西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计
Programming in C

1011014

主讲：姜学锋，计算机学院

调用函数 - 调用形式

1、递归算法策略

2、分治算法策略

- 1．递归法
- 在问题求解时，有时将一个不能或不好直接求解的大问题转化为一个或几个与原问题相似的小问题来解决，再把这些小问题进一步分解成更小的小问题来解决，如此分解，直至每个小问题都可以直接解决，在逐步求解小问题后，再返回得到大问题的解，这种方法称为递归法。

4.11 函数应用程序举例

- 递归的运行效率往往很低，会消耗额外的时间和存储空间。但递归也有长处，它能使一个蕴含递归关系且结构复杂的程序简洁精炼，只需要少量的步骤就可描述解题过程中所需要的多次重复计算，所以大大的减少了代码量。

- 基本思想
- 找出递归子结构性质（原问题的解包含了子问题的解）、用子问题的解来递归定义原问题的解、找出递归终止条件。

4.11 函数应用程序举例

- 递归算法设计的关键在于：找出递归关系式和边界条件（即递归终止条件）。
- 递归关系就是使问题向边界条件转化的过程，所以递归关系必须能使问题越来越简单，规模越来越小。因此递归算法设计，通常有以下三个步骤：
- ①分析问题，得出递归关系式。
- ②设置边界条件，控制递归。
- ③设计函数，确定参数。

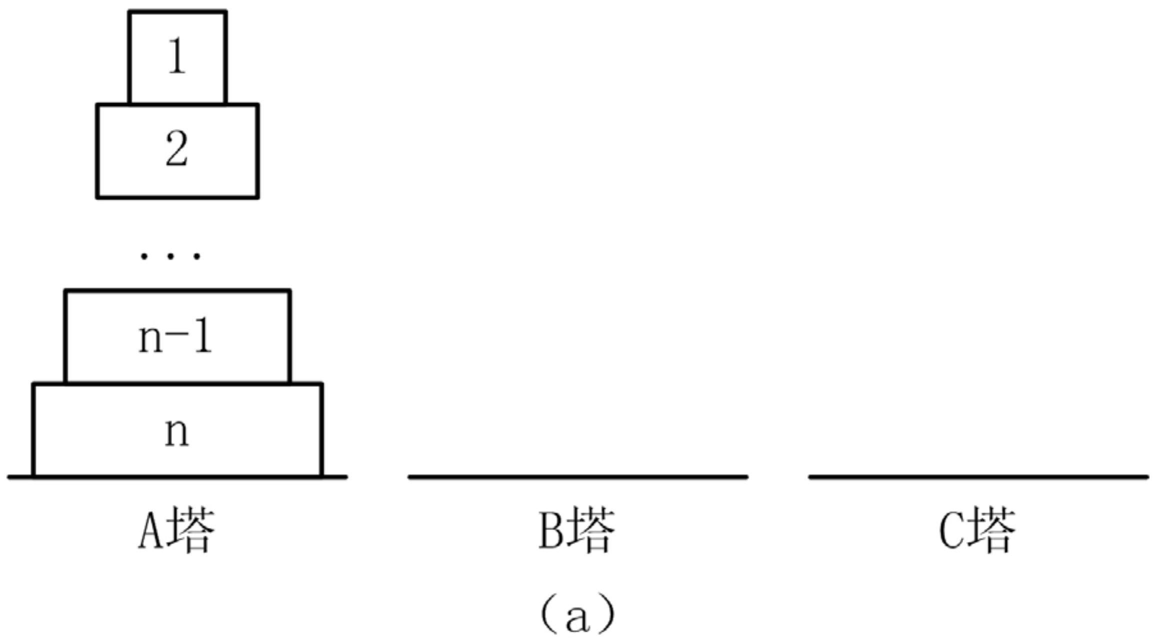
- 编程模式：
- 一般地，递归算总是可以表达成如下的形式：

```
if(最简单情形) {
   直接得到最简单情形的解
}
else {
   将原始问题转化为简单一些（降低问题规模）的一个或多个子问题
   以递归方式逐个求解上述子问题
   以合理有效的方式将这些子问题的解组装成原始问题的解
}
```

