实验 3 中缀算术表达式求值的指导

王天一 320200931301

1 题目

输入一个中缀算术表达式，计算其结果。对输入的表达式，做如下假设：

（1） 只考虑+、-、*、/这四种运算符，中缀表达式中只有一种括号（）；

（2）输入的中缀表达式中数字只有整数，没有小数；

（3）假定输入表达式是合法的。

定义输入方式：每个数字，操作符（括号也视为操作符）

1 + (2 - 3) * 4 + 4 / 2 （输入格式样例）

如果加入乘方'^'运算，如何设计？

2 测试用例

中缀表达式：（1）1+(2-3)*4+4/2 （2）a+(b-c)*d

后缀表达式：（1）1 2 3-4*+4 2 / + （2）abc-d*+

测试结果：（分别用两种方式计算）

```
PS C:\Users\wty02> & C:/Users/wty02/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/
top/数据结构实验/Data_Structure_Lab/lab3/lab3.py
Please input the infix expression:(' ' between two numbers):1 + ( 2 - 3 ) * 4 + 4 / 2
Sovle using postfix expression:
-1.0
Sovle using infix expression:
-1.0
PS C:\Users\wty02>
```

3 算法

计算中缀算术表达式是有关栈的经典算法题，很多公司面试或者笔试的时候都会把计算中缀表达式作为一个考察点。这里先讨论如何将中缀表达式转化为后缀表达式，再讨论如何计算后缀表达式，最后讨论如何一步到位，直接计算中缀表达式的值。

3.1 中缀表达式转为后缀表达式

中缀转化为后缀，要先设一个栈存储运算符，从左到右遍历表达式，主要遵循以下原则：

（1）读到操作数，直接输出；

（2）栈为空时，遇到运算符，入栈；

（3）遇到左括号，将其入栈；

（4）遇到右括号，执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出， 左括号也要弹出；

（5）遇到其他运算符’+”-”*”/’时，弹出所有优先级大于或等于该运算符的栈顶元素，然后将该运算符入栈， 遇到左括号时不能再弹出；

（6）最终将栈中的元素依次出栈，输出。

经过上面的步骤，得到的输出就是转换得到的后缀表达式。

说明：输出的后缀表达式可以一个队列存储。

具体实现：

首先我们按照思路设置了两个栈，符号和运算数栈。

然后设置权重字典。

然后用 for i in * 遍历，实现运算，具体实现参考代码。

实现代码：（压缩包内附有文件）

```
def InfixToPostfix(x):
    """translate infix expression to postfix expression

    Args:
        x (List): the infix expression

    Returns:
        float : the result of postfix expression
    """
    stack1 = [] # stack1 for operator
    stack2 = [] # stack2 for number
    WeightValue = {'^': 4, '*': 3, '/': 3, '+': 2, '-': 2, '(': 1} # operator
    weight
    for i in x: # for each element in the infix expression
        if i.isdigit(): # if the element is a number
            stack2.append(i) # push the number to stack2
        elif i in ['+', '-', '*', '/', '^']: # if the element is an operator
            if stack1 == [] or stack1[-1] == '(': # if stack1 is empty or the
            last element in stack1 is '('
                stack1.append(i) # push the operator to stack1
            elif WeightValue[i] > WeightValue[stack1[-1]]: # if the weight of
            the operator is greater than the last element in stack1
                stack1.append(i) # push the operator to stack1
            elif WeightValue[i] <= WeightValue[stack1[-1]]: # if the weight of
            the operator is less than or equal to the last element in stack1
                while stack1 != [] and WeightValue[i] <= WeightValue[stack1[-1]]
                : # if the weight of the operator is less than or equal to the
                last element in stack1
                    stack2.append(stack1.pop())# pop the last element in stack1
                    and push it to stack2
                stack1.append(i)# push the operator to stack1
        elif i == '(': # if the element is '('
            stack1.append(i) # push the element to stack1
        elif i == ')': # if the element is ')'
            while stack1[-1] != '(': # if the last element in stack1 is not '('
                stack2.append(stack1.pop()) # pop the last element in stack1
                and push it to stack2
            stack1.pop() # pop the last element in stack1
    while stack1 != []: # if stack1 is not empty
        stack2.append(stack1.pop()) # pop the last element in stack1 and push
        it to stack2
    return stack2
```

