Appointment

Precedence and Associativity

Precedence : 10 + 20*20 => 10+40 => 50

Associativity(L to R): 10 + 2~~0~~-3 => 12-3 => 9

Associativity (R to L): 2^1^2 => 2^1 => 2

$\uparrow$ precidence

operators	Associativity
^	R to L
*, /	L to R
+, -	L to R

( )

Prefix and Postfix

Infix	Prefix	Postfix
x+y*z	+x*yz	xyz*+
(x+y)*z	*+xyz	xy+z*

Steps for Postfix Conversion:

x+y*z => (x+(y*z))
=> (x+(yz*))
=> xyz*+

(x+y)*z => ((x+y)*z)
=> ((xy+)*z)
= xy+z*

Infix to postfix Conversion:

I/P: infix = "a+b*c"
O/P: postfix = "bc*a+"

I/P: infix = "(a+b)*c"
O/P: postfix = "ab+c*"

I/P: infix : "a^b^c"
O/P: postfix: "abc^^"

I/P: infix : "(a+b)*(c+d)"
O/P: postfix : ab+cd+*

Naive Approach:

infix = $a+b*c \Rightarrow (a+(b*c)) \Rightarrow (a+(bc*)) \Rightarrow abc*+$

infix = $(a+b)*c \Rightarrow ((a+b)*c) \Rightarrow ((ab+)*c) \Rightarrow ab+c*$

infix = $a \wedge b \wedge c \Rightarrow (a \wedge (b \wedge c)) \Rightarrow (a \wedge (bc\wedge)) \Rightarrow abc\wedge\wedge$

infix = $(a+b)*(c+d) \Rightarrow ((a+b)*(c+d)) \Rightarrow ((ab+)*(cd+)) \Rightarrow ab+cd+*$

Infix to Postfix using Stack: Efficient Approach

① Create an empty stack (st)

② Do following for every character x from left to right.

③ If x is :
a) Operand : Output it
b) Left parenthesis : Push to st
c) Right parenthesis : Pop from st until left parenthesis is found. Output the popped operators.
d) Operator : If st is empty, push x to st.
Else compare with st top.
(i) Higher precedence (than st top), push to st.
(ii) Lower precedence, pop st top and output until a higher precedence operator is found. Then push current to st.
(iii) Equal precedence, ~~we~~ use associativity.

④ Pop and output everything from st.

2002

Appointment

Infix to postfix using stack

Q1) Input: a+b*c

26 December Thursday

Week-52

360-005

1st Month '03

1 We, 2 Th, 3 Fr, 4 Sa, 5 Su, 6 Mo, 7 Tu, 8 We, 9 Th, 10 Fr, 11 Sa, 12 Su, 13 Mo, 14 Tu, 15 We, 16 Th, 17 Fr, 18 Sa, 19 Su, 20 Mo, 21 Tu, 22 We, 23 Th, 24 Fr, 25 Sa, 26 Su, 27 Mo, 28 Tu, 29 We, 30 Th, 31 Fr

JAN

Input Symbol	Stack	Result (Postfix)
a		a
+	+	a
b	+	ab
*	* +	ab
c	* +	abc
Pop everything one by one and print	+	abc*
		abc*+

^

*, /

+, −

(

Precedence

2) Input: (a+b)*c

Input Symbol	Stack	Result (Postfix)
(	(	
a	(	a
+	+ (	a
b	+ (	ab
)		ab+
*	*	ab+
c	*	ab+c
Pop everything		ab+c*

Appointment

Algorithm for evaluation of postfix

```
1) Create an empty stack st.
2) Traverse through every symbol x of given postfix
    1) If x is operand, push to st.
    2) Else (x is an ~~operand~~ operator)
        (i) op1 = st.pop();
        (ii) op2 = st.pop();
        (iii) Compute op2 x op1 and push the result
              to st.
    3) Return st.top
```

Postfix evaluation example   Input: 10 2 3 ^ ^

Input Symbol(x)	Stack(st)
10	10
2	2 / 10
3	3 / 2 / 10
^	8 / 10
^	10^8

Infix to Prefix Conversion (Naive Approach)

Two steps:
① Fully parenthesize ② start converting from innermost to outermost.

$x+y*z \Rightarrow (x+(y*z)) \Rightarrow (x+(*yz)) \Rightarrow +x*yz$

$(x+y)*z \Rightarrow ((x+y)*z) \Rightarrow ((+xy)*z) \Rightarrow *+xyz$

$x \wedge y \wedge z \Rightarrow (x \wedge (y \wedge z)) \Rightarrow (x \wedge (\wedge yz)) \Rightarrow \wedge x \wedge yz$

$(x+y)*(z+w) \Rightarrow ((x+y)*(z+w)) \Rightarrow ((+xy)*(+zw)) \Rightarrow *+xy+zw$

Infix to ~~pr~~ prefix (efficient) Example: $x+y/z-w*u$

Input Symbol(c)	Stack	Prefix
u		u
*	*	u
w	*	uw
-	-	uw*
z	-	uw*z
/	/ -	uw*z
y	/ -	uw*zy
+	+ -	uw*zy/
x	+ -	uw*zy/x
		uw*zy/x+-

output: $-+x/yz*wu$

23
December
Monday

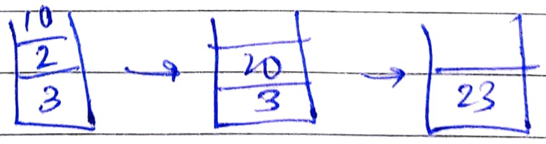
Appointment

Infix to Prefix (efficient)

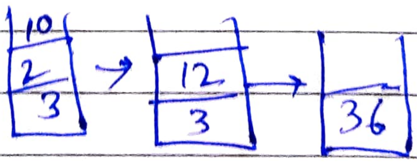
① Create an empty stack st.
② Create an empty string, prefix
③ Do the following for every character x from right to left
④ If x is:
a) Operand : Push it to prefix
b) Right parenthesis : Push to st.
c) Left parenthesis : Pop from st ~~ved~~ until right parenthesis is found. Append the popped character to prefix.
d) Operator: If st is empty, push x to st.
Else compare with st top.
(i) Higher Precedence (than st top): Push x to st.
(ii) Lower precedence: Pop st top and append the popped item to prefix until a higher precedence operator is found (or st becomes empty). Push x to st.
(iii) Equal precedence : Use Associativity.
⑤ Pop everything from st and append to prefix
⑥ Return reverse of prefix.

Evaluation of Prefix

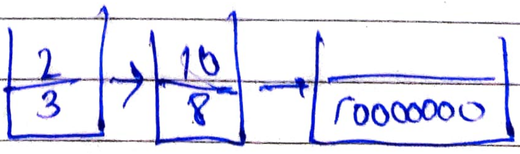
I/P : + * 10 2 3
O/P : 23



I/P: * + 10 2 3
O/P: 36



I/P : ^ 10 ^ 2 3
O/P: 100000000


2
3
10
8
10000000