Data Structures

1- Introduction

The choice of a suitable data structure can influence the design of an efficient algorithm significantly. In this lecture, we briefly present some of the elementary data structures.

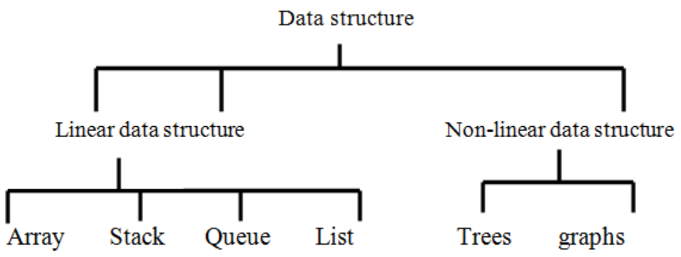
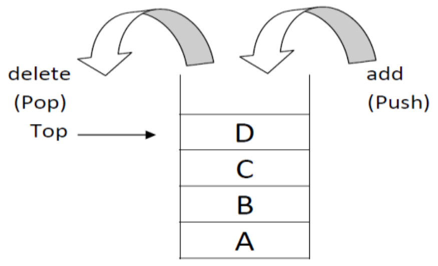
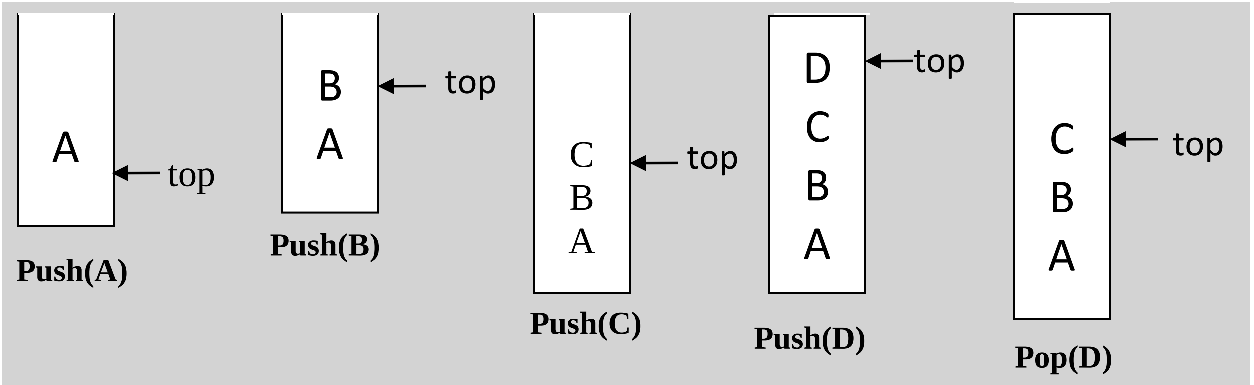


Fig (2) Types of Data Structure

2-Stacks

A stack is a linked list in which insertions and deletions are permitted only at one end, called the top of the stack. It may as well be implemented as an array. This data structure supports two basic operations: pushing an element into the stack and popping an element off the stack. If S is a stack, the operation pop(S) returns the top of the stack and removes it permanently. If x is an element of the same type as the elements in S, then push(S; x) adds x to S and updates the top of the stack so that it points to x.





Figs (2) push and pop operations.

Algorithm 1: Push operation (add element to the stack)
Inputs: stack, Top (index of the stack), size (the maximum size of the stack), n (the element that will added).
Outputs: add n to stack.

```
Begin
        If (stack is full) then
                Write "the stack is overflow"
      Else
           {
                Top=Top+1
                stack[Top]=n
           }
End
```

Algorithm 2: Pop operation (return the top element of the stack)
Inputs: stack, Top (index of the stack).
Outputs: n (the top element of the stack).

```
Begin
        If (stack is empty) then
                Write "the stack is empty"
        Else
        {
                n=stack [Top]
                Top=Top-1
        }
        Return n
End
```

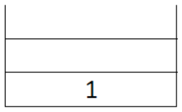
Algorithm 3: print ()

```
  Begin
     If (stack is empty) then
        Print" no element to display"
      Else
          for (i=top ; i >= 0 ; i--)
          Print stack[i]
        End
```

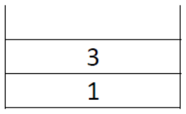
Ex: 1
Let S is (stacking), it's for push item to the stack, and U is (un stacking), it's for pop item from the stack, and the set of inputs to the stack are (M, B, Y, N, R), what are the outputs from these operations:
A) SSUUSUSUSU
B) SSSUSUUSUU
Sol.
The meaning the inputs items to the stack are chose item M first, then item B,...R. we can't pop N before previous items.

