清華大學 Tsinghua University

腾讯会议ID： 567 606 504/ 2021

计算机程序设计基础

第8讲 深入理解函数/二维数组

沈瑜（010-62782951）

shenyu@tsinghua.edu.cn

清华大学电机系

2021.11.2

前7讲知识点回顾

- **main**函数、头文件；
- 整型、实型、字符类型；
- 算数表达式、赋值表达式、逗号表达式、逻辑表达式；　运算的优先级：附录**D**，**p378**
- 语句（复合语句）；
- **printf**、**scanf**；
- 选择结构；循环结构；跳转语句；
- 函数
- 数组(一维数组、字符数组)

主要内容

- 函数
- 多维数组
- 增量式开发
- 函数递归调用

参考教材： 第6.2-6.3节、第7章

8.1 关于函数使用的两个问题

- 问题1：函数只有一个返回值

  若自己编写数学函数，思路清晰

  float f_result = my_sin(0.1);

  若需要返回多个值，如何处理？

  解决方案之一：改用全局变量

  弊端：模块化效果不好

- 问题2：关于全局变量和局部变量

  最好，变量不要重名。

  如果重名，“屏蔽”原则！

```
void printTime(int hour, int minute) {
    printf("time is %2d:%2d\n",hour,minute);
}

void main() {
    int hour = 23;
    int minute = 59;
    printTime(hour,minute);
}
```

变量名尽量不要重名

- *问题3：同名函数　　C语言中：不允许有同名函数存在
  - C++中：允许有同名函数存在，参数类型或者个数不同　　编译时，函数名字修饰（Decorated Name）

```
error LNK2001: unresolved external symbol ?MakeFun@@YGJJ@Z
```

  - C++中：不能仅利用返回值类型不同来区分函数

```
float max(int a, int b);
int max(int a, int b);
```

◆float　pow (float, int)

◆double pow (double, int)

◆long double pow(long double,int)

扩展阅读：https://blog.csdn.net/aastoneaa/article/details/9715919

问题：pow（28,28 ）为什么报错？

pow

Error: 有多个 重载函数 "pow" 实例与参数列表匹配:
函数 "pow(double _X, int _Y)"
函数 "pow(float _X, int _Y)"
函数 "pow(long double _X, int _Y)"
参数类型为: (int, int)

- float pow（float, int）
- double pow (double,int)
- long double pow(long double,int)

```
int main()
{

    int x=28;
    float v1;
    double v2;
    for(int i=0;i<30;i++)
    {
        v1=pow((float)x,i);
        v2=pow((double)x,i);
        printf("%2d %45f \t%lf \n",i,v1,v2);
    }
    return 0;
}
```

类型不匹配，需强制类型转换

【续】问题：pow（28,28 ）为什么报错？

C:\Windows\system32\cmd.exe

0	1.000000	1.000000
1	28.000000	28.000000
2	784.000000	784.000000
3	21952.000000	21952.000000
4	614656.000000	614656.000000
5	17210368.000000	17210368.000000
6	481890304.000000	481890304.000000
7	13492928512.000000	13492928512.000000
8	377801998336.000000	377801998336.000000
9	10578456215552.000000	10578455953408.000000
10	296196782424064.000000	296196766695424.000000
11	8293509371002880.000000	8293509467471872.000000
12	232218264535564290.000000	232218265089212420.000000
13	6502111544434753500.000000	6502111422497947600.000000
14	182059123244173100000.000000	182059119829942530000.000000
15	5097655380468102600000.000000	5097655355238391000000.000000
16	142734348401307060000000.000000	142734349946674950000000.000000
17	3996561755236597700000000.000000	3996561798506898500000000.000000
18	111903732605389250000000000.000000	111903730358193160000000000.000000
19	3133304328483458200000000000.000000	3133304450029408700000000000.000000
20	87732521197536830000000000000.000000	87732524600823435000000000000.000000
21	2456510669088895000000000000000.000000	2456510688823056300000000000000.000000
22	68782298734489059000000000000000.000000	68782299287045578000000000000000.000000
23	1925904325880067400000000000000000.000000	1925904380037275900000000000000000.000000
24	53925321743611907000000000000000000.000000	53925322641043728000000000000000000.000000
25	1509909088049295900000000000000000000.000000	1509909033949224400000000000000000000.000000
26	42277455733030886000000000000000000000.000000	42277452950578284000000000000000000000.000000
27	1.#INF00	1183768682616191900000000000000000000000.000000
28	1.#INF00	33145523113253375000000000000000000000000.000000
29	1.#INF00	928074647171094550000000000000000000000000.000000

