Spondon Ganguli
An Educator, Artist & Author

# *Chapter: Simple Data Structure*

What is a Data Structure?
- Abstract Data Types

Classification Of Data Structure
- Linear & Non-linear data structure
- Homogeneous & non-homogeneous data structure
- Static & Dynamic data structure

Stack Concept Of Data Structure
- Push Operation in a Stack
- Pop Operation in a Stack

Applications Of Stack

Queue Concept Of Data Structure
- Insert operation (Enqueue) in a Queue
- Delete operation (Delqueue) in a Queue

Spondon Ganguli
An Educator, Artist & Author

## 1. What is a Data Structure?

A data structure is basically a collection of elements whose organization is characterized by a set of operations that are used to store and retrieve the individual element from it.
Operations performed on data structures are –

1. Traversing – processing of each element of the specific data structure exactly once.
2. Searching – searching a particular data item and its location from the specific data structure.
3. Insertion – addition of new elements from the data structure, if it is not empty.
4. Deletion – deletion of existing elements from the data structure, if it is not empty.
5. Sorting – arranging data elements in either ascending or descending order depending upon a certain key.

## 1.2 Abstract Data Types

The term ADT refers to a programmer defined data type together with a set of operations that are performed on that data. It is called abstract just to distinguish it from basic built-in data types such as int, char and double. In other words, the definition of an ADT consists of two main points – the internal representation of the ADT's data and the functions to manipulate this data.
In C programming, we can define an ADT by using the keywords – typedef and struct and defining the functions thus data abstraction.
In Java programming, we can define an ADT by using the keywords – class and interface and defining the functions thus data abstraction.

Spondon Ganguli
An Educator, Artist & Author

## 2. Classification Of Data Structure

### 2.1 Linear & Non-linear data structure

A data structure is said to be linear if its elements are sequentially stored in definite order.
In linear data structures, processing of data elements is possible in linear fashion, i.e. data elements are processed one by one sequentially. Array, stack, queue and linked list are example of linear data structures.
If the elements are not sequentially stored, then the data structure is said to be Non-linear. In non-linear data structures, processing of data elements is not possible in linear fashion. Tree and graph are example of non-linear data structure.

### 2.2 Homogeneous & Non- Homogeneous data structure

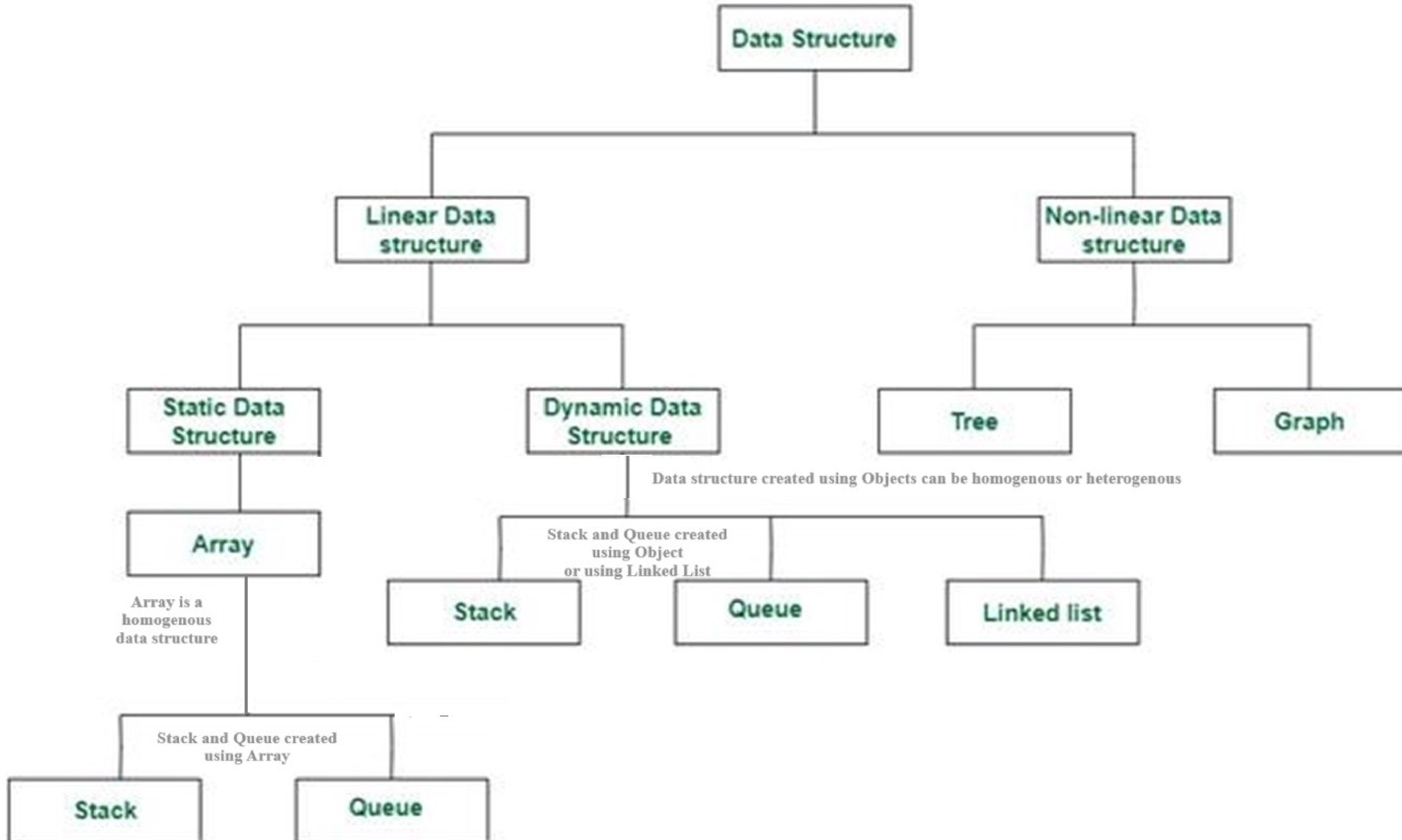
A data structure is said to be homogeneous if it's all elements are of same data type. For example - array. Whereas a data structure is said to be non-homogeneous if any one or all of its elements are of different data type. For example – structure/class/interface.

