



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计 Programming in C



1011014

主讲：姜学锋，计算机学院

函数之间数据交换的高效方法

- ◆ 1、指针作为函数参数.....●
- ◆ 2、数组作为函数参数的实质.....●

7.5 指针与函数

- ▶ 指针最重要的应用是作为函数参数，它使得被调函数除了返回值之外，能够将更多的运算结果返回到主调函数中，即**指针是函数参数传递的重要工具**。

7.5.1 指针作为函数参数

- ▶ 函数参数不仅可以是基本类型等普通对象，还可以是数组、指针变量。

7.5.1 指针作为函数参数

- ▶ 1. 指针变量作为函数形参
- ▶ 函数形参可以是指针类型，一般形式为：

```
返回类型 函数名(指向类型 *指针变量名, . . . . .)  
{  
    函数体  
}
```

- ▶ 相应的，调用函数时必须用相同指向类型的指针（或地址）作为函数实参。

7.5.1 指针作为函数参数



【例7.18】

输入a和b两个整数，按从小到大的顺序输出a、b。

7.5.1 指针作为函数参数

例7.18

```
1 #include <stdio.h>
2 void swap(int *p1, int *p2)
3 {
4     int t;
5     t=*p1 , *p1=*p2, *p2=t; //交换*p1和*p2
6 }
7 int main()
8 {
9     int a, b;
10    scanf("%d%d",&a,&b); //输入
11    if (a>b) swap(&a, &b);
12    printf("min=%d,max=%d\n",a,b); //输出
13    return 0;
14 }
```

7.5.1 指针作为函数参数

例7.18

程序运行屏幕



7.5.1 指针作为函数参数

- ▶ swap函数的作用是交换a和b的值。由于swap函数的两个形参是指向int型的指针变量，因此调用swap函数时实参必须是指向int型的指针或地址，例如：

```
swap(&a, &b); //正确 实参&a、&b为int*，与形参要求一致
```

- ▶ 如果换成

```
swap(a, b); //错误 实参a、b为int，与形参指针类型要求不一致
```

- ▶ 是错误的。

7.5.1 指针作为函数参数

- ▶ 即便是函数参数，C语言不会对任何指针类型做隐式类型转换。

7.5.1 指针作为函数参数

例7.18

```
1 #include <stdio.h>
2 void swap(int *p1, int *p2)
3 {
4     int t;
5     t=*p1 , *p1=*p2, *p2=t; //交换*p1和*p2
6 }
```

下面分析swap函数是如何交换a和b的值。

调用swap时，实参分别是变量a和b的地址，根据函数参数的值传递规则，swap函数的形参指针变量p1得到了变量a的地址，p2得到了变量b的地址。换句话说，此时p1指向变量a，p2指向变量b，*p1等价于a，*p2等价于b。

7.5.1 指针作为函数参数

例7.18

```
1 #include <stdio.h>
2 void swap(int *p1, int *p2)
3 {
4     int t;
5     t=*p1 , *p1=*p2, *p2=t; //交换*p1和*p2
6 }
```

那么

```
t=*p1 , *p1=*p2, *p2=t;
```


实际效果相当于

```
t=a, a=b, b=t;
```

结果是a和b的值交换了。

7.5.1 指针作为函数参数

- ▶ 可不可以直接在swap函数写“t=a , a=b, b=t;”交换a和b呢？
- ▶ 答案是不可以。因为a和b是main函数定义的局部变量，其作用域仅在main函数内部有效，对其他任何函数来说不可见。

```
void swap(...)  
{  
    int t;  
    t=a , a=b, b=t;   
}
```

7.5.1 指针作为函数参数

- ▶ 如果将swap函数写成

```
void swap(int p1, int p2)
{
    int t;
    t=p1, p1=p2, p2=t;
}
```

- ▶ 在main函数的调用写成

```
swap(a, b);
```

7.5.1 指针作为函数参数

- ▶ 能不能实现a和b交换呢？答案是不行。
- ▶ 因为这样写的含义是：a和b的值传递给了形参p1和p2，p1和p2是a和b的副本，在swap函数中交换了p1和p2的值，当swap调用结束返回到main函数中，形参p1和p2存储空间释放，而main函数a和b的值始终未变。