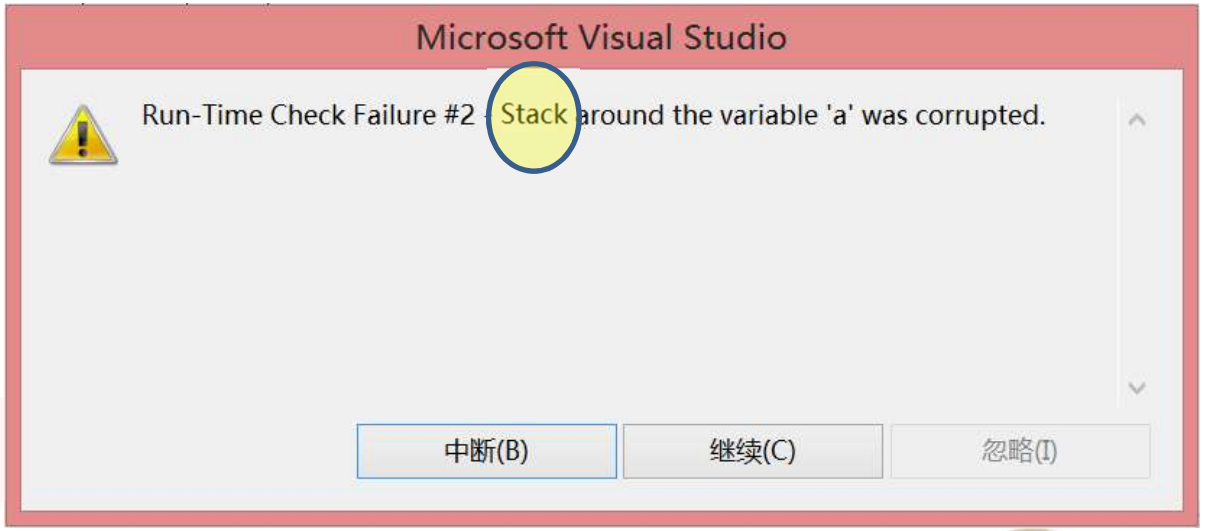
8.2 二维数组

复习一维数组

```
#define N 10

int a[N]={ 0,1,2,3,4 };

for( int i=0; i<N; i++)

{

    a[i] *= 2;

}

int b[10];

b = a;   //错误的代码
```

使用数组时最严重、常见的错误——“下标越界”

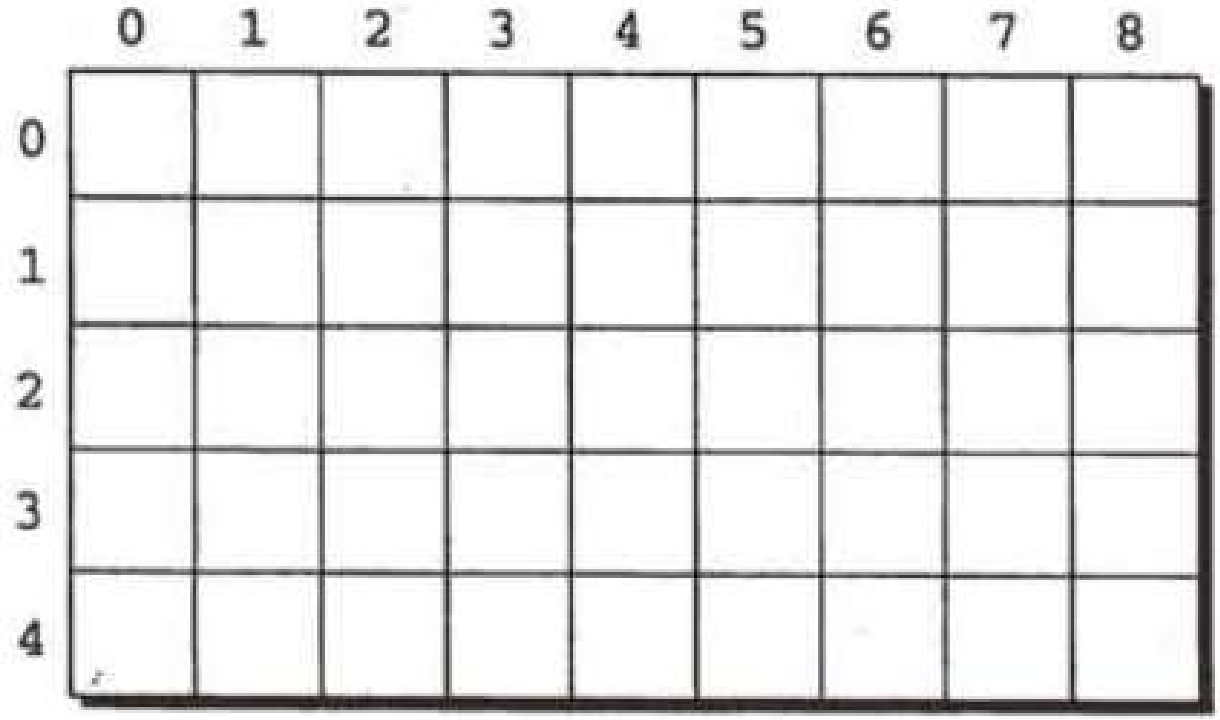
```
#include <stdio.h>
#include <string.h>

#define N 10
int main()
{
    int a[N]={ 0,1,2,3,4,5,6,7,8,9 };
    for( int i=0; i<=N; i++)
    {
        a[i] *= 2;
    }

    return 0;
}
```



