前缀中缀后缀相互转换

符号说明

- 为了表示简便，程序中符号如下

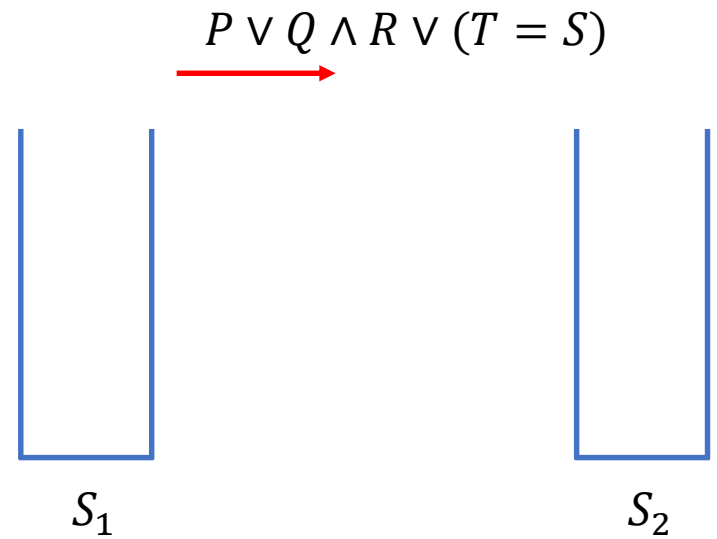
  - ¬,　　非
  - ∨,　　或
  - ∧,　　与
  - →,　　推出
  - ＝,　　等价

中缀转后缀

- 例：$P \vee Q \wedge R \vee (T = S)$　　输出：$PQR \wedge \vee TS = \vee$

算法流程：

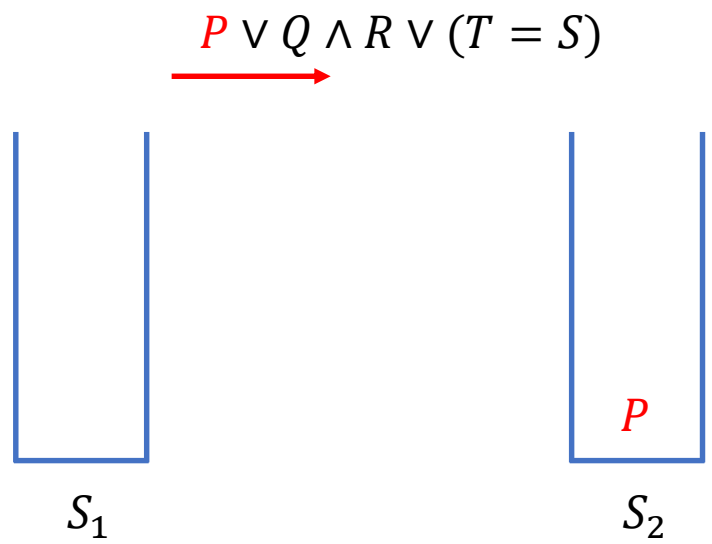
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符
    3. 遇到括号



中缀转后缀

算法流程：

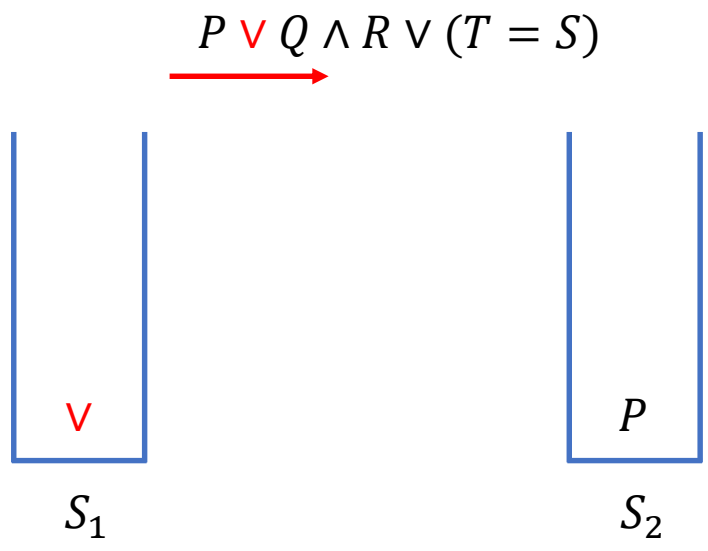
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数：直接压入S_2
    2. 遇到运算符
    3. 遇到括号



