МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет по лабораторной работе № 1.

“Постфиксная запись выражений. Вычисление арифметических выражений”

Выполнила: студентка 3 курса

Группы А-05-19

Абросимова М.Н.

Москва
2022

Лабораторная работа 1

Постфиксная запись выражений. Вычисление арифметических выражений

Задание. Разработать алгоритм и реализовать программу для перевода арифметических выражений в постфиксную запись и их вычисление.

Для арифметических выражений допустимы операции: сложения, вычитания, умножения, деления.

Обращаем внимание, что арифметическое выражение может содержать идентификаторы, значения которых задаются при запуске.

Разработанная программа должна выводить:

-постфиксную запись выражения

-вычислять значение выражения по постфиксной записи

-вычислять значение средствами языка программирования.

Метод

Преобразование в постфиксное выражение (обратную польскую запись):

1. Преобразуем изначальное выражение в список, так, чтобы после каждого операнда и оператора присутствовал пробел. Это необходимо, чтобы было проще преобразовывать в список
2. В постфиксной записи, в отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. В инфиксной записи конкретная операция определяется тем, в какой позиции находится знак, поэтому чтобы записать выражение, содержащее унарную операцию, придется ввести для операции изменения знака отдельное обозначение, например,«#»: Тогда выражение 5* (-3 +8) в постфиксной записи примет вид 5 3 # 8 + *
3. Обрабатываем список слева направо.
4. Если операнд, то добавляем в конец итогового списка.
    - 4.1.1. Если левая скобка, добавляем в стек для хранения операторов.
    - 4.1.2. Если правая скобка, то вытащить элемент из стека для хранения операторов. Продолжать до тех пор, пока не найдем соответствующую левую скобку.
    - 4.1.3. Если оператор *, /, +, - , # помещаем в стек для хранения операторов. Перед этим необходимо удалить любой оператор, который находится в данном стеке, имеющий равный или же больший приоритет, и добавить в результирующий список.

Вычисление выражения:

1.1. Производим обработку списка слева направо.

1.1.1. Если операнд, то если это число сразу добавляем в стек операндов, если же буква, то просим пользователя ввести значение.

1.1.2. Если оператор *, /, +, -, то вытаскиваем из стека два операнда и выполняем необходимую арифметическую операцию и записываем в тот же стек.

1.1.3. Если оператор #(унарный минус), то вытаскиваем из стека один операнд и выполняем необходимую арифметическую операцию и записываем в тот же стек.

1.2. В итоге после обработки необходимо забрать единственный результат из стека.

Вариант:

1. (r/t/y/u + i*o + p – a + s)+(d – f)*(g + h)*(j + k)

