K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Data Structure

ENCS205

School of Engineering & Technology (SOET)
K.R. MANGALAM University

UNIT-2

Session 24: Infix to Postfix Conversion (Stack App)

Recap

- Stack definitions
- Its operations
  - Push
  - Pop
  - Peek
- Representation
  - Using array
- uses

Unit 2

Stack:

- **Definition**: LIFO data structure.
- **Operations**: Push, Pop, Peek, is Empty.
- **Implementation:** Array-based with a top pointer.
- Applications of Stack
- **Expression Notation:**
- **Conversion:** Infix to Postfix
- **Application**: Tower of Hanoi.

Sessions 24 Outlook

- Notations for Arithmetic Expression
  - INFIX
  - POSTFIX
  - PREFIX
- Conversions of an arithmetic expression
- Practice Questions

Notations for Arithmetic Expression

Evaluate an expression represented by a String. The expression can contain parentheses, you can assume parentheses are well-matched. For simplicity, you can assume only binary operations allowed are +, -, *, and /. Arithmetic Expressions can be written in one of three forms:

Infix Notation

Prefix Notation

Postfix Notation

Notations for Arithmetic Expression

Type of Expression	Description	Example
Infix	Operators are placed in between the operands	x * y + z 3 *(4 + 5) (x + y)/(z * 5)
Prefix (Polish)	Operators are placed before the corresponding operands	+z*xy *3+45 /+xy*z5
Postfix (Reverse Polish)	Operators are placed after the corresponding operands	xy*z+ 345+* xy+z5*/

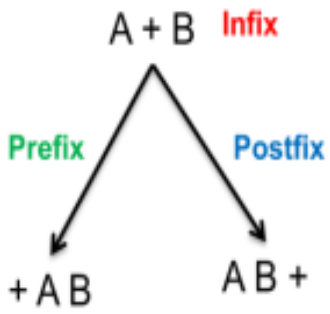
https://www.sarthaks.com/3456368/notations-for-arithmetic-expressions

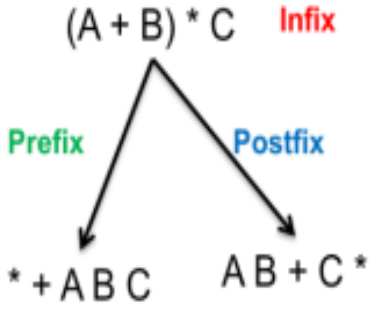
The standard precedence rules for arithmetic expression

Operators	Associativity	Precedence
^ exponentiation	Right to left	Highest followed by *Multiplication and /division
*Multiplication, /division	Left to right	Highest followed by + addition and - subtraction
+ addition, - subtraction	Left to right	lowest

Conversion

- Infix to Postfix
- Infix to Prefix
- Postfix to Prefix
- Postfix to In fix
- Pre fix to Infix
- Pre fix to postfix





Infix to Postfix

Algorithm for Conversion of Infix to Postfix :

- 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 parenthesis '(', then Push it onto the Stack.
- If the reading symbol is right parenthesis ')', then Pop all the contents of the stack until the respective left parenthesis is popped and append each popped symbol to Postfix Expression.

Infix to Postfix

Algorithm for Conversion of Infix to Postfix :

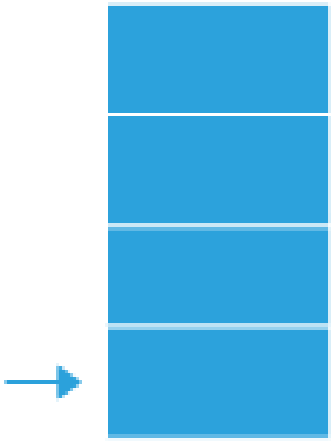
- If the reading symbol is an operator (+, –, *, /), then Push it onto the Stack. However, first, pop the operators which are already on the stack that have higher or equal precedence than the current operator and append them to the postfix. If an open parenthesis is there on top of the stack then push the operator into the stack.
- If the input is over, pop all the remaining symbols from the stack and append them to the postfix.

Infix to Postfix

Infix expression => a * (b + c + d)

(1)

Infix	Stack	Postfix
a * (b + c + d)		a

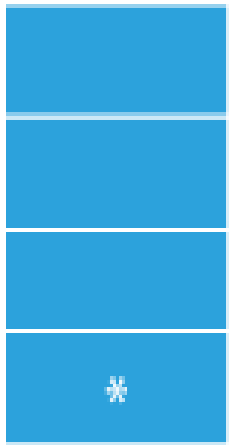


'a' is an operand append it to the post fix

Infix to Postfix

(2)

Infix	Stack	Postfix
a * (b + c + d)		a

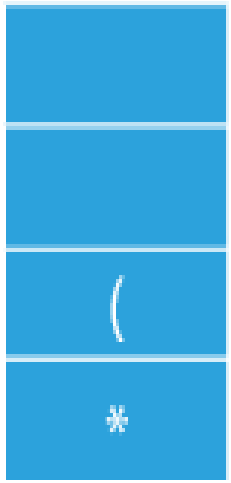


'*' is an operator push it into the stack after removing the higher or equal precedence operator.

Infix to Postfix

(3)

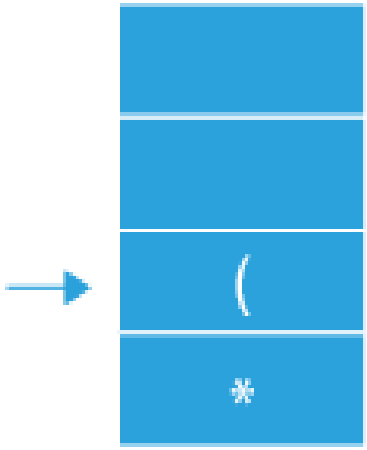
Infix	Stack	Postfix
a * (b + c + d)		a



'(' is left parenthesis
push it into the stack

Infix to Postfix

(4)

