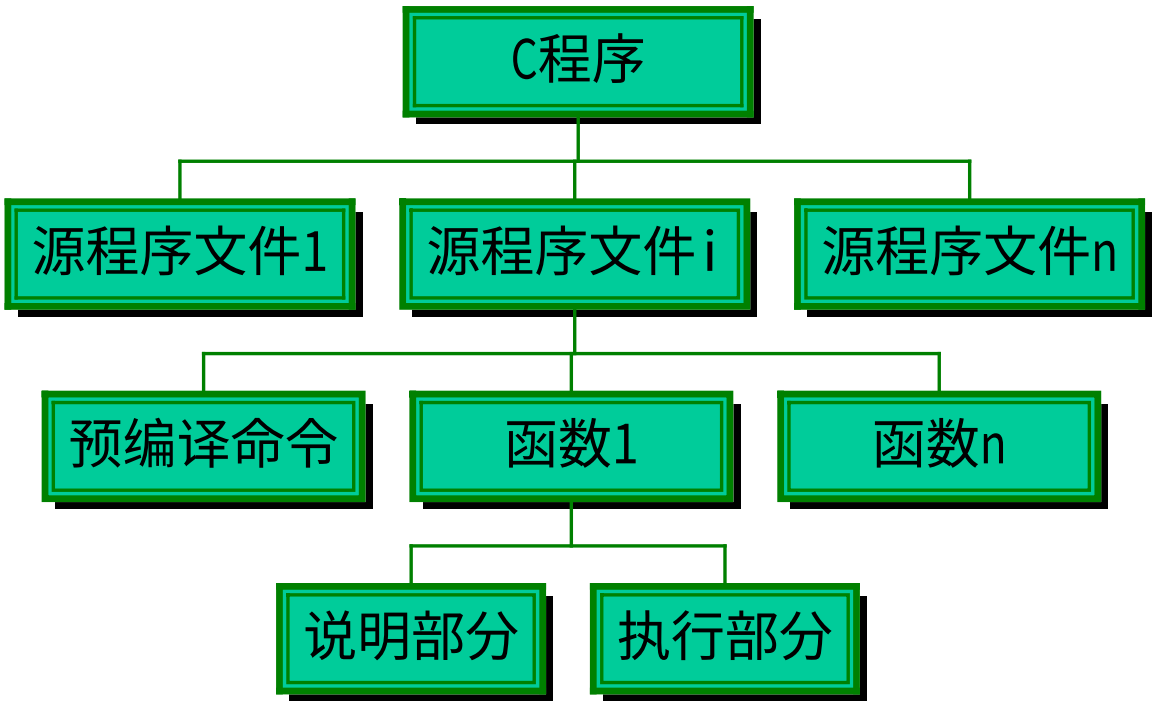
函数

模块化程序设计



C程序结构

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，
- 特点：
  - 各模块相对独立、功能单一、结构清晰、接口简单
  - 降低了程序设计的复杂性
  - 提高元件的可靠性
  - 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

特点

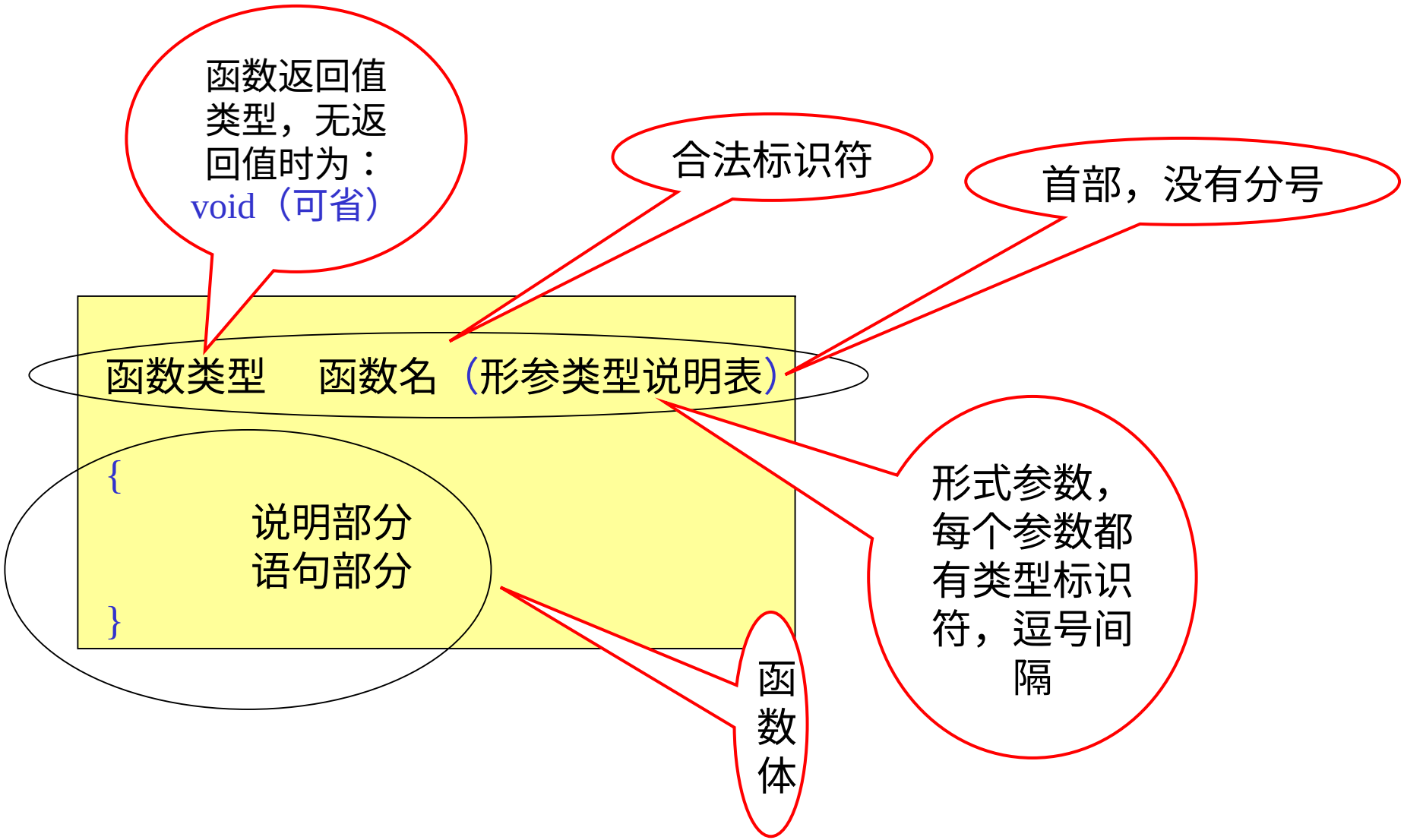
- C是函数式语言
- 一个程序由多个函数组成
- 必须有且只能有一个名为main的主函数
- C程序的执行总是从main函数开始，在main中结束
- 函数不能嵌套定义,可以嵌套调用
- 所有函数都是平行结构