Программный код

```
from pythonds.basic.stack import Stack
import sys
flag_unary = False
fl_neg= False
fl_shift = True
def is_number(string):
   if string.isdigit():
       return True
   else:
       try:
           float(string)
           return True
       except ValueError:
           return False

def reverse_polish_notation(infix):
   # удалим все пробелы из строки для удобства
   infix = infix.replace(" ", "")

   with_enter = ''
   fl_shift = True
   flag = False

   if infix[0] == '+':
       infix = infix[1:]
   #print ("len(infix): ", len(infix))

   if (infix[0] == '-'):
        #fl_shift = False
        infix = "#" + infix[1:]

   for i in range(1, len(infix)-1):
       if (infix[i] == "-")  and (is_number(infix[i + 1]) or infix[i + 1].lower() in
"abcdefghijklmnopqrstuvwxyz"  ) and ((infix[i-1] == "(") ):  #or (infix[i-1] == "")):
           #заменим унарный минус на #
           infix = infix[:i]+ "#" + infix[i+1:]
           #fl_shift = False

   # for i in range(0, len(infix) ):
   #     print("i", i, ": ", infix[i])
```

```
   if (infix[len(infix)-1] == '+' or infix[len(infix)-1] == '-' or infix[len(infix)-1] ==
'*' or \
           infix[len(infix)-1] == '/' or infix[len(infix)-1] == '#'):
       raise Exception("Ошибка 7. Некорректный(-ые) оператор(ы)")



   for i in range(len(infix)):
       if infix[i] == '.' or flag:
           flag = False
           with_enter += infix[i]
       else:
           # print ("------i", infix[i])
           if infix[i] != '(' and infix[i] != ')': #без учета скобок
               # Проверить что происходит чередование символов
               if (fl_shift) :
                    # Надо считать до момента пока не встретили пробел
                   # Текущий эл. переменная или число, тогда
                   if infix[i] != '#':
                        if infix[i].lower() in "abcdefghijklmnopqrstuvwxyz" or
is_number(infix[i]): # or (infix[i]=="#")
                            if i != len(infix) - 1: # если не достигли конца строки - 1
                                if (not is_number(infix[i + 1]) and infix[i + 1] != '.'  and
infix[i + 1] != '#'):
                                     fl_shift = False
                                 # if (infix[i] == '#'):
                                 #     fl_shift = False
                        else:
                             raise Exception("Ошибка 1. Некорректный(-ые) операнд(ы)")
               else:
                   if infix[i] == '+' or infix[i] == '-' or infix[i] == '*' or infix[i] ==
'/'  or infix[i] == '#':  # or infix[i] == '#'
                       fl_shift = True
                   else:
                       raise Exception("Ошибка 2. Некорректный(-ые) оператор(ы) или нет
оператора")
                #print("i--------------", i, ": ", infix[i])
               # if not check_num:
               # проверим нужно ли добавлять пробел
               if i != len(infix) - 1 or infix[-1] == ')' or infix[-1] == '(':
                   if (is_number(infix[i + 1]) or infix[i + 1] == '.') and infix[i] != '+'
and infix[i] != '*' and \
                            infix[i] != '-' and infix[i] != '/' and infix[i] != '#' :  # and
infix[i] != '#'
                       with_enter += infix[i]
                   else:
                       with_enter += infix[i] + ' '
               else:
                   with_enter += infix[i] + ' '
           else:
               with_enter += infix[i] + ' '


   infix = with_enter
   priority_operation = {}
   priority_operation["#"] = 4
   priority_operation["*"] = 3
   priority_operation["/"] = 3
   priority_operation["+"] = 2
   priority_operation["-"] = 2
   priority_operation["("] = 1
   op_stack = Stack()
   postfix_list = []

   token_list = infix.split()
```

```
    for token in token_list:
        if token.lower() in "abcdefghijklmnopqrstuvwxyz" or is_number(token):
            postfix_list.append(token)
        elif token == '(':
            op_stack.push(token)
        elif token == ')':
            if (not op_stack.isEmpty()):
                top_token = op_stack.pop()
            else:
                raise Exception("Ошибка 3. Лишняя скобка")
            while top_token != '(':
                postfix_list.append(top_token)
                # top_token = op_stack.pop()
                if (not op_stack.isEmpty()):
                    top_token = op_stack.pop()
                else:
                    raise Exception("Ошибка 4. Лишняя скобка")
        else:
            while (not op_stack.isEmpty()) and (priority_operation[op_stack.peek()] >=
priority_operation[token]):
                postfix_list.append(op_stack.pop())
            op_stack.push(token)

    while not op_stack.isEmpty():
        postfix_list.append(op_stack.pop())

    if '(' in postfix_list or ')' in postfix_list:
        raise Exception("Ошибка 5. Не хватает открывающей или закрывающей скобки(-ок)")
    return " ".join(postfix_list)


def calculation(operator, operand1, operand2):
    if operator == "#": #and not (fl_neg)
        return -operand1 #operand2 = 0
    elif operator == "*":
        return operand1 * operand2
    elif operator == "/":
        return operand1 / operand2
    elif operator == "+":
        return operand1 + operand2
    else:
        return operand1 - operand2


def postfix_calculation(postfix_expr):
    operand_stack = Stack()
    token_list = postfix_expr.split()
    dict_symbols = {}

    for token in token_list:
        if token.lower() in "abcdefghijklmnopqrstuvwxyz" or is_number(token):
            if token.lower() in "abcdefghijklmnopqrstuvwxyz":
                if (dict_symbols.get(token)== None):
                    num = float(input(token + " = "))
                    operand_stack.push(num)
                    dict_symbols[token] = num
                else:
                    operand_stack.push(dict_symbols.get(token))
            else:
                # если работаем с числами
                # print (token, "token")
                operand_stack.push(float(token))
        else:
            if (token == '#'):
                # print ("Унарная операция", token)
                operand2 = 0
                operand1 = operand_stack.pop()
```

```
                result = calculation(token, operand1, operand2)
                operand_stack.push(result)
            else:
                # print("Знак операции", token)
                operand2 = operand_stack.pop()
                operand1 = operand_stack.pop()
                result = calculation(token, operand1, operand2)
                operand_stack.push(result)
    # print(operand_stack.pop())
    buf = operand_stack.pop()
    # print(buf)
    return buf


fl = True
while fl:
    try:
        # print("Выражение для 1 варианта: \n", "(r/t/y/u + i*o + p - a + s) + (d - f)*(g + h)*(j + k)")
        # v_1 = reverse_polish_notation("(r/t/y/u + i*o + p - a + s) + (d - f)*(g + h)*(j + k)")
        # print("Постфиксная запись: \n", v_1)
        # res_v1 = postfix_calculation(v_1)
        # print("Вычисление значения для данного выражения: ", res_v1 , "\n")
        #print("----")

        infix = input("Введите выражение: ")
        postfix = reverse_polish_notation(infix)
        if not (flag_unary):
            if not postfix and (
                    '+' not in postfix and '-' not in postfix and '*' not in postfix and '/' not in postfix):
                raise Exception("Ошибка 6. Некорректный ввод")
            print("Постфиксная запись: {}".format(postfix))
        try:
            if not (flag_unary):
                res = postfix_calculation(postfix)
            else:
                res = - postfix_calculation(postfix)
            print("Результат = {}".format(res))
        except ValueError:
            print("Ошибка. Некорректный тип данных, ожидалось число ")
        except ZeroDivisionError:
            print("Ошибка. Попытка деления на 0 ")
    except Exception:
            e = sys.exc_info()[1]
            print(e.args[0])
    print("\n\n")
```