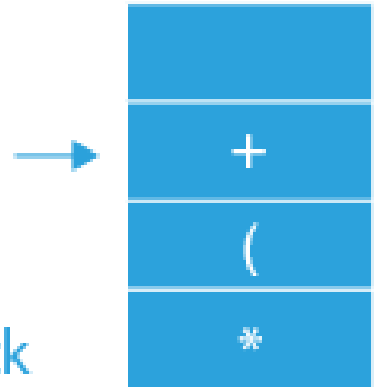
Infix	Stack	Postfix
a * (b + c + d)		ab

‘b’ is operand
append to postfix

Infix to Postfix

(5)

Infix	Stack	Postfix
a * (b + c + d)		ab

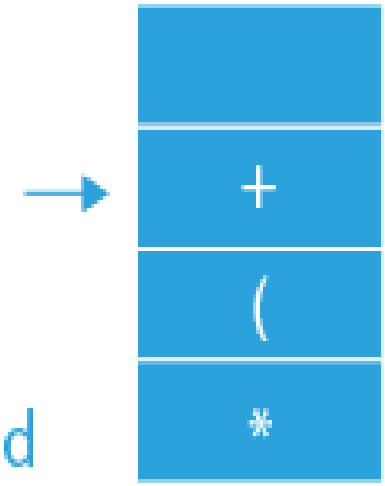


'+' is an operator follow the step 5 of algorithm. Top of stack is open parantheses, So just push it.

Infix to Postfix

(6)

Infix	Stack	Postfix
a * (b + c + d)		abc



'c' is an operator follow the step 2 of algorithm and append it to the postfix.

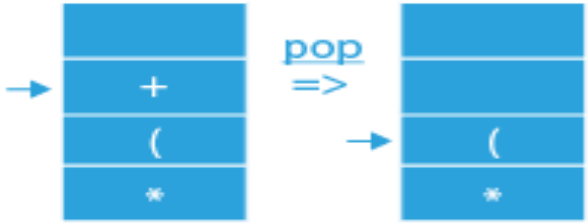
Infix to Postfix

(7) Infix

a * (b + c + d)

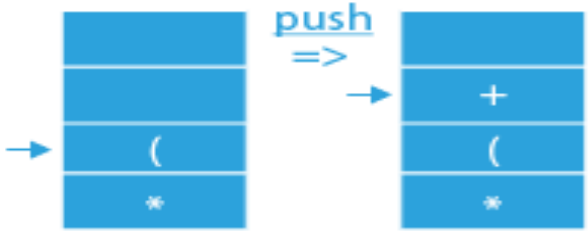
=> '+' is an operator, so push it into stack after removing higher or equal priority operators from top of stack.

=> Current top of stack is '+', it has equal precedence with with input symbol '+'. So, pop it and append to postfix.



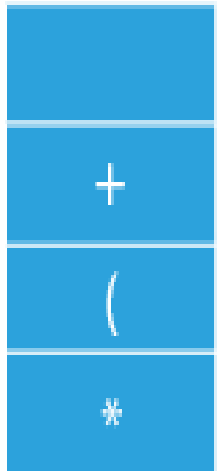
postfix => a b c +

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



Infix to Postfix

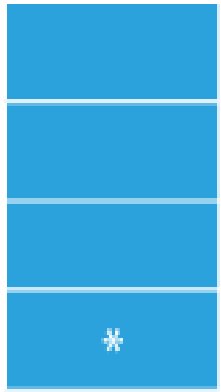
(8)

Infix	Stack	Postfix
a * (b + c + d)		a b c + d
↑ 'd' is an operand. append it to postfix.	→ 	

Infix to Postfix

(9)

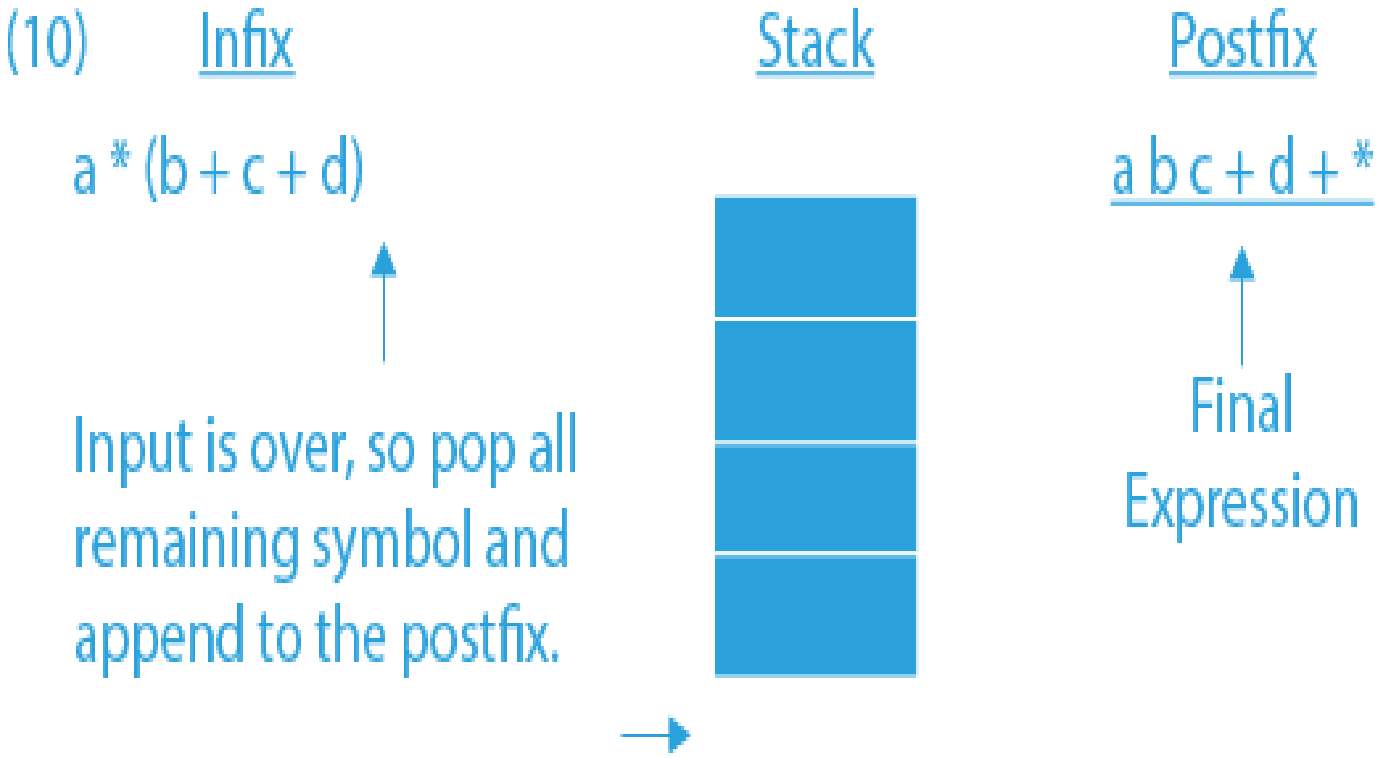
Infix	Stack	Postfix
a * (b + c + d)		a b c + d +



