Expression evaluation

Expression conversion

✓ $a+b*c-d*f$

$= ((a+(b*c))-(d*f))$

$= ((a+(bc*))-(df*))$

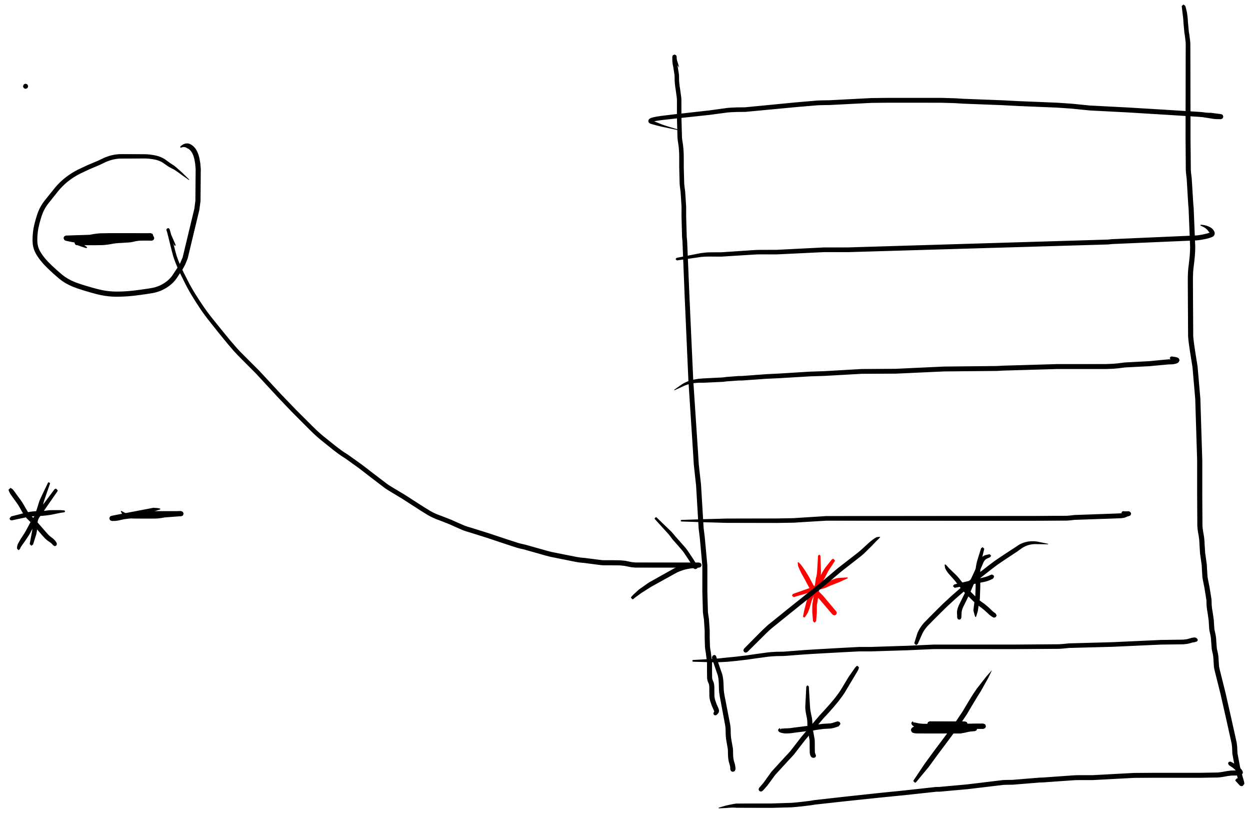
$= ((abc*+)-(df*))$

✓ $= abc*+df*-$ $\Longleftarrow$

Observation:- The order of the operands are invariant.

