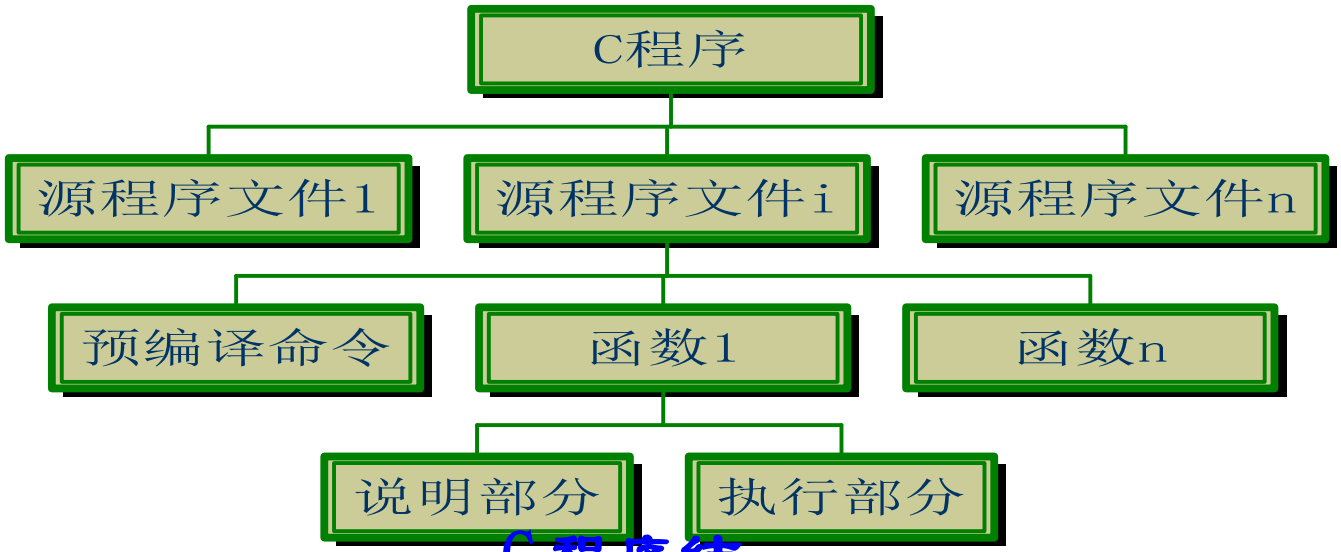
函数

- 函数的定义和调用
- 函数参数传递
- 函数与数组
- 函数的嵌套调用
- 函数的递归调用

- 模块化程序设计
  - 基本思想：
    - 将一个大的程序按功能分割成一些小模块，
    - 每一个模块用来实现一个特定的功能。
  - 特点：
    - 各模块相对独立、功能单一、结构清晰、接口简单。
    - 控制了程序设计的复杂性。
    - 提高程序的可靠性。
    - 缩短开发周期。
    - 避免程序开发的重复劳动。
    - 易于维护和功能扩充。
  - 开发方法：自上向下，逐步分解，分而治之。

C 是模块化程序设计语言



C 程序结构

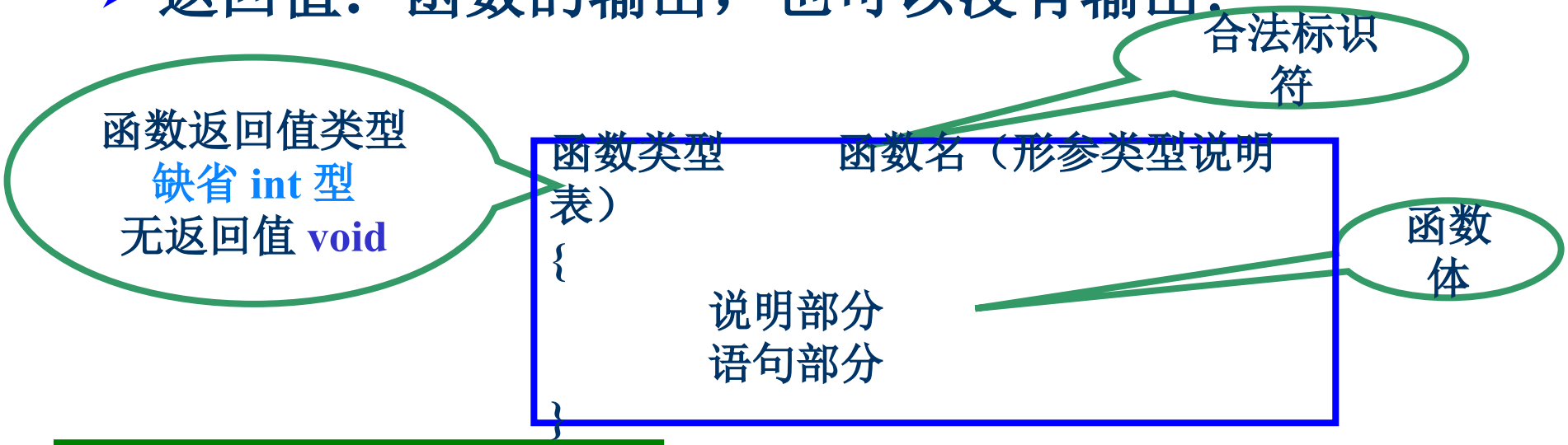
- C 是函数式语言。
- 必须有且只能有一个名为 main 的主函数。
- C 程序的执行总是从 main 函数开始，在 main 中结束。
- main 函数可以调用其它函数。
- 函数不能嵌套定义，可以嵌套调用。

- 函数分类
  - 从函数定义的角度
    - 标准函数（库函数）：由系统提供
    - 用户自定义函数
  - 从函数形式
    - 无参函数
    - 有参函数
  - 从有无返回值的角度
    - 有返回值函数
    - 无返回值函数

