

# 第8章 善于利用指针

**8.1 指针是什么**

**8.2 指针变量**

**8.3 通过指针引用数组**

**8.4 通过指针引用字符串**

**8.5 指向函数的指针**

**8.6 返回指针值的函数**

**8.7 指针数组和多重指针**

**8.8 动态内存分配与指向它的指针变量**

**8.9 有关指针的小结**

## 8.1 指针是什么

- 如果在程序中定义了一个变量，在对程序进行编译时，系统就会给该变量分配内存单元
- 编译系统根据程序中定义的变量类型，分配一定长度的空间
- 例如，**VC++**为整型变量分配**4**个字节，对单精度浮点型变量分配 4 个字节，对字符型变量分配 1 个字节



## 8.1 指针是什么

- 内存区的每一个字节有一个编号，这就是“地址”，它相当于旅馆中的房间号。
- 在地址所标识的内存单元中存放数据，这相当于旅馆房间中居住的旅客一样。
- 由于通过地址能找到所需的变量单元，我们可以说，地址指向该变量单元。
- 将地址形象化地称为“指针”



➤ 务必弄清楚存储单元的**地址**和存储单元的**内容**这两个概念的区别

例如：



```
int i=3,j=6,k;
```

```
printf("%d",i);
```

通过变量名*i*

找到*i*的地址  
2000，从而从存储单元读取3

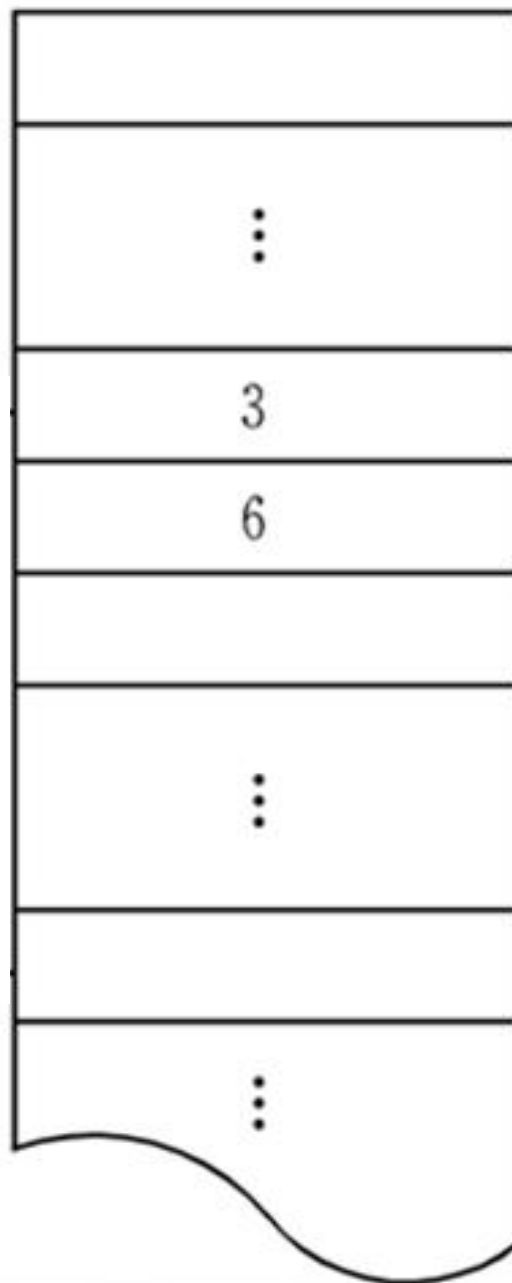
2000

2004