### 2.3 Static & Dynamic data structure

A data structure is said to be static, if during runtime, its size and associated memory allocation are fixed and definite. For example - array, stack, queue etc. where as if the data structure shrinks (we can reduce the size) or expands (we can increase the size) as per requirements during runtime, it is called dynamic. Some examples are linked list, stack and queue using linked list, tree, graph etc. Array is a static concept whereas pointers/objects are dynamic.

# Types of Data Structure

Spondon Ganguli
An Educator, Artist & Author



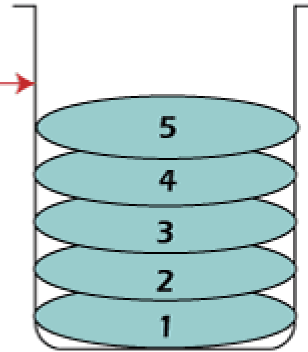
Spondon Ganguli
An Educator, Artist & Author

# Stack

TOP

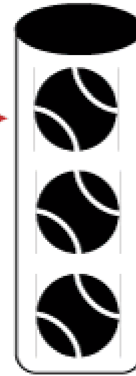
Stack of Coins

TOP

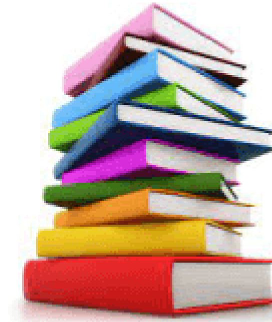



Stack of Plates

TOP



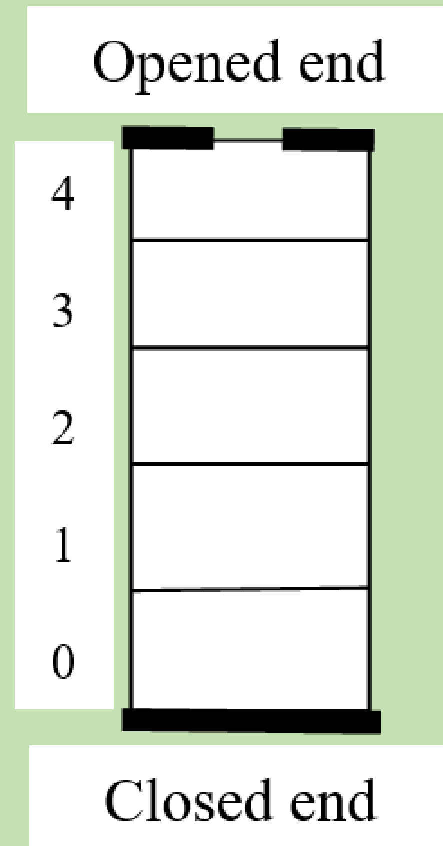
Can of Tennis Balls



Stack of Books

# 3. Stack Concept of Data Structure

In the array, elements can be inserted anywhere, i.e. at the beginning, at the end or at any position in between. A stack is a linear data structure in which insertion and deletion operations are permitted only at the end. The LIFO principle is implemented in a stack. A stack is a dynamic data structure. It can grow or shrink. The number of its elements can increase or decrease.




We can add elements only at the open end and we can remove elements only from the open end. Thus stack follows LIFO principle – Last In First Out

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

# Applications of Stack

- Convert infix expression to postfix and prefix expressions
- Evaluate the postfix expression
- Reverse a string
- Check well-formed (nested) parenthesis
- Reverse a string
- Process subprogram function calls
- Parse (analyze the structure) of computer programs
- Simulate recursion
- In computations like decimal to binary conversion
- In Backtracking algorithms (often used in games)

Spondon Ganguli
An Educator, Artist & Author

## Basic Stack operations:

We can perform the following operations on a stack-

- **push()** to insert an element into the stack
- **pop()** to remove an element from the stack
- **top() / peep()** Returns the top element of the stack.
- **isEmpty()** returns true if the stack is empty else false.
- **size()** returns the size of the stack.

Spondon Ganguli
An Educator, Artist & Author

## *3.1 Push Operation in a Stack*

The push operation inserts one element at a time in the stack. The Push operation in a stack is implemented in two parts.

(i) Top is incremented one step up of right/top if STACK OVERFLOW does not occur

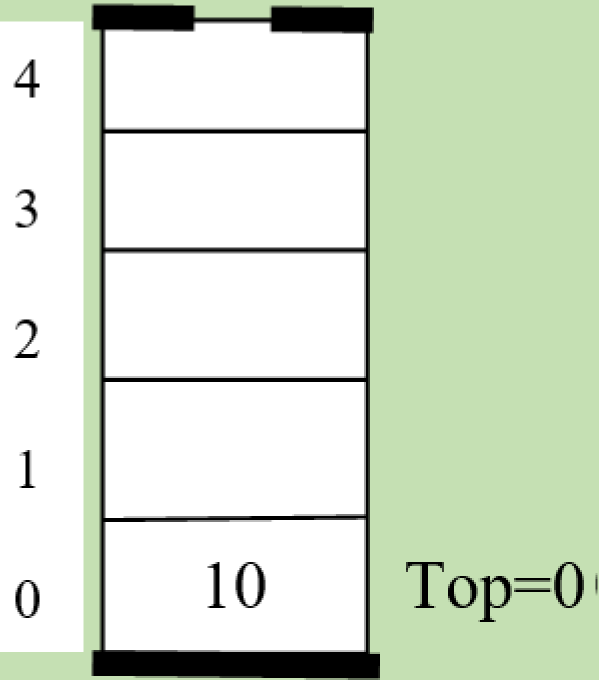
(ii) The new element is inserted at the top or at the right position.