$E = a + b * c - d * f$

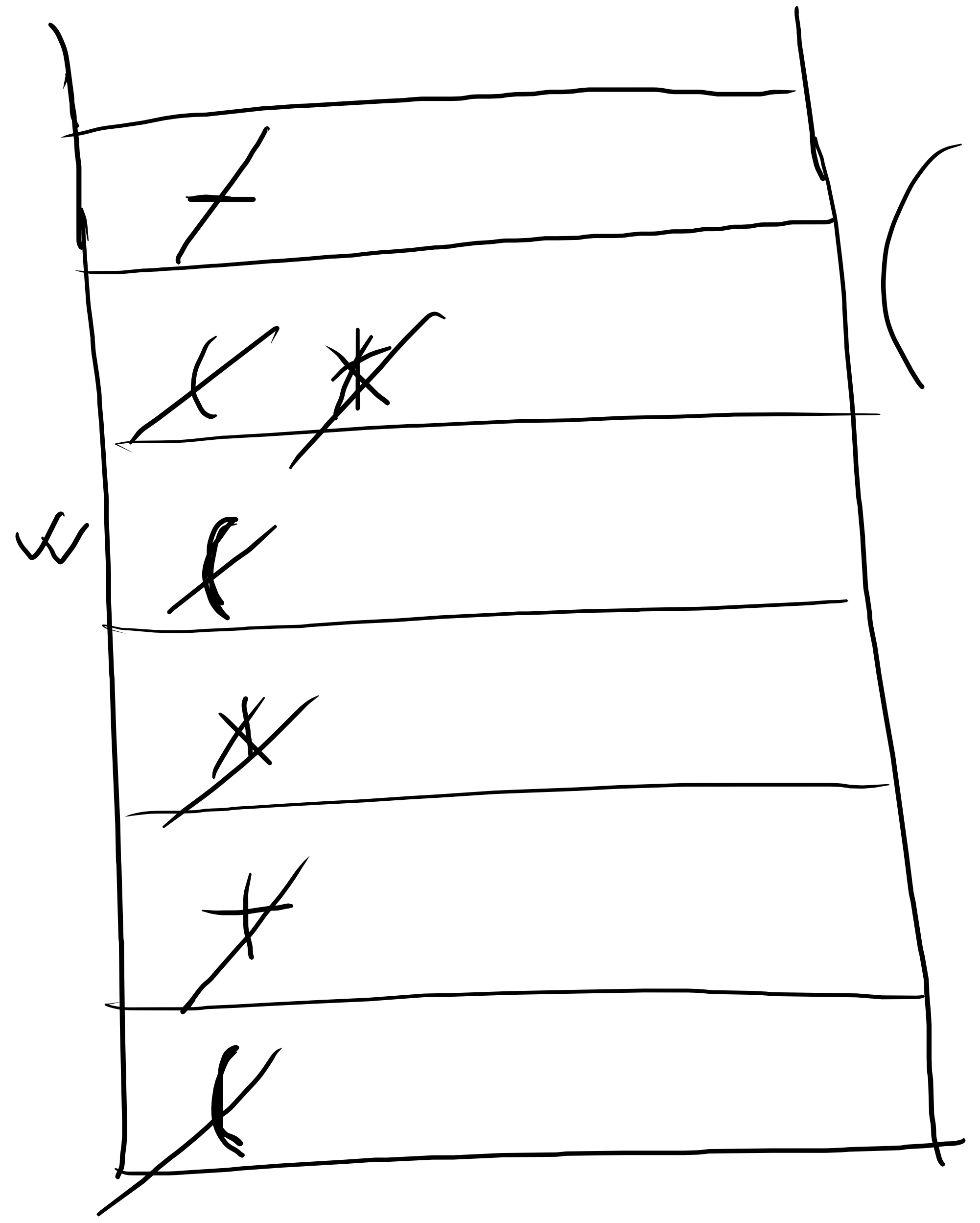
$F = a\,b\,c\,*\,+\,d\,f\,*\,-$



```
Infix_postfix ( E )
 create a stack S
  n = leng (E)
  for i = 0 to n-1
        if E(i) is an operand
              F = F + E(i)
        if E(i) is an operator
              while S is not empty and higherprecedence(S.top(), E(i))
                    F = F + S.top()
                    S.pop()
              S.push(E(i))
 while S is not empty
       F = F + S.~~top~~()
       S.pop()
```

$F = (a + b) * ((c - d) * f)$

( ) (( ))



