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.

**Теоретичні відомості**

**Зворотний польський запис**

Звичною формою запису виразів є інфіксна, коли знак бінарної операції записують між позначеннями операндів цієї операції, наприклад, a + b. Розглянемо запис знаків операцій після позначень операндів, тобто постфіксний запис, наприклад, a b +. Такий запис має також назву зворотного польського, бо його запропонував польський логік Ян Лукасевич. Далі словосполучення: «зворотний польський запис» позначатимемо ЗПЗ. Позначення для функції традиційно записують перед аргументами. Природно такий запис назвати префіксним. При описі ЗПЗ переважно обмежуються перетворенням інфіксного запису у ЗПЗ.

Зворотний польський запис має чудові властивості, які перетворюють її на ідеальну проміжну ланку при трансляції коду програми.

Обчислення виразу, записаного в зворотному польському записі, можна проводити шляхом однократного перегляду ЗПЗ.

Зворотний польський запис виразу з арифметичними діями та піднесенням до степеня можна отримати, дотримуючись алгоритму, запропонованого Дейкстpою. Алгоритм отримав назву «сортувальна станція», за подібність його операцій із тим, що відбувається на залізничних сортувальних станціях. Як і алгоритм обчислення ЗПЗ, алгоритм сортувальної станції ґрунтується на стеку. У перетворенні беруть участь дві текстові змінні: вхідний і вихідний рядки. У процесі перетворення використовується стек, що зберігає ще не додані до вихідного рядка операції. Перетворювальна програма читає вхідний рядок послідовно символ за символом (символ – це не обов'язково буква), виконує на кожному кроці деякі дії залежно від того, який символ було прочитано.

**Алгоритм**

У випадку, коли є символи для обробки необхідно зчитати символ.

Якщо символ є числом або додаємо його до вихідного рядка.

Якщо символ є дужкою, поміщаємо його в стек.

Якщо символ є дужкою, що закривається то доки верхнім елементом стека не стане відкриваюча дужка, виштовхуємо елементи зі стека у вихідний рядок. При цьому дужка, що відкриває, видаляється зі стека, але у вихідний рядок не додається. Якщо стек закінчився раніше, ніж ми зустріли дужку, це означає, що у виразі або неправильно поставлений роздільник, або не узгоджені дужки.

Якщо символ є бінарною операцією та операція на вершині стеку має більший або такий самий пріоритет, то необхідно “виштовхнути” верхній елемент до вихідного рядка. Помістити операцію в стек.

Коли вхідний рядок закінчився, виштовхуємо всі символи зі стека у вихідний рядок

**Пріоритетність операцій:**

Найвищий – вираз в дужках.

Високий – піднесення до степеня.

Середній – множення або ділення.

Низький – додавання або віднімання.

**Приклад:**

|  |
| --- |
| Вхід: 3 + 4 \* 2 / (1 - 5) ^ 2  Читаємо «3»  Додаємо «3» до вихідного рядка  Вихід: 3  Читаємо «+»  Кладемо «+» у стек  Вихід: 3  Стек: +  Читаємо «4»  Додамо «4» до вихідного рядка  Вихід: 3 4  Стек: +  Читаємо «\*»  Кладемо «\*» у стек  Вихід: 3 4  Стек: + \*  Читаємо «2»  Додамо «2» до вихідного рядка  Вихід: 3 4 2  Стек: + \*  Читаємо «/»  Виштовхуємо «\*» зі стека у вихідний рядок, кладемо «/» у стек  Вихід: 3 4 2 \*  Стек: +/  Читаємо «(»  Кладемо «(» у стек  Вихід: 3 4 2 \*  Стек: + / (  Читаємо «1»  Додамо «1» до вихідного рядка  Вихід: 3 4 2 \* 1  Стек: + / (  Читаємо «−»  Кладемо «−» у стек  Вихід: 3 4 2 \* 1  Стек: + / ( −  Читаємо «5»  Додамо «5» до вихідного рядка  Вихід: 3 4 2 \* 1 5  Стек: + / (-  Читаємо «)»  Виштовхуємо «−» зі стека у вихідний рядок, виштовхуємо «(»  Вихід: 3 4 2 \* 1 5 −  Стек: +/  Читаємо «^»  Кладемо «^» у стек  Вихід: 3 4 2 \* 1 5 −  Стек: +/^  Читаємо «2»  Додамо «2» до вихідного рядка  Вихід: 3 4 2 \* 1 5 − 2  Стек: +/^  Кінець виразу  Виштовхуємо всі елементи зі стека в рядок  Вихід: **3 4 2 \* 1 5 − 2 ^ / +** |

**Обчислення виразу**

Використовуючи алгоритм ЗПЗ математичний вираз 3 + 4 \* 2 / (1 - 5) ^ 2 був записаний у вигляді 3 4 2 \* 1 5 − 2 ^ / +

Обчислення проводиться зліва направо. Якщо в запису зустрічається число, то число поміщається в стек. Якщо в запису зустрічається оператор, то він застосовується до двох верхніх елементів стеку які виштовхуються із стеку, а результат виконання поміщається в стек.

Запис інтерпретується як зазначено у наведеній нижче таблиці (зазначено стан стека після виконання операції, вершина стека виділена червоним кольором)

|  |  |  |
| --- | --- | --- |
| Символ | Дія | Стек |
| 3 | помістити в стек | 3 |
| 4 | помістити в стек | 3 4 |
| 2 | помістити в стек | 3 4 2 |
| \* | множення | 3 8 |
| 1 | помістити в стек | 3 8 1 |
| 5 | помістити в стек | 3 8 1 5 |
| - | віднімання | 3 8 -4 |
| 2 | помістити в стек | 3 8 -4 2 |
| ^ | піднесення до степеню | 3 8 16 |
| / | ділення | 3 0.5 |
| + | додавання | 3.5 |

Результат **3.5**, в кінці обчислень знаходиться на вершині стека.