**Algorithm for PUSH operation (Initially top is at -1)**

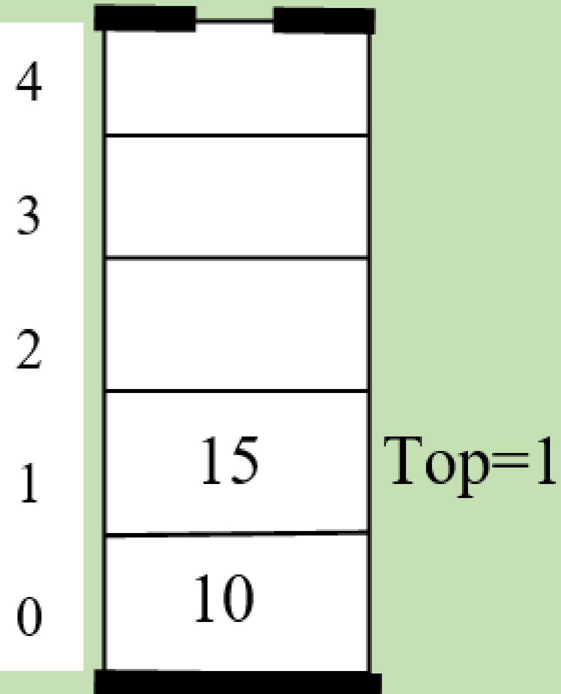
- **Step 1 – Increment top (Top++)**
- **Step 2 -** Check whether the **stack** is **FULL**. (**top == SIZE-1**)
- **Step 2 -** If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT FULL**, then set stack[top] to value (**stack[top] = value**).

Spondon Ganguli
An Educator, Artist & Author

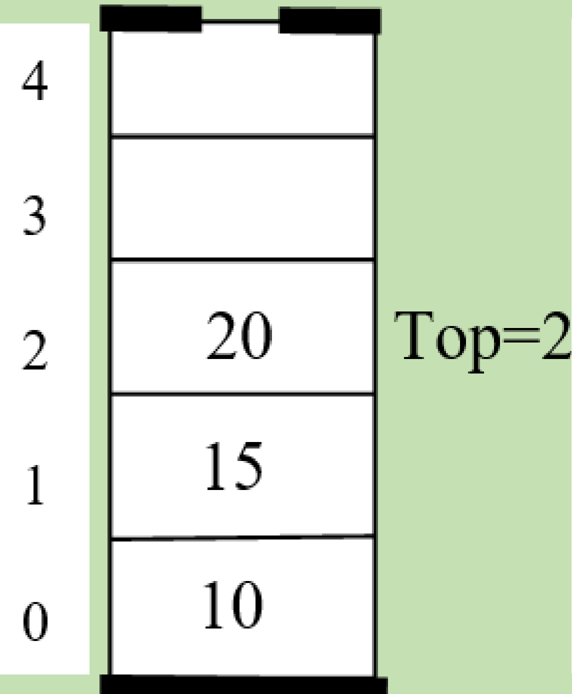
1. Push(10)



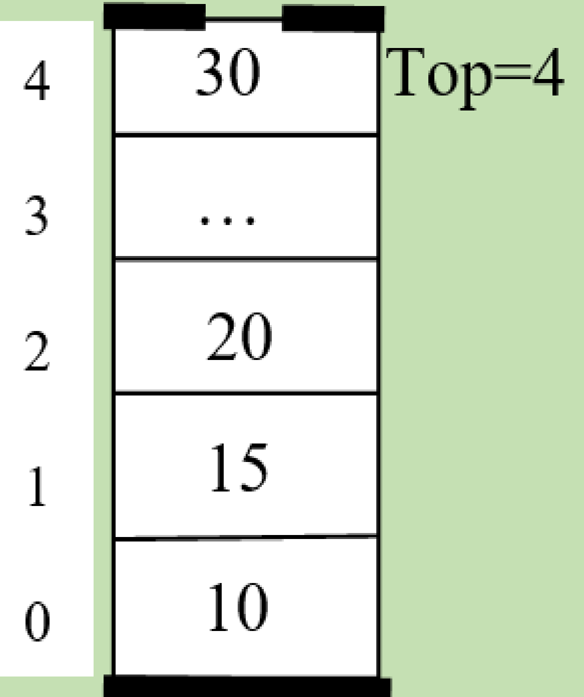

2. Push(15)




3. Push(20)




5. Push(30)




6. Push(35) **!!! STACK OVERFLOW !!!**

Spondon Ganguli
An Educator, Artist & Author

## *3.2 Pop Operation in a Stack*

The pop operation deletes one element at a time from a stack. The Pop operation in a stack is implemented in two parts.

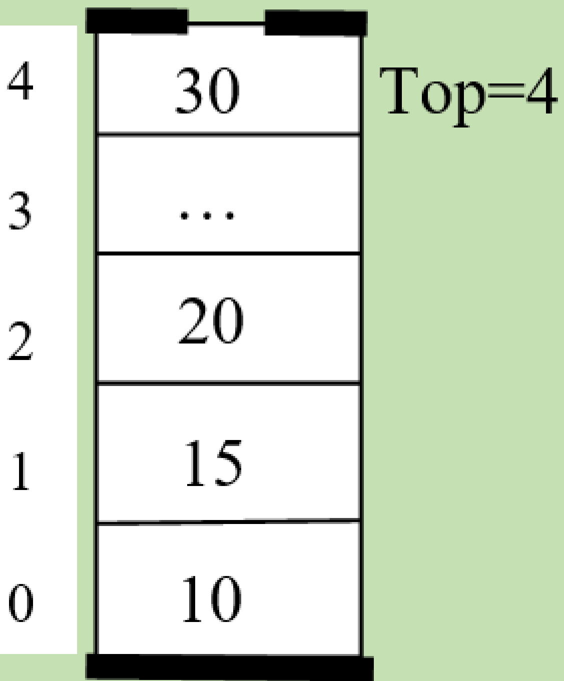
i. One element is extracted from the stack unless STACK UNDERFLOW occurred
ii. The top is decremented by one step down or left.

### Algorithm for POP operation

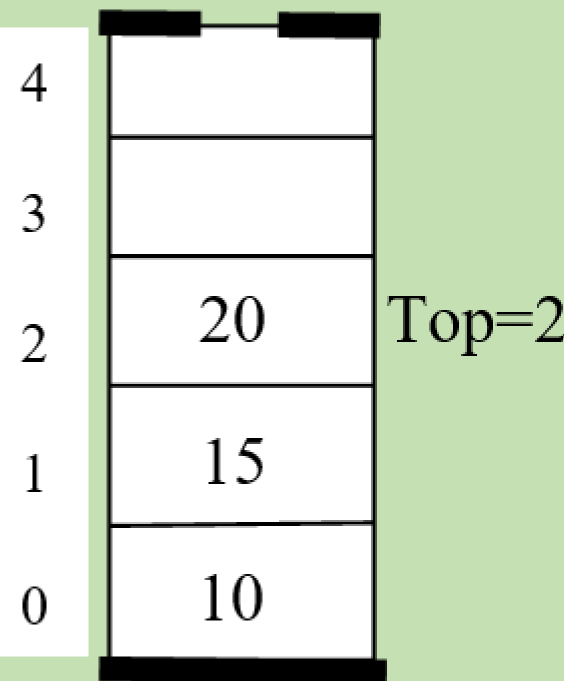
- **Step 1 -** Check whether the **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then delete **stack[top]** and decrement the **top** value by one (**top--**).

