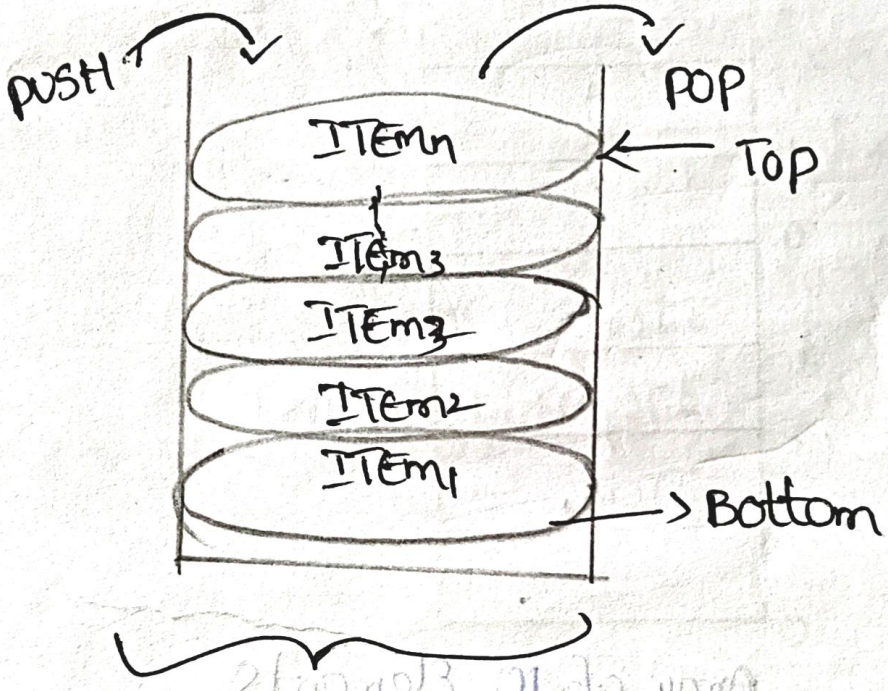
STACKS :—



STACK Representation

→ A STACK is a collection of homogenous data Element where Insertion (PUSH) and deletion (pop) takes place only at one End called TOP.

→ An Element in a STACK is termed as "ITEM".

→ The maximum number of Elements that a STACK can hold is called "SIZE" of the STACK.

Ex:- (1) Goods in a cargo

(2) plates on a Tray

(3) Train in a railway yard.

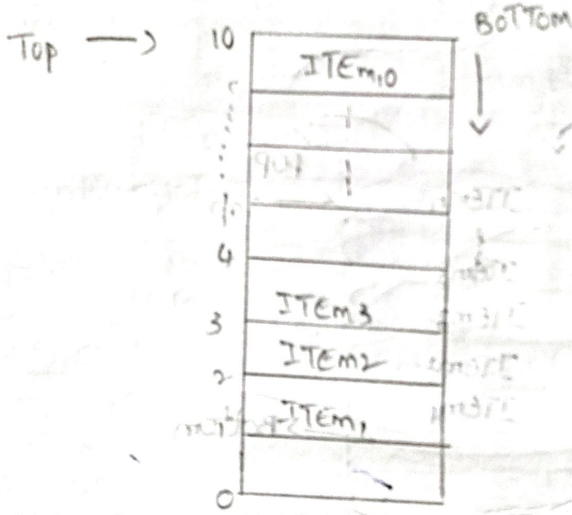
Representation of STACK :-

→ Representation of stack is memory is done using 2 ways.

→ i) one-directional Array

ii) using linked list

i) STACK representation using Arrays:



Array of 10 Elements

u = upper BOUND

L = lower BOUND

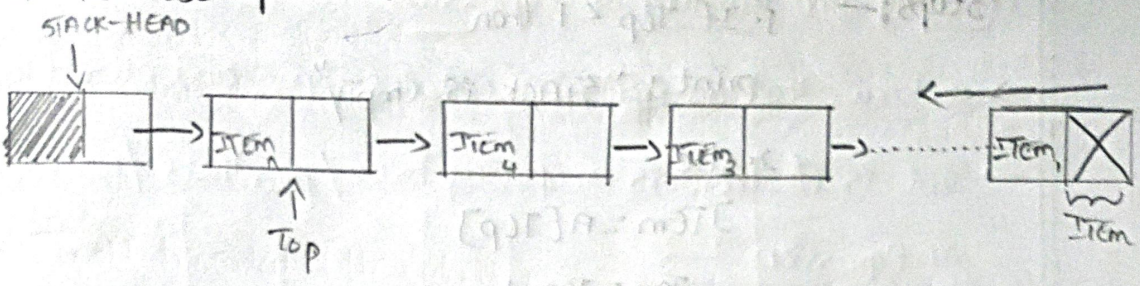
Size = u + L - 1

→ Although Array representation is Easy because of its fixed size.

→ But some applications need more size to represent.

→ To avoid representing of fixed size we relay on linked list representation.

ii) Linked list representation of STACK:



→ Here data field is for Item and link field points to next ITEM.

operations on stack using Arrays

i) PUSH

ii) pop

i) PUSH – A (ITEM) → (Array)

Input: new ITEM to be inserted.

output: A STACK with newly pushed item at Top position.

data structure: An array 'A' with Top as pointer.

steps:

1. If Top = size then
   point "STACK is full"
2. Else
   Top = Top+1
   A[Top] = ITEM
3. end if
4. STop

ii) pop – A [ITEM] → Array

Input :– A stack with some Elements

output :– Remove an ITEM from Top of the STACK if it is not Empty.

data structure: An array 'A' with Top as pointer.

steps:— 1. If Top < 1 then

print : "STACK is Empty"

