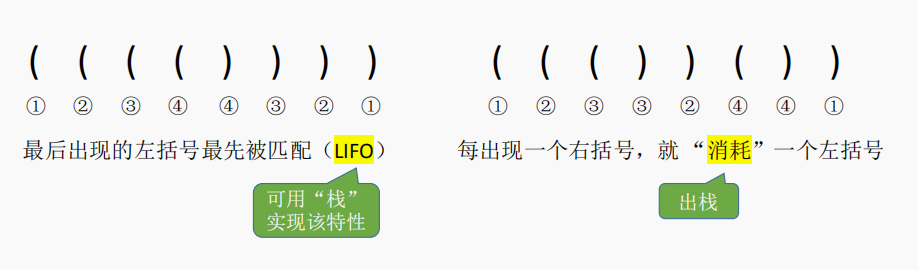
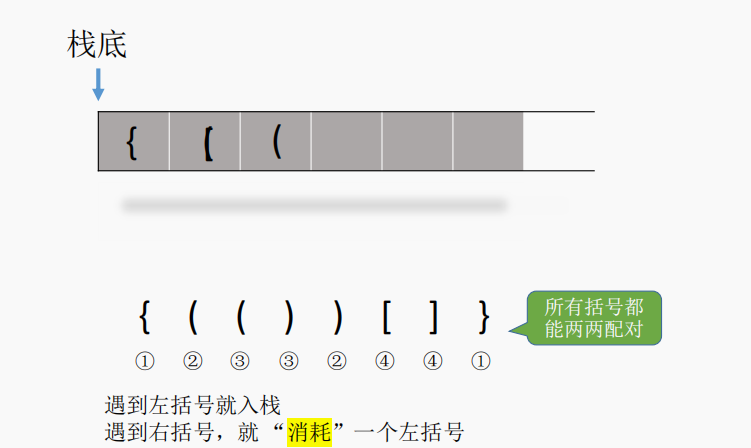
栈的应用——括号匹配

括号匹配问题:



**最后出现的左括号最先被匹配**（**LIFO**）---可用“**栈**”实现该特性

每出现一个**右括号**，就 “**消耗**”一个**左括号**---**出栈**



原理：遇到**左括号**就**入栈**；遇到**右括号**，就 “**消耗**”一个**左括号**

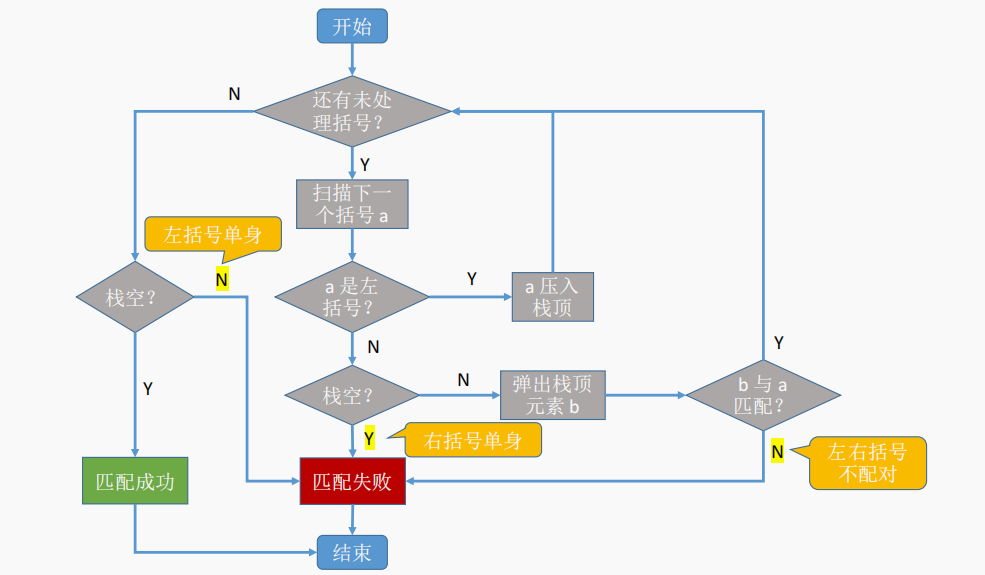
情况：

（1）所有括号都能**两两配对**

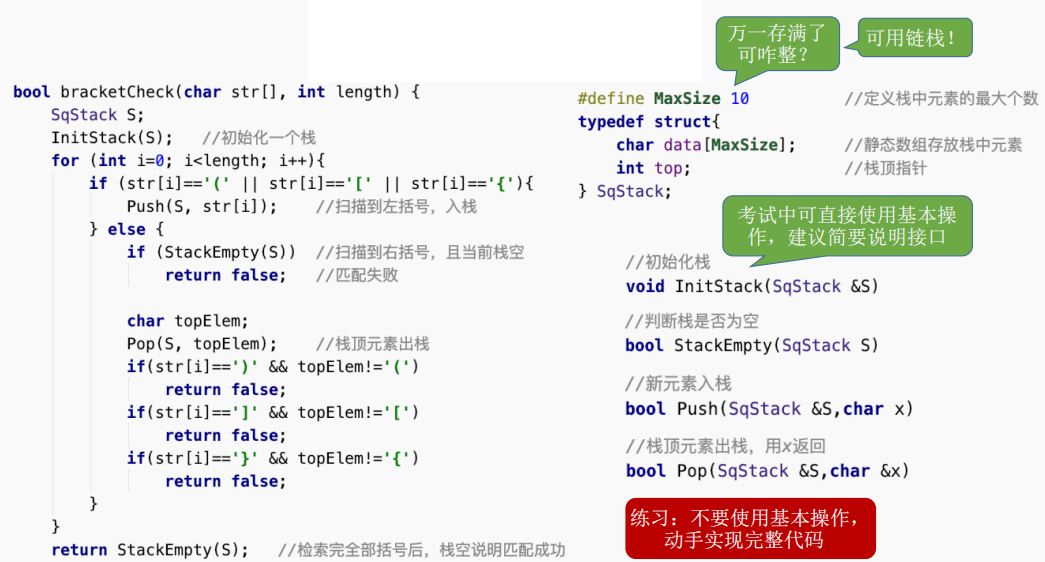
（2）当前扫描到的右括号与**栈顶左括号不匹配**

（3）扫描到**右括号且栈空**，——**右括号单身**

（4）**处理完**所有括号后，**栈非空**——**左括号单身**



算法实现：



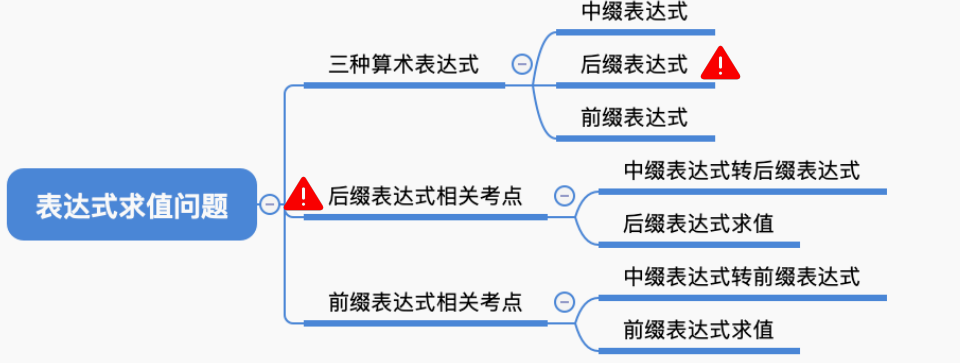
用栈实现括号匹配：

依次扫描**所有字符**，遇到**左括号入栈**，遇到**右括号**则**弹出栈顶**元素**检查是否匹配**。

匹配失败情况：

①左括号单身②右括号单身③左右括号不匹配

栈的应用——表达式求值



算数表达式：

