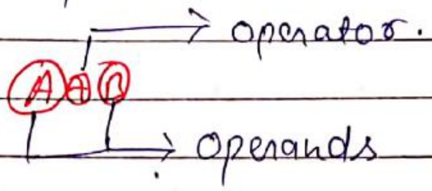
* operator :- Operator are the special kind of Symbols (+, -, *, /) that are used to perform any specific task like mathematical or Logical.

e.g. +, -, *, sizeof, &&, ||

* Operands :- Operands are Expression or values on which operator works or operates.

e.g.



* Arithmatic Expression :- An Arithmetic Expression Consist of operators and operands where operands may be either Numeric Constant or Numeric variable.

e.g. 9+6/3

In this Example if division perform before addition then the result will be 11, while if addition performe before division then result would be 5,

To Avoid this Ambiguity (puzzle), we can use paranthesis to specify which operation will perform first.

e.g. (9+6)/3 or 9+(6/3)

→ Each operator is given some priority, And the operator which have highest priority will be peform first.

→ Anything that is between the parentheses (bracket) will be Evaluated first.

→ precedence of operators.

^ (Exponentiation) → Higher priority

*(multiplication) and /(division) → Next High priority

+ and - → Lowest priority.

Another solution to solve this 9+6/3 expression is with priority or precendence of operation.
In this expression the highest priority operator is /(divide), so we perform division first, Now, After division only one operator left that is +(Addition) so, we Add rest expression.

e.g. 9+6/3
⇒ 9+2
⇒ 11

e.g. 9+6/3*2-5
⇒ 9+2*2-5
⇒ 9+4-5
⇒ 13-5
⇒ 8

→ If two operators have Same priority (like * and /) then the Expression is scanned from left to Right and whichever operator Comes first will be Evaluated first. This is ~~also~~ called Associativity.

This Is How We Can Evaluate Arithmetic Expression manually. We have to Know the preecedence rules and take care of the parathesies. And Associativity Also.

→ Now, Let use see How computer can Evaluate these Expreaions. If we use the same precedence then there will be the repeated scanning from left to Right which is inefficient. It would be Nice if we could transform the Above Expressions in some forms which does Not have paratheses and all operators are Arranged in proper oreder. Acc. to their precedence.

→ The polish Notations are used for this purpose.

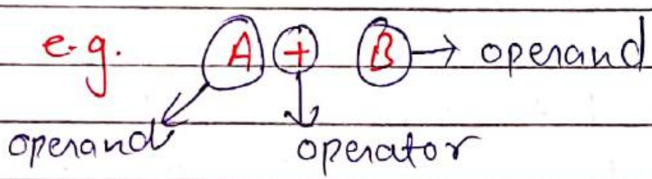
The method of writing operators of An Expression ~~or~~ Either before their operands Or After their operands is called polish Notation.

→ **Ways of writing :-**

(i) Infix Notation
(ii) pre-fix Notation also called polish Notation
(iii) post-fix Notation also called Reverse polish Notation

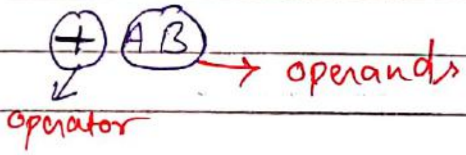
(i) Wen the operators Exist between two operands then the Expression is called Infix Notation

e.g.



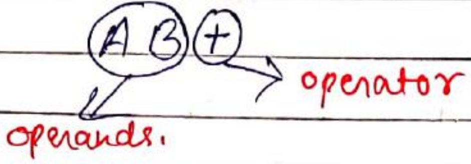
(ii) When the Operators are written before their operands then the expression is called the prefix Notation.

e.g.



(iii) When the operators are written After their operands then the expression is called post fix Notation.

e.g.



