EXP 10: Intermediate code generation – Postfix, Prefix

AIM:

To perform Intermediate code generation – Postfix, Prefix using python.

Algorithm:

1. Scan the infix expression from left to right.

2. If the scanned character is an operand, output it.

3. Else,

1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a ‘(‘ ), push it.

2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an ‘(‘, push it to the stack.

5. If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ is encountered, and discard both the parenthesis.

6. Repeat steps 2-6 until infix expression is scanned.

7. Print the output

8. Pop and output from the stack until it is not empty.

Source 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]  
  
  
### THREE ADDRESS CODE GENERATION ###  
*def* generate3AC(*pos*):  
 print("### THREE ADDRESS CODE GENERATION ###")  
 exp\_stack = []  
 t = 1  
  
 *for* i *in pos*:  
 *if* i *not in* OPERATORS:  
 exp\_stack.append(i)  
 *else*:  
 print(f't{t} := {exp\_stack[-2]} {i} {exp\_stack[-1]}')  
 exp\_stack = exp\_stack[:-2]  
 exp\_stack.append(f't{t}')  
 t += 1  
  
  
expres = input("INPUT THE EXPRESSION: ")  
pre = infix\_to\_prefix(expres)  
pos = infix\_to\_postfix(expres)  
generate3AC(pos)

# OUTPUT:

C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe "F:/Python/DAA/Compiler Design/predictive-parser-master/ifxtopoandpre.py"

INPUT THE EXPRESSION: A\*(B+C)/D

PREFIX: /\*A+BCD

POSTFIX: ABC+\*D/

### THREE ADDRESS CODE GENERATION ###

t1 := B + C

t2 := A \* t1

t3 := t2 / D

Process finished with exit code 0

Result:

Hence the Intermediate code generation – Postfix, Prefix is performed successfully.