使用库函数应注意：
1、函数功能；
2、函数参数的数目和顺序，
　　及各参数意义和类型；
3、函数返回值意义和类型；
4、需要使用的包含文件。

1　函数的定义和调用

- 函数：完成一定功能的、相对独立的代码段；
- 形式参数：完成函数功能的输入参数，可无输入；
- 返回值：函数的输出，也可以没有输出；



例　无参函数

```
void printstar( )
{  printf("**********\n");  }
```

或

```
void printstar(void )
{  printf("**********\n");  }
```

例　有参函数

```
int max(int x, int y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

函数的返回值

- 返回语句
  - 形式：　return(表达式) ;
  - 　　或　　return　表达式 ;
  - 　　或　　return;
  - 功能：使程序控制从被调用函数返回到调用函数中，
  - 　　同时把返回值带给调用函数。
  - 说明：
    - 函数中可有多个 return 语句。
    - 若无 return 语句，遇 } 时，自动返回调用函数。
    - 无返回值函数，可以明确定义为“空类型” void

函数调用三种方式

- 函数语句：
- 例　printstar( );
- printf(“Hello,World!\n”);
- 函数表达式：
- 例　m=max(a,b)*2;
- 函数参数：
- 例　printf(“%d”, max(a,b));
- m=max(a, max(b,c));

- 对被调用函数要求：
  - 必须是已存在的函数
    - 库函数：　#include <*.h>
    - 用户自定义函数：函数声明

- 对被调用函数的声明
  - 函数声明——函数原型，先声明或定义，后使用函数；
  - 一般形式：函数类型　函数名 (形参类型　[形参名],.....);
  - 作用：告诉编译系统函数类型、参数个数及类型，以便检验
  - 函数定义与函数声明不同
  - 函数声明位置：程序的数据说明部分（函数内或外）
    - 在函数 A 内声明函数 B，在函数 A 内可以调用函数 B；
    - 在函数外声明函数 B，在声明位置以后可以调用函数 B；
  - 对在函数调用之前定义的函数，不需要声明；

函数声明举例

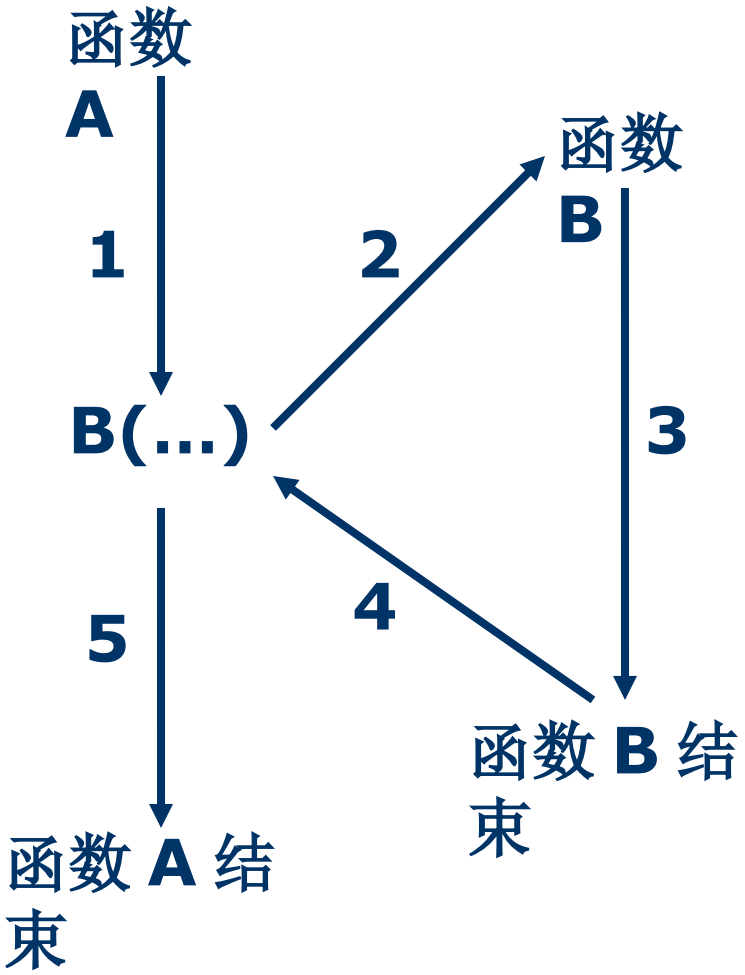
```
int main( )
{  float add(float, float);  /* 函数声明 */
   /* 在 main 函数中可调用函数 add*/
   float a,b,c;
   scanf("%f,%f",&a,&b);
   c=add(a,b);
   printf("sum is %f",c);
   return 0;
}
float add(float x, float y)
{
   float z;
   z=x+y;
   return(z);
}
```

函数声明举例

函数外的声明

```
float max(float, float ) ;
int main( )
{  float a,b;
   float c;
   scanf("%f,%f",&a,&b);
   c=max(a,b);
   printf("Max is %d\n",c);
   return 0;
}
float max(float x, float y)
{
   float z;
   z=x>y?x:y;
   return(z);
}
```

➢ 函数的调用与返回过程



- 函数 A 调用被调函数 B，在调用函数 B 前需要在函数 B 的数据存储区中保存返回地址（程序流程返回到函数 A 后，继续执行的指令的地址）和 CPU 寄存器当前的内容（也称为保存 CPU 现场）；
- 函数 B 执行结束，需要恢复 CPU 现场，并取出返回地址，使得程序流程返回到函数 A 。
- 保存 CPU 现场和返回地址，恢复 CPU 现场和返回地址，由系统自动完成

➢函数的数据存储区

- 函数被调用时，需要为函数分配数据存储区，进行返回地址、主调函数的 CPU 现场和被调函数的局部变量的存储。
- 函数的局部变量包括形参变量和函数体内定义的变量。
- 局部变量在[redacted]内存；调用时为其分配内[redacted]
- 函数执行结[redacted]局部数据存储区 。



2 函数参数传递

- 形参与实参
  - 形式参数：定义函数时函数名后面括号中的变量名。
  - 实际参数：调用函数时函数名后面括号中的表达式。

- 说明：
  - ❖实参必须有确定的值。
  - ❖形参必须指定类型。
  - ❖形参与实参类型一致，个数相同，按顺序一一对应。
  - ❖若形参与实参类型不一致，自动按形参类型转换——函数调用转换。
  - ❖实参表求值顺序，因系统而定。

例：比较两个数并输出大者

int x = a;
int y = b;
形参 = 实参

```
c=max(a,b);        （main 函数）
-------------------------------
max(int x, int y)  （max 函数）
{  int z;
   z=x>y?x:y;
   return(z);
}
```