→ Conversion of Infix to prefix and postfix Notation :-

e.g. (A+B)*C

⇒ (+AB)*C → considered As 1 operand

⇒ *+AB → prefix

(A+B)*C

AB+ *C

AB+C* → postfix

e.g. A+(B*C)

⇒ A+*BC

⇒ +A*BC → prefix

A+(B*C)

A+BC*

ABC*+ → postfix

e.g. (A+B)/(C-D)

⇒ +AB / -CD

⇒ /+AB-CD → prefix

(A+B)/(C-D)

⇒ AB+ / CD-

⇒ AB+CD-/ - postfix

e.g 10+2-8+3

⇒ +10 2 - +8 3

⇒ - +10 2 + 8 3 - prefix

10+2-8+3

⇒ 10 2+ - 8 3+

⇒ 10 2+ 8 3+ -

* Conversion of postfix to Infix.

E.g. AB+

for conversion of postfix to Infix we read the Eqn from Left to Right

then Infix of AB+ will be
A+B

E.g. $\rightarrow$ ABC*+
A (B*C) +
A+ B*C

E.g. AB+CD−/
(A+B), (C−D), /
⇒ (A+B) / (C−D)

E.g. 10, 2, +, 8, 3, +, −

⇒ (10+2), 8, 3, +, −

⇒ (10+2), (8+3), −

⇒ (10+2) − (8+3)

* Conversion of prefix to Infix

→ for conversion of prefix to Infix we read the given eqn from Righ to Left.

e.g. + A B ←

⇒ A+B → Infix

e.g. + A * B C

⇒ + A, $\underline{B*C}$

⇒ A + B*C → Infix

e.g. / + AB − CD ←

⇒ /, +, A,B, C−D

⇒ /, A+B, C−D

⇒ (A+B)/(C−D) → Infix.

E.g. −, +, 10, 2, +, 8, 3 ←

⇒ −, +, 10, 2, 8+3

⇒ −, ~~+~~ 10+2, 8+3

⇒ (10+2) − (8+3)

Exam.

DATE: / /

PAGE:

Q Convert A+(B*C-(D/E^F)*G)*H into postfix notation

A+(B*C-(D/E^F)*G)*H

⇒ A+(B*C-(D/EF^)*G)*H

⇒ A+(B*C-DEF^/*G)*H

⇒ A+(BC*-DEF^/*G)*H

⇒ A+(BC*-DEF^/G*)*H

⇒ ~~A+(BC*-DEF^/G*-)*H~~

⇒ ~~A+ BC-DEF^/G*-*H~~

⇒ ~~A+ BC-DEF^/G-H*~~

⇒ ~~ABC-DEF^/G-H*+~~

~~ABC-DEF^/G-H*H~~

A+(BC*DEF^G*-)*H

A+ BC*DEF^G*-H*

ABC*DEF^G*-H*+

An.

Q. Convert A+(B*C-(D/E^F)*G)*H into prefix.

A+(B*C-(D/E^F)*G)*H

⇒ A+(B*C-(D/^EF)*G)*H

⇒ A+(B*C-/D^EF)*G)*H

⇒ A+(*BC-*/D^EFG)*H

⇒ A+-*BC*/D^EFG*H

Scanned by CamScanner

A + * - * BC * / D ^ E F G H

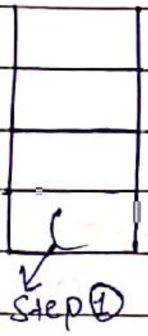
=> + A * - * BC * / D ^ E F G H

* Conversion of Infix Expr. into postfix Exp. using STACK.

Infix Exp. = A+B) → Step ①   postfix Exp = AB+ → Step ⑥

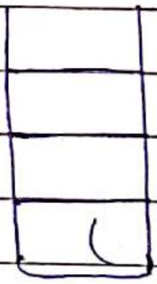
Step ② Step ④

Step ①



(i) push open parentheses in stack and put close parantheses to Infix Eqn

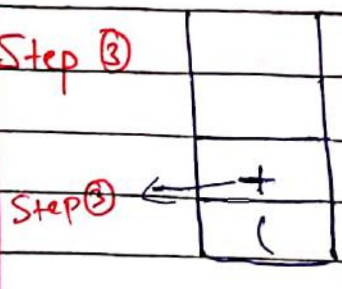
Step ②



(i) Start Reading the Infix Eqn from Left to Right

(ii) If there is any operand came then push it into postfix expression

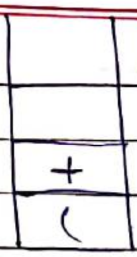
Step ③



(i) Now Next one is + operator so push it into stack.

