函 数（续）

Wang Houfeng
EECS,PKU
wanghf@pku.edu.cn

内容

- **返回值**
- 参数传递
- 变量的作用域
- 变量的存储属性

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：控制被调用函数返回到调用函数中，同时把值返回给调用函数
  - 说明：
    - 函数中可有**多个return语句**
    - 若函数类型与return语句中表达式值的类型不一致，按函数类型返回值
    - void型函数可以不返回值
    - **程序中可以无return语句，遇函数最后 } 时，自动返回调用函数**

到return后立刻停止

无返回值的函数

```
#include <iostream>
using namespace std;
void printstar()
{   cout<<"**********";
}
int main()
{  int a=10;
   printstar();
   cout<<a<<endl;
   return 0;
}
```

输出： **********10

```
#include <iostream>
using namespace std;
void printstar()
{       cout<<"**********";
}
int main()
{  int a;
   a=printstar();
   cout<<a<<endl;
   return 0;
}
```

编译错误！

两个例子

```
float fmax(float x, float y)
{   float z;
    z=x>y?x:y;
    return(z);
}
main()
{   float a,b;
    int c;
    cin>>a>>b;
    c=fmax(a,b);
    cout<<"Max is "<<c;
}
```

类型一致！

```
int imax(float x, float y)
{   float z;
    z=x>y?x:y;
    return(z);
}
main()
{   float a,b;
    int c;
    cin>>a>>b;
    c=imax(a,b);
    cout<<"Max is "<<c;
}
```