2008

3020

内存用户数据区

变量*i*变量*j*变量*k*变量*i\_pointer*

```
int i=3,j=6,k;
```

```
k=i+j;
```

从这里取3

从这里取6

将9送到这里

直接存取

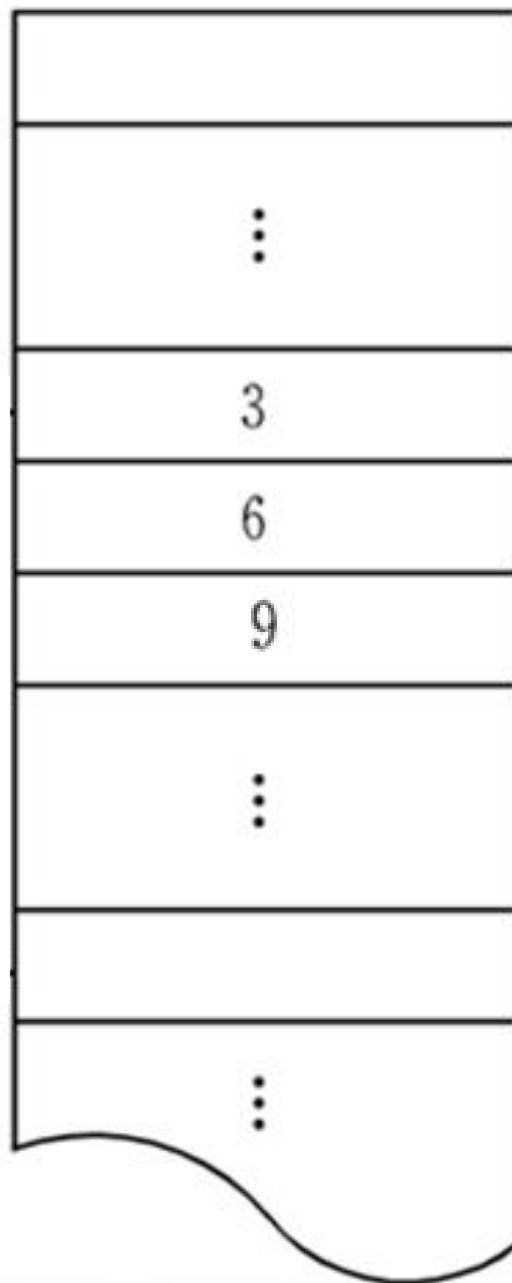
2000

2004

2008

3020

内存用户数据区



```
int i=3,j=6,k;
```

定义特殊变量 `i_pointer`

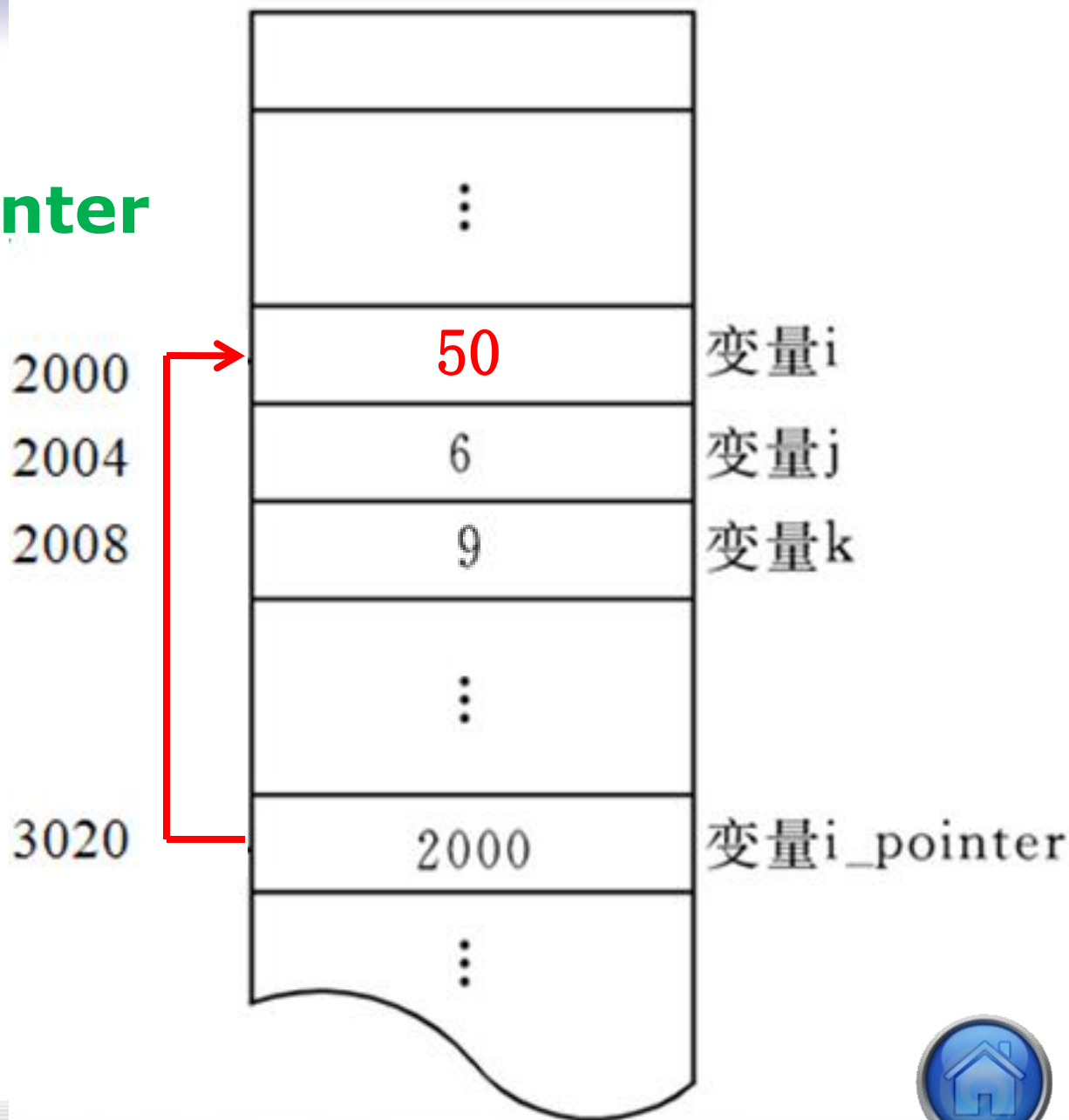
```
i_pointer=&i;
```

```
*i_pointer=50;
```