复习字符数组

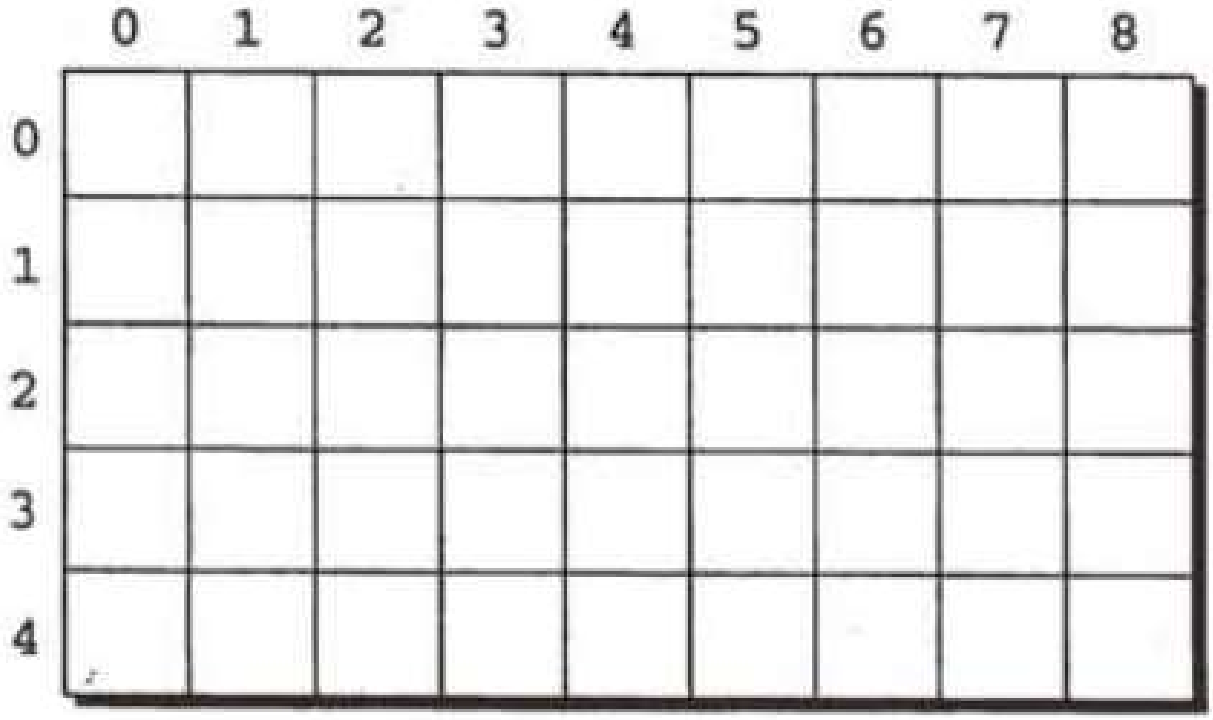
```c
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 100
int main()
{
    char str[MAX_LENGTH]="C Program";  //最后一个字符是’\0’
    printf("str:\t%s\n", str);                  //输出
    char str2[MAX_LENGTH]="C P\0rogram";
    printf( "%d\n", strlen(str2) );   //输出为3.  遇到’\0’则结束

    int nResult = strcmp( str, str2 ); //比较两个字符串
    printf("strcmp( str, str2 ) = %d\n", nResult );
    if( strcmp( str, str2 ) == 0 ) //返回值0表示完全相等
        printf( "两个字符串完全相等！\n" );
    else
        printf( "两个字符串不完全相等！\n" );

    strcpy( str, str2  ); //将str2拷贝到str（遇’\0’即停）
    printf("str:\t%s\n", str);                  //输出

    printf("strcmp( str, str2 ) = %d\n", strcmp( str, str2 ) );
    if( strcmp( str, str2 ) == 0 )
        printf( "两个字符串完全相等！\n" );

    return 0;
}
```

多维数组



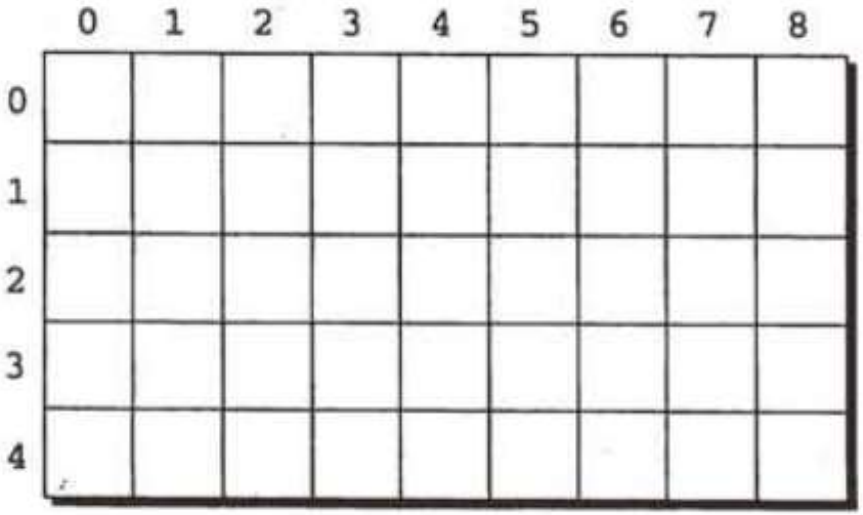
1. 多维数组定义

数据类型 变量名[维度1][维度2]



int m[5][9];

实际存储



- int m[5][9];

先存第0行，再存第1行，...

问题：int m1[5][9],m2[5*9] 需要的存储空间一样多吗?

- int m[5][9];

先行后列，顺序存储

- int m[5*9];

顺序存储

例1 多维数组定义

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

#define N 10
int main(){
    int n=10;
    int a[N][N];
    double a1[N][N];
    int b[N, 2*N]; //错误代码
    int b1[n][N];  //错误代码

    return 0;
}
```

```
\test.cpp(10): error C2057: 应输入常里表达式
\test.cpp(10): error C2466: 不能分配常里大小为 0 的数组
\test.cpp(10): error C2133: “b1”: 未知的大小
```

例2 多维数组的初始化

```
#define N 3
int main()
{
//1. 完全初始化
    int a[N][N] = { {1,2,3}, {2,3,1}, {3,2,1}};
//2. 部分行，不完全初始化
    int b[N][N] = { {1,2,3}, {2,3,1}, };
//3. 部分列，不完全初始化
    int c[N][N] = { {1,2,3}, {2,}, {3,2,1}};
//4. 特例：全部初始化为0
    int d[N][N] = { {0} };

    return 0;
}
```

多维数组的初始化方法续

```
#define N 3
int main()
{
//亦可用一维数组形式初始化
    int b[N][N] = { 1,2,3,  2,3,1 };
//特例：全部初始化为0
    int d[N][N] = { 0 };

    return 0;
}
```

不推荐！可读性不强。

3. 多维数组的使用

- 数组元素引用：数组名[下标1][下标2]

- 注意：数组下标从0开始

例3 矩阵元素赋值

```
#define N1 10
#define N2 12
int main()
{
    int a[N1][N2], i, j;
    for(i=0; i< N1; i++)
        for(j=0; j< N2; j++)
            if(i==j)
                a[i][j] =1;
            else
                a[i][j] =0;
    return 0;
}
```

程序作用?

例4 矩阵相乘

- 求两个3×3矩阵的乘积

$$c[i,j]=\sum_{k=1}^{n}A[i,k]B[k,j]$$