3.2 后缀表达式求值

设后缀表达式的栈为 post,求值计算采用辅助栈 result 实现，算法如下：

(1)从后缀表达式栈 post 中弹出栈顶元素，若是操作数，压入 result；若是运算符 oprater，则从 result 中取出两个元素，进行 oprator 运算，结果压入 result。

(2)若 post 为空，结束，result 存放的结果即为运算结果；否则转向（1）。

假定待求值的后缀表达式为：6 5 2 3 + 8 * + 3 + *，则其求值过程如下：

- 遍历表达式，遇到数字首先放入栈，此时栈如下 6 5 2 3
- 接着读到+，则弹出 3 和 2，执行 3+2，将结果 5 压栈 6 5 5
- 读到 8，压栈 6 5 5 8

- 读到 *，弹出 8 和 5，执行 8*5，将结果 40 压栈 6 5 40
- 读到 +,弹出 40 和 5，执行 40+5，将结果 45 压栈 6 45
- 读到 3,压栈 6 45 3
- 读到 +,弹出 3 和 45，执行 3+45，将结果 48 压栈 6 48
- 读到 *,弹出 48 和 6，执行 48*6，将结果 288 压栈 288
- 最后结果 288

链接：https://www.jianshu.com/p/fcd2b521a3e2

具体实现：

建立 result 栈和 post 栈，然后挨个 pop 出 result 中的元素，挨个运算。

实现代码：（压缩包内附有文件）：

```
def caculatePostfix(x):
    """calculate the result of postfix expression

    Args:
        x (List): the postfix expression

    Returns:
        float: the result of postfix expression
    """
    result = [] # result for the result of postfix expression
    post = x
    for i in post: # for each element in postfix expression
        if i.isdigit():# if the element is a number
            result.append(i) # push the number to result
            #print(i,"is a number")
        else:
            num1 = result.pop()
            num2 = result.pop()
            #print("caculate",int(num1),int(num2),i)
            result.append(caculateTheResult(i, float(num1), float(num2)))
    return result[0]
```

3.3 直接计算中缀表达式

设置两个工作栈，一个操作数栈，一个操作符栈，在（自左至右）扫描算术表达式时，遇到操作数直接入操作数栈，若遇到操作符，则根据操作符优先级判断下一步操作（“操作符优先级规则”）：若其优先级高于栈顶操作符，则入栈，否则（相等或小于），弹出栈顶算符并从操作数栈弹出两个操作数，计算，将计算结果入操作数栈，继续比较与栈顶操作符的优先级，若仍然等于或低于之，则计算，直至大于之，则将此操作符入栈；左括号一定入栈，且其优先级低于后续来到的任何操作符，右括号一定出栈并计算直至遇到左括号，另外栈的开始以“#”开始，算术表达式以“#”结束，做结束标志。当栈空时，计算结束。

