第8章 函数

- 概述
- 函数定义的一般形式
- 函数参数和函数的值
- 函数的调用
- 函数的嵌套调用
- 函数的递归调用
- 数组作为函数参数
- 局部变量和全局变量
- 变量的存储类别
- 内部函数和外部函数
- 运行一个多文件的程序

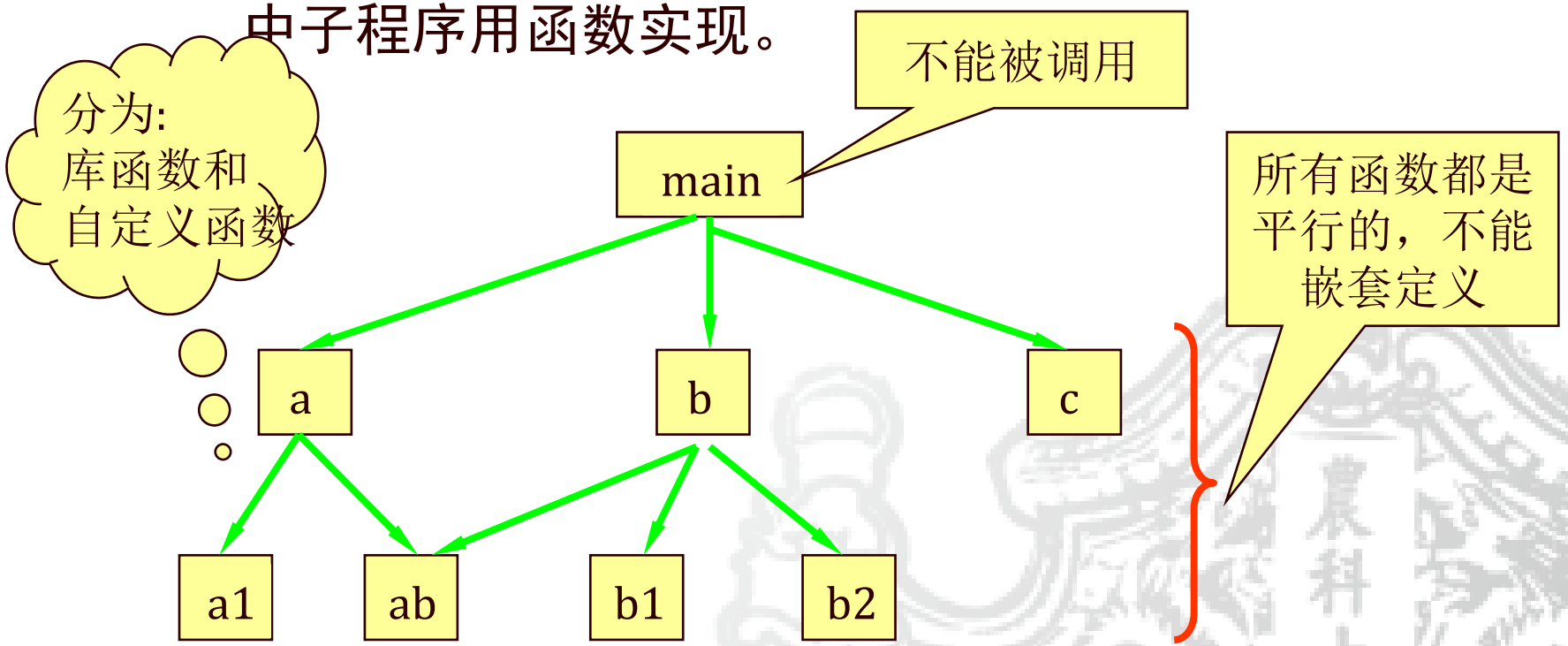
学习目标：

- ★认识到函数是一种简化程序结构的重要手段；
- ★理解函数调用和函数调用过程中的参数传递；
- ★理解函数原型(声明)和怎样写函数原型；
- ★能够用前几章的知识实现简单的函数；
- ★能够用return语句实现函数的返回值；
- ★能够理解函数调用过程中形式参数和实际参数的关系，理解数组名作为函数参数时代表的意义；
- ★能够理解函数的嵌套调用和递归调用机制 。

§ 8.1 概述

★函数的概念

一个大的程序一般应分为若干个程序模块，每个模块实现一个特定的功能，这些模块称为子程序，在C语言中子程序用函数实现。



一、常规方法：各函数包含在一个文件中

例1

```
#include <stdio.h>
void printstar( )
{printf ("**************************\ n" ); }

void print_message( )
{printf ("_____How_do_you_do!\n") ;}

int main( )
{
  printstar( );
  print_message( );
  printstar( );
  return 0;
}
```

运行结果：

```
**************************
How do you do!
**************************
```

一次函数定义
多次函数调用

★几点说明：

(1)一个源文件由一个或者多个函数组成。

(2)一个C程序由一个源文件组成。

(3)C程序的执行从main 函数开始。

(4)所有的子函数都是平行的。

(5)从用户的角度看，函数分库函数和自定义函数。

(6)函数形式：

①**无参函数**：主调函数无数据传送给被调函数,可带或不带返回值。

②**有参函数**：主调函数与被调函数间有参数传递,主调函数可将实参传送给被调函数的形参,被调函数的数据可返回主调函数。

- 根据(1)(2)(3)可知，
- 逻辑上:一个C语言程序是由函数构成的，C语言程序从主函数开始执行，在主函数中调用其他函数，这些函数可能又调用别的函数，主函数执行完毕代表整个程序结束。主函数只能调用不能被调用。
- 物理上:一个程序由一个或者若干个文件(源文件)构成，函数分别放置在这些文件中。

§ 8.2 函数定义的一般形式

★无参函数的定义形式

❖类型标识符：

- 用于指定函数带回的值的类型，不写时为int 型。

合法标识符

```
例  无参函数
void printstar( )
{  printf("**********\n");  }
```

```
类型标识符    函数名（）
{   说明部分
    语句
}
```

函数体

★有参函数定义的一般形式

函数返回值类型
隐含为int型

现代风格:

```
类型标识符   函数名（形式参数表列）
{ 说明部分
  语句
}
```

函数体

例 有参函数

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

例 有参函数

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

★空函数

❖为扩充功能预留，在主调函数中先占一个位置。

```
类型标识符　函数名（）
{　}
```

```
例　空函数
 dummy( )
{ }
```

函数体为空

§8.3 函数参数和函数返回值

★形式参数和实际参数

- ❖在调用函数时，大多数情况下，主调函数和被调用函数之间有数据传递关系。这就是前面提到的有参函数。
- ❖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

形式参数　　实际参数

```
c=max(a,b);        （main 函数）
```

```
max(int a, int  b)（max 函数）
{   int c;
    c=a>b?a:b;
    return (c);
}
```

运行：7,8,9↵
Max is 8
Max is 9

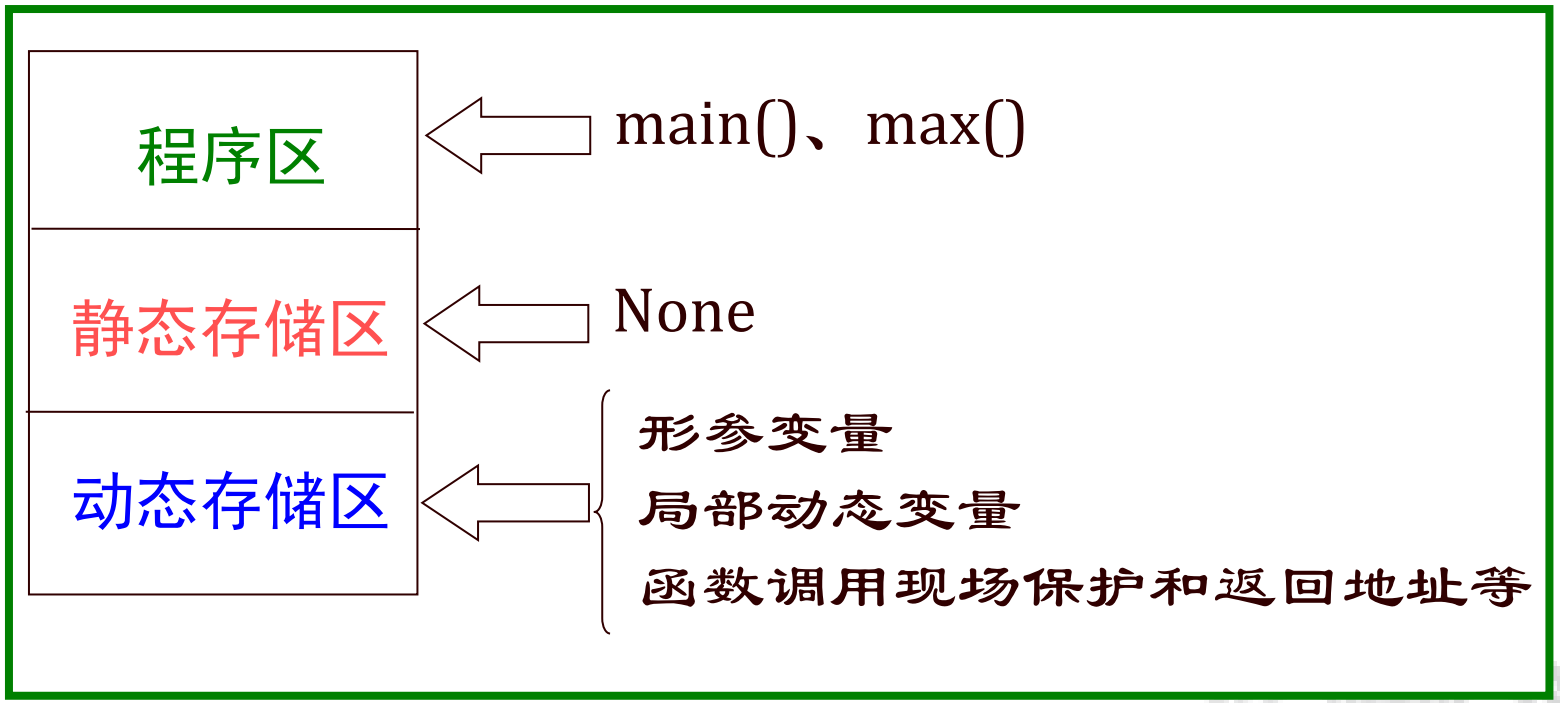
例2 比较两个数并输出大者

```
#include <stdio.h>
int max(int  a, int  b)
{ int c;
  c=a>b?a:b;
  return (c);
}

int main()
{ int max(int a,int b);
  int a,b,c,d;
  scanf("%d,%d,%d",&a,&b,&c);
  d=max(a,b);
  printf("Max is %d",d);

  d=max(b,c);
  printf("Max is %d",d);
}
```

形参　实参　实参



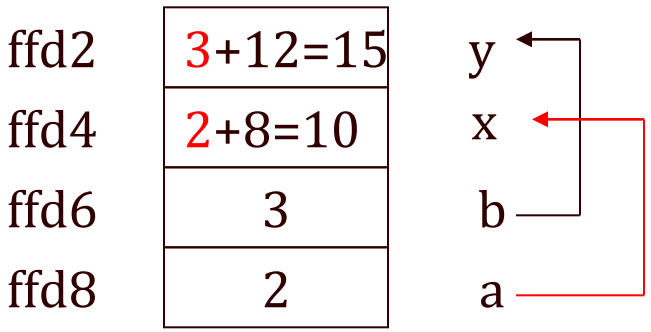
函数执行时内存的变化

几点说明：

- 实参可以是常量、变量或表达式。必须有确定的值。当函数调用时，将实参的值传递给形参，若是数组名，则传送的是数组首地址。
- 形参必须指定类型，只能是简单变量或数组，不能是常量或表达式
- 形参与实参类型一致，个数相同顺序相同。
- 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
- 形参在函数被调用前不占内存;函数调用时为形参分配内存；调用结束，内存释放
- 实参对形参的数据传送是值传送，也是单向传送，当被调函数的形参发生变化时，并不改变主调函数实参的值。形、实参占据的是不同的存储单元