由三个部分组成：**操作数**、**运算符**、**界限符**

（**界限符**是必不可少的，反映了计算的**先后顺序**）

一个灵感：可以**不用界限符**也能**无歧义地表达运算顺序**

Reverse Polish notation（逆波兰表达式 = 后缀表达式）

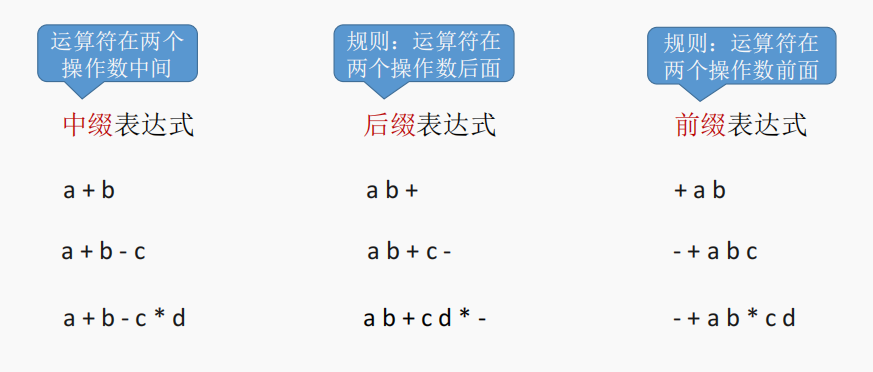
Polish notation（波兰表达式 = 前缀表达式）

中缀、后缀、前缀表达式：

**中缀**表达式：**运算符**在两个操作数**中间**

**后缀**表达式：**运算符**在两个操作数**后面**

**前缀**表达式：**运算符**在两个操作数**前面**

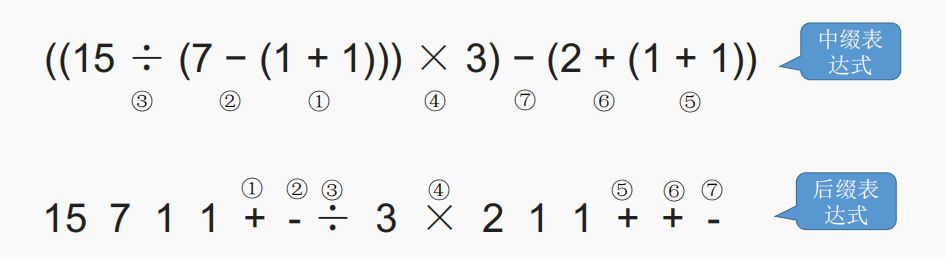


**中缀表达式转后缀表达式**（**手算**）

① **确定中缀**表达式中**各个运算符的运算顺序**

② 选择下一个**运算符**，按照**「左操作数 右操作数 运算符」**的方式**组合成一个新的操作数**

③ 如果**还有运算符没被处理**，就**继续 ②**

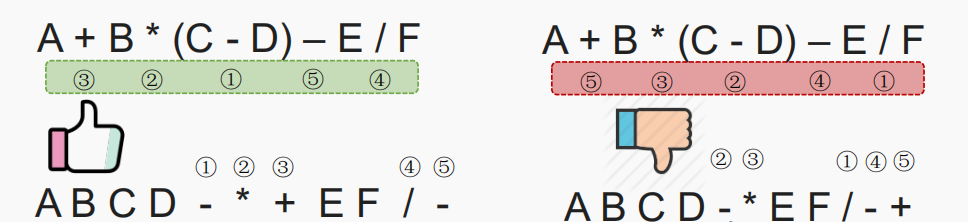