```
int main(void)
{  int max(int, int);
   int a,b,c;
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf("Max is %d",c);
   return 0;
}
int max(int  x, int  y)
{
   int z;
   z=x>y?x:y;
   return(z);
}
```

实参

形参

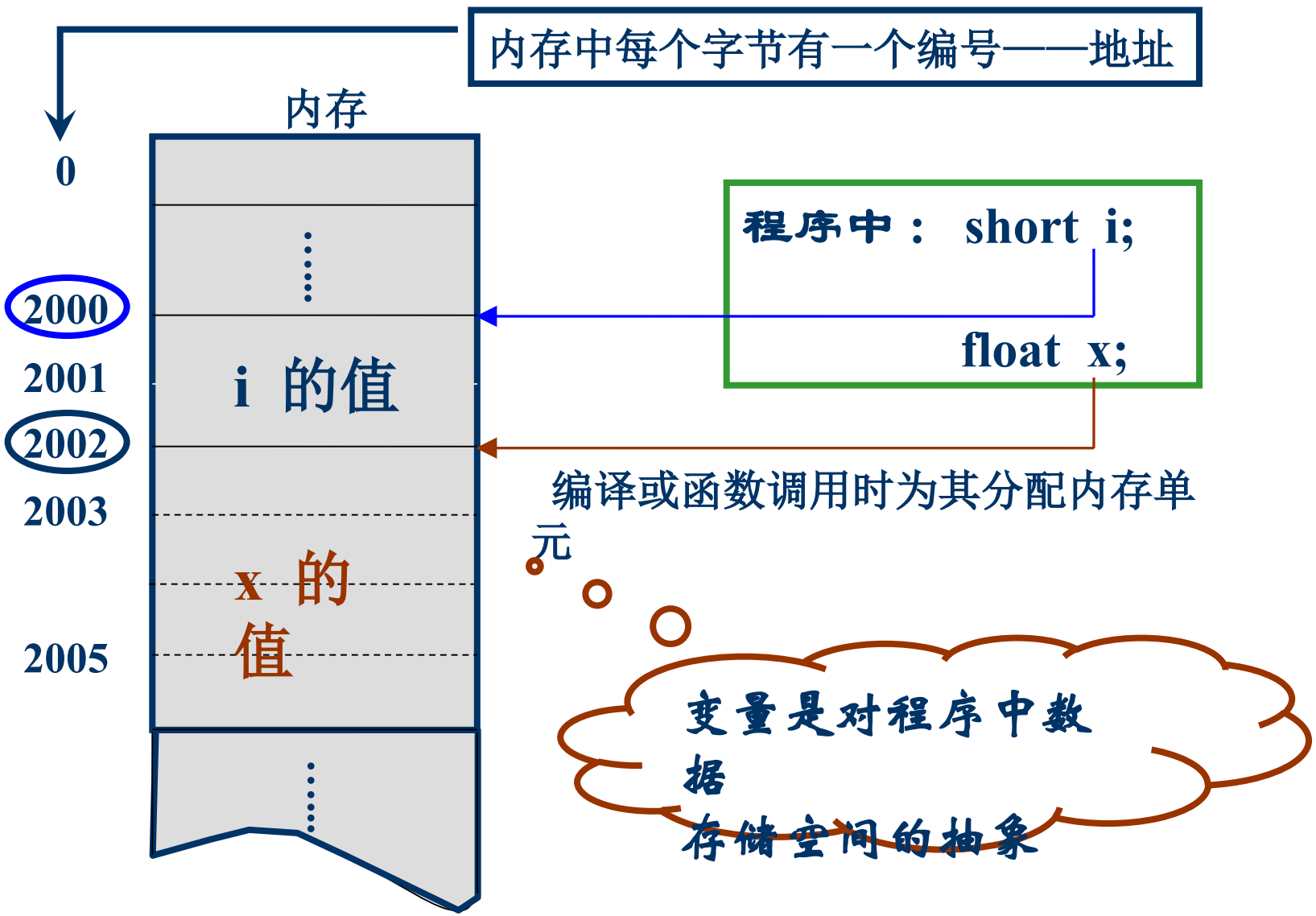
- find 函数判断一个数是否为水仙花数；
- main 函数循环调用 find 函数，根据返回值求出（100~999）所有符合条件的数以及总个数，并输出。
- 输出：

153
370
371
407
总个数 :4

