前中后缀表达式

概念

- 前缀表达式
  - 运算符写在前面，操作数写在后面
  - 形式：*+ab+cd
  - 实质上是表达式树的先根遍历
- 中缀表达式
  - 正常的运算方式
  - 形式：(a+b)*(c+d)
  - 实质上是表达式树的中根遍历
- 后缀表达式
  - 又称逆波兰式，指的是不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行（不再考虑运算符的优先规则）
  - 形式：ab+cd+*
  - 实质上是表达式树的后根遍历

中缀转后缀

算法

1. 任何中缀表达式都由运算数，运算符，括号（大，中，小），这三部分组成。
2. 从中缀表达式的左边开始扫描，若遇到运算数时，则直接将其输出（不压入堆栈）。
3. 若遇到左括号，则将其压栈。
4. 若遇到右括号，表达括号内的中缀表达式已经扫描完毕。这时需将栈顶的运算符依次弹出并输出，直至遇到左括号，左括号弹出但不输出。
5. 若遇到的是运算符：
   1. 如果该运算符的优先级大于栈顶运算符的优先级时，将其压栈
   2. 如果该运算符的优先级小于栈顶运算符的优先级时，将栈顶运算符弹出并输出，接着和新的栈顶运算符比较，若大于，则将其压栈，若小于，继续将栈顶运算符弹出并输出，一直递归下去，直至运算符大于栈顶云算符为止。
6. 最后一步，若扫描到中缀表达式的末尾，若堆栈中还有存留的运算符依次弹出并输出即可。

例子 2*(9+6/3-5)+4

```
1. out: 2                  stack:
2. out: 2                  stack: *
3. out: 2                  stack: *
4. out: 2 9                stack: * (
5. out: 2 9                stack: * ( +    // 在堆栈中括号的优先级最低
6. out: 2 9 6              stack: * ( +
7. out :2 9 6              stack: * ( + /
8. out :2 9 6 3            stack: * ( + /
9. out :2 9 6 3 /          stack: * ( +
10. out: 2 9 6 3 / +       stack: * (
11. out: 2 9 6 3 / +       stack: * (-
12. out: 2 9 6 3 / + 5     stack: * (-    // 遇到了右括号
13. out: 2 9 6 3 / + 5 -   stack: * (
14. out: 2 9 6 3 / + 5 -   stack: *
15. out: 2 9 6 3 / + 5 -   stack: *      // 括号弹出但不输出
16. out: 2 9 6 3 / + 5 - * stack:        // 遇到了+
17. out: 2 9 6 3 / + 5 - * stack: +
18. out: 2 9 6 3 / + 5 - * 4  stack: +
19. out: 2 9 6 3 / + 5 - * 4 +  stack:
```

后缀转中缀

算法

1. 把后缀表达式逐个元素的压入到栈中
   1. 当压入的都是字符，则不采取任何操作
   2. 当压入的是运算符，则把运算符下面的两个数字弹出和运算符进行运算，然后把结果继续压入到栈中。
2. 对后缀表达式中的所有元素执行该操作，直到结束。

例子 6523+8\+3+*

```
1. in:6523                 stack:6523
2. in:+                    stack:65(2+3)              // 弹出 3，2，得到结果(2+3)，入栈
3. in:8                    stack:65(2+3)8
4. in:*                    stack:65((2+3)*8)          // 弹出（2+3），8，得到结果（（2+3）*8），入栈
5. in:+                    stack:6(((2+3)*8)+5)       // 弹出（（2+3）*8），5，得到结果(((2+3)*8)+5)，入栈
6. in:3                    stack:6(((2+3)*8)+5)3
7. in:+                    stack:6((((2+3)*8)+5)+3)   // 弹出(((2+3)*8)+5)，3，得到结果((((2+3)*8)+5)+3)，入栈
8. in:*                    stack:(((((2+3)*8)+5)+3)*6)  // 弹出((((2+3)*8)+5)+3)，6，得到结果(((((2+3)*8)+5)+3)*6)
```

中缀转前缀

算法

1. 任何中缀表达式都由运算数，运算符，括号（大，中，小），这三部分组成。
2. 从中缀表达式的右边开始往左扫描，若遇到运算数时，则直接将其输出（不压入堆栈）。
3. 若遇到右括号，则将其压栈。
4. 若遇到左括号，表达括号内的中缀表达式已经扫描完毕。这时需将栈顶的运算符依次弹出并输出，直至遇到右括号，右括号弹出但不输出。
5. 若遇到的是运算符
   1. 如果该运算符的优先级大于栈顶运算符的优先级时，将其压栈
   2. 如果该运算符的优先级小于栈顶运算符的优先级时，将栈顶运算符弹出并输出，接着和新的栈顶运算符比较，若大于，则将其压栈，若小于，继续将栈顶运算符弹出并输出，一直递归下去，直至运算符大于栈顶云算符为止。
6. 若扫描到中缀表达式的末尾，若堆栈中还有存留的运算符依次弹出并输出即可。
7. 将得到的表达式翻转

例子 1+2*(3-4)-5*6

```
1. out:6           stack:
2. out:6           stack:*
3. out:65          stack:*
4. out:65*         stack:-    // 运算符-优先级小于*，-入栈，*出栈
5. out:65*         stack:-)
6. out:65*4        stack:-)
7. out:65*4        stack:-)-
8. out:65*43       stack:-)-
9. out:65*43-      stack:-    // 输出）（之间的所有运算符
10. out:65*43-     stack:-*
11. out:65*43-2    stack:-*
12. out:65*43-2*   stack:-+
13. out:65*43-2*1  stack:-+
14. out:65*43-2*1+- stack:    // 输出栈内所有运算符
15: out:-+1*2-34*56 stack:    // 反转输出
```

前缀转中缀

算法

1. 从右向左把前缀表达式逐个元素的压入到栈中
   1. 当压入的都是字符，则不采取任何操作
   2. 当压入的是运算符，则把运算符下面的两个数字弹出和运算符进行运算，然后把结果继续压入到栈中。
2. 对前缀表达式中的所有元素执行该操作，直到结束。

例子 +-*^ABCD/E/F+GH

```
1.  in:HG                  stack:HG
2.  in:+                   stack:(G+H)
3.  in:F                   stack:(G+H),F
4.  in:/                   stack:F/(G+H)
5.  in:E                   stack:F/(H+G),E
6.  in:/                   stack:E/F/(H+G)
7.  in:ABCD                stack:E/F/(H+G),D,C,B,A
8.  in:^                   stack:E/F/(H+G),D,C,A^B
9.  in:*                   stack:E/F/(H+G),D,A^B*C
10. in:-                   stack:E/F/(H+G),A^B*C-D
11. in:+                   stack:A^B*C-D+E/F/(H+G)
```