

18CSC304J/ Compiler Design

Submitted By:- ANANNYA P. NEOG (RA1911003010367)

Exp-10: Intermediate Code Generation - Postfix, Prefix

Aim:- To write code for Intermediate Code Generation - Postfix, Prefix

Codes:-

```
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)
```

```
main.py
1 OPERATORS = set(['+', '-', '*', '/', '(', ')'])
2
3 PRI = {'+': 1, '-': 1, '*': 2, '/': 2}
4
5 # INFIX -> POSTFIX
6
7 def infix_to_postfix(formula):
8     stack = [] # only pop when the coming op has priority
9
10    output = ''
11
12    for ch in formula:
13
14        if ch not in OPERATORS:
15
16            output += ch
17
18        elif ch == '(':
19
20            stack.append('(')
21
22        elif ch == ')':
23
24            while stack and stack[-1] != '(':
25                output += stack.pop()
26
27            stack.pop() # pop '('
28
29        else:
30
31            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
32                output += stack.pop()
33
34            stack.append(ch)
35
36        # Leftover
37
```

```
main.py
37
38 while stack:
39     output += stack.pop()
40
41 print(f'POSTFIX: {output}')
42
43 return output
44
45 # INFIX -> PREFIX
46
47 def infix_to_prefix(formula):
48     op_stack = []
49
50     exp_stack = []
51
52     for ch in formula:
53
54         if not ch in OPERATORS:
55
56             exp_stack.append(ch)
57
58         elif ch == '(':
59
60             op_stack.append(ch)
61
62         elif ch == ')':
63
64             while op_stack[-1] != '(':
65                 op = op_stack.pop()
66
67                 a = exp_stack.pop()
68
69                 b = exp_stack.pop()
70
71                 exp_stack.append(op + b + a)
72
73             op_stack.pop() # pop '('
74
```

```
main.py
74
75     else:
76
77         while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
78             op = op_stack.pop()
79
80             a = exp_stack.pop()
81             b = exp_stack.pop()
82
83             exp_stack.append(op + b + a)
84
85             op_stack.append(ch)
86
87         # Leftover
88
89
90 while op_stack:
91     op = op_stack.pop()
92
93     a = exp_stack.pop()
94     b = exp_stack.pop()
95
96     exp_stack.append(op + b + a)
97
98 print(f'PREFIX: {exp_stack[-1]}')
99
100 return exp_stack[-1]
101
102 expres = input("INPUT THE EXPRESSION: ")
103
104 pre = infix_to_prefix(expres)
105
106 pos = infix_to_postfix(expres)
```

Output:-

i) When input is:

(A+B)*(C-D)

as shown in the following output

```
INPUT THE EXPRESSION: (A+B) * (C-D)
PREFIX: *+AB-CD
POSTFIX: AB+CD-*

...Program finished with exit code 0
Press ENTER to exit console.
```

i) When input is:

A+B^C/R

as shown in the following output

```
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+

...Program finished with exit code 0
Press ENTER to exit console.
```