')' is an right paranthesis, flow step n of algorithm and pop all content of stack until respective '(' is popped and append to the output.

https://www.prepbytes.com/blog/stacks/infix-to-postfix-conversion-using-stack/

Infix to Postfix



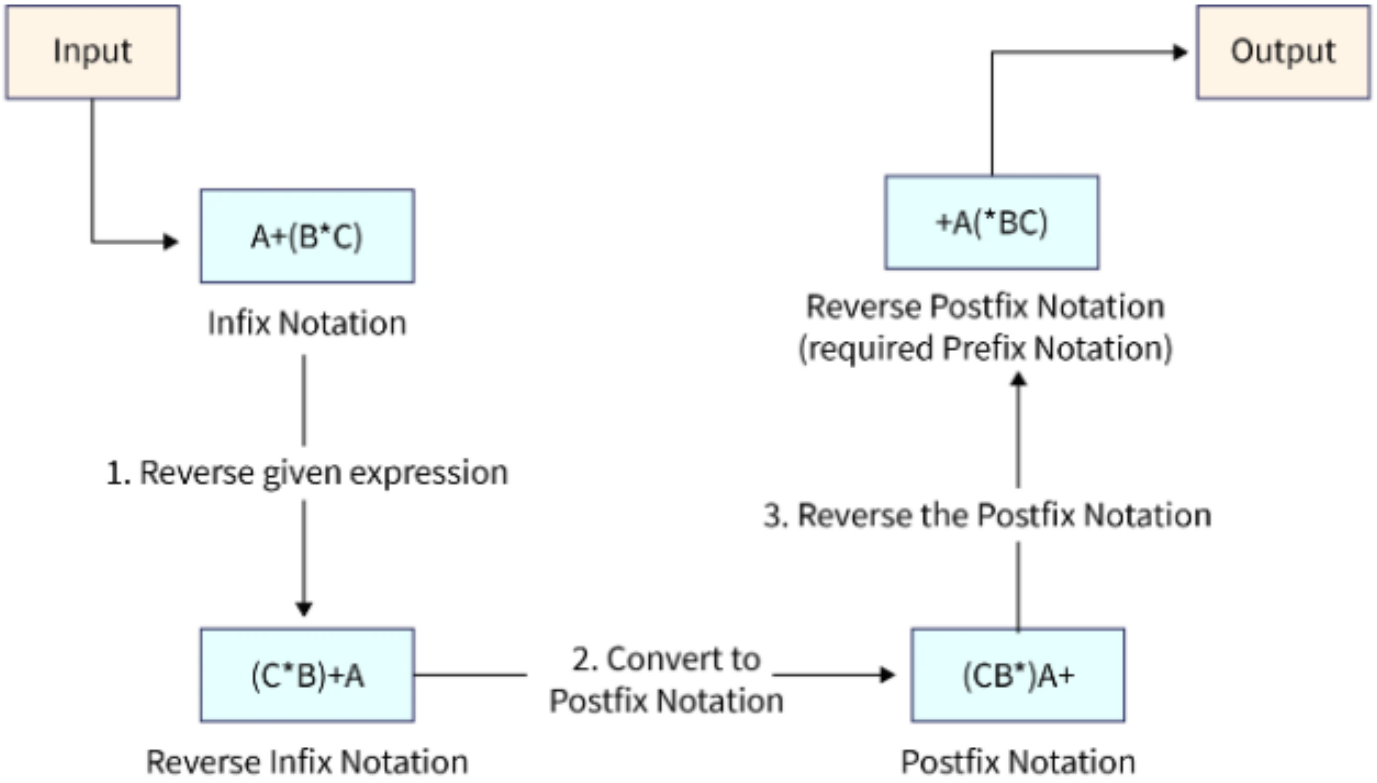
https://www.prepbytes.com/blog/stacks/infix-to-postfix-conversion-using-stack/

Infix to Prefix

Converting Infix Expression to Prefix Expression:

- First, flip the infix expression. Keep in mind that when reversed, each "(" will become a ")" and each ")" a "(".
- The second step is to "nearly" postfix the reversed infix expression.
- Instead of performing the pop operation to remove operators with greater than or equal precedence during the conversion to postfix expression, we will only remove the operators from the stack that have a higher precedence in this case.
- Reverse the postfix expression in step three.
- Infix expressions are converted to postfix forms using the stack

Infix to Prefix



Postfix to Pre fix

Conversion of Postfix to Prefix expression :

- Scan all symbols one by one from left to right in the given prefix expression .
- If the reading symbol is an operand , push it into the stack .

- If the reading symbol is an operator , then a. Pop two expression from the stack , operand1 and operand2 , which is operand for the current operator b. Push operator + operand2 + operand1 into the stack

- If there is no symbol left then stop the process . Top of the stack will have the required infix expression .

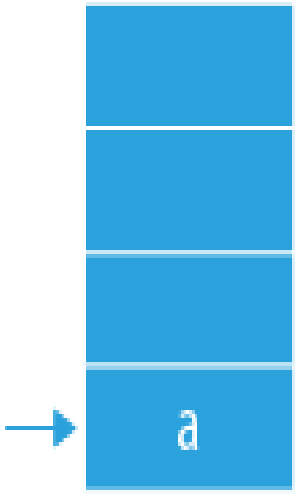
Postfix to Pre fix

(1) Postfix expression

a b c / - a d / e - *

'a' is an operand push it into the stack.

