```
import re

def generate_three_address_code(expression):
    # Tokenize the expression
    tokens = re.findall(r'\b\w+\b|[+\-*/()]', expression)

    # Define precedence for operators
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2}

    # Initialize stack for operators and operands
    operator_stack = []
    operand_stack = []

    # Function to generate new temporary variable name
    def new_temp():
        nonlocal temp_counter
        temp_counter += 1
        return f't{temp_counter}'

    # Initialize temporary variable counter
    temp_counter = 0

    # Process each token
    for token in tokens:
        if token.isalnum():
            operand_stack.append(token)
        elif token == '(':
            operator_stack.append(token)
        elif token == ')':
            while operator_stack[-1] != '(':
                operator = operator_stack.pop()
                operand2 = operand_stack.pop()
                operand1 = operand_stack.pop()
                temp_var = new_temp()
                operand_stack.append(temp_var)
                print(f"{temp_var} = {operand1} {operator} {operand2}")
            operator_stack.pop()  # Remove '('
        else:  # Token is an operator
            while (operator_stack and operator_stack[-1] != '(' and
                   precedence.get(operator_stack[-1], 0) >= precedence.get(token, 0)):
                operator = operator_stack.pop()
                operand2 = operand_stack.pop()
                operand1 = operand_stack.pop()
                temp_var = new_temp()
                operand_stack.append(temp_var)
                print(f"{temp_var} = {operand1} {operator} {operand2}")
            operator_stack.append(token)

    # Process remaining operators in the stack
    while operator_stack:
        operator = operator_stack.pop()
        operand2 = operand_stack.pop()
        operand1 = operand_stack.pop()
        temp_var = new_temp()
        operand_stack.append(temp_var)
        print(f"{temp_var} = {operand1} {operator} {operand2}")

# Input expression
expression = "a = ( b*b + c*c ) * (p - q - r)"

# Generate three-address code
generate_three_address_code(expression)
```

```
t1 = b * b
t2 = c * c
t3 = t1 + t2
t4 = p - q
t5 = t4 - r
t6 = t3 * t5
```