Spondon Ganguli
An Educator, Artist & Author

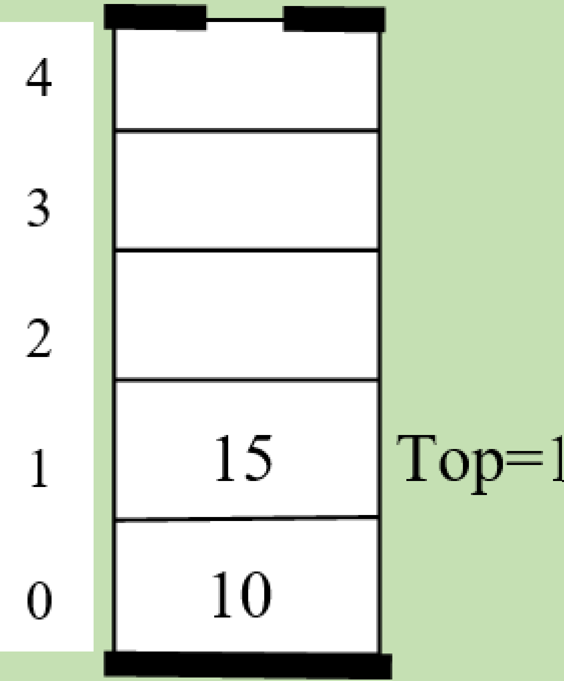
Initial stack



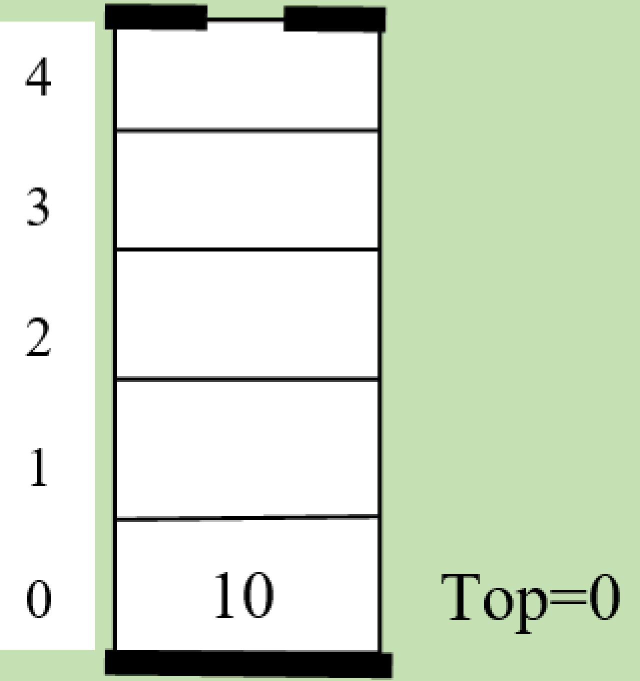

Stack after 2 Pop( )




3rd Pop( )




4th Pop( )




5th Pop( ) -> Top =-1

6th Pop( ) **!!! STACK UNDERFLOW !!!**

Spondon Ganguli
An Educator, Artist & Author

## *3.3 Display Operation in a Stack*

### Algorithm for display operation

- **Step 1 -** Check whether the **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define a variable **'i'** and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).
- **Step 4 -** Repeat the above step until the **i** value becomes '-1'.

Spondon Ganguli
An Educator, Artist & Author

Write a program in Java to implement the Stack concept through Array. The class structure is given as below:

Class name: **StackofInt**

Data members/Instance variables:

- int num[ ] – a stack on integer to store 5 integers in an array
- int top – to store the top index of the stack

Member functions/Methods:

- StackofInt( ) – default constructor to set the top at -1
- void push(int s) – store one element (s) to the stack
- int pop( ) – delete one element from the stack and return the same. If no more elements to delete, it will return -999
- void display( ) – to display all the elements present in the stack at any point in time.

In the main( ) method, create an object of **StackofInt** class and call the functions accordingly.

Spondon Ganguli
An Educator, Artist & Author

```
import java.util.*;
class StackofInt//class to represent stack of integer
{
    int num[];//array of integer(represented as a Stack)
    int top;//pointer to access the index position in the stack
    public StackofInt()
    {
        top=-1;//intial position of the pointer top
        num=new int[5];//declaring the array and assigning with 0
    }

    public void push(int s)//insertion operation in stack, it will add one element to the stack
    {
        top++;//increasing the top at the begining
        if(top==5)
        {
            System.out.println("Stack full");
            top--;//to take the top at previous position
            return;
        }
        num[top]=s;//storing the num into the stack
    }
```

Spondon Ganguli
An Educator, Artist & Author

```java
public int pop()//deletion operation in stack
{
    if(top==-1)
    {
        System.out.println("Stack is empty");
        return -999;
    }
    int s=num[top];
    top--;//decrementing one at a time
    return s;
}

public void display()
{
    System.out.println("\nDisplay method invoked");
    if(top==-1)
        System.out.println("Stack is already Empty");
    for(int i=top;i>=0;i--)
        System.out.println(num[i]);
}
```

```
    public static void main(String args[])
    {   Scanner sc=new Scanner(System.in);
        StackofInt obj2=new StackofInt();//creating an object
        int i,s;
        while(true)//loop for insertion operation
        {   System.out.print("enter a number");
            s=sc.nextInt();
            obj2.push(s);
            System.out.print(" added at "+obj2.top);
            if(obj2.top<4)
            {   System.out.print("\nenter another 1/0");
                i=sc.nextInt();
                if(i!=1)
                    break;
            }
            else
            break;
        }
        obj2.display();
        System.out.println("\nPop method invoked");
        while(true)//loop for deletion
        {   s=obj2.pop();
            if(s!=-999)
                System.out.println("Element deleted:"+s+" Position of Top:"+obj2.top);
            else
                break;
        }
        obj2.display();
    }
}
```