**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.

Текст програми разом зі звітом розмістити в директорії lab\_04. Директорію lab\_04 розмістити в директорії, що використовується для виконання практичних завдань по кожній лекції та має назву **TP-KB-22[1 or 2]-Name-Surname**.

**Хід роботи**

**Перетворення інфіксного виразу у ЗПЗ**

**Опис функції infix\_to\_postfix**

1. def infix\_to\_postfix(expression):
2. precedence = {'+': 1, '-': 1, '\*': 2, '/': 2, '^': 3}  # Пріоритет операторів
3. output = []  # Вихідний рядок (список для ЗПЗ)
4. stack = []   # Стек для операторів
6. for token in expression:
7. if token.isnumeric():  # Якщо символ - число
8. output.append(token)
9. elif token == '(':  # Якщо символ - відкриваюча дужка
10. stack.append(token)
11. elif token == ')':  # Якщо символ - закриваюча дужка
12. while stack and stack[-1] != '(':
13. output.append(stack.pop())
14. stack.pop()  # Видалити відкриваючу дужку зі стека
15. else:  # Якщо символ - оператор
16. while (stack and stack[-1] != '(' and
17. precedence.get(token, 0) <= precedence.get(stack[-1], 0)):
18. output.append(stack.pop())
19. stack.append(token)
21. while stack:  # Виштовхнути всі оператори зі стека до вихідного рядка
22. output.append(stack.pop())
24. return output

**Розбір коду**:

* Визначення пріоритету операторів у словнику precedence.
* Створення двох порожніх списків: output для вихідного рядка (ЗПЗ) і stack для зберігання операторів.
* Перебір кожного символу в вхідному виразі:
  + Якщо символ є числом, додаємо його до output.
  + Якщо символ є відкриваючою дужкою, додаємо його до stack.
  + Якщо символ є закриваючою дужкою, виштовхуємо оператори зі stack до output до тих пір, поки не зустрінемо відкриваючу дужку.
  + Якщо символ є оператором, виштовхуємо оператори зі stack до output відповідно до їх пріоритету, потім додаємо поточний оператор до stack.
* Після обробки всіх символів виштовхуємо залишкові оператори зі stack до output.

**Обчислення виразу у ЗПЗ**

**Опис функції evaluate\_postfix**:

1. def evaluate\_postfix(postfix):
2. stack = []
4. for token in postfix:
5. if token.isnumeric():
6. stack.append(float(token))
7. else:
8. b = stack.pop()
9. a = stack.pop()
10. match token:
11. case '+':
12. stack.append(a + b)
13. case '-':
14. stack.append(a - b)
15. case '\*':
16. stack.append(a \* b)
17. case '/':
18. stack.append(a / b)
19. case '^':
20. stack.append(a \*\* b)
22. return stack[0]

**Розбір коду**:

* Створення порожнього списку stack.
* Перебір кожного символу в ЗПЗ виразі:
  + Якщо символ є числом, перетворюємо його у float і додаємо до stack.
  + Якщо символ є оператором, виймаємо два верхні числа зі стека, виконуємо операцію і додаємо результат назад до стека.
* Повертаємо результат, який залишається на вершині стека.

#### Основний блок виконання

**Опис функції main**:

1. def main():
2. expression = input("Введіть математичний вираз: ")
3. tokens = list(expression.replace(" ", ""))  # Розбиваємо вираз на токени
4. postfix = infix\_to\_postfix(tokens)
5. print("Зворотний польський запис:", " ".join(postfix))
6. result = evaluate\_postfix(postfix)
7. print("Результат обчислення:", result)

**Розбір коду**:

* Зчитування математичного виразу від користувача.
* Видалення пробілів і розбиття виразу на токени.
* Виклик функції infix\_to\_postfix для перетворення виразу у ЗПЗ.
* Виведення отриманого ЗПЗ виразу.
* Виклик функції evaluate\_postfix для обчислення результату.
* Виведення результату обчислення.

**Повний код:**

#Перетворення інфіксного виразу у ЗПЗ

def infix\_to\_postfix(expression):

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

    output = []

    stack = []

    for token in expression:

        if token.isnumeric():

            output.append(token)

        elif token == '(':

            stack.append(token)

        elif token == ')':

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

                output.append(stack.pop())

            stack.pop()

        else:                           # Якщо символ - оператор

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

                   precedence.get(token, 0) <= precedence.get(stack[-1], 0)):

                output.append(stack.pop())

            stack.append(token)

    while stack:

        output.append(stack.pop())

    return output

#Обчислення виразу у ЗПЗ

def evaluate\_postfix(postfix):

    stack = []

    for token in postfix:

        if token.isnumeric():

            stack.append(float(token))

        else:

            b = stack.pop()

            a = stack.pop()

            match token:

                case '+':

                    stack.append(a + b)

                case '-':

                    stack.append(a - b)

                case '\*':

                    stack.append(a \* b)

                case '/':

                    stack.append(a / b)

                case '^':

                    stack.append(a \*\* b)

    return stack[0]

def main():

    expression = input("Введіть математичний вираз: ")

    tokens = list(expression.replace(" ", ""))  # Розбиваємо вираз на токени

    postfix = infix\_to\_postfix(tokens)

    print("Зворотний польський запис:", " ".join(postfix))

    result = evaluate\_postfix(postfix)

    print("Результат обчислення:", result)

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

    main()

**Висновок**

На даній лабораторній роботі я навчився 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