第四章　递归算法

前面已经介绍了关于递归调用这样一种操作，而递归程序设计是C++语言程序设计中的一种重要的方法，它使许多复杂的问题变得简单，容易解决了。递归特点是：函数或过程调用它自己本身。其中直接调用自己称为直接递归，而将A调用B，B以调用A的递归叫做间接递归。

【例1】给定n（n>=1）,用递归的方法计算1+2+3+4+...+(n-1)+n。

【算法分析】

本题可以用递归方法求解，其原因在于它符合递归的三个条件：

(1)本题是累加问题：s(n)=s(n-1)+n;

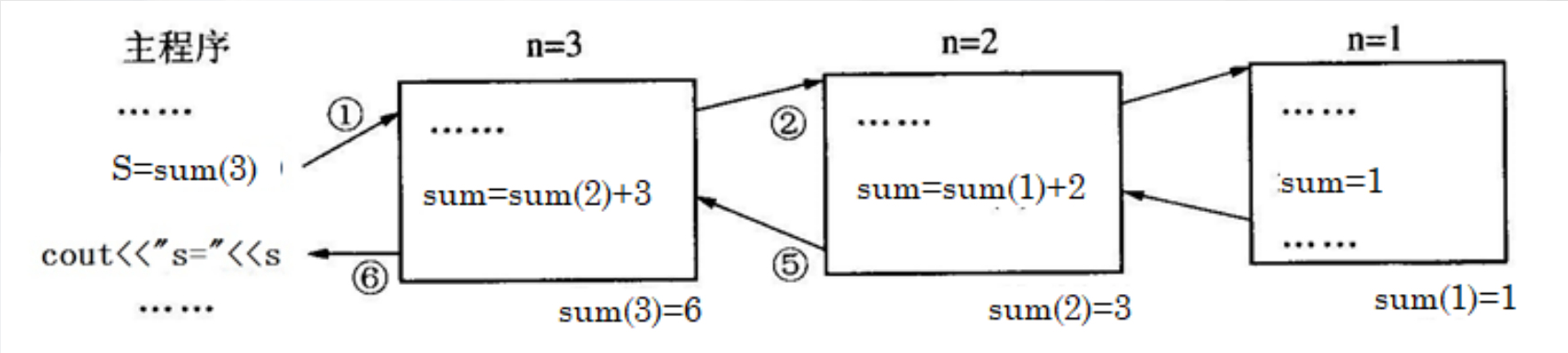
(2)给定n,所以是有限次的递归调用；

(3)结束条件是当n=1，则s=1。

【参考程序】

```
#include<iostream>
using namespace std;
int sum(int);                                   //递归函数
int main()
{
      int t;
      cin>>t;                                   //输入t的值
      cout<<"s="<<sum(t)<<endl;       //计算1到t的累加和，输出结果
}
int sum(int n)
{
      if (n==1) return 1;
      else return (sum(n-1)+n);                 //调用下一层递归
}
```

运行程序，当T=5时，输出结果：S=15，其递归调用执行过程是：（设T=3）



递归调用过程，实质上是不断调用过程或函数的过程，由于递归调用一次，所有子程序的变量（局部变量、变参等）、地址在计算机内部都有用特殊的管理方法——栈（先进后出）来管理，一旦递归调用结束，计算机便开始根据栈中存储的地址返回各子程序变量的值，并进行相应操作。

【例2】 设有N个数已经按从大到小的顺序排列，现在输入X，判断它是否在这N个数中，如果存在则输出：“YES” 否则输出“NO”。

【算法分析】

该问题属于数据的查找问题，数据查找有多种方法，通常方法是：顺序查找和二分查找，当N个数排好序时，用二分查找方法速度大大加快。

二分查找算法：

（1） 设有N个数，存放在A数组中，待查找数为X，用L指向数据的高端，用R指向数据的低端，MID指向中间：

（2） 若X=A[MID] 输出 “YES”；

（3）若X<A[MID]则到数据后半段查找：R不变，L=MID+1，计算新的MID值，并进行新的一段查找；

（4） 若X>A[MID]则到数据前半段查找：L不变，R=MID-1，计算新的MID值，并进行新的一段查找；

（5）若L>R都没有查找到，则输出“NO”。

该算法符合递归程序设计的基本规律，可以用递归方法设计。

【参考程序】

```
#include<iostream>
#include<cstdlib>
using namespace std;
void search(int[],int,int,int);
int main()                                          //主程序
{     int n,x,a[100];
      cin>>n>>x;
      for (int k=1;k<=10;k++)
                  cin>>a[k];
      search(a,x,1,10);
}
void search(int a[],int x,int top,int bot)      //二分查找递归过程
{     int mid;
      if (top<=bot)
      {
      mid=(top+bot)/2;                          //求中间数的位置
                  if (x==a[mid]) cout<<"YES"<<endl;              //找到就输出
                  else
                  if (x<a[mid]) search(a,x,mid+1,bot);   //判断在前半段还是后半段查找
                  else search(a,x,top,mid-1);
      }
      else cout<<"NO"<<endl;
}
```