Spondon Ganguli
An Educator, Artist & Author

BlueJ: Terminal Window - DataStructure_Stack_Queue

Options

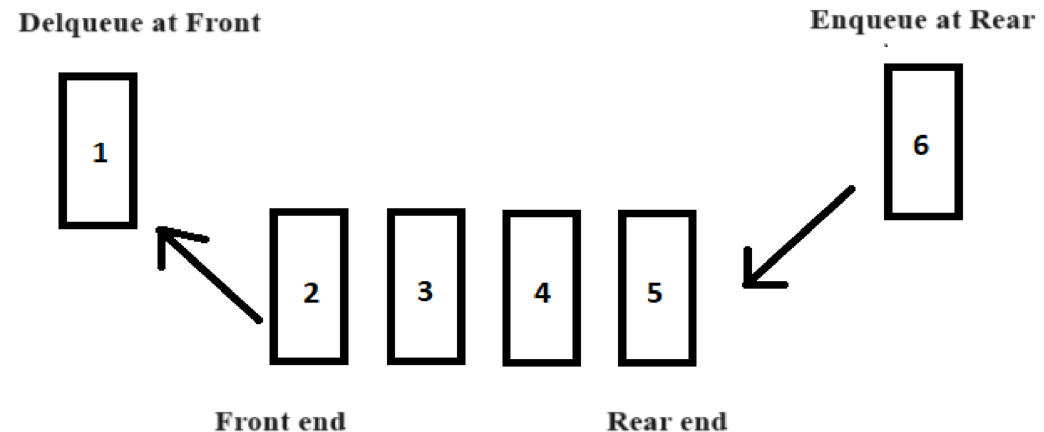
```
enter a number10
 added at 0
enter another 1/01
enter a number12
 added at 1
enter another 1/01
enter a number14
 added at 2
enter another 1/01
enter a number16
 added at 3
enter another 1/01
enter a number18
 added at 4
Display method invoked
18
16
14
12
10

Pop method invoked
Element deleted:18 Position of Top:3
Element deleted:16 Position of Top:2
Element deleted:14 Position of Top:1
Element deleted:12 Position of Top:0
Element deleted:10 Position of Top:-1
Stack is empty
```

Spondon Ganguli
An Educator, Artist & Author

# 4. Queue Concept of Data Structure

A queue is a data structure with a FIFO paradigm. It is implemented either in an array or a linked list. The queue is a collection of data elements in which all insertions are made at one end, called the rear, of the queue and all deletions are made at the other end, called the front, of the queue. The typical characteristic of a queue is that the first element to be inserted into a queue is the first element to be deleted.

We can add elements only at the rear end and
we can remove elements only from the front end.
Thus queue follows FIFO principle – First In First Out

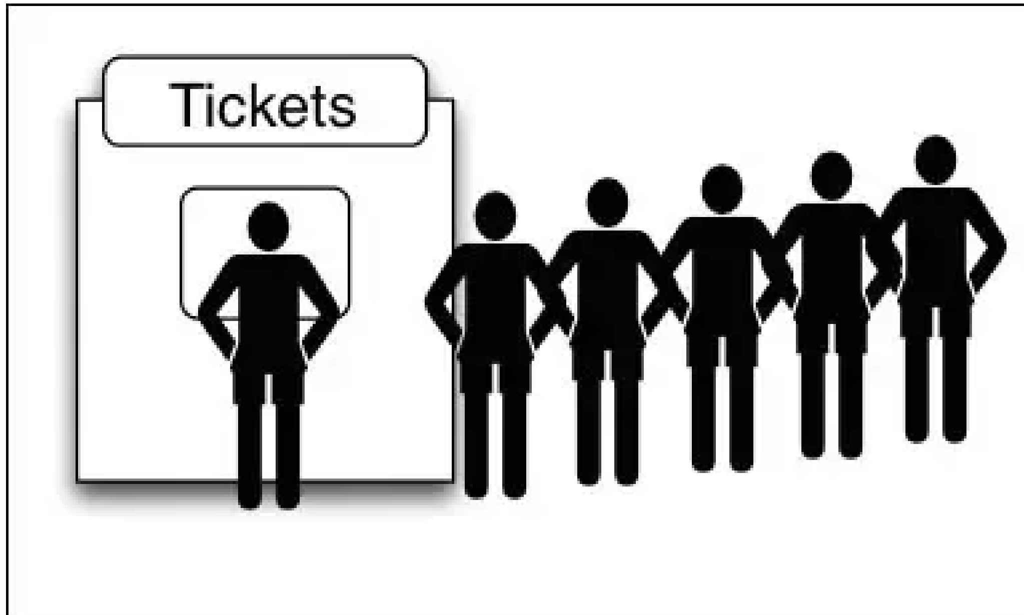


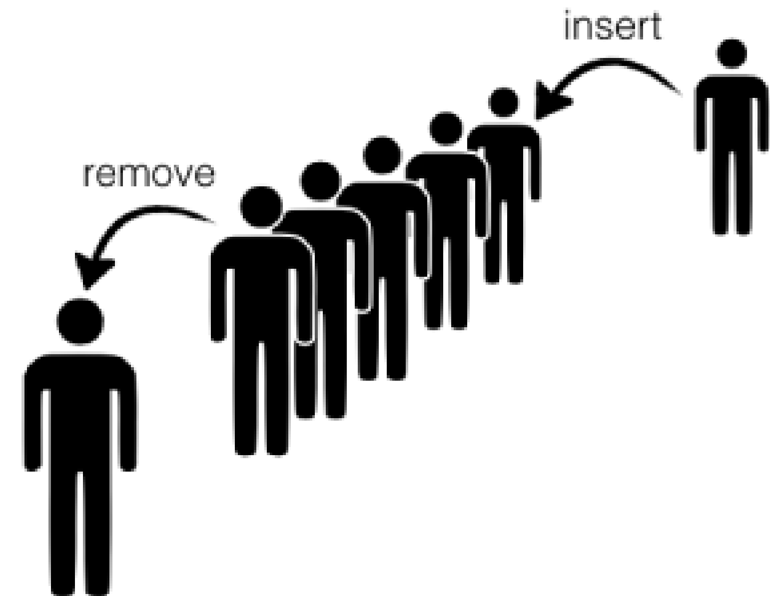

Queue Data Structure

# Some real-life examples of QUEUE

Spondon Ganguli
An Educator, Artist & Author

- Luggage checking machine
- Vehicles on toll tax bridge
- One way exits
- Patients waiting outside the doctor's clinic
- Phone answering systems
- Cashier line in a store