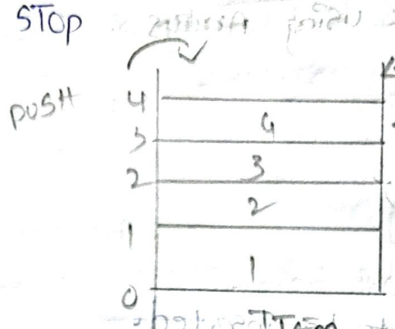
2. else

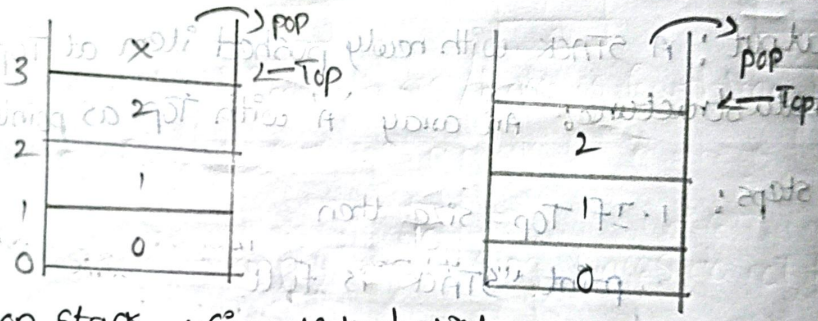
ITEm = A[TOP]

Top = Top - 1

3. End if

4. STOP





operations on stack using linked list.

i) push

ii) pop

→ linked list

i) Algorithm PUSH - L [ITEm)

Input : Item to be inserted into the list

output : Inserted ITEm into the list at Top position.

datastructure :

A SLL whose pointer towards Header node is known.

STaps :-

1) new = GETNODE (NODE)

2) new.DATA = ITEm

3) new.LINK = Top

4. Top = new

5. STACK - HEADER . LINK = Top

6. STOP

pop-steps :-

1) If Top = null then

i) Print "STACK is empty"

ii) exit

2. else

i) ptr = Top . LINK

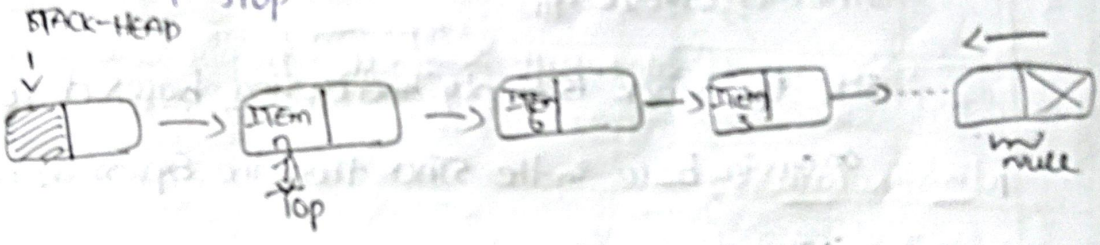
ii) ITem = Top . DATA

iii) STACK - HEADER = ptr

iv) Top = ptr

3. End if

4. STOP



Applications of STACKS :-

1) Evalution of expression
   - Infix form
   - prefix
   - post fix form

2) Syntax check the validity of given expression

3) decimal to binary conversion

4) Parsing of well formed parenthesis.

5) Reversing of a given string

2) checking the validity of Expression :

→ we know that an Expression contains open and closed braces

→ That is if there is an opening brace / parenthsis / brackets in an Expression

Then there must be a closing brace / parenthesis / brackets.

→ If any of this rule is not matched while validation then we can say that the given Expression is Invalid.

Steps :-

1) when a parenthesis is Encountered then it is pushed onto the STACK.

2) when Ever a closing parenthesis / brace is found then the STACK is Examined.

3) If the STACK is Empty and closing braces do not have an opening brace in the STACK then the Expression is Invalid.

4) when STACK is not Empty then we pop the STACK and check whether the ITEMS coresponds to closing Braces or not.

5) If match found, we continue otherwise the Expression is Invalid.

6) when End of Expression is reached the STACK must be Empty.

Ex :- validate the following Expression and verify it is valid or not?

$$[(A+B)-\{C+D\}] - [F+G]$$

	Symbol Scanned	STACK
1	[	[
2	(	[, (
3	A	[, (
4	+	[, (
5	B	[, (
6	)	[
7	-	[

	Symbol Scanned	STACK
8	{	[, {
9	C	[, {
10	+	[, {
11	D	[, {
12	}	[
13	]	
14	-	
15	[	[
16	F	[
17	+	[
18	G	[
19	]	Empty

∴ The given Expression is a valid Expression.

(8) Decimal to Binary conversion :—

Steps :— 1. Accept the number 'N' in decimal form

2. Divide the number 'N' by 2.

3. Store the quotient in 'N' and remainder in 'R'.

4. Now push the value of 'R' into the STACK.

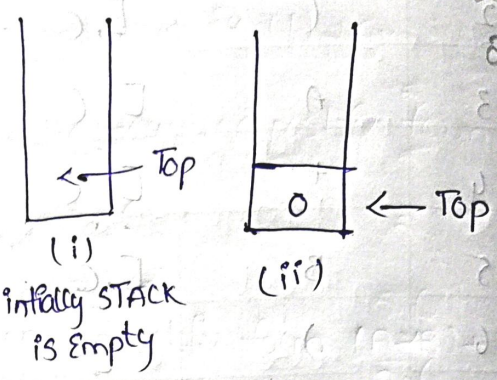
5. Check whether the value of 'N' is not zero. If "yes" goto step 2.

6. If the value of N is zero then retrive (pop) the values from STACK one by one until STACK is Empty.

7. STOP

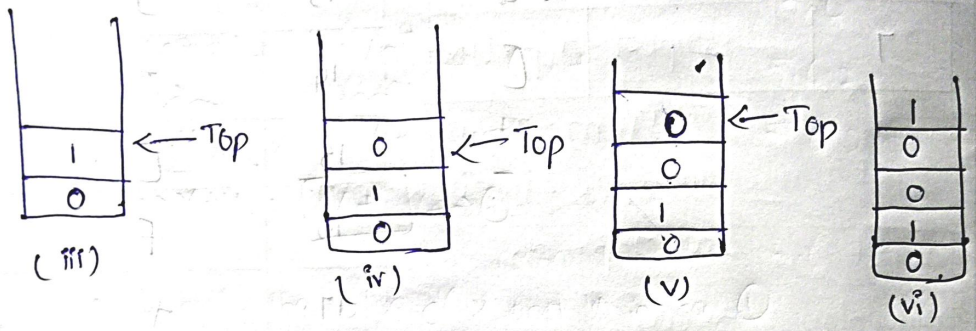
Ex :- convert 18 into binary using STACKS.