【例3】用递归的方法求斐波那契数列中的第N个数

$$f_n=\begin{cases}0 & n=0\\ 1 & n=1\\ f_{n-1}+f_{n-2} & n>1\end{cases}$$

【参考程序】

```
#include<iostream>
using namespace std;
int a[11];
int fib(int);
int main()
{
   int m;
   cin>>m;
   cout<<"fib("<<m<<")="<<fib(m);
}
```

```
int fib(int n)
    {
      if (n==0) return 0;
          //满足边界条件，递归返回
      if (n==1) return 1;
              //满足边界条件，递归返回
      return (fib(n-1)+fib(n-2));
                  //递归公式，进一步递归
    }
```

输入 15

输出 fib(15)=610

【例4】Hanoi汉诺塔问题

有N个圆盘，依半径大小（半径都不同），自下而上套在A柱上，每次只允许移动最上面一个盘子到另外的柱子上去（除A柱外，还有B柱和C柱，开始时这两个柱子上无盘子），但绝不允许发生柱子上出现大盘子在上，小盘子在下的情况，现要求设计将A柱子上N个盘子搬移到C柱去的方法。

【算法分析】

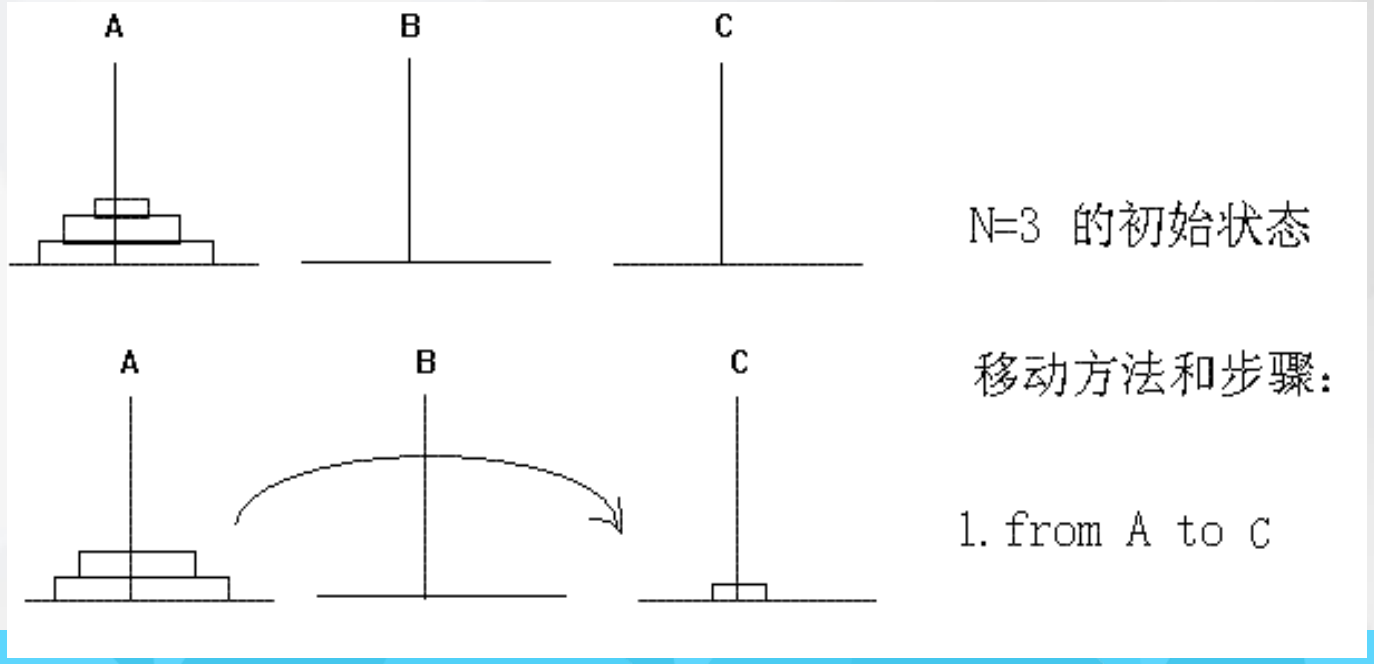
本题是典型的递归程序设计题。

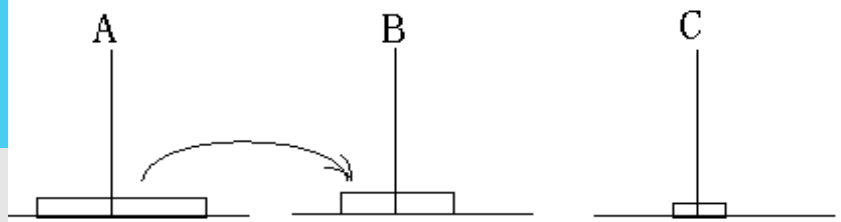
(1)当N=1 时，只有一个盘子，只需要移动一次:A—>C;