– 函数分类
  - 从用户角度
    – 标准函数（库函数）：由系统提供，include 包含
    – 用户自定义函数
  - 从函数形式
    – 无参函数
    – 有参函数

什么时候定义独立的函数

- 当某一功能要被反复使用时（如输入/输出）。
- 当某一功能相对独立时（复杂问题的划分）。

函数的定义——一般格式


函数返回值类型，无返回值时为：void（可省）
合法标识符
首部，没有分号
函数类型 函数名（形参类型说明表）
{
说明部分
语句部分
}
形式参数，每个参数都有类型标识符，逗号间隔
函数体

例　无参函数

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

或

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

例　空函数

```
 dummy( )
{ }
```

函数体为空

例　有参函数（现代风格）

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

int x, int y

(×)

例子

- 计算两个整数的最大值

```
int  imax (int x, int y)
{
      int temp;
      if (x>y)  temp=x;
      else  temp=y;
      return (temp);
}
```

说明

函数首部（三个部分）

- 类型标识符（可有/可无）：表明函数的结果类型。如果函数名前带有类型标识符，函数体内必须对应有 return 语句返回一致类型的表达式值。
- 函数名部分：满足标识符的命名规则
- 参数表部分：表示形式参数，指明自变量的类型和名称，特别注意，每个自变量前必须有类型标识符（不能写成：int imax (int x, y) ）。参数表内可以没有参数，但是括号必须出现。函数的参数只在函数体内起作用！

传统风格(最好不用)

```
函数类型    函数名（形参表）
形参类型说明
{
        说明部分
        语句部分
}
```

例　有参函数（传统风格）

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

“形参”与“实参”

- 形参：定义函数时，在参数表中出现的参数；变量形式。
- 实参：调用函数时对应的参数；可以是表达式形式。
- 形参与实参的严格对应关系！

对函数的调用

- 基本格式：
  函数名(参列表)
- 出现的位置：
  - 以语句形式出现；
  - 以表达式形式出现（包括出现在其它函数参数中）；
- 对库函数的调用，需要用 #include 包含库的名字；
- 调用函数中的参数，称为实在参数（实参）

例子

```
#include <stdio.h>
int imax (int, int)；  // 函数原型
void main (void)   // 主函数, 没有参数，没有类型
 {
    int x1,x2,x3,x4,z;
    scanf(“%d,%d,%d,%d”,&x1,&x2,&x3,&x4);
    z = imax(imax(x1,x2),imax(x3,x4));  //调用
    printf(“%d, %d, %d, %d, %d\n”,x1,x2,x3,x4,z);
  } // 可以将 z 赋值语句的右部直接取代 printf 中的  z
int imax (int x,int y)   // 计算最大值函数的定义
{
    return(x>y?x:y);
}
```

实参

形参

函数原型声明

- 被调用的函数在后面定义，则被调用之前一定要有一个原型声明。原型声明(不要忘记分号;)有两种方式：
  - 直接声明（见前面例子，）,格式：

  函数类型 函数名(参数类型1,参数类型2,…,);

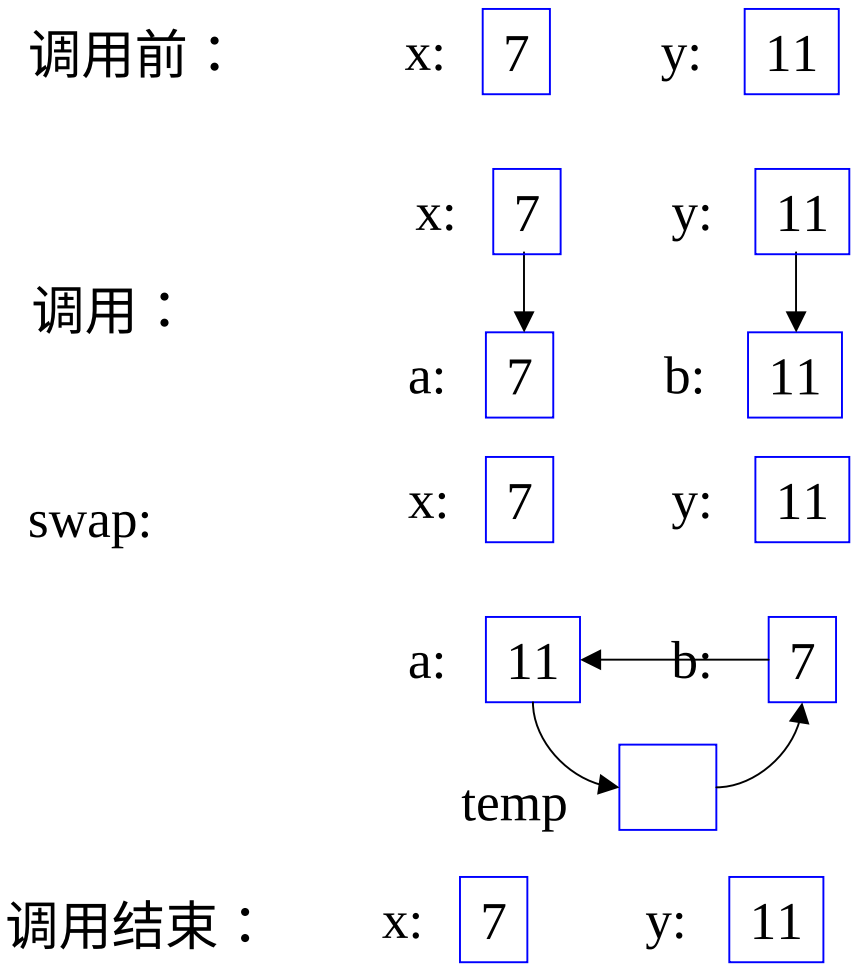
  函数类型 函数名(参数类型1,参数1，参数类型2,参数2,…,);
  - 间接声明（用 #include ）
- 如被调用函数先定义后调用，则可以不声明原型。
- 为什么需要函数原型（先定义，后使用规范！）

参数传递(续)

- 形参传递
  - 实际参数传给形式参数是传值，不是传地址！因此，每个实参会分配临时存储单元。函数调用完后释放。实参也不要求是变量，可以是表达式
  - 实参与形参个数相同
  - 实参与形参满足赋值兼容性以及内部类型转换原则；
  - 实参不能回传值，参数传递是单向的。