7.5.1 指针作为函数参数

- ▶ 如果将swap函数写成

```
void swap(int *p1, int *p2)
{
    int *t;
    t=p1, p1=p2, p2=t; //指针交换
}
```

- ▶ 能不能实现a和b交换呢？答案是不行。因为这样写的含义是：在swap函数中交换形参p1和p2的指针值，即交换后仅是p1和p2的指向发生了变化，而被指向的a和b的值始终未变。

7.5.1 指针作为函数参数

- ▶ 为了使被调函数能够改变主调函数的变量，应该
 - ▶ 用指针变量作为形参，
 - ▶ 将变量的指针（或地址）传递到被调函数中，
 - ▶ 通过指针间接引用达到修改变量的目的。
-
- ▶ 当函数调用结束后，这些变量值的变化依然保留下来。换个角度看，这些变量带回了被调函数所做的修改，将运算结果返回到了主调函数中。

7.5.1 指针作为函数参数

- ▶ 显然，函数返回运算结果的前提有三个：
- ▶ ①使用指针变量作为函数形参；
- ▶ ②用接受运算结果的变量的指针（或地址）作为实参调用函数；
- ▶ ③函数中通过指针间接引用修改这些变量。

7.5.1 指针作为函数参数

- ▶ 函数通过返回值只能返回一个运算结果，若要返回多个，就需要使用全局变量（因为全局变量对两个函数是可见的）。但全局变量使得函数模块化程度降低，现代程序设计思想要求尽量避免全局变量的使用。
- ▶ 通过将指针作为函数参数的方法，既可以返回多个运算结果，又避免了使用全局变量。

7.5.1 指针作为函数参数



【例7.19】

编写函数，计算并返回a和b的平方和、自然对数和、几何平均数、和的平方根。

7.5.1 指针作为函数参数

例7.19

```
1 #include <stdio.h>
2 #include <math.h>
3 double fun(double a, double b, double *sqab, double *lnab,
double *avg)
4 {
5     *sqab=a*a+b*b; /*sqab返回平方和
6     *lnab=log(a)+log(b); /*lnab返回自然对数和
7     *avg=(a+b)/2; /*avg返回几何平均数
8     return (sqrt(a+b)); //函数返回和的平方根
9 }
10 int main()
11 {
12     double x=10,y=12,fsq,fln,favg,fsqr;
13     fsqr=fun(x, y, &fsq, &fln, &favg);
```

7.5.1 指针作为函数参数

例7.19

```
14    printf("%lf,%lf,%lf,%lf,%lf,%lf\n",x,y,fsq,fln,favg,fsq  
r);  
15    return 0;  
16 }
```

程序运行结果如下:

10.000000,12.000000,244.000000,4.787492,11.000000,4.690416

7.5.1 指针作为函数参数

- ▶ 2. 数组作为函数形参
- ▶ 在前面介绍过（一维或多维）数组作为函数的形参，例如：

```
double average(double A[100], int n)
{
    ... //函数体
}
```

- ▶ 函数调用形式如下：

```
double X[100], f;
f = average(X, 100);
```

7.5.1 指针作为函数参数

- ▶ 由于实参数组名X表示该数组的首地址，因此形参应该是一个指针变量（只有指针变量才能存放地址），即函数定义

```
double average(double A[100], int n)
```

- ▶ 等价于

```
double average(double *A, int n)
```


7.5.1 指针作为函数参数

- ▶ 在函数调用开始，系统会建立一个指针变量A，用来存储从主调函数传来的实参数组首地址，则A指向数组X，其后可以通过指针访问数组元素。例如：
 - ▶ ①A[i]：指针下标法访问数组元素X[i]；
 - ▶ ②*(A+i)：指针引用法访问数组元素X[i]；
 - ▶ ③&A[i]、A+i：指向数组元素X[i]。