【例4.12】

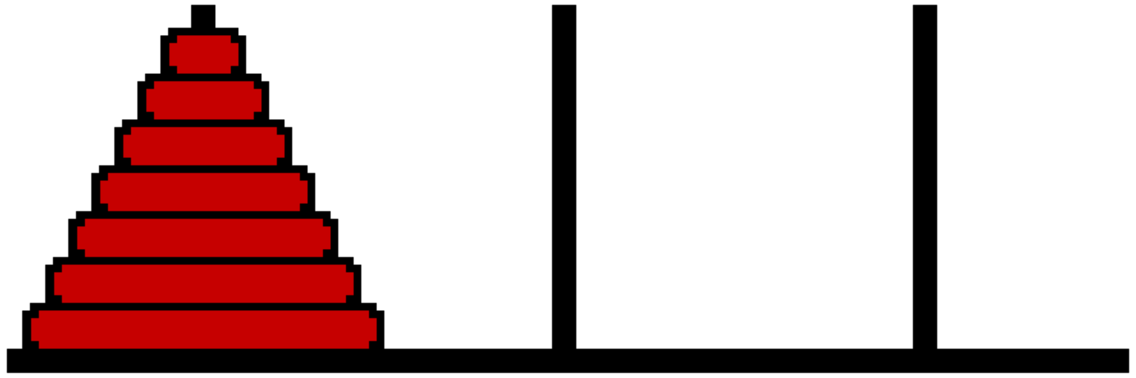
Hanoi塔问题：如图所示，设有A、B、C三个塔座，在塔座A上共n有个圆盘，这些圆盘自上而下，由小到大地叠在一起。现要求将塔座A上这叠圆盘移到塔座C上，并仍按同样顺序放置，且在移动过程中遵守规则：①每次只能移动1个圆盘；②不允许将较大的圆盘压在较小的圆盘之上；③移动中可以使用A、B、C任一塔座上。编出程序显示移动步骤。