2	18
2	9 0
2	4 1
2	2 0
2	1 0



(i) initially STACK is Empty

(ii)



(iii) (iv) (v) (vi)

Since the value of 'N' is zero we pop all the numbers from STACK The final Result is : 10010 } Binary value for decimal number 18.

polish notations :-

The order in which the expression has to be Evaluated is completly determined by the position of operators and operands.

→ polish notation Explains about the order in which the Expression has to be arranged.

→ There are 3 types of notations used to represent an expression.

(i) Infix

(ii) prefix

(iii) postfix

Infix → (Operand1, operator, operand2)

prefix →

Operator, operand1, operand2

postfix :-

Operand1, operand2, operator

Example :-

Infix	prefix	postfix
a*b	*ab	ab*
a+b*c	+a*bc	abc*+
(a+b)*c	*+abc	ab+c*

Also known as (Reverse polish notation)

Note :- A computer evaluates the arithmatic expression written in infix form in 2 steps.

i) convert it into postfix notation

ii) Then Evaluate the postfix Expression using order of priority of operators, called "BODMAS"

Highest priority of operators :-

Exponential (^/↑)

multiplication (*)

division (÷ (or) /) } have same order of priority

lowest priority of operators :-

Addition (+) or

Subtraction (−)

1) convert the following Expression to postfix form

(A+B) / (C−D) } Infix form

↓ ↓

(AB+) / (CD−)

x y

put X = (AB+)

Y = (CD−)

⇒ X/Y ⇒ XY/ — ①

Now put the value of X and Y in Eq (1)

AB+CD−/ is the required postfix form for (A+B)/(C−D) Expression.

2) convert the following Expression to postfix form (A*B)+(C/D)

(A*B) + (C/D)

X Y

⇒ X+Y = XY+ → ①

Now put the value of X and Y in Eq ①

~~((A*B+C/D))~~ AB*CD/+ is required form.

H.W 3) (A+(B*C−(D/E^F)*G)*H) convert the above Expression to postfix form.

Sol: (A+B*C−(D/E^F)*G)*H)

⇒ (A+(B*C−(D/E F^)*G)*H)

⇒ (A+(B*C−(D/X)*G)*H)

⇒ (A+(B*C−(DX/)*G)*H)

⇒ (A+(B*C−(Y*G))*H)

⇒ (A+(B*C−(YG*))*H)

A+(B*C−T)*H

A+((BC*)−T)*H

U

A+(U-T)*H

A+(UT-)*H

(U → W)

A+(W*H)

A+(WH*)

(WH* → Z)

A+Z

AZ+ —→ (1)

Now resubstitute the values in the given Eq (1) we get final postfix form as

ABC+DEF^/G*-H*+

Infix to postfix conversion using STACKS :-

steps :- 1. Scan the expression from left to right we shall get a symbol which may be an operator (or) a paranthesis (or) an operand.

2. The symbol is treated as follows.

i) if the symbol is an operand add it to postfix notation.

ii) If the symbol is an opening paranthesis then push it on to the STACK.

iii) If the symbol is an operator then check the Top of the STACK.

3. If the precedance of the operator at the Top of the STACK is higher or same as current operator then repeatedly pop and add it to postfix notation, otherwise it is PUSHED onto STACK.

4. If the symbol is a closing paranthesis the repeatedly pop all the operator and add to Each operator to postfix expression until corresponding opening paranthesis is Encountered/found.

5. finally remove opening paranthesis from STACK.

1) Ex:- convert 5*(6+2)-(12/4) into postfix form.

Sol:-

S.NO	Scanned symbol	STACK	postfix Expression
1.	5		5
2.	*	*	5
3.	(	*, (	5
4.	6	*, (	5, 6
5.	+	*, (, +	5, 6
6.	2	*, (, +	5, 6, 2
7.	)	*	5 6 2 +
8.	-	-	
		(since * has high presandance it is POPED	5, 6, 2, +, *
9.	(	-, (	5, 6, 2, +, *
10.	12	-, (	5, 6, 2, +, *, 12
11.	/	-, (	5, 6, 2, +, *, 12
12.	4	-, (	5, 6, 2, +, *, 12, 4
13.	)	-	5, 6, 2, +, *, 12, 4, /
14.	[Empty symbol] no symbol to scan it is Empty	pop symbol from STACK and add to postfix Expression	5, 6, 2, +, *, 12, 4, /, - it is the final postfix form Expression.

H.W 2) a+b*c+(d*e+f)*g convert the above Infix Expression to postfix Expression.

A) a+b*c+(d*e+f)*g

S.NO	Scanned Symbol	STACK	postfix Expression
1.	a		a
2.	+	+	a
3.	b	+	a,b
4.	*	+,*	a,b
5.	c	+,*	a,b,c
6.	+	+,*,+	a,b,c
7.	(	+,*,+,(	a,b,c
8.	d	+,*,+,(	a,b,c,d
9.	*	+,*,+,(,*	a,b,c,d
10.	e	+,*,+,(,*	a,b,c,d,e
11.	+	+,*,+,(,*,+	a,b,c,d,e
12.	f	+,*,+,(,*,+	a,b,c,d,e,f
13.	)	+,*,+	a,b,c,d,e,f,+,*
14.	*	+,*,+,*	a,b,c,d,e,f,+,*
15.	g	+,*,+,*	a,b,c,d,e,f,+,*,g
16.	no symbol to scan It is Empty	Pop symbol from stack and add to postfix Expression	a,b,c,d,e,f,+,*,g, *,+,*,+ is final post fix Expression.