例3：形、实参占据的是不同的存储单元

```
#include <stdio.h>
int add(int x,int y)
{x=x+8; y=y+12;
 printf("x=%d,y=%d\n",x,y);
 printf("&x=%x,&y=%x\n",&x,&y);
}
int main( )
{int  a=2,b=3;
 printf ("a=%d, b=%d\n",a, b);
 printf("&a=%x,&b=%x\n" ,&a,&b);
 add(a,b);
 printf("a=%d,b=%d\n", a,b);
 printf("&a=%x,&b=%x\n", &a,&b);
 return 0;
}
```



运行结果：

a=2,b=3

&a=ffd6,&b=ffd8

x=10,y=15

&x=ffd2,&y=ffd4

a=2, b=3

&a=ffd6,&b=ffd8

★函数的返回值

❖返回语句形式：

return(表达式)；或 return 表达式;

❖功能：1. 使程序控制从被调用函数返回到调用函数中(即终止被调用函数的)；

2. 同时把返值带给调用函数。

- 说明：
  - 函数的返回值，必须用 return 语句带回。
  - return 语句只能把一个返值传递给调用函数。
  - 函数中可有多个return语句，执行哪一个由程序执行情况来定。

```
if(a>b)   return a;
else      return b;
```

  - return 后的值可以是一个表达式，如：

```
return x > y ? x : y;
```

  - 返回值的类型为定义的函数类型 。

如：
```
int   max(int x, int y)
float  min(float a,float b)
double  abc(float d1,float d2)
```

- 若 return 语句中表达式类型与函数类型不一致，则转换为函数类型。
- 若无return语句，遇函数体结束时，自动返回调用函数。可能返回一个不确定或无用的值 。
- 无返回值的函数，定义为 void类型。

例4 无return语句，函数带回不确定值

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

编译错误！

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

输出：10

例5　无返回值函数

```
#include <stdio.h>
int main()
{  void swap(int x,int y );
   int a,b;
   scanf("%d,%d",&a,&b);
   printf("%d %d\n", a,b);
   swap (a,b);
   printf("%d %d\n", a,b);
}
void swap(int x,int y )
 {   int temp;
     temp=x;
     x=y;
    y=temp;
 }
```

输入：
　　1，2
输出：
　　1　2
　　1　2

例6 函数返回值类型转换

输入：1.5，2.5
输出：Max is 2

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

§ 8.4 函数的调用

主调函数：主动去调用其它函数

被调函数：被其它函数所调用

★函数调用的一般形式

函数名（实参表列）

❖说明：

- 实参表列：有确定值的数据或表达式
- 实参与形参个数相等，类型一致，按顺序一一对应，当有多个实参时，实参间用“，”分隔
- 实参表求值顺序，因系统而定（Turbo C 自右向左）
- 调用无参函数时，实参表列为空，但（ ）不能省，例 printstar();

例7 参数求值顺序

```
#include <stdio.h>
int f(int a, int b)
{   int c;
    if(a>b)  c=1;
    else if(a==b)   c=0;
    else c=-1;
    return (c);
}
int main()
{
    int i=2,p;
    p=f(i,++i);
    printf("%d\n",p);
    printf("%d\n",i);
}
```

按自右向左求值
函数调用等于f(3,3)
运行结果：0

按自左向右求值
函数调用等于f(2,3)
运行结果：- 1

需自右向左求值时，
改为：j=++i;
p=f (j , j) ;

需自左向右求值时，
改为：j = i ;
k = ++ i ;
p = f (j , k) ;

printf("%d,%d",i,i++); /*同样存在此情况*/

★函数调用的方式

按函数在程序中出现的位置，有三种调用方式：

- ❖函数语句：以独立的语句去调用函数。不要求有返回值，仅完成一定的操作。

例 printstar();

printf("Hello,World!\n");

- ❖函数表达式：

函数返回一个确定值，以参加表达式的运算。

例 m=max(a,b)*2;

- ❖函数参数：函数调用作为另一个函数的参数。

例 printf("%d",max(a,b)); /*输出大数*/

m=max(a,max(b,c)); /*三数比大小*/

★对被调用函数的声明和函数原型

- ❖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：如果被调函数定义在主调函数之后，那么在主调函数中对被调函数作声明。

- 函数声明
  - 一般形式：函数类型 函数名(形参类型 [形参名],.....);
    或 函数类型 函数名();
  - 作用：告诉编译系统函数类型、参数个数及类型，以便检验
  - C语言中函数声明称为函数原型。
  - 函数定义与函数声明不同，声明只与函数定义的第一行相同。声明可以不写形参名，只写形参类型。
  - 函数说明位置：程序的数据说明部分（函数内或外）

例8 对被调用的函数作声明

```
#include <stdio.h>
float add(float x,float y );  /*对被调用函数的声明*/
int main()
{ float add(float x,float y );  /*对被调用函数的声明*/
   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)   /*函数首部*/
{  float z;                   /*函数体*/
    z=x+y;
    return (z);
}
```

float add(float,float);

输入：3.6 ， 6.5
输出：sum is
10.100000

❖说明：

- 被调用函数的定义（程序）在主调函数之前，可以不加函数声明。
- 在所有函数定义前，已在函数外部做了函数声明，则在各主调函数中可以不加函数声明。

❖说明：

- 被调用函数的定义（程序）在主调函数之前，可以不加函数声明。
- 在所有函数定义前，已在函数外部做了函数声明，则在各主调函数中可以不加函数声明。

被调函数出现在主调函数之前，不必函数说明

```
#include <stdio.h>
float add(float x, float y)
{   float z;
    z=x+y;
    return(z);
}
main()
{   float a,b,c;
    scanf("%f,%f",&a,&b);
    c=add(a,b);
    printf("sum is %f",c);
}
```

❖说明：

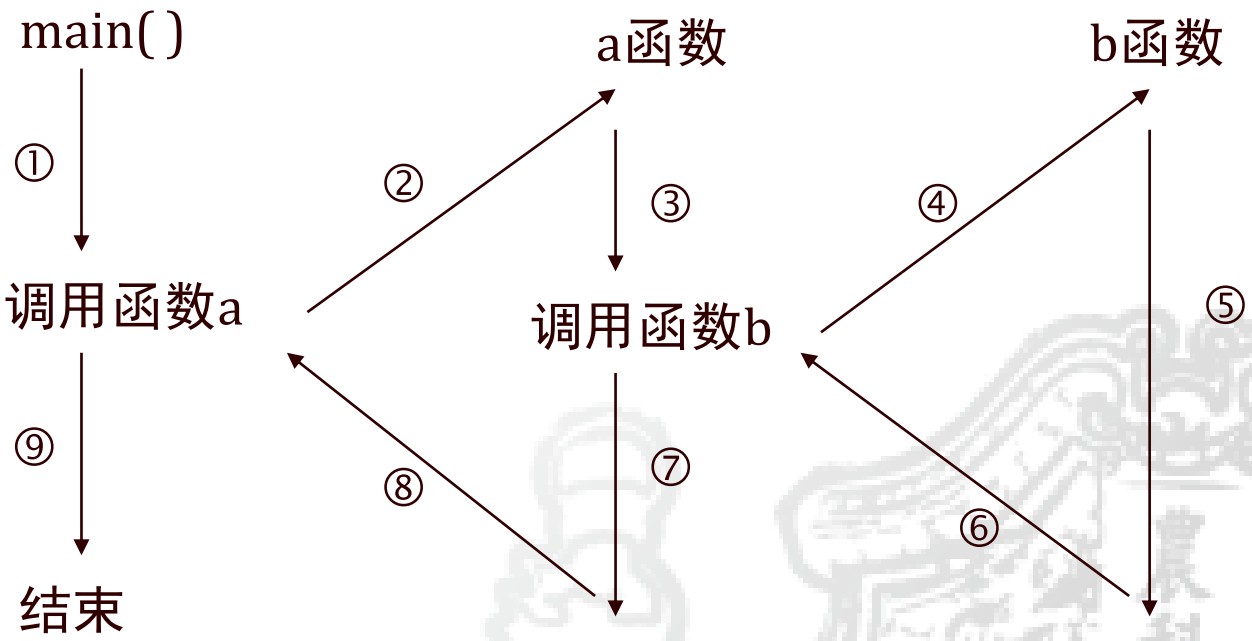
- 被调用函数的定义（程序）在主调函数之前，可以不加函数声明。
- 在所有函数定义前，已在函数外部做了函数声明，则在各主调函数中可以不加函数声明。

在函数外面做函数说明

```
char letter(char,char);
float f(float,float);
int I(float,float);
int main()
{……}
char letter(char c1,char c2)
{……}
float f(float x,float y)
{……}
int I(float j,float k)
{……}
```

§8.5 函数的嵌套调用

C中的函数：
- 不允许嵌套定义，函数间的关系是平行的、独立的。
- 允许嵌套调用，即在调用某函数过程中又调用另一函数。



例9　输入两个整数，求平方和

```
#include <stdio.h>
int fun1(int x,int y);
int main(void)
{ int a,b;
  scanf("%d%d",&a,&b);
  printf("The result is：%d\n",fun1(a,b) );
  return 0;
}

int fun1(int x,int y)
{ int fun2(int m);
  return ( fun2(x)+fun2(y) );
}

int fun2(int m)
{ return (m*m);
}
```

输入：　3　4
输出：　The result is: 25

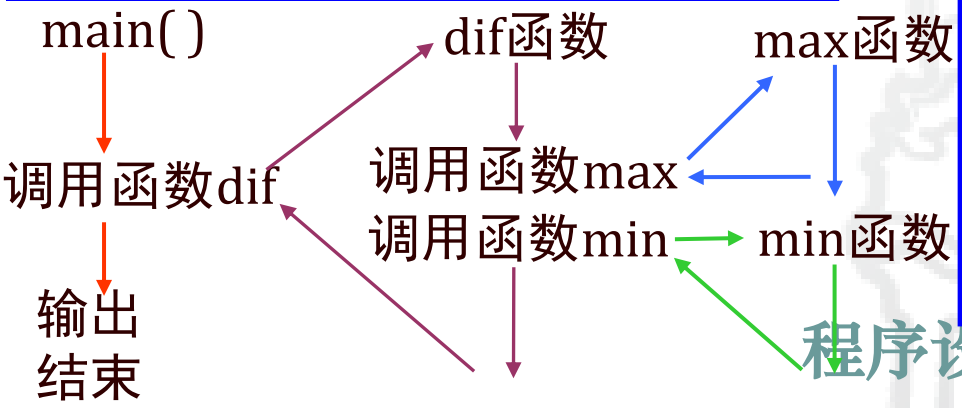
例10 求三个数中最大数和最小数的差值

```
#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);
}
```

```
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);
}
```