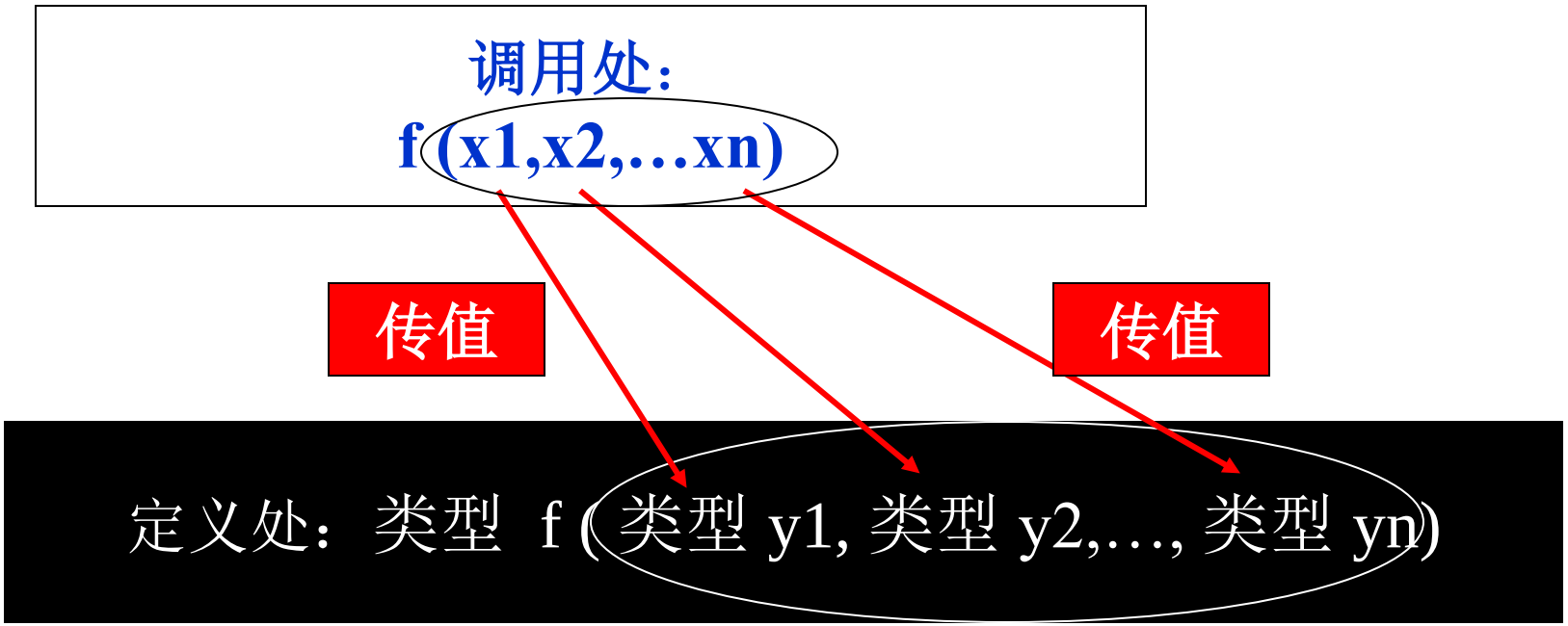
类型不一致！

类型要尽量一致！

内容

- 返回值
- **参数传递**
- 变量的作用域
- 变量的存储属性

参数传递



参数传递

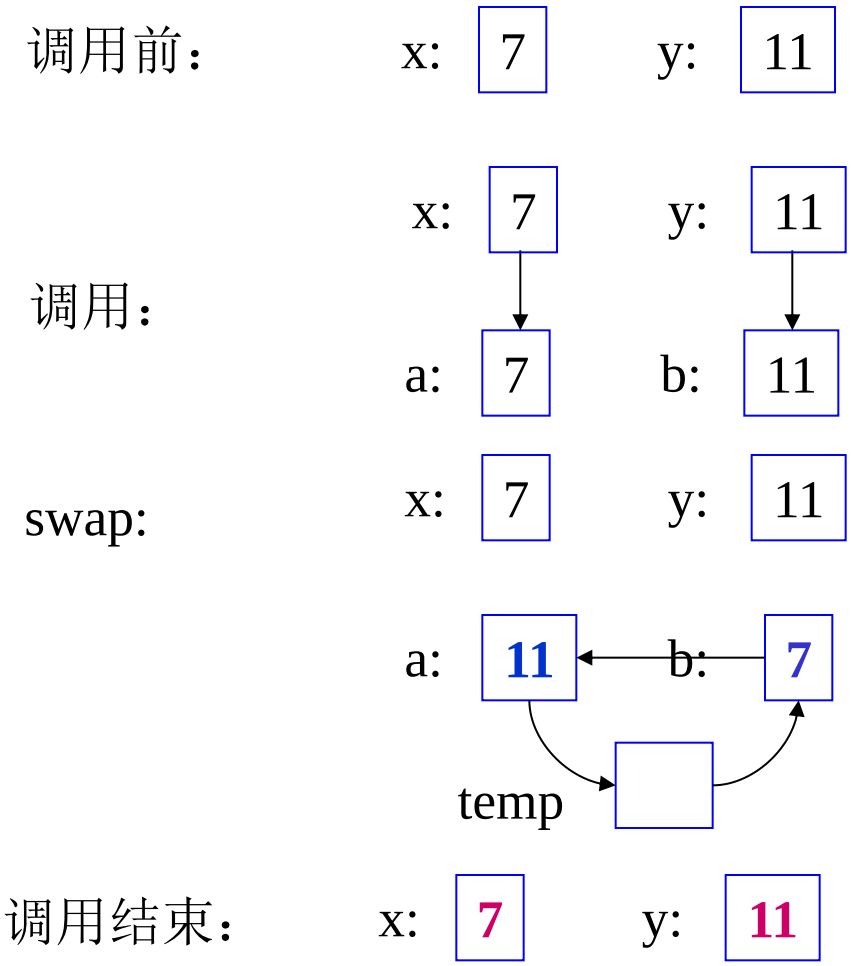
- 实际参数传给形式参数，直接 **copy传值**；
- 每个参数会分配临时存储单元存放实际参数的值，函数调用完后释放。实参可以是表达式；
- 实参与形参个数相同；
- 实参与形参满足赋值兼容性以及内部类型转换原则；
- 参数传递是**单向**的，不能通过实参带回值。

交换两个数（能交换吗！）

```
#include <iostream>
using namespace std;
void swap(int a,int b)
{   int temp;
    temp=a; a=b; b=temp;
}

int main()
{  int x=7,y=11;
    cout<<"x="<<x<<"y="<<y<<endl;
    cout<<"swapped:"<<endl;
    swap(x,y);
    cout<<"x="<<x<<"y="<<y<<endl;
    return 0;
}
```

最后输出什么？



数组作为函数参数

- 两种参数传递方式：
  - 数组元素作为参数；
  - 数组整体本身作为参数；

数组元素作为参数传递——传值

例 两个数组大小比较

a和b是各有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 <iostream>
using namespace std;

int main()
{   int a[10],b[10],i,n=0;
    cout<<"Enter array a:"<<endl;
    for(i=0;i<10;i++)
            cin>>a[i];
    cout<<"Enter array b:"<<endl;
    for(i=0;i<10;i++)
            cin>>b[i];
    for(i=0;i<10;i++)
    {
       n=n+large(a[i],b[i]);
    }
    if n>0 cout<<" a>b";
    /*  下面的输出自己补充 */
}
```

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

元素作为参数等同于
单个变量传递

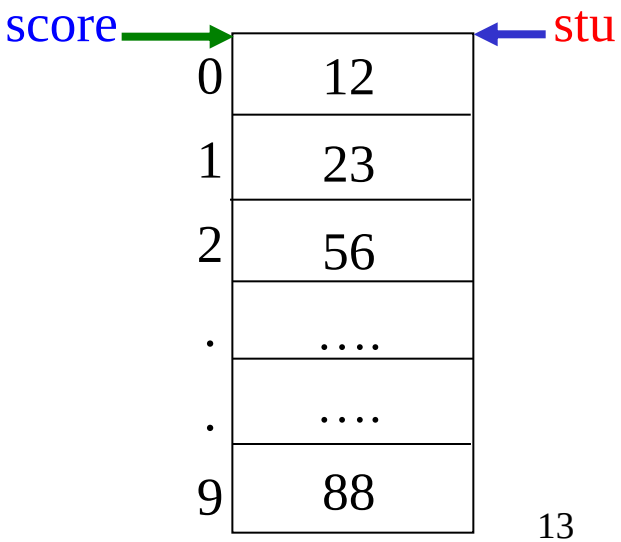
数组作函数参数——地址传递

```
#include <iostream>
using namespace std;
float average(int stu[], int n);
void main()
{ int score[10], i;
  float  av;
  cout<<"Input  10  scores：  "<<endl;
  for( i=0; i<10; i++ )
     cin>>score[i];
  av=average(score,10);
  cout<<"Average  is：  "<< av<<endl;
}
```

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= total+stu[i];
  av = total/n;
  return av;
}
```