Stack



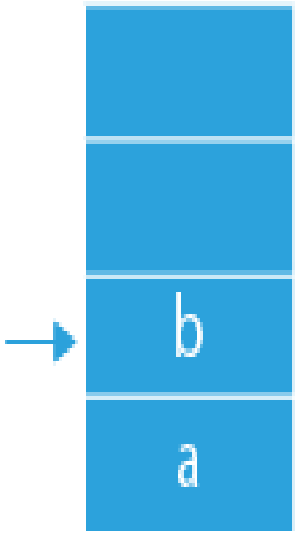
Postfix to Pre fix

(2) Postfix expression

Stack

a b c / - a d / e - *

'b' is an operand push it into the stack.



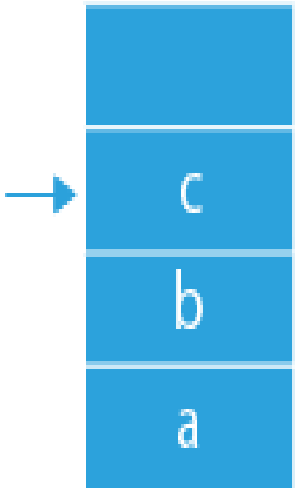
Postfix to Pre fix

(3) Postfix expression

a b c / - a d / e - *

'c' is an operand. So, push it into the stack.

Stack



Postfix to Pre fix

(4) Postfix expression

a b c / - a d / e - *

↑

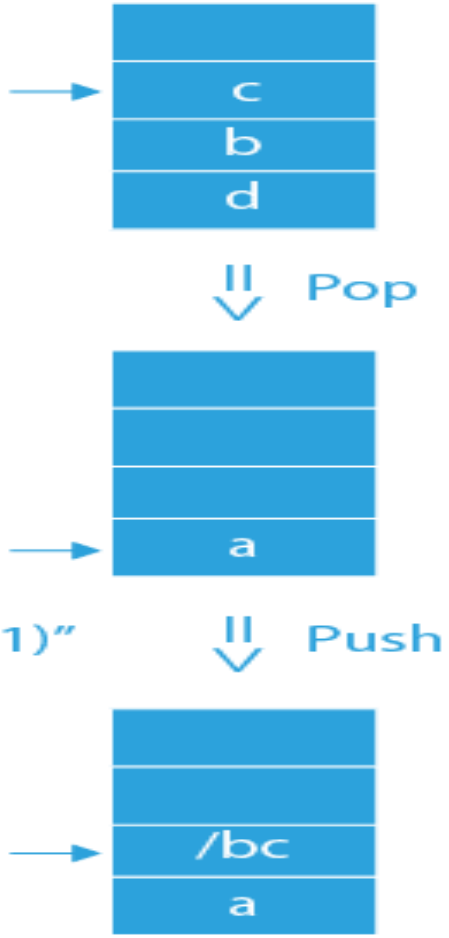
'/' is an operator. Pop two expressions from the stack.

Operand 1 = c
Operand 2 = b
Operator = /

Push
"(Operand 2, Operator , Operand 1)"

=> / b c
into the stack.

Stack



Postfix to Pre fix

(5) Postfix expression

a b c / - a d / e - *

'-' is an operator. So, operand 1 = /bc

Operand 2 = a

push (operand 2 operator operand 1)
i.e, (a - (b / c)) into the stack.

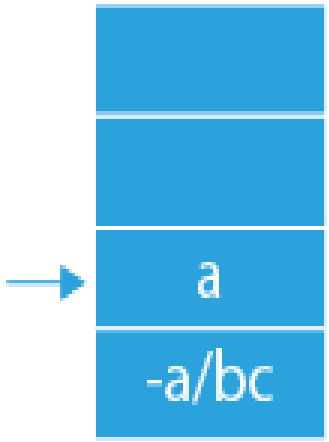
Postfix to Pre fix

(6) Postfix expression

a b c / - a d / e - *

'a' is an operand. So, push it into the stack.

Stack



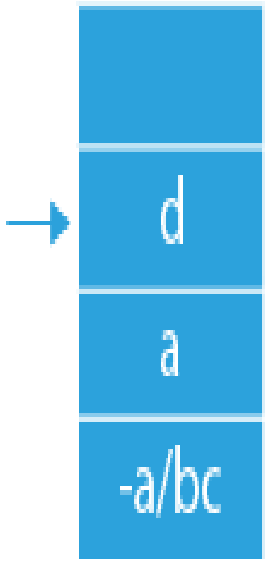
Postfix to Pre fix

(7) Postfix expression

Stack

a b c / - a d / e - *

'd' is an operand. So, push it into the stack.



Postfix to Pre fix

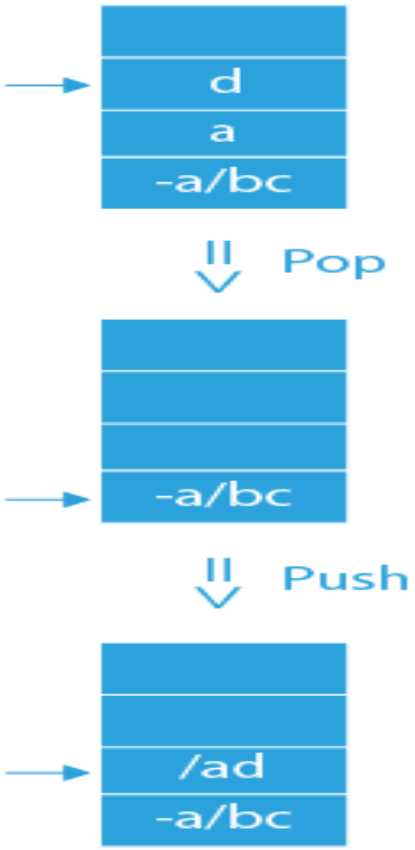
(8) Postfix expression

a b c / - a d / e - *

'/' is an operator.
Operand 1 = d
Operand 2 = a

Push, '/ad' into the stack.