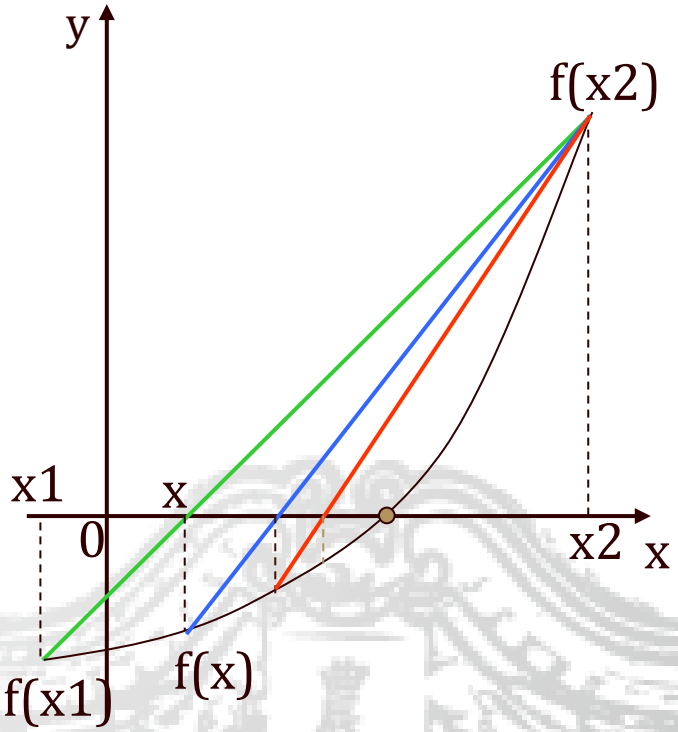


例11　用弦截法求方程　$x^3-5x^2+16x-80=0$　的根

1. 取x1,x2两点，求得f(x1), f(x2) 。
   异号：x1,x2之间必有一根。
   同号：改变x1, x2，直到f(x1), f(x2)异号为止。

2. 连f(x1),f(x2)两点（弦）交x轴于x。
   X点的坐标求法：
   ①求X点的x坐标　$x=\dfrac{x_1\cdot f(x_2)-x_2\cdot f(x_1)}{f(x_2)-f(x_1)}$
   ②从x值得f(x)

3. 若f(x)与f(x1)同号，
   则根必在(x,x2)区间，此时将x1=x;
   若f(x)与f(x2)同号，
   则根必在(x1,x)区间，此时将x2=x;



4. 重复步骤2和3，直到| f(x) | < ε为止，设ε < 10^{-6}，则 f(x)≈0

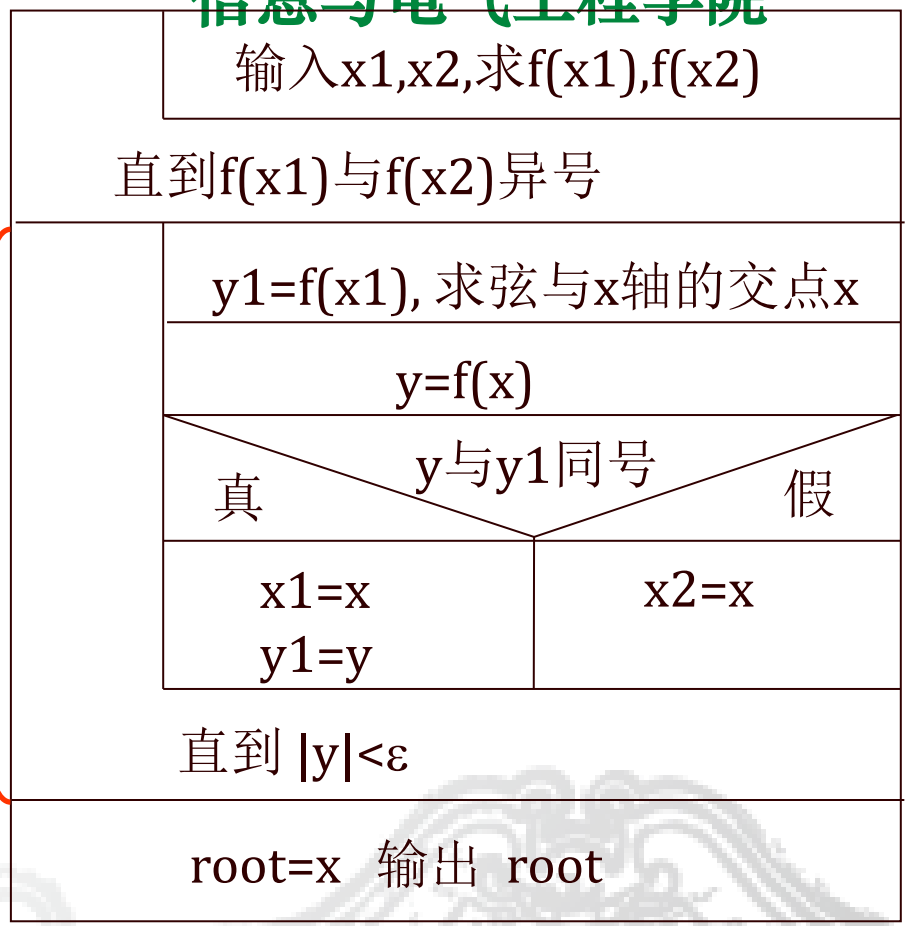
用三个函数实现各部分的功能:

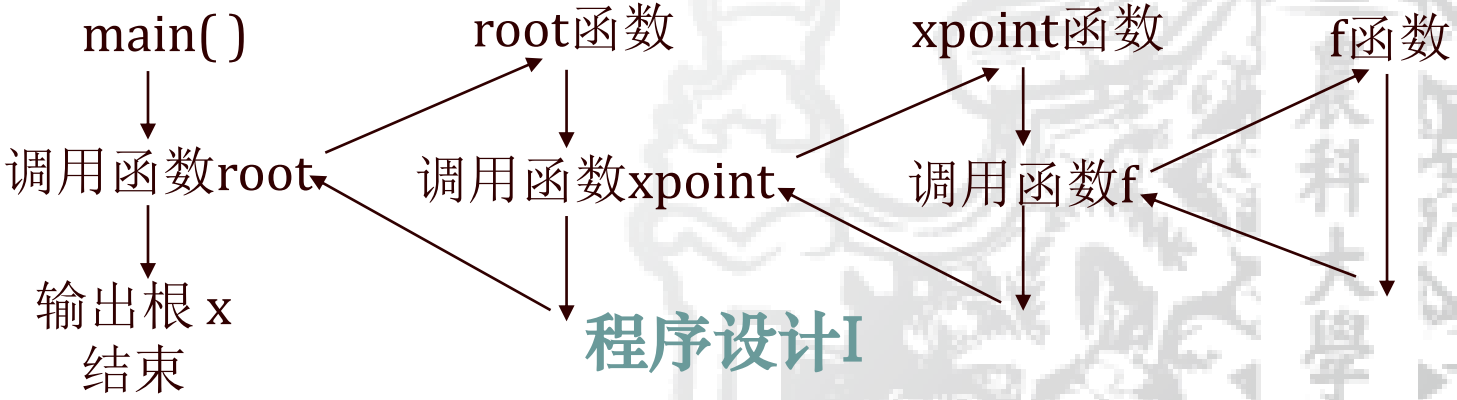
① 函数f(x)：
求x的函数:$x^3 - 5x^2 + 16x - 80$

② 函数xpoint(x1,x2)：
求弦与x轴交点X的x坐标

③ 函数root(x1, x2)：
求(x1, x2)区间的实根

root函数





```
#include <stdio.h>
#include  <math.h>
float  f(float x)
{ float  y;
  y=((x-5.0)*x+16.0)*x-80.0;
  return(y);
}
float  xpoint(float x1, float x2)
{ float  x;
  x =(x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
  return(x);
}
```

```
float  root(float x1,float x2)
{ float  x, y, y1;
  y1=f(x1);
  do {
       x=xpoint(x1, x2);
       y=f(x);
       if(y*y1 > 0)
           { y1=y;x1=x;}
       else  x2=x;
   } while(fabs(y) >=0.000001);
  return(x);
}
```

```
int main( )
{  float  x1, x2, f1, f2, x;
   do {   printf("input  x1, x2:\n");
        scanf("%f%f",&x1,&x2);
        f1=f(x1); f2=f(x2);
     } while(f1*f2 >=0);
   x=root(x1, x2);  printf("A  root  of  equation  is  %8.4f\n",x);
}
```

运行情况：

Input x1,x2:

2,6↵

A root of equation is 5.0000

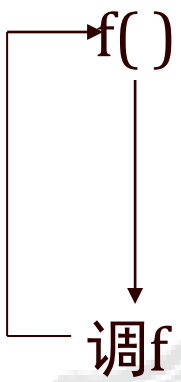
§ 8.6 函数的递归调用

递归：在函数调用过程中，直接或间接的调用自身。

★递归调用方式

❖直接递归调用：在函数体内又调用自身

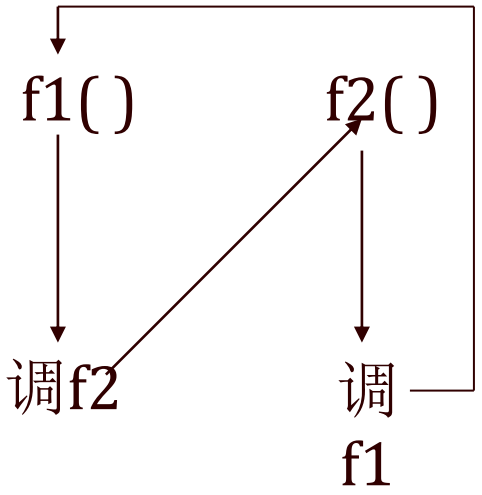
```
int  f(int x)
{    int y,z;
      ......
     z=f(y);
     .......
    return(2*z);
}
```



❖间接递归调用：当函数1去调用另一函数2时,而另一函数2反过来又调用函数1自身。

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

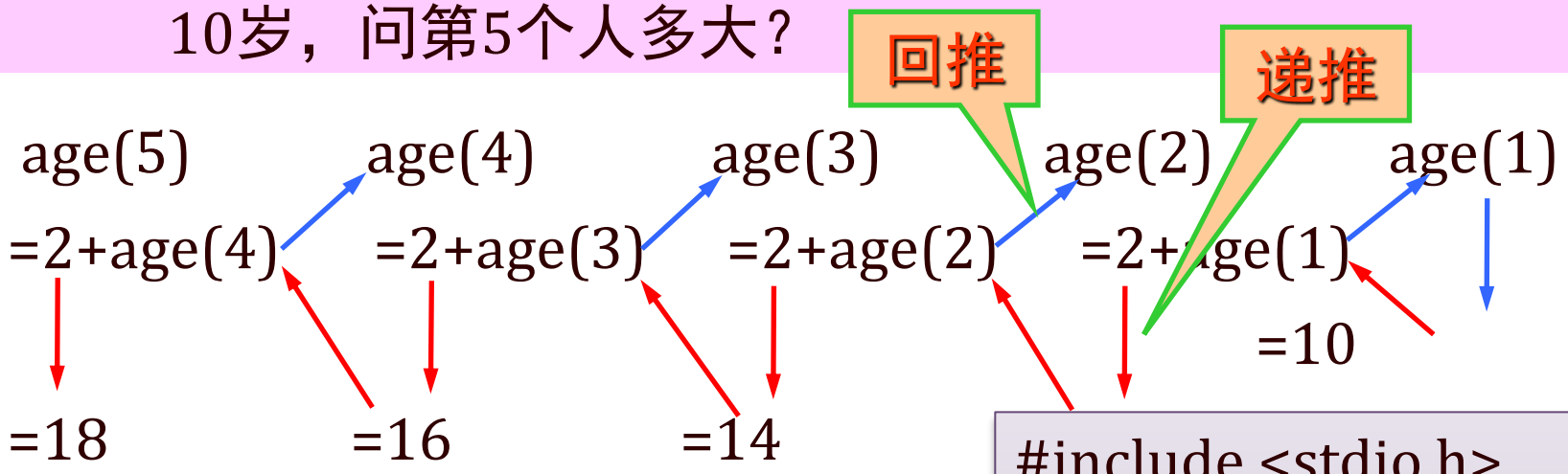
```
int f2(int t)
{   int a,c;
     ……
    c=f1(a);
     …….
    return(3+c);
}
```



❖解决无终止递归调用的方法是：确定好结束递归的条件。

用if语句控制 { 条件成立，进行递归
条件不成立，结束递归

例12 有5个人，第5个人比第4个人大2岁，第4个人比第3个人大2岁，……第2个人比第1个人大2岁，第1个人10岁，问第5个人多大？



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

运行结果：18

```
#include <stdio.h>
int age(int n)
{int  c;
if(n==1)   c=10;
else  c=2+age(n-1);
return(c); }
int main( )
{printf("%d\ n", age(5));}
```

有些问题，可以用递推，也可以用递归的方法解决。

- 递推:从一个已知的事实出发,按一定规律推出下一个事实,再从已知的新的事实,推出下一个新的事实.