图4.14 Hanoi塔



4.11 函数应用程序举例

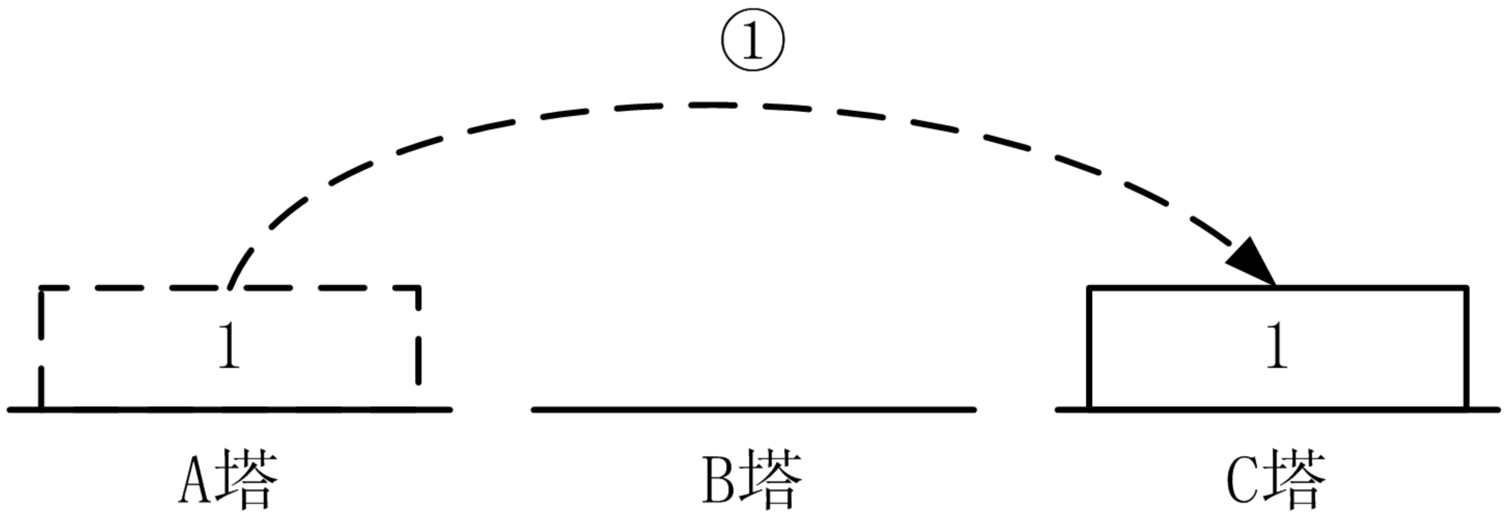
Hanoi塔示意图



例题分析

①若有一块圆盘时，如图（b）所示，则直接A－>C；

图4.14 Hanoi塔求解示意

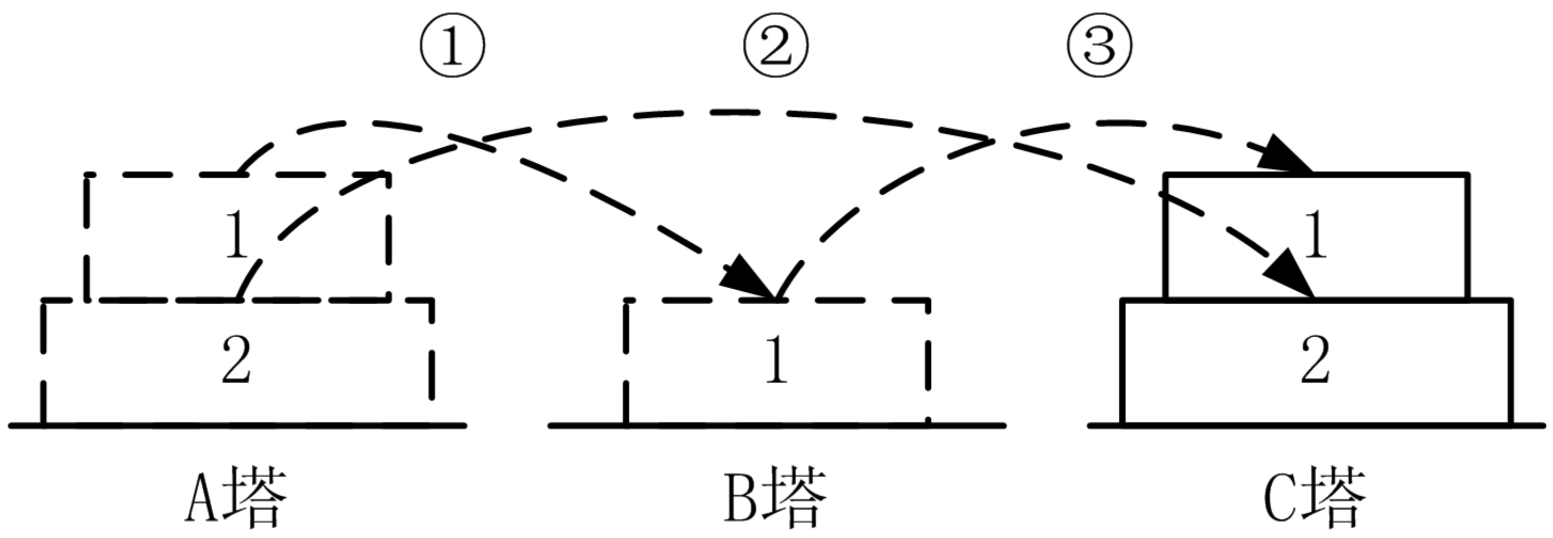


（b）

例题分析

②若有两块圆盘时，如图（c）所示，则#1块A－>B，#2块直接A－>C，#1块B－>C；

图4.14 Hanoi塔求解示意

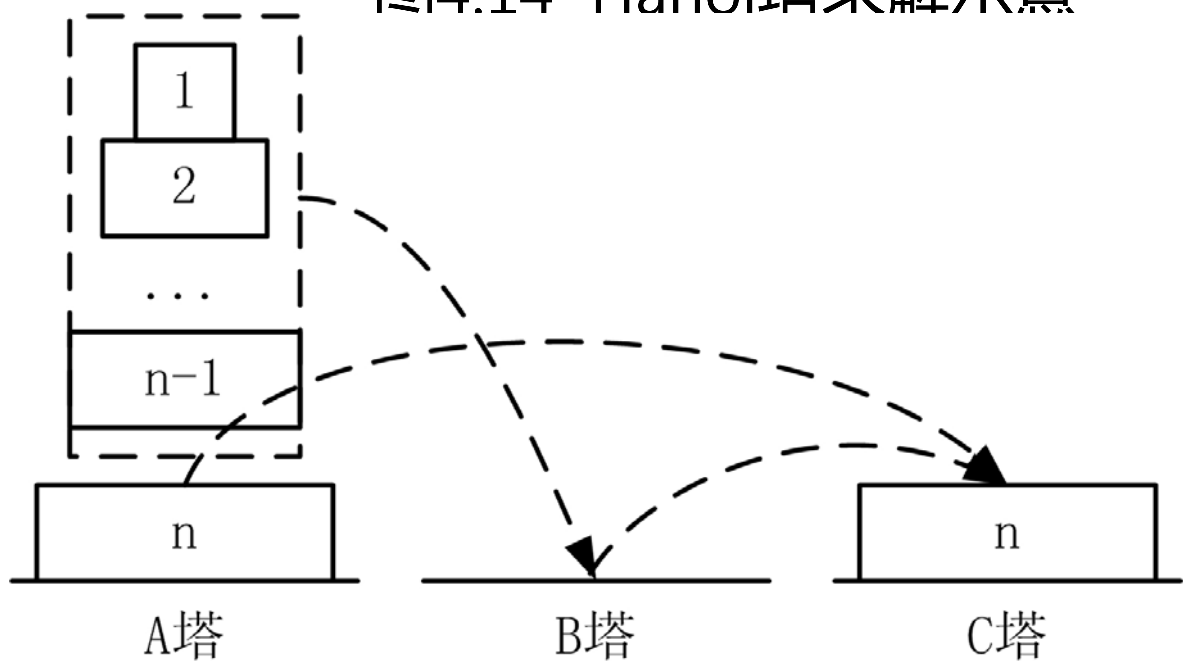


（c）

例题分析

③若有n块圆盘时，如图（d）所示，可以将上面n-1块当作"一块"，记为m，则#m块A－>B，#n块直接A－>C，#m块B－>C；

图4.14 Hanoi塔求解示意



（d）

例题分析

显然这是符合递归求解的问题，其递归关系式可以如下描述：

$$\begin{cases} f_1(A,B,C) = A \to C \\ f_2(A,B,C) = A \to B, A \to C, B \to C \\ f_n(A,B,C) = f_{n-1}(A,C,B), A \to C, f_{n-1}(B,A,C) \end{cases}$$

例题分析

边界条件为n=1。可以定义Hanoi函数：

函数原型： void Hanoi(int n, char A, char B, char C);

返 回 值： 无

函数参数：n：圆盘数目，A：起始塔，B：中间塔，C：目标塔

例4.12

```c
#include<stdio.h>
void Hanoi(int n, char A, char B, char C)
{
  if (n==1) printf("%c->%c ",A,C); //只有一个盘子直接A->C
  else {
    Hanoi(n-1, A, C, B); //上面n-1块盘子A->B
    printf("%c->%c ", A, C); //第n块盘子直接A->C
    Hanoi(n-1, B, A, C); //B塔n-1块盘子B->C
  }
}
int main(void)
{
  int n;
  printf("input n:");
  scanf("%d",&n);
```

例4.12

```
    Hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

4.11 函数应用程序举例

例4.12