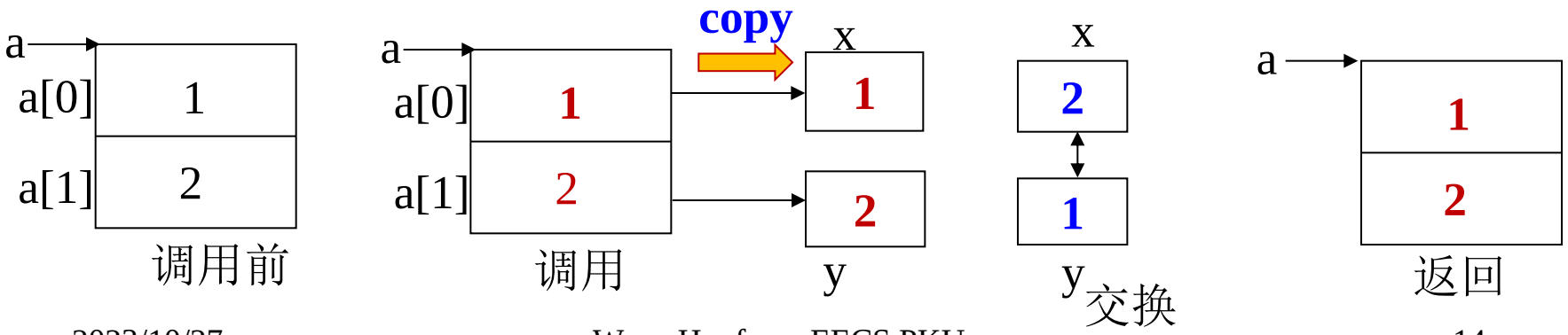
计算平均成绩



数组元素与数组传递比较

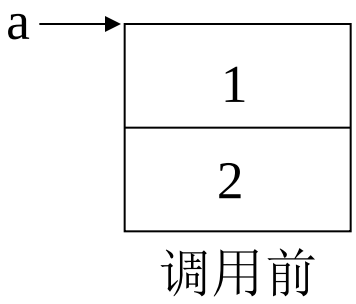
值传递，
不带回值

```
#include <iostream>
using namespace std;
void swap2(int x,int y)
{   int z;
    z=x;      x=y;      y=z;
}
int main()
{   int a[2]={1,2};
    swap2(a[0],a[1]);
    cout<<"a[0]="<<a[0]<<endl<<"a[1]="<<a[1]<<endl;
    return 0;
}
```

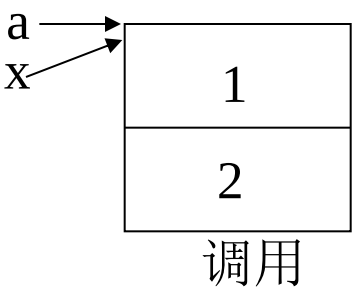


地址传递，带回值

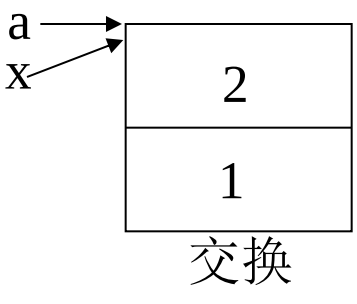
```
#include <iostream>
using namespace std;
void swap2(int x[])
{   int z;
    z=x[0];     x[0]=x[1];     x[1]=z;
}
int main()
{  int a[2]={1,2};
    swap2(a);
    cout<<"a[0]="<<a[0]<<endl<<"a[1]="<<a[1]<<endl;
    return 0;
}
```



