4.栈与队列

逆波兰表达式

- 转换

将欲去之，必固举之

将欲夺之，必固予之

将欲灭之，必先学之

邓俊辉

deng@tsinghua.edu.cn

infix到postfix：手工转换

❖ 例如：(0 ! + 1) ^ (2 * 3 ! + 4 - 5)

假设：事先未就运算符之间的优先级关系做过任何约定

1）用括号显式地表示优先级

{ ([0 !] + 1) ^ ([(2 * [3 !]) + 4] - 5) }

2）将运算符移到对应的右括号后

{ ([0] ! 1) + ([(2 [3] !) * 4] + 5) - } ^

3）抹去所有括号

0 ! 1 + 2 3 ! * 4 + 5 - ^

4）稍事整理，即得

0 ! 1 + 2 3 ! * 4 + 5 - ^

❖ 中缀式求值算法evaluate()略做扩展，亦可同时完成RPN转换...

infix到postfix：转换算法

```
❖float evaluate( char* S, char* & RPN ) { //RPN转换
   /* ..................................... */
   while ( ! optr.empty() ) { //逐个处理各字符，直至运算符栈空
     if ( isdigit( * S ) ) //若当前字符为操作数，则直接将其
       { readNumber( S, opnd ); append( RPN, opnd.top() ); } //接入RPN
     else //若当前字符为运算符
       switch( orderBetween( optr.top(), *S ) ) {
         /* ..................................... */
         case '>': { //且可立即执行，则在执行相应计算的同时将其
           char op = optr.pop(); append( RPN, op ); //接入RPN
           /* ..................................... */
         } //case '>'
   /* ..................................... */
```

思考

- 这里的多数实例中，为何操作数都是顺序排列？
- “3 + 4”除了可以转换为“3 4 +”，是否也可转换为“4 3 +”？
- 既然evaluate()算法已经能够求值，同时完成RPN转换又有何意义？
- 相对于原表达式，存储RPN所需的空间是否一定更少？
- 在数学意义上完全对称的前缀表达式，为何在类似问题中很少应用？