中国科学技术大学
University of Science and Technology of China

程序设计II
第7讲 递归

计算机学院 黄章进
zhuang@ustc.edu.cn

内容

- 例题：汉诺塔问题
- 例题：放苹果 1664
- 例题：神奇的口袋 2755
- 例题：八皇后 2754
- 例题：逆波兰表达式 2694

递归的基本思想

- 总体思想：将待求解问题的解看作输入变量x的函数f(x)，通过寻找函数g，使得f(x) = g(f(x-1))，并且已知f(0)的值，就可以通过f(0)和g求出f(x)的值。
- 这样一个思想也可以推广到多个输入变量x,y,z等，x-1也可以推广到 x - x1，即f(x) = g(f(x-x1))，只要递归朝着出口的方向走就可以了。

递归与枚举

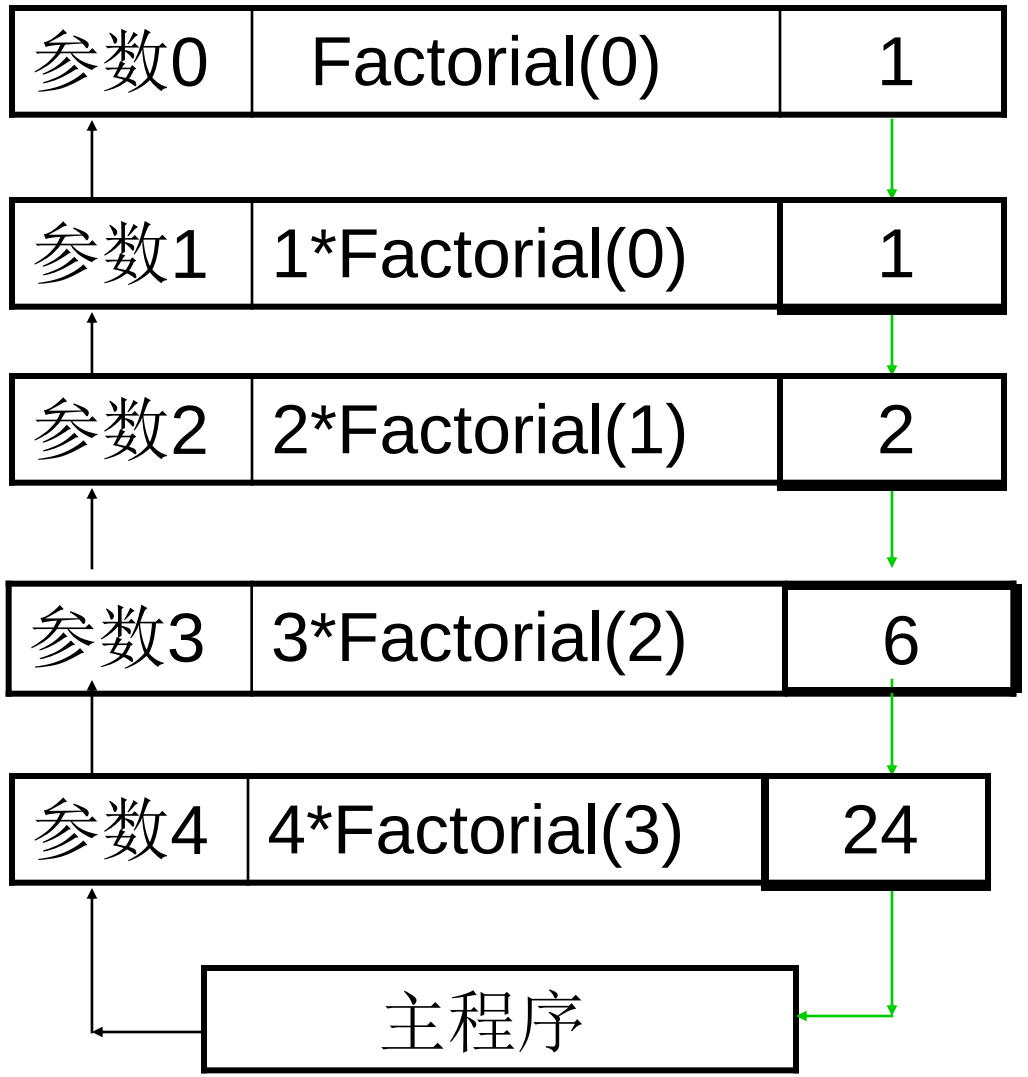
- 枚举: 把一个问题划分成一组子问题, 依次对这些子问题求解
  - 子问题之间是横向的、同类的关系
- 递归: 把一个问题逐级分解成子问题
  - 子问题与原问题之间是纵向的、同类的关系
  - 语法形式上：在一个函数的运行过程中，调用这个函数自己
    - 直接调用：在fun()中直接执行fun()
    - 间接调用：在fun1()中执行fun2()；在fun2()中又执行fun1()