NOTE :-

i) If the scanned operator is greater than the operator in stack. Then the second operator is also pushed in the stack.

ii) If the scanned operator is less than or Equal to the operator in the stack then the stack operator must be popped out.

Evaluation of Expression : when the given Expression is in postfix form)

i) Scan the postfix Expression from left to right.

ii) If the scanned characters is an operand then it is PUSHED into STACK.

iii) If the scanned characters is an operator then that will be operated on Top two Elements on STACK and result is stored / pushed into operand STACK.

iv) Repeat step ① to ③ until Expression is Empty.

Ex:- Evaluate the following postfix Expression 5,6,2,+,*,12,4,/,-

S.NO	Scanned Symbol	STACK	operation (B op A)
1	5	5	
2	6	5,6	
3	2	5,6,2	
4	+	5,6+2=8 5,8	B A ↑ ↑ => B+A 6+2 = [8] -> pushed onto STACK
5	*	5*8 = 40	B * A => B*A ↑ ↑ 5 * 8 = [40] -> pushed onto STACK
6	12	40,12	
7	4	40,12,4	
8	/	40,12/4 40,3	B/A = 6/4 - pushed onto STACK
9	-	B A ↑ ↑ 40 - 3 => [37]	B-A = 40-3 = [37] pushed onto STACK

∴ The result is 37

Ex2:- Evaluate the following postfix Expression

5,9,8,+,4,6,*,+,7,-,*

S.NO	scanned symbol	STACK	operation (B op A)
1	5	5	
2	9	5,9	
3	8	5,9,8	
4	+	5,9+8=17 (5,11)	B A ↑ ↑ 9+8=[17] → pushed onto STACK
5	4	5,17,4	
6	6	5,17,4,6	B A ↑ ↑ 4*6=24 => pushed onto STACK
7	*	5,17,4*6 → 5,17,24	
8	+	5,17,+24,5,41	B A ↑ ↑ => B+A 17+24=41 pushed onto STACK
9	7	5,41,7	
10	−	5,41−7 5,34	
11	*	5*34	5*34=170 → pushed onto STACK

∴ The result is 170.

Ex-3:- Evaluate the following postfix Expression using STACK.

7,2,−,1,3,+,/

S.NO	scanned symbol	STACK	Operation (B op A)
1	7	7	
2	2	7,2	B A ↑ ↑ 7−2=[5] → pushed onto STACK
3	−	7−2=5	
4	1	5,1	B A ↑ ↑ 1+3=[4] → pushed onto STACK
5	3	5,1,3	
6	+	5,1+3 5,4	
7	/	5/4=1.25	5/4=[1.25] pushed onto STACK

∴ The result is 1.25

Ex:- convert the following infix Expression to postfix Expression using STACK Method. 2 * 3 / (2-1) + 5 * (4-1)

S.NO	scanned symbol	stack	postfix Expression
1	2		2
2	*	*	2
3	3	*	23
4	/	/	23*
5	(	/(	23*
6	2	/(	23*2
7	-	/(-	23*2
8	1	/(-	23*21
9	)	/	23*21-
10	+	+	23*21-/
11	5	+	23*21-/5
12	*	+*	23*21-/5
13	(	+*(	23*21-/5
14	4	+*(	23*21-/54
15	-	+*(-	23*21-54
16	1	+*(-	23*21-54/

Conversion of infix Expression to postfix Expression.

Ex1:- A/B^C-D

Given Expression is A/B^C-D

=> A/(B^C)-D

=> A/(ABC)-D

=> (A/X)-D

=> (/AX)-D

=> Y-D

=> -YD

=> -(A*D)

=> -/A^BCD

convert postfix Expression for the following infix expression.

A-(B/C)*(D*E-f)

(A-/BC)*(D*E-f)

=> (-/ABC)*(D*E-f)

x

=> x*(*DE-f)

=> x*(*-DEf)

y

=> x*y

=> *xy

=> *(-/ABC)(*-DEf)

=> *-/ABC*-DEf

=> *-/*-ABCDEf

convert the following postfix expression to prefix Expression

ABCDE/*-f/G++

ABC/DE*-f/G++

AB*C/DE-f/G++

A-B*C/DEf/G++

A/-B*C/DEfG++

A+/-B*C/DEfG+

+A+/-B*C/DEfG is required prefix Expression.

convert the following postfix Expression to infix Expression

ABCDE/*-f/G++

Sol: ABCDE/*-f/G++

Consider the Expression

ABCDE/*-f/G++

Scan from left to right

ABCDE/*-f/G++

ABC(D/E)*-f/G++

AB(C*(D/E))-f/G++

A(B-(C*(D/E))f/G++

A(B-(C*(D/E))/fG++

A(B-(C*(D/E))f+G+

A+(B-(C*(D/E)))/f)+G) is the required infix Expression.

convert the following postfix expression to prefix Expression

infix Expression 234^^93/+43+2+5----

infix :- Given postfix Expression

a [34^] ^93/+43*2+5---

2[(3^4)] ^93/+43*2+5--

[(2^(3^4))] 93/+43*2+5--

[2^(3^4))] [(9/3)]+43*2+5--

[(2^(3^4))]+[(9/3)]43*2+5--

[(2^(3^4))]+[(9/3)][(4*3)]2+5--

((2^(3^4))+(9/3))((4*3)+2)5--

((2^(3^4))+(9/3))((4*3)+2)-5)

((2^(3^4))+(9/3))-(((4*3)+2)-5) is required infix Expression.

prefix Expression :-

234^^93/+43*2+5--

=> 2 34^ 93/43*2+5--

2(^34)^ 93/+43*2+5--

^2(^34)) 93/ +43*2+5--

(^2(^34)193+ 43*2+5--

(+^2(^34)(93) 43* 2+5--

(+^2(^34)193) (*43)2+5--

(+^2(^34)/93)(*43)(+25)--

(+^2(^34)/93) (+*4325) --

(+^2(^34)/93)(-+*4325)--

--+^2^34/93-+*4325 is the final prefix Expression

convert to Infix and prefix form abcdg/*++

prefix form:—

Given postfix Expression abcdg/*++

abcdg/*++

=> abc/dg*++

=> ab*c/dg++

=> *abc/dg++

=> * a+bc/dg+

=> + *a+bc/dg

=> + * + abc/dg

∴ + * + abc/dg is the required Expression.

