8/4/2022

Experiment 10 : Intermediate Code Generation using Prefix and Postfix.

Aim : A program to implement intermediate code generation using prefix and postfix.

Algorithm :

1. Declare set of operators
2. Initialize an empty stack
3. To convert Infix to Postfix :

i) Scan input from left to right

ii) If the scanned character is an operand, output it.

iii) Else, if the precedence of the scanned operator is greater than the precedence of the operator in the stack, push it.

iv) Else, pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After that, push the scanned operator to the stack.

v) If the scanned character is an '(', push to stack.

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

vii) Pop and output from stack until it is not empty.

4. To convert Infix to Prefix.

i) Reverse infix expression given in problem

ii) Scan expression from left to right.

iii) Whenever the operands arrive, print them

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

v) Repeat steps [illegible] until stack is empty.

Manual Working.

Input : A + B ^ C / R

Output:

Prefix : + ^ / C R

Postfix : A B ^ C R / +

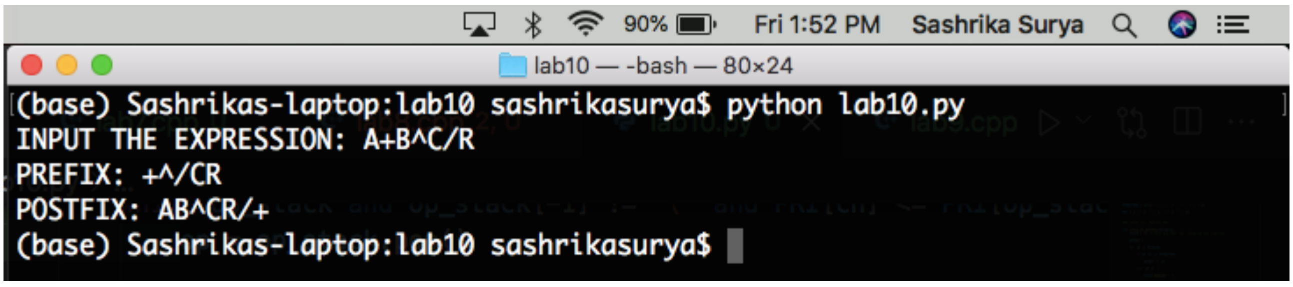
Sashrika Surya
RA1911027010092
Section N2
Date: 8/4/2022

Lab10: Representation of Intermediate Code - infix, prefix and postfix

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)

    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:



```
(base) Sashrikas-laptop:lab10 sashrikasurya$ python lab10.py
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+
(base) Sashrikas-laptop:lab10 sashrikasurya$
```

Result:
Hence, Intermediate Code was generated and infix was converted to prefix and postfix.