例13　用递推法求n!　,即从1开始, 乘2, 乘3....一直到n

```
#include <stdio.h>
int main( )
{ int  i, s=1;
  for(i=1;i<=5;i++)
     s=s* i;
  printf("s=%d\ n",s);
  return 0;
}
```

运行结果：s=120

递归:在函数调用自身时,要给出结束递归的条件。

- 先回推再递推
- 如：n！，

5!=5 × 4!

4!=4 × 3!

3!=3 × 2!

2!=2 × 1!

1!=1

0!=1

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

运行:
input a integer number： 10
10! = 3628800

例14 用递归方法求n!

```
#include <stdio.h>
int main()
{ float fac(int n);
  int n;  float y;
  printf("Input a integer number:");
  scanf("%d",&n);
  y=fac(n);
  printf("%d! =%f",n,y);
  return 0;  }
float fac(int n)
{ float f;
  if(n<0)  printf("n<0,data error!");
  else if(n==0||n==1)  f=1;
  else  f=fac(n-1)*n;
  return(f);}
```

- 其他例子：
- 斐波那契数列
- 1 1 2 3 5 8 13 ...
- 等差数列求和
- 等比数列求和

例15 Hanoi（汉诺）塔问题

十九世纪末，欧洲珍奇商店出现一种汉诺塔游戏，推销材料介绍说：古代印度布拉玛庙里的僧侣们正在玩这种游戏，如果游戏结束，世界末日即来临。是一个只能用递归方法解决的问题。

规则及分析：

n个盘子从一根针移到另一根针，每次只能移动一个盘子，不允许大盘在小盘上面。

共有三根针，n个盘子由A移到C，需移动的次数是2^n-1, 若64个盘子移动的次数为：

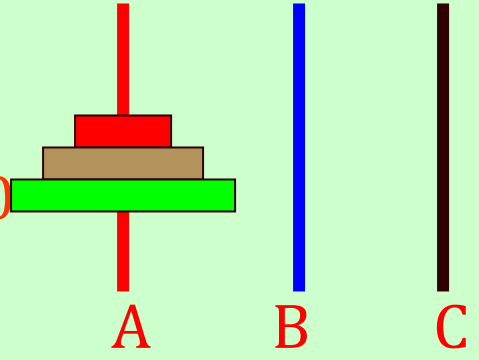
$2^{64}-1=18,446,744,073,709,551,600$

一年的秒数是：365 x 24 x 60 x 60=31536000

18446744073709511600÷31536000

=584942173 55年

即:5849亿年, 从能源角度推算, 太阳系寿命只有150亿年



❖简化实例：将A上3个盘子移到C

步骤：1. A上两个盘子借助C移到B

2. A上最后一个盘子移到C （可直接完成）

3. B上两个盘子借助A移到C

第一步进一步分解：

1.1 A上一个盘子从A→C

1.2 A上一个盘子从A→B

1.3 C上一个盘子从C→B

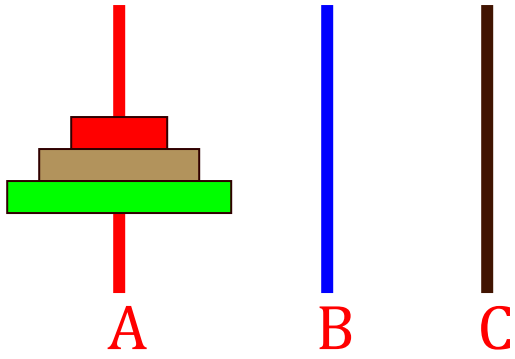
第三步进一步分解：

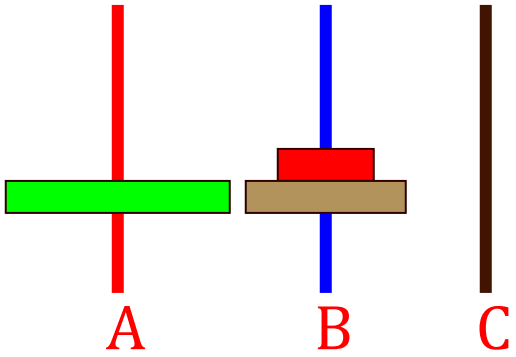
3.1 B上一个盘子从B→A

3.2 B上一个盘子从B→C

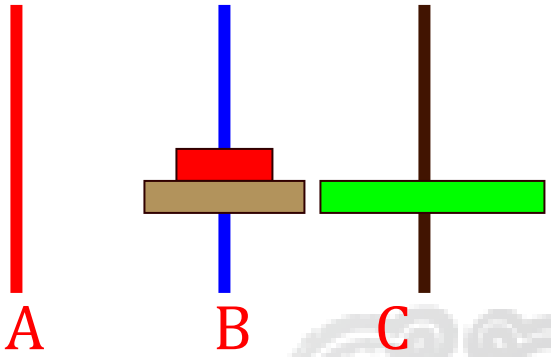
3.3 A上一个盘子从A→C

共移动7步：2^3-1次

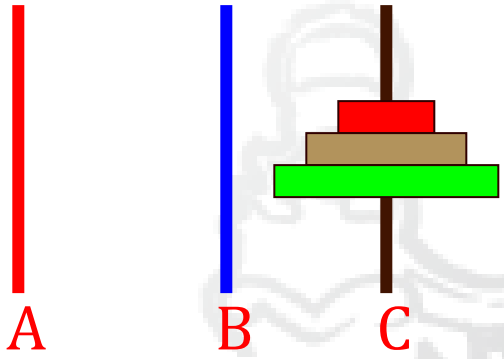




(1)



(2)



(3)

❖方法与步骤

- 将A上n-1个盘子借助C移到B 。
- 把A上剩下一个盘子送到C
- 将n-1个盘子从B借助A移到C

n个盘要移动2^n-1次

❖结论：上面三个步骤包含两类操作

- 步骤1和3都是将n-1个盘子从一个针移到另一个针上(n>1时)，这是一个递归的过程；方法一样，只是针的名称不同而已，为使问题一般化，将步骤1和3表示为：将one 针上的n-1个盘子移到two针，借助 three针，只是对应关系不同。

  第一步对应关系：one → A　　two → B　　three → C

  第三步对应关系：one → B　　two → C　　three → A

- 将1个盘子从一个针上移到另一针上。

❖因此，可以用两个函数分别实现上面两类操作，用hanoi函数实现第一类操作，用move函数实现第2类操作。

- hanoi(n,one,two,three)　将n个盘从 one → three借助two
- move(x,y)　将1个盘从x → y座，x、y根据情况取代ABC座中的1个。

例16 用递归方法解决Hanoi（汉诺）塔问题的程序


```
#include <stdio.h>
int main()
{ void hanoi(int n,char one,char two,char three);
  int m;
  printf("Input the number of diskes:");
  scanf("%d",&m);
  printf("The step to moving %3d diskes:\n",m);
  hanoi(m,'A','B','C');    return 0;}

void hanoi(int n,char one,char two,char three)
{void move(char  x, char  y);
 if(n==1)  move(one,three);
 else {  hanoi(n-1,one,three,two);
         move(one,three);
         hanoi(n-1,two,one,three);
       }}

void move(char  x, char  y)
{ printf("%c--->%c\n",x, y); }
```

运行：

input number of diskes: 3↵

the step to moving 3 diskes:

A →C

A →B

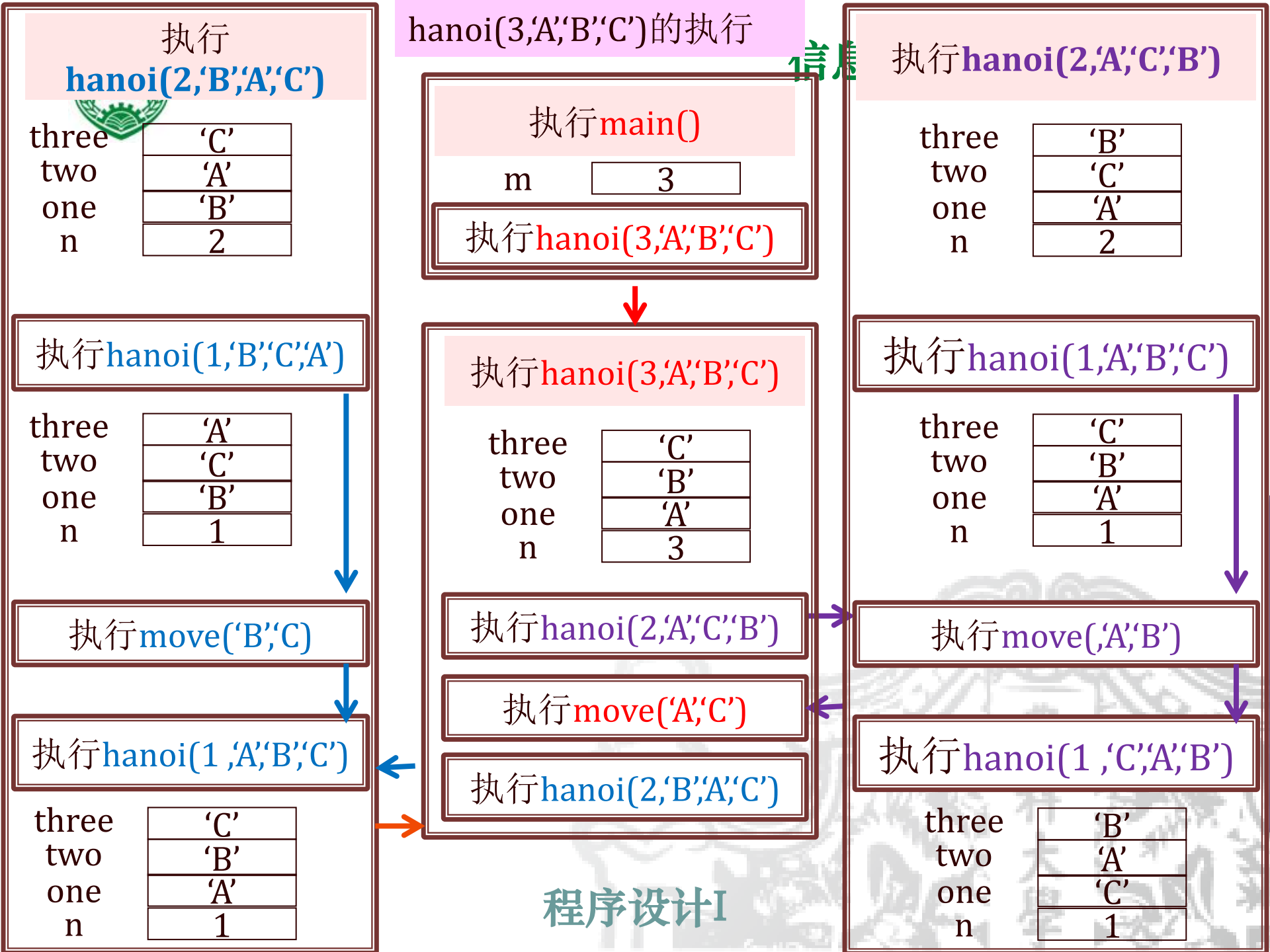
C →B

A →C

B →A

B →C

A →C


hanoi(3,'A','B','C')的执行
执行main()
m 3
执行hanoi(3,'A','B','C')
执行hanoi(3,'A','B','C')
three 'C'
two 'B'
one 'A'
n 3
执行hanoi(2,'A','C','B')
执行move('A','C')
执行hanoi(2,'B','A','C')
执行hanoi(2,'A','C','B')
three 'B'
two 'C'
one 'A'
n 2
执行hanoi(1,'A','B','C')
执行move('A','B')
执行hanoi(1,'B','C','A')
three 'A'
two 'C'
one 'B'
n 1
执行hanoi(1,'C','A','B')
three 'C'
two 'B'
one 'A'
n 1
执行hanoi(1,'A','C','B')
执行move('B','C')
执行hanoi(1,'A','B','C')
three 'C'
two 'B'
one 'A'
n 1
three 'B'
two 'A'
one 'C'
n 1
程序设计I