Infix form :—

Given postfix expression

abcdg /* + +

=> abcd/g * + +

=> abc * d/g + +

=> a + b c * d/g +

=> a + b + c * d/g

It is required infix expression.

Algorithm for

Infix to prefix conversion using STACK:

step1 : Read the Given input string from right to left.

step 2 : Examine the next element in the input.

step 3 : If it is an operand add it to output string.

step 4 : If Element is a closing paranthesis PUSH it on to STACK.

step 5 : If Element is an operator then:

i) if STACK is Empty PUSH operator on to STACK.

ii) if Top of the STACK is a closing paranthesis then PUSH it onto STACK.

iii) If Element has same (or) highest priority than Top of STACK then PUSH operator on to STACK.

iv) Else pop the operator from STACK and add it to output string repeat step 5.

step 6: If Element is an opening paranthesis, pop operators from STACK and add it to output string.

step 7: If there are more inputs goto step 2.

step 8: If there are no more inputs, STACK the operations and add them to output string.

step 9: After completion now reverse the output string.

Example :- convert $2*3/(2-1)+5*(4-1)$ into prefix form.

Sol :- 1. Reverse the given string / Expression

$$)1-4(*5+)1-2(/3*2$$

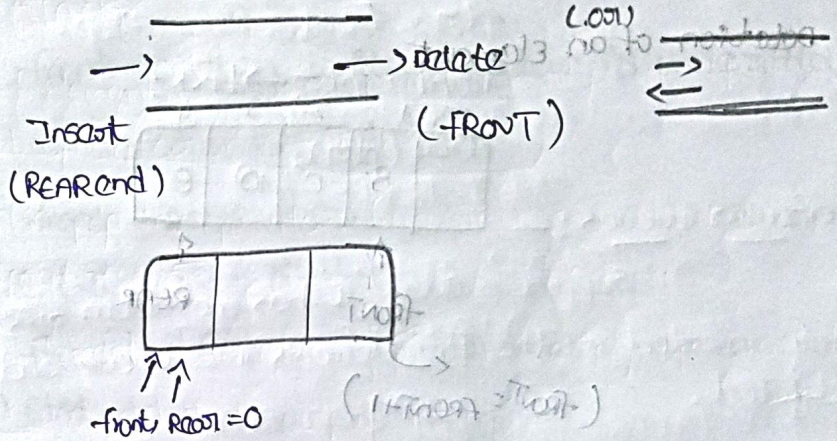
S.NO	character Scanned (Right to left)	STACK content	prefix Expression
1	)	)	
2	1	)	1
3	−	), −	1
4	4	), −	1,4
5	(	Empty	1,4,−
6	*	*	1,4,−
7	5	*	1,4,−,5
8	+	+	1,4,−,5,*
9	)	+,)	1,4,−,5,*
10	1	+,)	1,6,−,5,*,1
11	−	+,), −	1,4,−,5,*,1,
12	2	+,), −	1,4,−,5,*,1,2
13	(	+	1,4,−,5,*,1,2,−
14	/	+/	1,4,−,5,*,1,2,−
15	3	+/	1,4,−,5,*,1,2,−,3
16	*	+/*	1,4,−,5,*,1,2,3
17	2	+/*	1,4,−,5,*,1,2,3,2
18	NO content to SCAN	Empty STACK	1,4,−,5,*,1,2,3,2,*,/,+ → output string
19	NOW Reverse the output string to get prefix Expression.		

Ans:- +, /, *, 2, 3, 2, /, *, 5, -, 4, / is the required prefix Expression.

Queues :-

A Queue is a linear data structure in which insertion of an Element takes place from one End called as "REAR", and delation of Element takes place from another End called as "FRONT".

→ The Element that Enters into Queue is deleted first. so queues is also known as first-in first-out (FIFO).

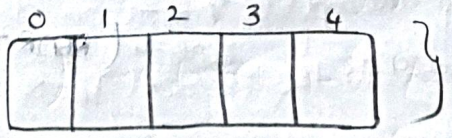


Representation of Queues :-

1) using Arrays

2) using Linked list

1) using Arrays :

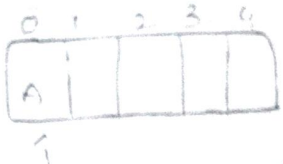


A queue with 5 Elements in an Array

FRONT = -1
REAR = -1

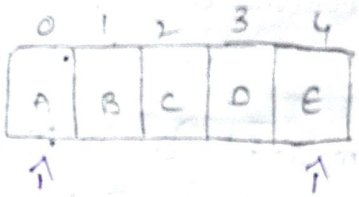
when queue is Empty

(i) suppose we Insert an Element 'A' in queue then



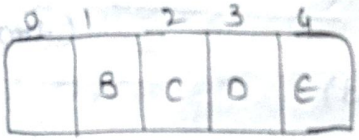
FRONT = REAR = 0

(ii)



FRONT = 0 REAR = 4

Deletion of an element



FRONT REAR

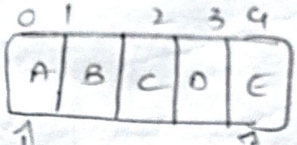
(FRONT = FRONT + 1)

REAR > max size

print "queue overflow

Insertion not possible

when we insert last element then.



front = 0 REAR = 4 (REAR = REAR + 1)