中缀转后缀

算法流程：

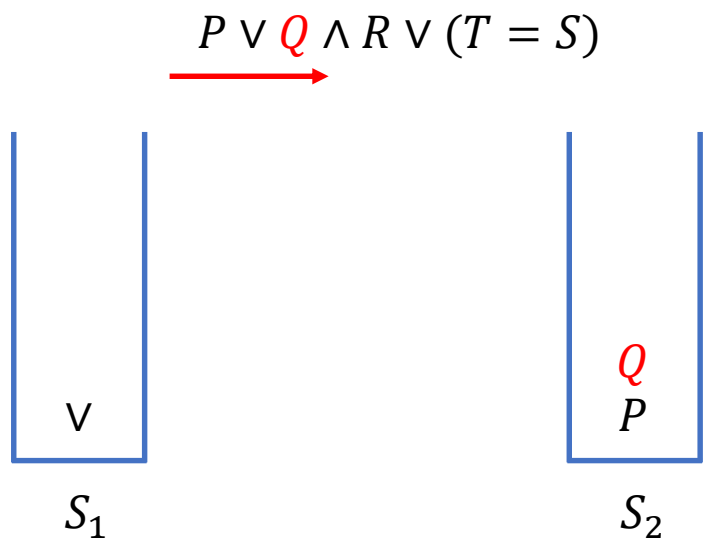
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符：
        1. 如果S_1为空，或S_1栈顶为左括号“(”，该运算符压入S_1；
        2. 若优先级高于栈顶运算符，该运算符压入S_1；
        3. 否则，S_1栈顶运算符弹出并压入S_2，重新进行2-2操作。
    3. 遇到括号



中缀转后缀

算法流程：

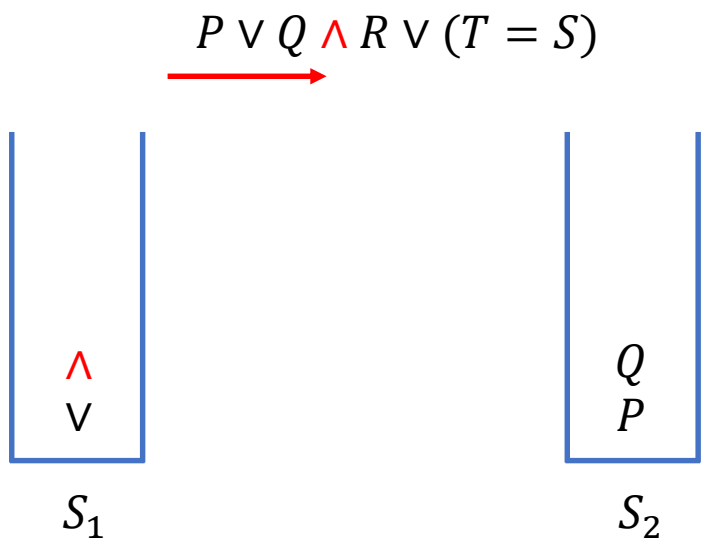
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数：直接压入S_2
    2. 遇到运算符
    3. 遇到括号



中缀转后缀

算法流程：

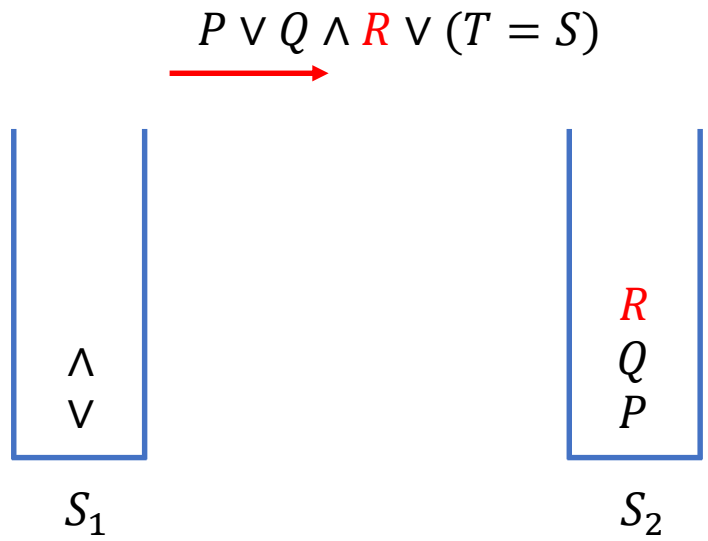
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符：
        1. 如果S_1为空，或S_1栈顶为左括号“(”，该运算符压入S_1；
        2. 若优先级高于栈顶运算符，该运算符压入S_1；
        3. 否则，S_1栈顶运算符弹出并压入S_2，重新进行2-2操作。
    3. 遇到括号