Stack

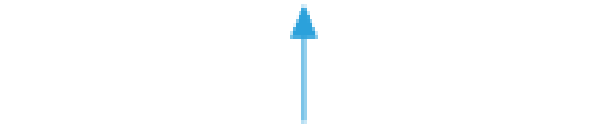


Postfix to Pre fix

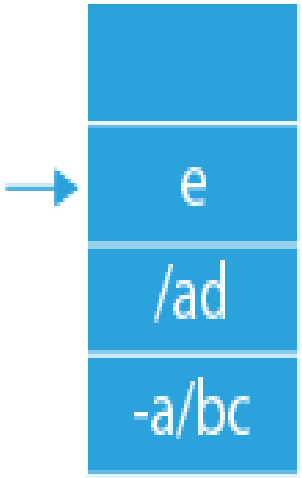
(9) Postfix expression

Stack

a b c / - a d / e - *



'e' is an operand. So, push it into the stack.



Postfix to Pre fix

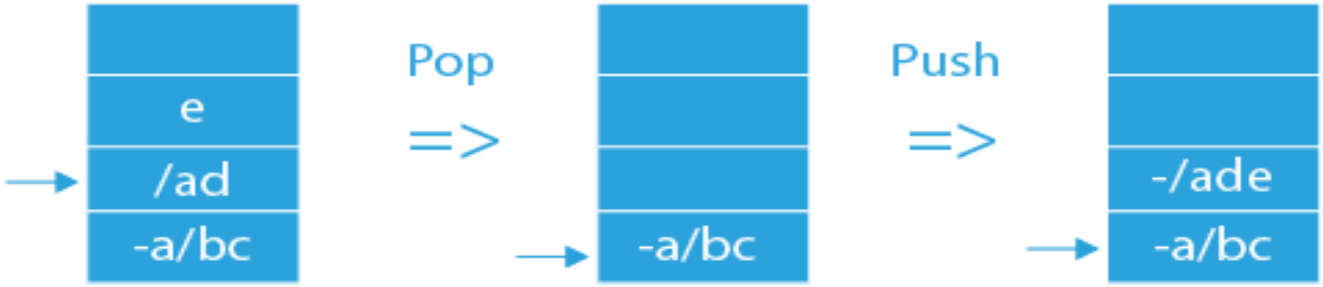
(10) Postfix expression

a b c / - a d / e - *

Operator = -

Operand 1 = e Operand 2 = /ad

Push, -/ade into the stack.

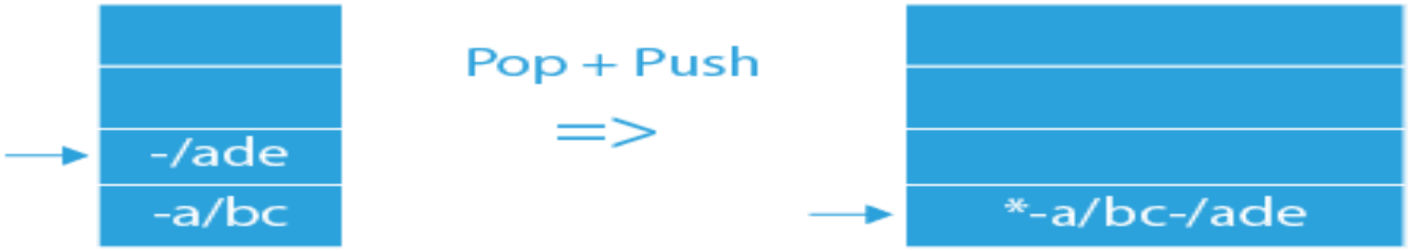


Postfix to Pre fix

(11) Postfix expression

a b c / - a d / e - *

Operator = *
Operand 1 = -/ade Operand 2 = -a/bc
Push, (operand 2 operator operand 1) i.e *-a/bc-/ade
Stack:



Top of stack is our required prefix expression.

https://www.prepbytes.com/blog/stacks/postfix-to-prefix conversion/#:~:text=For%20converting%20Postfix%20to%20Prefix,will%20be%20our%20prefix%20expression%20.

Postfix to Infix

1. Scan the given postfix expression from **left to right** character by character.
2. If the character is an operand, push it into the stack.
3. But if the character is an operator, pop the top two values from stack. Concatenate this operator with these two values (**2nd top value+operator+1st top value**) to get a new string.
4. Now push this resulting string back into the stack.
5. Repeat this process untill the end of postfix expression. Now the value in the stack is the infix expression.

Postfix to Infix

Input String	Postfix Expression	Stack (Infix)
ab*c+	b*c+	a
ab*c+	*c+	ab
ab*c+	c+	(a*b)
ab*c+	+	(a*b)c
ab*c+		((a*b)+c)

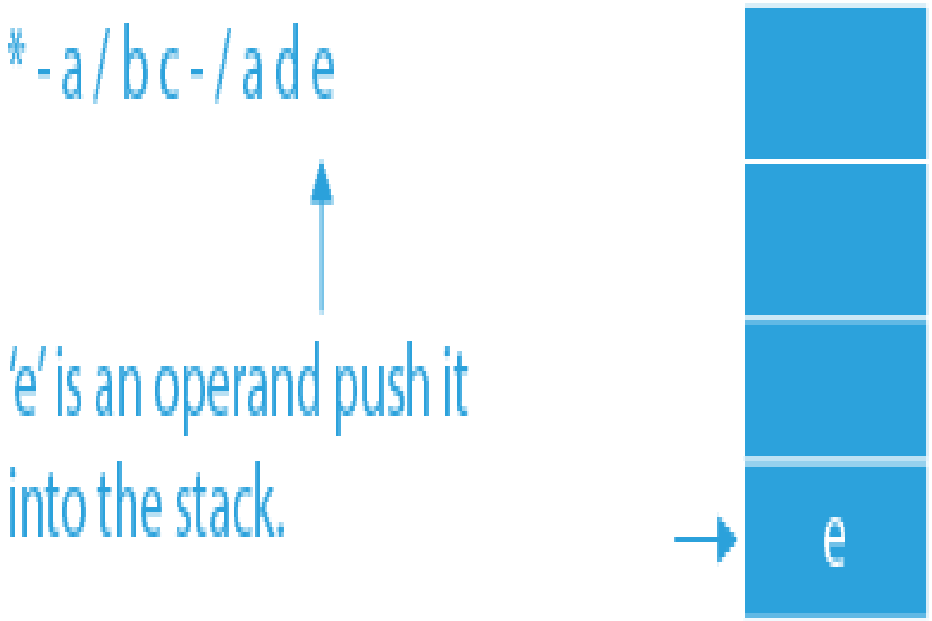
Prefix to Infix converter step by step

1. Start scanning the prefix expression from right to left.
2. If the current character is an operand (a number or a variable), push it onto a stack.
3. If the current character is an operator (+, -, *, /, ^), pop two operands from the stack and concatenates them with the operator in between to form an infix sub-expression. Then push the infix sub-expression onto the stack.
4. Repeat steps 2 and 3 until the entire prefix expression has been scanned.
5. The final infix expression will be the only item remaining on the stack.