(2)当N=2时，则需要移动三次：

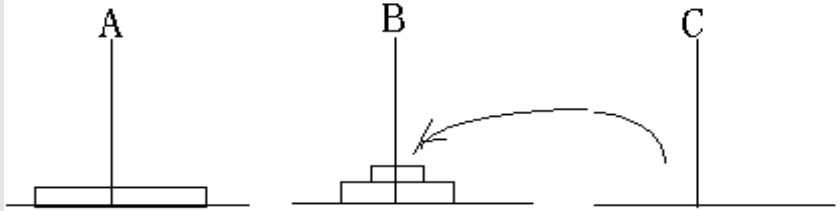
A------ 1 ------> B,　　A ------ 2 ------> C,　　B ------ 1------> C.

(3)如果N=3，则具体移动步骤为：

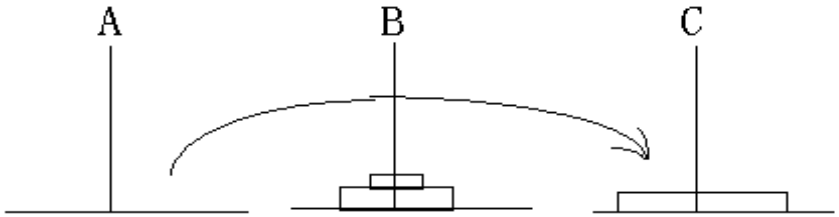




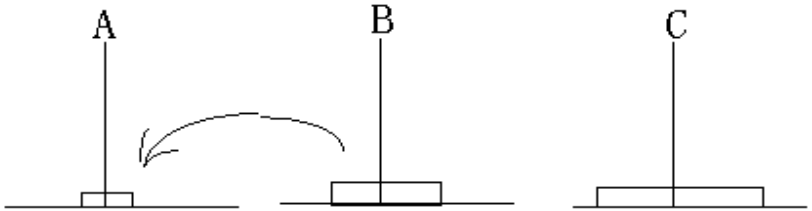
2. from A to B



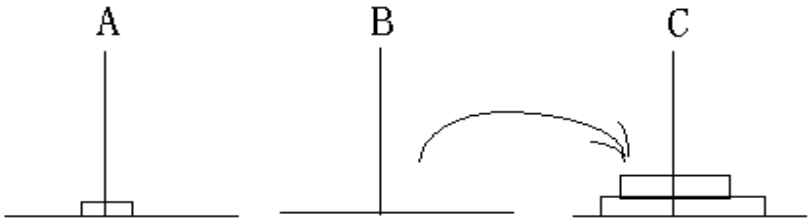
3. from C to B



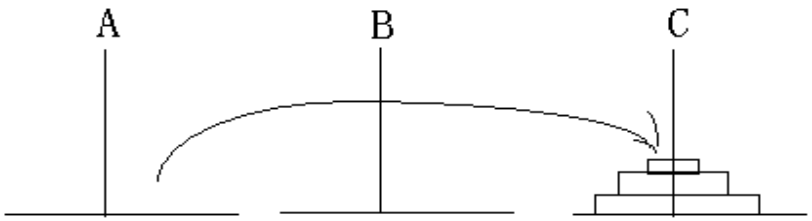
4. from A to C



5. from B to A

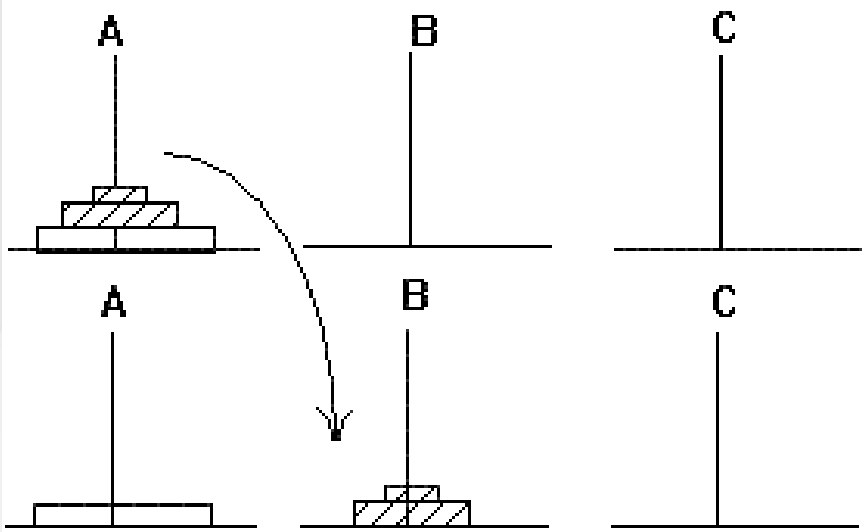


6. from B to C



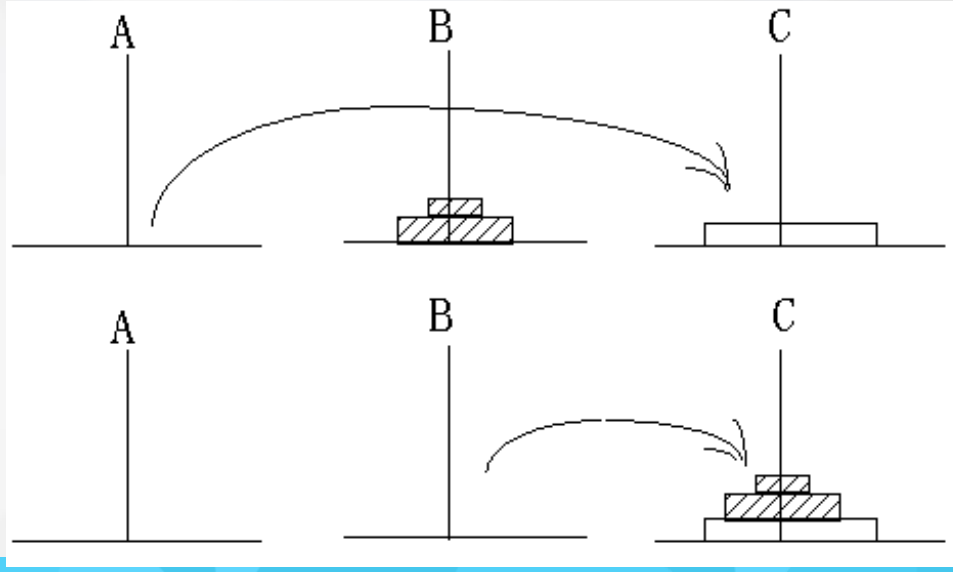
7. from A to C

假设把第3步，第4步，第7步抽出来就相当于N=2的情况（把上面2片看作是一个子问题）：



原问题：欲将A柱上的N片移到C柱上，B柱为过渡柱，记为（A，C，B）

1．将A柱上的(N-1)片移到B上，{A为源柱，B为目标，C为过渡柱，记为（A，B，C)}



2．将A柱上剩下的1片直接移到C上；｛不需调用过程｝

3．将B上的(N-1)片移到C上，结束。{B为源柱，C为目标柱，A为过渡柱，记为(B，C，A)}

递归算法描述

整个问题看作是将N个盘子从柱子A借助B移到C的问题，其解叫hanoi（n,a,b,c）。步骤设计：

①如果N=0，则退出，即结束程序;否则继续往下执行;

②用C柱作为协助过渡，将A柱上的（N-1）片移到B柱上，调用子问题解hanoi(n-1, a,c,b);

