INFIX NOTATION.

→ To add A, B, we write

A+B

→ To multiply A, B we write

A*B

→ The operators ('+' or '*') go in between the operands ('A' and 'B'). This is "Infix" Notation.

PREFIX NOTATION

→ Instead of saying "A plus B", we could say "add A,B" and write +AB

→ "Multiply A,B" would be written

*AB

POSTFIX NOTATION.

→ Another alternative is to put the operators after the operands as in

AB+

and AB*

→ This is POSTFIX NOTATION.

The terms infix, prefix and postfix tell us whether the operators go, between, before, or after the operands.

Precedence and Associativity of Operators

operator	Priority	Associativity
^	High	R→L
*, /	Medium	L→R
+, -	Low	L→R

INFIX TO POSTFIX CONVERSION

- Paranthesis Method
  - Step1 : Fully parenthesis the expression
  - Step2: Replace right paranthesis (closing paranthesis) by moving operator it.
  - Delete left paranthesis (opening paranthesis).
  - e.g Given Expression: A+B*C/D^E-F

$$A\ \big((A+((B*C)/(D\hat{}E))-F\big)$$

- Moving operator to their right paranthesis

((A((BC*(DE^/+F-;

- Delete all left paranthesis
- Note it requires multiple traversals of the expression.

"Here are the steps of the algorithm to convert INFIX to POSTFIX using stack in C".

- Scan all the symbols one by one from left to right in the given Infix Expression.
- If the reading symbol is an operand, then immediately append it to the Postfix Expression.
- If the reading symbol is left Paranthesis '(', thus push it on to stack.
- If the reading symbol is right Paranthesis ')', thus Pop all the contents of the stack until the respective left Paranthesis is popped and append each popped symbol to Postfix Expression.
- If the reading symbol is an operator (+,-,*,/), then Push it onto the stack. However, first, pop the operator which are already on the stack that have higher precedence than the current operator and append them to the postfix. If an open paranthesis is there on Top of the stack, then push the operator into the stack.
- If the input is over, pop all the remaining symbols from the stack and append them to the postfix.

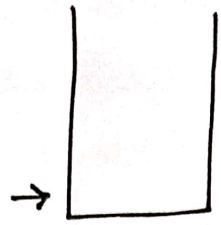
Scanned with OKEN Scanner

DryRun to Infix to Postfix Conversion using Stack in C.

Infix Expression → a * (b+c+d)

(1) Infix: a*(b+c+d) ↑

'a' is an operand append it to the postfix

Stack: 

Postfix: a

(2) a * (b+c+d) ↑

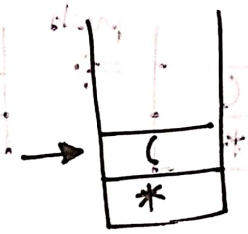
"*" operator push it onto stack after removing higher or equal precedence operator

Stack: 

Postfix: a

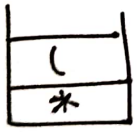
(3) a* (b+c+d) ↑

"(" → push left Paranthesis onto the stack

Stack: 

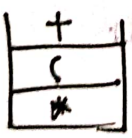
Postfix: a

(4) a*(b+c+d) ↑

Stack: 

Postfix: ab

(5) a*(b+c+d) ↑

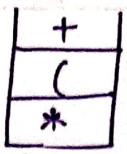
Stack: 

Postfix: ab

Infix

(6) a * (b+c+d)
↑

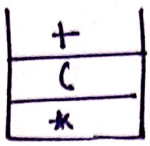
Stack



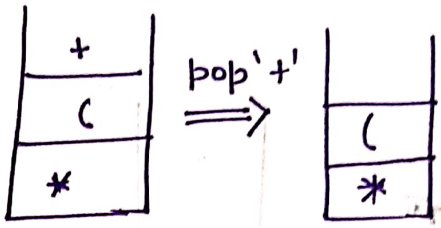
Postfix

abc

(7) a*(b+c+d)
↑



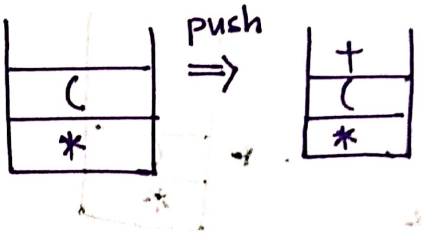
⇒ '+' is an operator, so push it into stack after removing higher or equal priority operators from top to stack.



Postfix ⇒ abc+

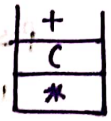
⇒ current top of stack is '+', it has equal precedence with the input symbol '+'. So pop it & append to postfix

⇒ Now the top of stack is '(', so push the input operator.



(8) a*(b+c+d)
↑

'd' is an operand append it to postfix

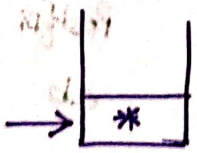


Postfix

abc+d

(9) a*(b+c+d)
↑

')' → right paranthesis, follow steps of an algo & pop all content of stack until respective.

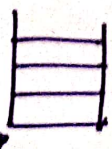


Postfix

abc+d+

(10) a*(b+c+d)↑

i/p → is over, so pop all remaining



Postfix

abc+d+*

FINAL EXPRESSION.