```
#include <stdio.h>
int find(int num)
{  int i, j, k;
   i = num/100;
   k = num%10;
   j = (num-i*100)/10;
   if(num==i*i*i + j*j*j + k*k*k)
      return 1;
   else   return 0;
}
int main( )
{  int i, c=0;
   for(i=100; i<=999; i++)
      if(find(i))
         { printf("%d\n", i); c++; }
   printf(“ 总个数 :%d\n”, c);
   return 0;
}
```

指针的概念

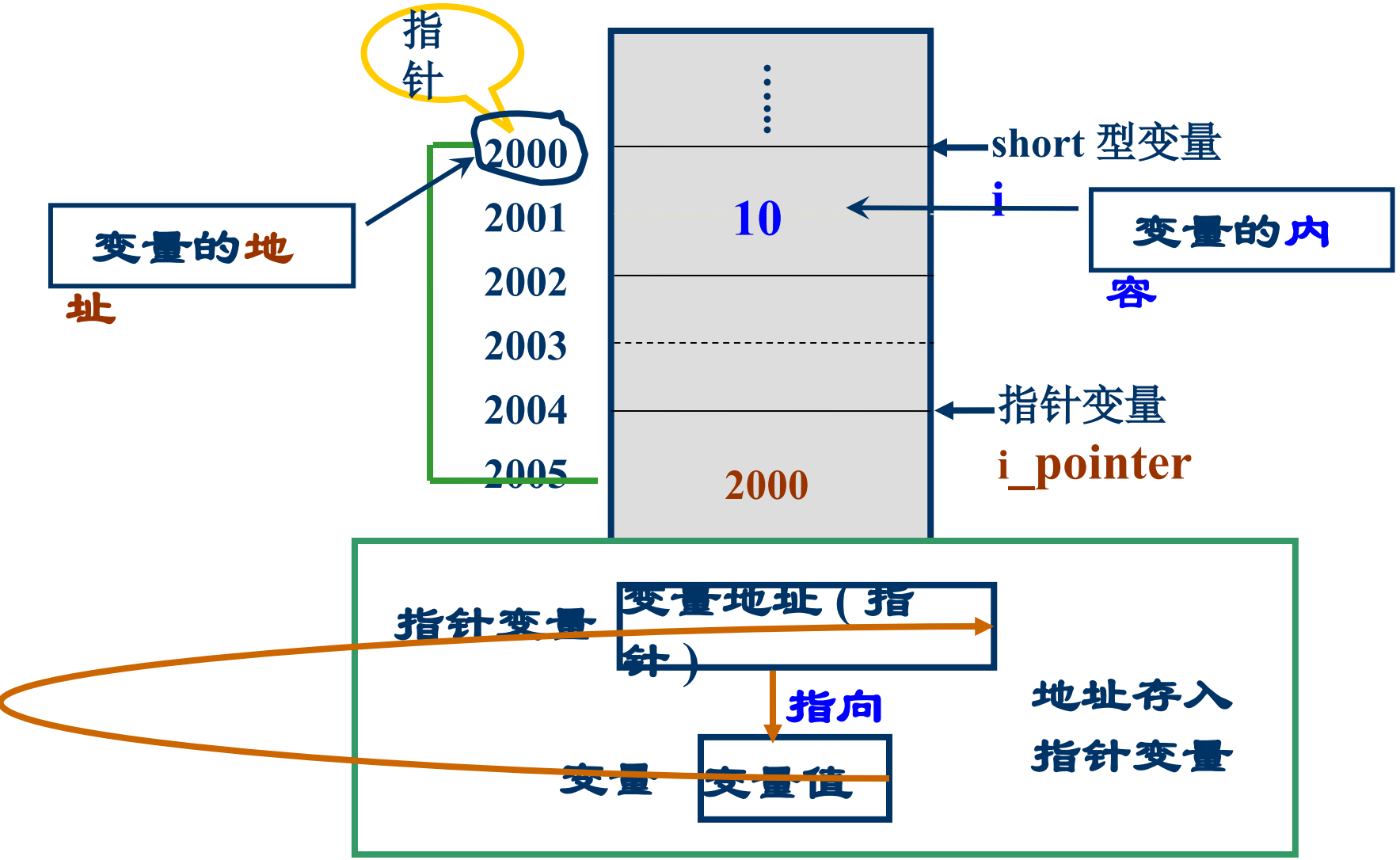
- 变量与地址


内存中每个字节有一个编号——地址
内存
0
2000
2001
2002
2003
2005
i 的值
x 的值
程序中： short i;
float x;
编译或函数调用时为其分配内存单元
变量是对程序中数据
存储空间的抽象

- 指针与指针变量

  指针：一个变量的地址。

  指针变量：专门存放地址的变量。



- & 与 * 运算符

含义：取变量的地址

两者关系：互为逆运算

单目运算符

理解：

优先级：第 2 高

结合性：自右向左

含义：取指针所指向变量的内容

单目运算符

优先级：第 2 高

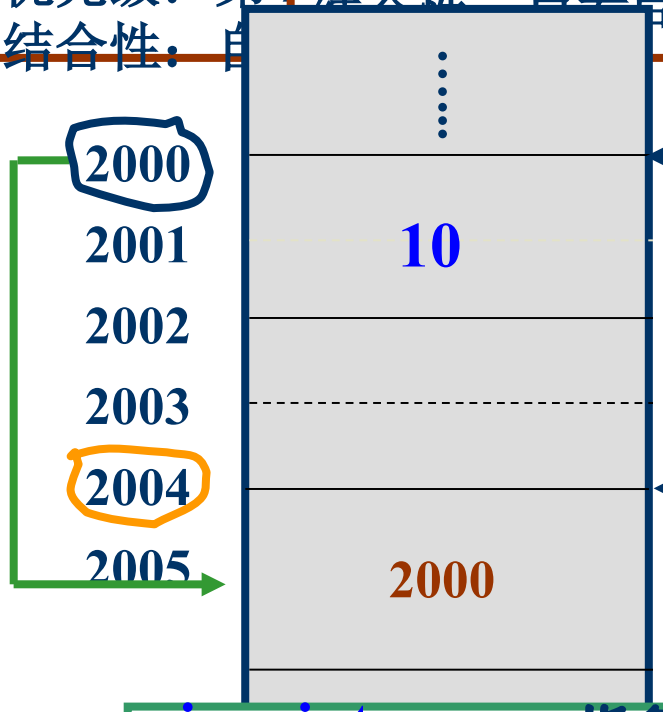