```c
#include <stdio.h>
#define N 3
int main()
{
    int a[N][N] = { {1,2,3}, {2,3,1}, {3,2,1}};
    int b[N][N] = { {0} };
    int c[N][N] = { 0 };
    int i, j, k;

    for(i=0; i<N; i++)
        b[i][i]=1;
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j]+=a[i][k]*b[k][j];

    for(i=0; i<N; i++)
    {
        for(j=0; j<N; j++)
            printf( " %d,", c[i][j] );
        printf("\n");
    }
    return 0;
}
```

3. 二维字符数组的使用

- 参见教材**p167 例6.9**
- 求最大的字符串

```
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 256
#define N 4
int main()
{
    char strings[N][MAX_LENGTH] = { "China", "Japan", "India", "USA"};
    char string_max[MAX_LENGTH];

    strcpy( string_max, strings[0] );
    for(int i=1; i<N; i++)
        if( strcmp( strings[i], string_max ) > 0 )
            strcpy( string_max, strings[i] );

    printf("The largest string is:\n%s\n",string_max );
    return 0;
}
```

单选题 1分

设置

```
int a[6][6],i,j;
for(i=1;i<6;i++)
{
   for(j=1;j<6;j++)
   {
         a[i][j]=(i/j)*(j/i);
          printf( “%2d” ,a[i][j]);
   }
  printf( “\n” );
}
```

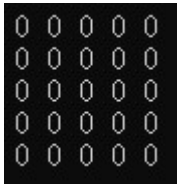
输出的结果为

A

B

C

D

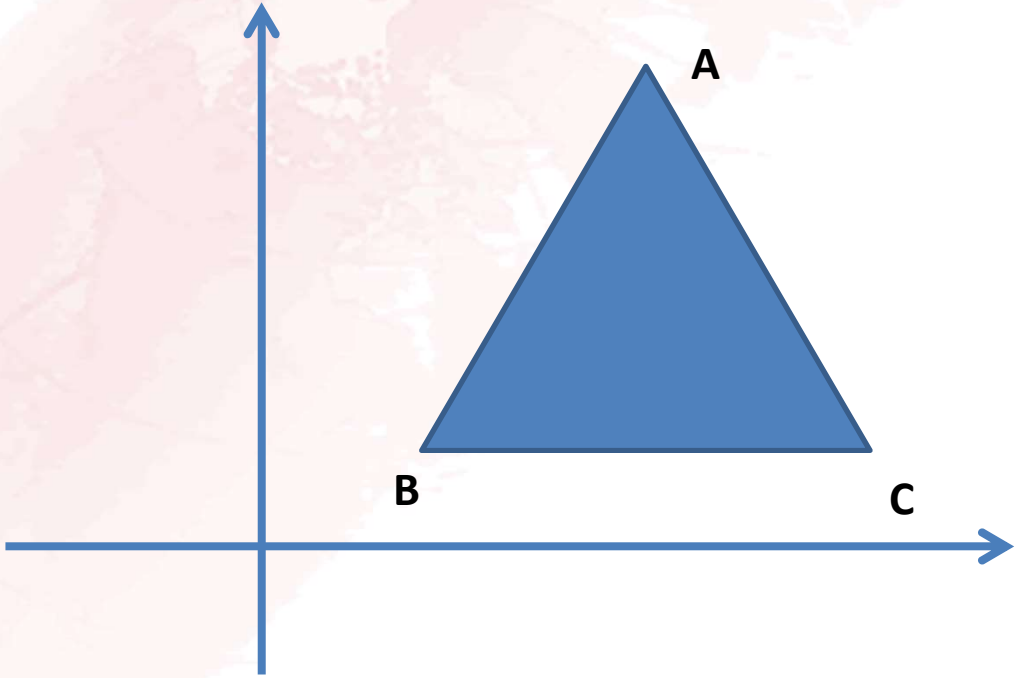


提交

8.3 增量式开发

- 问题的分解和转化

例5：已知三角形三个顶点坐标，求其面积。



步骤1：设计（找到思路，确定求解步骤）

- 无思路，“转化”成熟悉问题
- 有思路，“分解”成一系列步骤（算法）

步骤2：流程图

两点间距离公式：

$$a = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

海伦公式：

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

$$p = (a+b+c)/2$$

```
void main() {
   double x1=1,y1=10,  x2=4,y2=3,  x3=5,y3=6;
   double a = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
   double b = sqrt((x1-x3)*(x1-x3) + (y1-y3)*(y1-y3));
   double c = sqrt((x3-x2)*(x3-x2) + (y3-y2)*(y3-y2));
   double p = (a+b+c)/2;
   if (a+b>c && a+c>b && b+c>a) {
      double s = sqrt(p*(p-a)*(p-b)*(p-c));
      printf("所求三角形的面积为: %.3lf\n",s);
   }
   else printf("所给三点不构成三角形\n");
}
```

不足？ 代码有重复，容易出错。

用函数改写...

```
double distance(double,double,double,double);

void main() {
   double x1=1,y1=10,  x2=4,y2=3,  x3=5,y3=6;
   double a = distance(x1,y1,  x2,y2);
   double b = distance(x1,y1,  x3,y3);
   double c = distance(x3,y3,  x2,y2);
   double p = (a+b+c)/2;
   if (a+b>c && a+c>b && b+c>a) {
      double s = sqrt(p*(p-a)*(p-b)*(p-c));
      printf("所求三角形的面积为: %.3lf\n",s);
   }
   else printf("所给三点不构成三角形\n");
}
```

你能理解这段代码吗？

慢着，**distance**函数呢？