A)
Input -----> M B Y N R
Operations -----> S S U U S U S U S U
Output -----> BM Y N R
B)
Input -----> M B Y N R
Operations -----> S S S U S U U S U U
Output -----> Y N B R M

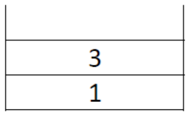
Ex: 2
If the set of inputs of a stack are (1.2.3.4.5) show the right outputs shown below according to the stack operation:
A) 2. 4. 5. 3. 1
B) 4. 5. 1. 2. 3
Sol
A) Outputs (2. 4. 5. 3. 1)
To pop item 2 must at first push two items 1. 2, the operation is SSU.



And to pop item 4 after item 2 must at first push two items 3. 4, the operation is SSUSSU



To pop item 5. After item 4, must push item 5 them pop item 5, the operation is SSUSSUSU



According to the state of a stack we can now pop item 3 then item 1, and the operation is now SSUSSUSUUU.
B) Outputs (4. 5. 1. 2. 3)
We can pop two items 4 & 5 by this sequence of operations
Input -----> 1 2 3 4 5
Operations -----> S S S S U S U
Output -----> 4 5
Now the stack items are: 321 we can't pop item 1 before two items 2 & 3, so this sequence of operations is wrong.
<u>Assignment</u>
Outputs (4. 3. 5. 2. 1) we can get these outputs when use the operations of pop & push by using these sequences: Input -----> 1 2 3 4 5
Operations?

3- Queue:

A queue is a list in which insertions are permitted only at one end of the list called its rear, and all deletions are constrained to the other end called the front of the queue. As in the case of stacks, a queue may also be implemented as an array. The operations supported by queues are the same as those for the stack except that a push operation adds an element at the rear of the queue.

Simple Queue Operations Algorithms:

Algorithm 1: Insert Q

Inputs: Q: array as integer, f: front pointer, r: rear pointer, n: size of queue, x: the new element

Outputs: insert x to Q.

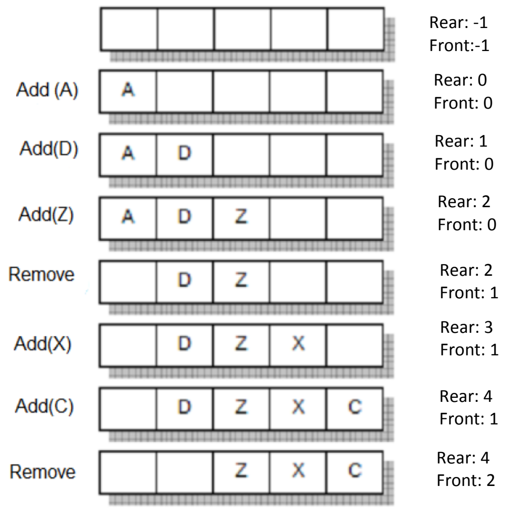
```
Begin
If (r = n-1) then
        Write ("Queue is over flow")
Else
        r = r + 1
        Q[r] =x
        If (f = -1) then
            F=0
End
```

Algorithm 2: Delete Q

Inputs: Q: array as integer, f: front, r: rear, n: queue size

Outputs: x: the deleted element

```
Begin
 If (((f = -1) &&(r=-1)) || (f>r)) then
        Write ("Queue is empty")
Else
        z = Q[f]
        f = f + 1
End
```



Exercise: Suppose Q is a queue, the size of queue is 4. Show Q after all of the following operations has been completed assuming the queue is empty to start with. Show how the front, rear and elements change.

Sol:

State of Queue	F pointer	R Pointer	Q[0]	Q[1]	Q[2]	Q[3]
Empty Q	-1	-1	-	-	-	-
Add item A	0	0	A	-	-	-
Add item B	0	1	A	B	-	-
Add item C	0	2	A	B	C	-
Delete item	1	2	-	B	C	-
Add item E	1	3	-	B	C	E
Delete item	2	3	-	-	C	E

Exercise: Suppose q is a queue, the size of queue is 6. Show Q after all of the following operations has been completed assuming the queue is empty to start with. Show how the front, rear and elements change.
Sol?

State of Queue	F pointer	R Pointer						
Empty Q								
Add item A								
Add item B								
Add item C								
Delete item								
Add item E								
Add item D								
Delete item								
Delete item								