结合性：自右向左

&i_pointer

i_pointer → 2000

*i_pointer → 10

i



i_pointer = &i = &(*i_pointer)

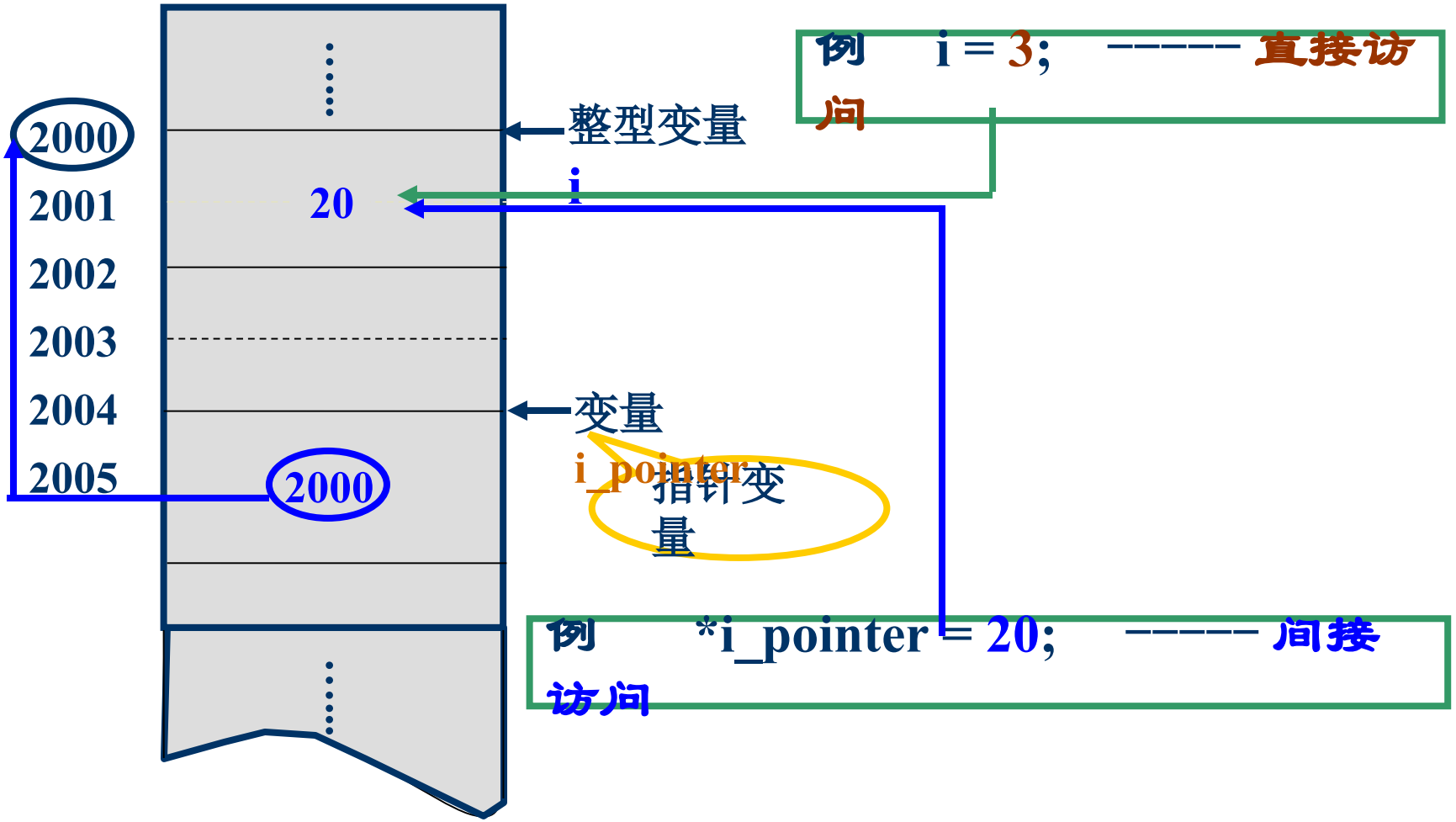
i = *i_pointer = *(&i)

i_pointer—— 指针变量，它的内容是地址

*i_pointer—— 指针的目标变量，它的内容是数据

&i_pointer—— 指针变量所占内存的地址

- 直接访问与间接访问
  - 直接访问：按变量地址存取变量值
  - 间接访问：通过存放变量地址的指针变量去访问变量



参数传递方式

- 函数调用时为形参分配单元，并将实参的值复制到形参中；
  调用结束时形参单元被释放，实参单元仍保留并维持原值。

- 值传递方式：实参 => 形参，调用时，用实参初始化形参
  - 方式：将实参的值（单向）复制给对应的形参。
  - 形参与实参占用不同的内存单元
  - 单向值传递，被调函数不能访问主调函数的存储空间

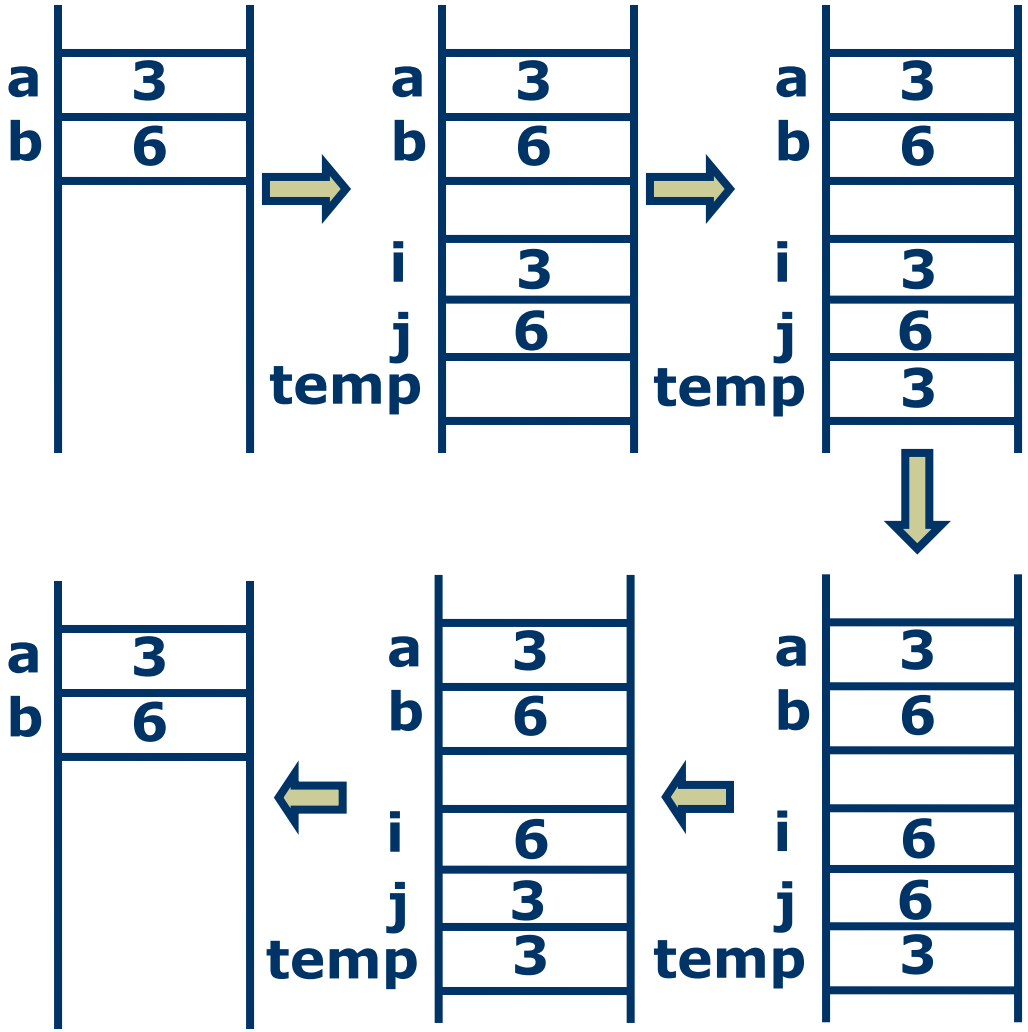
- 地址传递方式：实参 => 形参，调用时，用实参初始化形参
  - 方式：函数调用时，将数据的存储地址作为参数传递给形参。
  - 函数调用时，将地址的值传递给形参指针，作为形参指针的值，使得形参指针指向实参地址中的数据
  - 被调函数中的指针变量通过间接访问方式，可以修改主调函数的数据

例：交换两个数