③将A柱上剩下的一片直接移到C柱上;

④用A柱作为协助过渡，将B柱上的（N-1）移到C柱上，调用子问题解hanoi (n-1,b,a,c)。

【参考程序】

```
#include<iostream>
using namespace std;
int k=0,n;
void move(int k,char from,char to)
{
   cout <<k<<" :from "<<from <<"-->"<<to<<endl;
}
void hanoi(int n,char a,char b,char c)
//用b柱作为协助过渡，将a柱上的（n）移到c柱上
{
   if (n==0) return;          //如果n=0，则退出，即结束程序
   hanoi(n-1,a,c,b );          //用c柱作为协助过渡，将a柱上的（n-1）片移到b柱上
   k++;
   move(k,a,c);    //cout <<k<<" :from "<<a <<"-->"<<c<<endl;
   hanoi(n-1,b,a,c );          //用a柱作为协助过渡，将b柱上的（n-1）移到c柱上
}
int main()
{
          cout<<"n=";
   cin>>n;
   hanoi(n,'a','b','c');
}
```

【例5】集合的划分

【问题描述】

设S是一个具有n个元素的集合，S = {a_1，a2，……，an}，现将S划分成k个满足下列条件的子集合S1，S2，……，Sk，且满足：

1. $S_i \neq \Phi$
2. $S_i \cap S_j = \Phi$ $(1 \leqslant i, j \leqslant k \quad i \neq j)$
3. $S1 \cup S2 \cup S3 \cup \cdots \cup Sk = S$

