1 运算符表及说明

1. 以下是本次 lab 程序中支持的运算符总表，请您查阅。
2. 由于程序使用编程语言为 Python3，故各运算符的形式与定义大多与 Python3 中保持一致。但为对用户友好，也兼容一些类似 C++ 和 Python2 形式的运算符输入，详情请参见下表。
3. 您在输入中缀表达式时，请用空格分隔各个运算符和数值。
4. 本程序支持浮点数与负数。负数输入时请将负号和数字连写。
5. 在 Python3 中布尔型是整型的子类型。本程序支持逻辑表达式，您如果输入 True 或 False 作为数值，会被分别转化为整数 1 和 0 参与运算。

运算符	表达式	含义	优先级（数值越大越高）
**	a ** b	a 的 b 次幂	13
~	~ a	按位取反	12
*	a * b	a 乘 b	11
/	a / b	a 除以 b（这不是整除）	11
mod	a mod b	a 除以 b 得到的余数	11
%	a % b	a 除以 b 得到的余数, 同上	11
//	a // b	a 整除 b	11
+	a + b	a 加 b	10
-	a - b	a 减 b	10
«	a « b	a 左移 b 位，两数都需是整数	9
»	a » b	a 右移 b 位，两数都需是整数	9
&	a & b	按位与	8
^	a ^ b	按位异或	7
|	a | b	按位或	7
<=	a <= b	a 是否小于等于 b	6
<	a < b	a 是否小于 b	6
>	a > b	a 是否大于 b	6
>=	a >= b	a 是否大于等于 b	6

<>	a <> b	a 是否不等于 b	5
!=	a != b	a 是否不等于 b，同上	5
==	a == b	a 是否等于 b	5
not	not a	非 a	1
!	! a	非 a，同上	1
and	a and b	a 且 b	1
&&	a && b	a 且 b，同上	1
or	a or b	a 或 b	1
||	a || b	a 或 b，同上	1

表 1：程序支持的运算符总表

2 中缀表达式转为后缀表达式

1. 仅供展示和快速阅读，以 main.py 中实际代码为准。
2. 逐个读取各数值和符号，并做不同处理。
3. 如果是数值（包括正负数和布尔值），加入转化结果中。
4. 如果是符号，当栈为空或是左括号时，加入转化结果中。读到右括号就一直弹出栈中内容加入结果，直到左括号，并弹出左括号。读到其他符号，则一直弹出栈中内容加入结果，直到栈顶比此符号优先级小或栈为空，并向栈中加入此符号。
5. 最后弹出栈中剩余内容加入结果。

算法 1: 中缀转后缀核心算法

```
def convert(infix):
        opeStack = Stack()
        result = []
        for t in infix:
                if ((t[0] >= "0" and t[0] <= "9") \
                or (t[0] == "-" and len(t) > 1)):
                        result.append(t)
                elif(t == "True" or t == "False"):
```

```
                        result.append(int(t))
                elif(opeStack.empty() or t == "(" or t == "{"):
                        opeStack.push(t)
                elif(t == ")" or t == "}"):
                        while(opeStack.top() != pair[t]):
                                temp = opeStack.pop()
                                result.append(temp)
                        opeStack.pop()
                else:
                        while(not opeStack.empty() and\
                        priority[opeStack.top()] >= priority[t]):
                                temp = opeStack.pop()
                                result.append(temp)
                        opeStack.push(t)
        while(not opeStack.empty()):
                result.append(opeStack.pop())
        return result
```

3 计算后缀表达式

1. 仅供展示和快速阅读，以 main.py 中实际代码为准。
2. 逐个读取各数值和符号，并做不同处理。
3. 如果是数值，推入栈中。
4. 如果是符号，从栈中弹出所要求个数的数值，做运算后将结果推入栈中。

算法 2: 计算后缀表达式算法（部分）

```
def calculate(postfix):
        calStack = Stack()
        for t in postfix:
                if ((t[0] >= "0" and t[0] <= "9") or\
                (t[0] == "-" and len(t) > 1)):
                        if(isFloat(t)):
                                calStack.push(float(t))
                        else:
```

```
9                                   calStack.push(int(t))
10                          else:
11                                  y = calStack.pop()
12                                  x = calStack.pop()
13                                  if(t == "+"):
14                                          calStack.push(x + y)
15                                  elif(t == "-"):
16                                          calStack.push(x - y)
17                                  elif(t == "*"):
18                                          calStack.push(x * y)
19                                  elif(t == "/"):
20                                          if(y == 0):
21                                                  return "Error!"
22                                          calStack.push(x / y)
23          return calStack.top()
```