交换两个数（不能交换！）

```
/*ch7_2.c*/
#include <stdio.h>
main()
{   int x=7,y=11;
    printf("x=%d,\ty=%d\n",x,y);
    printf("swapped:\n");
    swap(x,y);
    printf("x=%d,\ty=%d\n",x,y);
}
swap(int a,int b)
{   int temp;
    temp=a; a=b; b=temp;
}
```



函数的返回值

- 返回语句
  - 形式： return(表达式)；
    或 return 表达式;
    或 return; //没有返回值
  - 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
    - 若函数类型与return语句中表达式值的类型不一致，按前者为准，自动转换------函数调用转换
    - void型函数

无返回值

```
 printstar()
{   printf("**********");
}
main()
{   int a=10;
    printstar();
    printf("%2d",a);
}
```

输出：********** 10

```
void  printstar()
{   printf("**********");
}
main()
{   int a;
    a=printstar();
    printf("%d",a);
}
```

编译错误！

例 函数返回值类型转换 (计算两个数的最大值)

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

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

函数常出现的位置

- 函数语句：

  例　printstar();

  　　printf(“Hello,World!\n”); //不需要返回值

- 函数表达式（必须返回值）：

  例　m=max(a,b)*2;

- 函数参数：

  例　printf(“%d”,max(a,b)); // 必须返回值

  　　m=max(a,max(b,c));　// 嵌套调用，三个数的最大值

详解

```
#include <stdio.h>
 long   sum(int a, int b);
 long   factorial(int n);
main()
  {    int n1,n2;
       long a;

       scanf("%d,%d",&n1,&n2);
       a=sum(n1,n2);
       printf("a=%1d",a);
  }
long sum(int a,int b)
{
     long c1,c2;
    c1=factorial(a);
    c2=factorial(b);
    return(c1+c2);
}
```

文件包含编译预处理命令

函数原型说明

函数调用

实参

函数定义

形参

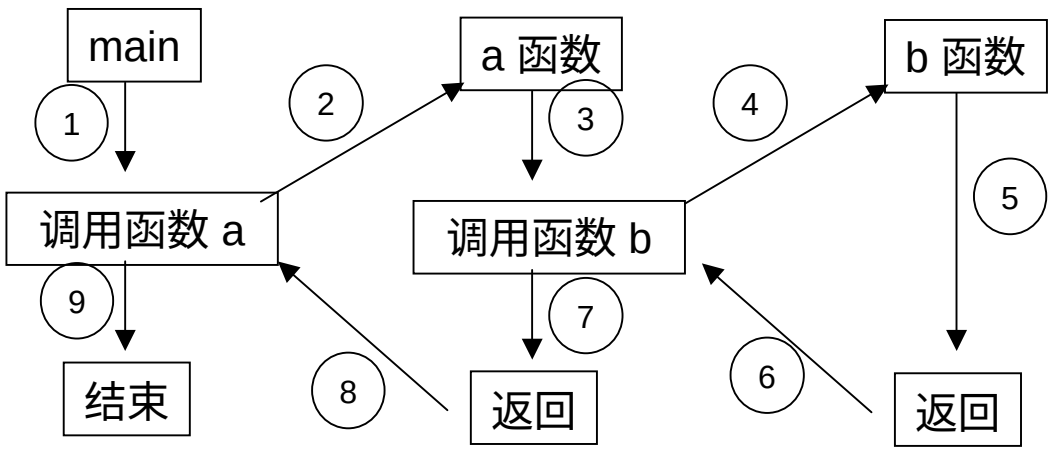
函数调用

函数返回值

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

函数的嵌套

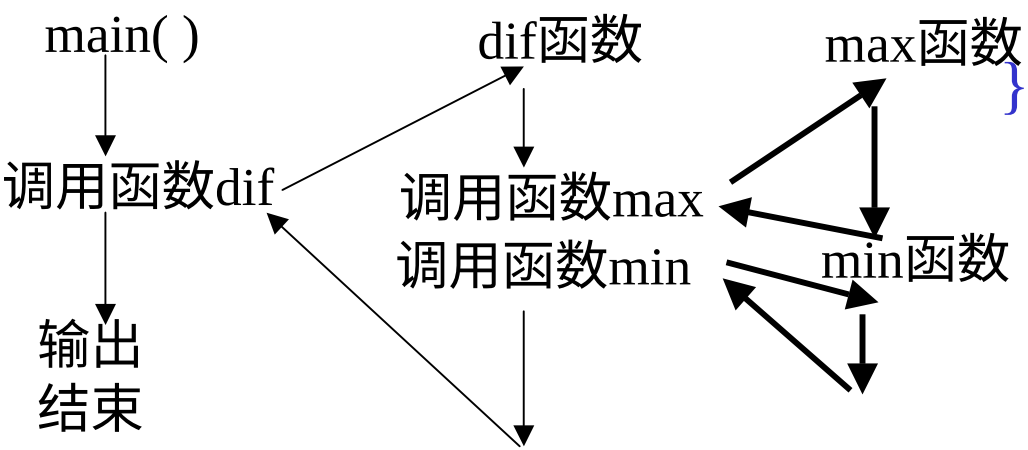
- C 语言的函数都是并列的结构，不能嵌套定义。但可以嵌套调用。
- 注意，像 imax(imax(x1,x2),imax(x3,x4))，一般不看成为函数嵌套（必须在定义时调用）。



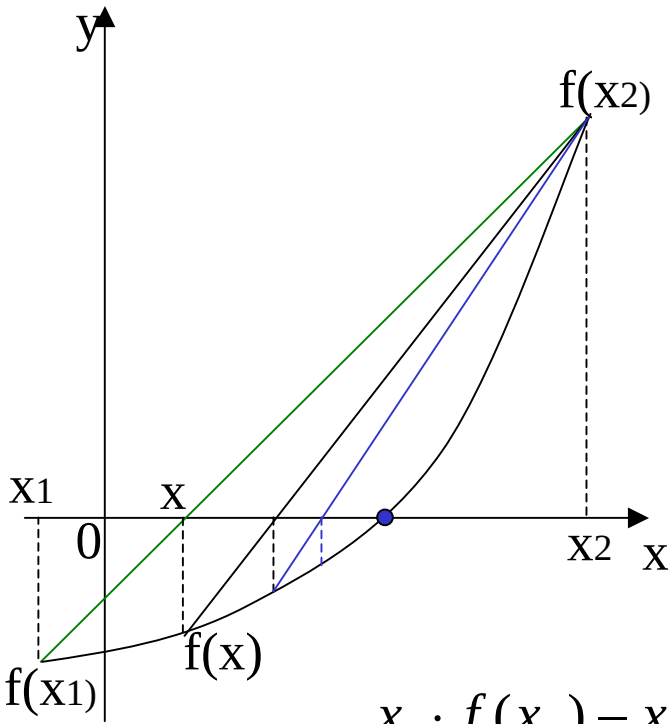
例　求三个数中最大数和最小数的差值

```
#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);
void main()
 {  int a,b,c,d;
    scanf("%d%d%d",&a,&b,&c);
    d=dif(a,b,c);
    printf("Max-Min=%d\n",d);
 }
```