```
double distance(double,double,double,double);

void main() {
  double x1=1,y1=10,  x2=4,y2=3,  x3=5,y3=6;
  double a = distance(x1,y1,  x2,y2);
  double b = distance(x1,y1,  x3,y3);
  double c = distance(x3,y3,  x2,y2);
  double p = (a+b+c)/2;
  if (a+b>c && a+c>b && b+c>a) {
    double s = sqrt(p*(p-a)*(p-b)*(p-c));
    printf("所求三角形的面积为: %.3lf\n",s);
  }
  else printf("所给三点不构成三角形\n");
}

double distance(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
```

含意？

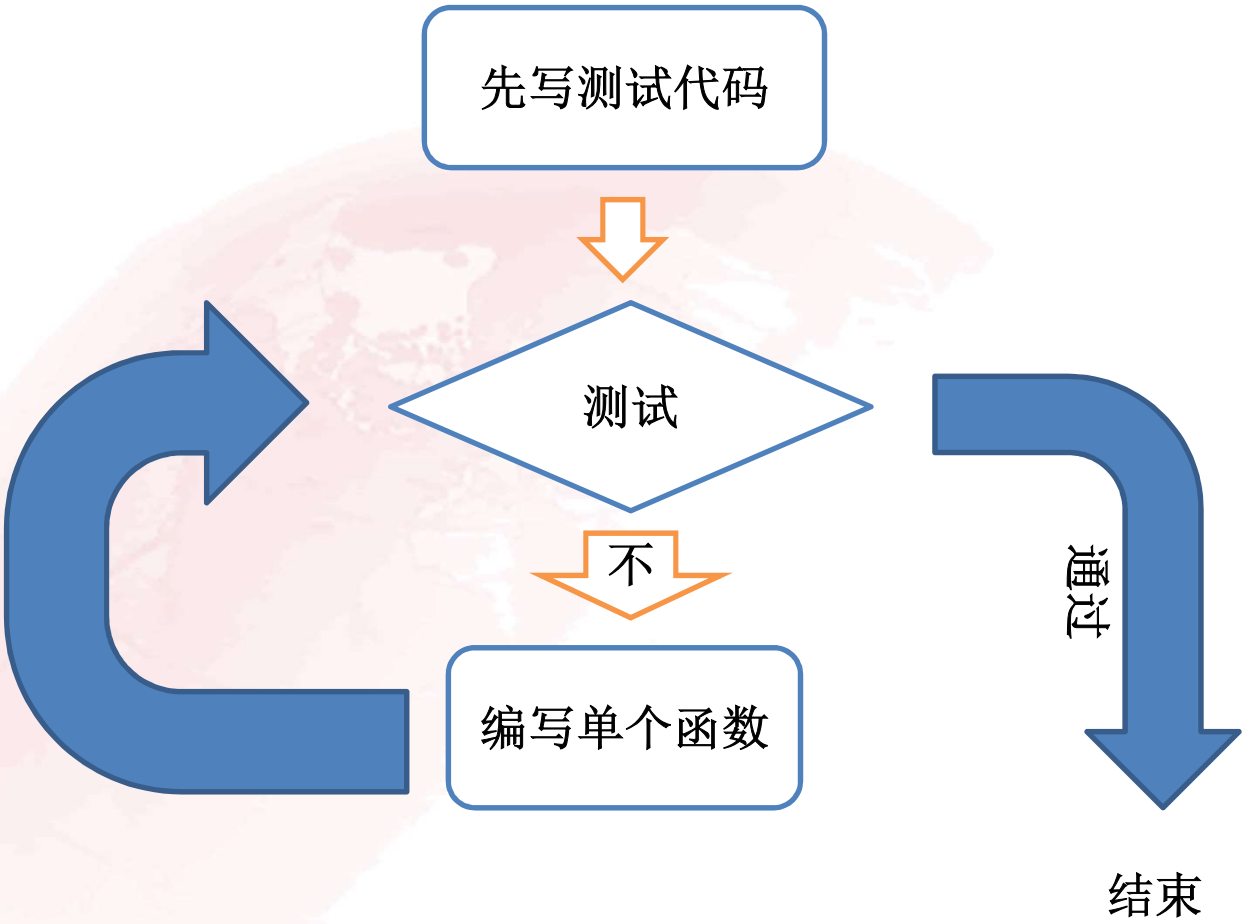
步骤3：测试

- 运行一下

结果正确吗？

让我们学习一点单元测试方法...

开发：单元测试



- 知识1： assert 函数
  - 断言某条件是否成立
  - 如果不成立，则终止程序，给出错误信息

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

#define DEBUG

double distance(double, double, double, double);

void main() {
#ifdef DEBUG
    double len1 = distance(0, 0, 0, 3);
    assert ( len1==3 );
#endif
}

double distance(double x1, double y1, double x2, double y2){
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
```

- 若成立，不做任何事情
- 定义在**assert.h**中

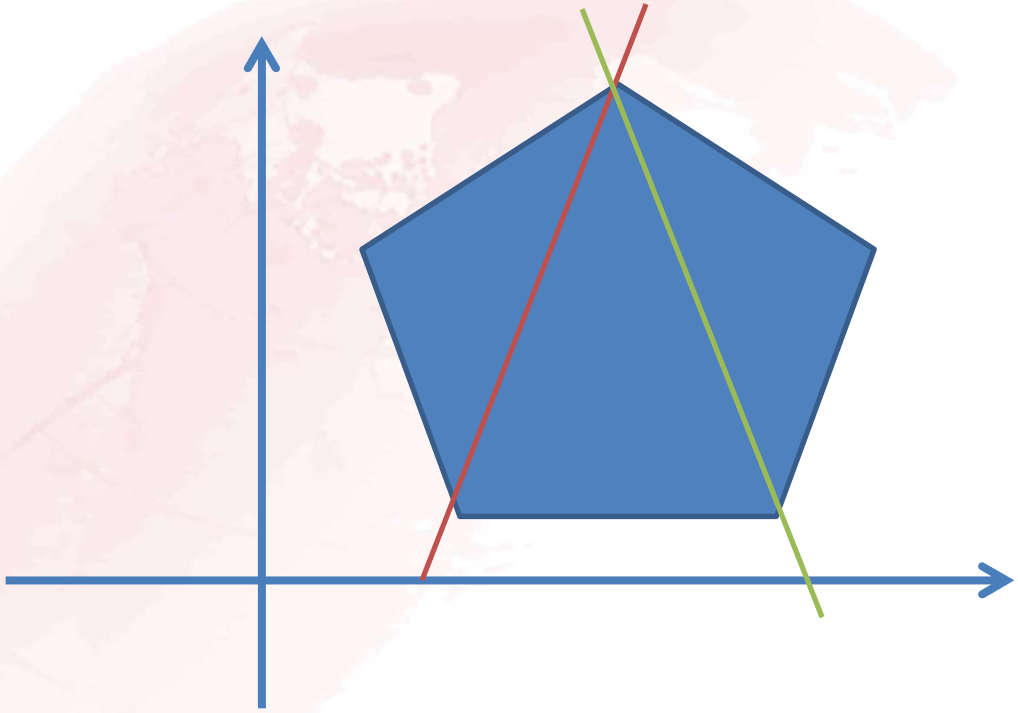
- 知识2： **#define,#ifdef**
  - 确定是否应该编译 **#ifdef,#endif** 包含的代码段
  - 若**#ifdef**后面的符号有定义，则编译
  - 否则不编译