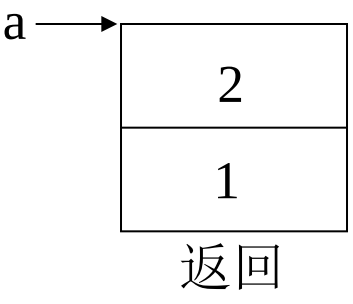
调用前



调用



交换

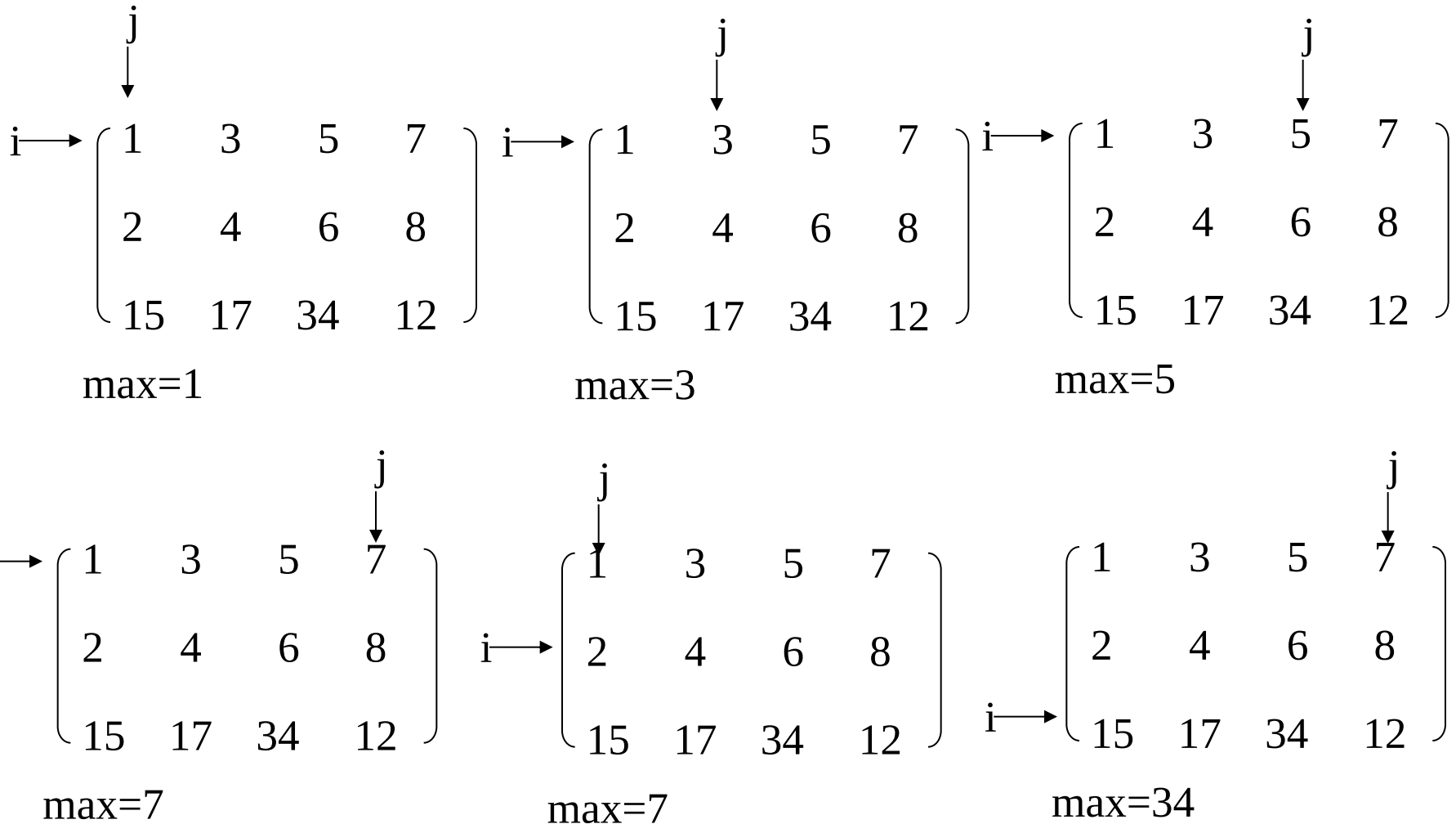


返回

数组与元素传递

- 数组传递：传地址，不为参数分配临时空间，函数运行期间，**直接使用原数组**；
- 元素（单变量）传递：传值，当函数被调用时，为参变量分配**临时空间**，将**实参值拷贝到临时空间，函数对参数的使用直接在临时空间使用**。

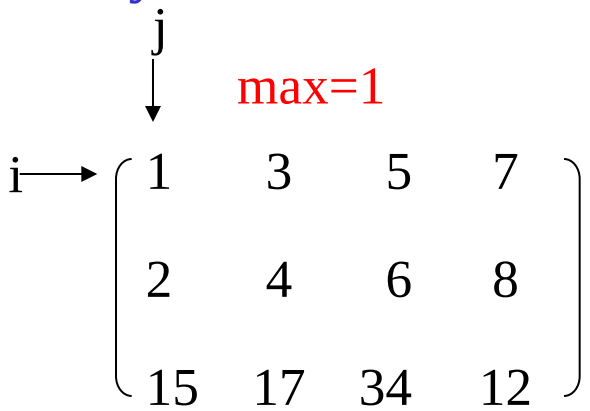
例 求二维数组中最大元素值



```
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()
{ int a[3][4]={{1,3,5,7},
                        {2,4,6,8},{15,17,34,12}};
    cout<<"max value is"<<max_value(a)<<endl;
    return 0;
}
```

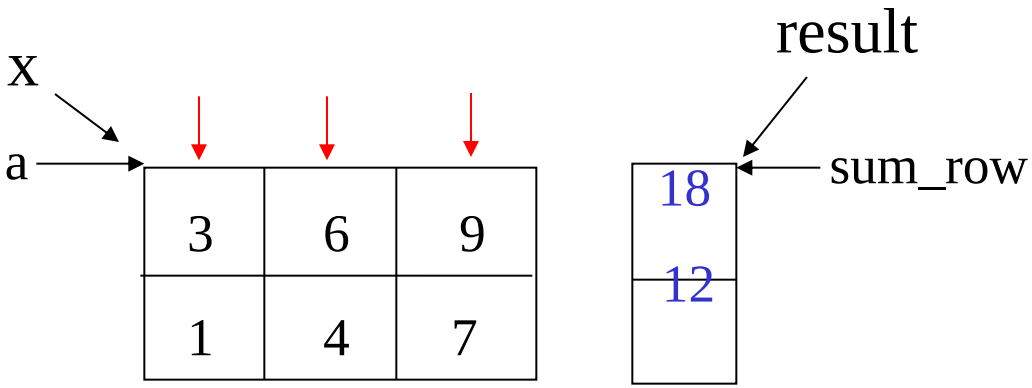
多维形参数组第一维维数可省略,第二维必须相同

⇔ int array[][4]