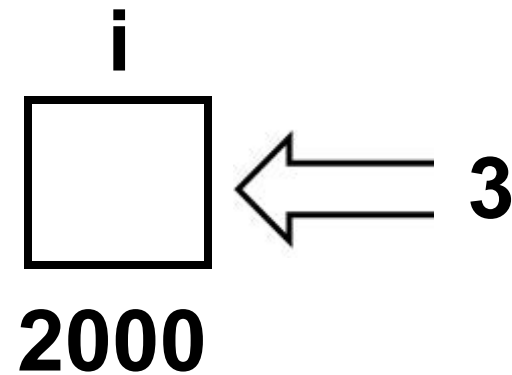
将i的地址  
存到这里

间接存取

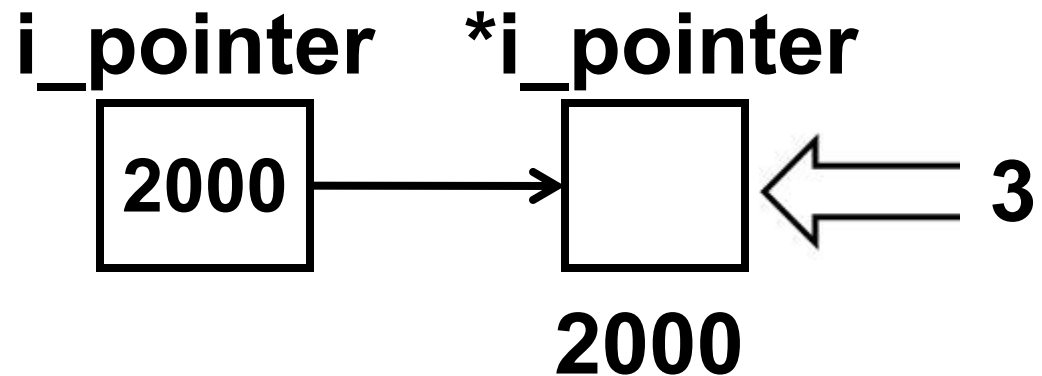
内存用户数据区



直接存取



间接存取



➤ 为了表示将数值 3 送到变量中，可以有两种表达方法：

**(1) 将3直接送到变量*i*所标识的单元中，例如：*i*=3;**

**(2) 将3送到变量*i\_pointer*所指向的单元（即变量*i*的存储单元），例如：**

***\*i\_pointer*=3;** 其中*\*i\_pointer*表示*i\_pointer*指向的对象



➤ 指向就是通过地址来体现的

◆ 假设 **i\_pointer** 中的值是变量 **i** 的地址  
(**2000**)，这样就在 **i\_pointer** 和变量 **i** 之间  
建立起一种联系，即通过 **i\_pointer** 能知道 **i**  
的地址，从而找到变量 **i** 的内存单元



- 由于通过地址能找到所需的变量单元，因此说，地址指向该变量单元
- 将地址形象化地称为“指针”。意思是通过它能找到以它为地址的内存单元



➤ 一个变量的地址称为该变量的“指针”

例如，地址**2000**是变量 *i* 的指针

➤ 如果有一个变量专门用来存放另一变量的地址（即指针），则它称为“指针变量”

➤ **i\_pointer**就是一个指针变量。指针变量就是地址变量，用来存放地址的变量，指针变量的值是地址（即指针）



- “**指针**”和“**指针变量**”是**不同的概念**
- 可以说变量*i*的指针是**2000**，而不能说*i*的指针变量是**2000**
- 指针是一个地址，而指针变量是存放地址的变量



## 8.2 指针变量

8.2.1 使用指针变量的例子

8.2.2 怎样定义指针变量

8.2.3 怎样引用指针变量

8.2.4 指针变量作为函数参数



## 8.2.1 使用指针变量的例子

**例8.1** 通过指针变量访问整型变量。

➤ **解题思路：**先定义**2**个整型变量，再定义**2**个指针变量，分别指向这两个整型变量，通过访问指针变量，可以找到它们所指向的变量，从而得到这些变量的值。



```
a=100,b=10
```

```
*pointer_1=100,*pointer_2=10
```

```
#include <stdio.h>
```

```
int main()
```

```
{ int a=100,b=10;
```

```
  int *pointer_1, *pointer_2;
```

```
  pointer_1=&a;
```

```
  pointer_2=&b;
```

```
  printf("a=%d,b=%d\n",a,b);
```

```
  printf("*pointer_1=%d,*pointer_2=
        %d\n",*pointer_1,*pointer_2);
```

```
  return 0;
```

```
}
```

定义两个指针变量

直接输出变量a和b的值

间接输出变量a和b的值



```
#include <stdio.h>
```

```
int main()
```

```
{ int a=100,b=10;
```

```
int*pointer_1,*pointer_2;
```

```
pointer_1=&a;
```

```
pointer_2=&b;
```

```
printf("a=%d,b=%d\n",a,b);
```

```
printf("*pointer_1=%d,*pointer_2=
%d\n",*pointer_1,*pointer_2);
```

```
return 0;
```

```
}
```

此处\*与类型名在一起。  
此时共同定义指针变量

此处\*与指针变量一起使用。此  
时代表指针变量所指向的变量



## 8.2.2 怎样定义指针变量

➤ 定义指针变量的一般形式为：

类型 \* 指针变量名；

如： **int \*pointer\_1, \*pointer\_2;**

◆ **int** 是为指针变量指定的“**基类型**”

◆ 基类型指定指针变量可指向的变量类型

◆ 如 **pointer\_1** 可以指向整型变量，但不能指向浮点型变量



## 8.2.2 怎样定义指针变量

➤ 下面都是合法的定义和初始化:

```
float *pointer_3;
```

```
char *pointer_4;
```

```
int a,b;
```

```
int *pointer_1=&a,*pointer_2=&b;
```

```
pointer_3=2000; pointer_3=&a; 错误
```