```
int main()
{
  int a=3,b=6;
  void swap(int i, int j);
  swap(a, b);
  printf("%d,%d\n",a,b);
  return 0;
}

void swap(int i, int j)
{
   int temp;
   temp=i;
   i=j;
   j=temp;
}
```

i = a; j = b;

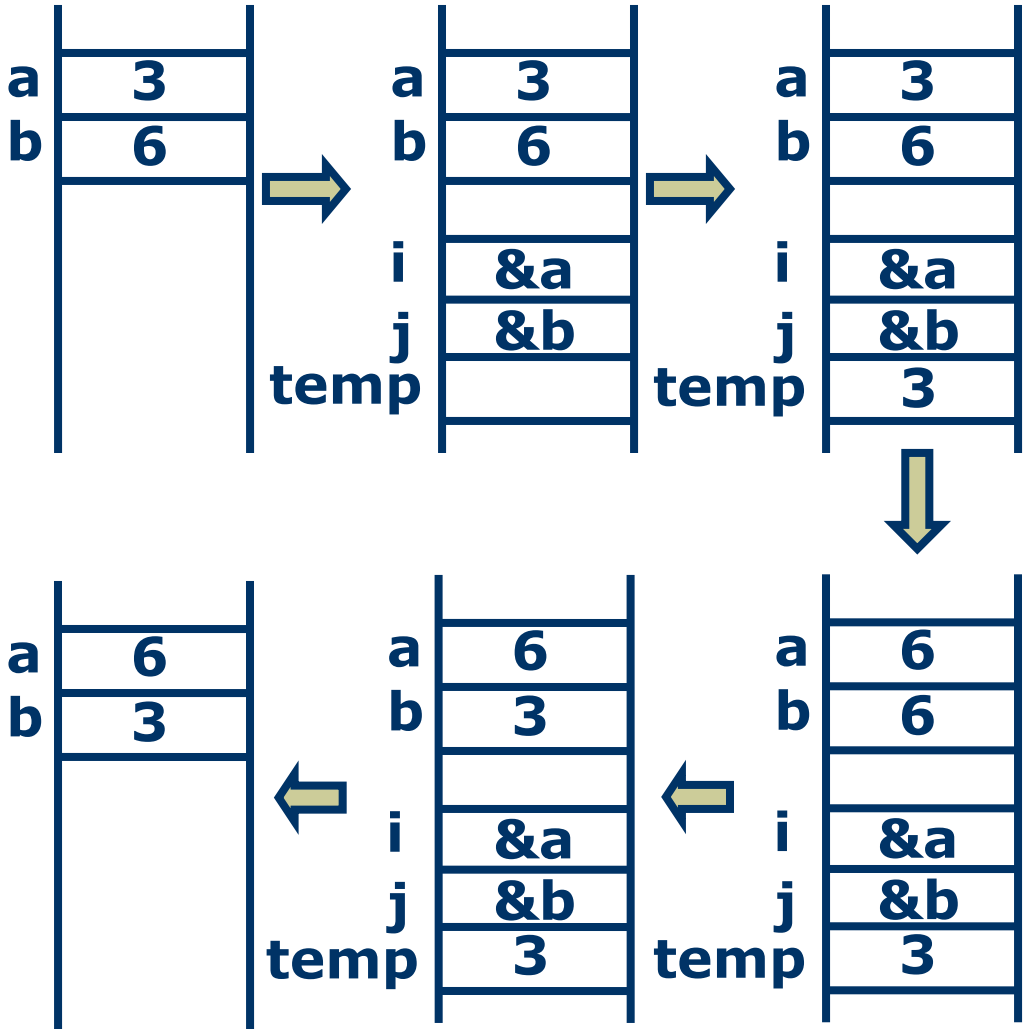


例：交换两个数

```
int main()
{
  int a=3,b=6;
  void swap(int *i, int *j);
  swap(&a, &b);
  printf("%d,%d\n",a,b);
  return 0;
}

void swap(int *i, int *j)
{
  int temp;
  temp=*i;
  *i=*j;
  *j=temp;
}
```

i = &a; j = &b;



3 函数与数组

- 实参 => 形参，函数调用时，用实参初始化形参
- 数组元素作函数实参——值传递
- 数组名作函数参数——地址传递
  - 实参与形参都应用数组名（或指针变量）；
  - 在主调函数与被调函数分别定义数组，且类型应一致；
  - 形参数组大小（多维数组第一维）可不指定；
  - 数组名作参数时，不是传送元素的值，而是把实参数组的起始地址传递给形参，这样两个数组共占同一段内存，形参数[illegible]的值同时改变。

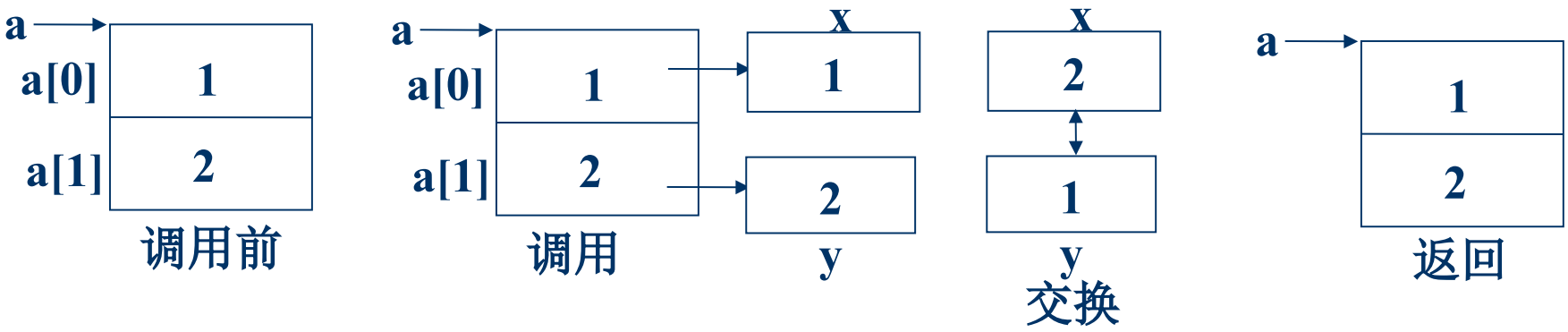
实参	形参
数组名	数组名
数组名	指针变量
指针变量	数组名
指针变量	指针变量

数组元素作函数参数：

- 数组元素与普通变量并无区别，数组元素作为函数实参，与普通变量作为实参是完全相同的。

元素值传递

```
#include <stdio.h>
void swap1(int x,int y){
    int z;
    z=x;      x=y;      y=z;
}
int main( ){
   int a[2]={1,2};
    swap1(a[0],a[1]);
    printf("a[0]=%d\na[1]=%d\n",a[0],a[1]);
    return 0;
}
```

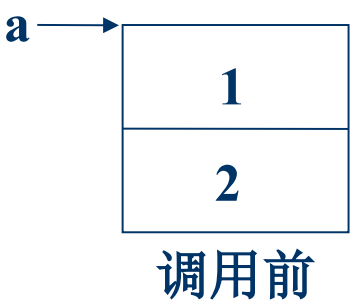


x = a[0]; y = a[1];

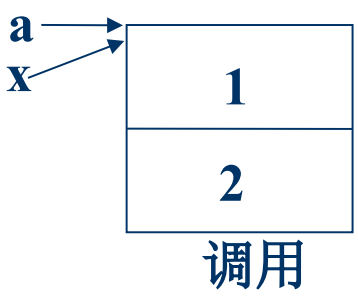
数组名作函数参数：

地址传递

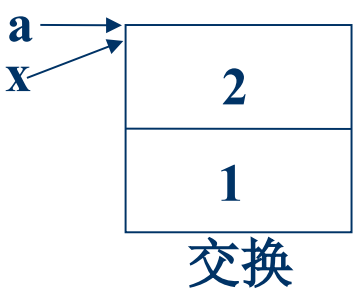
```
#include <stdio.h>
void swap2(int x[]){
    int z;
    z=x[0];     x[0]=x[1];     x[1]=z;
}
int main( ){
     int a[2]={1,2};
    swap2(a);
    printf("a[0]=%d\na[1]=%d\n",a[0],a[1]);
    return 0;
}
```



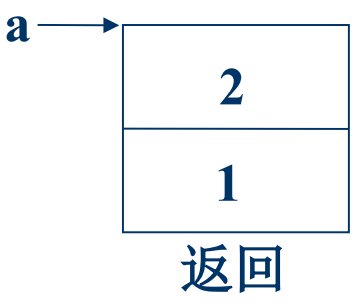
调用前



调用



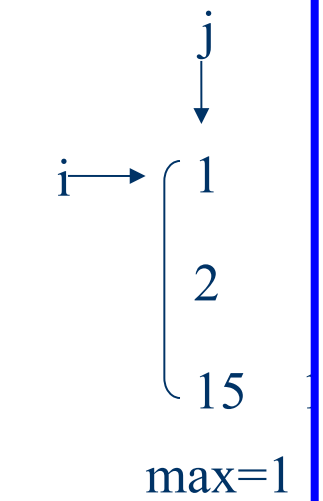
交换

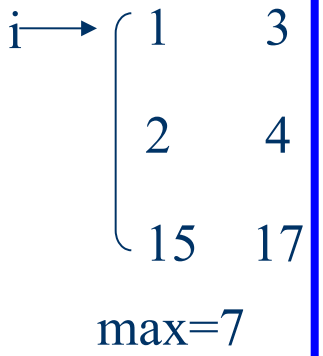


返回

x = a;

例：求二维数组中最大元素值

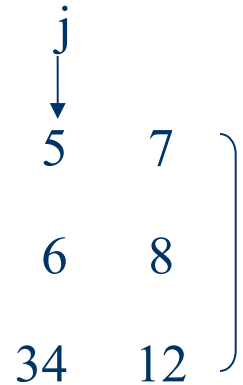


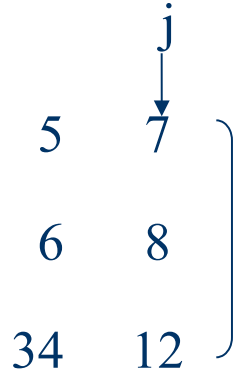