例 求二维数组中各行元素之和

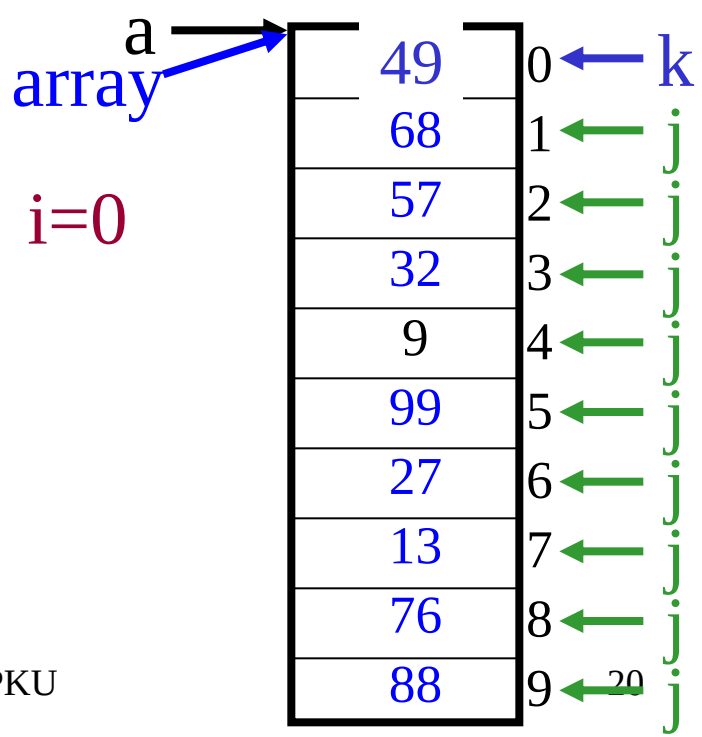
```
void get_sum_row(int  x[][3], int  result[] ,int  row, int  col)
{   int i,j;
    for(i=0;i<row;i++)
    {     result[i]=0;
            for(j=0;j<col;j++)
                result[i]+=x[i][j];
    }
}
int main()
{   int a[2][3]={3,6,9,1,4,7};
    int sum_row[2],row=2,col=3,i;
    get_sum_row(a,sum_row,row,col);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

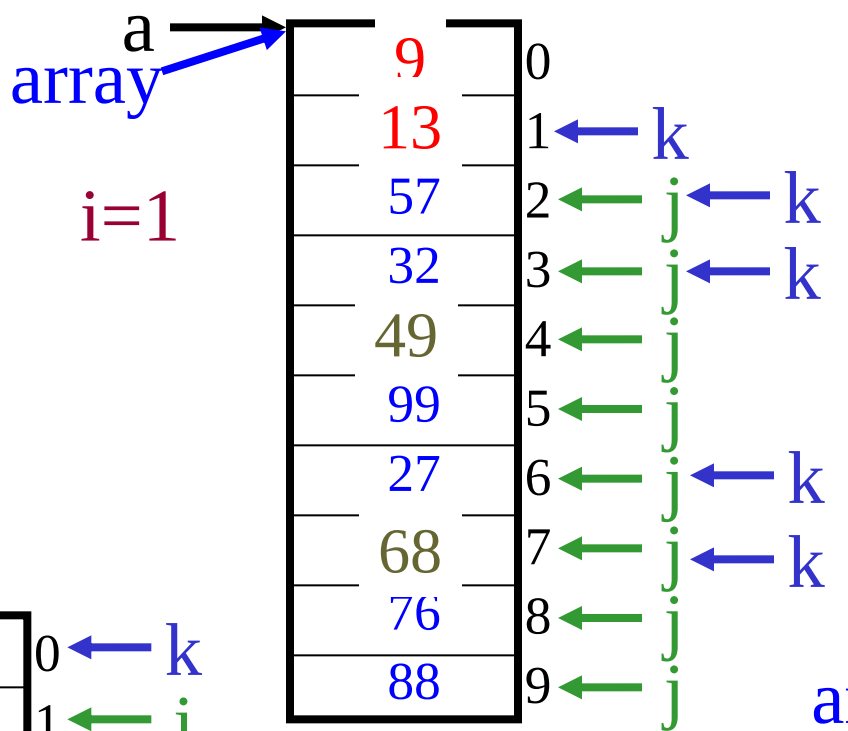


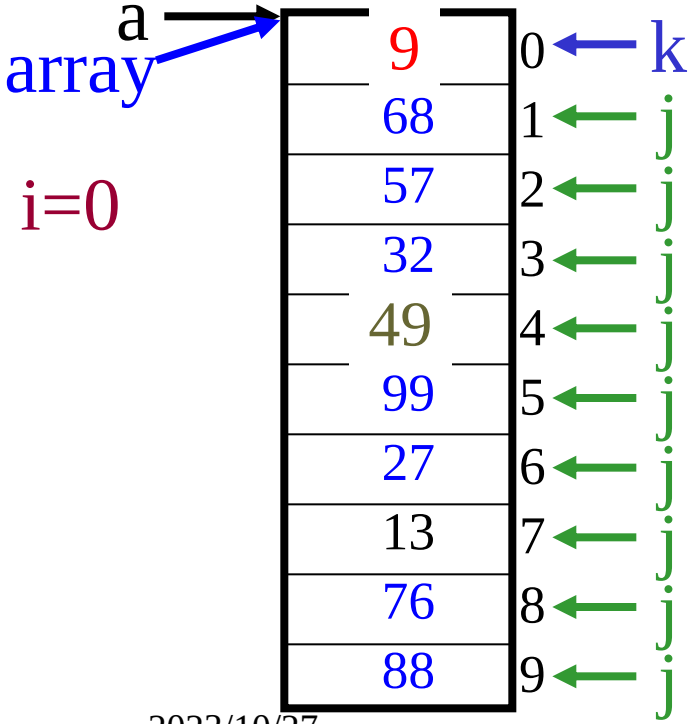
数组排序----简单选择排序

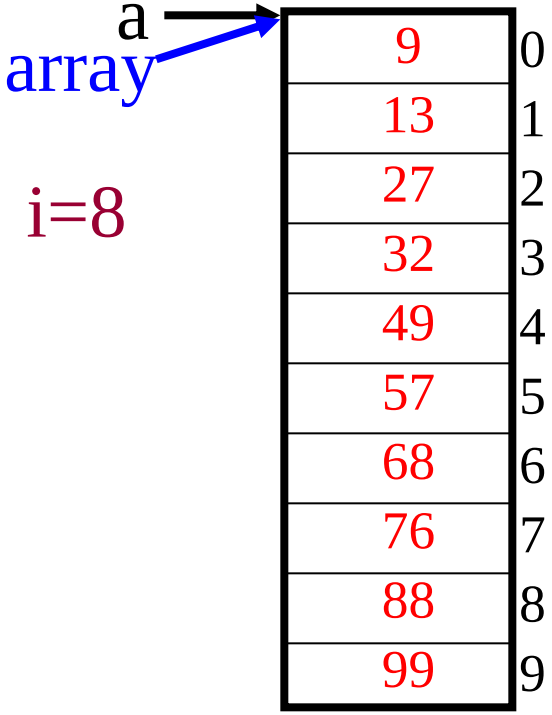
```
void sort(int  array[],int  n)
{    int i,j,k,t;
     for(i=0;i<n-1;i++)
     {     k=i;
           for(j=i+1;j<n;j++)
             if(array[j]<array[k])   k=j;
           if(k!=i)
           {    t=array[i];
                array[i]=array[k];
                array[k]=t;
           }
     }
}
```