中缀转后缀

算法流程：

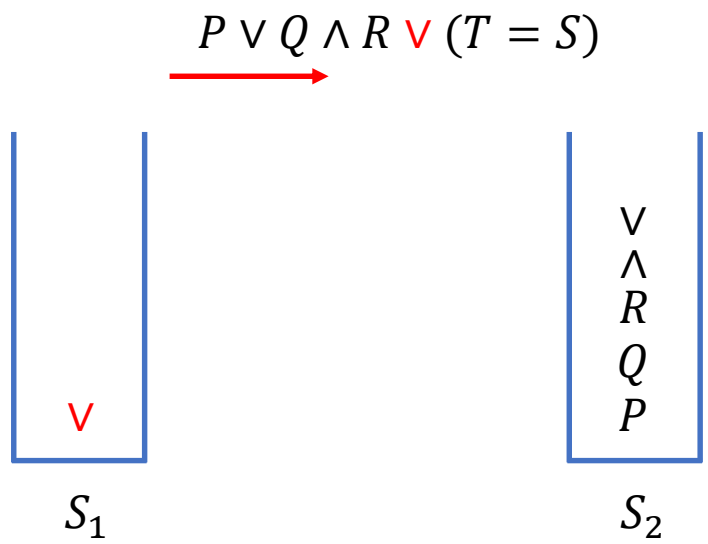
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数：直接压入S_2
    2. 遇到运算符：
    3. 遇到括号



中缀转后缀

算法流程：

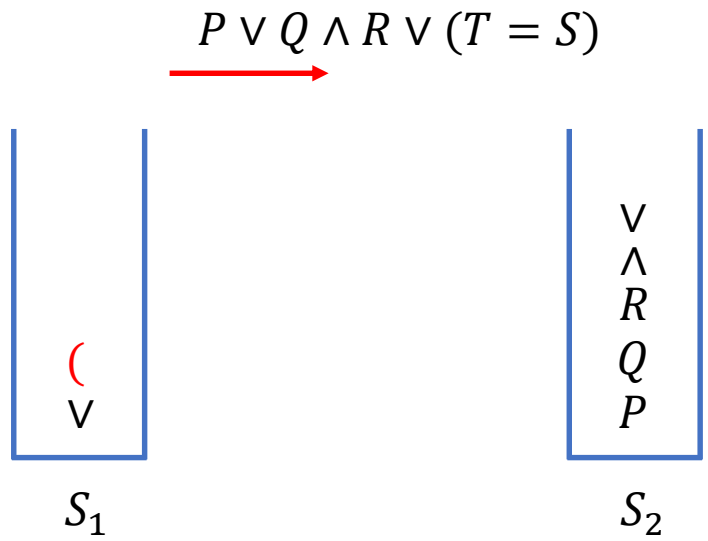
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符：
        1. 如果S_1为空，或S_1栈顶为左括号“(”，该运算符压入S_1；
        2. 若优先级高于栈顶运算符，该运算符压入S_1；
        3. 否则，S_1栈顶运算符弹出并压入S_2，重新进行2-2操作。
    3. 遇到括号