则称S1，S2，……，Sk是集合S的一个划分。它相当于把S集合中的n个元素a1，a2，……，an 放入k个（$0 < k \leq n < 30$）无标号的盒子中，使得没有一个盒子为空。请你确定n个元素a1，a2，……，an 放入k个无标号盒子中去的划分数S(n，k)。

【输入样例】setsub.in

23 7

【输出样例】setsub.out

4382641999117305

【算法分析】

先举个例子，设S = {1，2，3，4} ，k = 3，不难得出S有6种不同的划分方案，即划分数S(4，3)=6，具体方案为：

$\{1，2\} \cup \{3\} \cup \{4\}, \{1，3\} \cup \{2\} \cup \{4\}, \{1，4\} \cup \{2\} \cup \{3\}$

$\{2，3\} \cup \{1\} \cup \{4\}, \{2，4\} \cup \{1\} \cup \{3\}, \{3，4\} \cup \{1\} \cup \{2\}$

考虑一般情况，对于任意的含有n个元素a_1 ，a_2，……，a_n的集合S，放入k个无标号的盒子中去，划分数为S(n，k)，我们很难凭直觉和经验计算划分数和枚举划分的所有方案，必须归纳出问题的本质。其实对于任一个元素a_n，则必然出现以下两种情况：

1、 $\{a_n\}$是k个子集中的一个，于是我们只要把a_1，a_2，……，a_{n-1}划分为k - 1子集，便解决了本题，这种情况下的划分数共有S(n - 1，k - 1)个；

2、 $\{a_n\}$不是k个子集中的一个，则a_n必与其它的元素构成一个子集。则问题相当于先把a_1，a_2，……，a_{n-1} 划分成k个子集，这种情况下划分数共有S(n - 1，k)个；然后再把元素a_n加入到k个子集中的任一个中去，共有k种加入方式，这样对于a_n的每一种加入方式，都可以使集合划分为k个子集，因此根据乘法原理，划分数共有k * S(n - 1，k)个。

综合上述两种情况，应用加法原理，得出n个元素的集合{a_1，a_2，……，a_n}划分为k个子集的划分数为以下递归公式：S(n，k)＝S(n－1，k－1)＋k＊S(n－1，k) (n>k，k>0)。

下面，我们来确定S(n，k)的边界条件,首先不能把n个元素不放进任何一个集合中去，即k=0时，S(n，k)＝0；也不可能在不允许空盒的情况下把n个元素放进多于n的k个集合中去，即k＞n时,S(n，k)＝0；再者，把n个元素放进一个集合或把n个元素放进n个集合，方案数显然都是1，即k=1或k=n时，S(n,k)=1。

因此，我们可以得出划分数S(n，k)的递归关系式为：

S(n，k)＝S(n－1，k－1)＋k＊S(n－1，k)　　(n>k，k>0)

S(n，k)＝0　　(n<k)或(k＝0)

S(n，k)＝1　　(k=1)或(k＝n)

【参考程序】

```
#include<iostream>
using namespace std;

int s(int n, int k)                          //数据还有可能越界，请用高精度计算
{
    if ((n < k) || (k == 0)) return 0;          //满足边界条件，退出
    if ((k == 1) || (k == n)) return 1;
    return s(n-1,k-1) + k * s(n-1,k);          //调用下一层递归
}

int main()
{
  int n,k;
    cin >> n >> k;
    cout << s(n,k);
    return 0;
}
```

【例6】数的计数（Noip2001）

【问题描述】

我们要求找出具有下列性质数的个数(包含输入的自然数n):

先输入一个自然数n（n≤1000），然后对此自然数按照如下方法进行处理：

1．不作任何处理；

2．在它的左边加上一个自然数，但该自然数不能超过原数(输入的n)的一半；

3．加上数后，继续按此规则进行处理，直到不能再加自然数为止。

【输入样例】

6

【输出样例】

6

注释：如输入n为6，则满足条件的数为

6
16
26
126
36
136

【方法一】

用递归，f(n)=1+f(1)+f(2)+…+f(div/2)，当n较大时会超时，时间应该为指数级。

【参考程序】

```
#include<iostream>
using namespace std;
int ans;
void dfs(int m)                          //统计m所扩展出的数据个数
{
    int i;
    ans++;                               //每出现一个原数，累加器加1；
    for (i = 1; i <= m/2; i++)           //左边添加不超过原数一半的自然数，作为新原数
      dfs(i);
}
int main()
{
    int n;
    cin >> n;
    dfs(n);
    cout << ans;
    return 0;
}
```

【方法二】：用记忆化搜索，实际上是对方法一的改进。设h[i]表示自然数i满足题意三个条件的数的个数。如果用递归求解，会重复来求一些子问题。例如在求h[4]时，需要再求h[1]和h[2]的值。现在我们用h数组记录在记忆求解过程中得出的所有子问题的解，当遇到重叠子问题时，直接使用前面记忆的结果。

