- Elements can be inserted only at one end.
- Elements can be removed from both the ends.

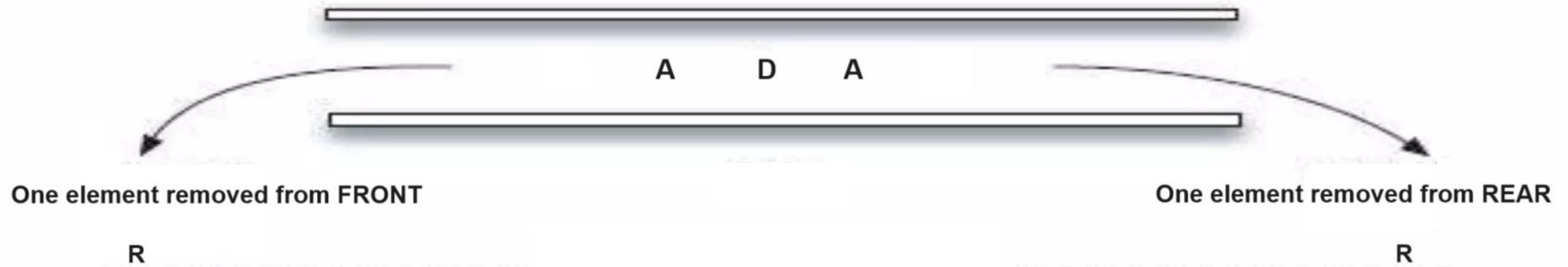
Output restricted Deque

- Elements can be removed only at one end.
- Elements can be inserted from both the ends.

Spondon Ganguli
An Educator, Artist & Author

Palindrome-checker

Added "RADAR" to the Queue



If they are matched, the step is repeated else aborted.
If the Queue becomes empty, the string is a Palindrome

Spondon Ganguli
An Educator, Artist & Author

Deque as Stack and Queue

As STACK

- When insertion and deletion is made at the same side.

As Queue

- When items are inserted at one end and removed at the other end.

Spondon Ganguli
An Educator, Artist & Author

APPLICATIONS OF DEQUE

- **Steal job scheduling algorithm**

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

- **Undo-Redo** operations in Software applications.

Spondon Ganguli
An Educator, Artist & Author

OPERATIONS IN DEQUE

- Insert element at back
- Insert element at front
- Remove element at front
- Remove element at back

Dequeue through illustration

Spondon Ganguli
An Educator, Artist & Author

0	1	2	3	4		FRONT	REAR
					EMPTY	0	0
F/R=0							
ADDING 3 ELEMENTS: 10, 12, 15							
10	12	15				0	3
F			R		INSERTION AT FRONT NOT POSSIBLE		
DELETE ONE ELEMENT AT FRONT: 10							
	12	15				1	3
	F		R				
ADDING 1 ELEMENT AT FRONT: 16							
16	12	15				0	3
F			R		INSERTION AT FRONT NOT POSSIBLE		
ADDING 1 ELEMENT AT REAR: 17							
16	12	15	17		FULL	0	4
F				R	INSERTION AT REAR NOT POSSIBLE		
DELETE TWO ELEMENTS FROM FRONT: 16, 12							
		15	17			2	4
		F		R	INSERTION AT REAR NOT POSSIBLE		
DELETE TWO ELEMENTS FROM REAR: 17, 15							
					EMPTY	2	2
		F/R					

Write a program in Java to implement the Double-ended queue concept. The class structure is given as below:

Class name: **DeQueue**

Data members/Instance variables:

- int DQue[ ] – the size of the queue is fixed at 5
- int Front, Rear

Member functions/Methods:

- DeQueue( ) – default constructor, setting Front and Rear at 0
- void pushAtRear(int s) – store one element (s) in the queue at the rear end
- void pushAtFront(int s) – store one element (s) in the queue at the front end
- int popAtFront( ) – delete one element from the queue at the front and return the same.
- int popAtRear( ) – delete one element from the queue at the rear and return the same.

  In both cases, if no more elements to delete, they will return -999
- void display( ) – to display all the elements present in the queue at any point in time.

Spondon Ganguli
An Educator, Artist & Author

```java
import java.util.*;
class DeQueue
{
    int Dque[]=new int[5];
    int front, rear;
    public DeQueue()
    {
        front=0;
        rear=0;
    }
    public void pushAtRear(int n)//inserting an element in a queue at rear end
    {
        if(rear==4)
        {
            System.out.println("Insertion at rear not possible");
            return;
        }
        Dque[rear++]=n;//first add the data then increment the rear
    }
    public void pushAtFront(int n)//inserting an element in a queue at front end
    {
        if(front==0)
        {
            System.out.println("Insertion at front not possible");
            return;
        }
        Dque[--front]=n;
    }
```

```java
public int popAtFront()//delete operation in a queue from front end
{
    if(front==rear)
    {
        System.out.println("Queue is empty");
        return -9999;
    }
    return Dque[front++];
}
public int popAtRear()//delete operation in a queue from rear end
{
    if(front==rear)
    {
        System.out.println("Queue is empty");
        return -9999;
    }
    return Dque[--rear];
}
public void display()
{
    if(front==rear)
    {
        System.out.println("Queue already empty");
        return;
    }
    for(int i=front;i<rear;i++)
    System.out.print(Dque[i]+" ");
    System.out.println();
}
}
```