具体实现：

首先定义两个栈：数字栈和操作数栈，然后从头到尾挨个取出来，然后挨个进行运算，具体实现看代码和注释。

实现代码：（压缩包内附有文件）：

```
def caculateInfix(x):
    """calculate the result of infix expression

    Args:
        x (List): the infix expression

    Returns:
        float: the result of infix expression
    """
    numberStack = []
    operatorStack = []
    WeightValue = {')': 5, '^': 4, '*': 3, '/': 3, '+': 2, '-': 2, '(': 1}#
    operator weight
    for i in x:
        if i.isdigit(): # if the element is a number
            numberStack.append(i) # push the number to numberStack
        elif i in ['+', '-', '*', '/', '^']: # if the element is an operator
            if operatorStack == [] or WeightValue[i] > WeightValue[
                    operatorStack[-1]]:# if operatorStack is empty or the
                    weight of the operator is greater than the last element in
                    operatorStack
                operatorStack.append(i)
            elif WeightValue[i] <= WeightValue[operatorStack[-1]]:# if the
            weight of the operator is less than or equal to the last element in
            operatorStack
```

```
            while operatorStack != [] and WeightValue[i] <= WeightValue[
                    operatorStack[-1]]:# if the weight of the operator is
                    less than or equal to the last element in operatorStack
                number1 = numberStack.pop()# pop the last element in
                numberStack
                number2 = numberStack.pop()# pop the last element in
                numberStack
                operator = operatorStack.pop()# pop the last element in
                operatorStack
                numberStack.append(
                    caculateTheResult(operator, float(number1),
                                      float(number2)))# push the result of
                                      caculate to numberStack
            operatorStack.append(i)# push the operator to operatorStack
    elif i == '(':
        operatorStack.append(i)# push the element to operatorStack
    elif i == ')':
        while operatorStack[-1] != '(':# if the last element in
        operatorStack is not '('
            number1 = numberStack.pop()# pop the last element in numberStack
            number2 = numberStack.pop()# pop the last element in numberStack
            operator = operatorStack.pop()# pop the last element in
            operatorStack
            numberStack.append(
                str(caculateTheResult(operator, float(number1),
                                      float(number2))))# push the result of
                                      caculate to numberStack
        operatorStack.pop()# pop the last element in operatorStack
while operatorStack != []: # if operatorStack is not empty
    number1 = numberStack.pop()# pop the last element in numberStack
    number2 = numberStack.pop()# pop the last element in numberStack
    operator = operatorStack.pop()# pop the last element in operatorStack
    #print(caculateTheResult(operator, int(number1), int(number2)))
    numberStack.append(
        str(caculateTheResult(operator, float(number1), float(number2))))#
        push the result of caculate to numberStack
return numberStack[0]
```

4 算法对数据的操作过程

例子：中缀表达式 a + b * c + (d * e + f) * g ，其转换成后缀表达式则为 a b c * + d e * f + g * +。

步骤	扫描	stack 栈	输出
1	a	空	a
2	a+	+	a
3	a+b	+	ab
4	a+b*	+*	ab
5	a+b*c	+*	abc
6	a+b*c+	+	abc*+
7	a+b*c+(	+(	abc*+
8	a+b*c+(d	+(	abc*+d
9	a+b*c+(d*	+(*	abc*+d
10	a+b*c+(d*e	+(*	abc*+de
11	a+b*c+(d*e+	+(+	abc*+de*
12	a+b*c+(d*e+f	+(+	abc*+de*f
13	a+b*c+(d*e+f)	+	abc*+de*f+
14	a+b*c+(d*e+f)*	+*	abc*+de*f+
15	a+b*c+(d*e+f)*g	+*	abc*+de*f+g
16	a+b*c+(d*e+f)*g#	空	abc*+de*f+g*+

注意：第 6 步，读到+,因为栈顶元素*的优先级高，所以*出栈，栈中下一个元素+优先级与读到的操作符+一样，所以也要弹出,然后再将读到的+压入栈中。第 13 步，读到)，则直接将栈中元素弹出直到遇到(为止。这里左括号前只有一个操作符+被弹出。

5 扩展部分设计

如果加入乘方'^'运算，要注意乘方运算是右结合的,需要修改两处：

(1)将乘方添加到优先级中：

1：（

2：+ -

3：* /

4：^

5：）

(2)在读中缀表达式的时候，如果读到乘方^，就将它放进符号栈中。因为乘方的优先级是最高的，而且是右结合的，所以无论它前面出现的是什么运算，这些运算都不能执

行。而且它本身能否执行也是不知道的，因此只能进栈。

https://blog.csdn.net/qq_26286193/article/details/80214805

实现：

```
    WeightValue = {')': 5, '^': 4, '*': 3, '/': 3, '+': 2, '-': 2, '(': 1}#
    operator weight
```