Spondon Ganguli
An Educator, Artist & Author

# Here are some applications of queues

- CPU scheduling- to keep track of processes for the CPU
- Handling website traffic - by implementing a virtual HTTP request queue
- Printer Spooling - to store print jobs
- In routers - to control how network packets are transmitted or discarded
- Traffic management - traffic signals use queues to manage intersections

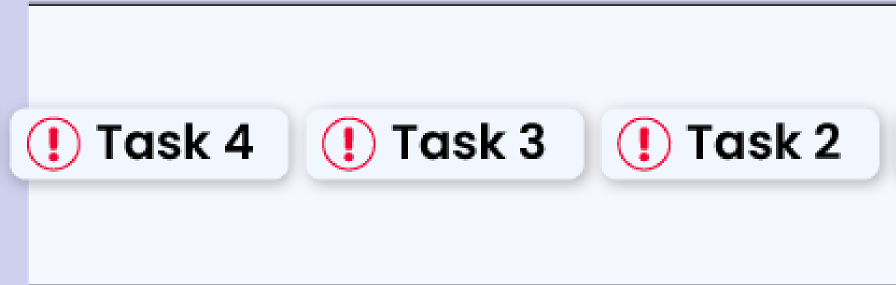
Spondon Ganguli
An Educator, Artist & Author

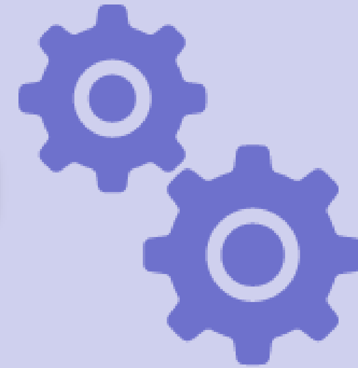
# PROCESSING OF A TASK IN A QUEUE

Pool of tasks

Task 13
Task 14
Task 15
Task 10
Task 6
Task 16
Task 5
Task 8
Task 7
Task 9
Task 11
Task 12

In the queue



Processed task

**Basic Queue operations:**

We can perform the following operations on a queue-

- **Enqueue()** – This is the process of adding or storing an element to the rear end (back-end) of a queue.
- **Delqueue()** – It refers to removing or accessing an element from the front end of a queue.
- **isEmpty()** – It checks if the queue is empty.
- **isFull()** – It checks if the queue is full.

## *5.1 Insert operation (Enqueue) in a Queue*

The insert operation inserts one element at a time in the queue. The Enqueue operation in a queue is implemented in two parts.

(i) The Rear index is incremented one step right if QUEUE OVERFLOW does not occur.

(ii) The new element is inserted at the Rear index.

**Algorithm for ENQUEUE operation**

- **Step 1 -** Check whether the **queue** is **FULL**. (**rear == SIZE-1**)
- **Step 2 -** If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT FULL**, then increment the **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

**SIMPLE QUEUE ENQUEUE OPERATION**

Let us take a queue of size 5

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| | | | | |
| Front/Rear = 0 | | | **Queue is EMPTY** | |

In the beginning, when
the QUEUE is empty -
Both front and rear points to 0

When the rear reached the last index,
the QUEUE becomes FULL and no more
ENTRY allowed.

When the first element is added to the queue, the rear moves to 1

Enqueue(10)

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **10** | | | | |

Front=0 and Rear=1

Enqueue(11)

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **10** | **11** | | | |

Front=0 and Rear=2

Enqueue(12)

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **10** | **11** | **12** | | |

Front=0 and Rear=3

Enqueue(14)

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **10** | **11** | **12** | **14** | |

Front=0 and Rear=4 **Queue is FULL**

Here, you can see, the rear points to the next empty cell after the last entry. Thus rear always points to an empty cell in a queue.

Spondon Ganguli
An Educator, Artist & Author

## *5.2 Delete operation (Delqueue) in a Queue*

The delete operation deletes one element at a time from the queue. The Delqueue operation in a queue is implemented in two parts.

i. One element is extracted from the queue unless QUEUE UNDERFLOW occurred
ii. The front index is incremented by one step right.

**Algorithm for POP operation**

- **Step 1 -** Check whether the **queue** is **EMPTY**. (**front == rear**)
- **Step 2 -** If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as the deleted element. Then check whether both **front** and **rear** are equal (**front == rear**), if it is **TRUE**, then set both **front** and **rear** to '**-1**' (**front = rear = -1**).

## SIMPLE QUEUE DELQUEUE OPERATION

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **10** | **11** | **12** | **14** | |

Front=0 and Rear=4 **Queue is FULL**

Initially, the QUEUE is FULL
So, the front is at 0 and the rear is at 4

At first delete operation, the front moved forward at 1

Delqueue( )

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| | **11** | **12** | **14** | |

Front=1 and Rear=4 **Element deleted: 10**

Delqueue( )

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| | | **12** | **14** | |

Front=2 and Rear=4 **Element deleted: 11**

Here, you can see as one element removed from the queue, the front moved one cell towards the rear.

Delqueue( )

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| | | | **14** | |

Front=3 and Rear=4 **Element deleted: 12**

Delqueue( )

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| | | | | |

Front=4 and Rear=4 **Queue is EMPTY**

When the front and rear points to the same index, it means the QUEUE is EMPTY

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

## *5.3 Display Operation in a Queue*

**Algorithm for display operation**

- **Step 1 -** Check whether the **queue** is **EMPTY**. (**front == rear**)
- **Step 2 -** If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.
- **Step 4 -** Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until the '**i**' value reaches to **rear** (**i** <= **rear**)

Spondon Ganguli
An Educator, Artist & Author

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

Class name: **Queue**

Data members/Instance variables:

- int ar[ ] – a queue on integer to store 5 integers in an array
- int front – to store the front index of the queue
- int rear – to store the rear index of the queue

Member functions/Methods:

- Queue( ) – default constructor to set the front and rear at 0
- void enqueue(int n) – store one element (n) to the queue
- int delqueue( ) – delete one element from the queue and return the same. If no more elements to delete, it will return -999
- void display( ) – to display all the elements present in the stack at any point in time.

