INTERMEDIATE CODE GENERATION – QUADRUPLE, TRIPLE, INDIRECT TRIPLE

EX. NO. 11

YUVRAJ SINGH CHAUHAN

RA1911027010058

AIM: To write a program for Immediate Code Generation – Quadruple, Triple, Indirect Triple.

ALGORITHM:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction MOV y' , L to place a copy of y in L.
3. Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

PROGRAM:

```
OPERATORS = set(['+', '-', '*', '/', '(', ')'])
PRI = {'+':1, '-':1, '*':2, '/':2}


def infix_to_postfix(formula):
    stack = []
    output = ''
    for ch in formula:
        if ch not in OPERATORS:
            output += ch
```

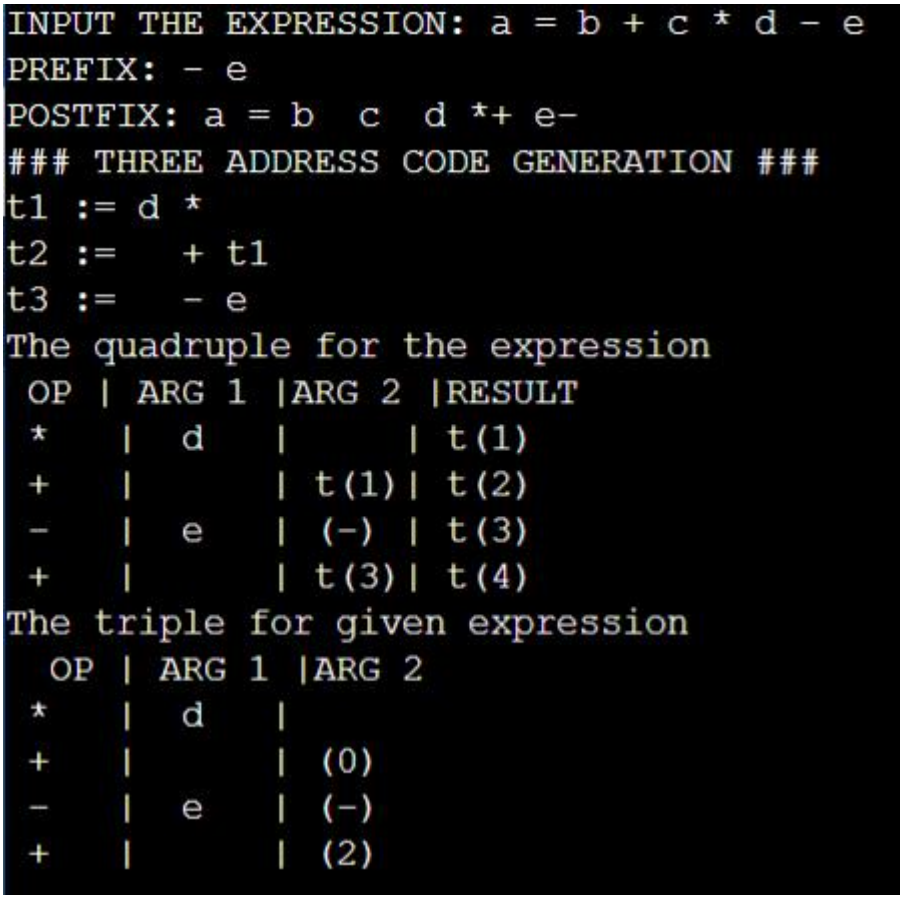
```
        elif ch == '(':
            stack.append('(')
        elif ch == ')':
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop()
        else:
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
                output += stack.pop()
            stack.append(ch)
    while stack:
        output += stack.pop()
    print(f'POSTFIX: {output}')
    return output

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()
        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]

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)
def Quadruple(pos):
  stack = []
```

```
  op = []
  x = 1
  for i in pos:
    if i not in OPERATORS:
       stack.append(i)
    elif i == '-':
        op1 = stack.pop()
        stack.append("t(%s)" %x)
        print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op1,"(-)"," t(%s)" %x))
        x = x+1
        if stack != []:
          op2 = stack.pop()
          op1 = stack.pop()
          print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format("+",op1,op2," t(%s)" %x))
          stack.append("t(%s)" %x)
          x = x+1
    elif i == '=':
      op2 = stack.pop()
      op1 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op2,"(-)",op1))
    else:
      op1 = stack.pop()
      op2 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op2,op1," t(%s)" %x))
      stack.append("t(%s)" %x)
      x = x+1
print("The quadruple for the expression ")
print(" OP | ARG 1 |ARG 2 |RESULT  ")
Quadruple(pos)

def Triple(pos):
        stack = []
        op = []
```

```
    x = 0
    for i in pos:
      if i not in OPERATORS:
        stack.append(i)
      elif i == '-':
        op1 = stack.pop()
        stack.append("(%s)" %x)
        print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op1,"(-)"))
        x = x+1
        if stack != []:
          op2 = stack.pop()
          op1 = stack.pop()
          print("{0:^4s} | {1:^4s} | {2:^4s}".format("+",op1,op2))
          stack.append("(%s)" %x)
          x = x+1
      elif i == '=':
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op1,op2))
      else:
        op1 = stack.pop()
        if stack != []:
          op2 = stack.pop()
          print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op2,op1))
          stack.append("(%s)" %x)
          x = x+1
print("The triple for given expression")
print("  OP | ARG 1 |ARG 2  ")
Triple(pos)
```

OUTPUT :



RESULT :

Immediate Code Generation – Quadruple, Triple, Indirect Triple was implemented successfully using python language.