

Data Structure

It is basically a way of organizing data in the memory. Data structure is a combination of two words, data and structure i.e., we have to store data in the form of structure in main memory or secondary memory.

In order to represent a storage of data in memory an appropriate model is needed, the different models used to organize data and operations that are defined in the main memory is referred to as data structure.

Data structure is a representation of logical relationship existing among individual elements of data.

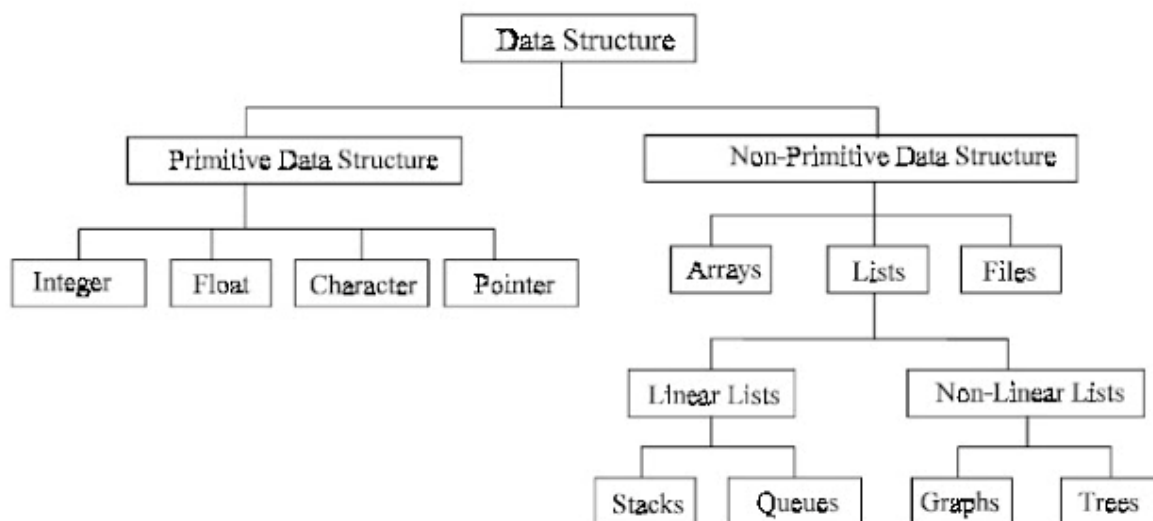
In other words, data structure is a way of organizing all data items that considers not only the element storage but also their relationship to each other. We can also define data structure as a mathematical or logical model of particular organization of data items. Data structure mainly specifies following three things:

- Organization of data
- Accessing method means how data elements are stored.
- Degree of associativity (shows relationship among data elements)

Data Structure = Organisation of Data in memory
+
Applied Operations on it

CLASSIFICATION OF DATA STRUCTURE

Following diagram gives a brief idea how data structure is categorized:



Primitive data structure and Non-Primitive data structure

The data structures that are atomic or indivisible are known as primitive data structure. These are also basic and inbuilt data types. For example- Integer, Float, Char. The data structures are not atomic or not indivisible are known as non- primitive data structure. These are also derived and user-defined data types. For example- List, Array, String.

Linear and Non –Linear data structure

In linear data structure, the data items are arranged in a linear sequence. For example, stack and queue. Stack follows LIFO (Last In First Out) order and Queue follows FIFO (First In First Out) order.

In non- linear data structure the data items are not arranged in a linear sequence. For example, tree and graph. Tree represents hierarchical relationship and in graph there may be cycle with no particular order defined among vertices.

Data structure operations

Basic operations performed on data structure are:

- **Traversing** : Accessing each and every record exactly once so that certain items in the record may be processed
- **Searching** : Finding the location of the record with a given key value or finding the locations of all records which satisfy one or more conditions
- **Sorting** : Arranging the data elements either in ascending or descending order
- **Insertion** : Adding a new record to a data structure
- **Deletion** : Removing an existing record from the data structure
- **Merging** : Combining the records of two different files into a single file

We will discuss here only Linear data structure i.e., Stacks and Queues

STACKS

A stack is a linear structure in which items may be added or removed only at one end, or we can say that a stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. This means that, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

This phenomenon is called LIFO i.e., Last In First Out. This means that the last item to be added to a stack is the first item to be removed.