Spondon Ganguli

```java
import java.util.*;
class Queue
{
    int ar[];
    int front, rear; //two pointers that is used to point to the first and last cell in the queue
    public Queue()
    {
        front=rear=0;//they are already at the 1st index and the queue is empty
        ar=new int[5];
    }

    public void enqueue(int n)//inserting an element in a queue
    {
        if(rear==4)
        {
            System.out.println("Queue is full");
        }
        ar[rear++]=n;//first add the data then increment the rear
    }
```

```java
public int delqueue()//delete operation in a queue
{
    if(front==rear)//both points to same index means the queue is empty
    {
        System.out.println("Queue is empty");
        front=rear=0;//repostioning the pointers to 0
        return -999;
    }
    return ar[front++];
}

public void display()
{
    if(front==rear)
    {
        System.out.println("Queue already empty");
        return;
    }
    System.out.println("Queue elements are: ");
    for(int i=front;i<rear;i++)
        System.out.print(ar[i]+" ");
    System.out.println();
}
```

Spondon Ganguli
An Educator, Artist & Author

```
public static void main(String args[])
{    Scanner sc=new Scanner(System.in);
     Queue obj2=new Queue();//creating an object
     int s;
     while(true)//loop for insertion operation
     {
          System.out.print("enter a number to be added at "+obj2.rear+" : ");
          s=sc.nextInt();
          obj2.enqueue(s);
          if(obj2.rear==4)
               break;
     }
     obj2.display();
     System.out.println("\nDelqueue method invoked");
     while(true)//loop for deletion
     {
          s=obj2.delqueue();
          if(s!=-999)
               System.out.println("Element deleted:"+s+" Position of Front:"+obj2.front);
          else
               break;
     }
}
}
```

Spondon Ganguli
An Educator, Artist & Author

Spondon Ganguli
An Educator, Artist & Author

BlueJ: Terminal Window - DataStructure_Stack_Queue

Options

```
enter a number to be added at 0 : 10
enter a number to be added at 1 : 11
enter a number to be added at 2 : 12
enter a number to be added at 3 : 13
Queue elements are:
10 11 12 13

Delqueue method invoked
Element deleted:10 Position of Front:1
Element deleted:11 Position of Front:2
Element deleted:12 Position of Front:3
Element deleted:13 Position of Front:4
Queue is empty
```

# Differentiate between Stack and Queue forms of data structure.

Spondon Ganguli
An Educator, Artist & Author

| Stack | Queue |
|---|---|
| It is a data structure that follows the Last In First Out (LIFO) technique for operation. | It is a data structure that follows the First In First Out (FIFO) technique for operation. |
| The stack requires only one pointer/counter (called Top) for moving within the stack. | The Queue requires two pointers/counters (named Front & Rear) for moving within the stack. |
| The top is responsible for insertion as well as deletion operations in the Stack. | The rear is responsible for insertion whereas the Front is responsible for deletion in the Queue. |
| After deletion, if the stack gets emptied, the top remains in its own position. | After deletion, if the queue gets emptied, the front and rear need to be placed at -1. |
| Operational areas of the stack:<br>• Recursion<br>• Arithmetic evaluation<br>• Parenthesis evaluation<br>• Reversing a string<br>• Infix to prefix and postfix | Operational areas of the queue:<br>• Printing sequence<br>• CPU scheduling<br>• Parallel processing<br>• BFS and DFS in Graph |

Another stack program using a String type array to hold the names of 5 students.

```
import java.util.*;
class StackofString
{
   String names[]=new String[5];
   int top;
   public StackofString()
   {
      top=-1;
   }
   public void push(String s)
   {
      top++;
      if(top==5)
      {
         System.out.println("Stack overflowed");
         top--;
         return;
      }
      names[top]=s;
   }
   public String pop()
   {
      if(top==-1)
      {
         System.out.println("Stack is empty");
         return null;
      }
      return (names[top--]);
   }
 public void display()
  {
     System.out.println("Stack elements:");
     if(top==-1)
     {
        System.out.println("Stack empty");
        return;
     }
     for(int i=top; i>=0;i--)
        System.out.println(names[i]);
  }
  public static void main(String args[])
  {
     Scanner obj=new Scanner(System.in);
     StackofString obj2=new StackofString();
     String s;
     for(int i=1;i<=5;i++)
     {
        System.out.println("enter a name");
        s=obj.nextLine();
        obj2.push(s);
     }
     obj2.display();
     while(obj2.top>=0)
        System.out.println("Element deleted:"+obj2.pop());
  }
}
```

Spondon Ganguli
An Educator, Artist & Author

BlueJ: Terminal Window - DataStructure_Stack_Queue

Options

```
enter a name
aman
enter a name
bimal
enter a name
chirag
enter a name
deep
enter a name
naman
Stack elements:
naman
deep
chirag
bimal
aman
Element deleted:naman
Element deleted:deep
Element deleted:chirag
Element deleted:bimal
Element deleted:aman
```

# Some practice programs for the students:

Spondon Ganguli
An Educator, Artist & Author

**Question 1**

In a computer game, a vertical column and a pile of rings are displayed. The objective of the game is to pile up rings on the column till it is full. It can hold 10 rings at the most. Once the column is full, the rings have to be removed from the top till the column is empty and then the game is over. Define the class RingGame with the following details:

**Class name : RingGame**

**Data members/instance variables**

ring [ ] : array to hold rings (integer)
max : integer to hold maximum capacity of ring array
upper : integer to point to the upper most element

**Member functions :-**

RingGame(int m) constructor to initialize, max = m & upper to –1.
void jump-in(int) adds a ring to the top of the column, if possible, otherwise displays a message "Column full. Start removing rings".
void jump-out( ) removes the ring from the top, if column is not empty otherwise, outputs a message, "Congratulations. The game is Over".

Specify the class RingGame giving the details of the constructor and functions void jump-in(int) and void jump-out( ). Also define the main function to create an object and call methods accordingly to enable the task.

**Question 2**

An ice-cream stall is put up at a school fete. The customers will be served on a First-Come-First served basis. Only 20 customers can be handled at a time. A class has been designed for this purpose.

