20/03/23

EXPERIMENT-10

INTERMEDIATE CODE GENERATION - POSTFIX, PREFIX

AIM: A program to implement code generation - Postfix, Prefix.

ALGORITHM:

1. Declare set of operators
2. Initialize an empty stack
3. To convert INFIX to POSTFIX follow following steps.
4. Scan infix expression from left to right
5. If scanned character is an operand, output it.
6. Else, if precedence of scanned operator is greater than precedence of operator in stack push it.
7. Else Pop all operators from stack which are greater than or equal to in precedence.
8. Push scaned operator into stack. If scanned operator is an '(' push it to stack.
9. If scanned character is ')' pop stack and output it until a '(' is encountered and discard both paranthesis.
10. Pop and output from stack until it is not empty.
11. To convert INFIX to PREFIX follow below steps:
12. first reverse infix expression given
13. Scan exp from left to right
14. Whenever operands arrive print them.
15. If operator arrives, stack found empty, simply push operator into stack.
16. Repeat 6 to 9 steps.

PROGRAM:

```
//Infix to postfix
#include <stdio.h>
# include <stdlib.h>
# include <ctype.h>
# include <string.h>
# define SIZE 100
char stack [SIZE];
  int top = -1;
x
OPERATORS = set(['+','-','*','/','(',')'])
PRI = {'+':1,'-':1,'*':2,'/':2}
# INFIX => POSTFIX #
def infix_to_postfix (formula):
 stack = [] # only pop when coming up has
 priority
   output = ''
  for ch not in OPERATORS:
     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()
```

Output

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

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

O/p Verified

[illegible] 11/4/23

RESULT: Hence code generation for Prefix and Post fix is run succesfully.

[illegible]

```
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()
#leftover
while op-stack:
  exp-stack.append (op+b+a)
 print( f'PREFIX': {exp-stack [-1]}')
  return exp-stack [-1]
express = input ("INPUT THE EXPRESSION: ")
pre =infix_to_prefix (express)
pos = infix_to_postfix (express)
```