```
int main()
{   int a[10],i;
    for(i=0;i<10;i++)
          cin>>a[i];
    sort(a,10);
    for(i=0;i<10;i++)
        cout<<a[i];
    cout<<endl;
    return 0;
}
```




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

数组作为参数传递小结

- 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，编译器不对实参个数（元素个数）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，最高维可以不指定大小，但其它维必须指定。

内容

- 返回值
- 参数传递
- **变量的作用域**
- 变量的存储属性

变量的定义与使用

- 任何变量必须先声明，后使用（声明之后使用）
- 回忆：函数也应该先定义，后使用（否则，应引入原型声明）

变量在何处声明

```
#include <iostream>
using namespace std;
int k=6;     // 所有程序外面
int max (int x, int y)  //参数部分
{
    int  temp;  //函数体中
    temp=x;
    if (y>x) temp=y;
    return temp;
}
int m;  // 所有函数外部
int main()
{
  int n;   //函数体中
  cin>>m>>n;
    {
      int k;  //复合语句中
      k=max(m,n);
    }
  cout<<k<<endl;
}
```

若输入 **4 5**
结果？

局部 VS 全局

- C语言程序的函数具有并列结构，使得变量的声明不是在函数的内部就是在函数的外部。
- 具体讲，变量可以在程序文件的四个地方：
  - 在一个函数的函数体内部；
  - 在一个函数内的复合语句中；
  - 在函数的参数部分出现（形式参数）；
  - 在所有函数的外部；

前三种情况称为局部变量，后一种称为全局变量

作用域：局部变量与全局变量

- 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
    - » 不同函数可以有同名变量，表示不同变量，占不同内存单元
    - » 形参属于局部变量
    - » 可在复合语句中定义变量，为局部变量
    - » 局部变量可用存储类型：auto，register，static （默认为auto）
    - » 局部具有相对性！

局部变量

- 局部变量是相对的。

```
for( j=0; j<10;j++)
  {
        int k;
        …
  }
```

其中 k 属于局部变量。只在循环语句中起作用，所属函数的其它变量，又成为相对全局。

- 任何局部变量只在声明它的局部有效。当超出其范围时，变量没有任何意义（变量不存在）。
- 在不同的局部可以有相同名字的变量，但表示不同变量，互不干扰。

不同函数中同名变量（实际上是不同变量）

```
void sub()
{   int a,b;
    a=6;
    b=7;
    cout<<"sub:a="<<a<<"b="<<b<<endl;
}
int main()
{   int a,b;
    a=3;
    b=4;
   cout<<"main:a="<<a<<"b="<<b<<endl;
   sub();
   cout<<"main:a="<<a<<"b="<<b<<endl;
   return 0;
}
```

运行结果：

main:a=3,b=4

sub:a=6,b=7

main:a=3,b=4

不同层次内可以声明同名变量（表示不同变量）

```
#define   N   5
int main()
{  int i，  temp=8;
    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++)
      cout<<a[i];
   return 0;
}
```

变量 i, a 相对于内层更局部的变量 temp 又为的全局。外层的 temp 与复合语句中的temp表示不同变量

功能是什么？
输出什么？

建议尽量减少同名情况的发生，以免影响对程序的理解

全局变量

- 定义：在函数外定义，可为本文件所有函数共用
  - 注意有效范围：从声明变量的位置开始到本文件结束前起作用

应尽量少使用全局变量，因为：

✡ 全局变量在程序全部执行过程中占用存储单元，导致空间使用的浪费

✡ 降低了函数的通用性、可靠性，可移植性

✡ 降低程序清晰性，容易出错

✡ 关联性太强，使函数本身的独立性太弱