Algorithm for Q-insertion

1. If REAR = MAX then "print queue overflow

Insertion not possible" and STOP

2. Else Read DATA

3. If FRONT = 0 then

(a) front ← front+1

(b) REAR ← REAR+1

(c) Q(REAR) ← DATA

Else

REAR ← REAR+1

Q(REAR) ← DATA

5. STOP

Algorithm for Q-Deletion :-

1. If front = 0 then write "Queue is underflow" and STOP

2. If front = REAR then

i) Q(front) ← NULL

ii) front ← 0

iii) Else

iv) Q(front) ← NULL

v) front ← front+1

3. STOP

Algorithm to insert an element in a queue using linked list

ptr → info = item

ptr → link = NULL,

if (front = NULL) // If Queue is Empty

front = ptr;

Else

rear → LINK = ptr;

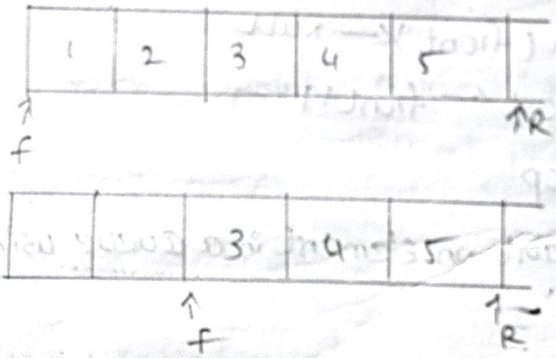
rear = ptr;

Algorithm for deletion in linked list

```
if (front == NULL) then
print "Queue is underflow";
Return;
if (front == REAR) then
    free (FRONT);
    rear = NULL;
    Else
    ptr = front;
    front = ptr -> LINK;
    free (ptr);
```

Circular queues :—

Suppose a queue 'Q' has maximum size as 5 and we have inserted all the five elements into queue and two elements are deleted.



Now if we attempt to insert an element, eventhough there is a provision to insert the element we cannot perform the insertion because insertion is always performed from REAR End. from our diagram the rear pointer is located at the End of the queue (reached the maximum size of array) so it is not possible to increment pointer again.

* the list will always contain 'full'.

To overcome this limitation circular queue was introduced.

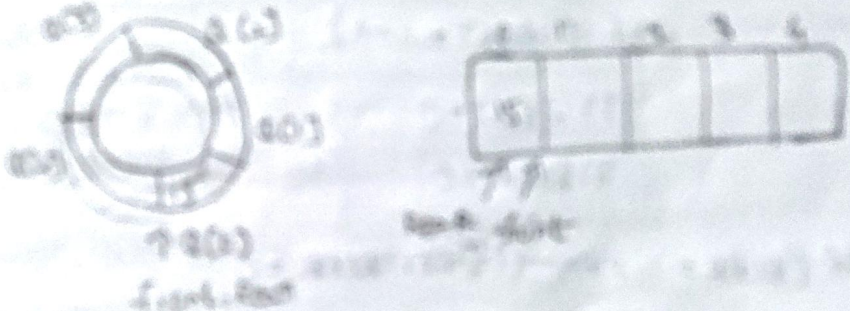
* At any time the position of element to be inserted in circular queue can be calculated as Rear = (Rear+1) % size.

* When deleting an element in circular queue the position of front is calculated as front = (front+1) % size.

* To pictorial representation of circular queue and its implementation:

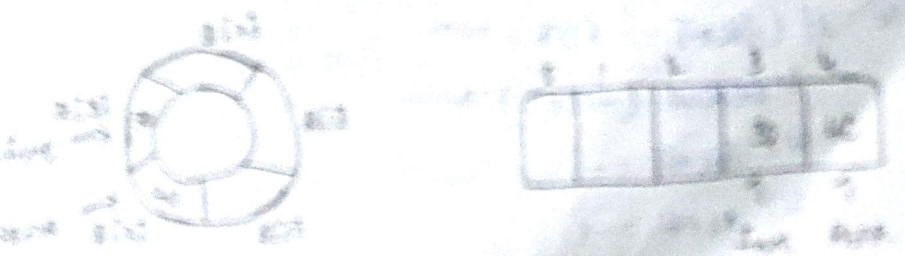
Array of size '5'

(1)



(2) After inserting 10, 20, 30 and 40 in queue.

After deleting 5, 10, 20 from queue.



so the list will always show as "full".

To overcome this limitation Circular Queues was introduced.

note :-

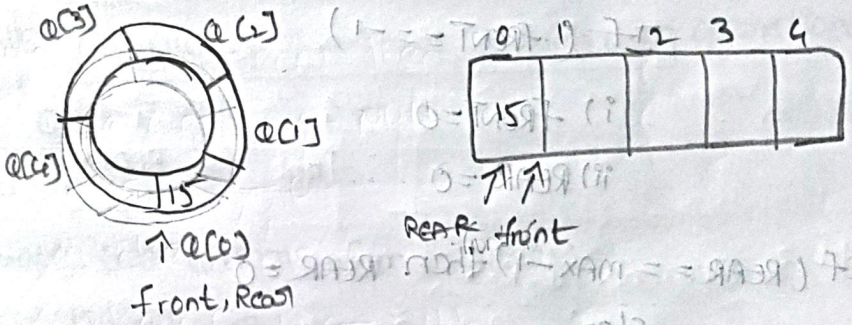
1) At any time the position of Element to be inserted in circular queue can be calculated as REAR = (REAR+1) % size.

2) After deleting an Element in circular queue the position of front is calculated as front = (front+1) % size.