**Class Name: IceCreamQueue**

**Instance variables :**

int Ordno[ ] : to store order number for the customers
int qty [ ] : to store the ordered quantity
front : to store the first order
rear : to store the last order

**Member Methods:**

1. IceCreamQueue( ) : constructor to create the arrays ordno[ ] and qty[ ], initialize front and rear and both the array elements with 0.
2. int isEmpty ( ) : to check if any customers are out at the stall. It returns 1 for yes and 0 for no
3. int isFull ( ) : to check if any more orders are possible. It returns 1 for yes and 0 for no.
4. void add (int ic, int qt ) : to add the next order no, qty to the list
5. void remove ( ) : to remove the current order details from the list and print the order details along with the bill to be paid upon the quantity where the price of ice-cream is Rs 20/-

Specify the class given above giving details of the constructor **IceCreamQueue(..)** and all other methods . No need to write the main ( ) method.

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

## *6.1 Circular queue*

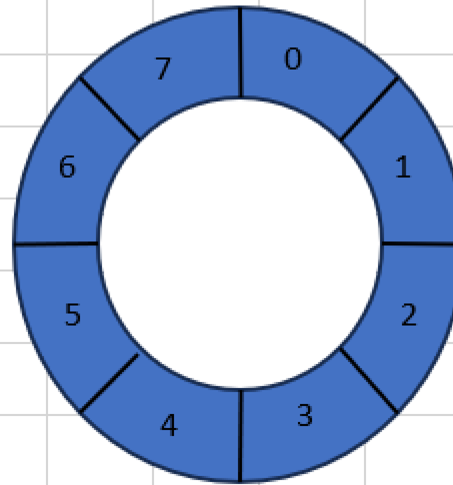
In a circular queue, all the elements are arranged in a circle, instead of a row or line. In the circular queue, a pointer may be used to ensure the maximum size is not exceeded. The pointer is reset to the initial value of 1, as and when the number of elements in the circular queue reaches the optimum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

Spondon Ganguli
An Educator, Artist & Author




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** | | | | | | |

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.

Spondon Ganguli
An Educator, Artist & Author

```
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);
 }
```

```java
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();
}
}
```

Spondon Ganguli
An Educator, Artist & Author

```
/*
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

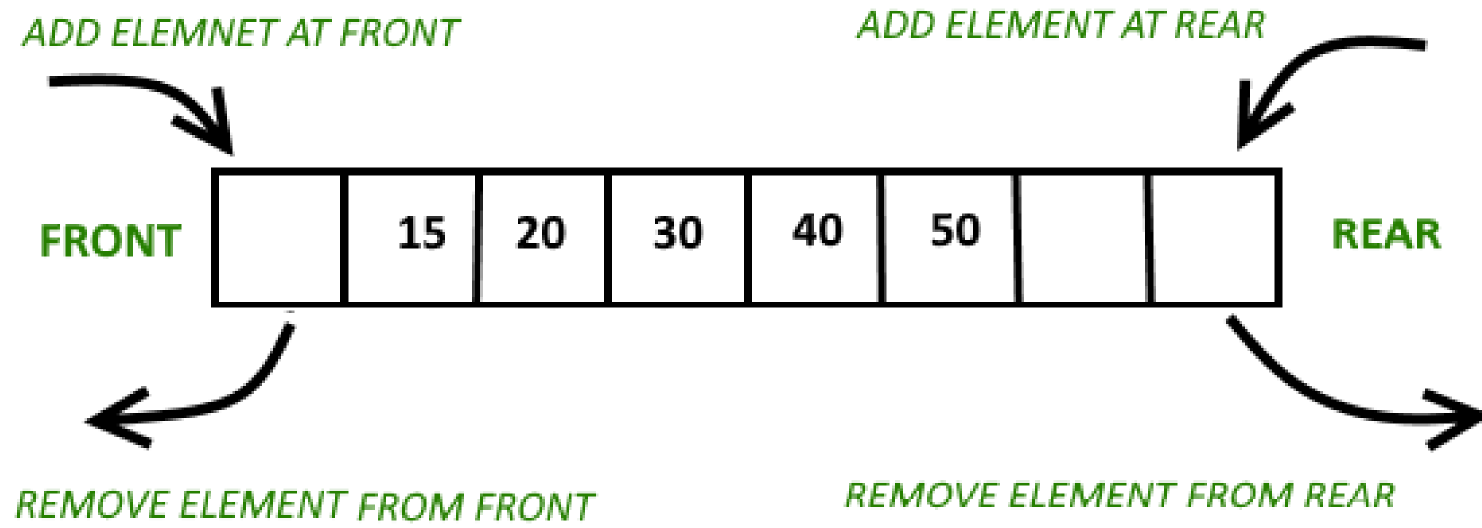
## 6.2 Deque (Double-ended queue)

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.

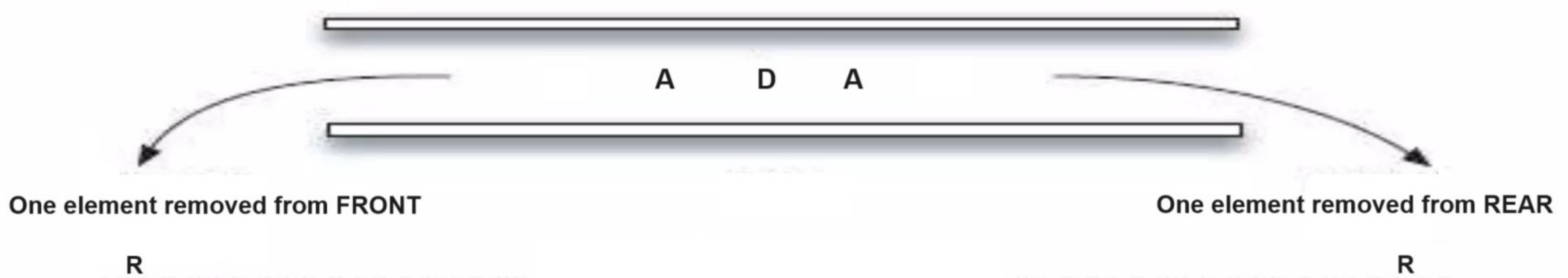
## Output restricted Deque

- 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.

```
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;
    }
```

Spondon Ganguli
An Educator, Artist & Author

```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

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( ).