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

#define DEBUG

double distance(double, double, double, double);

void main() {
#ifdef DEBUG
    double len1 = distance(0, 0, 0, 3);
    assert ( len1==3 );
#endif
}

double distance(double x1, double y1, double x2, double y2){
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}
```

- **#define** 定义符号
- **#ifdef,#endif** 成对出现

```
void main() {
   double x1=1,y1=10, x2=4,y2=3, x3=5,y3=6;
   double s= calArea(x1,y1, x2,y2, x3,y3);
   printf("所求三角形的面积为: %.3lf\n",s);
}

double calArea(double x1, double y1,double x2,
                  double y2, double x3,double y3) {
   double a = distance(x1,y1, x2,y2);
   double b = distance(x1,y1, x3,y3);
   double c = distance(x3,y3, x2,y2);
   double p = (a+b+c)/2;
   double s = 0;
   if (a+b>c && a+c>b && b+c>a)
      s = sqrt(p*(p-a)*(p-b)*(p-c));
   return s;
}
```

- 例6：已知多边形顶点坐标，求其面积。



8.4 函数递归调用 *

什么是递归？

德罗斯特效应

递归：自描述

如果一个对象的描述中包含它本身，我们就称这个对象是递归的。

递归作用

- 递归算法一般用于解决两类问题：
- (1)数据本身，按递归定义的(如Fibonacci函数)
- (2)问题解法，按递归算法实现(回溯)

第1类 数据本身，按递归进行定义

例7　阶乘函数

阶乘函数可递归地定义为：

$$n!=\begin{cases}1 & n=0\\ n(n-1)! & n>0\end{cases}$$

边界条件

递归方程

边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

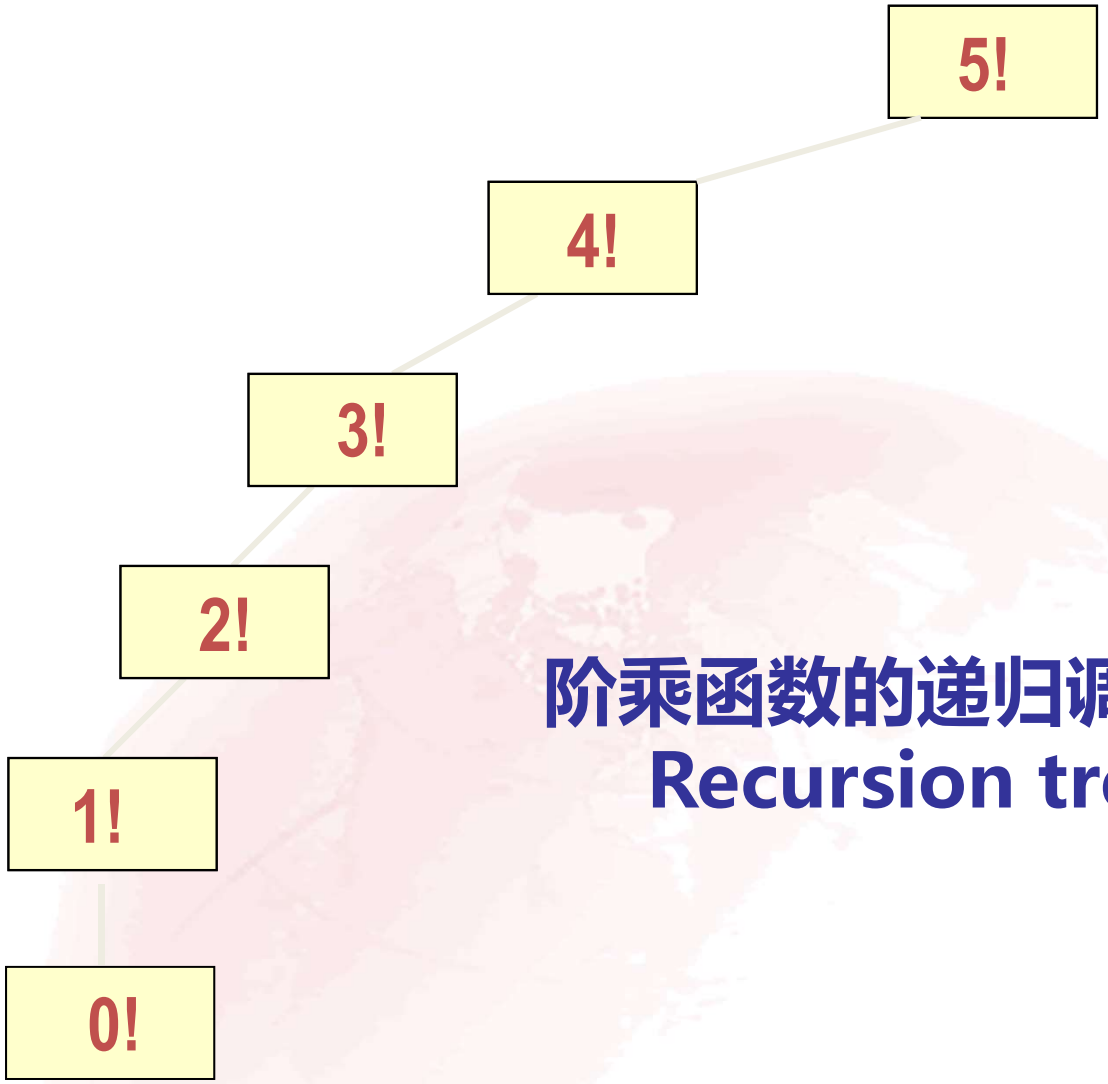
第1类 数据本身，按递归进行定义

$$n!=\begin{cases}1 & n=0\\ n(n-1)! & n>0\end{cases}$$