汉诺塔动图示例

视频可参考网上各类资源。例如下方链接
https://www.bilibili.com/video/BV16K411A7fo/?spm_id_from=autoNext

§8.7 数组作为函数参数

★数组元素作函数实参——值传递

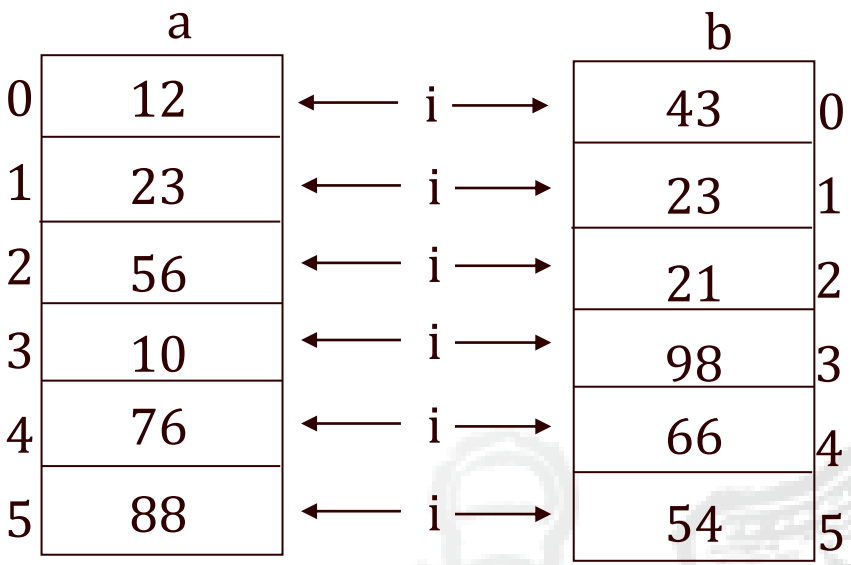
例17 两个数组比较大小

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分别记录a[i]>b[i], a[i]==b[i], a[i]<b[i]的次数。
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

§ 8.7 数组作为函数参数

★数组元素作函数实参——值传递

例18 两个数组比较大小



n=0	n=0	n=0	n=1	n=1	n=2	n=3
m=0	m=0	m=1	m=1	m=1	m=1	m=1
k=0	k=1	k=1	k=1	k=2	k=2	k=2

```
#include <stdio.h>
int main()
{ int large(int x,int y);
  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("\n");
  printf("Enter array b:\n");
  for(i=0;i<10;i++) scanf("%d",&b[i]);
  printf("\n");

  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; }

  printf("a[i]>b[i]%d times\n",n);
  printf("a[i]=b[i]%d times\n",m);
  printf("a[i]<b[i]%d times\n",k);
  if(n>k) printf("array a is larger than array b\n");
  else if(n<k) printf("array a is smaller than array b\n");
       else printf("array a is equal to array b\n");}
```

运行：

```
enter array a:
1 3 5 7 9 8 6 4 2 0↵
enter array b:
5   3 8 9 -1 -3 5 6 0 4↵
a[i] > b[i]   3 time
a[i] = b[i]   1 time
a[i] < b[i]   1 time
array a is large then array b
```

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

数组名可作函数参数

实参和形参都应用数组名

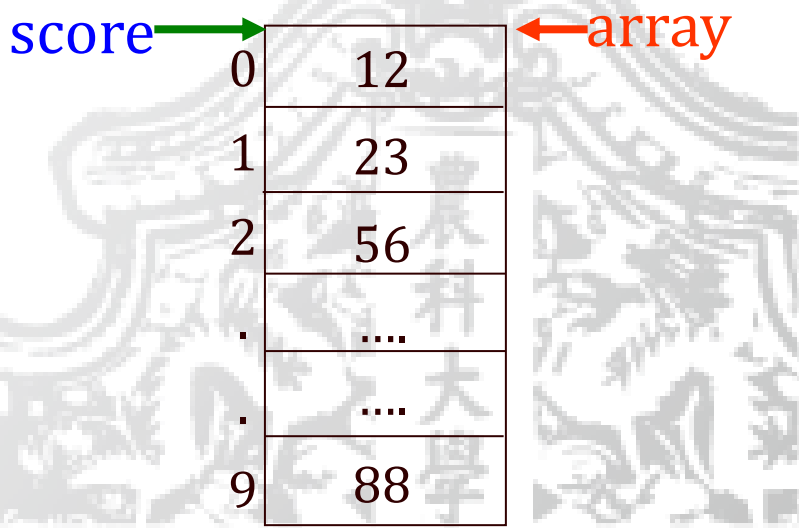
形参用数组定义⇔float array[ ]

例19 求学生的平均成绩

```
#include <stdio.h>
int main()
{ float average(float array[10]);
  float score[10], aver;
  int i;
  printf("Input 10 scores: \n");
  for( i=0; i<10; i++ )
     scanf("%f", &score[i]);
  printf("\n");
   aver=average(score);
   printf("Average is: %5.2f", aver);
}
```

实参用数组名

```
float  average(float array[10])
{ int i;
  float aver,sum=array[0];
  for( i=0; i<10; i++ )
     sum=sum+array[i];
  aver=sum/10;
  return (aver);
}
```



几点说明：

- 地址传递
  - 调用函数时，对形参数组元素的操作，实际上也是对实参数组元素的操作。
- 在主调函数与被调函数分别定义数组,且类型应一致
  - 如：array是形参数组名，score是实参数组名。

- 形参数组大小(多维数组第一维)可不指定
  - 在定义数组时在数组名后面跟一个空的方括弧
  - C编译对形参数组大小不检查，即使定义了也不起作用。
- 形参数组名是地址变量
  - 调用时，只是将实参数组的首地址传给形参数组，
  - 因此score[n]和array[n]指的是同一单元

例20 求两组学生的平均成绩，形参数组长度缺省

```
#include <stdio.h>
int main()
{ float   average(float array[ ],int n);
  float score_1[5]={98.5,97,91.5,60,55};
  float score_2[10]={67.5,89.5,99,69.5,77,89.5,76.5,54,60,99.5};
  printf("The average of clase A is %6.2f\n",average(score_1,5));
  printf("The average of clase B is %6.2f\n",average(score_2,10));
}
float   average(float array[ ],int n)
{ int i;
   float aver,sum=array[0];
   for( i=1; i<n; i++ )
      sum=sum+array[i];
   aver=sum/n;
   return (aver);
}
```

另设一个参数，传递需要处理的数组元素个数

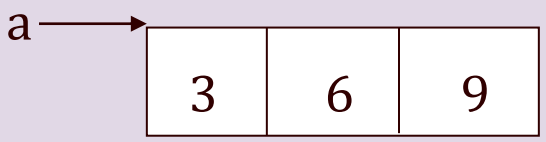
运行：

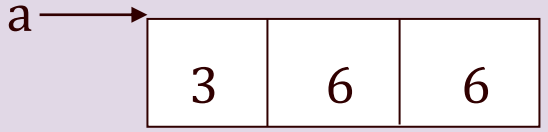
```
The average of class A is 80.40
The average of class B is 78.20
```

- 数组名作函数参数时，实参和形参两个数组共占同一段内存单元，形参数组的元素值改变会使实参数组元素的值**同时变化**。

例21 一维数组

```
void change_value(int  x[])
{
  x[2]=x[1];
}
int main()
{ int a[3]={3,6,9};
  int i;
   change_value (a);
   for(i=0;i<3;i++)
         printf("%d\n",a[i]);
   return 0;
}
```





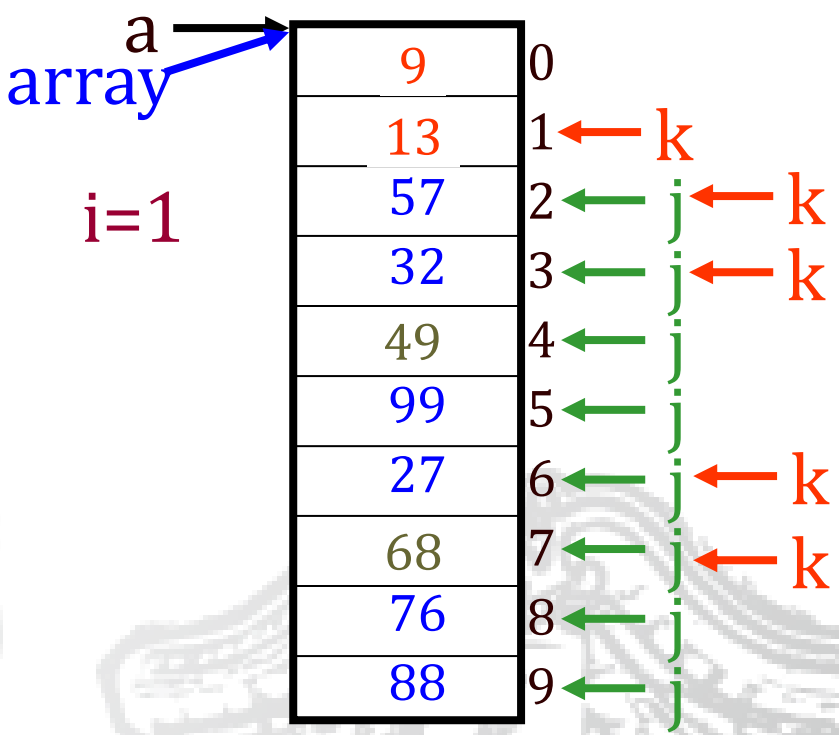
注意：不能return x;也不需要return x。

例22 用选择法对数组中的10个整数按由小到大排序

```
#include <stdio.h>
int main()
{  int a[10],i;
   printf("enter the array\n");
   for(i=0;i<10;i++)
       scanf("%d",&a[i]);
    sort(a,10);
    printf("the sorted array: \n");
    for(i=0;i<10;i++)
         printf("%d ",a[i]);
     printf("\n");
     return 0; }
```

选择法排序思路

```
int 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;
     t=array[i]; array[i]=array[k];
     array[k]=t;
   }
}
```



数值交换(整型、整型数组)


```
void change_number(int a, int b)
{
        int temp;
        temp = a;
        a = b;
        b = temp;
        printf("In function:%d,%d\n",a,b);
}
int main()
{
        int a=1,b=9,i;
        change_number(a,b);
        printf("Out of function:%d,%d\n",a,b);
        int c[10]={2,4,6,8,10,12,14,16,18,20};
        int d[10]={1,3,5,7,9,11,13,15,17,19};
        change_10_numbers(c,d);
        printf("Out of function:\n");
        for(i=0;i<10;i++)
        {printf("%d,%d\n",c[i],d[i]);}
}
```

```
void change_10_numbers(int a[], int b[])
{
        int i;
        for(i=0; i<10; i++)
        {
                int temp;
                temp = a[i];
                a[i] = b[i];
                b[i] = temp;
        }
        printf("In function:\n");
        for(i=0;i<10;i++)
        {printf("%d,%d\n",a[i],b[i]);}
}
```

★用多维数组名作函数参数

❖可以用多维数组名作实参和形参

❖形参数组定义时，只能省略第一维的大小说明。

●C编译不检查第一维的大小，而且数组名作函数参数是地址传送，所以形参数组第一维大小任意，可以和实参数组的维数不同。

实参数组定义：int score[5][10]