Prefix to Infix converter step by step

(1) Prefix expression

Stack



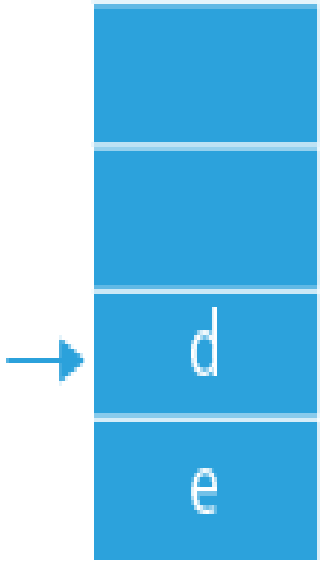
Prefix to Infix converter step by step

(2) Prefix expression

Stack

*-a/bc-/ade

'd' is an operand push it into the stack.



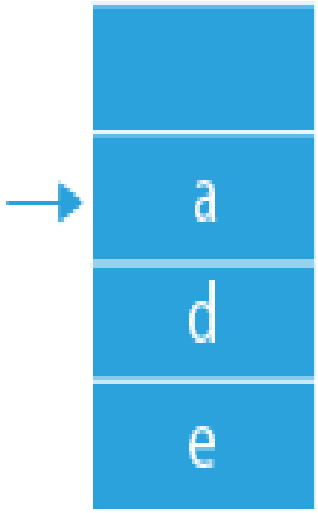
Prefix to Infix converter step by step

(3) Prefix expression

Stack

* - a / b c - / a d e

'a' is an operand. So, push it into the stack.



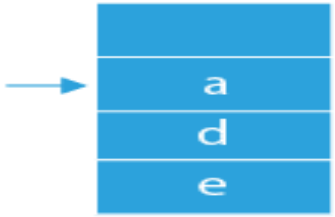
Prefix to Infix converter step by step

(4) Prefix expression

Stack

* - a / b c - / a d e

'/' is an operator. Pop two expressions from the stack.



Pop

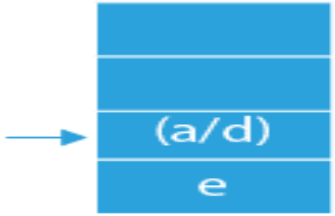
Operand 1 = a
Operand 2 = d
Operand 3 = /

Calculate the expression
"(Operand 1 + Operator + Operand 2)"

Push

=> (a/d)

NOTE: + Indicate concatenation.



Prefix to Infix converter step by step

(5) Prefix expression

* - a / b c - / a d e

‘-’ is an operator. So, operand 1 = (a/d)

Operand 1 = (a/d) Operand 2 = e Operand = -

push (+ operand 1 + operator + operand 2 +) into the stack

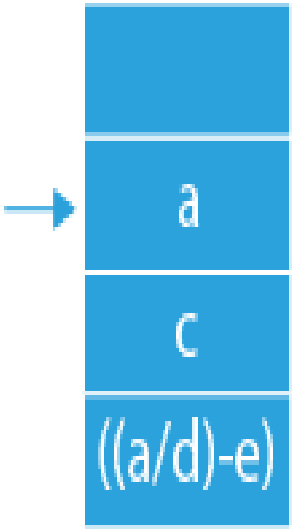
Prefix to Infix converter step by step

(6) Prefix expression

Stack

*-a/bc-/ade

'c' is an operand. So, push it into the stack.



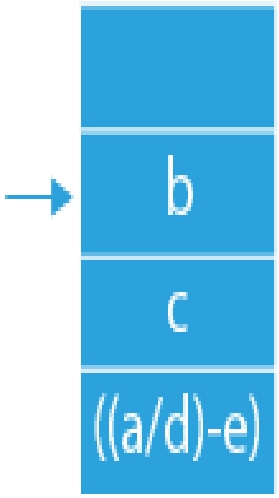
Prefix to Infix converter step by step

(7) Prefix expression

Stack

* - a / b c - / a d e

'b' is an operand. So, push it into the stack.



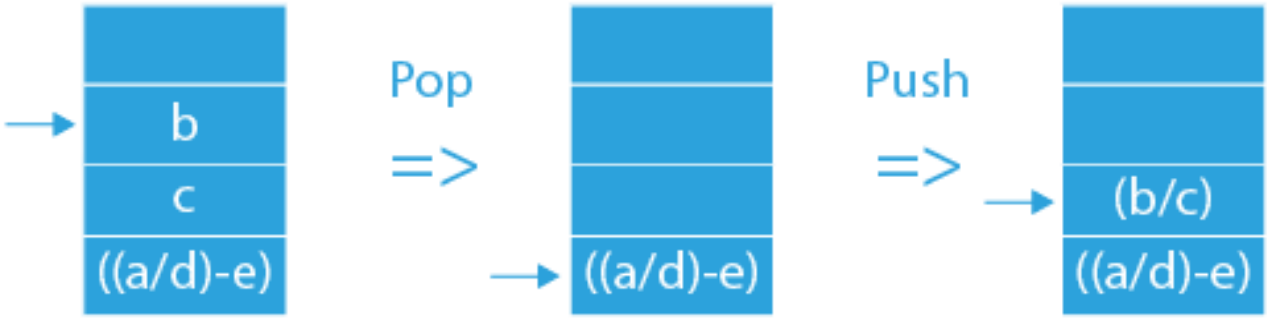
Prefix to Infix converter step by step

(8) Prefix expression

* - a / b c - / a d e

'/' is an operator.