## 8.2.3 怎样引用指针变量

使p指向a

➤ 在引用指针变量时，可能有三种情况：

◆ 给指针变量 **\*p** 相当于 **a** **= &a;**

◆ 引用指针变量指向的变量。如有

**p = &a; \*p = 1;**

则执行 **printf("%d", \*p);** 将输出 **1**

◆ 引用指针变量的值。如：**printf("%o", p);**

以八进制输出a的地址



## 8.2.3 怎样引用指针变量

➤ 要熟练掌握两个有关的运算符： ■

(1) & 取地址运算符。

**&a**是变量**a**的地址

(2) \* 指针运算符（“间接访问”运算符）

如果：**p**指向变量**a**，则**\*p**就代表**a**。

**k = \*p;**      (把**a**的值赋给**k**)

**\*p = 1;**      (把**1**赋给**a**)

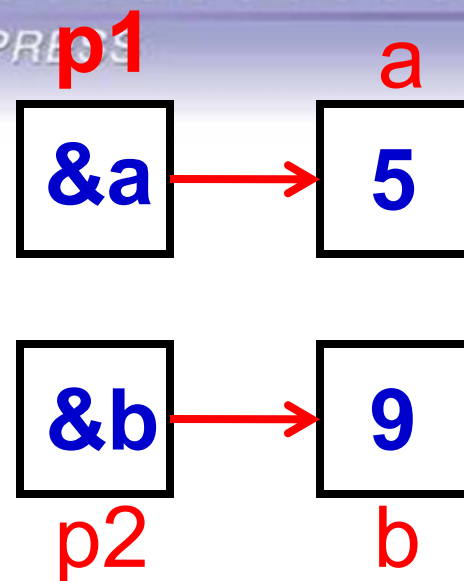
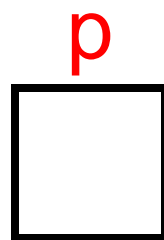


**例8.2** 输入**a**和**b**两个整数，按先大后小的顺序输出**a**和**b**。

- 解题思路：用指针方法来处理这个问题。  
不交换整型变量的值，而是交换两个指针变量的值。



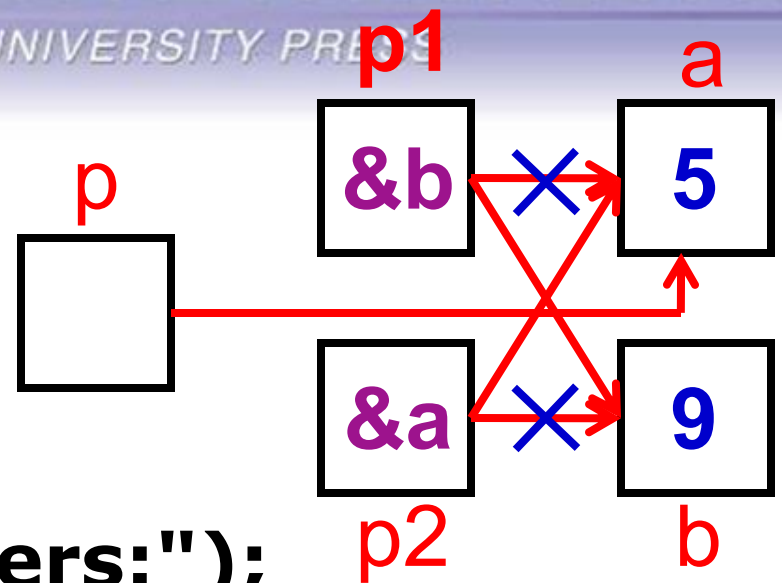
```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);
  p1=&a; p2=&b;
  if(a<b) 成立
  { p=p1; p1=p2; p2=p; }
  printf("a=%d,b=%d\n",a,b);
  printf("%d,%d\n",*p1,*p2);
  return 0;
}
```