形参数组定义：int array[3][10] 或 int array[8][10]

●合法的定义：int array[3][10]; 或 int array[ ][10]

●错误的定义：int array[ ][ ]; int array[3][ ];

例23 求3×4矩阵中各元素的最大值

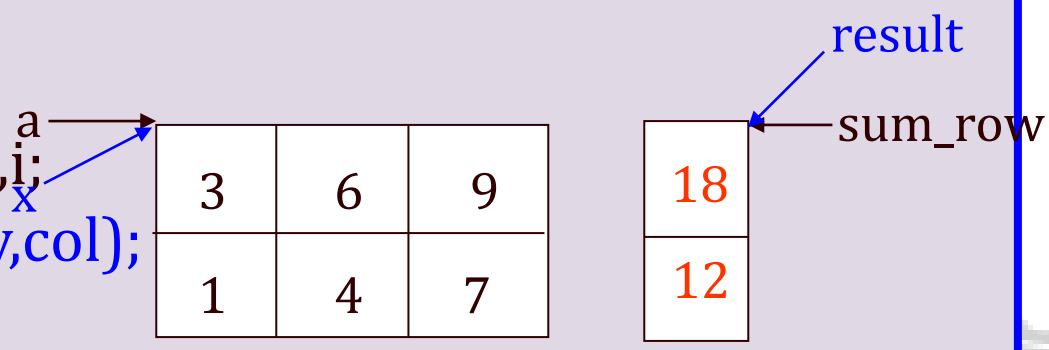
```
#include <stdio.h>
int main()
{int  max_value(int  array[ ][4]);
 int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
 printf("max value is %d\n",max_value(a));
}
int  max_value(int  array[3][4])
{ int i,j,k,max;
  max=array[0][0];
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      if(array[i][j]>max) max=array[i][j];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

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24 求二维数组中各行元素之和

```
void 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];}
}
int 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]);
  return 0;
}
```



注意：如果一个函数运行的结果只能保存在数组中，加一个数组形参（例如，result[]）

§8.8 局部变量和全局变量

变量按其作用域，可分为局部变量和全局变量。

★局部变量——内部变量

❖定义：在函数内定义，只在本函数内有效

❖说明：

- main中定义的变量只在main中有效
- 不同函数中同名变量，占不同内存单元
- 形参属于局部变量
- 可定义在复合语句中有效的变量
- 局部变量可用存储类型：auto　register　static

  （默认为auto）

```
float  f1(int a)
{  int b,c;
   .......
}

char f2(int x,int y)
{   int i,j;
   ......
}

main()
{  int m,n;
   .......
}
```

a,b,c有效

x,y,i,j有效

m,n有效

例 不同函数中同名变量

```
int main()
{ int a,b;
  a=3;
  b=4;
  printf("main:a=%d,b=%d\n",a,b);
  sub();
  printf("main:a=%d,b=%d\n",a,b);
}
sub()
{ int a,b;
  a=6;
  b=7;
  printf("sub:a=%d,b=%d\n",a,b);
}
```

运行结果：
main:a=3,b=4
sub:a=6,b=7
main:a=3,b=4

```
main()
{  int a, b;
     ┆
   { int c;
     c=a+b;
     ┆
   }
   ┆
}
```

c 范围

a,b 范围

例25 复合语句中变量

```
#define  N  5
int main()
{ int i;
  int a[N]={1,2,3,4,5};
  for(i=0;i<N/2;i++)
   { int temp;
     temp=a[i];
     a[i]=a[N-i-1];
     a[N-i-1]=temp;
   }
  for(i=0;i<N;i++)
    printf("%d  ",a[i]);
}
```

运行结果：5 4 3 2 1

★全局变量——外部变量

- ❖定义：在函数外定义，可为本文件所有函数共用，也叫外部变量。
- ❖有效范围：从定义变量的位置开始到本源文件结束，及有extern说明的其它源文件
- ❖几点说明：
  - ●全局变量的使用,增加了函数间数据联系的渠道,同一文件中的所有函数都能引用全局变量的值,当某函数改变了全局变量的值时,便会影响其它的函数。

- 习惯上，全局变量名的第一个字母用大写。
- 使用全局变量可以减少函数的实参和形参个数。
- 不必要时不要使用全局变量
  - 全局变量在程序执行的全过程都占用存储单元。
  - 不利于程序的移植。程序的可读性变差。
- 全局与局部变量重名时，在函数内部将屏蔽全局变量。

```
int  p=1,q=5;
float  f1(a)
int a;
{  int b,c;
    .......
}
int   f3()
{.....
}
char c1,c2;
char f2(int x,int y)
{   int i,j;
    ......
}
main()
{  int m,n;
   .......
}
```

p,q的作用范围

c1,c2的作用范围

外部变量

局部变量和全局变量同名，局部变量作用域中外部变量被屏蔽

例26 全局变量的作用域及其使用情况

```
int  a=1;
f1()  {int  b; b=a+3; printf("f1:a=%d, b=%d\n",a, b); }
f2()  {int  a, b; a=5; b=a+3; printf("f2: a=%d, b=%d\n",a, b);
f3()  {int  b; a=6; b=a+3; printf("f3:a=%d, b=%d\n",a, b); }
int main()
{ int  b=3;
  printf("1.main : a=%d, b=%d\n",a, b); f1();
  printf("2.main : a=%d, b=%d\n",a, b); f2();
  printf("3.main : a=%d, b=%d\n",a, b); f3();
  printf("4.main : a=%d, b=%d\n",a, b);
}
```

局部变量

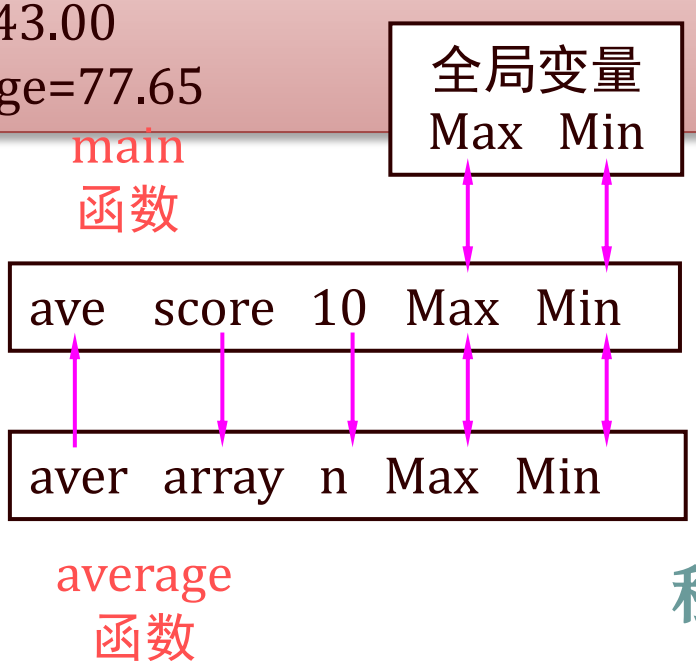
运行：

```
1.main:a=1, b=3
f1:a=1, b=4
2.main:a=1, b=3
f2:a=5, b=8
3.main:a=1, b=3
f3:a=6, b=9
4.main:a=6, b=3
```

全局变量增加了函数间传送数据的联系

例27
一维数组内存放了10个学生成绩，求平均分、最高分和最低分。

运行：input 10 numbers:
99 45 78 97 100 67.5 89 92 66 43↵
max=100.00
min=43.00
average=77.65




```
#include <stdio.h>
float  Max=0,Min=0;
int main()
{ float average(float array[ ],int n);
  int i; float ave,score[10];
  for(i=0;i<10;i++) scanf("%f",&score[i]);
  ave=average(score,10);
  printf("max=%6.2f\nmin=%6.2f\n
          average=%6.2f\n",Max,Min,ave);
}
float  average(float  array[], int  n)
{ int i;
  float  aver, sum=array[0];
  Max=Min=array[0];
  for(i=1;i<n;i++)
    { if(array[i]>Max) Max=array[i];
      else  if(array[i]<Min) Min=array[i];
      sum=sum+array[i];
    }
  aver=sum/n;
  return(aver);}
```

程序

原意输出5行*号，使用外部变量i 后只输出一行*号。

例28　外部变量与局部变量同名

```
#include <stdio.h>
int a=3,b=5;
int main()
{ int max(int a, int b);
  int a=8;
  printf("max=%d",max(a,b));
}
int max(int a, int b)
{ int c;
  c=a>b?a:b;
  return(c);
}
```

局部变量a=8将外部变量a=3屏蔽

运行结果：max=8

例29　外部变量副作用

```
int i;
main()
{  int prt();
   for(i=0;i<5;i++)
      prt();
}
int prt()
{   for(i=0;i<5;i++)
       printf("%c",'*');
    printf("\n");
}
```

运行结果：*****

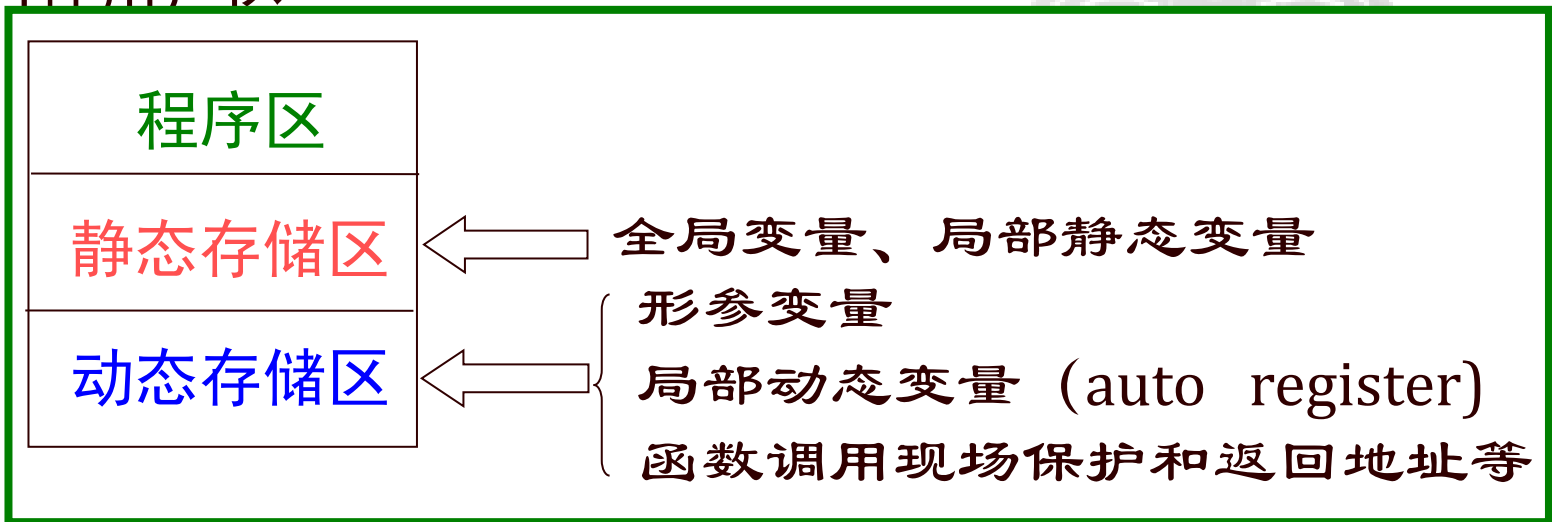
§8.9 变量的存储类别

★动态存储方式与静态存储方式

❖变量分类：

- 按数据类型：整型、实型、字符型
- 按作用域：全局变量、局部变量
- 存储方式：
  - ◆静态存储：程序运行期间分配固定的存储空间。
  - ◆动态存储：程序运行期间根据需要动态分配存储空间。

❖内存用户区



§8.9 变量的存储类别

★动态存储方式与静态存储方式

❖变量分类：

- 按数据类型：整型、实型、字符型
- 按作用域：全局变量、局部变量
- 存储方式：
  - ◆静态存储：程序运行期间分配固定的存储空间。
  - ◆动态存储：程序运行期间根据需要动态分配存储空间。

❖内存用户区

❖生存期：

- 静态变量：从程序开始执行到程序结束
- 动态变量:从包含该变量定义的函数开始执行至函数执行结束

★auto变量

- ❖函数内部无static声明的局部变量均为动态存储类别，被分配在动态区 。
- ❖存储类别为自动时，声明符auto可省；自动变量被 分配在动态区，未赋初值时，其值未定义，每次调用重新赋值。

例如：