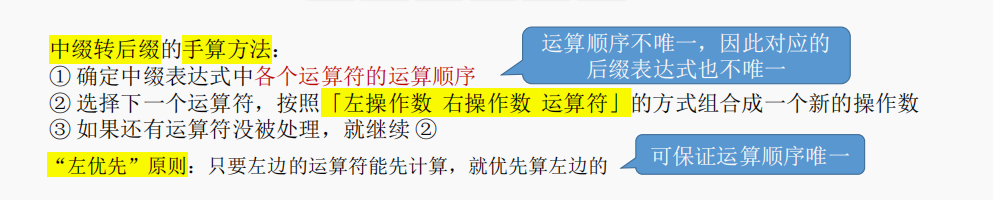
**“左优先”原则**：

由于中缀表达式运算顺序不唯一，因此对应的后缀表达式也不唯一

“**左优先**”原则：只要左边的运算符能先计算，就优先算左边的

保证手算和机算结果相同





**中缀表达式转后缀表达式**（机算）：

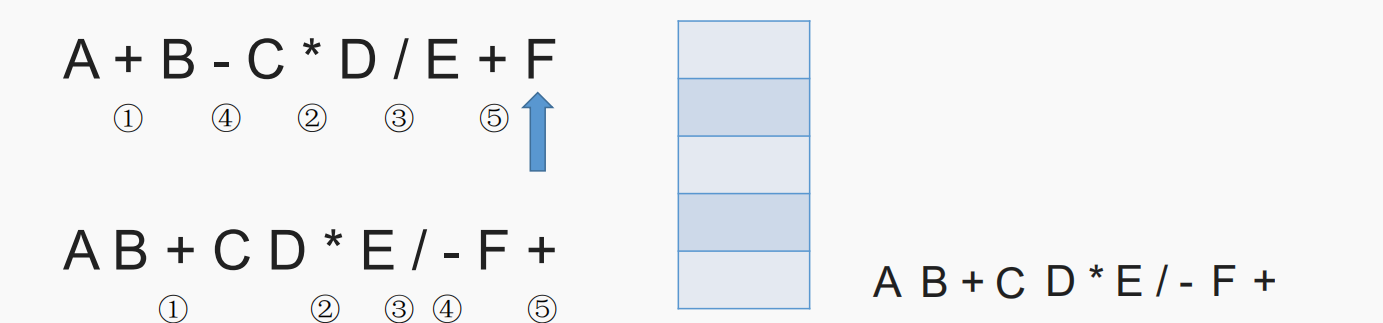
初始化一个**栈**，用于**保存**暂时**还不能确定运算顺序的运算符**。

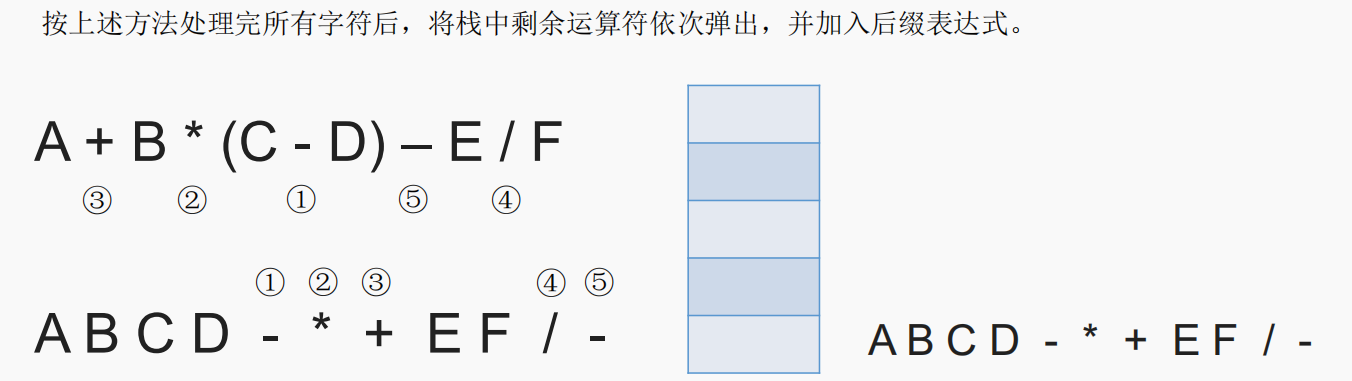
**从左到右**处理各个元素，直到末尾。可能遇到三种情况：

① 遇到**操作数**。直接加入**后缀表达式**。

② 遇到**界限符**。**遇到“(”直接入栈**；**遇到“)”**则**依次弹出栈内运算符**并**加入后缀表达式，直到弹出“(”为止**。注意：“(”不加入后缀表达式。

③ 遇到**运算符**。依次弹出栈中**优先级高于或等于当前运算符的所有运算符**，并**加入后缀表达式**，若**碰到“(” 或栈空则停止。之后再把当前运算符入栈**。