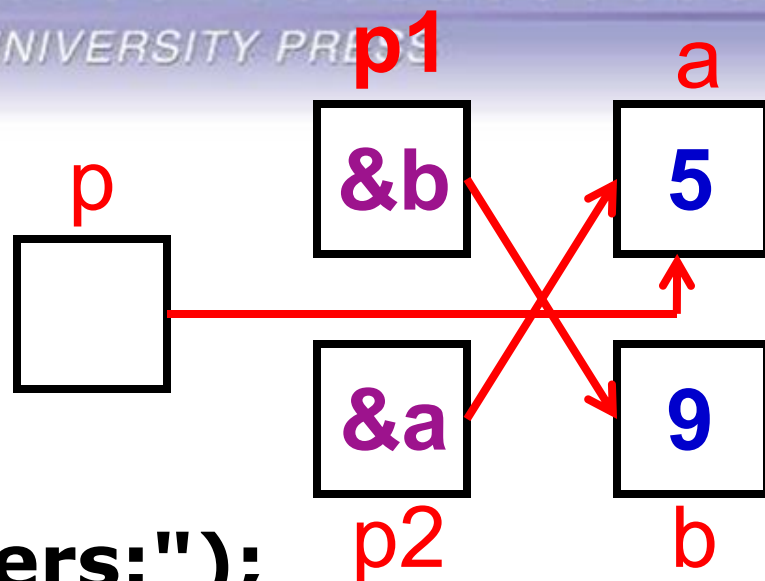
5,9



```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);
  p1=&a;  p2=&b;
  if(a<b)
  { p=p1; p1=p2; p2=p; }
  printf("a=%d,b=%d\n",a,b);
  printf("%d,%d\n",*p1,*p2);
  return 0;
}
```



```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);
  p1=&a;  p2=&b;
  if(a<b)
  { p=p1; p1=p2; p2=p; }
  printf("a=%d,b=%d\n",a,b);
  printf("%d,%d\n",*p1,*p2);
  return 0;
}
```



a=5, b=9  
9, 5



```
#include <stdio.h>
```

```
int main()
```

```
{ int *p1,*p2,*p,a,b;
```

```
    printf("integer numbers:");
```

```
    scanf("%d,%d",&a,&b);
```

```
    p1=&a;    p2=&b;
```

```
    if(a<b)
```

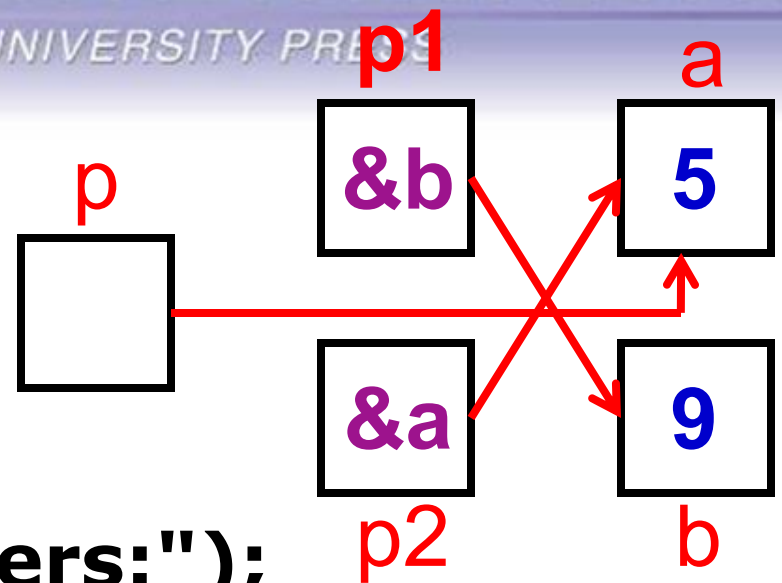
```
    { p=p1; p1=p2; p2=p; }
```

```
    printf("a=%d,b=%d\n",a,b);
```

```
    printf("%d,%d\n",*p1,*p2);
```

```
    return 0;
```

```
}
```



可否改为  $p1 = \&b;$   $p2 = \&a;$  ?