Everyday examples of stack are-
A stack of dishes, a stack of books and stack of folded towels.

Basic operations on Stacks

“PUSH” is the term used to insert an element into a stack.

“POP” is the term used to delete an element from a stack.

Representation of Stacks

Stack can be represented in memory by two ways:

- Representation of Stacks through Array
- Representation of Stacks through Linked List

Here, we will focus only on Array Representation of Stacks.

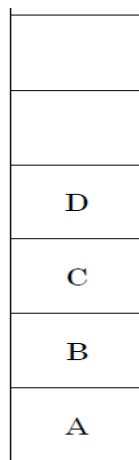
Array Representation of Stacks

Stacks will be maintained by a linear array Stack; a pointer variable Top, which contains the location of the top element of the stack; and a variable MAXSTK or N which gives the maximum number of elements that can be held by the stack.

The condition $\text{Top} = \text{N}$ or $\text{Top} = \text{MAXSTK}$ will indicate that the stack is full.

The condition $\text{Top} = 0$ or $\text{Top} = \text{NULL}$ will indicate that the stack is empty.

In below picture, $\text{N} = 6$
and $\text{Top} = 4$



Here, $\text{Top} = 4$ means the stack has four elements, where N or $\text{MAXSTK} = 6$ means there is room for two more items in the stack.

The operations of adding (pushing) an item onto a stack and the operation of removing (popping) an item from a stack may be implemented, respectively, by the following procedures called PUSH and POP.

For PUSH operation, one must first test whether there is room in the stack for the new item; if not, then this condition is known as overflow.

For POP operation, one must first test whether there is an element in the stack to be deleted; if not, this condition is called underflow.

Algorithm for PUSH operation

- First check is stack full?
- If the stack is full, then overflow condition occurs and terminate the operation.
- If the stack is not full, then we first increment the top pointer by one.
- Insert the elements onto the stack.

Push (Top, N, Stack, Item)

If TOP=N then Print “Overflow” & Exit

Set TOP= TOP +1 // Increment TOP by one

Set Stack [TOP] = Item // Insert the item in new TOP position

Exit

Algorithm for POP operation

- First check is the stack empty?
- If the stack is empty then elements can't be deleted, so underflow condition occur and terminate the operation.
- If the stack is not empty, then delete the top element from the stack
- Decrement top pointer by one.

POP (TOP, N, Stack, Item)

If TOP = NULL then Print “Underflow” & Exit.

Set Item = Stack [TOP] // Assign TOP element to Item

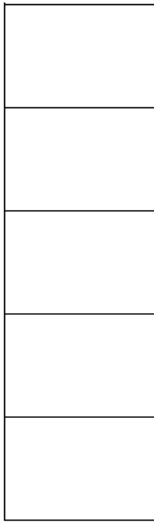
Set TOP = TOP -1 // Decrement TOP by one.

Exit

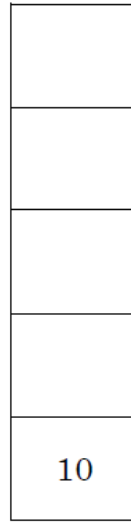
Example of PUSH and POP operations

These five values are taken for this purpose: 10, 20, 30, 40, 50

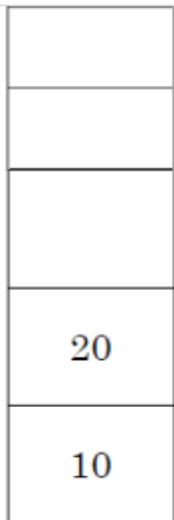
Push Operation



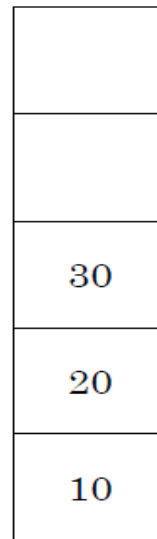
Initially Top = 0



Top = Top + 1
Now Top = 1
Insert 10 at Stack[1]
Stack[1]=10



Top = Top + 1
Now Top = 2
Insert 20 at Stack[2]
Stack[2]=20



Top = Top + 1
Now Top = 3
Insert 30 at Stack[3]
Stack[3]=30

40
30
20
10

Top = Top +1
 Now Top =4
 Insert 40 at Stack[4]
 Stack[4]=40

50
40
30
20
10

Top = Top +1
 Now Top =5
 Insert 50 at Stack[5]
 Stack[5]=50

Now, Top=N=5 means overflow occurs, no more items can be pushed (added) onto stack.