求阶乘的递归程序


```
int Factorial(int n)
{
   if (n == 0)
        return 1;
    else
        return  n * Factorial(n - 1);
}
```

阶乘的栈


```
int Factorial(int n)
{
   if (n == 0)
        return 1;
    else
        return
   n * Factorial(n - 1);
}
```



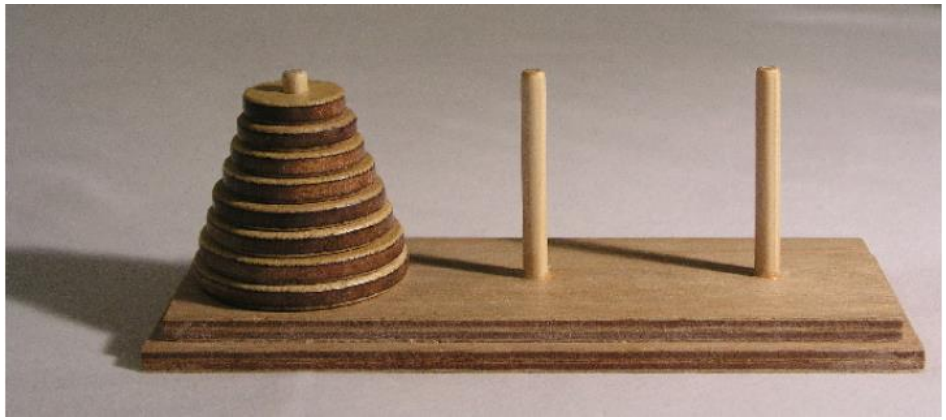
递归解决问题的关键

1) 找出递推公式
2) 找到递归终止条件

- 注意事项：由于函数的局部变量是存在栈上的，如果有体积大的局部变量，比如数组，而递归层次又可能很深的情况下，也许会导致栈溢出，因此可以考虑使用全局数组或动态分配数组。