## ➤ 注意:

- ◆ **a**和**b**的值并未交换，它们仍保持原值
- ◆ 但**p1**和**p2**的值改变了。**p1**的值原为**&a**，后来变成**&b**，**p2**原值为**&b**，后来变成**&a**
- ◆ 这样在输出**\*p1**和**\*p2**时，实际上是输出变量**b**和**a**的值，所以先输出**9**，然后输出**5**



## 8.2.4 指针变量作为函数参数

例**8.3** 题目要求同例**8.2**，即对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。

- 解题思路：定义一个函数**swap**，将指向两个整型变量的指针变量作为实参传递给**swap**函数的形参指针变量，在函数中通过指针实现交换两个变量的值。



```
#include <stdio.h>
```

```
int main()
```

```
{void swap(int *p1,int *p2);
```

```
int a,b; int*pointer_1,*pointer_2;
```

```
printf("please enter a and b:");
```

```
scanf("%d,%d",&a,&b);
```

5,9

```
pointer_1=&a;
```

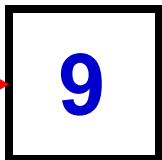
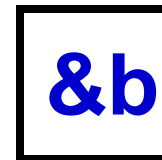
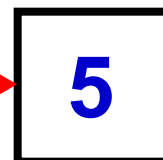
pointer\_1

a

pointer\_2

b

```
pointer_2=&b;
```



```
if (a<b) swap(pointer_1,pointer_2);
```

```
printf("max=%d,min=%d\n",a,b);
```

```
return 0;
```

```
}
```



```
void swap(int *p1,int *p2)
```

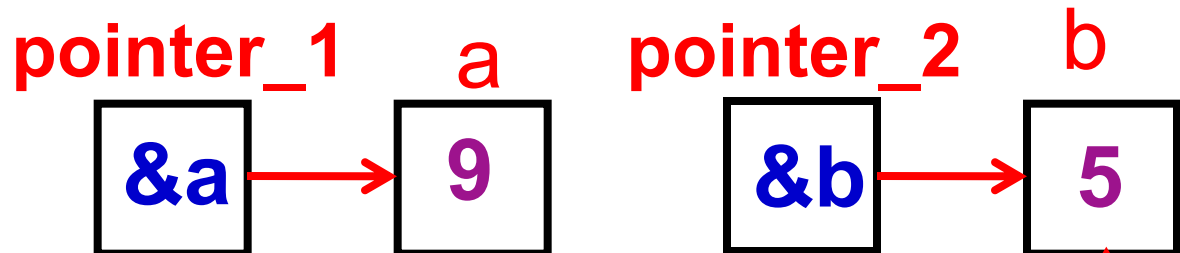
```
{ int temp;
```

```
    temp=*p1;
```

```
    *p1=*p2;
```

```
    *p2=temp;
```

```
}
```



```
please enter a and b:5,9
max=9,min=5
```



```
void swap(int *p1,int *p2)
```

```
{ int temp;
```

```
temp=*p1;
```

```
*p1=*p2;
```

```
*p2=temp;
```

```
}
```

**错!!!**  
无确定的指向

```
void swap(int *p1,int *p2)
```

```
{ int *temp;
```

```
*temp=*p1;
```

```
*p1=*p2;
```

```
*p2=*temp;
```

```
}
```



```
#include <stdio.h>
```

```
int main()
```

```
{.....
```

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

```
    printf("max=%d,min=%d\n",a,b);
```

```
    return 0;
```

```
}
```

```
void swap(int x,int y)
```

```
{ int temp;
```

```
    temp=x; x=y; y=temp;
```

```
}
```

错!!!  
无法交换a,b

a

5

b

9



- 如果想通过函数调用得到  $n$  个要改变的值：
- ① 在主调函数中设  $n$  个变量，用  $n$  个指针变量指向它们 ■
  - ② 设计一个函数，有  $n$  个指针形参。在这个函数中改变这  $n$  个形参的值
  - ③ 在主调函数中调用这个函数，在调用时将这  $n$  个指针变量作实参，将它们的地址传给该函数的形参 ■
  - ④ 在执行该函数的过程中，通过形参指针变量，改变它们所指向的  $n$  个变量的值 ■
  - ⑤ 主调函数中就可以使用这些改变了值的变量