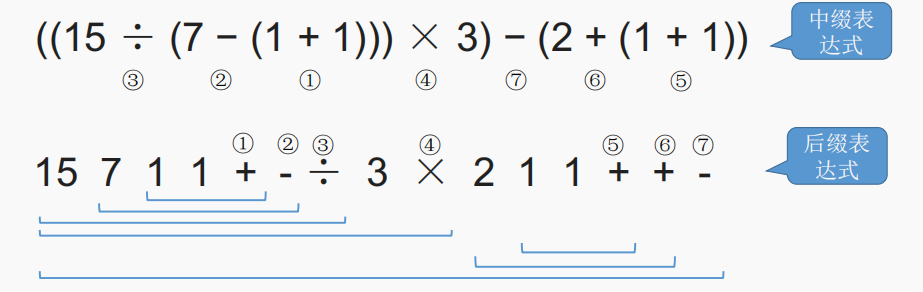




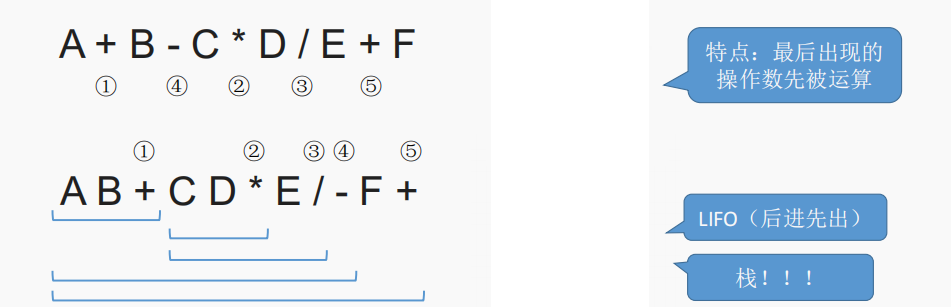
**后缀表达式的计算（手算）**：

后缀表达式的手算方法：

**从左往右扫描**，每遇到一个**运算符**，就让**运算符前面最近的两个操作数**执行对应**运算**，**合体为一个操作数**（注意：两个操作数的左右顺序）



特点：特点：最后出现的操作数先被运算---LIFO（后进先出）---栈



**后缀表达式的计算**（机算）：

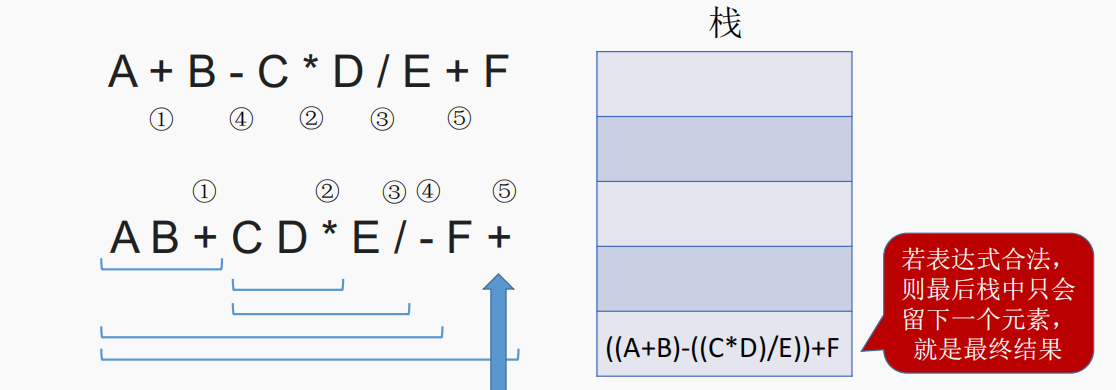
**用栈实现**后缀表达式的计算：

①从左往右扫描下一个元素，直到处理完所有元素

②若扫描到**操作数**则**压入栈**，并回到①；否则执行③

③若扫描到**运算符**，则**弹出两个栈顶元素**，执行相应**运算**，**运算结果压回栈顶**，回到①