7.5.1 指针作为函数参数

- ▶ 从应用的角度看，形参数组从实参数组那里得到了首地址，因此形参数组与实参数组本质上是同一段内存单元。在被调函数中若修改了形参数组元素的值，也就是修改了实参数组元素。因此，用数组作为函数参数，也能够使函数返回多个运算结果。

7.5.1 指针作为函数参数

- ▶ 需要注意，形参数组“double A[100]”与数组定义写法一致，但含义不同。数组定义时A是数组名，是一个指针常量。而“double *A”是一个指针变量，在函数执行期间，它可以做赋值、自增自减运算等。例如：

```
void fun(double *A, double B[100])
{
    A++; //正确，A是指针变量
    B++; //错误，B是指针常量，不能做自增自减
    A=B; //正确，A是指针变量，可以重新指向B
    B=A; //错误，B是指针常量，不能被赋值
}
```

7.5.1 指针作为函数参数

- ▶ 当函数调用开始时，形参指针变量的值是实参传来的地址，如果在函数中修改了形参指针变量，则实参传来的地址就会“丢失”，一般要将此地址保存下来。

7.5.1 指针作为函数参数



【例7.20】

编写函数average，返回数组n个元素的平均值。

7.5.1 指针作为函数参数

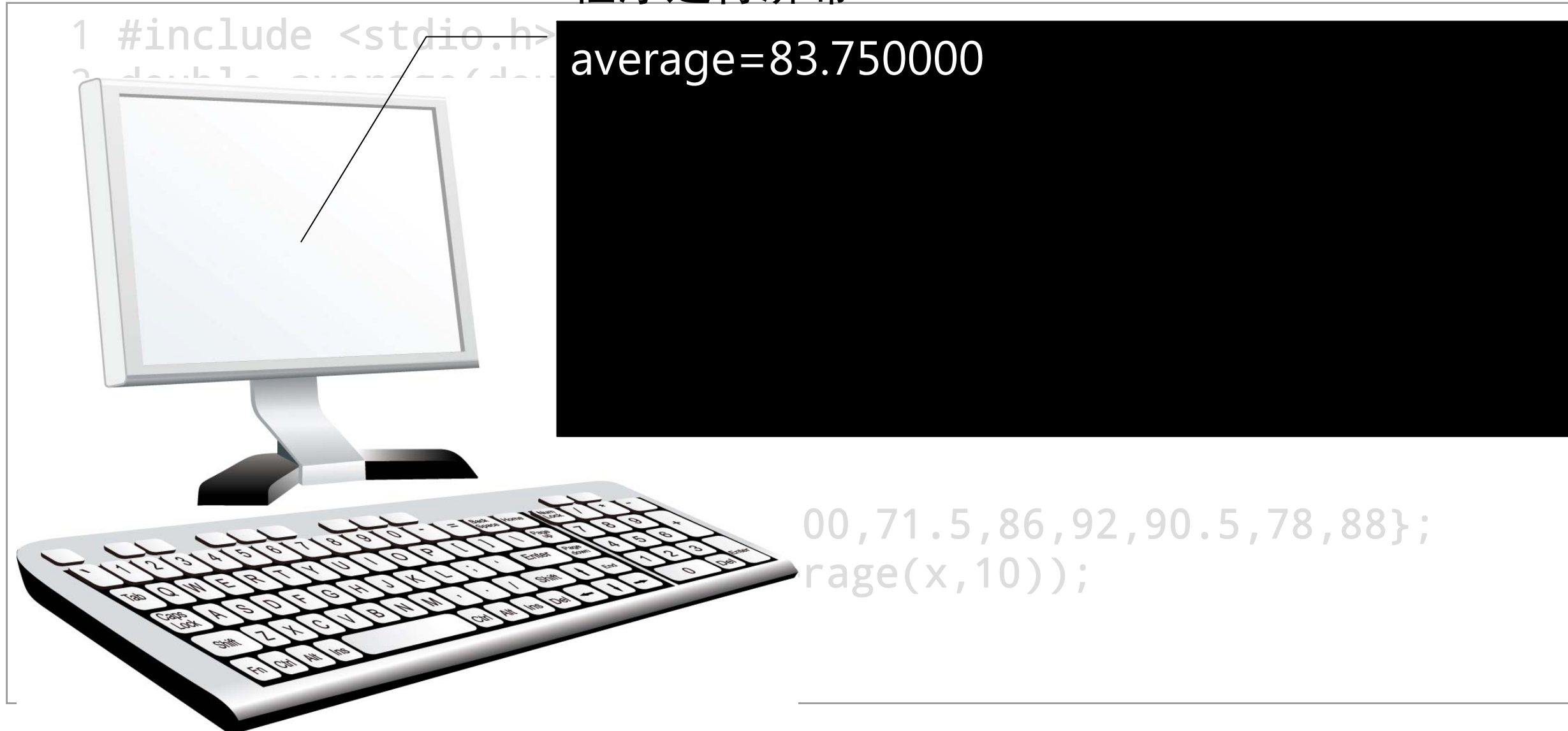
例7.20

```
1 #include <stdio.h>
2 double average(double *a, int n)
3 { //等价于average(double a[], int n)
4     double avg=0.0, *p=a;
5     int i;
6     for (i=1;i<=n;i++,p++) avg=avg+*p; //等价于avg=avg+p[i]
7     return n<=0 ? 0 : avg/n ;
8 }
9 int main()
10 {
11     double x[10]={66,76.5,89,100,71.5,86,92,90.5,78,88};
12     printf("average=%lf\n",average(x,10));
13     return 0;
14 }
```