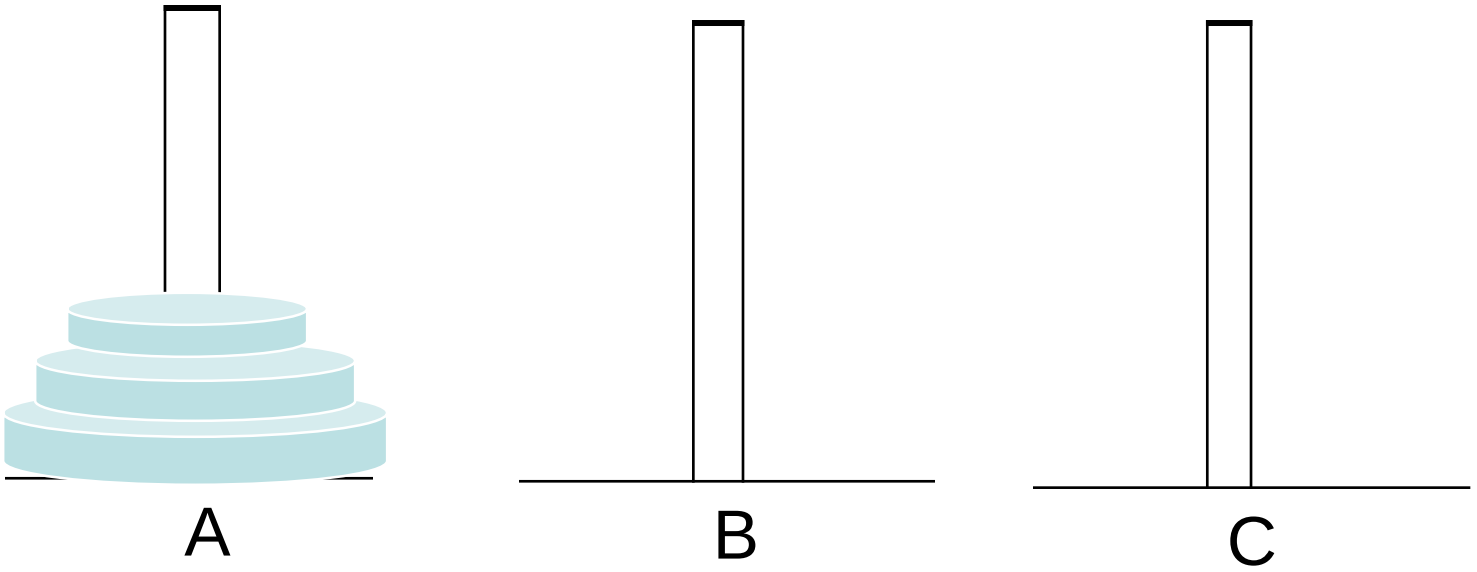
汉诺塔问题

- 法国数学家爱德华·卢卡斯于1883年发明的，给定一个由8个圆盘组成的塔(Tower of Hanoi)，这些圆盘按照大小递减的方式套在三根桩柱中的一根上



- 将整个塔移动到另一根桩柱上，每次只能移动一个圆盘，且较大的圆盘在移动过程中不能放置在较小的圆盘上面

汉诺塔问题



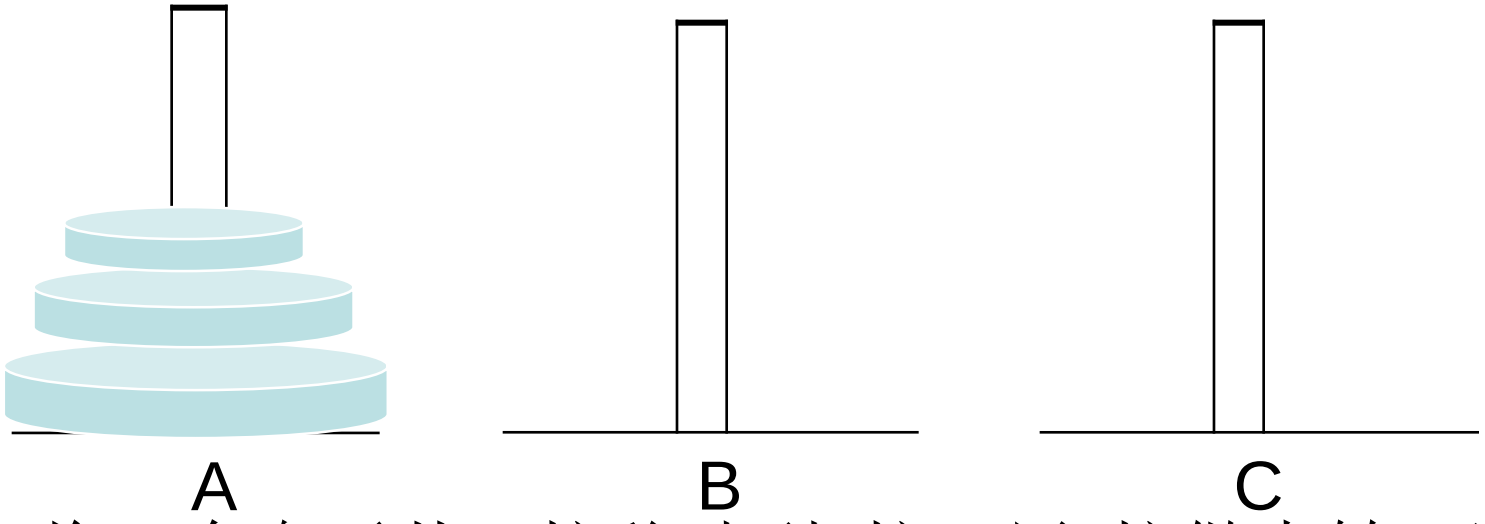
- 以C柱为中转，将盘从A柱移动到B柱上，一次只能移动一个盘，而且大盘不能压在小盘上面