POP Operation

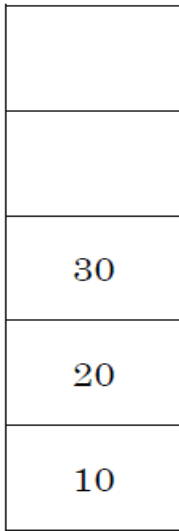
Now, deletion of these five items takes place:

50
40
30
20
10

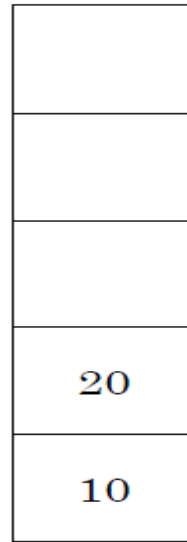
Here, Top=5
50=Stack[5]
Delete 50 from Stack[5]
Now, Top=Top-1
Top=4

40
30
20
10

Here, Top=4
40=Stack[4]
Delete 40 from Stack[4]
Now, Top=Top-1
Top=3



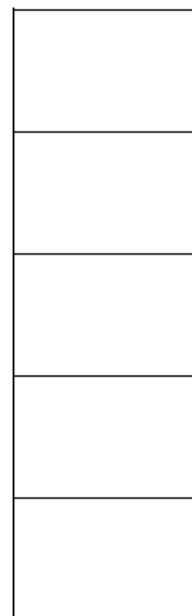
Here, Top=3
30=Stack[3]
Delete 30 from Stack[3]
Now, Top=Top-1
Top=2



Here, Top=2
20=Stack[2]
Delete 20 from Stack[2]
Now, Top=Top-1
Top=1



Here, Top=1
10=Stack[1]
Delete 10 from Stack[1]
Now, Top=Top-1
Top=0



Now, Top=0
Stack is empty
Underflow Occurs

QUEUES

A queue is a linear list of elements in which deletion can take place only at the one end called the front, and insertion can take place only at the other end called the rear end. A queue is a list of elements where deletions are performed from beginning or front of the list while insertions are performed only at the end or rear of the list.

Queue is a linear data structure like stack. Queues are also called First In First Out (FIFO) lists, since the first element enter a queue will be the first element 'out' of the queue.

In other words, the order in which elements enter a queue is the order in which they leave. For example, people waiting in line at a bank form a queue, where the first person in line is the first person to be waited on; and so on.

Representation of Queues

- Array Representation
- Linked List Representation

Here, we will focus only on Array Representation of Queue.

Array Representation of Queues

Queues will be maintained by a linear array QUEUE and two pointer variables: FRONT, containing the location of the front element of the queue; and REAR, containing the location of the rear element of the queue.

The condition $FRONT = NULL$ will indicate that the queue is empty. Whenever an element is deleted from queue, the value of FRONT is incremented by 1; this can be implemented by assignment

$$FRONT = FRONT + 1$$

Likewise, whenever an element is added to queue, the value of REAR is incremented by 1:

$$REAR = REAR + 1$$

Example: We have to insert A, B, C, D, E

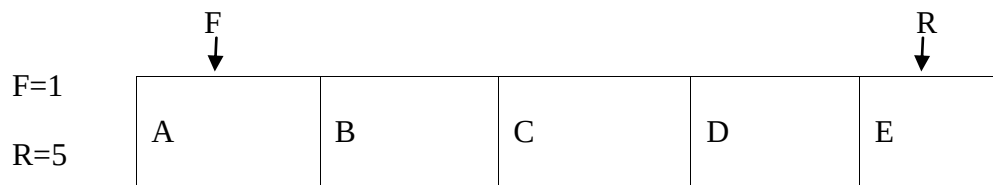
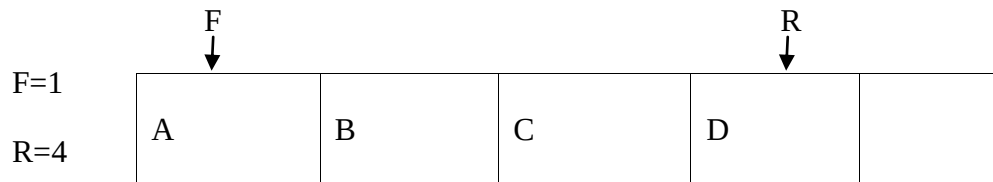
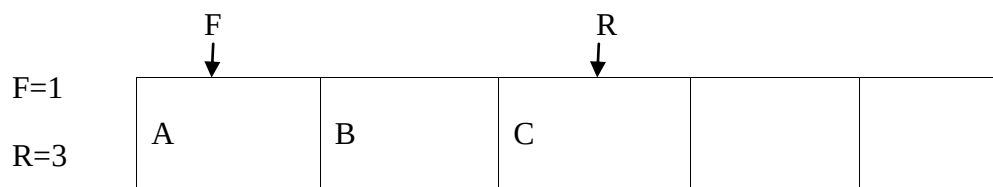
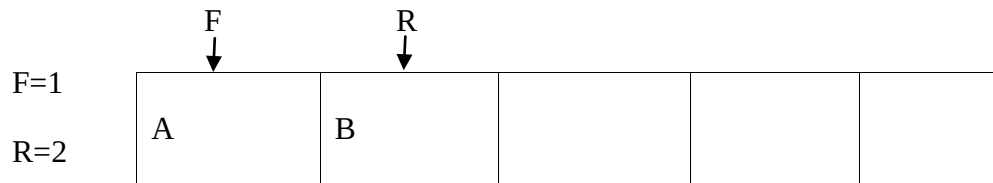
INSERTION

F=0

R= 0

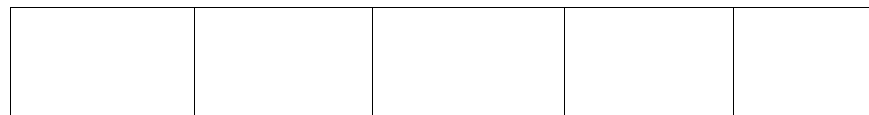
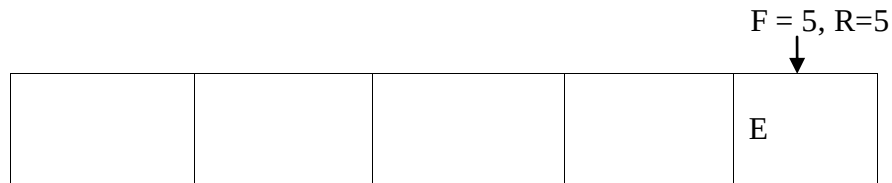
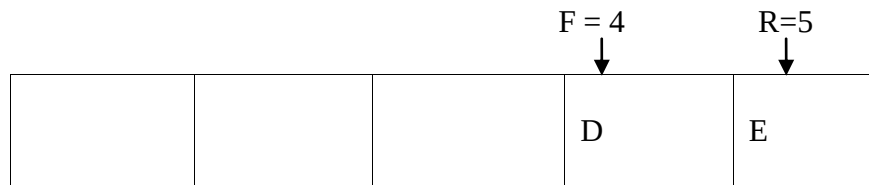
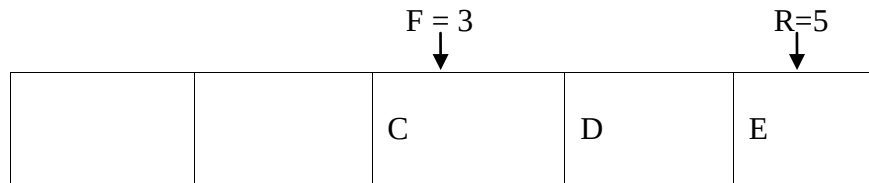
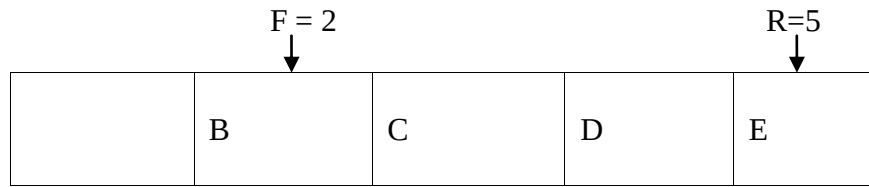
--	--	--	--	--

Here, F stands for FRONT pointer and R stands for REAR pointer



Now, We have to delete A, B, C, D, E

$F=1$ 					$R=5$ 	
A	B	C	D	E		



Now, F = 0
R = 0

This is called Underflow condition means no element is left for deletion.