```
float   max,min;
float  average(float   array[],  int  n)
{  int i;   float  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+=array[i];
   }
   return(sum/n);
}
main()
{   int i;
    float ave,score[10];
   /*Input  */
   ave=average(score,10);
   printf("max=%6.2f\nmin=%6.2f\n
          average=%6.2f\n",max,min,ave);
}
```

如不用全局变量，如何带回最大/小值

max

min

作用域

同名变量的作用范围

- **全局变量与局部变量同名，在局部时，局部变量起作用**

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

运行结果：max=8

全局变量的副作用

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

没有同名局部变量时，全局变量起作用

运行结果：

*****

不是25个 *

内容

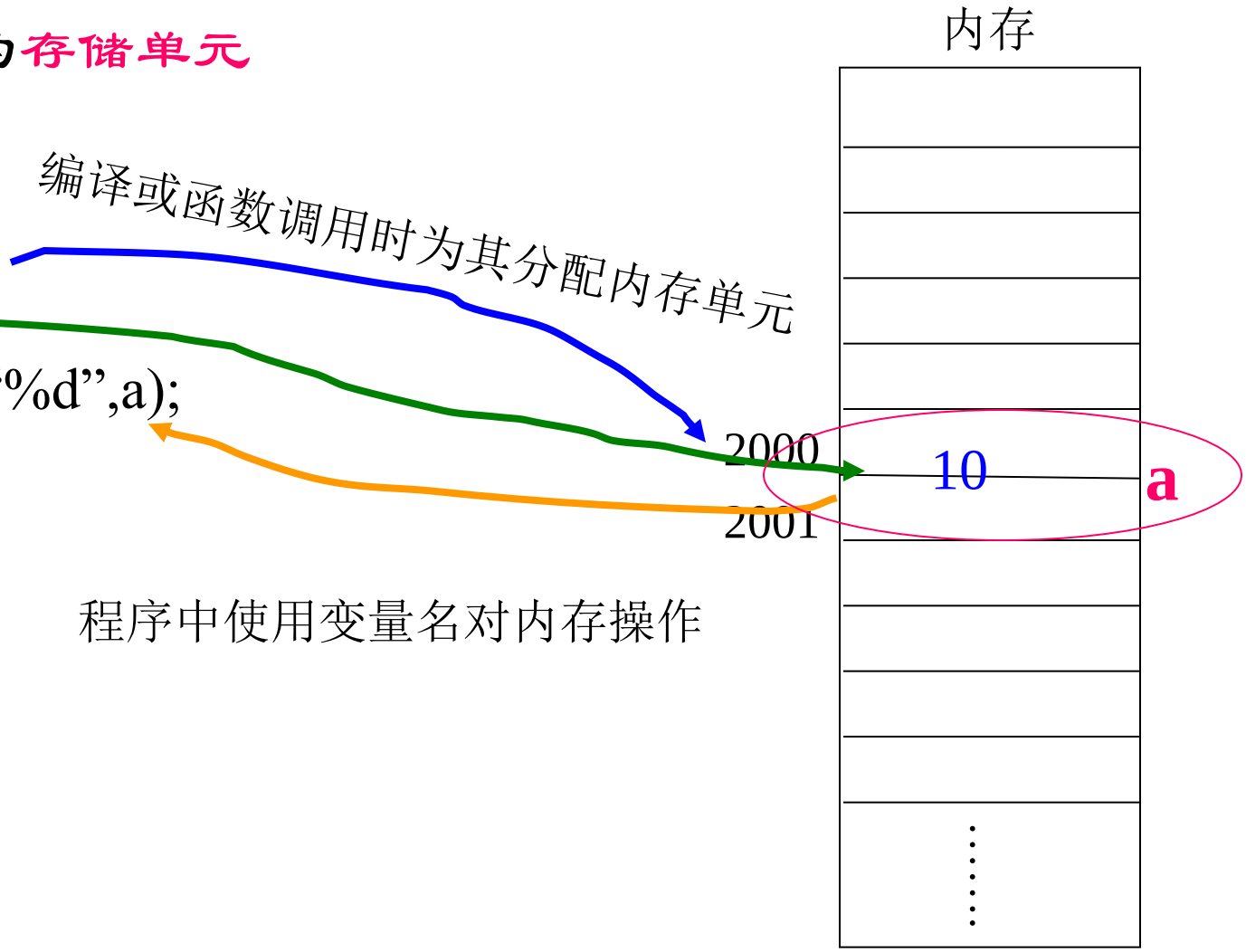
- 返回值
- 函数调用的参数传递
- 变量的作用域
- **变量的存储属性**

变量的存储属性

— 概述

- 变量是命名的存储单元

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



程序中使用变量名对内存操作

变量的存储属性

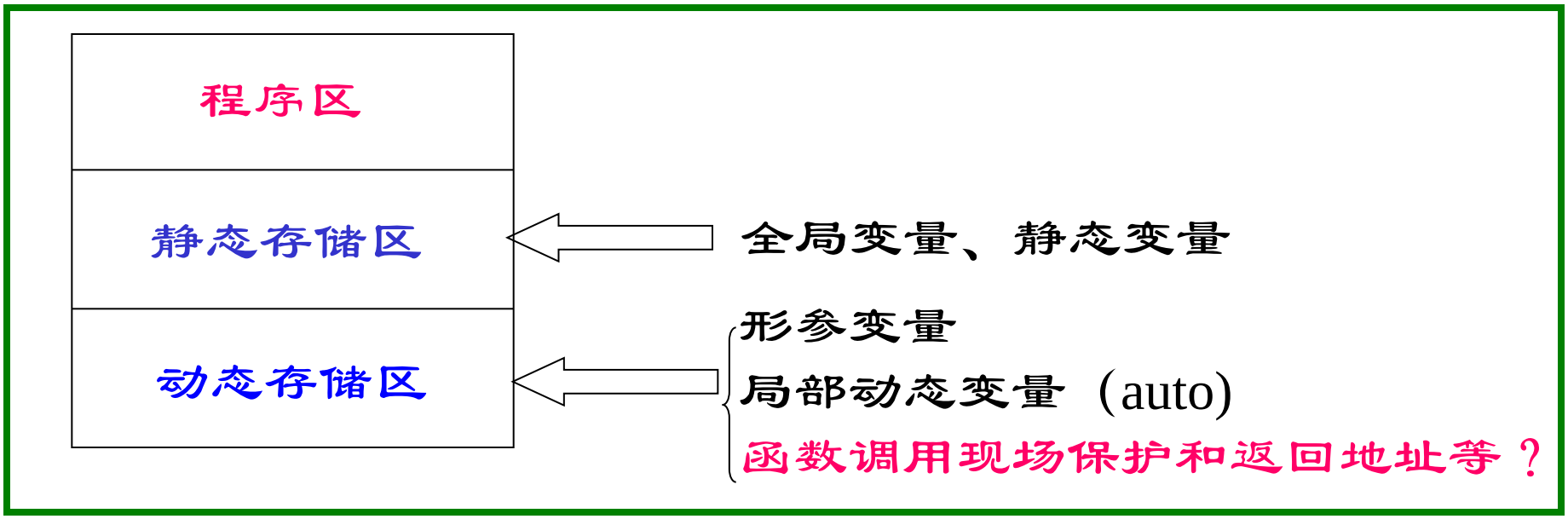
– 概述

- **变量是对存储空间（单元）的抽象**
- 变量的属性
  - 数据类型：变量中的数据类型（整型、浮点型、…）
  - 存储属性
    - » 存储器类型：寄存器、静态存储区、动态存储区
    - » 生存期：变量在某时间段有效-------静态变量与动态变量
    - » 作用域：变量在某范围内有效-------局部变量与全局变量
- 变量的存储类型
  - auto -----自动型 (√)
  - register-----寄存器型
  - static ------静态型(√)
  - extern -----外部型

例子：
```
int   sum;
auto   int   a,b,c;
register   int   i;
static   float   x,y;
```

- 变量定义格式: [存储类型] 数据类型 变量表;

C 语言的存储区及其意义



- 生存期
  - 静态变量：从程序开始执行到程序结束都起作用
  - 动态变量：从包含该变量定义的函数开始执行至函数结束（只在声明范围内）

	局部变量			全局变量	
存储类别	auto	register	static	全局	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变量具有全局寿命和局部可见性
- 局部static变量具有可继承性
- extern不是变量定义，可扩展外部变量作用域

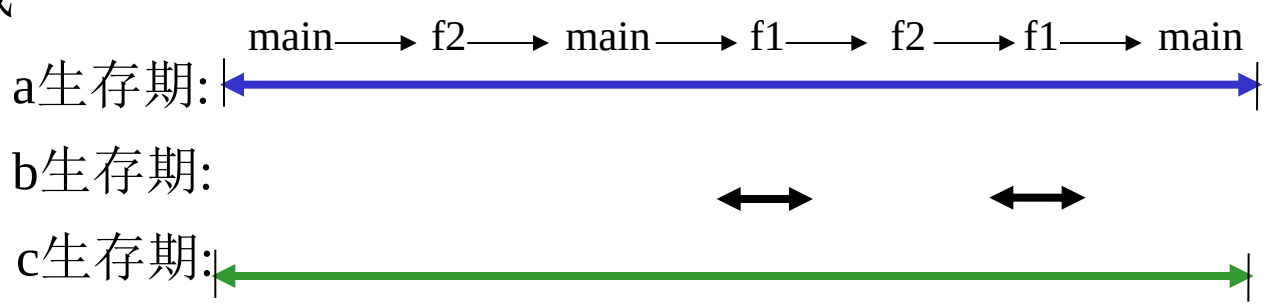
例　文件file1.c