汉诺塔问题

Please input disc number:
3

The solution is:
A->B
A->C
B->C
A->B
C->A
C->B
A->B

汉诺塔问题



- 将 n 个盘子从 a 柱移动到b柱，用c柱做中转,可以分以下3步实现：
  - 先将n-1个盘子，以b为中转，从a柱移动到c柱
  - 将一个盘子从a移动到b
  - 将c柱上的n-1个盘子，以a为中转，移动到b柱
- 终止条件：n == 1时，直接移动盘子即可，不用递归

汉诺塔问题


```
//将 n 个盘子从 a 柱移动到b柱，用c柱做中转
void Hanoi(int n, char a, char b, char c)
{
    if( n == 1 ) {
        printf("%c->%c\n", a, b);
        return ;
    }

    //先将n-1个盘子，以b为中转，从a柱移动到c柱，
    Hanoi( n-1, a, c, b);
    //将一个盘子从a移动到b
    printf("%c->%c\n", a, b);
    //将c柱上的n-1个盘子，以a为中转，移动到b柱
    Hanoi( n-1, c, b, a);
}
```

汉诺塔问题


```
#include <stdio.h>

int main()   {
    int N;
    printf("Please input disc number: \n" );
    scanf("%d", &N);
    printf("The solution is: \n" );
    Hanoi( N, 'A', 'B', 'C');
    return 0;
}
```

汉诺塔问题

- 时间复杂度？

  $T(n) = 2 * T(n-1) + 1$

  $T(n) = 2^n - 1$

- 婆罗贺摩塔(Tower of Brahma)：64个纯金圆盘堆放在三座钻石做成的方尖塔上，当僧侣们把这些金圆盘从一座方尖塔移动到另一座方尖塔上后，塔将坍塌，世界毁灭
  - 如果每秒移动一次，需要5845.54亿年以上，因为$2^{64} - 1 = 18446744073709551615$

例题：放苹果

- 问题描述：
  - 把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用K表示）5，1，1和1，5，1 是同一种分法。
- 输入：
  - 第一行是测试数据的数目t（0 <= t <= 20）。以下每行均包含二个整数M和N，以空格分开。1<=M，N<=10。
- 输出：
  - 对输入的每组数据M和N，用一行输出相应的K。

放苹果

- 输入样例

1

7 3

- 输出样例

8

算法

- 设f(a, d) 为a个苹果，d个盘子的放法数目，则先对d作讨论
- 如果d > a，必定至少有d-a个盘子是空的，去掉这些空盘子对摆放苹果方法数目不产生影响，即

```
if (d > a)
    f(d, a) = f(a, a);
```

算法

- 当d <= a时，不同的放法可以分成两类：
  - 至少一个盘子空着：假定该情况下的放法数目为g(a, d)
  - 所有盘子都有苹果，即没有盘子空着：假定该情况下的放法数目为 k(a, d)

则显然： f(a, d) = g(a, d) + k(a, d)