Тестовые примеры

№	Входные данные	Результат
1	(r/t/y/u + i*o + p - a + s) + (d - f)*(g + h)*(j + k) r = 8 t = 2 y = 2 u = 2 i = 1	Введите выражение: (r/t/y/u + i*o + p - a + s) + (d - f)*(g + h)*(j + k) Постфиксная запись: r t / y / u / i o * + p + a - s + d f - g h + * j k + * + r = 8 t = 2 y = 2

	o = 3 p = 4 a = 2 s = 5 d = 4 f = 2 g = 5 h = 3 j = 1 k = 2	u = 2 i = 1 o = 3 p = 4 a = 2 s = 5 d = 4 f = 2 g = 5 h = 3 j = 1 k = 2 Результат = 59.0
2	a - b a = 7 b = 2	Введите выражение: a - b Постфиксная запись: a b - a = 7 b = 2 Результат = 5.0
3	1 + 3	Введите выражение: 1 + 3 Постфиксная запись: 1 3 + Результат = 4.0
4	(5 * 11)	Введите выражение: (5 * 11) Постфиксная запись: 5 11 * Результат = 55.0
5	a/b a = 3 b = 7	Введите выражение: a/b Постфиксная запись: a b / a = 3 b = 7 Результат = 0.42857142857142855
6	1 / (a - b) a = 2 b = 2	Введите выражение: 1 / (a - b) Постфиксная запись: 1 a b - / a = 2 b = 2 Ошибка. Попытка деления на 0
7	-a a = 5	Введите выражение: -a Постфиксная запись: a # a = 5 Результат = -5.0
8	a + b * c	Введите выражение: a + b * c

	a = 1 b = 2 c = 3	Постфиксная запись: a b c * + a = 1 b = 2 c = 3 Результат = 7.0
9	a*(b*(c-d)) a = 2 b = 3 c = 5 d = 1	Введите выражение: a*(b*(c-d)) Постфиксная запись: a b c d - * * a = 2 b = 3 c = 5 d = 1 Результат = 24.0
10	(7*(20+10))	Введите выражение: (7*(20+10)) Постфиксная запись: 7 20 10 + * Результат = 210.0
11	ф + и	Введите выражение: ф + и Ошибка 1. Некорректный(-ые) операнд(ы)
12	a @ b	Введите выражение: a @ b Ошибка 2. Некорректный(-ые) оператор(ы) или нет оператора
13	)	Введите выражение:) Ошибка 3. Лишняя скобка
14	b+c-d)	Введите выражение: b+c-d) Ошибка 4. Лишняя скобка
15	((a - b) + c	((a - b) + c Ошибка 5. Не хватает открывающей или закрывающей скобки(-ок)
16	()	Введите выражение: () Ошибка 6. Некорректный ввод
17	a + b * c a = one	Введите выражение: a + b * c Постфиксная запись: a b c * + a = one Ошибка. Некорректный тип данных, ожидалось число
18	a/ (b * c) a = 1 b = 3 c = 0	Введите выражение: a/ (b * c) Постфиксная запись: a b c * / a = 1 b = 3 c = 0 Ошибка. Попытка деления на 0

19	-(-a+1) a = 4	Введите выражение: -(-a+1) Постфиксная запись: a # 1 + # a = 4 Результат = 3.0
20	-(-a-1) a = 4	Введите выражение: -(-a-1) Постфиксная запись: a # 1 - # a = 4 Результат = 5.0

21	-6*a+1/a
	a = 2
		Введите выражение: -6*a+1/a
		Постфиксная запись: 6 # a * 1 a / +
		a = 2
		Результат = -11.5

22	-6*a+1/b
	a = 2
	b = 1
		Введите выражение: -6*a+1/b
		Постфиксная запись: 6 # a * 1 b / +
		a = 2
		b = 1
		Результат = -11.0

23	-(a) a = 3	Введите выражение: -(a) Постфиксная запись: a # a = 3 Результат = -3.0
24	(a+(b-c)	Введите выражение: (a+(b-c) Ошибка 5. Не хватает открывающей или закрывающей скобки(-ок)
25	(a+b))	Введите выражение: (a+b)) Ошибка 3. Лишняя скобка