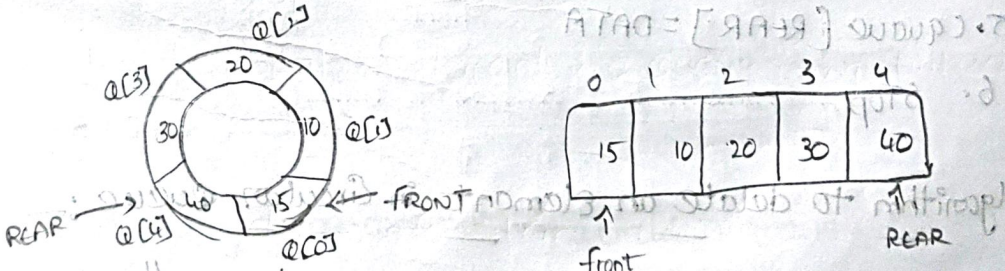
Ex :- The pictorial representation of circular queue and its arrays representation :

Array of size 5

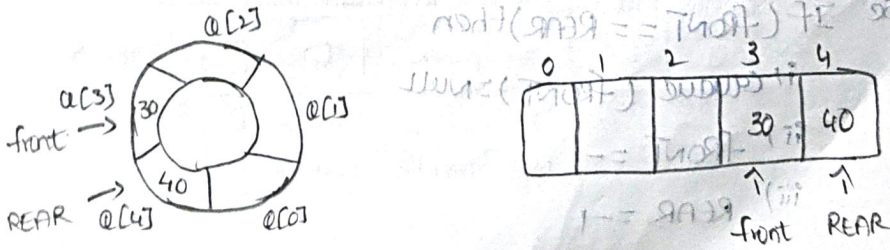
(i)



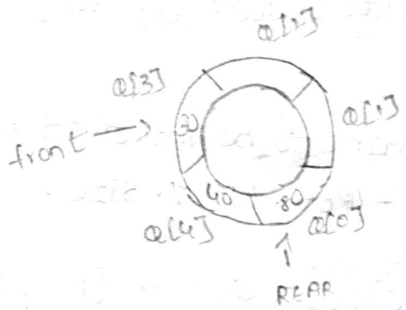
(ii) After Inserting 10, 20, 30 and 40 in queue.

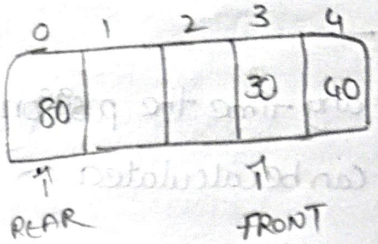


iii) After deleting 15, 10, 20 from queue.



iv) After inserting 80 into circular queue.





Algorithm to insert an element in circular queue :-

1. If (FRONT == 0 && REAR == max-1) || (FRONT = REAR+1)
   then "Queue is overflow" STOP.

2. Else Read data to Insert

3. If (FRONT == -1)
   i) FRONT = 0
   ii) REAR = 0

4. If (REAR == MAX-1) then REAR = 0

5. Else
   REAR = REAR+1

5. Cqueue [REAR] = DATA

6. STop

Algorithm to delete an element in circular queue :-

1. If (FRONT == -1) then write "Queue underflow" and STOP.

2. Else If (FRONT == REAR) then
   i) Cqueue (FRONT) = NULL
   ii) FRONT = -1
   iii) REAR = -1

3. If (FRONT = = max-1) then

i) cqueue (FRONT = NULL)

ii) FRONT = 0

Else

i) cqueue (FRONT = NULL)

ii) FRONT = FRONT + 1

4. STOP

Consider a circular queue which consists of maximum 6 Elements

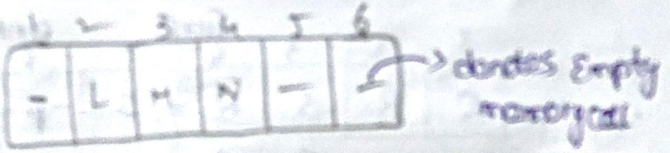
Condition-1

REAR = 4

front = 2

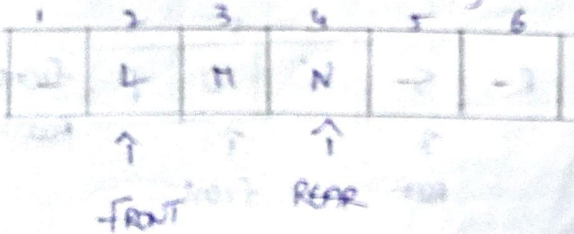
describe the circular queue for the following operations.

1. Add 'O'
2. Add 'P'
3. delete two letters from the queue
4. Add Q, R, S
5. delete one letter from queue

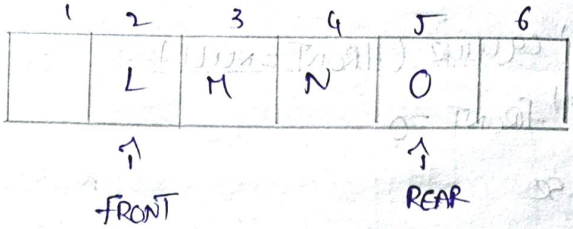


Give data regarding constructing a circular queue:

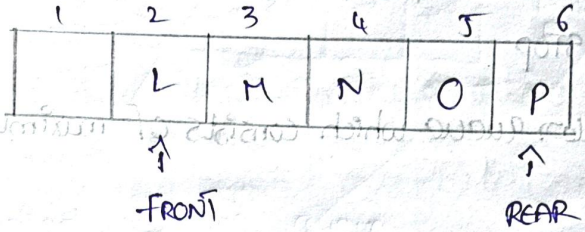
REAR = 4 FRONT = 2 Array consisting of 6 Elements



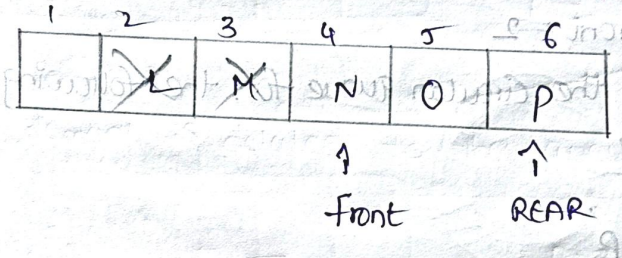
i) Add 'O' to the queue.