```
#include<stdio.h>
```

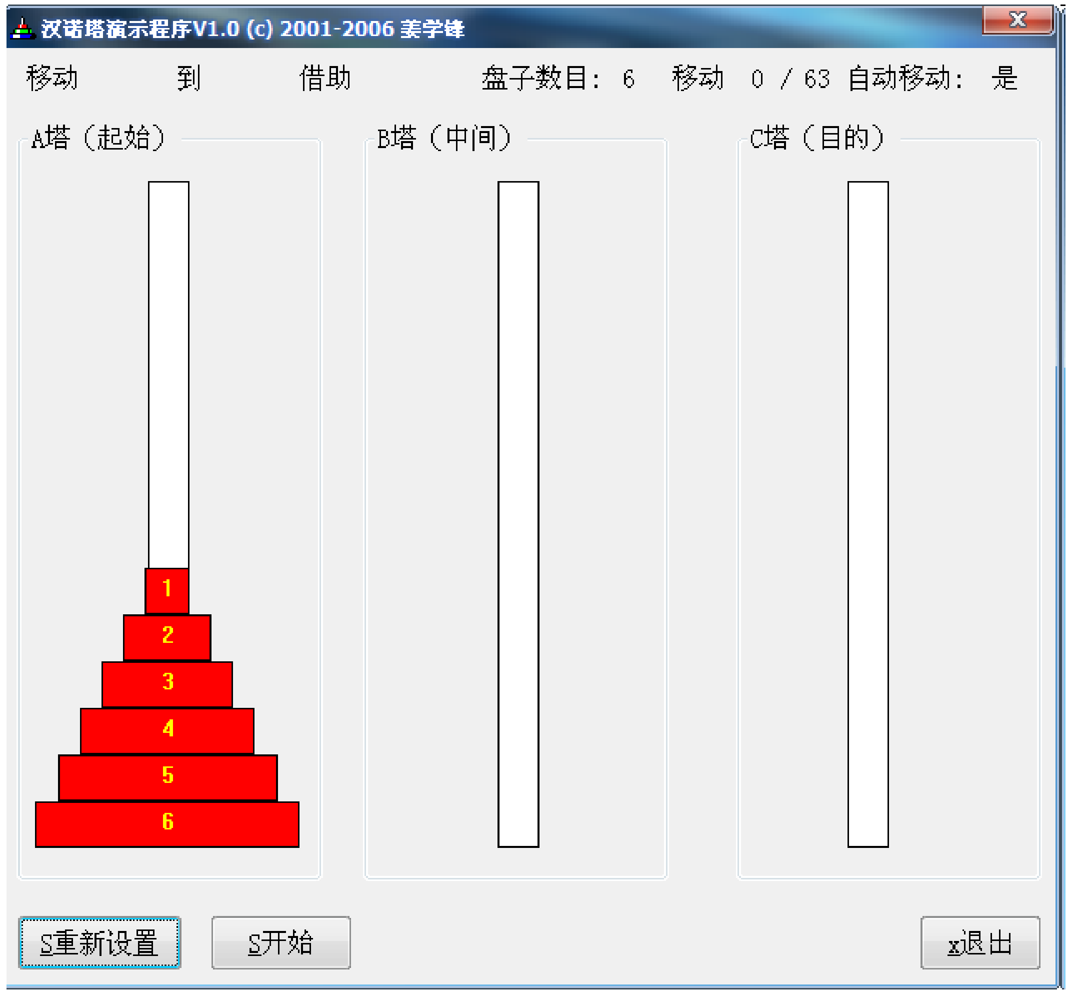
程序运行屏幕

```
A->C A->B C->B A->C B->A B->C A->C
```

```
input n:3↙
```

4.11 函数应用程序举例

▸ Hanoi塔程序运行动画



- 2．分治法（divide and conquer）
- 分治法即“分而治之”，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破。

- 基本思想
- 分治法的基本思想是将一个规模为n的问题分解为k（1<k≤n）个规模较小的子问题，这些子问题互相独立且与原问题相同。递归地解这些子问题，然后将各子问题的解合并得到原问题的解。

4.11 函数应用程序举例

▶编程模式