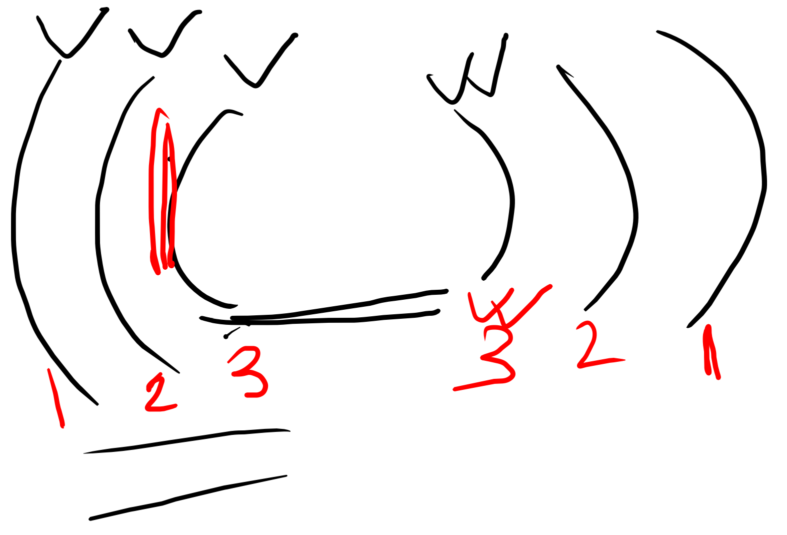
Pseudocode

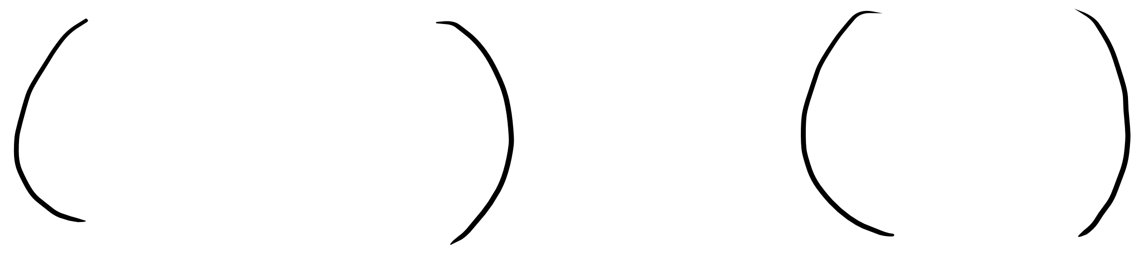
H.W.

F = ab+cd-f**

Infix to prefix

H.W.

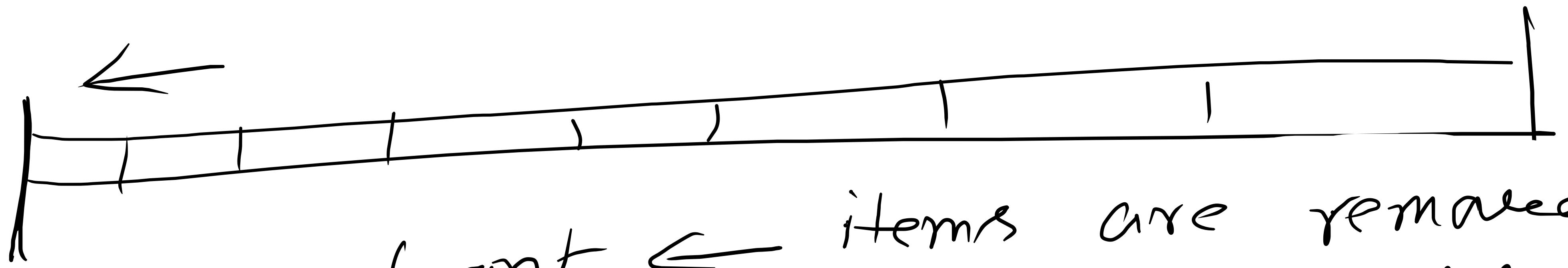




Queue

It is a data structure with the property that insertion can be performed at one end and deletion can be performed at the other end.

call : FIFO property
First in First out.



- parameters. front ← items are removed
rear ← items are added.