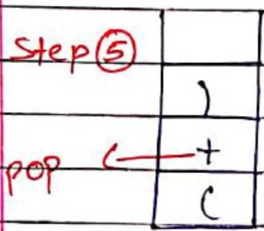
Note:- While pushing Any operator in the Stack first we should Check if there is Any operator Available of same priority or higher priority. first we pop that operator and push into postfix expression. Then push that operator in Stack.

Step ④



(i) Now there is operand in ~~itix~~ infix eqn so we push it into postfix expression.

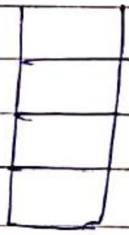
Step ⑤



(i) Now, only close parentheses left in infix eqn, so when close parentheses came we pop the operators from Top of stack and push it into postfix ~~of~~ Exp. until we got the open parentheses.

(ii) So, here we pop + and push it into postfix exp when ~~opera~~ ~~open~~ we got open parantheses we remove it also from stack.

Step. ⑥



postfix exp. = AB+

* Rules. ~~Steps~~ to Convert Infix to postfix

(i) If symbol is an operand — Add (push) it to postfix expr.

(ii) If symbol is '(' left parentheses — push it on the stack.

(iii) If symbol is ')' Righ parentheses — pop all the operator from the stack upto the first left parentheses and Add (push) these operator to postfix exp. and Discard both left And Right parentheses.

(iv) If symbol is operator — pop the operator which have precedence (priority) greater than or Equal to the precendence of the Symbol operator, And Add these popped operators to postfix exp.

- If there is No operator at the top of stack which have higher or ~~sam~~ Equal precedence than scanned operator then we simply push that Operator into Stack.

E.g. Infix= A+B/C*(D+E)-F)

	Symbol	Action	Stack	postfix.
1.	A	Add A to postfix	(	A
2.	+	push + into stack	(+	A
3.	B	Add B to postfix	(+	AB
4.	/	push '/' in stack	(+/	AB
5.	C	Add c to postfix	(+/	ABC
6.	*	pop '/' from stack and Add to postfix then push * to Stack.	(+*	ABC/
7.	(	push '(' in stack	(+*(	ABC/
8.	D	Add ~~to~~ D to postfix	(+*(	ABC/D
9.	+	push '+' ~~to~~ in stack	(+*(+	ABC/D
10.	E	Add E to postfix	(+*(+	ABC/DE
11.	)	pop '+' And Add to postfix	(+*	ABC/DE+
12.	–	~~push '-'~~ pop '*' and '+' and Add to postfix and then push '–' in stack	(–	ABC/DE+*+
13.	F	Add 'F' to postfix	(–	ABC/DE+*+F
14.	)	pop all symbols from and Add it to postfix until you get left '(' parentheses		ABC/DE+*+F–

E.g. Infix :- (A^B*C/(D*E-F)) → First of All we put open and close parentheses to Infix Eqn

	Symbol	Action	Stack	postfix
1.	(	push in Stack	(	
2.	A	Add 'A' to postfix	(	A
3.	^	push '^' in Stack	(^	A
4.	B	Add 'B' to postfix	(^	AB
5.	*	pop '^' and Add iat to postfix and then push * in stack	(*	AB^
6.	C	Add 'C' to postfix	(*	AB^C
7.	/	pop '*' and Add it to postfix and then push / in Stack	(/	AB^C*
8.	(	push in Stack	(/(	AB^C*
9.	D	Add 'D' to postfix	(/(	AB^C*D
10.	*	push * in Stack	(/(*	AB^C*D
11.	E	Add E to postif	(/(*	AB^C*DE
12.	-	pop * and Add it to postfix And then push '-' in stack	(/(-	AB^C*DE*
13.	F	Add F to postfix	(/(-	AB^C*DE*F
14.	~~(~~)	pop '-' and Add to postfix	(/	AB^C*DE*F-
15	)	pop '/' ~~Aad~~ Add it to postfix		AB^C*DE*F-/

postfix Exp. ⇒ AB^C*DE*F-/

DATE : / /

PAGE :