- 然而：

  g(a,d) = f(a, d-1)

  k(a,d) = f(a-d, d)

- 因此： f(a, d) = f(a, d-1) + f(a-d, d)

递推公式

递推公式：

```
if(d > a)
        f(a,d) = f(a,a);
else
        f(a,d) = f(a,d-1)+f(a-d,d);
```

终止条件是什么呢？

终止条件

- 终止条件：
  - 当d=1时，所有苹果都必须放在一个盘子里，所以返回1；
  - 当没有苹果可放a=0时，所有盘子都是空的，这当然是一种放法，所以也返回1
- 递归的两条路：
  - 第一条：d会逐渐减少，终会到达出口d==1;
  - 第二条：a会逐渐减少，因为d>a时，我们会return f(a,a)，所以终会到达出口a==0.

代码


```
int f(int a, int d){
    if( d == 1 || a ==0)
        return 1;

    if(d > a )
        return f(a, a);

    return f(a, d-1) + f(a-d, d);
}
```

例题：神奇的口袋

- 问题描述
  - 有一个神奇的口袋，总的容积是40，用这个口袋可以变出一些物品，这些物品的总体积必须是40。John现在有n个想要得到的物品，每个物品的体积分别是a1，a2……an。John可以从这些物品中选择一些，如果选出的物体的总体积是40，那么利用这个神奇的口袋，John就可以得到这些物品。现在的问题是，John有多少种不同的选择物品的方式。

神奇的口袋

- 输入
  - 输入的第一行是正整数n (1 <= n <= 20)，表示不同的物品的数目。接下来的n行，每行有一个1到40之间的正整数，分别给出a_1，a_2……a_n的值。
- 输出
  - 输出不同的选择物品的方式的数目。

- 输入样例

3
20
20
20

- 输出样例

3

解题思路

- 给出小于40的正整数a1,a2,…,an，问凑出和为40有多少种方法？
- 设f(a1,a2,…,an,40)为用a1,a2,…,an凑出40的方法数，考虑凑数的方法分为两类，用到a1和用不到a1,则有

$$f(a1,a2,\ldots,an,40) = f(a2,\ldots,an,40-a1) + f(a2,\ldots,an,40)$$

实现

- 考虑到在实现上函数的参数个数要统一，所以把a1,a2,…,an放到一个数组中，用numbers[] 存放所有的整数，n表示数组中整数的个数，ith表示从数组中第ith个数开始凑数(ith前面的不用），sum表示要凑出的总数，则

```
f(numbers, n, ith, sum) =
        f(numbers, n, ith+1, sum-numbers[ith])//用到ith
    + f(numbers, n, ith+1, sum)                  //不用ith
```

终止条件

f(numbers, n, ith, sum) =

f(numbers, n, ith+1, sum-numbers[ith])//用到ith

+ f(numbers, n, ith+1, sum) //不用ith

- 终止条件
  - 1) 如果sum == 0，则应该返回 1 （一个数都不取，也是一种取法）
  - 2) 如果sum < 0，则应该返回 0 （没法凑出负数）
  - 3) 如果 ith == n ，则应该返回 0 (所有的数都用完了，和还没有凑够）

2755 神奇的口袋

- 在实现上有两种考虑：
  - 因为numbers, n与递归没有太大关系，可以考虑使用全局量；
  - 如果不使用全局量，而通过参数传递numbers和n，可以增强递归函数的独立性

```
int numbers[50], n; //将numbers, n设为全局量
int count(int ith, int sum){
    if(sum == 0)          return 1;
    if(ith==n || sum <0)        return 0;
    return count(ith+1, sum-numbers[ith])  + count(ith+1, sum);
}
```