```
int f(int a)          /*定义f函数，a为形参*/
{auto  int  b,c=3;  /*定义b、c为自动变量*/
   …
}
```

又如：

```
auto   int  b,c=3;
       int  b,c=3;          /*两者等价*/
```

例30 auto 变量的作用域

```
main()
{ int x=1;
  int  prt();
  { int x=3;
    prt();
    printf("2nd x=%d\n",x);
  }
  printf("1st x=%d\n",x);
}
int  prt()
{ int x=5;
  printf("3th  x=%d\n",x);
}
```

x=1作用域

x=3作用域

x=1作用域

x=5作用域

运行结果：
3th x=5
2nd x=3
1st x=1

★用static声明局部变量

若希望函数调用结束后，局部变量的值保留，则指定该变量为静态局部变量，用static对变量加以声明。

例31 局部静态变量值具有可继承性

```
int main()
{ int  increment();
  increment();
  increment();
  increment();
}
int  increment()
{ int x=0;
  x++;
  printf("%d\n",x);
}
```

运行结果: 1
1
1

```
int main()
{ int  increment();
  increment();
  increment();
  increment();
}
int  increment()
{ static int x=0;
  x++;
  printf("%d\n",x);
}
```

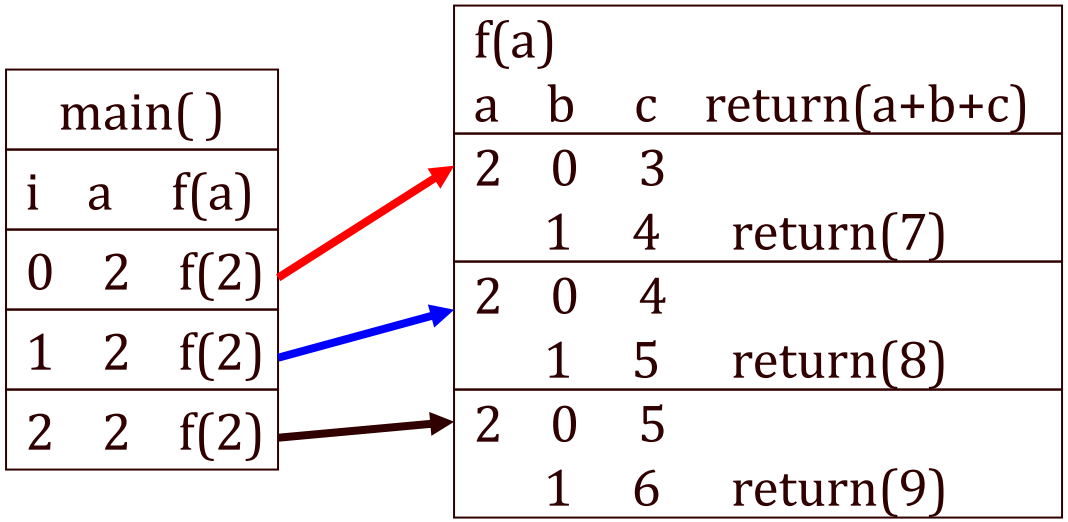
运行结果: 1
2
3

例32 考察静态局部变量的值

```
#include <stdio.h>
int main( )
{ int f(int) ;
  int  a=2, i;
  for(i=0; i<3; i++)
  printf("%d  ", f(a));
}

int f(int a)
{ auto  int b=0;
  static  int c=3;
  b=b+1; c=c+1;
  return(a + b + c);
}
```

运行结果：7 8 9



对静态局部变量的说明：

(1)分配在静态区，程序运行结束释放存储单元。
(2)赋初值一次未赋初值时为0，前次结果保留。
(3)局部动态变量若未赋初值，其值不确定，局部静态变量未赋初值，其值为0或‘\0’(字符变量)。
(4)静态局部变量在函数调用结束后虽存在，但其它函数不能引用它。

使用局部静态变量的场合

(1)需要保留上一次调用结束时的值

例33 打印1到5的阶乘值

```
#include <stdio.h>
int main( )
{int  fac(int n);
 int  i;
 for(i=1; i<=5; i++)
    printf("%d != %d\n", i, fac(i));
}
int  fac(int n)
{ static  int  f=1;
 f=f*n;
 return(f);
}
```

main( )		fac(n)		输出结果
i	fac(i)	n	f=f*n	f=1
1	fac(1)	1	f=1*1	1!=1
2	fac(2)	2	f=1*2=2	2!=2
3	fac(3)	3	f=2*3=6	3!=6
4	fac(4)	4	f=6*4=24	4!=24
5	fac(5)	5	f=24*5=24	5!=120

(2)**初始化后变量只被引用而不改变其值**，则用静态局部变量较方便，以免每次调用时重新赋值，但会一直占用内存浪费系统资源。

register（寄存器型）变量

如果有一些变量使用频繁（例如，在一个函数中执行10000次循环，每次循环中都要引用某局部变量），则为存取变量的值要花费不少时间。为提高执行效率，允许将局部变量的值放在CPU中的寄存器中，需要用时直接从寄存器取出参加运算，不必再到内存中去存取。由于寄存区的存取速度高于对内存的存取速度，因此这样做可以提高效率。

这种变量叫做寄存器变量，用关键字**register**作声明。如

```
register int f;
```

register（寄存器型）变量

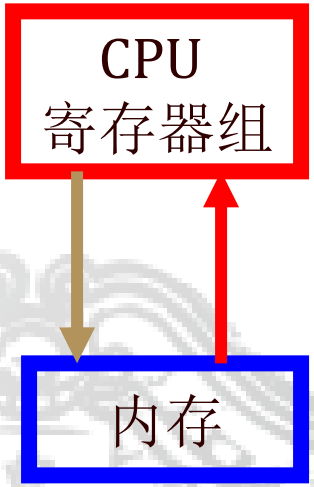
CPU内有寄存器可用来存放数据，若把数据声明为寄存器类型，则将该类型的数据存放在寄存器中，其优点是：减少数据与内存之间的交换频率，提高程序的效率和速度。

例34 使用寄存器变量

```
#include <stdio.h>
int main( )
{ long  fac(long);
  long  i, n;
  scanf("%ld",&n);
  for(i=1; i<=n; i++)
     printf("%ld != %ld\ n", i, fac(i));}
long  fac(long n)
{ register  long  i, f=1;
  for(i=1; i<=n; i++)
  f=f * i; return(f);  }
```

运行结果：

```
1 ! = 1
2 ! = 2
3 ! = 6
4 ! = 24
5 ! = 120
```



寄存器类型变量的几点说明：

1. 局部自动变量类型和形参可定义为寄存器变量。
2. 不同C系统对寄存器的使用个数，对register变量的处理方法不同，对寄存器变量的数据类型有限制。
3. 局部静态变量不能定义为寄存器变量。
4. double, float等变量不能设为register型，因为超过寄存器长度

★用extern声明外部变量

外部变量也称全局变量，在函数外部定义，其作用域是从变量的定义处开始，到本程序文件的末尾。在定义的作用域内，全局变量可为程序中各个函数所引用。

可以用extern声明外部变量，以扩展外部变量的作用域。

❖在一个文件内声明外部变量

- 外部变量没在文件开头定义，其作用域为定义处到文件结束。定义处之前的函数要使用，则在引用前用关键字extern作“外部变量声明”。

★用extern声明外部变量

外部变量也称全局变量，在函数外部定义，其作用域是从变量的定义处开始，到本程序文件的未尾。在定义的作用域内，全局变量可为程序中各个函数所引用。

可以用extern声明外部变量，以扩展外部变量的作用域。

例35　用extern声明外部变量，扩展作用域

```
int  max(int x, int y)
{ int z;
  z=x>y ? x:y;
  return(z);
}
main( )
{ extern int A,B;
  printf("%d",max(A,B));
}
int  A=13,B=-8;
```

运行结果：
13

一般应将外部变量定义放在程序最前面，从而省去extern声明

可以省略为：
extern A，B；

例36　用extern扩展作用域

运行结果：

1: x=0　　y=0

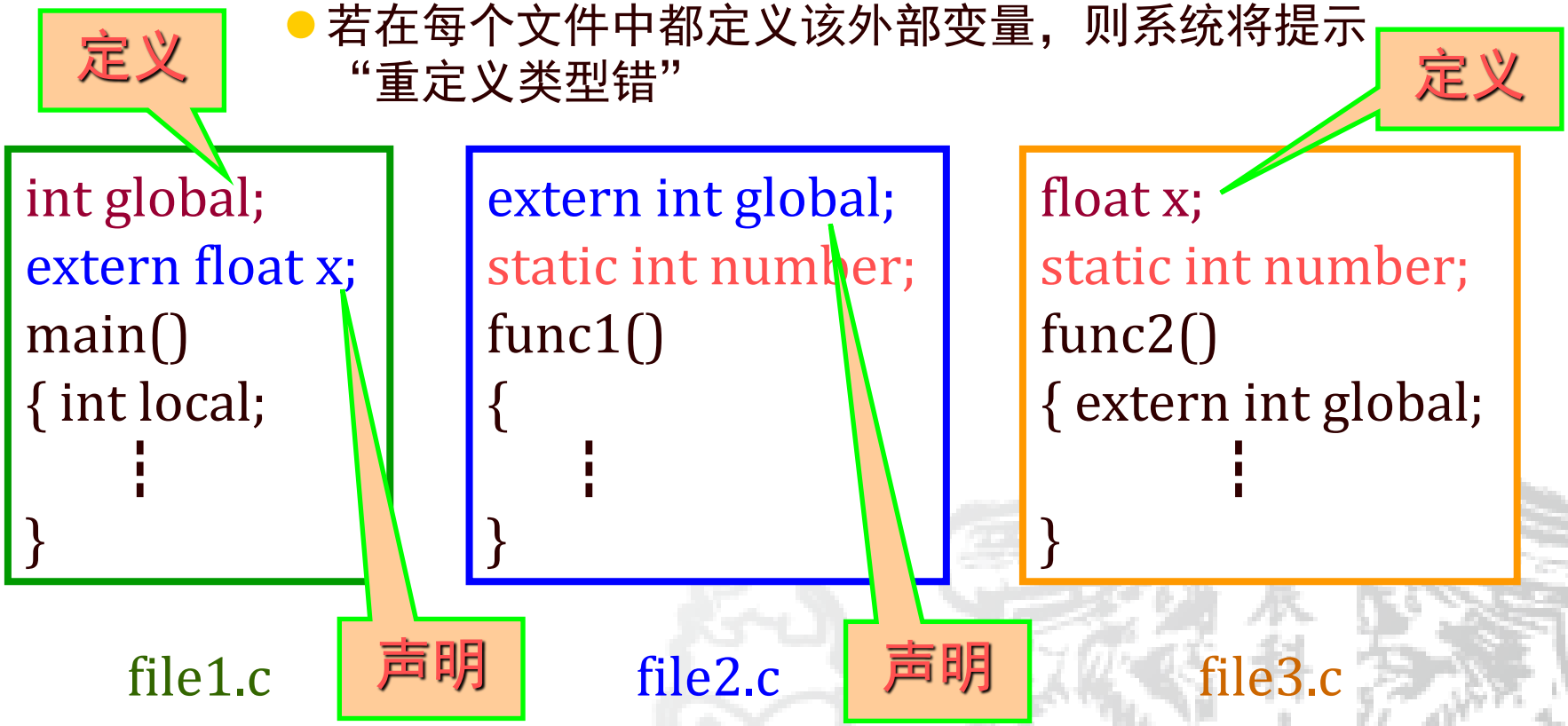
2: x=135　 y=246

3: x=135　 y=246