中缀转后缀

算法流程：

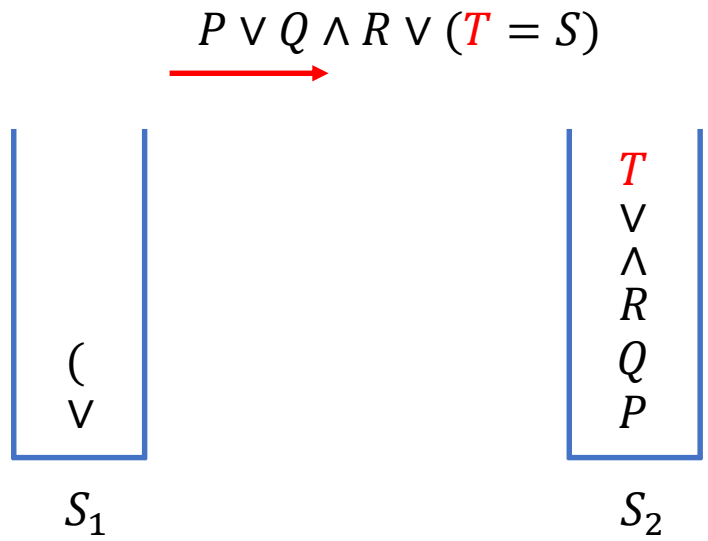
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符
    3. 遇到括号：
        1. 如果是左括号“(”，直接压入S_1；
        2. 如果是右括号“)”，则依次弹出S_1栈顶的运算符，并压入S_2，直到遇到左括号为止，此时将这一对括号丢弃。



中缀转后缀

算法流程：

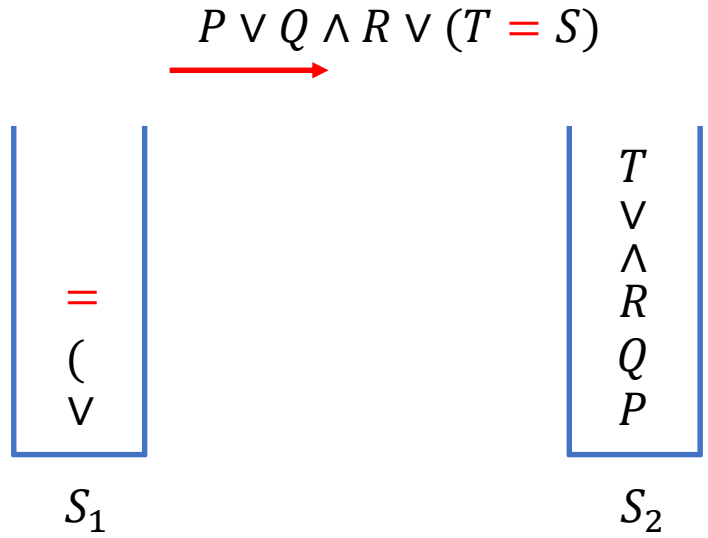
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数：直接压入S_2
    2. 遇到运算符
    3. 遇到括号



中缀转后缀

算法流程：

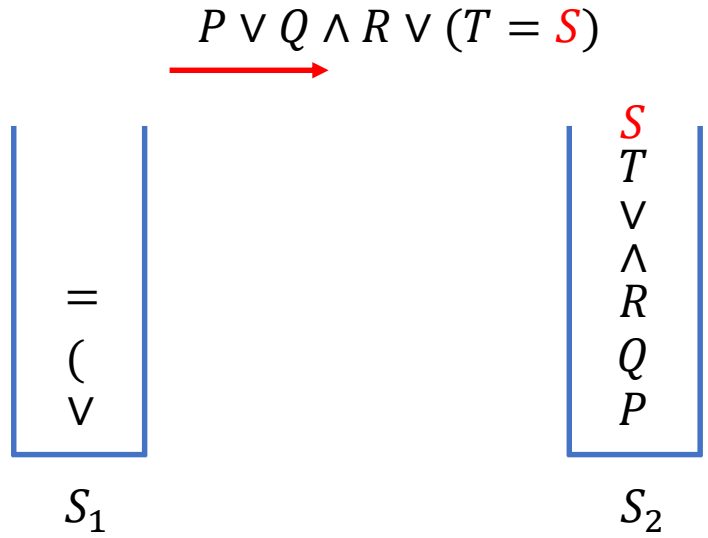
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符：
        1. 如果S_1为空，或S_1栈顶为左括号“(”，该运算符压入S_1；
        2. 若优先级高于栈顶运算符，该运算符压入S_1；
        3. 否则，S_1栈顶运算符弹出并压入S_2，重新进行2-2操作。
    3. 遇到括号