（注意：**先出栈**的是“**右操作数**”）--若表达式合法，则最后栈中只会留下一个元素， 就是最终结果。



**中缀表达式转前缀表达式**（手算）：

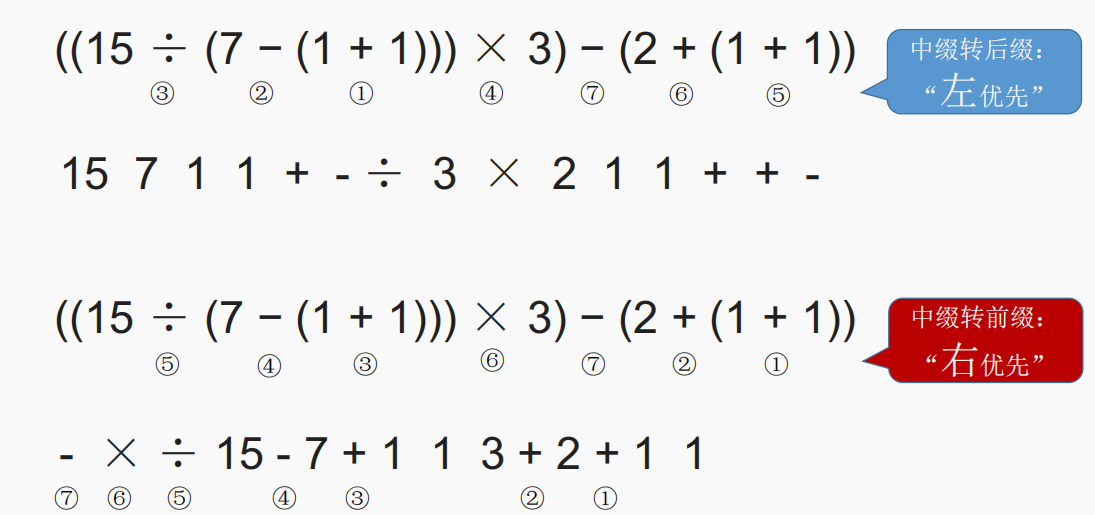
中缀转前缀的手算方法：

① 确定中缀表达式中各个运算符的**运算顺序**

② 选择**下一个运算符**，按照「**运算符 左操作数 右操作数**」的方式**组合**成一个**新的操作数**

③ 如果还有运算符没被处理，就继续 ②

“**右优先**”原则：只要右边的运算符能先计算，就优先算右边的



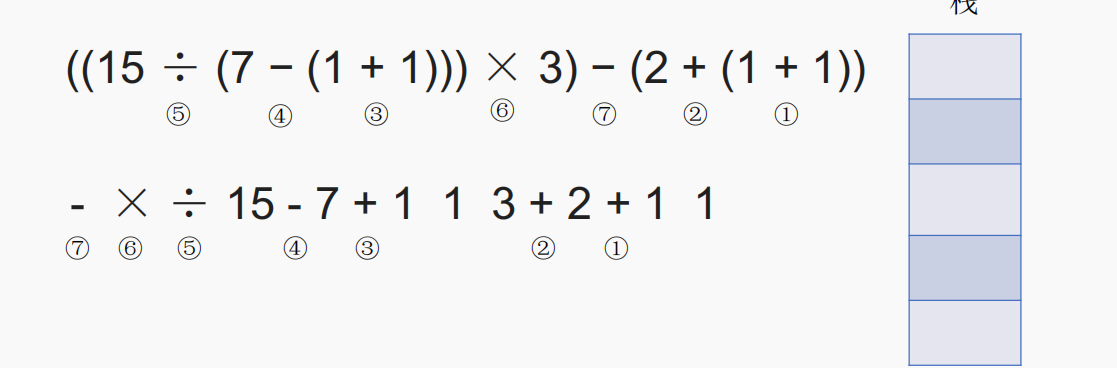
**前缀表达式的计算**

用栈实现前缀表达式的计算：

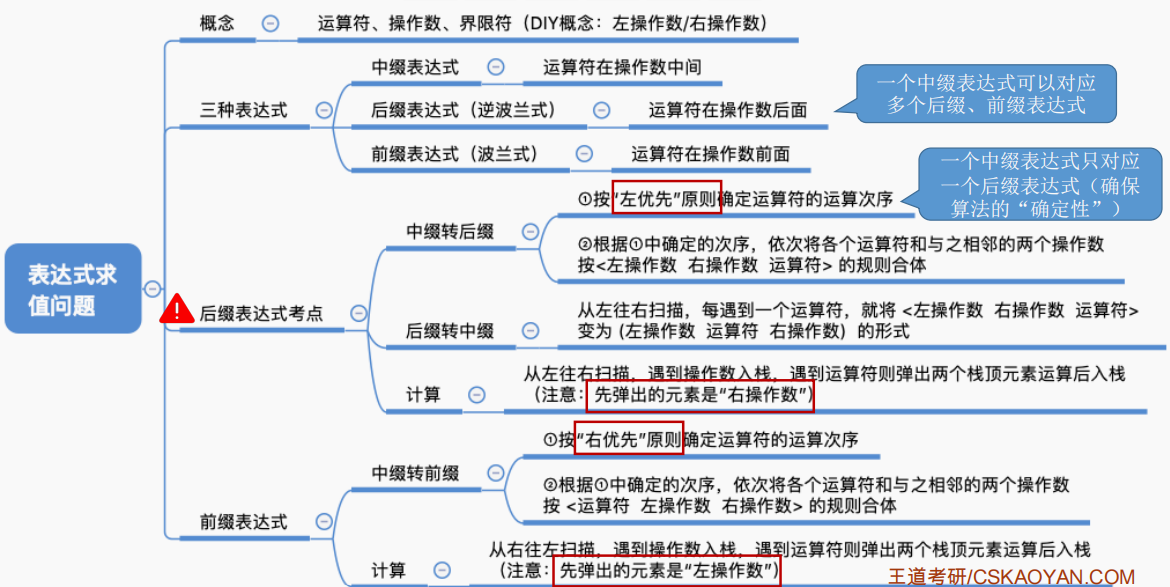