```
int main()
{ int  gx(),gy();
  extern  int X,Y;
  printf("1: x=%d\ty=%d\n",X,Y);
  Y=246;
  gx();
  gy();
}
int  gx()
{ extern  int  X,Y;
  X=135;
  printf("2: x=%d\ty=%d\n“,X,Y);
}
int X,Y;
int  gy()
{ printf("3: x=%d\ty=%d\n",X,Y);
}
```

❖在多文件的程序中声明外部变量

- 如果一个程序由多个文件组成，而一个外部变量需要在几个文件中引用，此时，可以在任一文件中定义该外部变量，在其它文件中用extern加以声明
- 若在每个文件中都定义该外部变量，则系统将提示“重定义类型错”



例37　用extern将外部变量的作用域扩展到其它文件

程序的作用是给定b的值，输入a和m，求a×b和a^m的值。

```
#include <stdio.h>   /*文件file1.c*/
#include “file2.c”
int A;
int main()
{ int power(int);
  int b=3,c,d,m;
  printf("Enter the number a and its power m:\n");
  scanf("%d,%d",&A,&m);
  c=A*b;
  printf("%d*%d=%d\n",A,b,c);
  d=power(m);
  printf("%d**%d=%d",A,m,d);
}
```

```
/*文件file2.c*/
extern A;
int power(int n)
{ int i,y=1;
  for(i=1;i<=n;i++)
    y*=A;
  return(y);
}
```

★用static声明外部变量

- 如果外部变量只允许本文件使用，不允许其它文件引用，则定义时加static声明。称为“静态外部变量”
- 只在工程方法中有效，在文件包含中则不起作用
- 常用于多人编同一程序，又使用同名变量时
- 加或不加static声明的外部变量都是静态存储，但其作用域不同

```
int global;
extern float x;
main()
{ int local;
      ⋮
}
```

file1.c

```
extern int global;
static int number;
func1()
{
      ⋮
}
```

file2.c

```
float x;
static int number;
func2()
{ extern int global;
      ⋮
}
```

file3.c

```
/*文件file1.c*/
#include  "file2.c"
static int A;
main()
{ A=5; printf("A1=%d\n", A);
  f1();
  printf("A4=%d\n",A);
}
/*文件file2.c */
extern int A;
f1()
{ printf("A2=%d\n",A);
  A=A*A;printf("A3=%d\n",A);
}
```

说明:
A1=5
A2=5
A3=25
A4=25
如果file3.prj内容：file1.c
file2.c
此时若有static 则编译指出:
A不可被引用

(1)在运行file1.c时,无论有无static,其结果都一样。
(2)如果标识符A在file2.c中无定义, 也可无static，这是因为用文件包含的方法，相当把file2.c包含进file1.c中来,使其成为file1.c的一部分了.
(2)若用工程的方法,则在file1.c中去掉#include “file2.c” ,建立工程文件file3.prj

★关于变量的声明和定义

- ❖函数：由“声明部分”和“执行语句”组成
- ❖声明部分：
  - 对有关的标识符(变量,函数,结构体)的属性进行说明，对于函数，声明和定义区别明显，声明是函数原型，定义是函数本身，是一个独立的程序模块；
- ❖变量的声明有两种情况
  - 定义性声明：需建立存储空间，如int a；也称定义。
  - 引用性声明：不建立存储空间，如extern A
  - 外部变量“定义性声明”只能一次，“引用性声明”多次。

广义地讲，声明包括定义，但并非所有的声明都是定义；
如：int A；既包含声明又包含定义；
extern A；只是声明，而无定义。

约定：建立存储空间的声明称定义；
不建立存储空间的声明称声明。

- Static定义性声明变量作用二个：
  - 局部变量用static定义性声明，分配的存储空间在程序执行期间始终存在，但作用域 只限定义它的函数或分程序。
  - 全局变量用static定义性声明，变量的作用域仅限本文件模块
- Auto,register,static是在定义变量的基础上加上这些关键字，不能单独作用。

存储类别小结

	局部变量			外部变量	
存储类别	auto	register	局部static	外部static	外部
存储方式	动态		静态		
存储区	动态区	寄存器	静态存储区		
生存期	函数调用开始至结束		程序整个运行期间		
作用域	定义变量的函数或复合语句内			本文件	其它文件
赋初值	每次函数调用时		编译时赋初值，只赋一次		
未赋初值	不确定		自动赋初值0或空字符		

- 局部变量默认为auto型
- register型变量个数受限，且不能为long，double，float型
- 局部static变量具有全局寿命、局部可见性和可继承性
- extern不是变量定义，可扩展外部变量作用域

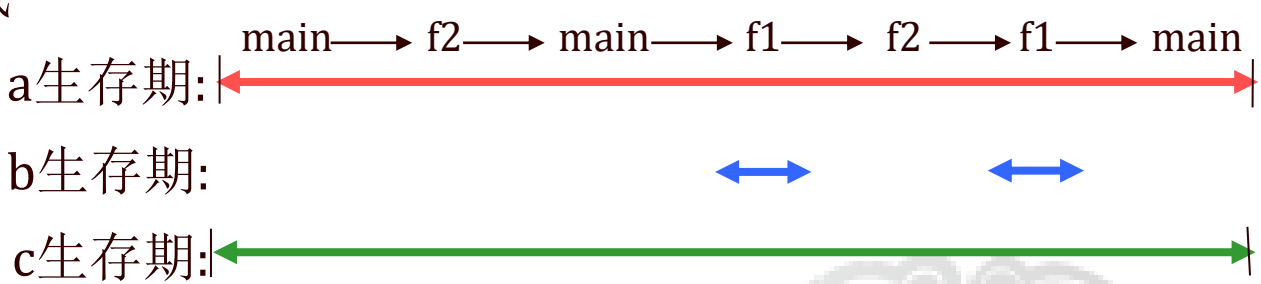
例 文件file1.c

```
int a;
main( )
{   .....
    .....
      f2;
    .....
      f1;
    .....
}
f1( )
{  auto int b;
    .........
     f2;
    ........
}
f2( )
{  static int c;
    .........
}
```

a作用域

b作用域

c作用域



§ 8.10 内部函数和外部函数

根据函数能否被其它源文件调用，将函数分为内部函数和外部函数。

★内部函数——静态函数

- ❖只能被本文件中其它函数所调用
- ❖定义形式：

  static 类型标识符 函数名（形参表）

  如： static int fun(int a, int b)

- ❖内部函数，其作用域仅限于定义它的所在文件。此时，在其它的文件中可以有相同的函数名，它们相互之间互不干扰。

★外部函数

- ❖能被其它文件中的函数所调用
- ❖定义形式：

  extern 类型标识符 函数名（形参表）

如：extern int fun(int a, int b)

- ❖省略extern，隐含为外部函数
- ❖调用此函数的文件中也可以用extern声明所用函数是外部函数

例38　add函数与main函数不在同一个文件中

t1.c

```
#include<stdio.h>
int add(int a, int b)
{
        return a+b;
}
```

t2.c

```
#include<stdio.h>
#include"t1.c"
int main()
{
        //extern add(int a,int b);
        int a = 1;
        int b = 2;
        int c;
        c = add(a,b);
        printf("%d\n",c);
}
```

例39　有一个字符串，内有若干个字符，今输入一个字符，要求程序将字符串中该字符删去。用外部函数实现。

分析：该问题可以用四个函数解决。
主函数main( )，　　删除字符函数delete_char( )，
输入字符串函数enter_str( )，输出新字符串函数print_str( )，
将各函数放入四个文件，用extern声明实现各文件中函数的调用。

```
/*文件file1.c*/
#include"file2.c"
#include"file3.c"
#include"file4.c"
#include <stdio.h>
int main()
{ extern  enter_string(char str[80] );
  extern delete_string(char str[],char ch );
  extern print_string(char str[ ] );
  char  c;
  char  str[80];
  enter_string(str);
  scanf("%c",&c);
  delete_string(str,c);
  print_string(str);
}
```

初始	删除后	
a	a	str[0]
b	b	str[1]
c	d	str[2]
c	\0	str[3]
d	d	str[4]
c	c	str[5]
\0	\0	str[6]
⋮	⋮	⋮
\0	\0	str[79]

i	j	str[i] !='c' str[i] ← str[j++]
0	0	str[0] !='c' str[0] ← str[0]=a
	1	
1	2	str[1] !='c' str[1] ← str[1]=b
2		str[2] !='c'
3		str[3] !='c'
4	3	str[4] !='c' str[2] ← str[4]=d
5		str[5] !='c'
⋮	⋮	⋮
79		str[79] !='c' str[3] ← '\0'

```
/*文件file2.c*/
#include  "stdio.h "
int enter_string(char str[80])
{ gets(str); }
```

```
/*文件file3.c*/
#include <stdio.h>
int delete_string(char str[ ],char ch)
{ int  i, j;
  for(i=j=0; str[i] !='\0' ; i++)
  if(str[i] != ch) str[j++]=str[i];
  str[j]='\0';
}
```

```
/*文件file4.c*/
#include <stdio.h>
int print_string(char str[ ])
{ printf("%s",str);}
```

输入：
abccdc↵
c↵
输出：abd

初始	删除后	
a	a	str[0]
b	b	str[1]
c	d	str[2]
c	\0	str[3]
d	d	str[4]
c	c	str[5]
\0	\0	str[6]
⋮	⋮	⋮
\0	\0	str[79]

i	j	str[i] !='c' str[i] ← str[j++]
0	0	str[0] !='c' str[0] ← str[0]=a
	1	
1	2	str[1] !='c' str[1] ← str[1]=b
2		str[2] !='c'
3		str[3] !='c'
4	3	str[4] !='c' str[2] ← str[4]=d
5		str[5] !='c'
⋮	⋮	⋮
79		str[79] !='c' str[3] ← '\0'

本章小结

- ★函数包含一系列程序语句，它们被集中在一起并给它们起一个名字；
- ★每一个函数在被调用之前必须声明；
- ★每一个函数都必须有一个相应于其原型的函数实现，给出该函数的具体细节；
- ★一个有返回值的函数一定要返回一个与其声明时类型相匹配的值；
- ★一个void型的函数没有任何返回值；
- ★在函数内部定义的变量，包括函数的形式参数都是局部变量，它们在函数的外部是不可见的；
- ★当函数返回时，它精确地回到它被调用的那一点；
- ★一般运用逐步求精的方法，从主函数开始逐步实现各个功能函数。

★几点说明：

(1)一个源文件由一个或者多个函数组成。

(2)一个C程序由一个或者多个源文件组成。

(3)C程序的执行从main 函数开始。

(4)所有的子函数都是平行的。

(5)从用户的角度看，函数分库函数和自定义函数。

(6)函数形式：

①**无参函数**：主调函数无数据传送给被调函数,可带或不带返回值。

②**有参函数**：主调函数与被调函数间有参数传递,主调函数可将实参传送给被调函数的形参,被调函数的数据可返回主调函数。

- 根据(1)(2)(3)可知，
- 逻辑上:一个C语言程序是由函数构成的，C语言程序从主函数开始执行，在主函数中调用其他函数，这些函数可能又调用别的函数，主函数执行完毕代表整个程序结束。主函数只能调用不能被调用。 "
- 物理上:一个程序由一个或者若干个文件(源文件)构成，函数分别放置在这些文件中。