中缀转后缀

算法流程：

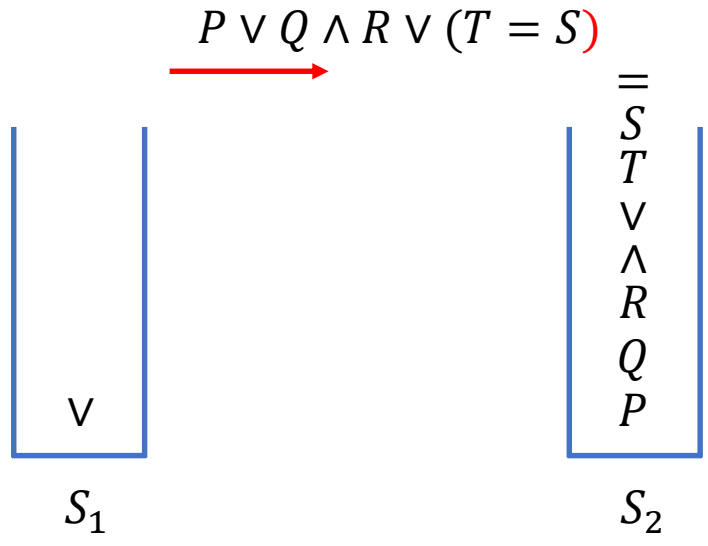
1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数：直接压入S_2
    2. 遇到运算符
    3. 遇到括号



中缀转后缀

算法流程：

1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符
    3. 遇到括号：
        1. 如果是左括号“(”，直接压入S_1；
        2. 如果是右括号“)”，则依次弹出S_1栈顶的运算符，并压入S_2，直到遇到左括号为止，此时将这一对括号丢弃。

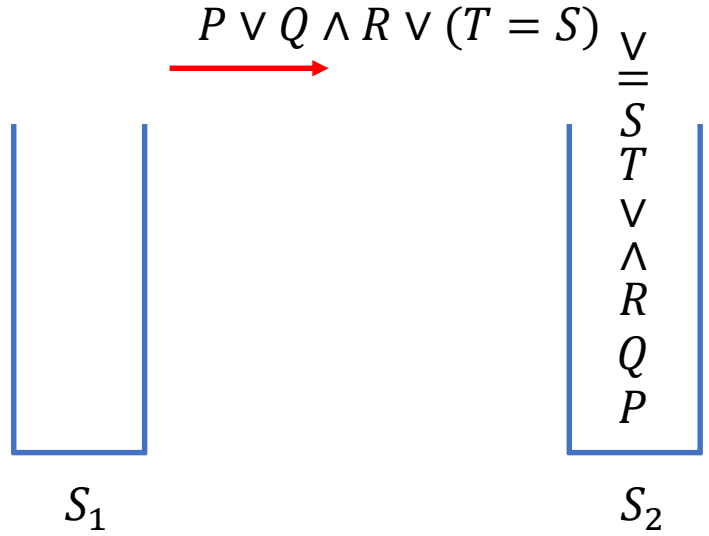


中缀转后缀

例：$P \vee Q \wedge R \vee (T = S)$ 输出：$PQR \wedge \vee TS = \vee$

算法流程：

1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从左至右扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符
    3. 遇到括号
3. 将S_1中剩余的运算符依次弹出并压入S_2；
4. 依次弹出S_2中的元素并输出，结果的逆序即为中缀表达式对应的后缀表达式。



中缀转前缀

例：$P \vee Q \wedge R \vee (T = S)$　　输出：$\vee\vee P \wedge QR = TS$

算法流程：

1. 初始化两个栈：运算符栈S_1，储存中间结果的栈S_2；
2. 从右至左扫描中缀表达式：
    1. 遇到操作数
    2. 遇到运算符：
        1. 如果S_1为空，或S_1栈顶为左括号“(”，该运算符压入S_1；
        2. 若优先级高于或等于栈顶运算符，该运算符压入S_1；
        3. 否则，S_1栈顶运算符弹出并压入S_2，重新进行2-2操作。
    3. 遇到括号
        1. 如果是右括号“)”，直接压入S_1；
        2. 如果是左括号“(”，则依次弹出S_1栈顶的运算符，并压入S_2，直到遇到右括号为止，此时将这一对括号丢弃。
3. 将S_1中剩余的运算符依次弹出并压入S_2；
4. 依次弹出S_2中的元素并输出，结果即为中缀表达式对应的前缀表达式。