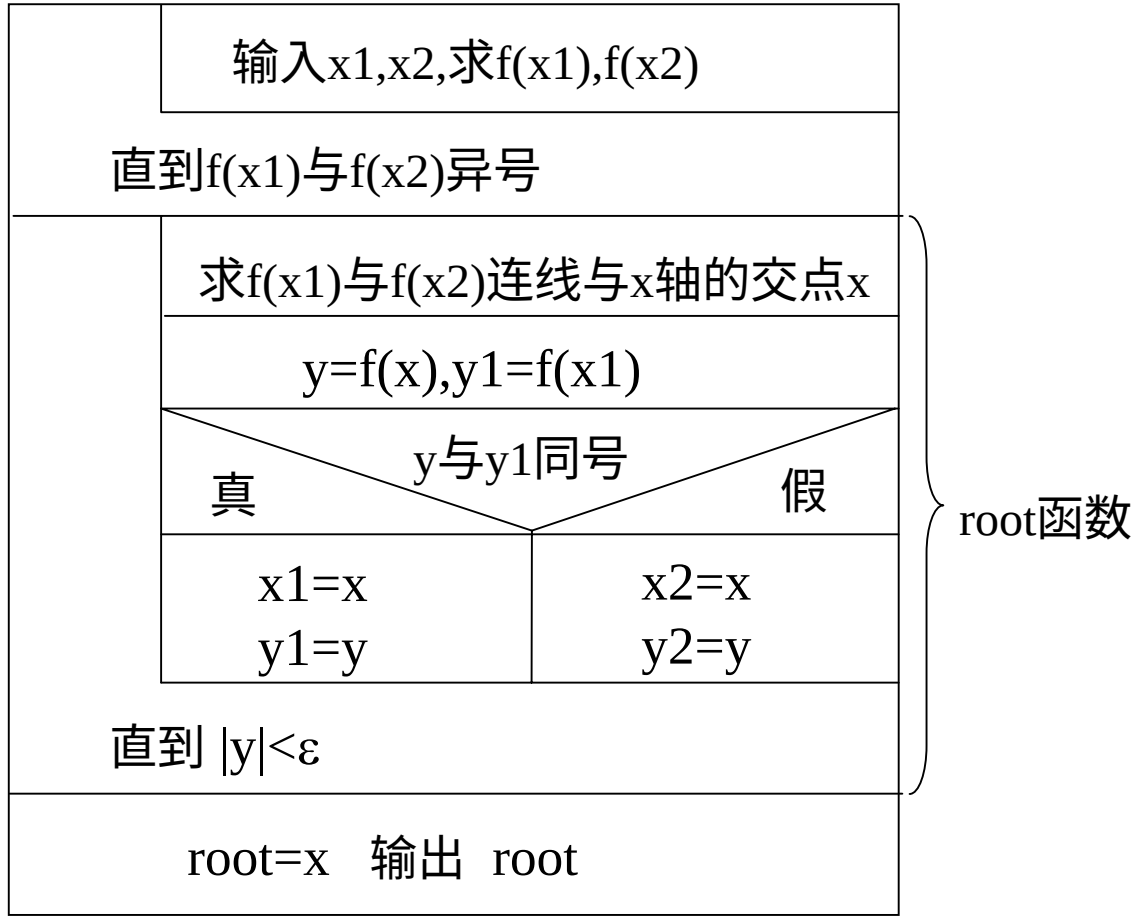
```
int dif(int x,int y,int z)
{  return max(x,y,z)-min(x,y,z); }
int max(int x,int y,int z)
 {    int r;
      r=x>y?x:y;
      return(r>z?r:z);
 }
int min(int x,int y,int z)
 {   int r;
      r=x<y?x:y;
      return(r<z?r:z);
 }
```

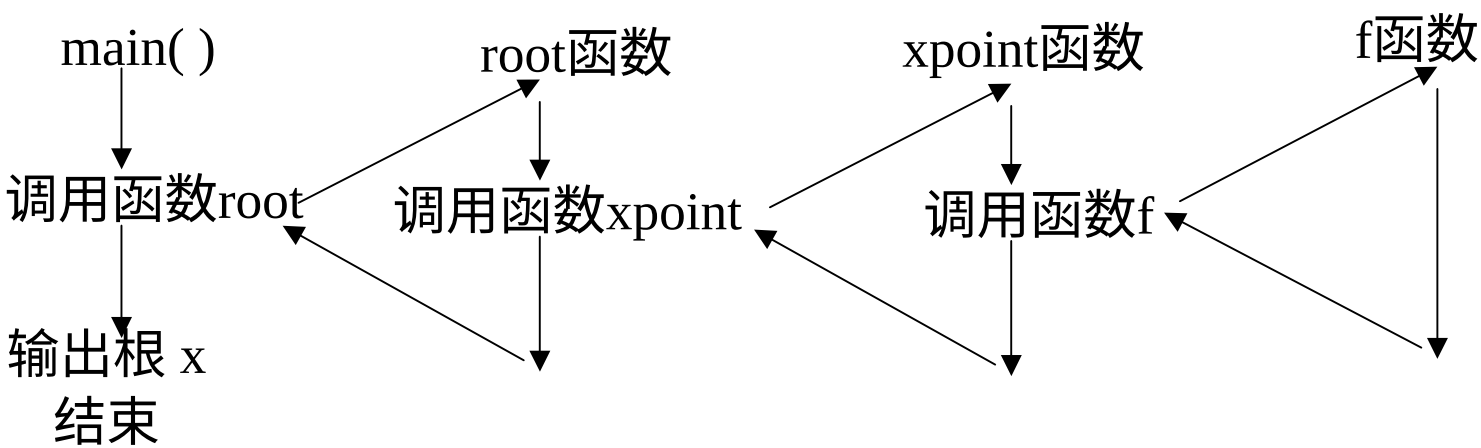


例 用弦截法求方程根： $X^3-5x^2+16x-80=0$



$$x=\frac{x_1\cdot f(x_2)-x_2\cdot f(x_1)}{f(x_2)-f(x_1)}$$


输入x1,x2,求f(x1),f(x2)
直到f(x1)与f(x2)异号
求f(x1)与f(x2)连线与x轴的交点x
y=f(x),y1=f(x1)
y与y1同号
真
假
x1=x
y1=y
x2=x
y2=y
直到 |y|<ε
root=x 输出 root
root函数


main( )
调用函数root
输出根 x
结束
root函数
调用函数xpoint
xpoint函数
调用函数f
f函数

递归

- 强有力的数学表示手段；
- 例子：
  - 例1，计算 1+2+…+n ，描述为：

    F(n) = 1+2+…+n，于是，f(1) = 1, f(n) = f(n-1)+1
  - 例2，计算 n!，如何表示？
  - 冒泡排序？
- 基本方法：2-Step，
  - 基始值（初始值）定义；
  - 递归定义（向初始值靠拢）