【参考程序】

```
#include<iostream>
using namespace std;
int h[1001];
void dfs(int m)
{
    int i;
    if (h[m] != -1) return;        //说明前面已经求得h[m]的值，直接引用即可，不需要再递归
    h[m] = 1;                      //将h[m]置为1，表示m本身为一种情况
    for (i = 1; i <= m/2; i++)
    {
        dfs(i);
        h[m] += h[i];
    }
}
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
      h[i] = -1;                   //h数组初始化为-1
    dfs(n);                        //由顶到下记忆化递归求解
    cout << h[n];
    return 0;
}
```

【方法三】

用递推，用h(n)表示自然数n所能扩展的数据个数，则h(1)=1, h(2)=2, h(3)=2, h(4)=4, h(5)=4, h(6)=6, h(7)=6, h(8)=10, h(9)=10. 分析以上数据，可得递推公式：h(i)=1+h(1)+h(2)+...+h(i/2)。此算法的时间度为O(n*n)。

- 设h[i]-i按照规则扩展出的自然数个数(1≤i≤n)。下表列出了h[i]值及其方案：

I	h[I]	自然数序列
1	1	1
2	2	2　12
3	2	3　13
4	4	4　14　24　124
5	4	5　15　25　125
…	…	…
I	$1+\sum_{k=1}^{\lfloor \frac{i}{2} \rfloor} h[k]$	1　1i　2i　12i …

由于 1 为最小自然数，因此 1 无法扩展出其它自然数。自然数 i（2≤i≤n）按照规则扩展出的自然数包括自然数 i；i 左边加上 1；i 左边加上 2 按规则扩展出的 h[2]个自然数……；由于 i 左邻的自然数不超过$\left\lfloor \frac{i}{2} \right\rfloor$，因此直至 i 左边加上 h[$\left\lfloor \frac{i}{2} \right\rfloor$]个自然数（这些自然数由$\left\lfloor \frac{i}{2} \right\rfloor$按规则扩展出）为止。由此得出递推的计数公式：

$$h[1]=1$$

$$h[i]=1+\sum_{k=1}^{\lfloor \frac{i}{2} \rfloor} h[k] \qquad (2\leqslant i\leqslant n)$$

从 1 出发，按照上述公式递推至自然数 n，便可得出 n 按规则扩展出的自然数个数 h[n]：

【参考程序】

```
#include<iostream>
using namespace std;
int h[10001];
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)   //按照递增顺序计算扩展出的自然数的个数
    {
        h[i] = 1;                  //扩展出的自然数包括i本身
        for (int j = 1; j <= i/2; j++)
                                   //i左边分别加上1...自然数 按规则扩展出的自然数
          h[i] += h[j];
    }
    cout << h[n];
    return 0;
}
```

【方法四】

是对方法三的改进，我们定义数组s，s(x)=h(1)+h(2)+...+h(x),h(x)=s(x)-s(x-1)，此算法的时间复杂度可降到O(n)。

【参考程序】

```
#include<iostream>
using namespace std;
int h[1001],s[1001];
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        h[i] = 1 + s[i/2];
        s[i] = s[i-1] + h[i];                //s是h的前缀累加和
    }
    cout << h[n];
    return 0;
}
```

【方法五】

还是用递推，只要作仔细分析，其实我们还可以得到以下的递推公式：(1)当i为奇数时，h(i)=h(i-1);

(2)当i为偶数时,h(i)=h(i-1)+h(i/2).

【参考程序】

```
#include<iostream>
using namespace std;
int h[1001];
int main()
{
    int n;
    cin >> n;
    h[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        h[i] = h[i-1];
        if (i % 2 == 0) h[i] = h[i-1] + h[i/2];
    }
    cout << h[n];
    return 0;
}
```

【课堂练习】

1、输入一串以‘！’结束的字符，按逆序输出。（用递归做）

2、背包问题

问题：假设有n件质量分配为w_1，w_2，…，w_n的物品和一个最多能装载总质量为T的背包，能否从这n件物品中选择若干件物品装入背包，使得被选物品的总质量恰好等于背包所能装载的最大质量，即$w_{i1}+w_{i2}+\ldots+w_{ik}=T$。若能，则背包问题有解，否则无解。

(例如：有5件可选物品，质量分别为8千克、4千克、3千克、5千克、1千克。假设背包的最大转载质量是10千克。)

3、阿克曼（Ackmann）函数A(x，y)中，x，y定义域是非负整数，函数值定义为：

$$Ack(m,n)=\begin{cases} n+1 & m=0 \\ Ack(m-1,1) & m\neq 0, n=0 \\ Ack(m-1,Ack(m,n-1)) & m\neq 0, n\neq 0 \end{cases}$$