```
int  max_value(int  array[3][4])
{
   int i, j, k, max;
   max=array[0][0];
   for(i=0;i<3;i++)
      for(j=0;j<4;j++)
            if(array[i][j]>max)
              max=array[i][j];
   return(max);
}
int main(void)
{ int a[3][4]={{1,3,5,7},{2,4,6,8},
                      {15,17,34,12}};
   printf("max value is %d\n",max_value(a));
   return 0;
}
```

多维形参数组第一维长度可省略，第二维必须相同

int array[ ][4]





- 写一程序，求一二维矩阵每行元素的平均值，输出二维矩阵的元素以及该矩阵每行元素的平均值。假设输入的矩阵 A 为：

$$A=\begin{bmatrix} 0 & 3 & 6 & 9 & 12 & 15 \\ 2 & 4 & 8 & 10 & 12 & 14 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 6 & 8 & 10 & 12 & 14 \\ 30 & 20 & 25 & 15 & 18 & 10 \end{bmatrix}$$

则程序输出：

The array is :

```
 0   3   6   9  12  15
 2   4   8  10  12  14
 3   7  11  15  19  23
 4   6   8  10  12  14
30  20  25  15  18  10
```

The average of each line is :

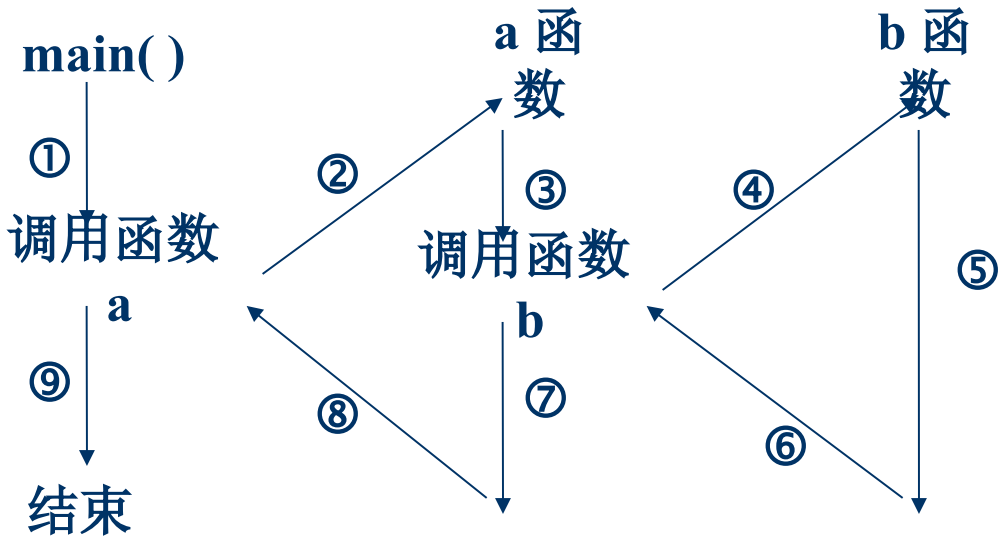
7.50 8.33 13.00 9.00 19.66

```
#include <stdio.h>
void fun(int b[5][6], float r[])
{  int row, col;
   for (row = 0; row < 5; row++)
   {  for (col = 0; col < 6; col++)
      {  r[row] += b[row][col];
       }
       r[row] /= 6.0;
   }
}
int main( )
{
   int row, a[5][6] = {{0,3,6,9,12,15},{2,4,8,10,12,14},
                       {3,7,11,15,19,23},{4,6,8,10,12,14},{30,20,25,15,18,10}};
   float result[5] = {0};
   fun(a, result);
   printf("The average of each line is :   \n");
   for (row = 0; row < 5; row++)
       printf("%.2f\t",result[row]);
   return 0;
}
```

4 函数的嵌套调用和递归调用

➢函数的嵌套调用

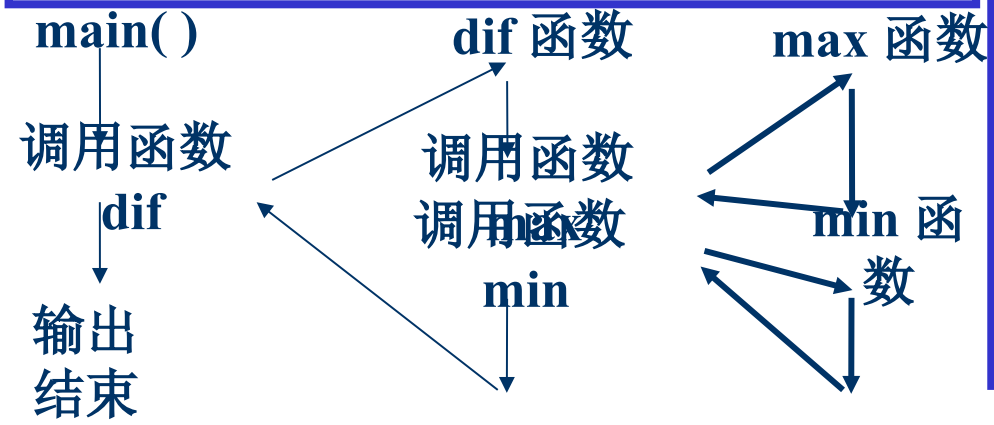
- C 规定：函数定义不可嵌套，但可以嵌套调用函数。



例：求三个数中最大数和最小数的差值