C 语言函数的递归

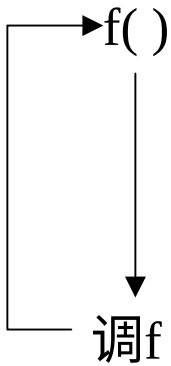
- 特点：嵌套调用中，存在自己调用自己的语句
  - 间接递归：

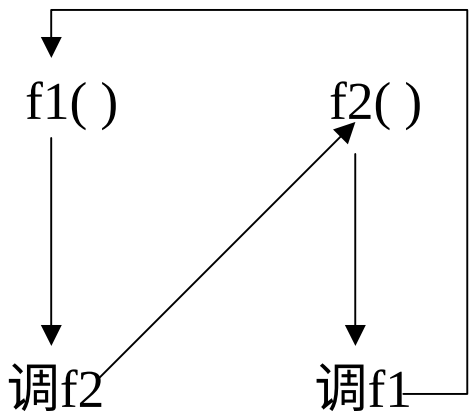
    A 调用 B， B 又调用 A 的方式
  - 直接递归：函数直接调用自身（A 调用 A）
- 函数递归一定要保证函数向结束的方向逼近，收敛于某一点。
  - (n-1)! =n!/n 是要求的递归吗？

```
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);
}
```





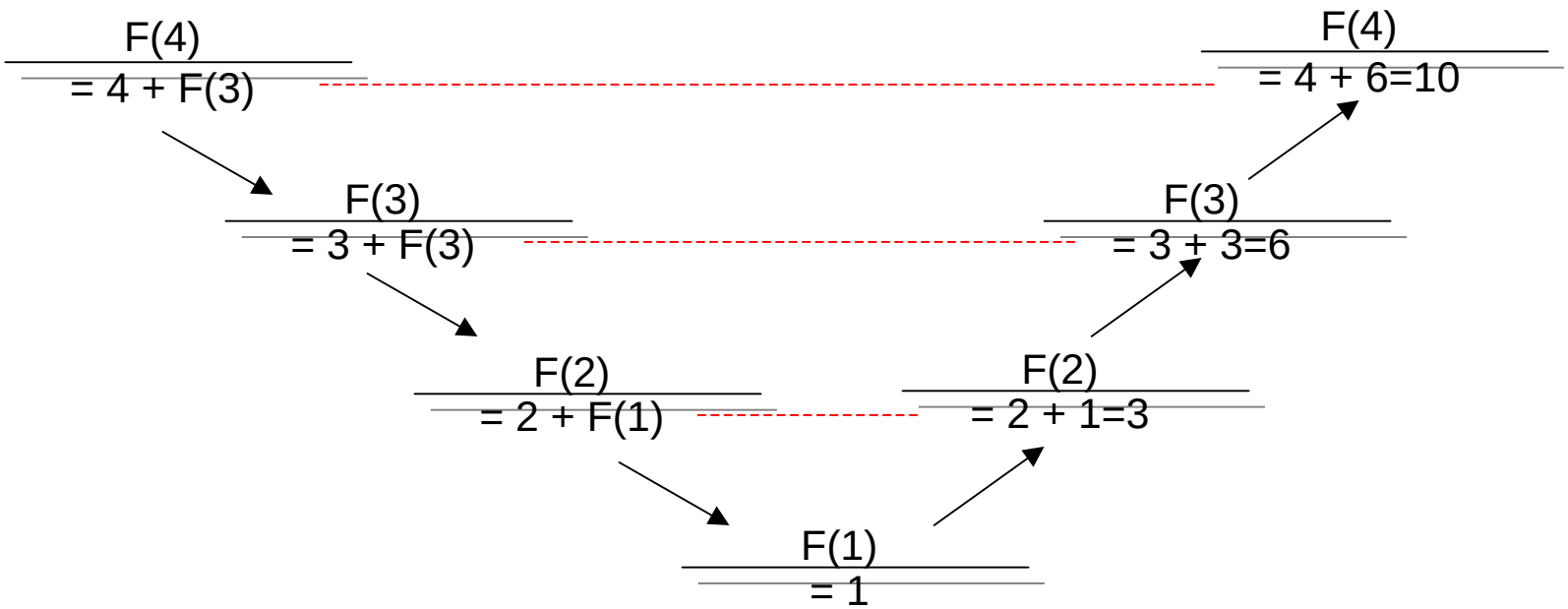
- 说明
  - C编译系统对递归函数的自调用次数没有限制
  - 每调用函数一次，在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出

- 计算 1+2+…+n (n>0, n 为正整数)
- 定义：

$$f(n)=\begin{cases}1, & \text{当 } n=1 \text{ 时}\\ n+f(n-1), & \text{当 } n>1 \text{ 时}\end{cases}$$

- 含义：要计算 f(n)，如果能先计算 f(n-1)，就能计算 f(n)。因此先计算 f(n-1)

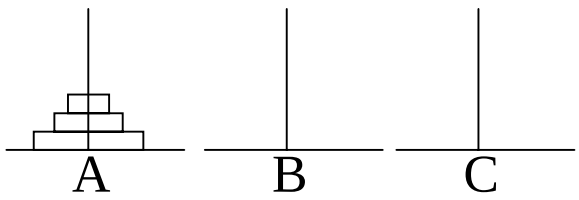
```
#include <stdio.h>
int f(int n)
{
  if (n=1) return(1);
 else return (n + f(n-1));  // 向初始值 f(1) 逼近。
}
//假定要计算 f(4), 如下是计算步骤：
main()
{   int n, y;
    printf("Input a integer number:");
    scanf("%d",&n);
    y=f(n);
    printf("%d! =%15d",n,y);
}
```


