Stack 2

Monday, December 16, 2024 10:09 AM

1st) $2 * 3/2 \rightarrow 3$; $2 * 3$

Integer division

2nd) $5/2 + 3 \rightarrow 3$

3rd) $(10+8) * 3/4 \rightarrow$ 18 / 4 / 8

$(18) * 3/4 \rightarrow$

$18/4 \rightarrow 10$

Precedence

( ) → parenthesis

^ → Right Associativity

* / → [Left → Right Associativity]

+ - → [Left → Right Associativity]

Eg: $10/5 + 2 * 5/5 - 4$

$10/5 + 2 * 5/5 - 4$

$\rightarrow 3 + 10/5 - 4 \rightarrow 9$

Traversing an expression from L-R, which one comes first, is it given from priority

Final ans:

$((10+8)/2)^3 + 4 * 2/4$

operators: ( ) ^ * / + -

operands: A B C D E

→ Expression

Infix | Prefix | Postfix

(A+B) | +AB | AB+

Infix

Note: We don't have any ( ) in postfix & prefix?

$(10+8) = 18$

→ $(10+8)/2)^3 + 4 * 2/4$

$18/2 + 4 * 2/4$

$9 + 4 * 2/4$

$9 + 8/4$

$9 + 2$

11

Curio: → Infix, Prefix expression longer than this?

→ $O(n)$ time

[Time No. of operands]

Infix to Postfix : → TC: $O(n)$ SC: $O(n)$

Intuition on fix

A B C D E

1) If we get an operand → Stack add to Postfix

2) If we get an '(' string and → Stack.

3) If we get an ')', pop elements from stack until you get the closer '(' from stack. Pop '(' and add them to postfix. Discard the '(' from stack.

4) If we get an operator string with to in stack.

→ Goal: Above can be done, if operator we are about to push has a higher precedence than operator at top of stack.

Else: Top element from top of stack add it to Postfix. Once again check the condition

Return) and (operand

Note: We can keep any operator on top of precedence

Note: If stack is not empty and all remaining leftover things to Postfix

Eg:

Step: we want to) Till we find Stack

Eg2: we want to ^

Eg3: we want to (and) to append

Eg4: we want to) and) to append

Eg5: → we want to) and) to append

Infix

$a + b * c + (d - e) ^ (f + g * h) - i$

a

+

b

*

(

c

^

d

-

e

)

^

(

f

+

g

*

h

)

-

i

Stack | Postfix

a | a

+ | ab

+ | ab

+ * (| abc

+ * (| abc

+ * (^ | abcd

+ * (^ | abcd

[pop ^] | abcd^

+ * (- | abcd^e

+ * (- | abcd^e

[pop till (or get ( ] + * | abcd^e-

+ * ^ | abcd^e-

+ * ^ (| abcd^e-

+ * ^ (+ | abcd^e-f

+ * ^ (+ * | abcd^e-fg

+ * ^ (+ * | abcd^e-fgh

+ * ^ | abcd^e-fgh*+

+ | abcd^e-fgh*+^*+

Infix: $(A/(B-C)^D+E)$

Stack | Postfix

(| A

A | A

/ | A

(| A

B | AB

- | AB

C | ABC

) | ABC-

Used | ABC-/

* | ABC-/D

D | ABC-/D*

+ | ABC-/D*E

E | ABC-/D*E+

) | ABC-/D*E+

Solving postfix : SC: $O(n)$ TC: $O(n)$

→ $(10/(4-2)^6 + 7) =$ 10 4 2 - / 6 ^ 7 + =

1st operand, 2nd operand, 1st oper, 2nd oper, 1st operand

Algo

1st → $(4-2)$ 2nd → $10/2$ 3rd → 5^6 4th → $20+7$

1) If operand Insert to Stack

2) If operator, [pop top 2 elements & stack, perform calculation based on operator, push result inside stack once again]

1st → $(4-2)$

2nd → $10/2$

3rd → 5^6

4th → $20+7$

$((10 + ((5^2)/(3-1))))$ = 10 5 2 ^ 3 1 - / + =

15

2nd oper, 1st oper, 1st operation which we perform, 4th operand

$2^5 = 10$ and 10

→ 1, 3 - 1 = 2

/ 2, 10 = 10/2 = 5

+ 5, 10 = 10+5 = 15

Assume we get ( ) → we pop ^ → we pop *

we get (^ *) before / in

L-R

postfix → we perform (^ *) calculate before (/)

* / , But in actual Postfix * is coming from, That means ^ has to be given better priority

Scenario II → Lotsy Pop =

) → 2 Times we pop

* - → In your postfix * - will come before ^

Since These operations come in ( ) That means we have to perform them first.

Lotsy C

Scenario III

C → We should directly Insert

→ Create a new function to stack

Insert back(n) :

Insert back(11) : Modified Stack

Insert back(): → Using another Stack → Pick all elements to dummy stack → Push(x) to original Stack → Pick all elements to Original Stack

TC: $O(n)$ SC: $O(n)$; Iterative approach

Call by reference

void Insert back(Stack&st, int x)

if (st.size()==0) { st.push(x) return }

int y = st.top();

st.pop() → ①

// st is a stack

// So this st we need to insert x to back.

Insert back (st, x) → ②

st.push(y) → ③

1) top, size, pop, push & Empty()

2) Steps: Return Stack st & x

→ Call Front x to back + st

3) Assumption : Insert x at back of stack

4) Main logic :

5) Base Condition : Empty stack

Insert to

Trace

rec( ) {

variable → S(1)

rec()

return

10 recursion calls :

10 returns at single stack

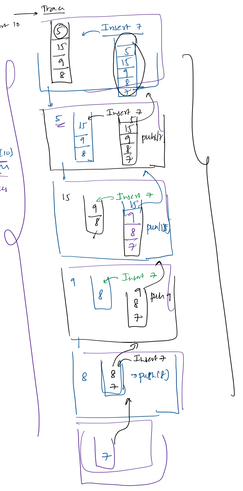
rec() {

rec()

returns S(10)

return

10 recursion calls



Infixy

$A + B^C/(E-F)$

Postfix

$A + B^C/(E-F)$

→ 1st → $(E-F)$

2nd → B^C

3rd → $B^C/E-F$

4th → $A + B^C/E-F$

Stack | Postfix

A B C ^ E F - / +

→ 1st : B^C

→ 2nd : E - F

→ 3rd step : $B^C/E-F$

→ 4th step : $B^C/E-F+A$

A/B

→ First we get A

→ Lor get B

A/B