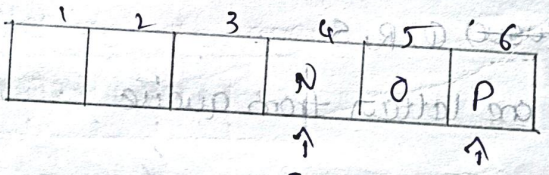
ii) Add 'P' to the queue.

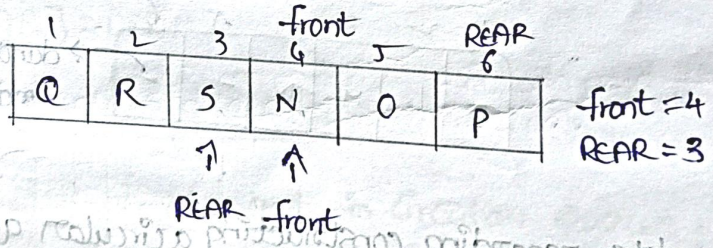


iii) Deleting two Elements from Queue.

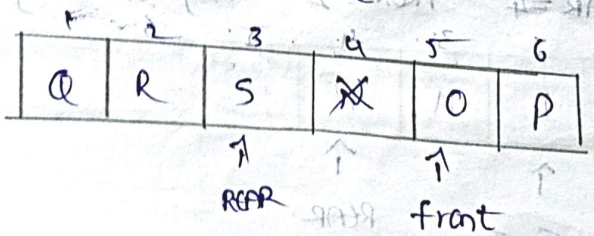


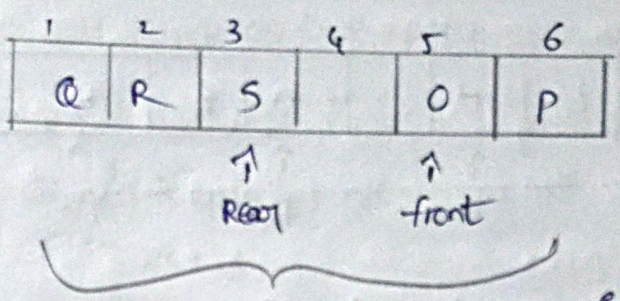
iv) Add 3 Elements Q, R, S to queue.





v) Delete one Element from Queue.





Is the final array after performing the given condition on circular queue.

Example 2:- consider a circular queue with 6 elements where front = 2 , REAR = 4 Queue :- - A C D - -

Perform the following operations on the queue.

① F is added to the queue

② two elements are deleted from queue.

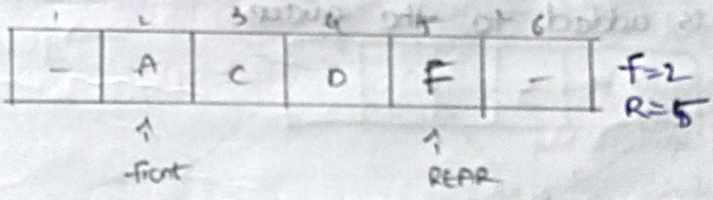
③ K, L, and M elements are added to the queue.

④ Two elements deleted again.

⑤ S is added to the queue, and one element is deleted.

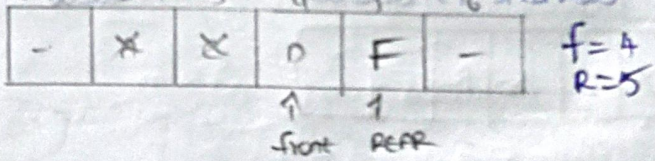
⑥ R is added and one element is deleted from queue.

(i)



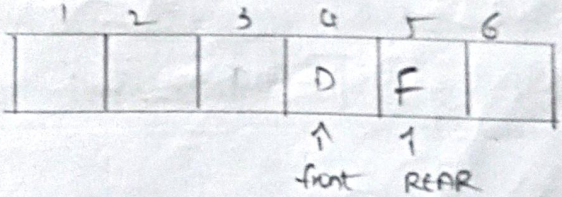
two elements are deleted from queue.

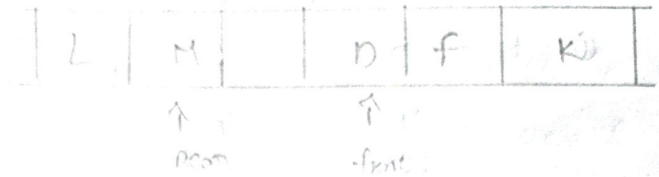
(ii)



K, L, and M three elements are added to queue.

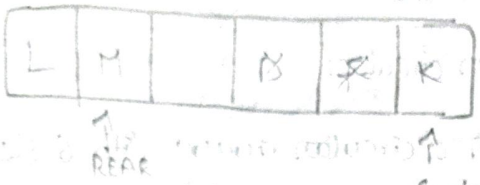
(iii)





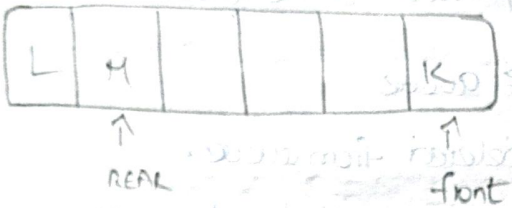
f=4
R=2

iii) Two Elements are deleted again.



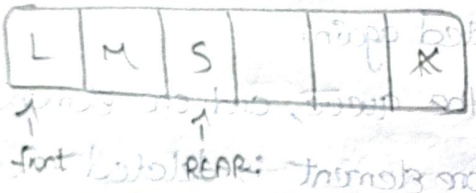
f=6
R=2

v) 'S' is added to the queue



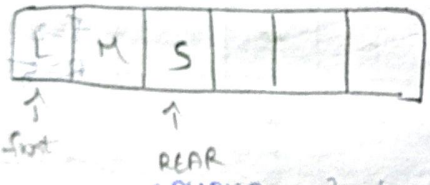
f=6
R=2

one Element is deleted to the queue.



f=1
R=3

vi) 'R' is added to the queue



one Element is deleted to the queue.