Spondon Ganguli
An Educator, Artist & Author

6.3 Priority Queue

A priority queue is a data structure in which elements are inserted arbitrarily but deleted according to their priority. If elements have equal priorities, then the usual rule applies, that is first elements inserted should be removed first. Priority queues are of two types:

i. **Ascending Priority Queue**
ii. **Descending Priority Queue**

In ascending priority queues, the elements are inserted arbitrarily but it deletes the element having the smallest priority. In descending priority queues, the elements are inserted arbitrarily but it deletes the elements having the largest priority.

Spondon Ganguli
An Educator, Artist & Author

A Priority Queue is used when the objects are supposed to be processed based on priority. It is known that a Queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue need to be processed according to the priority, that's when the Priority Queue comes into play.

The Priority Queue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

Priority Queue is a data structure in which elements are ordered by priority, with the highest-priority elements appearing at the front of the queue.

Spondon Ganguli
An Educator, Artist & Author

PRIORITY QUEUE OPERATION

	0	1	2	3	
INITIAL QUEUE					EMPTY

OPERATION	PRIORITY	ELEMENT DELETED	QUEUE CONTENT 0	1	2	3
ENQUEUE(A)	P2		A			
ENQUEUE(B)	P1		A	B		
ENQUEUE(C)	P2		A	B	C	
DELQUEUE( )	P1	B	A	C		
ENQUEUE(D)	P3		A	C	D	
ENQUEUE(E)	P1		A	C	D	E
DELQUEUE( )	P1	E	A	C	D	
ENQUEUE(F)	P1		A	C	D	F

Spondon Ganguli
An Educator, Artist & Author

Some application areas of Priority Queue

- **Task Scheduling**: In operating systems, priority queues are used to schedule tasks based on their priority levels. For example, a high-priority task like a critical system update may be scheduled ahead of a lower-priority task like a background backup process.
- **Emergency Room:** In a hospital emergency room, patients are triaged based on the severity of their condition, with those in critical condition being treated first. A priority queue can be used to manage the order in which patients are seen by doctors and nurses.
- **Network Routing**: In computer networks, priority queues are used to manage the flow of data packets. High-priority packets like voice and video data may be given priority over lower-priority data like email and file transfers.
- **Transportation**: In traffic management systems, priority queues can be used to manage traffic flow. For example, emergency vehicles like ambulances may be given priority over other vehicles to ensure that they can reach their destination quickly.
- **Job Scheduling**: In job scheduling systems, priority queues can be used to manage the order in which jobs are executed. High-priority jobs like critical system updates may be scheduled ahead of lower-priority jobs like data backups.

Some programs for brainstorming:

Spondon Ganguli
An Educator, Artist & Author

Question 1.

Stack is a concept of storage which follows LIFO (Last In First Out) order that means the last element pushed into the stack will be popped out first from the stack. Queue is a concept of storage which follows FIFO (First In First Out) order that means the first element inserted in will be the first element extracted out from the queue.

Declare a class named StackofString with following declaration:-

Class name - **StackofString**

Data members:-

int top - to hold the index position
int N - the size of the array (max size 50)
String names[ ] - array of size N to store names

Member methods:-

StackofString(int n) - constructor to store the size of the array.
void pushName(String nm) - to push one names to the array
String popName( ) - to pop out one name from the array
void display( ) - to display content of the array

(a) Specify the above class with all the member methods.
(b) Now using the above class object in main method, implement the queue concept of storage where FIFO order will be followed. That means the name entered first will be extracted first.

Question 2.

Queue is a concept of storage which follows FIFO (First In First Out) order that means the first element **inserted in** will be the first element **extracted out** from the queue. Declare a class with following declaration:-

Class name - **QueueofString**

Data members:-

int front, rear - to hold the index position
int N - the size of the array (max size 50)
String names[ ] - array of size N to store names

Member methods:-

QueueofString(int n) - constructor to store the size of the array and front=rear=0
void enQueueName(String nm) - to insert one name to the array using rear index only
String delQueueName( ) - to extract one name from the array using front index only
void display( ) - to display content of the array

(a) Specify the above class with all the member methods.
(b) Now using the above class object in main method, implement the stack concept of storage where LIFO order will be followed i.e. the name entered last will be extracted first. Use two objects of Queue in main( ).

Thank you