Технології програмування

Лабораторна робота №4

**Зворотний польський запис та обчислення математичного виразу**

**Мета роботи**: Використовуючи теоретичне підґрунтя про зворотний польський запис розробити програму на вхід якої подається математичний вираз, що має довільний набір операндів, операторів та дужок, на виході програма обчислює результат математичного виразу.

**Хід роботи**

В ході виконання лабораторної роботи була розроблена програма для обчислення математичних виразів, використовуючи Зворотній Польський Запис (ЗПЗ). Програма пройшла успішні тести та може використовуватися для обчислення складних математичних виразів.

Використання стеку та пріоритетів операцій дозволяє програмі ефективно перетворювати і обчислювати вирази різної складності. Алгоритм ЗПЗ виявляється важливим інструментом в області обчислення математичних виразів, а отримані результати свідчать про правильність його реалізації в даній програмі.

**Текст програми**

Lab04.py

digits = ('1', '2', '3', '4', '5', '6', '7', '8', '9', '0')

operations = ('+', '-', '\*', '/', '^')

def isNumber(unit):

    return all(char in digits for char in unit)

def getOper(operation):

    if operation in ('+', '-'):

        return 0

    elif operation in ('\*', '/'):

        return 1

    elif operation == '^':

        return 2

def infix\_to\_postfix(infix):

    stack = []

    output = []

    for token in infix.split(' '):

        if isNumber(token):

            output.append(token)

        elif token == '(':

            stack.append(token)

        elif token == ')':

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

                output.append(stack.pop())

            stack.pop()

        elif token in operations:

            opPriority = getOper(token)

            while stack and stack[-1] in operations and getOper(stack[-1]) >= opPriority:

                output.append(stack.pop())

            stack.append(token)

    while stack:

        output.append(stack.pop())

    return output

def getRpn(input):

    infix\_expression = input

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

    return postfix\_expression

def calcRpn(rpn):

    stack = []

    for token in rpn:

        if isNumber(token):

            stack.append(token)

        else:

            num2 = float(stack.pop())

            num1 = float(stack.pop())

            result = 0

            if token == '+':

                result = num1 + num2

            elif token == '-':

                result = num1 - num2

            elif token == '\*':

                result = num1 \* num2

            elif token == '/':

                result = num1 / num2

            elif token == '^':

                result = num1 \*\* num2

            stack.append(result)

    return stack

def main():

    input\_expression = '48 / 4 + 156 \* 24 \* ( 40 / 5 ) ^ 4'

    rpn\_expression = getRpn(input\_expression)

    result = calcRpn(rpn\_expression)

    print(f'Infix expression: {input\_expression}')

    print(f'Reverse Polish Notation (RPN): {rpn\_expression}')

    print(f"Result: {result[0]}")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

test\_lab04.py

import unittest

from lab04 import isNumber, getOper, infix\_to\_postfix, getRpn, calcRpn

class TestCalculator(unittest.TestCase):

    def test\_isNumber(self):

        self.assertTrue(isNumber('123'))

        self.assertFalse(isNumber('+'))

        self.assertFalse(isNumber('\*'))

    def test\_getOper(self):

        self.assertEqual(getOper('+'), 0)

        self.assertEqual(getOper('\*'), 1)

        self.assertEqual(getOper('^'), 2)

    def test\_infix\_to\_postfix(self):

        self.assertEqual(infix\_to\_postfix('22 / 11 + 23 \* 2 \* ( 24 / 2 ) ^ 2'), ['22', '11', '/', '23', '2', '\*', '24', '2', '/', '2', '^', '\*', '+'])

        self.assertEqual(infix\_to\_postfix('48 / 4 + 156 \* 24 \* ( 40 / 5 ) ^ 4'), ['48', '4', '/', '156', '24', '\*', '40', '5', '/', '4', '^', '\*', '+'])

    def test\_getRpn(self):

        self.assertEqual(getRpn('22 / 11 + 23 \* 2 \* ( 24 / 2 ) ^ 2'), ['22', '11', '/', '23', '2', '\*', '24', '2', '/', '2', '^', '\*', '+'])

        self.assertEqual(getRpn('48 / 4 + 156 \* 24 \* ( 40 / 5 ) ^ 4'), ['48', '4', '/', '156', '24', '\*', '40', '5', '/', '4', '^', '\*', '+'])

    def test\_calcRpn(self):

        self.assertEqual(calcRpn(['22', '11', '/', '23', '2', '\*', '24', '2', '/', '2', '^', '\*', '+']), [6626.0])

        self.assertEqual(calcRpn(['48', '4', '/', '156', '24', '\*', '40', '5', '/', '4', '^', '\*', '+']), [15335436.0])

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()