```
double factorial(int n) {
    if (n==0) return 1;

    return n * factorial(n-1);
}
```

递归调用：自调用

在函数的定义中直接或间接调用函数自身。



阶乘函数的递归调用树
Recursion tree

Rescuvie(5)

{5 * Rescuvie(4)}

{5 * {4 * Rescuvie(3)}}

{5 * {4 * {3 * Rescuvie(2)}}}

{5 * {4 * {3 * {2 * Rescuvie(1)}}}}

{5 * {4 * {3 * {2 * 1}}}}

{5 * {4 * {3 * 2}}}

{5 * {4 * 6}}

{5 * 24}

120

阶乘函数
递归调用过程

- 例8　Fibonacci数列
- 无穷数列1，1，2，3，5，8，13，21，34，55，……，称为Fibonacci数列。它可以递归地定义为：

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

边界条件

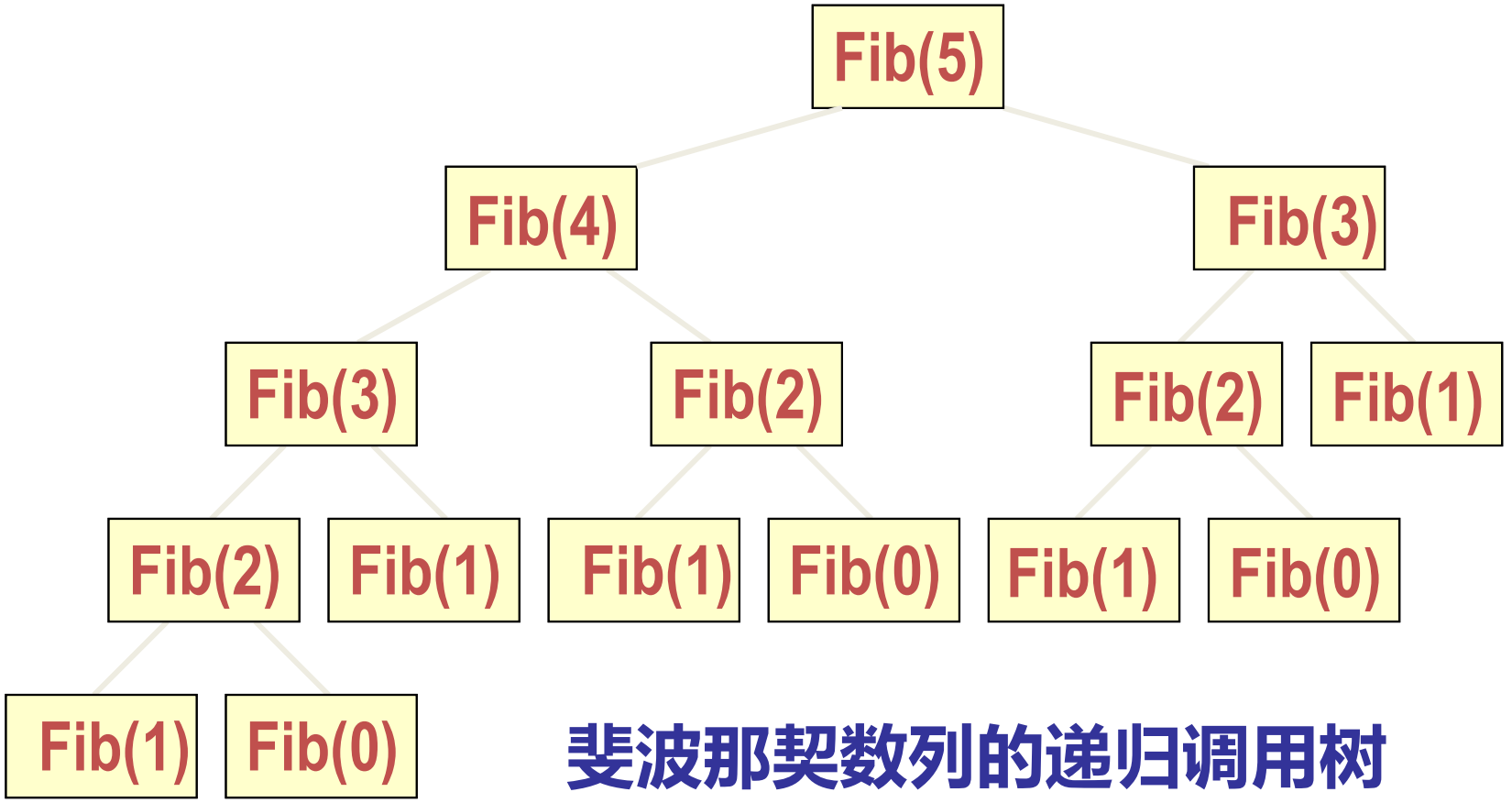
递归方程

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

