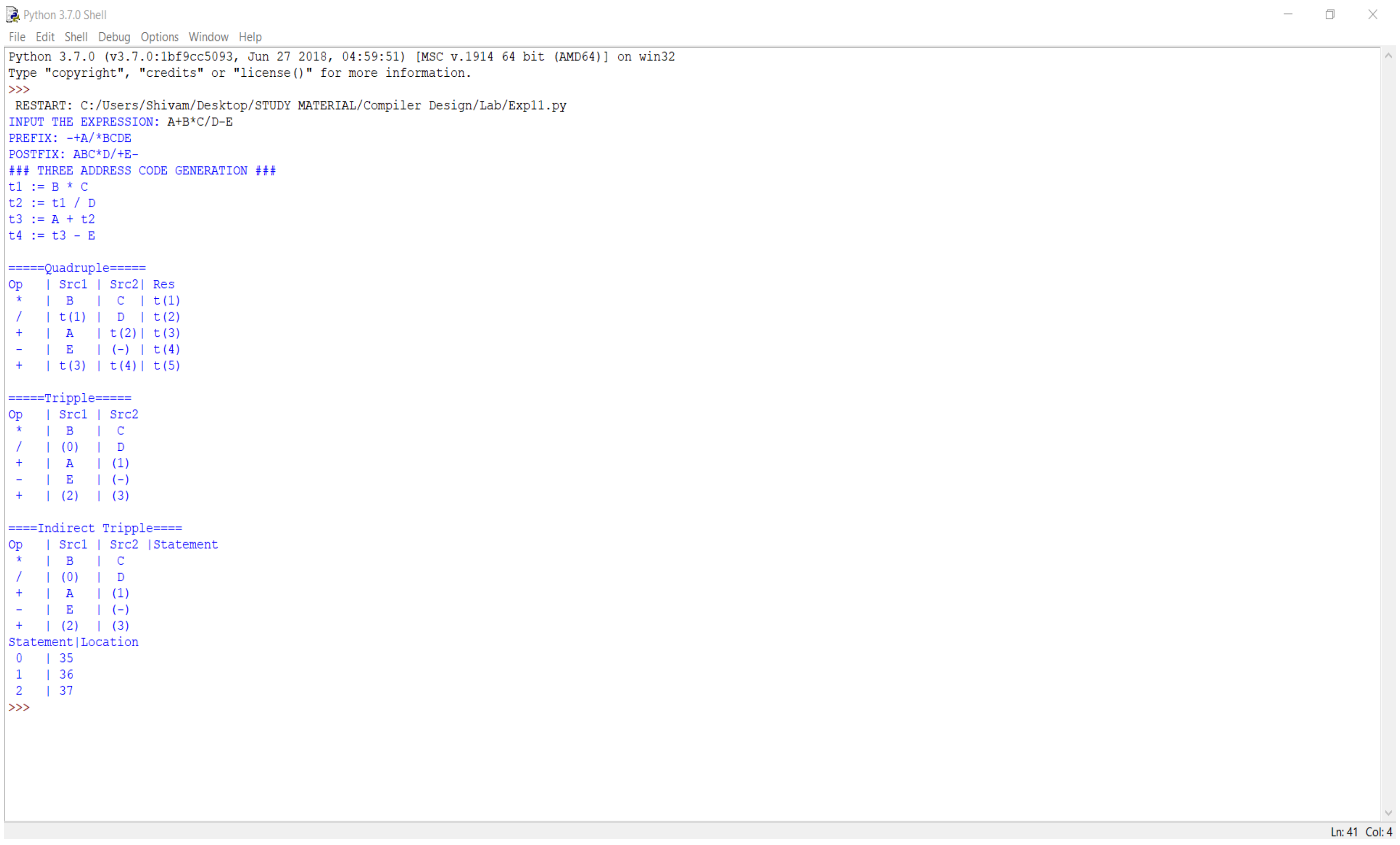
## Python CODE

```
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() # 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() # 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

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

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

def indirectTriple(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
            c=x
      z=35
      print("Statement|Location")
      for i in range(0, c):
          print("{0:^4d} |{1:^4d}".format(i,z))
          z=z+1


expres = input("INPUT THE EXPRESSION: ")
pre = infix_to_prefix(expres)
pos = infix_to_postfix(expres)
generate3AC(pos)
print("\n=====Quadruple=====")
print("Op   | Src1 | Src2| Res")
Quadruple(pos)
print("\n=====Tripple=====")
print("Op   | Src1 | Src2")
Triple(pos)
print("\n====Indirect Tripple====")
print("Op   | Src1 | Src2 |Statement")
indirectTriple(pos)
```

------------------------------------------------------------------------------------------------

## IMPLEMENTATION



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/Shivam/Desktop/STUDY MATERIAL/Compiler Design/Lab/Exp11.py
INPUT THE EXPRESSION: A+B*C/D-E
PREFIX: -+A/*BCDE
POSTFIX: ABC*D/+E-
### THREE ADDRESS CODE GENERATION ###
t1 := B * C
t2 := t1 / D
t3 := A + t2
t4 := t3 - E

=====Quadruple=====
Op    | Src1 | Src2| Res
 *    |  B   |  C  | t(1)
 /    | t(1) |  D  | t(2)
 +    |  A   | t(2)| t(3)
 -    |  E   | (-) | t(4)
 +    | t(3) | t(4)| t(5)

=====Tripple=====
Op    | Src1 | Src2
 *    |  B   |  C
 /    | (0)  |  D
 +    |  A   | (1)
 -    |  E   | (-)
 +    | (2)  | (3)

====Indirect Tripple====
Op    | Src1 | Src2 |Statement
 *    |  B   |  C
 /    | (0)  |  D
 +    |  A   | (1)
 -    |  E   | (-)
 +    | (2)  | (3)
Statement|Location
 0    | 35
 1    | 36
 2    | 37
>>>
```

## RESULT

Code was successfully implemented and the output was verified.