**Code: -**

def is\_operand(c):

    return c.isalnum()

def precedence(op):

    if op == '+' or op == '-':

        return 1

    if op == '\*' or op == '/':

        return 2

    return 0

def infix\_to\_postfix(expression):

    stack = []  # Stack to hold operators and parentheses

    result = []  # List to hold the postfix expression

    for char in expression:

        if is\_operand(char):

            result.append(char)  # Append operands (numbers/variables) directly to the result

        elif char == '(':

            stack.append(char)  # Push '(' onto the stack

        elif char == ')':

            while stack and stack[-1] != '(':

                result.append(stack.pop())  # Pop from stack until we encounter '('

            stack.pop()  # Pop '('

        else:  # Operator encountered

            while stack and precedence(stack[-1]) >= precedence(char):

                result.append(stack.pop())  # Pop operators of higher or equal precedence

            stack.append(char)  # Push current operator to stack

    while stack:

        result.append(stack.pop())  # Pop all remaining operators in the stack

    return ''.join(result)

# Example usage

infix\_expression = "A\*(B+C)/D"

postfix\_expression = infix\_to\_postfix(infix\_expression)

print("Infix Expression: ", infix\_expression)

print("Postfix Expression: ", postfix\_expression)

**OUTPUT:**