F(4)
= 4 + F(3)
F(3)
= 3 + F(3)
F(2)
= 2 + F(1)
F(1)
= 1
F(2)
= 2 + 1=3
F(3)
= 3 + 3=6
F(4)
= 4 + 6=10

例

- 冒泡排序（？）
- 计算 n!（自己看书，模仿前面例子，画出调用与返回的示意图）

例 Hanoi问题




```
void move(char  getone, char  putone)
{   printf("%c--->%c\n",getone,putone); }
void hanoi(int n,char one,char two,char three)
{   if(n==1)  move(one,three);
    else
    {     hanoi(n-1,one,three,two);
          move(one,three);
          hanoi(n-1,two,one,three);
    }
}
main()
{   int m;
    printf("Input the number of disks:");
    scanf("%d",&m);
    printf("The steps to moving %3d
disks:\n",m);
    hanoi(m,'A','B','C');
}
```

数组作为参数传递

- 数组元素的传递仍然按值传递。
- 数组整体传递按（起始）地址传递，因此，函数内参数值的变化也意味着外部参数的变化；
- 数组传递时，编译不对实参作大小检查，但建议保持一致；
- 形参是一维数组时，可以不指定大小，若是多维数组，第一维可以不指定大小，但其它维必须指定。

数组元素作为参数传递——值传递

例 两个数组大小比较

a和b为有10个元素的整型数组
比较两数组对应元素
变量n,m,k记录a[i]>b[i], a[i]==b[i], a[i]<b[i]的个数
最后　　若n>k,认为数组a>b
　　若n<k,认为数组a<b
　　若n==k,认为数组a==b

```
#include <stdio.h>
main()
{    int a[10],b[10],i,n=0,m=0,k=0;
     printf("Enter array a:\n");
     for(i=0;i<10;i++)
           scanf("%d",&a[i]);
    printf("Enter array b:\n");
     for(i=0;i<10;i++)
           scanf("%d",&b[i]);
     for(i=0;i<10;i++)
     {   if(large(a[i],b[i])==1)  n=n+1;
        else if(large(a[i],b[i])==0)  m=m+1;
        else k=k+1;
     }
     /*  Output  */
}
```

```
int  large(int x,int y)
{    int flag;
     if(x>y)  flag=1;
     else if(x<y) flag=-1;
     else flag=0;
     return(flag);
}
```

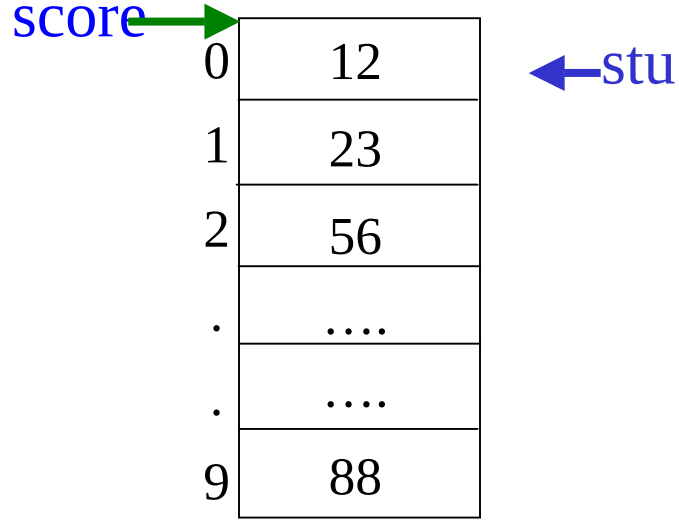
数组名作函数参数——地址传递

```
#include <stdio.h>
float average(int stu[10], int n);
void main()
{ int score[10], i;
   float   av;
   printf("Input  10  scores：\n");
   for( i=0; i<10; i++ )
       scanf("%d", &score[i]);
   av=average(score,10);
   printf("Average  is：%.2f", av);
}
```

计算平均成绩

实参用数组名

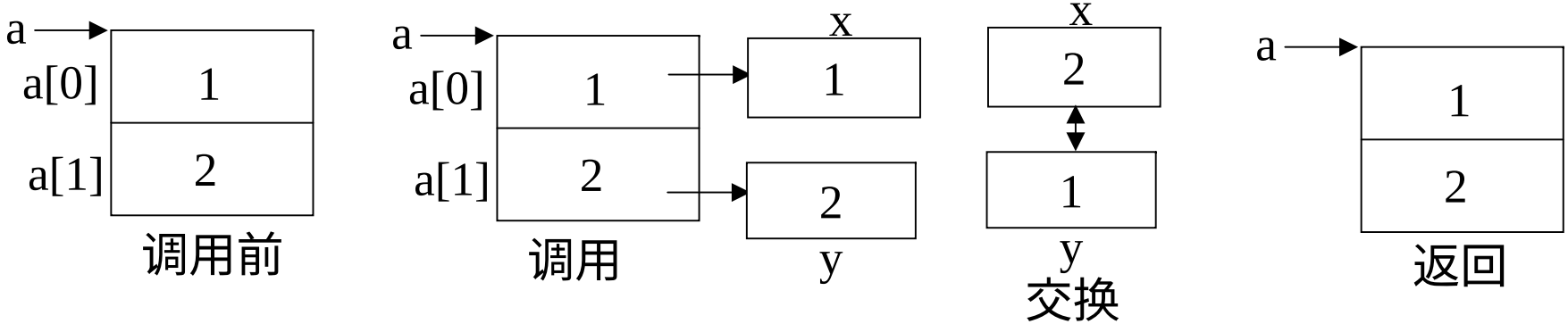
```
float   average(int stu[ ], int n)
 { int i;
   float av,total=0;
   for( i=0; i<n; i++ )
       total += stu[i];
   av = total/n;
   return av;
 }
```



数组元素与数组传递比较

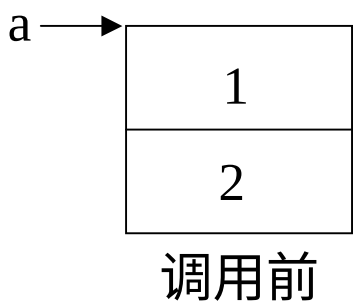
值传递，
不带回值

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

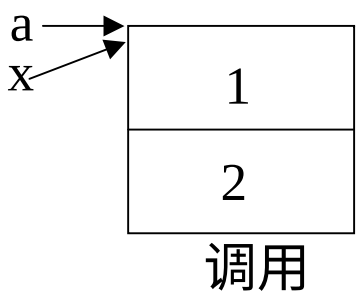


地址传递，
带回值

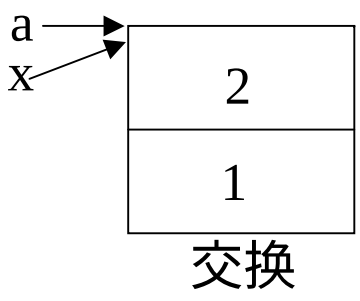
```
#include <stdio.h>
void swap2(int x[])
{   int z;
    z=x[0];     x[0]=x[1];     x[1]=z;
}
main()
{   int a[2]={1,2};
    swap2(a);
    printf("a[0]=%d\na[1]=%d\n",a[0],a[1]);
}
```