CONVERT POSTFIX TO INFIX USING A STACK.

1. Initialize an empty stack.
2. Traverse the Postfix expression from left to Right
   - If the symbol is an Operand (eg A,B,1,2) push it onto the stack.
   - If the symbol is an Operator (eg +,-,*,/)
     - pop the top elements from the stack. Let's call them Op1 and Op2.
     - form a new infix expression by enclosing op1 and op2 in paranthesis, placing the operator b/w them ((Op1 Operator Op2)).
     - Push this new expression back onto the stack.
3. At the end of the traversal, the stack will contain a single element, which is the final infix expression.

CONVERT INFIX TO PREFIX EXPRESSION

↳ Convert Infix to Prefix Expression using Stacks.

↳ Steps to converting Infix to Prefix using Stacks

1. Reverse the Infix Expression:
   - → traverse the expression from right to left.
   - → reverse the order of Operators and Operands.
   - → Swap opening Paranthesis. with closing paranthesis. and vice-versa.

2. Convert the reversed expression to postfix notation:
   - → use the Infix-to-Postfix algorithm (precedence and associativity rules using a stack.

3. Reverse the postfix expression: to obtain the final prefix expression.

↳ Example:→ A+B*C

→ Step-by-Step Solution.

1. Reverse the Infix Expression

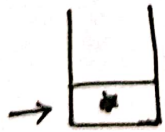
C*B+A

2. Apply Infix to Postfix expression

C * B + A

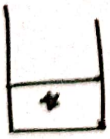
Postfix

C



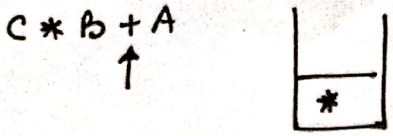
C * B + A

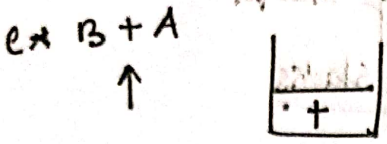
C

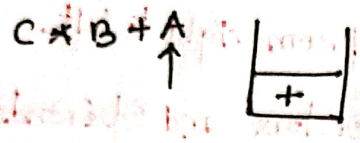


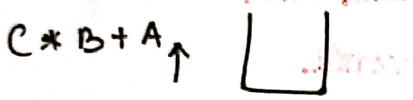
C * B + A

CB









Postfix of Reversed Expression CB*A+

Reverse the Postfix Expression ⇒ +A*BC

Infix to Prefix => +A*BC

FINAL ANSWER.

PREFIX TO INFIX.

1. Initialize an empty stack.
2. Traverse the prefix expression from RIGHT to LEFT.
   - If the symbol is an operand (e.g A, B, 1, 2), push it onto the stack.
   - If the symbol is an operator (e.g +, -, *, /)
     - ↳ Pop the two elements from the stack. Let's call them op1 and op2.
     - ↳ Form a new infix expression by placing op1 and op2 b/w the operator, enclosed in parenthesis (op1 operator op2).
     - ↳ Push this new expression back onto the stack.
3. At the end of the Traversal, the stack will contain a single element, which is the final infix expression.

POSTFIX TO PREFIX USING A STACK.

1. Initialize an empty stack

2. Traverse the postfix expression from left to right
   - If the symbol is an operand (e.g A, B, 1, 2), push it onto the stack.
   - If the symbol is an operator (e.g +, -, *, /)
     - pop the top two elements from the stack. Let's call them op1 and op2.
     - create a new expression by concatenating the operator with op1 and op2 in prefix form (operator op1 op2).
     - push this new expression back onto the stack.

3. At the end of the Traversal → the stack will contain a single element, which is the prefix expression.

EXAMPLE $ABC*+D-$

Step-by-step Solution:-

1. Initialize an empty stack.
2. Traverse each character in the postfix expression:-
   - Operand A → Push to stack : ["A"]. A B C * + D −
   - Operand B → Push to stack : ["A", "B"] A B C * + D −
   - Operand C → Push to stack: ["A", "B", "C"] A B C * + D −
   - operator *: A B C * + D −
     ↳ pop C and B from the stack
     ↳ from the expression *BC & push back on stack
     ["A", "*BC"]
   - Operator +: A B C * + D −
     ↳ pop *BC and A from the stack
     ↳ from the expression +A*BC and push back on the stack : stack ["+A*BC"]
   - Operand D:
     ↳ Push to stack: stack ["+A*BC", "D"]
     A B C * + D −
   - Operator −
     ↳ Pop D and +A*BC from the stack
     ↳ from the expression −+A*BCD.

Final Prefix expression.

PREFIX TO POSTFIX.

1. Initialize an empty stack.

2. Traverse the prefix expression from RIGHT to LEFT.
   - If the symbol is an Operand, push on to stack.
   - If the symbol is an Operator
     - ↳ Pop the top two element from the stack
     - ↳ form new postfix expression by concatenating op1 and op2 with the operator at the end (op1 and op2 operator)
     - ↳ Push this new expression back onto stack

3. At the end of the Traversal, the stack will contain a single element, which is the final postfix expression.