写出计算Ack(m，n)的递归算法程序。

4、某人写了n封信和n个信封，如果所有的信都装错了信封。求所有的信都装错信封共有多少种不同情况。

基本形式：d[1]=0;d[2]=1

递归式：d[n]= (n-1)*(d[n-1] + d[n-2])

5、有52张牌，使它们全部正面朝上，从第2张开始，凡是2的倍数位置上的牌翻成正面朝下；接着从第3张牌开始，凡是3的倍数位置上的牌，正面朝上的翻成正面朝下,正面朝下的翻成正面朝上；接着第三轮从第4张牌开始，凡是4的倍数位置上的牌按上面相同规则翻转，以此类推，直到第1张要翻的牌是第52张为止。统计最后有几张牌正面朝上，以及它们的位置号。

6、猴子吃桃问题

猴子第一天摘下若干桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉的一半，又多吃了一个。以后每天早上都吃掉了前一天剩下的一半零一个。到第10天早上想再吃时，见只剩下一个桃子了。求第一天共摘多少桃子。（答案：1534）

【上机练习】

1、斐波那切数列

【问题描述】

斐波那切数列0，1，1，2，3，5，8，13，21，34，55……从第三项起，每一项都是紧挨着的前两项的和。写出计算斐波那切数列的任意一个数据项递归程序。

【输入格式】

输入所求的项数。

【输出格式】

输出数据项的值。

【输入样例】fbi.in

10

【输出样例】fbi.out

34

2、倒序数

【问题描述】
用递归算法写程序，输入一个非负整数，输出这个数的倒序数。
【输入格式】
输入一个非负整数。
【输出格式】
输出倒序结果。
【输入样例】num.in
123
【输出样例】num.out
321

3、十进制转换成八进制

【问题描述】

用递归算法，把任一给定的十进制正整数转换成八进制数输出。

【输入格式】

输入一个正整数，表示需要转换的十进制数。

【输出格式】

输出一个正整数，表示转换之后的八进制的数。

【输入样例】change.in

15

【输出样例】change.out

17

4、求N！的值

【问题描述】
　用递归算法，求N！的精确值(N以一般整数输入)。
【输入样例】ni.in
10
【输出样例】ni.out
10!=3628800

5、求最大公约数

【问题描述】

用递归方法求两个数m和n的最大公约数。(m>0，n>0)

【输入格式】

输入二个数，即m和n的值。

【输出格式】

输出最大公约数。

【输入样例】

8 6

【输出样例】

gcd=2

6、双色Hanoi塔问题

【问题描述】

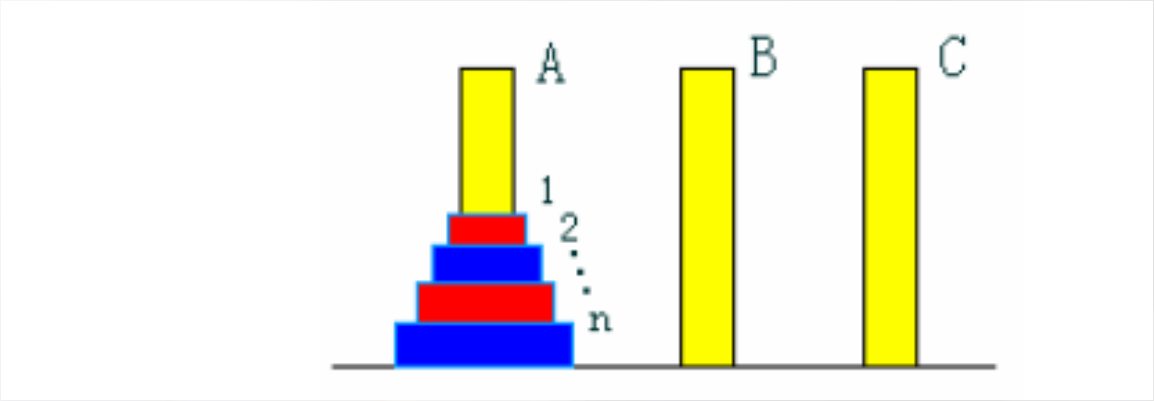
设A、B、C是3 个塔座。开始时，在塔座A 上有一叠共n 个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1，2，……，n，奇数号圆盘着蓝色，偶数号圆盘着红色，如图所示。现要求将塔座A 上的这一叠圆盘移到塔座B 上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：

