# Implementation of Intermediate code generation – Quadruple, Triple, Indirect triple

# AIM: To Implement the Intermediate code generation – Quadruple, Triple, Indirect triple.

**Algorithm:**

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'.
3. 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.
4. 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.
5. 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 descriptor.
6. 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.

**Code:-**

import random

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

def Quadruple(pos):

    print('\nQuadruple Representation\n')

    stack = []

    op = []

    x = 1

    print(' OP  | ARG1 | ARG2 | Result')

    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

def Triple(pos):

    print('Triple Representation\n')

    print('        OP  | ARG1 | ARG2')

    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} | {3:^4s}".format("(%s)" %x,i,op1,"(-)"))

            x = x+1

            if stack != []:

                op2 = stack.pop()

                op1 = stack.pop()

                print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,"+",op1,op2))

                stack.append("(%s)" %x)

                x = x+1

        elif i == '=':

            op2 = stack.pop()

            op1 = stack.pop()

            print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,i,op1,op2))

        else:

            op1 = stack.pop()

            if stack != []:

                op2 = stack.pop()

                print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,i,op2,op1))

                stack.append("(%s)" %x)

                x = x+1

def intrp(pos):

    print('Indirect Triple Representation\n')

    print('               OP  | ARG1 | ARG2')

    print

    stack = []

    op = []

    x = random.randrange(30,40)

    y = 0

    for i in pos:

        if i not in OPERATORS:

            stack.append(i)

        elif i == '-':

            op1 = stack.pop()

            stack.append("(%s)" %y)

            print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op1,"(-)"))

            x = x+1

            y = y+1

            if stack != []:

                op2 = stack.pop()

                op1 = stack.pop()

                print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,"+",op1,op2))

                stack.append("(%s)" %y)

                x = x+1

                y = y+1

            elif i == '=':

                op2 = stack.pop()

                op1 = stack.pop()

                print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op1,op2))

        else:

            op1 = stack.pop()

            if stack != []:

                op2 = stack.pop()

                print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op2,op1))

                stack.append("(%s)" %y)

                x = x+1

                y = y+1

expres = input("INPUT THE EXPRESSION: ")

pos = infix\_to\_postfix(expres)

Quadruple(pos)

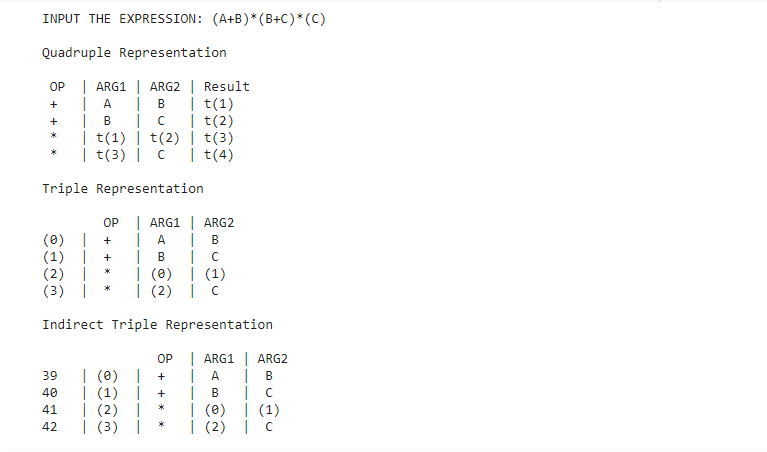
print()

Triple(pos)

print()

intrp(pos)

**OUTPUT:-**

****

**Result:** Intermediate code generation has been implemented successfully and we got Quadruple, Triple, Indirect triple as an output.