**CD LAB 12 – POSTFIX & PREFIX NOTATION**

**Aman Kalla**

**RA1911003010640**

**CODE:**

OPERATORS = set(['+', '-', '\*', '/', '(', ')'])

PRI = {'+': 1, '-': 1, '\*': 2, '/': 2}

### INFIX ===> POSTFIX ###

def infix\_to\_postfix(formula):

stack = [] # only pop when the coming op has priority

output = ''

for ch in formula:

if ch not in OPERATORS:

output += ch

elif ch == '(':

stack.append('(')

elif ch == ')':

while stack and stack[-1] != '(':

output += stack.pop()

stack.pop() # pop '('

else:

while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:

output += stack.pop()

stack.append(ch)

# leftover

while stack:

output += stack.pop()

print(f'POSTFIX: {output}')

return output

### INFIX ===> PREFIX ###

def infix\_to\_prefix(formula):

op\_stack = []

exp\_stack = []

for ch in formula:

if not ch in OPERATORS:

exp\_stack.append(ch)

elif ch == '(':

op\_stack.append(ch)

elif ch == ')':

while op\_stack[-1] != '(':

op = op\_stack.pop()

a = exp\_stack.pop()

b = exp\_stack.pop()

exp\_stack.append(op + b + a)

op\_stack.pop() # pop '('

else:

while op\_stack and op\_stack[-1] != '(' and PRI[ch] <= PRI[op\_stack[-1]]:

op = op\_stack.pop()

a = exp\_stack.pop()

b = exp\_stack.pop()

exp\_stack.append(op + b + a)

op\_stack.append(ch)

# leftover

while op\_stack:

op = op\_stack.pop()

a = exp\_stack.pop()

b = exp\_stack.pop()

exp\_stack.append(op + b + a)

print(f'PREFIX: {exp\_stack[-1]}')

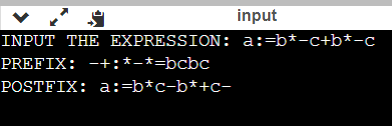
return exp\_stack[-1]

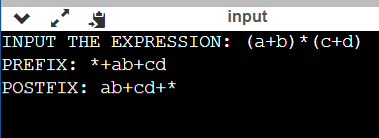
expres = input("INPUT THE EXPRESSION: ")

pre = infix\_to\_prefix(expres)

pos = infix\_to\_postfix(expres)

**OUTPUT:**





**RESULT:** Hence, the implementation of Postfix & Infix Notation is successfully done.