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.

**Завдання до лабораторної роботи**

Використовуючи теоретичне відомості розробити програму яка на вхід отримує математичний вираз з довільною кількістю операндів, операторів та дужок. В першу чергу сформувати послідовність символів у ЗПН. На другому етапі виконання лабораторної роботи вирахувати результат послідовності, що була сформована, використовуючи алгоритм запису математичного виразу у ЗПН.

**Хід роботи**

1. Перевіряємо, чи є заданий токен оператором (+, -, \*, / ,^).

def is\_operator(token):

return token in ['+', '-', '\*', '/', '^']

1. infix\_to\_rpn - це функція, яка перетворює інфіксний вираз у зворотньопольський запис. output - це вихідний список для ЗПН, operators - це стек для операторів, а precedence - словник, який визначає пріоритети операторів.

def infix\_to\_rpn(infix\_expression):

    output = []

    operators = []

    precedence = {'+': 1, '-': 1, '\*': 2, '/': 2, '^': 3}

1. Якщо токен є цифрою, додаємо його безпосередньо до виходу.

for token in infix\_expression:

if token.isdigit():

output.append(token)

1. Якщо токен - це відкриваюча дужка, додаємо її до стеку операторів. Якщо токен - це закриваюча дужка, викидаємо оператори зі стеку і додаємо їх до виходу, поки не зустрінемо відкриваючу дужку. Потім видаляємо саму відкриваючу дужку.

elif token == '(':

operators.append(token)

elif token == ')':

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

output.append(operators.pop())

operators.pop()

1. Якщо токен є оператором, викидаємо всі оператори зі стеку, чий пріоритет вищий або рівний пріоритету поточного оператора, і додаємо їх до виходу. Потім додаємо поточний оператор до стеку.

elif is\_operator(token):

            while operators and is\_operator(operators[-1]) and precedence[operators[-1]] >= precedence[token]:

                output.append(operators.pop())

            operators.append(token)

1. Додаємо залишкові оператори зі стеку до виходу та повертаємо вихідний ЗПН.

while operators:

        output.append(operators.pop())

    return output

1. evaluate\_rpn - це функція, яка обчислює результат ЗПН, використовуючи стек. Якщо токен є цифрою, додаємо її до стеку.

def evaluate\_rpn(rpn\_expression):

    stack = []

    for token in rpn\_expression:

        if token.isdigit():

            stack.append(int(token))

1. Якщо токен є оператором, викидаємо дві останні цифри зі стеку, застосовуємо до них операцію і додаємо результат назад до стеку. Повертаємо єдиний елемент у стеці, який містить результат обчислення виразу.

elif is\_operator(token):

            operand2 = stack.pop()

            operand1 = stack.pop()

            if token == '+':

                stack.append(operand1 + operand2)

            elif token == '-':

                stack.append(operand1 - operand2)

            elif token == '\*':

                stack.append(operand1 \* operand2)

            elif token == '/':

                stack.append(operand1 / operand2)

            elif token == '^':

                stack.append(operand1 \*\* operand2)

    return stack[0]

1. Функція main отримує від користувача інфіксний вираз, перетворює його в ЗПН, обчислює результат та виводить його на екран.

def main():

    infix\_expression = input("Введіть математичний вираз: ").split()

    rpn\_expression = infix\_to\_rpn(infix\_expression)

    result = evaluate\_rpn(rpn\_expression)

    print("Результат обчислення:", result)

1. Код програми повністю:

def is\_operator(token):

    return token in ['+', '-', '\*', '/', '^']

def infix\_to\_rpn(infix\_expression):

    output = []

    operators = []

    precedence = {'+': 1, '-': 1, '\*': 2, '/': 2, '^': 3}

    for token in infix\_expression:

        if token.isdigit():

            output.append(token)

        elif token == '(':

            operators.append(token)

        elif token == ')':

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

                output.append(operators.pop())

            operators.pop()

        elif is\_operator(token):

            while operators and is\_operator(operators[-1]) and precedence[operators[-1]] >= precedence[token]:

                output.append(operators.pop())

            operators.append(token)

    while operators:

        output.append(operators.pop())

    return output

def evaluate\_rpn(rpn\_expression):

    stack = []

    for token in rpn\_expression:

        if token.isdigit():

            stack.append(int(token))

        elif is\_operator(token):

            operand2 = stack.pop()

            operand1 = stack.pop()

            if token == '+':

                stack.append(operand1 + operand2)

            elif token == '-':

                stack.append(operand1 - operand2)

            elif token == '\*':

                stack.append(operand1 \* operand2)

            elif token == '/':

                stack.append(operand1 / operand2)

            elif token == '^':

                stack.append(operand1 \*\* operand2)

    return stack[0]

def main():

    infix\_expression = input("Введіть математичний вираз: ").split()

    rpn\_expression = infix\_to\_rpn(infix\_expression)

    result = evaluate\_rpn(rpn\_expression)

    print("Результат обчислення:", result)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. Приклад виконання програми:

****

1. Написала тестування цього коду:

self.assertEqual(infix\_to\_rpn(['3', '+', '4']), ['3', '4', '+']) - викликає метод assertEqual, який перевіряє, чи фактичний результат перетворення інфіксного виразу ['3', '+', '4'] дорівнює очікуваному результату ['3', '4', '+']. Якщо вони не рівні, тест завершиться з помилкою. Аналогічно, два інші рядки також перевіряють правильність перетворень для інших вхідних інфіксних виразів.

self.assertEqual(evaluate\_rpn(['3', '4', '+']), 7) - викликає метод assertEqual, який перевіряє, чи фактичний результат обчислення виразу ['3', '4', '+'] дорівнює очікуваному результату 7. Якщо вони не рівні, тест завершиться з помилкою. Аналогічно, два інші рядки.

infix\_expression = ['3', '+', '4'] - створюється інфіксний вираз, який буде використовуватися для тестування.

rpn\_expression = infix\_to\_rpn(infix\_expression) - викликається функція infix\_to\_rpn, щоб отримати вираз у зворотньому польському записі (ОПН).

result = evaluate\_rpn(rpn\_expression) - викликається функція evaluate\_rpn для обчислення результату з ОПН виразу.

self.assertEqual(result, 7) - порівняння отриманого результату з очікуваним результатом. Якщо фактичний результат і очікуваний не збігаються, тест завершиться з помилкою.

import unittest

from lab\_04 import infix\_to\_rpn, evaluate\_rpn

class TestCalculator(unittest.TestCase):

    def test\_infix\_to\_rpn(self):

        self.assertEqual(infix\_to\_rpn(['3', '+', '4']), ['3', '4', '+'])

        self.assertEqual(infix\_to\_rpn(['5', '\*', '(', '2', '+', '3', ')']), ['5', '2', '3', '+', '\*'])

        self.assertEqual(infix\_to\_rpn(['(', '1', '+', '2', ')', '\*', '3']), ['1', '2', '+', '3', '\*'])

    def test\_evaluate\_rpn(self):

        self.assertEqual(evaluate\_rpn(['3', '4', '+']), 7)

        self.assertEqual(evaluate\_rpn(['5', '2', '3', '+', '\*']), 25)

        self.assertEqual(evaluate\_rpn(['1', '2', '+', '3', '\*']), 9)

    def test\_combined\_functionality(self):

        infix\_expression = ['3', '+', '4']

        rpn\_expression = infix\_to\_rpn(infix\_expression)

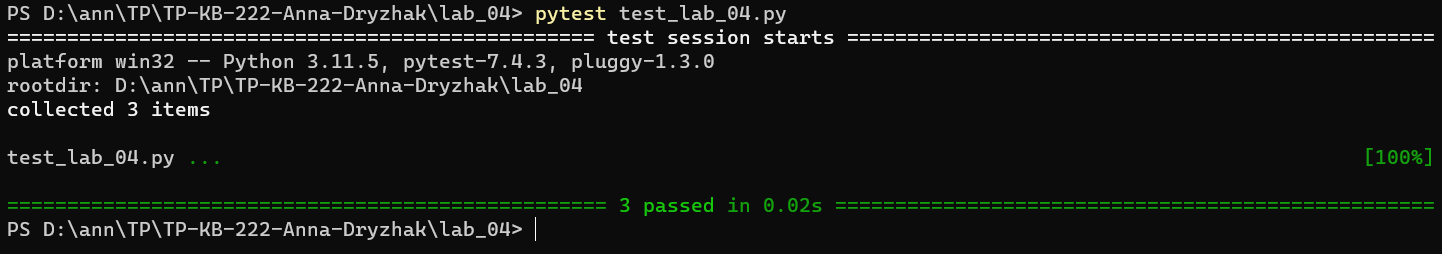
        result = evaluate\_rpn(rpn\_expression)

        self.assertEqual(result, 7)

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

    unittest.main()

1. Результат тестування:



**Висновок.** Під час виконання даної лабораторної роботи я використовуючи теоретичне підґрунтя про зворотний польський запис розробила програму на вхід якої подається математичний вираз, що має довільний набір операндів, операторів та дужок, на виході програма обчислює результат математичного виразу.