规则(1)：每次只能移动1 个圆盘；

规则(2)：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；

规则(3)：任何时刻都不允许将同色圆盘叠在一起；

规则(4)：在满足移动规则(1)-(3)的前提下，可将圆盘移至A，B，C 中任一塔座上。



试设计一个算法，用最少的移动次数将塔座A 上的n个圆盘移到塔座B 上，并仍按同样顺序叠置。

【编程任务】

对于给定的正整数n，编程计算最优移动方案。

【输入格式】

由文件hanoi.in给出输入数据。第1 行是给定的正整数n。

【输出格式】

将计算出的最优移动方案输出到文件hanoi.out。文件的每一行由一个正整数k和2个字符c1和c2组成，表示将第k个圆盘从塔座c1移到塔座c2上。

【输入样例】

3

【输出样例】

1 A B
2 A C
1 B C
3 A B
1 C A
2 C B
1 A B

7、背包问题

【问题描述】

简单的背包问题。设有一个背包，可以放入的重量为s。现有n件物品，重量分别为w_1,w_2…，w_n，（$1 \le i \le n$）均为正整数，从n件物品中挑选若干件，使得放入背包的重量之和正好为s。找到一组解即可。

【输入格式】

第一行是物品总件数和背包的载重量，第二行为各物品的重量。

【输出格式】

各所选物品重量。

【输入样例】

5 10
1 2 3 4 5

【输出样例】

number:1 weight:1
number:4 weight:4
number:5 weight:5

8、2的幂次方（Noip1998）

【问题描述】

任何一个正整数都可以用2的幂次方表示。例如：

$137=2^7+2^3+2^0$

同时约定方次用括号来表示，即a^b 可表示为a（b）。

由此可知，137可表示为：

2（7）+2（3）+2（0）

进一步：$7=2^2+2+2^0$　（2(1)用2表示）

3=2+2(0)

所以最后137可表示为：

2（2（2）+2+2（0））+2（2+2（0））+2（0）

又如：

$1315=2^{10}+2^8+2^5+2+1$

所以1315最后可表示为：

2（2（2+2（0））+2）+2（2（2+2（0）））+2（2（2）+2（0））+2+2（0）

【输入格式】

正整数（n≤20000）

【输出格式】

符合约定的n的0，2表示（在表示中不能有空格）

【输入样例】

137

【输出样例】

2(2(2)+2+2(0))+2(2+2(0))+2(0)

9、数的计数（Noip2001）

【问题描述】

我们要求找出具有下列性质数的个数(包含输入的自然数n):

先输入一个自然数n（n≤1000），然后对此自然数按照如下方法进行处理：

1．不作任何处理；

2．在它的左边加上一个自然数，但该自然数不能超过原数(输入的n)的一半；

3．加上数后，继续按此规则进行处理，直到不能再加自然数为止。

【输入样例】

6

【输出样例】

6

注释：如输入n为6，则满足条件的数为

6
16
26
126
36
136

10、集合划分问题

【编程任务】

给定正整数n 和m，计算出n 个元素的集合{1,2,…, *n* }可以划分为多少个不同的由m 个非空子集组成的集合。

【输入格式】

由文件stir.in提供输入数据。文件的第1 行是元素个数n和非空子集数m。

【输出格式】

程序运行结束时，将计算出的不同的由m个非空子集组成的集合数输出到文件stir.out中。

【输入样例】

4 3

【输出样例】

6

【算法分析】

所求的是第2类Stirling数，通过可递推出如下递归式：

S(n,m)=m*S(n-1,m)+S(n-1,m-1);

S(n,n+1)=0，S(n,0)=0，S(0,0)=1

【问题描述】

n个元素的集合{1,2,…, *n* }可以划分为若干个非空子集。例如，当n=4时，集合{1，2，3，4}可以划分为15 个不同的非空子集如下：

{{1}，{2}，{3}，{4}}，{{1，2}，{3}，{4}}，{{1，3}，{2}，{4}}，{{1，4}，{2}，{3}}，{{2，3}，{1}，{4}}，

{{2，4}，{1}，{3}}，{{3，4}，{1}，{2}}，{{1，2}，{3，4}}，{{1，3}，{2，4}}，{{1，4}，{2，3}}，

{{1，2，3}，{4}}，{{1，2，4}，{3}}，{{1，3，4}，{2}}，{{2，3，4}，{1}}，{{1，2，3，4}}

其中，集合{{1，2，3，4}}由1 个子集组成；集合{{1，2}，{3，4}}，{{1，3}，{2，4}}，{{1，4}，{2，3}}，{{1，2，3}，{4}}，{{1，2，4}，{3}}，{{1，3，4}，{2}}，{{2，3，4}，{1}}由2 个子集组成；集合{{1，2}，{3}，{4}}，{{1，3}，{2}，{4}}，{{1，4}，{2}，{3}}，{{2，3}，{1}，{4}}，{{2，4}，{1}，{3}}，{{3，4}，{1}，{2}}由3 个子集组成；集合{{1}，{2}，{3}，{4}}由4 个子集组成。