Operand 1 = b Operand 2 = c



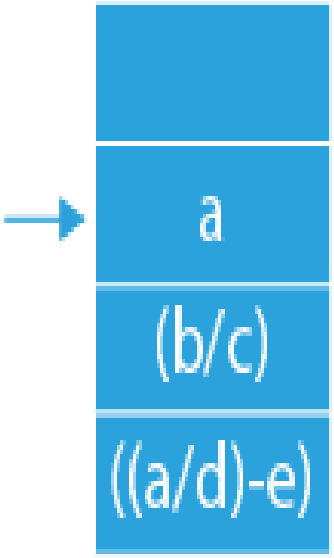
Prefix to Infix converter step by step

(9) Prefix expression

Stack

* - a / b c - / a d e

'a' is an operand. So, push it into the stack.



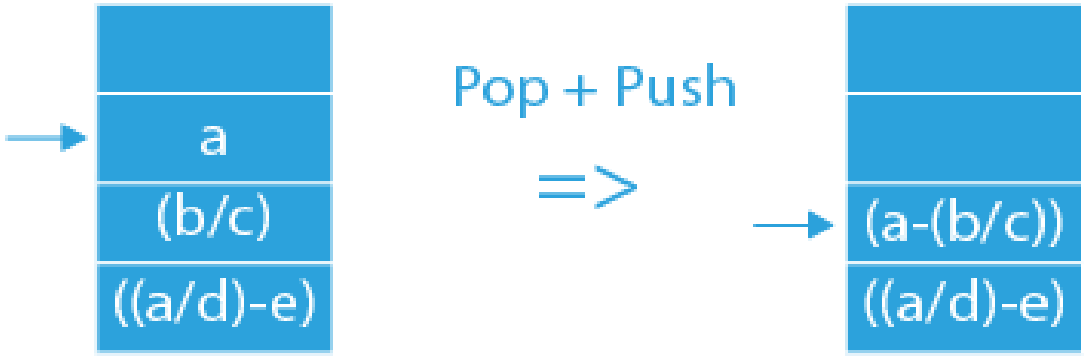
Prefix to Infix converter step by step

(10) Prefix expression

* - a / b c - / a d e

Operator = -

Operand 1 = a Operand 2 = (b/c)



Prefix to Infix converter step by step

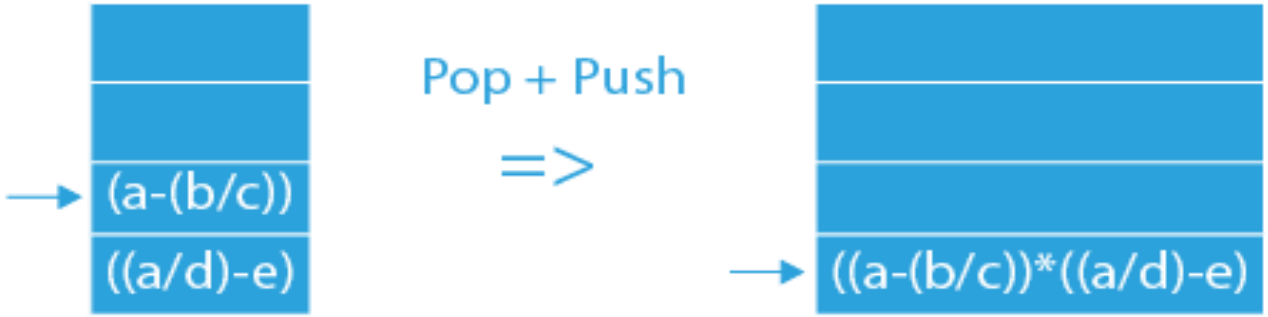
(11) Prefix expression

* - a / b c - / a d e

Operator = *

Operand 1 = (a-(b/c)) Operand 2 = ((a/d)-e)

Stack:



https://www.prepbytes.com/blog/stacks/conversion-of-prefix-expression-to-infix-expression/

Prefix to Postfix converter step by step

Step 1: Scan all symbols one by one from right to left in the given prefix expression.

Step 2: If the reading symbol is an operand, push it into the stack.

Step 3: If the reading symbol is an operator, then

a. Pop two expressions from the stack, operand1, and operand2, which is the operand for the current operator

b. Push operand1 + operand2 + operator into the stack

Step 4: If there is no symbol left then stop the process. The top of the stack will have the required postfix expression.

Prefix to Postfix converter step by step

Step 1: Scan all symbols one by one from right to left in the given prefix expression.

Step 2: If the reading symbol is an operand, push it into the stack.

Step 3: If the reading symbol is an operator, then

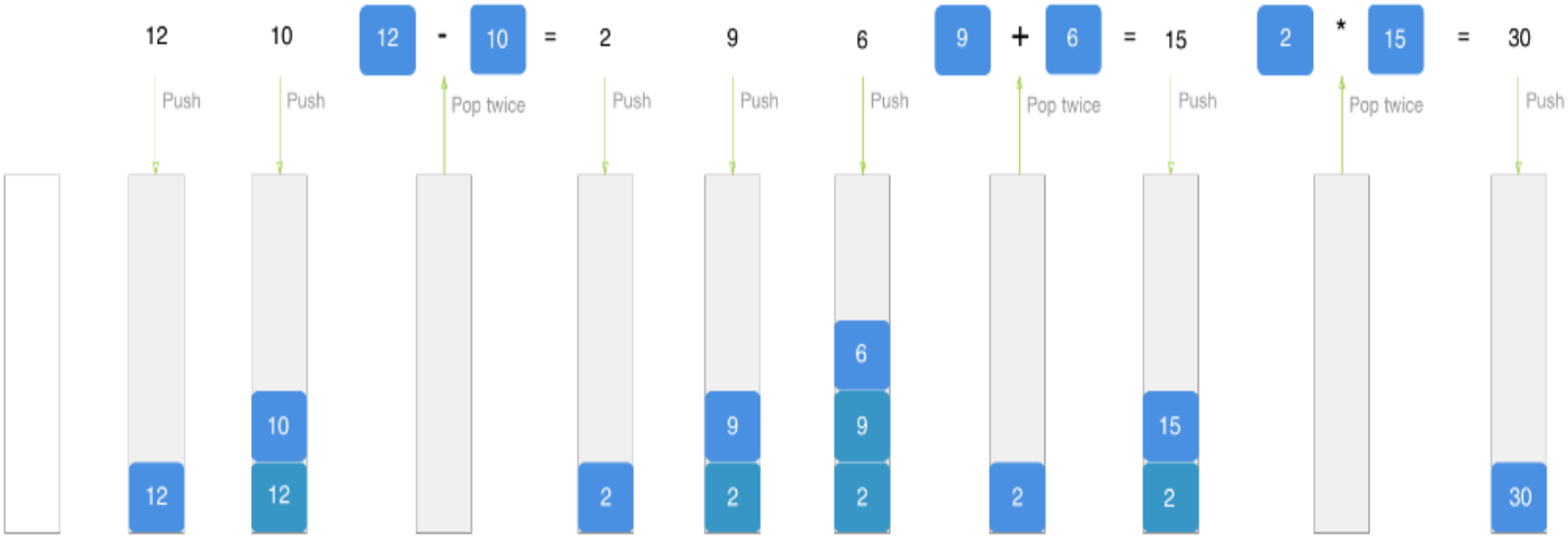
a. Pop two expressions from the stack, operand1, and operand2, which is the operand for the current operator