Exam

Q. Q = (A+(B*C-(D/E^F)*G)*H)

S.No.	Symbol	STACK	postfix
1.	(	(	
2.	A	(	A
3.	+	(+	A
4.	(	(+(	A
5.	B	(+(	AB
6.	*	(+(*	AB
7.	C	(+(*	ABC
8.	-	(+(-	ABC*
9.	(	(+(-(	ABC*
10.	D	(+(-(	ABC*D
11.	/	(+(-(/	ABC*D
12.	E	(+(-(/	ABC*DE
13.	^	(+(-(/^	ABC*DE
14.	F	(+(-(/^	ABC*DEF
15.	)	(+(-	ABC*DEF^/
16.	*	(+(-*	ABC*DEF^/
17.	G	(+(-*	ABC*DEF^/G
18.	)	(+	ABC*DEF^/G*-
19.	*	(+*	ABC*DEF^/G*-
20.	H	(+*	ABC*DEF^/G*-H
21	)		ABC*DEF^/G*-H*+

postfix Exp. ⇒ ABC*DEF^/G*-H*+

Q. ((A+B)/D) ^((E-F)*G)

Let Q= (((A+B)/D) ^((E-F)*G))

S.NO.	Symbol	STACK	postfix
1.	(	(	
2.	(	((	
3.	(	(((	
4.	A	(((	A
5.	+	(((+	A
6.	B	(((+	AB
7.	)	((	AB+
	/	((/	AB+
	D	((/	AB+D
	)	(	AB+D/
	^	(^	AB+D/
	(	(^(	AB+D/
	(	(^((	AB+D/
	E	(^((	AB+D/E
	-	(^((-	AB+D/E
	F	(^((-	AB+D/EF
	)	(^(	AB+D/EF-
	*	(^(*	AB+D/EF-
	G	(^(*	AB+D/EF-G
	)	(^	AB+D/EF-G*
	)		AB+D/EF-G*^

postfix= AB+D/EF-G*^

Exam

~~Q.~~ ~~Infix~~

Q. Convert the postfix Expression into Infix Notation.

Q = 12, 7, 3, −, /, 2, 1, 5, +, *, +

scanning from left to Right.

→ 12, (7−3), /, 2, 1, 5, +, *, +

→ 12/(7−3), 2, 1, 5, +, *, +

→ 12/(7−3), 2, (1+5), *, +

→ 12/(7−3), 2*(1+5), +

→ 12/(7−3)+2*(1+5)

Infix exp. = 12/(7−3)+2*(1+5)

Exam

Q. (A+B)/(C*D +E)

Let Q= (A+B)/(C*D+E)

→ $\underline{(AB+)}/(\underline{CD*}+E)$

→ $\underline{(AB+)}/(\underline{CD*E+})$

→ $\underline{AB+CD*E+/}$

postfix Exp. = AB+CD*E+/

Exam

Q Evaluate the following postfix expression:-

2,3,10,+,*,8,2,1,−

→ $2, \underline{(3+10)}, *, 8, 2, 1, -$

→ $\underline{2*(3+10)}, 8, 2, 1, -$

→ $\underline{2*(3+10)}, \underline{8/2}, -$

→ $2*(3+10) - 8/2$

→ $2*(3+10) - 4$

→ $2*13 - 4$

⇒ $52-4$

⇒ 48 Ans.

Q. Translate Infix expr. into its equivalent postfix Expression :-

$$A*(B+D)/E-F*(G+H/K)$$

$\Rightarrow\ A*(\underline{BD+})/E - F*(G+\underline{HK/})$

$\Rightarrow\ A*(\underline{BD+})/E - F*\ \underline{GHK/+}$

$\Rightarrow\ \underline{ABD+*}\ /E - \underline{FGHK/+*}$

$\Rightarrow\ \underline{ABD+*E/} - \underline{FGHK/+*}$

$\Rightarrow\ ABD+*E/FGHK/+*-$

An.