(A)12.Write a program to generate three address code for the given simple expression.

INPUT

w = u\*u - u\*v+ v\*v  
  
  
import re

def generate\_tac(expression):

# Regex to tokenize the identifiers, numbers, and operators

tokens = re.findall(r'[a-zA-Z]+|\d+|\\*|\+|\-|/', expression)

# A stack to keep operands

stack = []

# A counter to generate temporary variable names

temp\_count = 1

# Operator precedence

precedence = {'+': 1, '-': 1, '\*': 2, '/': 2}

# Output list for TAC

output = []

# Helper function to compare operator precedence

def greater\_precedence(op1, op2):

return precedence[op1] > precedence[op2]

# Converting infix expression to postfix using the shunting yard algorithm

postfix = []

ops\_stack = []

for token in tokens:

if token.isalnum(): # If the token is an operand

postfix.append(token)

elif token == '(': # If the token is a left parenthesis

ops\_stack.append(token)

elif token == ')': # If the token is a right parenthesis

while ops\_stack and ops\_stack[-1] != '(':

postfix.append(ops\_stack.pop())

ops\_stack.pop() # Pop the left parenthesis

else: # The token is an operator

while (ops\_stack and ops\_stack[-1] != '(' and

greater\_precedence(ops\_stack[-1], token)):

postfix.append(ops\_stack.pop())

ops\_stack.append(token)

while ops\_stack:

postfix.append(ops\_stack.pop())

# Generate TAC from postfix expression

for token in postfix:

if token.isalnum(): # If the token is an operand, push onto stack

stack.append(token)

else: # The token is an operator

op2 = stack.pop()

op1 = stack.pop()

temp\_var = f't{temp\_count}'

temp\_count += 1

output.append(f'{temp\_var} = {op1} {token} {op2}')

stack.append(temp\_var)

return output

# Example expressions

expressions = [

"w = u\*u - u\*v + v\*v",

"y = x\*x + w - v / r + r",

"t = o\*a - o\*b + o\*c",

"t = j / k - y / u - i",

"a = m \* n - o - p / q",

"a = f ^ r - u \* f \* t - p",

"a = ( b\*b + c\*c ) \* (p - q - r)"

]

# Generate and print TAC for each expression

for i, expression in enumerate(expressions):

tac = generate\_tac(expression)

print(f"({i+1}) {expression}")

for code in tac:

print(code)

print("+" \* 40)