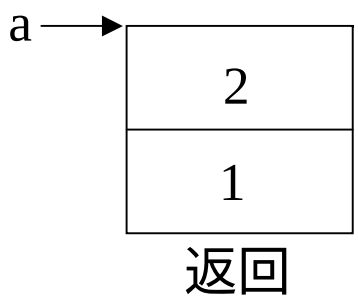
调用前



调用

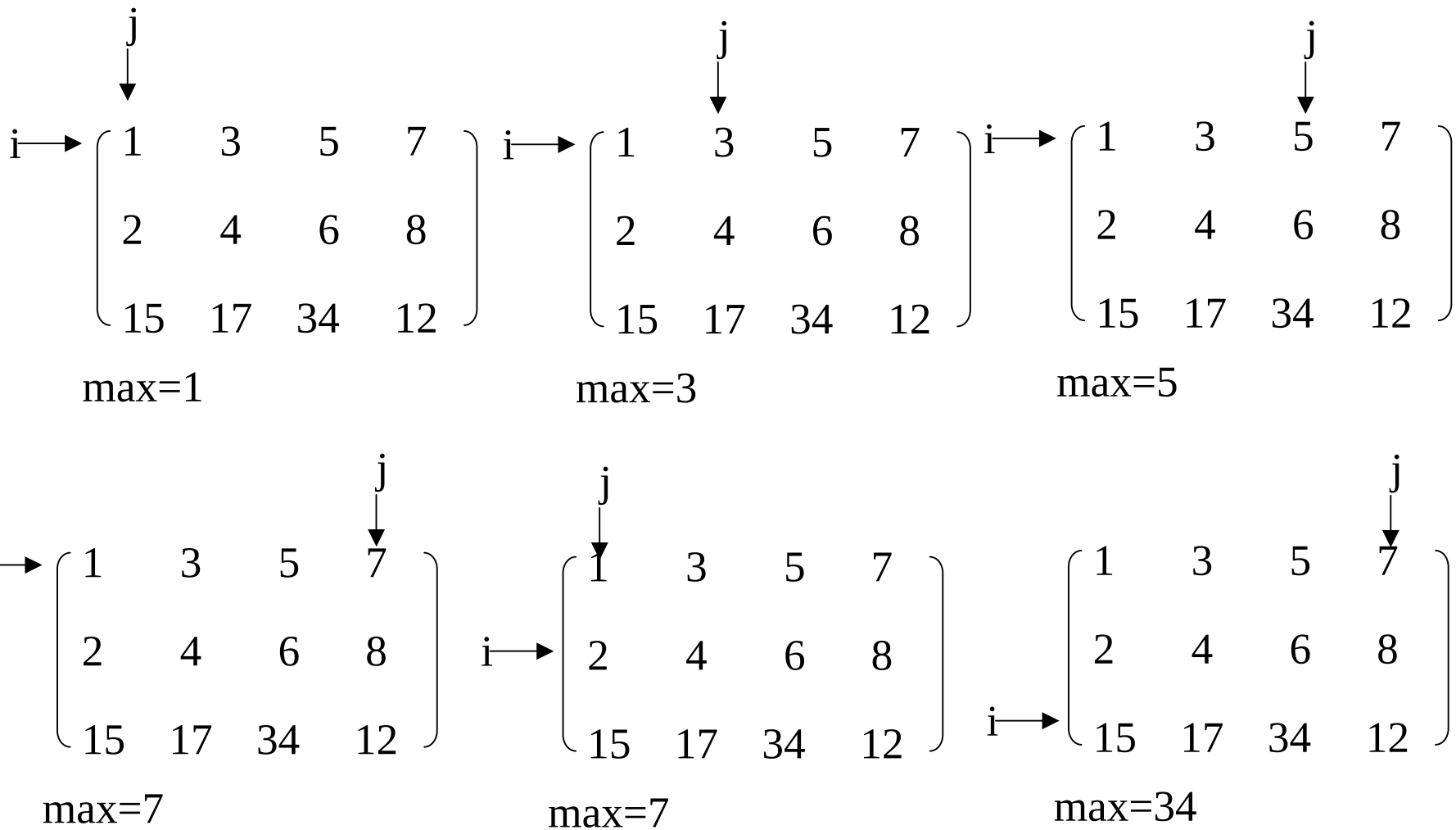


交换



返回

例 求二维数组中最大元素值



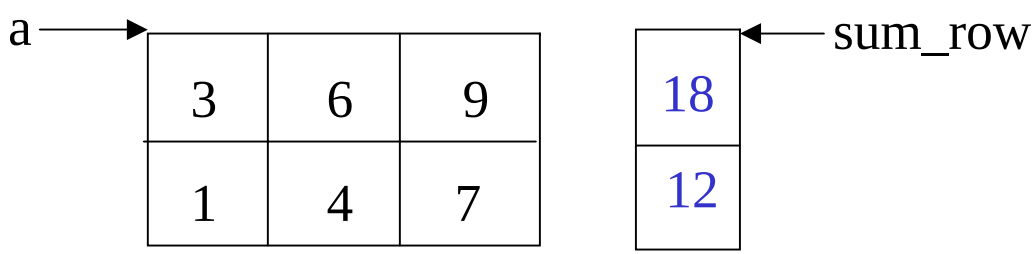
```
int  max_value(int  array[3][4])
{   int i,j,k,max;
    max=array[0][0];
    for(i=0;i<3;i++)
       for(j=0;j<4;j++)
            if(array[i][j]>max)
              max=array[i][j];
    return(max);
}
main()
{ int a[3][4]={{1,3,5,7},
                    {2,4,6,8},{15,17,34,12}};
    printf("max value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

多维形参数组第一维维数可省略,第二维必须相同

⇔ int array[][4]

例 求二维数组中各行元素之和

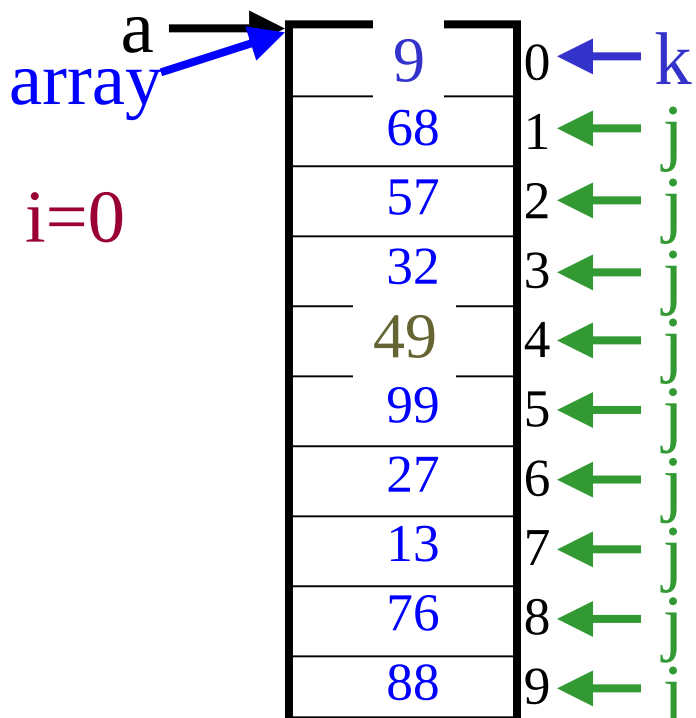
```
get_sum_row(int  x[][3], int  result[] ,int  row, int  col)
{   int i,j;
    for(i=0;i<row;i++)
    {     result[i]=0;
          for(j=0;j<col;j++)
               result[i]+=x[i][j];
    }
}
main()
{   int a[2][3]={3,6,9,1,4,7};
    int sum_row[2],row=2,col=3,i;