Algorithm for Insertion Operation

1. First check is queue full?
If queue is full, then print “Overflow” and Exit
2. Now check is queue empty? If queue is empty. Then set Front and Rear is equal to one.
F & R = 1

3. If queue is not full, then set

$\text{Rear} = \text{Rear} + 1$

4. Insert value in Queue at Rear position

QINSERT (QUEUE, N, FRONT, REAR, ITEM)

1. Queue already filled ?

If $\text{Front} = 1$ and $\text{Rear} = N$,

Then print “Overflow” and Exit

2. If $\text{Front} = 0$ $\text{Rear} = 0$

Then

Set $\text{Front} = 1$ and $\text{Rear} = 1$

3. Set $\text{Rear} = \text{Rear} + 1$

4. $\text{QUEUE}[\text{Rear}] = \text{Item}$ // inserts new elements

5. Exit

Algorithm for Deletion Operation

1. First check in queue empty.

If yes print “Underflow” & Exit.

2. If queue is not empty, then delete Front element,

3. If queue is empty, then after deletion of element, set Front and $\text{Rear} = 0$

4. Increment Front by one

QDELETE (QUEUE, N, FRONT, REAR, ITEM)

1. [Queue already empty?]

If $\text{Front} = 0 = \text{NULL}$, then print “Underflow” and Exit

2. Set $\text{Item} = \text{QUEUE}[\text{Front}]$

3. If $\text{Front} = \text{REAR}$, Then Set

$\text{Front} = \text{Rear} = \text{Null}$ or 0

4. Set $\text{Front} = \text{Front} + 1$

5. Exit

Example of Insertion and Deletion Operations

Insert values: 10, 20, 30, 40, 50

1	2	3	4	5

Initially, $F=0$

$R=0$

	1	2	3	4	5
F=1					
R=1	10				

Set R=1 & F=1

Insert 10 at Rear Position

Queue [1] = 10

	1	2	3	4	5
F=1					
R=2	10	20			

Set R= R+1

R=2

Insert 20 at Rear Position

Queue [2] = 20

	1	2	3	4	5
F=1					
R=3	10	20	30		

Set R= R+1

R=3

Insert 30 at Rear Position

Queue [3] = 30

	1	2	3	4	5
F=1					
R=4	10	20	30	40	

Set R= R+1

R=4

Insert 40 at Rear Position

Queue [4] = 40

	1	2	3	4	5
F=1					
R= 5	10	20	30	40	50

Set $R = R + 1$

$R = 5$

Insert 50 at Rear Position

Queue [5] = 50

Delete values: 10, 20, 30, 40, 50

	1	2	3	4	5
F=1					
R=5	10	20	30	40	50

Here, Front=1

$10 = \text{Queue}[1]$

Delete 10 from Queue

Now, $\text{Front} = \text{Front} + 1$

	1	2	3	4	5
F=2					
R=5		20	30	40	50

Here, Front=2

$20 = \text{Queue}[2]$

Delete 20 from Queue

Now, $\text{Front} = \text{Front} + 1$

	1	2	3	4	5
F=3					
R=5			30	40	50

Here, Front=3

$30 = \text{Queue}[3]$

Delete 30 from Queue

Now, $\text{Front} = \text{Front} + 1$

	1	2	3	4	5
F=4				40	50
R=5					

Here, Front=4
 40=Queue[4]
 Delete 40 from Queue
 Now, Front=Front+1

	1	2	3	4	5
F=5					50
R=5					

Here, Front=5
 50=Queue[5]
 Delete 50 from Queue
 Now, Front=Front+1

	1	2	3	4	5
F=0					
R=0					

Set F=R=0

The Queue is empty. That is, all values are deleted now.