- 将 n 个数放入数组 numbers后， count(0,40); 就能求出结果

例题：八皇后

- 问题描述
  - 会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将8个皇后放在棋盘上（有8＊8个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。
  - 对于某个满足要求的8皇后的摆放方法，定义一个皇后串a与之对应，即$a=b_1b_2...b_8$，其中b_i为相应摆法中第i行皇后所处的列数。已经知道8皇后问题一共有92组解（即92个不同的皇后串）。
  - 给出一个数b，要求输出第b个串。串的比较是这样的：皇后串x置于皇后串y之前，当且仅当将x视为整数时比y小。

八皇后

- 输入
  - 第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数b(1 <= b <= 92)
- 输入样例

2

1

92

- 输出
  - 输出有n行，每行输出对应一个输入。输出应是一个正整数，是对应于b的皇后串。
- 输出样例

15863724

84136275

思路

- 八皇后问题可用八重循环解决。但是N皇后问题呢?
- 递归可以用来实现任意多重循环
  - 如果有多个变量，每个变量有各自的取值范围，要覆盖这些变量值的全部组合，可以用递归实现。
  - 此题没有什么直接的递推关系，写递归就是为了起到多重循环的作用

2754 八皇后


```
#include <stdio.h>
#include <math.h>
#include <string.h>
const int QueenNum = 8;
int anResult[92][QueenNum];//存放找到的摆法
int anQueen[QueenNum]; //纪录当前正在尝试的摆法
int nFoundNum = 0;//当前已经找到多少种摆法
void Queen( int n); //摆放第n行以及以后的皇后(行号从0开始算)
int main()
{
   int n,b;
   Queen(0);
   scanf("%d",&n);
   while ( n -- )
   {
      scanf("%d", &b);
      for( int i = 0; i < QueenNum; i ++ )
         printf("%d", anResult[b-1][i]+1);
      printf("\n");
   }
   return 0;
}
```

2754 八皇后


```
//摆放第n行以及以后的皇后(行号从0开始算)
void Queen( int n) {
   if( n == QueenNum ) { //前QueenNum行都成功摆好了，记下摆法
      memcpy(anResult[nFoundNum++], anQueen, sizeof(anQueen));
      return ;
   }
   for( int i = 0; i < QueenNum; i ++ ) { //尝试第n行所有位置
      int j;
      for( j = 0; j < n; j ++ ) {
         //对每个位置，判断是否和已经摆好的皇后冲突
         if( i == anQueen[j] ||
               abs( i - anQueen[j]) == abs(n - j ))
            break;
      }
      if( j == n ) {
         //如果没有冲突，则第n行摆好了，记下来，再摆第n+1行
         anQueen[n] = i;
         Queen(n+1);
      }
   }
}
```

例题：波兰表达式

- 问题描述
  - 波兰表达式是一种把运算符前置的算术表达式，例如普通的表达式2 + 3的逆波兰表示法为+ 2 3。波兰表达式的优点是运算符之间不必有优先级关系，也不必用括号改变运算次序，例如(2 + 3) * 4的波兰表示法为* + 2 3 4。本题求解波兰表达式的值，其中运算符包括+ - * /四个。

逆波兰表达式

- 输入
  - 输入为一行，其中运算符和运算数之间都用空格分隔，运算数是浮点数。
- 输出
  - 输出为一行，表达式的值。可直接用printf("%f\n", v)输出表达式的值v。