Operations

Static

- front
- Isfull
- Is empty

Dynamic

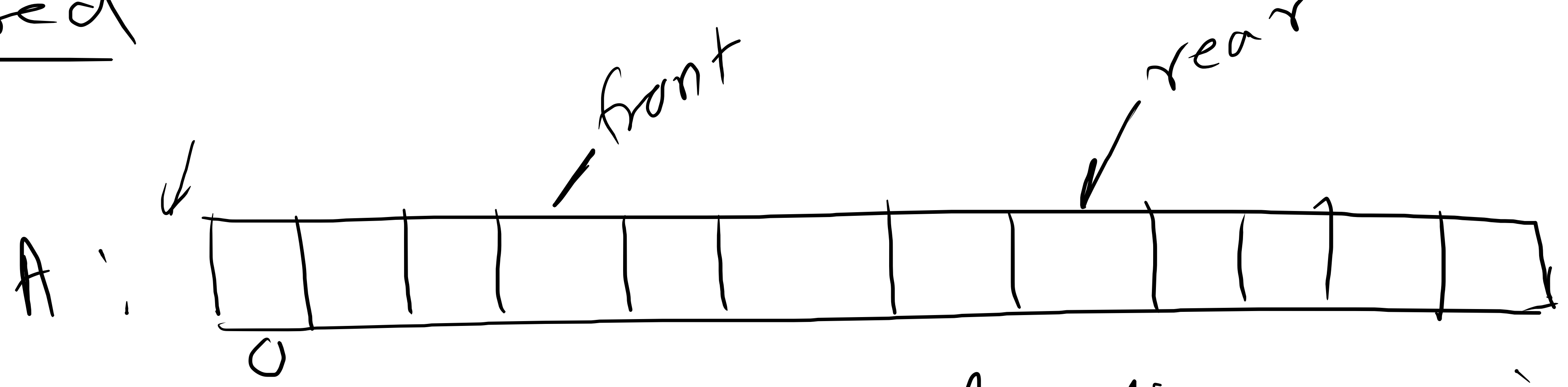
Enqueue ← Insertion

Dequeue ← deletion.

Queue implementation

- array based
- Linked-list based.

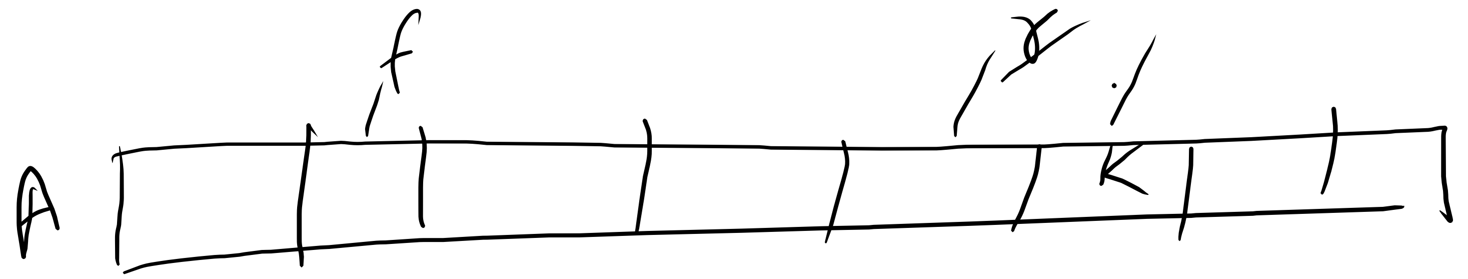
Array based



The queue elements under consideration are in front - rear.

```
front (A)
    return front.

Isempty (A)
    if front == -1 and rear == -1
        return true
    else
        return false.

Isfull (A)
if rear == size(A) - 1
        return true
  else
      return false.
```

```
Enqueue (A, K)

  if Isfull (A)
        enqueue not possible

  If Isempty (A)
        rear = 0, front = 0
        A [rear] = K .

  else
        rear = rear + 1
        A [rear] = K .
```



```
Dequeue(A)
If isempty(A)
        ~~D~~dequeue not possible
  if front = rear
        front = -1
        rear = -1
  else
        front = front + 1
```