```
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



auto 变量的作用域

```
#include <stdio.h>
void  prt(void);
main()
{   int x=1;
     {   int x=3;
         prt();
         printf("2nd x=%d\n",x);
     }
     printf("1st x=%d\n",x);
}
void  prt(void)
{   int x=5;
    printf("3rd  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：

```
3rd   x=5
2nd  x=3
1st   x=1
```

例　局部静态变量值具有可继承性

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

运行结果：1
1
1

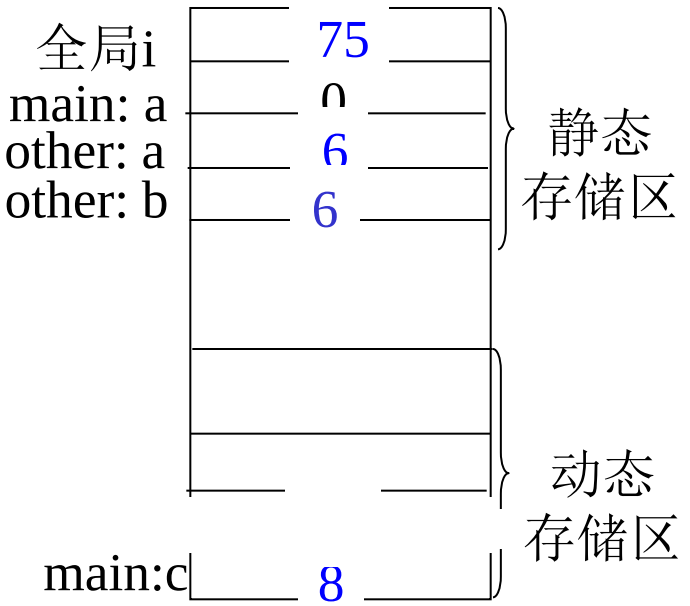
```
void  increment(void);
main()
{
    increment();
    increment();
    increment();
}
void  increment(void)
{   static int x=0;
    x++;
    printf("%d\n",x);
}
```

运行结果：1
2
3

例 变量生命期与可见性

```
#include <stdio.h>
int i=1;
main()
{   static int a;
    register int b=-10;
    int c=0;
    printf("-----MAIN------\n");
    printf("i:%d a:%d  \
      b:%d c:%d\n",i,a,b,c);
    c=c+8;
    other();
    printf("-----MAIN------\n");
    printf("i:%d a:%d   \
        b:%d c:%d\n",i,a,b,c);
    i=i+10;
    other();
}
```

```
-------Main------
i:1  a:0  b:-10  c:0
------Other------
i:33  a:4  b:0  c:15
-------Main-----
i:33  a:0  b:-10  c:8
-------Other-------
i:75  a:6  b:4  c:15
```




```
other()
{   static int a=2;
    static int b;
    int c=10;
    a=a+2;   i=i+32;  c=c+5;
    printf("-----OTHER------\n");
    printf("i:%d a:%d   \
        b:%d c:%d\n",i,a,b,c);
    b=a;
}
```

外部变量（一般了解）

- 外部变量说明： extern 数据类型 变量表；
- 若外部变量与局部变量同名，在局部时，外部变量被屏蔽
- 主要用于多文件表示源程序的情况
- 作用域同全局变

函数问题要点

- 何时设计函数？
  - 有利于功能共享时
  - 有利于结构清晰时
- 参数传递的两种形式
  - 传值（单变量参数）
  - 传地址（数组变量）

函数问题要点

- 生命期
  - 全局变量、静态（static）变量总处于活动状态;
  - 局部变量仅在局部运行时活动;
- 作用域
  - 全局变量在定义处往后总起作用;
  - 局部变量仅在局部起作用;
- 初始化
  - 静态变量仅在第一次使用时初始化。