- 输入样例

* + 11.0 12.0 + 24.0 35.0

- 输出样例

1357.000000

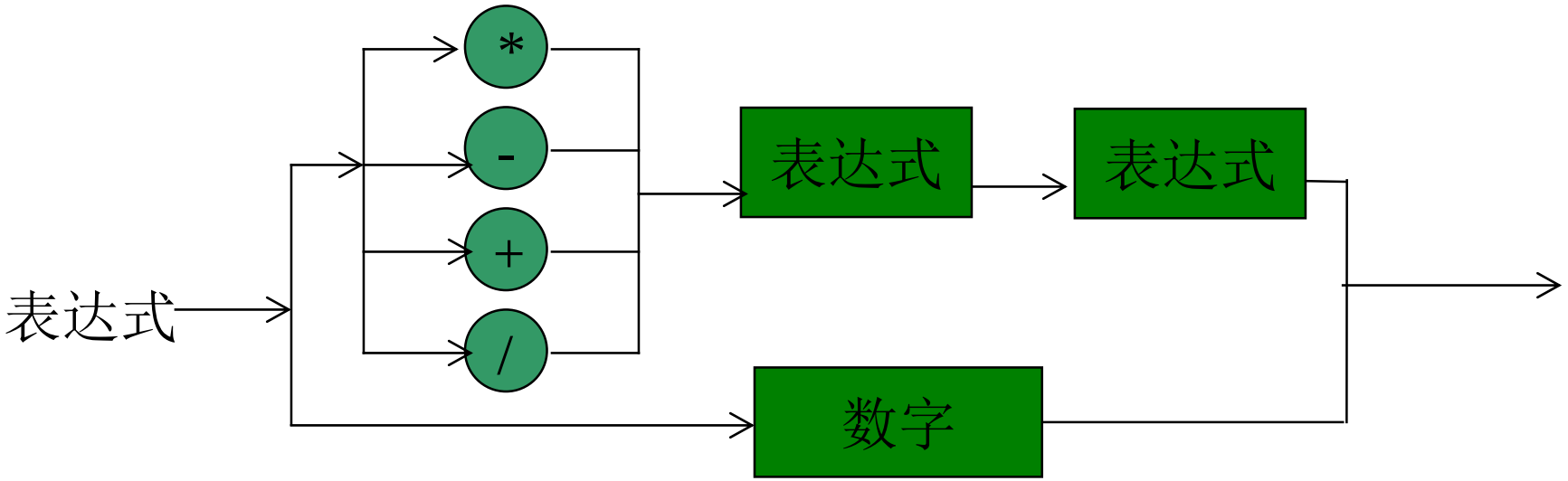
波兰表达式

- 1920年，波兰科学家扬·武卡谢维奇就发明了一种**不需要括号**的表示法，可以用来表示一个计算表达式。即将操作符号写在操作数之前，也就是前缀表达式，即波兰式（Polish Notation, PN）
- 比如2 + 3 * (5 - 1)这个表达式的前缀表达式为+ 2 * 3 - 5 1来表示。

波兰表达式

- 比如2 + 3 * (5 - 1)这个表达式的前缀表达式为+ 2 * 3 - 5 1来表示。
- 阅读这个表达式需要从左至右读入表达式，如果一个操作符后面跟着两个操作数时，则计算，然后将结果作为操作数替换这个操作符和两个操作数，重复此步骤，直至所有操作符处理完毕。
- 从左往右依次读取，直到遇到- 5 1，做计算后，将表达式替换为+ 2 * 3 4，然后从左往右再次读取，直到遇到* 3 4，做计算后将表达式替换为+ 2 12，然后从左往右依次读取，读到+ 2 12，计算得到14，到此结束。

逆波兰表达式的递归定义



思路

- 根据波兰表达式的定义进行递归求解
- 在递归函数中，针对当前的输入有5种情况
  - 输入是常数，则表达式的值就是常数
  - 输入是+，则表达式的值是再继续读入两个表达式并计算它们的值，然后将它们的值相加
  - 输入是-
  - 输入是*
  - 输入是/

2964 逆波兰表达式


```
1.  #include <stdio.h>
2.  #include <math.h>
3.  double exp(){
4.      char a[10];
5.      scanf("%s", a);
6.      switch(a[0]){
7.          case'+': return exp( ) + exp( );
8.          case'-': return exp( ) - exp( );
9.          case'*': return exp( ) * exp( );
10.         case'/': return exp( ) / exp( );
11.         default: return atof(a); //字符串转浮点数
12.     }
13. }
14. int main()
15. {
16.     double ans;
17.     ans=exp();
18.     printf("%f", ans);
19. }
```