①从右往左扫描下一个元素，直到处理完所有元素

②若扫描到**操作数**则**压入栈**，并回到①；否则执行③

③若扫描到**运算符**，则**弹出两个栈顶元素**，执行相应**运算**，运算结果**压回栈顶**，回到①



总结：

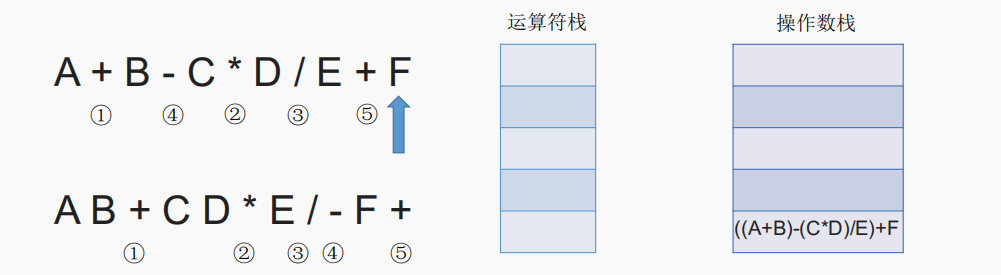


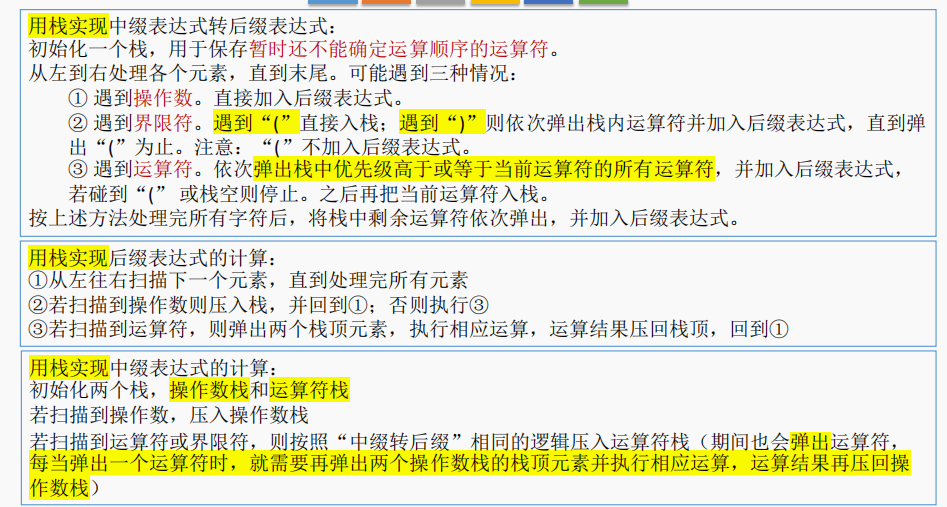
中缀表达式的计算（用栈实现） ：（中缀转后缀+后缀表达式求值）

初始化两个栈，**操作数栈**和**运算符栈**

若扫描到**操作数**，**压入操作数栈**

若扫描到**运算符**或**界限符**，则按照“**中缀转后缀**”**相同的逻辑压入运算符栈**（期间也会弹出运算符，每当弹出一个运算符时，就需要再弹出两个操作数栈的栈顶元素并执行相应运算，运算结果再压回操作数栈）