b. Push operand1 + operand2 + operator into the stack

Step 4: If there is no symbol left then stop the process. The top of the stack will have the required postfix expression.

Postfix Evaluation Example

Example : 12 10 - 9 6 + *



Prefix Evaluation Example

Expression: - * + 2 2 3 10

Reversed Expression: 10 3 2 2 + * -

Scanned Character	Stack	Explanation
10	10	The scanned character is an operand, so push it onto the stack.
3	10 3	The scanned character is an operand, so push it onto the stack.
2	10 3 2	The scanned character is an operand, so push it onto the stack.
2	10 3 2 2	The scanned character is an operand, so push it onto the stack.
+	10 3 4	The scanned character is an operator, so pop two elements from stack, apply the operator to them, push the result back into the stack.
*	10 12	The scanned character is an operator, so pop two elements from stack, apply the operator to them, push the result back into the stack.
-	2	The scanned character is an operator, so pop two elements from stack, apply the operator to them, push the result back into the stack.

Infix Expression to Postfix Expression: Demo

Question: Validate the given expression: 69 ^ 87 - 67 / 29 + 77 * 44

Step 1: (69 ^ 87 **)** - 67 / 29 + 77 * 44

Note: Since ^ has highest priority it is parenthesized first.

Step 2: (69 ^ 87) - **(** 67 / 29 **)** + 77 * 44

/ and * have the next priority but / is parenthesized first because they follow left associativity.

Step 3: (69 ^ 87) - (67 / 29) + **(** 77 * 44 **)**

/ and * have the next priority but * is parenthesized second because they follow left associativity.

Step 4: ((69 ^ 87) - (67 / 29)) + (77 * 44)

Step 5: (((69 ^ 87) - (67 / 29)) + (77 * 44))

Infix expressions into Postfix expressions

Solution:

(a) (A–B) * (C+D)

[AB–] * [CD+]

AB–CD+*

(b) (A + B) / (C + D) – (D * E)

[AB+] / [CD+] – [DE*]

[AB+CD+/] – [DE*]

AB+CD+/DE*–

Postfix expressions into Infix expressions

Solution:

Postfix: (**ABC/-AK/L-***

Infix expression: **((A-(B/C))*((A/K)-L))**

Test Yourself

1. Why is there a need to convert from infix to postfix and then evaluate, instead of directly evaluating infix?

a: It is easier to read for humans

b: It is easier to It is easier to parse (be read by computer).

c: Both a and b

d: None of these.

2. Postfix evaluation of 1 2 + 3 9 - + 4 + is _____.

a: 3

b: 2

c: 1

d: 4

3. Convert the expression from infix to postfix 1 + 1 - 3 * 4 / 2 + 3 - 8?

a: 1 1 + 3 - 4 * 2 / + 3 8 -

b: 1 1 + 3 - 4 / 2 * 3 + 8 -

c: 1 1 + 3 4 * 2 / - 3 + 8 -

d: 1 1 * 3 + 4 - 2 + 3 / 8 –

4. Convert the expression from Infix to Postfix a + b * (c ^ d - e) ^ (f + g * h) - i?

a: ab^cd-ef+d*h^*-i+

b: ab^cd-ef+d*h*^-i+

c: abcd^e-fgh*+^*+i-

d: abcd^e-fgh*+*^+i-

5. Convert the expression from Infix to Postfix a + b * c - d ^ e ^ f?

a: abc * + def ^^-

b: abc * + de ^ f ^-\

c: ab + c *d - e ^ f ^

d: -+ a * bc ^^ de

Correct Answers

1.b
2.c
3.c
4.c
5.a

Practice Questions

- Translate the following infix expression to its postfix equivalent expression: A + B * (C + D) / F + D * E Also write the algorithm that you used for this conversion and evaluate the postfix expression's value for A=10, B=27, C=3 D=5 F=4 and ME= 2

- How can you use stack to convert an infix expression to postfix? Convert infix expression (A + B)*(C - D) to postfix using stack.

- Infix Expression: (AX + (B * C))

    a) Postfix Expression
    b) Prefix Expression

Practice Questions

- Infix Expression: (AX * (BX * (((CY + AY) + BY) * CX))) ;
  a) Postfix Expression
  b) Prefix Expression

- Infix Expression: ((H * ((((A + ((B + C) * D)) * F) * G) * E)) + J) ;
  a) Postfix Expression
  b) Prefix Expression

- Postfix: abc*d/+ed*-
  a) Infix Expression
  b) Prefix Expression

Review

Type of arithmetic Expression
Expression Evolutions
Operator Precedence
Conversion:

- Infix to Postfix
- Infix to Prefix
- Postfix to Prefix
- Postfix to In fix
- Pre fix to Infix
- Pre fix to postfix


THANK
YOU