```
#include <stdio.h>
int dif(int x,int y,int z);
int max(int x,int y,int z);
int min(int x,int y,int z);
int main( )
{  int a,b,c,d;
   scanf("%d%d%d",&a,&b,&c);
    d=dif(a,b,c);
    printf("Max-Min=%d\n",d);
    return 0;
}
```

```
int dif(int x,int y,int z)
{
    return max(x,y,z)-min(x,y,z);
}
int max(int x,int y,int z)
{
      int r;
      r=x>y?x:y;
      return(r>z?r:z);
}
int min(int x,int y,int z)
{
      int r;
      r=x<y?x:y;
      return(r<z?r:z);
}
```



函数的递归调用

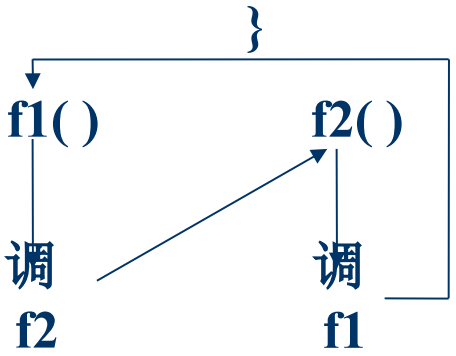
定义：函数直接或间接的调用自身叫函数的递归调用。

```
int  f(int x)
{    int y,z;
      ……
     z=f(y);
     …….
     return(2*z);
}
```




```
int  f1(int x)
{    int y,z;
      ……
     z=f2(y);
     …….
     return(2*z);
}
```

```
int  f2(int t)
{    int a,c;
      ……
     c=f1(a);
     …….
     return(3+c);
}
```



说明： C 编译系统对递归函数的自调用次数没有限制，每调用函数一次，都在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出。

- 例：有 5 个人坐在一起，
- 问第 5 个人多少岁？他说比第 4 个人大 2 岁。
- 问第 4 个人岁数，他说比第 3 个人大 2 岁。
- 问第 3 个人岁数，他说比第 2 个人大 2 岁。
- 问第 2 个人岁数，他说比第 1 个人大 2 岁。
- 最后问第 1 个人，他说是 10 岁。请问第 5 个人多大。
- 显然，这是一个递归问题。
- age(5)=age(4)+2
- age(4)=age(3)+2

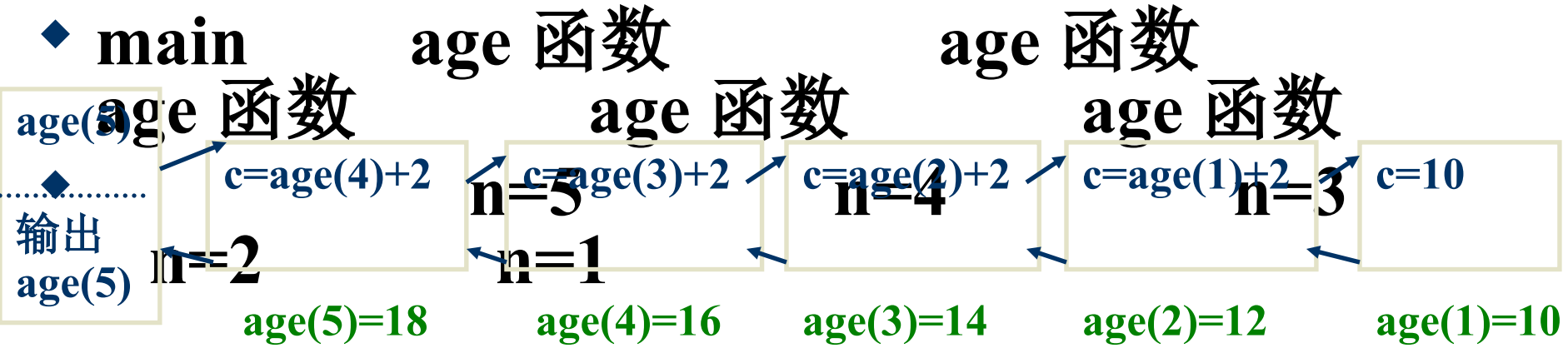
可以用式子表述如下：

$$age(n)=\begin{cases}10 & (n=1)\\ age(n-1)+2 & (n>1)\end{cases}$$

- 可以用一个函数描述上述递归过程：
- int age(int n)　　/* 求年龄的递归函数 */
- { int c;　　/*c 用作存放函数的返回值的变量 */
- 　if (n==1) c=10;
- 　else c=age(n-1)+2;
- 　return(c);
- }
- int main( )
- {
- 　printf("%d", age(5));
- 　return 0 ;
- }
- 运行结果：
- 18



- 函数调用过程如图：


main
age 函数
age 函数
age 函数
age 函数
age 函数
age(5)
.....
输出
age(5)
c=age(4)+2
c=age(3)+2
c=age(2)+2
c=age(1)+2
c=10
n=5
n=4
n=3
n=2
n=1
age(5)=18
age(4)=16
age(3)=14
age(2)=12
age(1)=10

-

小结

- 函数定义的一般形式：
  [extern/static] 类型说明符 函数名（形参类型说明表）
  - { 函数体 }
- 函数声明的一般形式：
  - [extern] 类型说明符 函数名 ([形参表]) ;
- 函数调用的一般形式： 函数名 ([实参表]) ;
- 函数的参数分为形参和实参两种，形参出现在函数定义中，实参出现在函数调用中，发生函数调用时，将把实参的值传送给形参。
- 函数的值是指函数的返回值，它是在函数中由 return 语句返回的。

```
#include <stdio.h>
 long   sum(int a, int b);
 long   factorial(int n);
int main( )
{     int n1,n2; long a;
      long   sum(int a, int b);
      scanf("%d%d",&n1,&n2);
      a=sum(n1,n2);
      printf("a=%ld",a);
      return 0;
}
long sum(int a,int b)
{    long   factorial(int n);
     long c1,c2;
     c1=factorial(a);
     c2=factorial(b);
     return(c1+c2);
}
```

文件包含编译预处理命令

函数声明

函数调用

实参

函数定义

形参

函数调用

函数返回值

```
long factorial(int n)
{    long rtn=1;
     int i;
     for(i=1;i<=n;i++)
     rtn*=i;
     return(rtn);
}
```