栈的应用——递归

函数调用的特点：**最后被调用的函数最先执行结束**（LIFO）

函数调用时，需要用一个栈存储：

① 调用返回地址

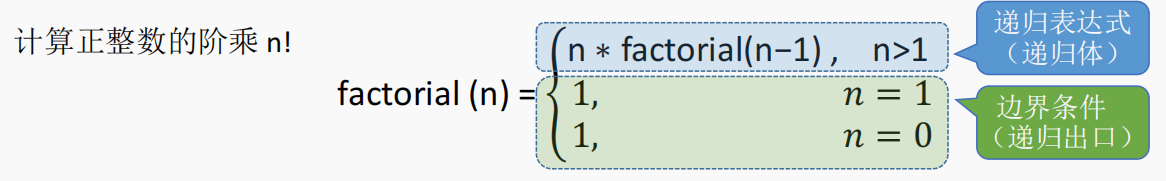
② 实参

③ 局部变量

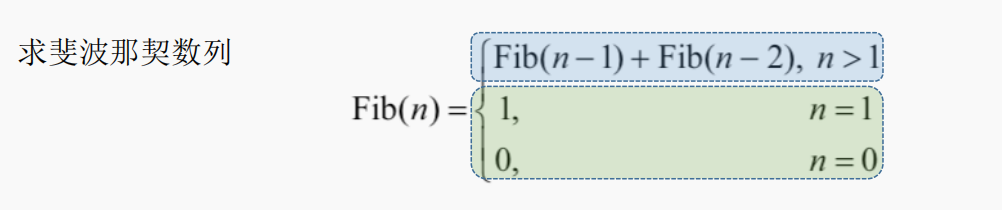
栈在递归中的应用：

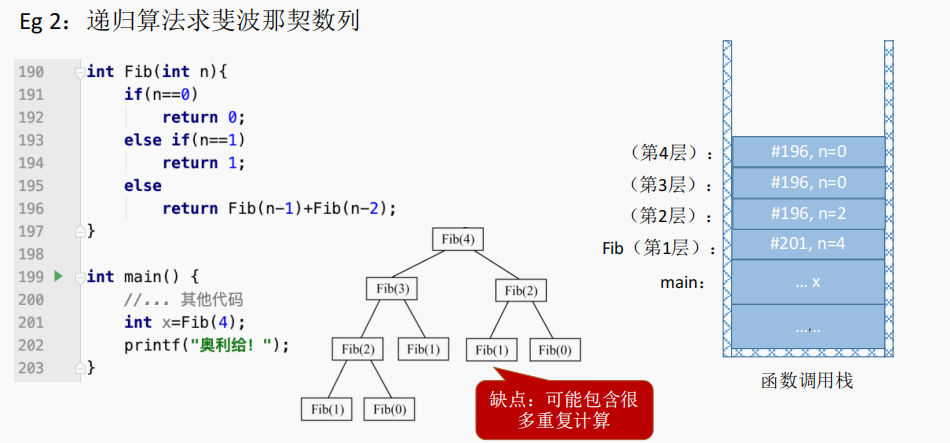
适合用“**递归**”算法解决：可以把原始问题转换为**属性相同**，但**规模较**小的问题

（1）递归算法求阶乘

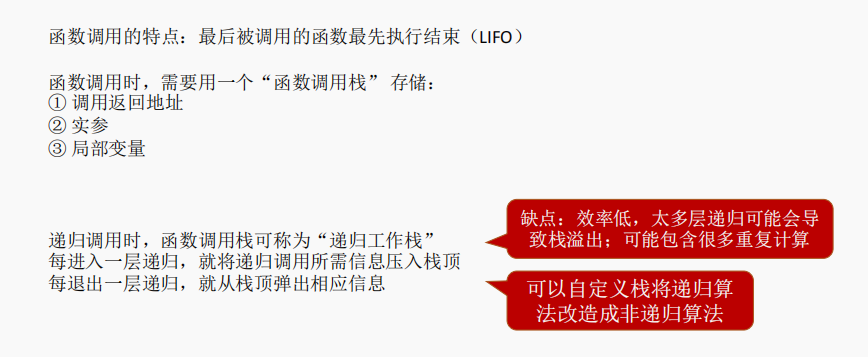






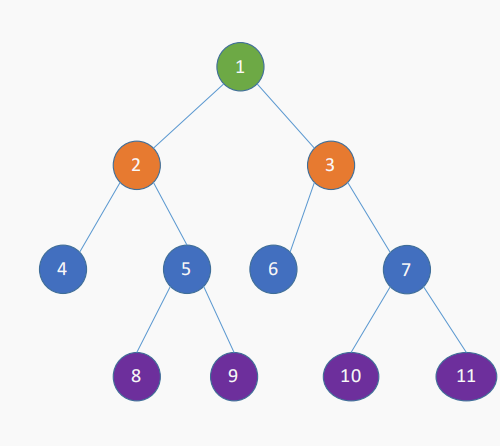


总结：

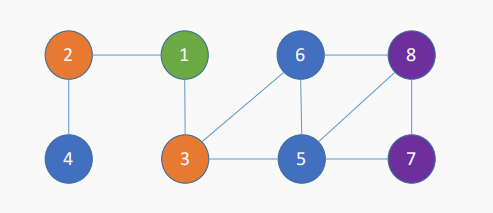


队列的应用

队列应用——树的层次遍历



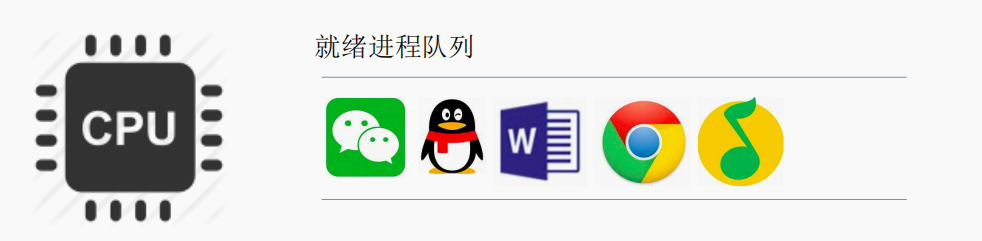
队列应用——图的广度优先遍历



队列在操作系统中的应用

**多个进程争抢着使用有限的系统资源**时，**FCFS**（First Come First Service，**先来先服务**）是一种常用策略。

1. CPU资源的分配



1. 打印数据缓冲区