```
int fib(int n)   {
    if (n <= 1) return 1;
    return fib(n-1) + fib(n-2);
}

void main() {
    int n = 6;
    printf("Fib(%2d) = %6d\n",
        n, fib(n));
    return;
}
```

递归调用：自调用

在函数的定义中直接或间接调用函数自身。



斐波那契数列的递归调用树
Recursion tree

调用次数 NumofCall(k) = 2*Fib(k+1) − 1。$O(2^n)$

例9　Ackerman函数

当一个函数及它的一个变量是由函数自身定义时，称这个函数是双递归函数。

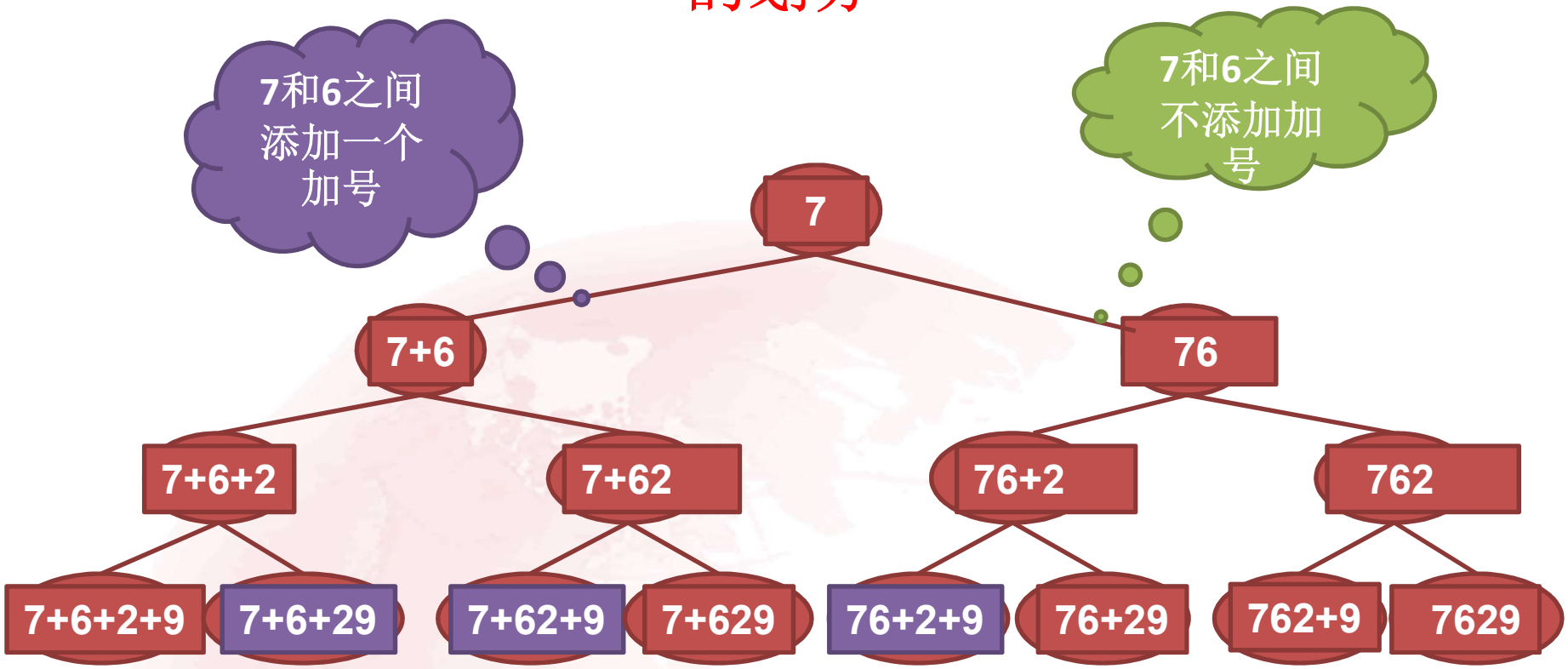
Ackerman函数A(n，m)定义如下：

$$
\begin{cases}
A(1,0) = 2 & \\
A(0,m) = 1 & m \geq 0 \\
A(n,0) = n+2 & n \geq 2 \\
A(n,m) = A(A(n-1,m),m-1) & n,m \geq 1
\end{cases}
$$

第2类 问题解法，按递归定义

- 我们有时会碰到一些题目，它们既不能通过建立数学模型解决，又没有现成算法可以套用，或者必须遍历所有状况才可以得出正确结果。这时，我们就必须采用搜索算法来解决问题。
- 搜索算法按搜索的方式分有两类，一类是深度优先搜索，一类是广度优先搜索。
- 对于搜索来说，我们需要使用到的一个技术就是递归与回溯。

问题：在数字7629中插入2个加号，找到和最小的划分



调用次数 NumofCall(n) =1+2+4+8, $O(2^n)$

第2类 问题解法，按递归定义

- 例10　汉诺塔问题

汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

第2类 问题解法，按递归定义

- 用递归思想解决汉诺塔问题是很方便的：
  - 1. 先将n-1个盘子从a柱移动到b柱
  - 2. 然后把最大的盘子从a柱移动到c柱
  - 3. 再把n-1个盘子从b柱移动到c柱。
- 第1、3步均是n-1阶的汉诺塔问题。
- 第2步是递归的终止条件，是一个能直接求解的最小子问题。把1、3步进一步分解下去，最后n阶汉诺塔问题可以被分解成若干个只需要移动一个盘子的能直接求解的子问题，使得整个问题得到解决。

汉诺塔问题

```
void Hanoi(int n, char a, char b, char c)
{
        if(n==1) //只有一个盘子，直接移动
             printf("move %c to %c\n", a, c);
        else
        {
             Hanoi(n-1, a, c, b); //n-1个盘子a柱->b柱
             printf("move %c to %c\n", a, c); //最后一个盘子a柱->c柱
             Hanoi(n-1, b, a, c); //将n-1个盘子从b柱移动到c柱
         }
}
```

```
int main()
{
        int n; //盘子个数
        char a='1', b='2', c='3'; //柱名
        printf("Enter the number of disks:");
        scanf("%d", &n);
        Hanoi(n, a, b, c);
        return 0;
}
```