后缀转中缀

- 为了后续说明，定义新的类型：$Expr$
  - 将每个表达式表示为一个$Expr$变量，并且拥有两个属性
    - $Expr.e$：表示表达式的字符串
    - $Expr.o$：将表达式转化为后缀表达时，最右侧的运算符
  - 举例
    - P：$Expr.e = P, Expr.o = None$
    - $\vee$：$Expr.e = \vee, Expr.o = \vee$
    - $P \vee Q$：$Expr.e = P \vee Q, Expr.o = \vee$
    - $P \wedge Q \vee R$：$Expr.e = P \wedge Q \vee R, Expr.o = \vee$

后缀转中缀

例：$PQR \wedge\vee TS =\vee$　　输出：$P \vee Q \wedge R \vee (T = S)$

算法流程：

1. 将初始表达式转化为$Expr$变量序列

如：$e_1e_2e_3e_4e_5e_6e_7e_8e_9 \triangleq PQR \wedge\vee TS = \vee$

$e_1.e = P, e_1.o = None$

$e_4.e = \wedge, e_4.o = \wedge$

后缀转中缀

例：$PQR \wedge \vee TS =\vee$　　输出：$P \vee Q \wedge R \vee (T = S)$

算法流程：

1. 将初始表达式转化为$Expr$变量序列
2. 从左至右扫描，当e_i为运算符时，考察它左侧的两个变量e_{i-1}, e_{i-2}
    1. 如果$e_{i-1}.o$的优先级小于$e_i.o$，则在e_{i-1}表达式左右加括号，即$e_{i-1}.e = (e_{i-1}.e)$
    2. 对e_{i-2}进行同样的操作
    3. 将三者合并为新的$Expr$变量s
        - $s.e = e_{i-2}.e\ \ e_i.e\ \ e_{i-1}.e$
        - $s.o = e_i.o$
3. 扫描完最后一个变量后，整个序列只剩下一个$Expr$变量，其e属性即为我们需要的中缀表示。

后缀转中缀

例：$PQR \wedge \vee TS =\vee$　　输出：$P \vee Q \wedge R \vee (T = S)$

表达式	$e_i.e$	$e_{i-1}.e$	$e_{i-2}.e$
$PQR \wedge \vee TS =\vee$	P	$None$	$None$
$PQR \wedge \vee TS =\vee$	Q	P	$None$
$PQR \wedge \vee TS =\vee$	R	Q	P
$PQ \wedge R \vee TS =\vee$	$\wedge$	R	Q
$P \vee Q \wedge RTS =\vee$	$\vee$	$Q \wedge R$	P
$P \vee Q \wedge RTS =\vee$	T	$P \vee Q \wedge R$	$None$
$P \vee Q \wedge RTS =\vee$	S	T	$P \vee Q \wedge R$
$P \vee Q \wedge R\ T = S \vee$	$=$	S	T
$P \vee Q \wedge R \vee (T = S)$	$\vee$	$T = S$	$P \vee Q \wedge R$

前缀转中缀

例：$\vee\vee P \wedge QR = TS$　　输出：$P \vee Q \wedge R \vee (T = S)$

算法流程：

1. 将初始表达式转化为$Expr$变量序列
2. 从右至左扫描，当e_i为运算符时，考察它右侧的两个变量e_{i+1}, e_{i+2}
    1. 如果$e_{i+1}.o$的优先级小于$e_i.o$，则在e_{i+1}表达式左右加括号，即$e_{i+1}.e = (e_{i+1}.e)$
    2. 对e_{i+2}进行同样的操作
    3. 将三者合并为新的$Expr$变量s
        - $s.e = e_{i+1}.e\ \ e_i.e\ \ e_{i+2}.e$
        - $s.o = e_i.o$
3. 扫描完最后一个变量后，整个序列只剩下一个$Expr$变量，其e属性即为我们需要的中缀表示。