    get_sum_row(a,sum_row,row,col);
    for(i=0;i<row;i++)
      printf("The sum of row[%d]=%d\n",i+1,sum_row[i]);
}
```

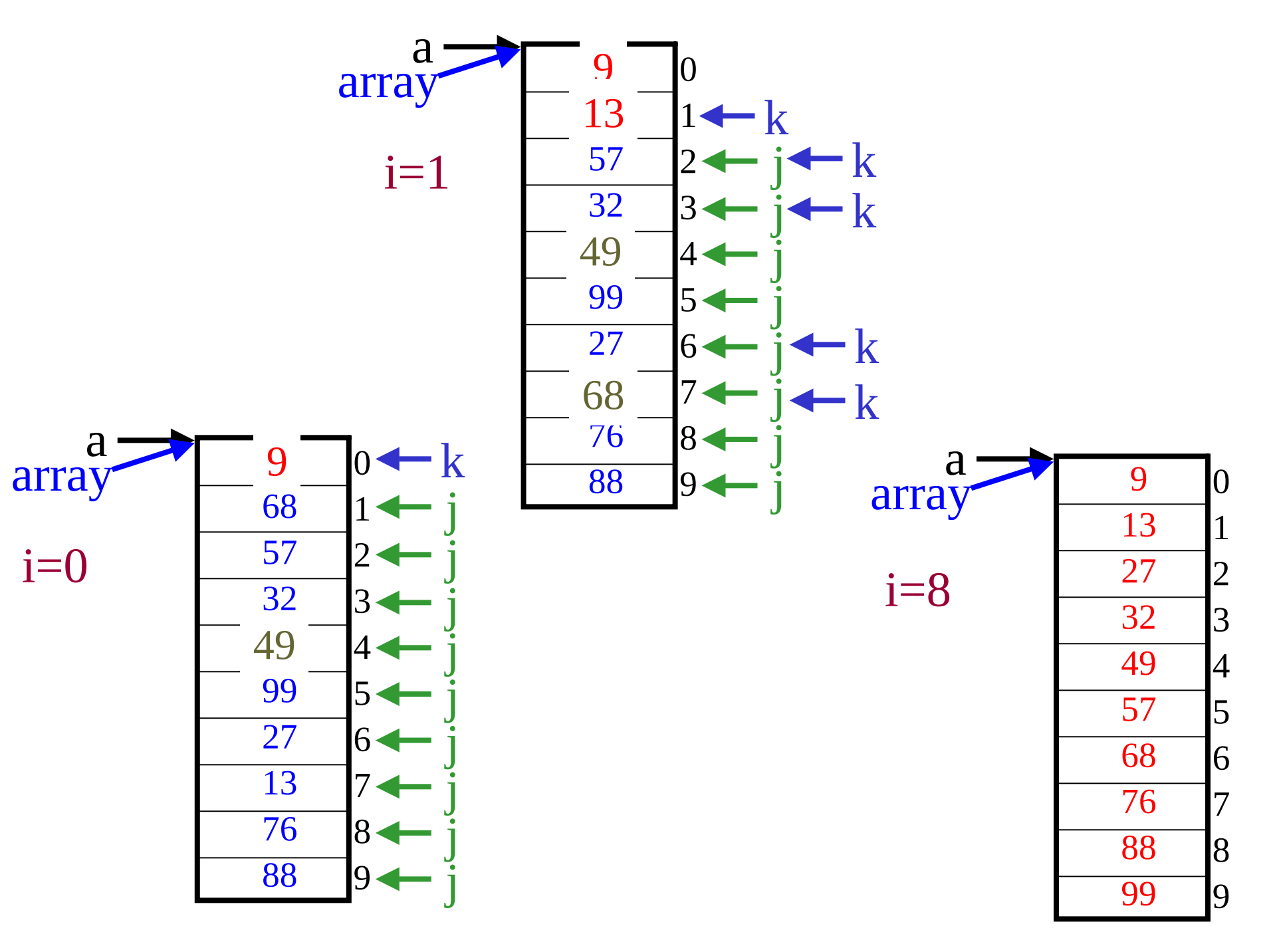


数组排序----简单选择排序

```
void sort(int  array[],int  n)
{    int i,j,k,t;
    for(i=0;i<n-1;i++)
    {     k=i;
          for(j=i+1;j<n;j++)
            if(array[j]<array[k])   k=j;
          if(k!=i)
          {    t=array[i];
               array[i]=array[k];
               array[k]=t;
          }
    }
}
```

```
main()
{   int a[10],i;
    for(i=0;i<10;i++)
          scanf("%d",&a[i]);
    sort(a,10);
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");
}
```




a
array
i=1
9
13
57
32
49
99
27
68
76
88
0
1
2
3
4
5
6
7
8
9
k
j
a
array
i=0
9
68
57
32
49
99
27
13
76
88
0
1
2
3
4
5
6
7
8
9
k
j
a
array
i=8
9
13
27
32
49
57
68
76
88
99
0
1
2
3
4
5
6
7
8
9

作业

- P186, T8.1, T8.3
- 写一个递归函数，判断一个串是否是回文串（正向看与逆向看相同），如：

abccba, abcba, 1357531 等