

|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное бюджетное образовательное учреждение высшего образования  **«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ**  **ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»** |  |

**Кафедра «Высшая математика»**

**Лабораторная работа № 2**

по дисциплине

«Структуры и алгоритмы обработки данных»

на тему:

**«**Постфикс/префикс**»**

**Выполнил:**

Учебная группа: 1бПМ   
ФИО: Борисов И. М.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель лабораторной работы:**

Должность: старший преподаватель

Звание: б/з

ФИО: Кутейников И. А.

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

Цель:

Решение программой примеров в префиксном/постфиксном виде, а также их преобразование из обычного вида.

Код:

import math

# Функция для определения приоритета оператора

def precedence(op):

    if op in ('+', '-'):

        return 1

    if op in ('\*', '/'):

        return 2

    if op in ('^'):

        return 3

    if op in ('sin', 'cos', 'tan', 'cot', 'log', 'sqrt'):

        return 4

    return 0

# Проверка, является ли данный символ оператором

def is\_operator(c):

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

# Проверка, является ли данная строка функцией

def is\_function(s):

    return s in ['sin', 'cos', 'tan', 'cot', 'log', 'sqrt']

# Функция для вычисления постфиксного выражения

def evaluate\_postfix(expression):

    stack = []

    tokens = expression.split()

    for token in tokens:

        if token.isdigit():  # если операнд

            stack.append(float(token))

        elif token == '+':

            b, a = stack.pop(), stack.pop()

            stack.append(a + b)

        elif token == '-':

            b, a = stack.pop(), stack.pop()

            stack.append(a - b)

        elif token == '\*':

            b, a = stack.pop(), stack.pop()

            stack.append(a \* b)

        elif token == '/':

            b, a = stack.pop(), stack.pop()

            stack.append(a / b)

        elif token == '^':

            b, a = stack.pop(), stack.pop()

            stack.append(a \*\* b)

        elif token == 'sin':

            a = stack.pop()

            stack.append(math.sin(math.radians(a)))

        elif token == 'cos':

            a = stack.pop()

            stack.append(math.cos(math.radians(a)))

        elif token == 'tan':

            a = stack.pop()

            stack.append(math.tan(math.radians(a)))

        elif token == 'cot':

            a = stack.pop()

            stack.append(1 / math.tan(math.radians(a)))

        elif token == 'log':

            a = stack.pop()

            stack.append(math.log10(a))

        elif token == 'sqrt':

            a = stack.pop()

            stack.append(math.sqrt(a))

    return stack[0] if stack else None

# Функция для преобразования инфиксного выражения в постфиксное

def infix\_to\_postfix(expression):

    stack = []

    output = []

    i = 0

    while i < len(expression):

        if expression[i].isalnum():

            operand = ''

            while i < len(expression) and (expression[i].isalnum() or expression[i] == '.'):

                operand += expression[i]

                i += 1

            output.append(operand)

            i -= 1

        elif i < len(expression) - 2 and expression[i:i+3] in ['sin', 'cos', 'tan', 'cot']:

            stack.append(expression[i:i+3])

            i += 2

        elif i < len(expression) - 3 and expression[i:i+3] == 'log':

            stack.append('log')

            i += 2

        elif i < len(expression) - 4 and expression[i:i+4] == 'sqrt':

            stack.append('sqrt')

            i += 3

        elif expression[i] == '(':

            stack.append(expression[i])

        elif expression[i] == ')':

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

                output.append(stack.pop())

            stack.pop()

        elif is\_operator(expression[i]):

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

                output.append(stack.pop())

            stack.append(expression[i])

        i += 1

    while stack:

        output.append(stack.pop())

    return ' '.join(output)

# Функция для преобразования инфиксного выражения в префиксное

def infix\_to\_prefix(expression):

    expression = expression[::-1]

    expression = expression.replace('(', 'temp').replace(')', '(').replace('temp', ')')

    result = infix\_to\_postfix(expression)

    return result[::-1]

# Основная программа

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

    expression = input("Введите инфиксное выражение: ")

    print("Инфиксное выражение:", expression)

    postfix\_expr = infix\_to\_postfix(expression)

    print("Постфиксное выражение:", postfix\_expr)

    prefix\_expr = infix\_to\_prefix(expression)

    print("Префиксное выражение:", prefix\_expr)

    try:

        result = evaluate\_postfix(postfix\_expr)

        print(f"Результат: {result}")

    except Exception as e:

        print(f"Ошибка при вычислении: {e}")

Результат:

1.Префикс

2.Постфикс

3.Из инфикса в префикс

4.Вычисление обычного выражения

Выберете пункт меню: 3

Введите выражение:

( 9 + 8 - 5 ) \* 8 - 12 + 3

Результат:

9 8 + 5 - 8 \* 12 - 3 +

Выберете пункт меню: 1

Введите выражение:

9 8 + 5 - 8 \* 12 - 3 +

Результат:

87.0

Вывод:

В ходе выполнения данной лабораторной работы были изучены и реализованы методы для работы с выражениями в префиксной и постфиксной формах. Программа успешно демонстрирует возможности преобразования стандартных инфиксных выражений в префиксные и постфиксные формы, что является важным навыком в области компьютерных наук.