```
divide-and-conquer(P)
{
  if (|P|<=n0) adhoc(P);
  divide P into smaller subinstaces P1,P2,…,Pk;
  for (i=1,i<=k,i++)
    yi=divide-and-conquer(Pi) ;
  return merge(y1,y2,…,yk);
}
```

- 其中，|P|表示问题P的规模。n_0为一个阈值，表示当问题P的规模不超过n_0时问题已容易解出，不必再继续分解。
- adhoc(P)是该分治法中的基本子算法，用于直接解小规模的问题P。当P的规模不超过n_0时，直接用算法adhoc(P)求解。算法 merge($y_1, y_2, \cdots, y_k$) 是合并子算法，用于将P的子问题
- $P_1, P_2, \cdots, P_k$ 的解 $y_1, y_2, \cdots, y_k$ 合并为P的解。

- 分割原则
- 原问题应该分为多少个子问题才较适宜？每个子问题的规模应该怎样才为适当？这些问题很难予以肯定的回答。但人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。换言之，将一个问题分成大小相等的k个子问题的处理方法是行之有效的。许多问题可以取k=2。这种使子问题规模大致相等的做法是出自一种平衡子问题的思想，它几乎总是比子问题规模不等的做法要好。

- 用分治法设计的算法一般是递归算法。因此，分治法的计算效率通常用递归方程来进行分析。一个分治法将规模为n的问题分成k规模为n/m的子问题求解。

- 设分解阈值n_0=1，且adhoc解规模为1的问题耗费1个单位时间，再设将原问题分解为k个子问题以及用merge将k个子问题的解合并为原问题的解需要f(n)个单位时间。则该分治法所需的计算时间为

$$T(n)=\begin{cases} O(1) & n=1 \\ kT(n/m)+f(n) & n>1 \end{cases}$$

4.11 函数应用程序举例

- 通常可以用展开递归式的方法来解这类递归方程，反复代入求解得

$$T(n)=n^{\log_m k}+\sum_{j=0}^{\log_m n-1} k^j f(n/m^j)$$

- 注意，递归方程及其解只给出n等于m的方幂时T(n)的值，如果T(n)足够平滑，由n等于m的方幂时T(n)的值可以估计T(n)的增长速度。一般地，假定T(n)是单调上升的。

- 基本步骤
- 分治法在每一层递归上都有三个步骤：
- ①分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；
- ②解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；
- ③合并：将各个子问题的解合并为原问题的解。

【例7.1】

整数划分问题

将一个正整数n表示成一系列正整数之和，$n=n_1+n_2+\ldots+n_k$，其中$n_1>=n_2>=\ldots>=n_k>=1,k>=1$。正整数n的一个这种表示称为n的一个划分，正整数n的不同划分个数称为正整数n的划分数，记作p(n)。求n的不同划分个数。例如正整数6有如下11种不同的划分，所以p(6) = 11：

6；
5+1；
4+2，4+1+1；
3+3，3+2+1，3+1+1+1；
2+2+2，2+2+1+1，2+1+1+1+1；
1+1+1+1+1+1

例题分析

在本例中，如果设p(n)为正整数n的划分数，则难以找到递归关系，因此考虑增加一个自变量：在正整数n的所有不同划分中，将最大加数n_1不大于m的划分个数记作q(n,m)。则可以建立q(n,m)的如下递归关系。

（1）q(n,1)=1,n >= 1;当最大加数n_1不大于1时，任何正整数n只有一种划分形式，即n = 1 + 1 + 1 + ... +1.

（2）q(n,m) = q(n,n),m >= n; 最大加数n_1实际上不能大于n。因此，q(1,m)=1。

例题分析

在本例中，如果设p(n)为正整数n的划分数，则难以找到递归关系，因此考虑增加一个自变量：在正整数n的所有不同划分中，将最大加数n_1不大于m的划分个数记作q(n,m)。则可以建立q(n,m)的如下递归关系。

（3）q(n,n)=1 + q(n,n-1); 正整数n的划分由n_1=n的划分和$n_1 \leqslant$ n-1的划分组成。

（4）q(n,m)=q(n,m-1)+q(n-m,m),n > m >1;正整数n的最大加数n_1不大于m的划分由 n_1 = m的划分和$n_1 \leqslant$ m-1 的划分组成。

例题分析

即：

$q(n,m) = 1$	$n = 1, m = 1$
$q(n,m) = q(n,n)$	$n = 1, m = 1$
$q(n,m) = 1 + q(n,n-1)$	$n = m$
$q(n,m) = q(n,m-1) + q(n-m,m)$	$n > m > 1$

正整数n的划分数$p(n) = q(n,n)$。

例4.21

```
#include <stdio.h>
int p(int n,int m)
{
  if (n<1 || m<1) return 0;
  if (n==1 || m==1) return 1;
  if (n<m) return p(n,n);
    if (n==m) return p(n,m-1)+1;
  return p(n,m-1)+p(n-m,m);
}
int main()
{
  int n;
  scanf("%d",&n);
  printf("%d\n",p(n,n));
  return 0;
```

CP程序设计