==

# DAY-07

infix to prefix:
infix : a*b/c*d+e*f-g*h --> string

prefix : hg*fe*dcba*/*+- --> empty string

cur ele =

stack:

- infix expression, empty postfix expression and stack which is initially empty

```
    step1: start scanning an infix expression from right to left

    step2:
    if( cur ele is an operand )
        append it into the prefix expression
    else//if cur ele is an operator
    {
        while( !stack is not empty && priority(topmost ele) > priority(cur ele))
        {
            pop ele from the stack and append it into the prefix expression
        }

        push cur ele into the stack
    }

    step3: repeat step1 & step2 till the end of infix expression
    step4: pop all remaining ele's from the stack one by one and append
    them into the prefix expression.
    step5: reverse prefix expression will be the final answer
```

----

infix expression	: a*b/c*d+e*f-g*h
postfix expression	: ab*c/d*ef*+gh*-
prefix expression	: -+*/*abcd*ef*gh

* to convert prefix expression into its equivalent postfix:

- algortithm:

```
step1: start scanning prefix expression from right to left
step2:
    if( cur ele is an operand )
        push it onto the stack
    else//if the cur ele is an operator
    {
        - pop two ele's from the stack
        - form a string by concatenating op1, op2 & opr(cur ele)
            "op1+op2+opr"
        - push result string onto stack
    }

ste3: repeat step1 & step2 till end of the prefix expression
```

prefix expression : - + * / * a b c d * e f *gh
postfix expression: ab*c/d*ef*+gh*-
ab*c/d*ef*+gh*-

+ postfix expression evaluation algorithm:

```
    infix expression    : 4 * 5 / 8 * 6 - 3 + 7
    postfix expreesion  :
```

=> 4 * 5 / 8 * 6 - 3 + 7
=> 45* / 8 * 6 - 3 + 7
=> 45*8/ * 6 - 3 + 7
=> 45*8/6* - 3 + 7
=> 45*8/6*3- + 7
=> 45*8/6*3-7+

postfix expression: 4 5 * 8 / 6 * 3 - 7 +

cur ele : 9 + 7

stack:

16

```
step1: start scanning postfix expression from left to right
step2:
        if( cur ele is an operand )
            push it onto the stack
        else//if cur ele is an operator
        {
            - pop two ele's from the stack
            - first popped ele will be the second operand
            and second popped ele will be the first operand
            op2 = peek(); pop();
            op1 = peek(); pop();

            - perform cur ele operation between op1 & op2
            res = op1 opr op2
            - push result back onto the stack
        }
```

step3: repeat step2 & step3 till the end of postfix expression
step4: pop the final result from the stack and return it to the calling function

----
+ "Queue": it is a collection/list of logically related similar type of elements
in which element can be added from one end referred as "rear" end, whereas element can
be deleted from another end referred as "front" end.
- in this list, element which was inserted first can be deleted first, so this list works in "first in first out" manner, and hence this list is also called as FIFO list/
"last in last out" i.e. LILO list.
- basically we can perform two operations onto queue data structure in O(1) time:
1. enqueue: to insert/push/add an ele into the queue from rear end
2. dequeue: to delete/remove/pop an ele from the queue which is at front end

- there are different types of queue:
1. "linear queue"
2. "circular queue"

3. "priority queue": it is a type of queue in which element can added randomly (i.e. without checking its priority), whereas element which is having highest priority can only be deleted first.
- priority queue can be implemented by using a linked list, whereas it can implemented efficiently by using binary heap.

4. "double ended queue (deque)": it is a type of queue in which elements can be added as well deleted from both the ends.

    - we can perform four basic operations on deque in O(1):
    1. push_front -> add_first()
    2. push_back -> add_last()
    3. pop_front -> delete_first()
    4. pop_back -> delete_last()

- deque can be implemented by using doubly circular linked list

+ implementation of a linear queue by using an array : static queue
+ implementation of a linear queue by using a linked list : dyanmic queue

+ implementation of a linear queue by using an array : static queue

```
    struct queue
    {
        int arr[5];
        int rear;
        int front;
    }queue_t;
    arr          : int []
    rear    : int
    front   : int


    is_queue_full  : rear == SIZE-1
    is_queue_empty      : rear == -1 || front > rear
```

```
1. enqueue: to insert/push/add an ele into the queue from rear end:
    step1: check is queue not full
    step2: increment the value of rear by 1
    step3: insert/push an ele into the queue from rear end
    step4: if( front == -1 )
              front = 0
```

```
2. dequeue: to delete/remove/pop an ele from the queue which is at front end:
    step1: check is queue not empty
    step2: increment the value of front by 1 (i.e. we are achieving deletion of an ele
    from the queue).
```

+ "limitation of linear queue':
    - vacant places cannot be reutlized in a linear queue

rear = 4 & front = 0
rear = 0 & front = 1
rear = 1 & front = 2
rear = 2 & front = 3
rear = 3 & front = 4

- if front is at next position of rear we can say queue is full:
    queue_full: front == (rear + 1) % SIZE

```
for rear = 0, front = 1
front == (rear + 1) % SIZE
1      == (0+1) % 5
1      == 1 % 5
1      == 1
```

```
for rear = 1, front = 2
front == (rear + 1) % SIZE
2      == (1+1) % 5
```

```
2          == 2 % 5
2          == 2



for rear = 2, front = 3
front == (rear + 1) % SIZE
3          == (2+1) % 5
3          == 3 % 5
3          == 3



for rear = 3, front = 4
front == (rear + 1) % SIZE
4          == (3+1) % 5
4          == 4 % 5
4          == 4

for rear = 4, front = 0
front == (rear + 1) % SIZE
0          == (4+1) % 5
0          == 5 % 5
0          == 0




front++;
=> front = front + 1

for incrementing value of front by 1
front = (front+1)%SIZE

if front = 0 => front = (front+1)%SIZE => (0+1)%5 => 1%5 => 1
if front = 1 => front = (front+1)%SIZE => (1+1)%5 => 2%5 => 2
if front = 2 => front = (front+1)%SIZE => (2+1)%5 => 3%5 => 3
if front = 3 => front = (front+1)%SIZE => (3+1)%5 => 4%5 => 4
if front = 4 => front = (front+1)%SIZE => (4+1)%5 => 5%5 => 0
```