姓名：徐建霞　学号：202113090055

ch3-1 栈的应用：算数表达式求值

一、题目

编程实现将一个给定的算术表达式转换为后序表达式并进行计算（只涉及+ - × / 和括号）。
要求：对于输入一个任意的算术表达式（可能有括号）
1、能够判断该算术表达式的合法性
2、输出相应的后序表达式
3、输出计算的结果
4、至少验证如下三个实验用例 （有合法的用例，也有非法的用例）：
4.1 (6+7) * (8-2) / (5+3)
4.2 15 - 23*(5+ 2*2*(3+1))
4.3 (((2+1) * 3 + 2) *5 + 100

作业文档以 PDF 格式提交：必须有程序和运行结果截图，对自己的算法思想进行阐述。

二、代码

```
from pythonds.basic import Stack

##判断算数表达式的合法性
def cutoper(expr):
    index = 0
    pars =[]

    while index < len(expr):
        if expr[index] in "([{)]}":
            pars.append(expr[index])
        index += 1
    return pars

def parChecker(expr):
    s = Stack()
    balanced = True
    index = 0
    while index < len(expr) and balanced:
        word = expr[index]
        if word == "(":
            s.push(word)
```

```
        else:
            if s.isEmpty():
                balanced = False
            else:
                s.pop()
        index = index + 1
    if balanced and s.isEmpty():
        return True
    else:
        return False


##中序表达式转后序表达式
def infixToPostfix(infixexpr):
    prec = {}
    prec["*"] = 3
    prec["/"] = 3
    prec["+"] = 2
    prec["-"] = 2
    prec["("] = 1

    opStack = Stack()
    postfixList = []

    tokenList = infixexpr.split()

    for token in tokenList:
        if token not in "([{}]}+-*/":
            # print(type(token))
            postfixList.append(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()
            while topToken != '(':
                postfixList.append(topToken)
                topToken = opStack.pop()

        else:
            while (not opStack.isEmpty()) and (prec[opStack.peek()] >= prec[token]):
                postfixList.append(opStack.pop())
            opStack.push(token)

    while (not opStack.isEmpty()):
```

```
        postfixList.append(opStack.pop())

    print(" ".join(postfixList))

    return " ".join(postfixList)

##后序表达式的计算
###辅助函数
def doMath(op,op1,op2):
    if op == "*":
        return op1 * op2
    elif op == "/":
        return op1/op2
    elif op == "+":
        return op1+op2
    else:
        return op1-op2


def postfixEval(postfixexpr):
    operandStack = Stack()

    tokenList = postfixexpr.split()

    for token in tokenList:
        if token not in "([{}]}+-*/":
            operandStack.push(int(token))
        else:
            operand2 = operandStack.pop()
            operand1 = operandStack.pop()
            result = doMath(token,operand1,operand2)
            operandStack.push(result)

    return operandStack.pop()

def output(expr):
    if (parChecker(cutoper(expr))):
        postfix = infixToPostfix(expr)
        result = postfixEval(postfix)
        print("expr=",result)
    else:
        print("the input is not valid!")

# (647) * (8-2) / (543)
```

```
# 15 - 23*(5+ 2*2*(3+1))
# (((2+1) * 3 + 2) *5 + 100

if __name__ == '__main__':
    expr1 = '( 6 + 7 ) * ( 8 - 2 ) / ( 5 + 3 )'
    expr2 = ' 15 - 23 * ( 5 + 2 * 2 * ( 3 + 1 ) )'
    expr3 = '( ( ( 2 + 1 ) * 3 + 2 ) * 5 + 100'

    output(expr1)
    output(expr2)
    output(expr3)
```

三、运行结果（用例验证）

```
D:\anaconda\python.exe D:/PycharmProjects/0925assignment/complete.py
6 7 + 8 2 - * 5 3 + /
expr= 9.75
15 23 5 2 2 * 3 1 + * + * -
expr= -468
the input is not valid!

Process finished with exit code 0
```

四、算法思想

（1）判断算术表达式的合法性：cutoper()　parChecker()

先用 cutoper()函数将算术表达式中除操作数以外的其他括号提取到一个列表 pars 中，再用 parChecker()函数检测括号是否匹配，以此来判断算术表达式是否合法。

具体算法：由一个空栈开始，从左到右依次处理括号。如遇左括号，便通过 push 操作将其加入栈中，以此表示稍后需要有一个与之匹配的右括号。反之，如遇右括号，就调用 pop 操作。只要栈中有任何一个左括号找不到与之匹配的右括号，则表达式不合法。

（2）中序表达式转后续表达式：infixToPostfix()

当遇左括号时，用栈保存下来，以表示接下来会遇到高优先级的运算符；那个运算符需要等到对应的右括号出现才能确定其位置；当右括号出现时，便可以将运算符从栈中取出来。

当从左到右扫描中序表达式时，遇到运算符将其保存到栈中，以便提供反转特性。

1. 创建用于保存运算符的空栈 opStack，以及一个用于保存结果的空列表。
2. 使用字符串方法 split 将输入的中序表达式转换成一个列表
3. 从左往右扫描这个标记列表、
   - 如果标记是操作数（不是运算符和括号），将其添加到列表的末尾
   - 如果是左括号，将其压入 opStack 中
   - 如果标记是右括号，反复从 opStack 栈中移除元素。直到移除对应的左括号。将栈中取出的每一个运算符都添加到结果列表的末尾。
   - 如果标记是运算符，将其压入 opStack 栈中。但是在这之前，需要取出优先级更高或相同的运算符，并将它们添加到结果列表的末尾。

4.当处理完输入表达式后，检查 opStack。将其中所有残留的运算符全部添加到结果列表的末尾。

（3）计算后序表达式：postfixEval()

计算后续表达式时，需要保存的是操作数而不是运算符。当遇到一个运算符时，需要用离它最近的两个操作数来计算。

具体算法：

1. 创建空栈 operandStack
2. 使用字符串方法 split 将会输入的后续表达式转换成一个列表
3. 从左到右扫描这个标记列表
   - 如果标记是操作数，将其转换成整数并且压入 operandStack 栈中
   - 如果标记是运算符，从 operandStack 栈中取出两个操作数。第一次取出右操作数，第二次取出左操作数。进行相应的算术运算，然后将结果压入 operandStack 栈中。

4.当处理完输入表达式时，栈中的值就是结果。将其从栈中返回。