7.5.1 指针作为函数参数

例7.20

程序运行屏幕



7.5.1 指针作为函数参数

- ▶ 综合前面指针变量和数组作为函数参数的结论，要想在函数中改变数组元素，实参与形参的对应关系有如下4种作用相同的情况。

7.5.1 指针作为函数参数

- ▶ (1) 形参和实参都用数组名，例如：

```
void fun(int x[100], int n); //函数原型  
int a[100];  
fun(a, 100); //函数调用
```

- ▶ 形参数组x接受了实参数组a的首地址，因此在函数调用期间，形参数组x与实参数组a是同一段内存单元。

7.5.1 指针作为函数参数

- ▶ (2) 形参用指针变量，实参用数组名，例如：

```
void fun(int *x, int n); //函数原型  
int a[100];  
fun(a, 100); //函数调用
```

- ▶ 形参指针变量x指向实参数组a。

7.5.1 指针作为函数参数

- ▶ (3) 形参与实参都用指针变量，例如：

```
void fun(int *x, int n); //函数原型  
int a[100], p=a;  
fun(p, 100); //函数调用
```

- ▶ 实参指针变量p的值传递到形参指针变量x中，则x也指向实参数组a。

7.5.1 指针作为函数参数

- ▶ (4) 形参用数组，实参用指针变量，例如：

```
void fun(int x[100], int n); //函数原型  
int a[100], p=a;  
fun(p, 100); //函数调用
```

- ▶ 形参数组x接受了实参指针变量传递进来的地址值，即数组a的首地址，因此可以理解形参数组x与实参数组a是同一段内存单元。

7.5.1 指针作为函数参数

- ▶ 上述4种情况，在函数fun中，均可以使用`x[i]`、`*(x+i)`、`*x++`等形式访问数组元素。

7.5.1 指针作为函数参数

- ▶ 无论形参是数组或是指针变量，在函数fun中无法检测到实参数组的实际长度。
- ▶ 实际编程中，要么象本例一样将实际长度传递到函数内部，要么象字符串那样放一个结束标志，在函数中只要检测到结束标志，就结束数组元素往前访问，避免数组越界。

7.5.1 指针作为函数参数

- ▶ 特别地，如果实参不是一个数组，例如：

```
void fun(int *x, int n); //函数原型  
int b, p=&b;  
fun(p, 100); //函数调用
```

- ▶ 在函数中如果把这个地址对应的内存单元当作数组来用，程序很容易出现严重错误。

CP 程序设计