RA1811028010049

SHUSHRUT KUMAR

CSE - CC J2

COMPILER DESIGN LAB 10

Intermediate code generation – Postfix, Prefix

Aim: A program to implement Intermediate code generation – Postfix, Prefix.

Algorithm:-

1. Declare set of operators.

2. Initialize an empty stack.

3. To convert INFIX to POSTFIX follow the following steps

4. Scan the infix expression from left to right.

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

6. Else, 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.

7. 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.

8. If the scanned character is an '(', push it to the stack.

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

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

11. To convert INFIX to PREFIX follow the following steps

12. First, reverse the infix expression given in the problem.

13. Scan the expression from left to right.

14. Whenever the operands arrive, print them.

15. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.

16. Repeat steps 6 to 9 until the stack is empty

Program:

```python
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:
```

```python
            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 == '(':
```

```python
            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:
```

```python
        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)
```
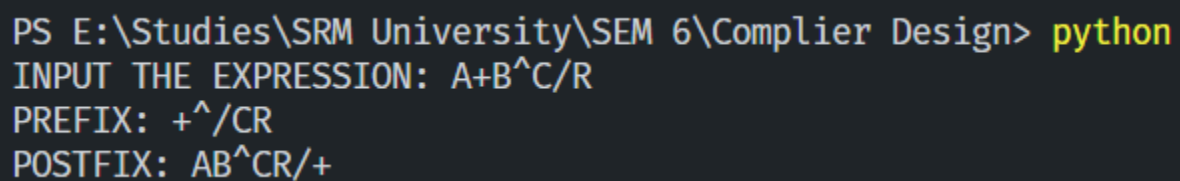
Output:-

```
PS E:\Studies\SRM University\SEM 6\Complier Design> python
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+
```

Result:-

The program was successfully compiled and run.