例**8.4** 对输入的两个整数按大小顺序输出。

➤ 解题思路：尝试调用**swap**函数来实现题目要求。在函数中改变形参(指针变量)的值，希望能由此改变实参(指针变量)的值



```
#include <stdio.h>
```

```
int main()
```

```
{void swap(int *p1,int *p2);
```

```
int a,b; int*pointer_1,*pointer_2;
```

```
scanf("%d,%d",&a,&b);
```

5,9

```
pointer_1=&a; pointer_2=&b;
```

```
if (a<b) swap(pointer_1,pointer_2);
```

```
printf("max=%d,min=%d\n",a,b);
```

```
return 0;
```

max=5,min=9

```
} void swap(int *p1,int *p2)
```

```
{ int *p;
```

```
    p=p1; p1=p2; p2=p;
```

```
}
```

错!!!

只交换形参指向



- 注意：函数的调用可以（而且只可以）得到一个返回值（即函数值），而使用指针变量作参数，可以得到多个变化了的值。如果不用指针变量是难以做到这一点的。
- 要善于利用指针法。



**例8.5** 输入**3**个整数**a,b,c**，要求按由大到小的顺序将它们输出。用函数实现。

- **解题思路：**采用**例8.3**的方法在函数中改变这**3**个变量的值。用**swap**函数交换两个变量的值，用**exchange**函数改变这**3**个变量的值。



```
#include <stdio.h>
```

```
int main()
```

```
{ void exchange(int *q1, int *q2, int *q3);
```

```
  int a,b,c,*p1,*p2,*p3;
```

```
  scanf("%d,%d,%d",&a,&b,&c);
```

20,-54,67

```
  p1=&a;p2=&b;p3=&c;
```

```
  exchange(p1,p2,p3);
```

调用结束后不会  
改变指针的指向

```
  printf("%d,%d,%d\n",a,b,c);
```

```
  return 0;
```

```
}
```



```
void exchange(int *q1, int *q2, int *q3)
{ void swap(int *pt1, int *pt2);
  if(*q1<*q2) swap(q1,q2);
  if(*q1<*q3) swap(q1,q3);
  if(*q2<*q3) swap(q2,q3);
}
```

20, -54, 67  
67, 20, -54

```
void swap(int *pt1, int *pt2)
{ int temp;
  temp=*pt1; *pt1=*pt2; *pt2=temp;
}
```

交换指针指向的变量值



## 8.3通过指针引用数组

8.3.1 数组元素的指针

8.3.2 在引用数组元素时指针的运算

8.3.3 通过指针引用数组元素

8.3.4 用数组名作函数参数

8.3.5 通过指针引用多维数组



## 8.3.1 数组元素的指针

- 一个变量有地址，一个数组包含若干元素，每个数组元素都有相应的地址
- 指针变量可以指向数组元素（把某一元素的地址放到一个指针变量中）
- 所谓数组元素的指针就是数组元素的地址



- ```
int a[10]={1,3,5,7,9,11,13,15,17,19};
```

**p=&a[0];**

P

 $\& a[0]$ 

1

a[0]

3

5

7

9

11

13

15

17

19

注意：数组名**a**不代表整个数组，只代表数组首元素的地址。“**p=a;**”的作用是“把**a**数组的首元素的地址赋给指针变量**p**”，而不是“把数组**a**各元素的值赋给**p**”。



a [9]

## 8.3.2 在引用数组元素时指针的运算

- 在指针指向数组元素时，允许以下运算：
  - ◆ 加一个整数(用 $+$ 或 $+=$ )，如 $p+1$
  - ◆ 减一个整数(用 $-$ 或 $-=$ )，如 $p-1$
  - ◆ 自加运算，如 $p++$ ， $++p$
  - ◆ 自减运算，如 $p--$ ， $--p$
  - ◆ 两个指针相减，如 $p1-p2$ （只有 $p1$ 和 $p2$ 都指向同一数组中的元素时才有意义）



**(1)** 如果指针变量**p**已指向数组中的一个元素，则**p+1**指向同一数组中的下一个元素，**p-1**指向同一数组中的上一个元素。

**float a[10], \*p=a;**

假设**a[0]**的地址为**2000**，则

◆**p**的值为**2000**

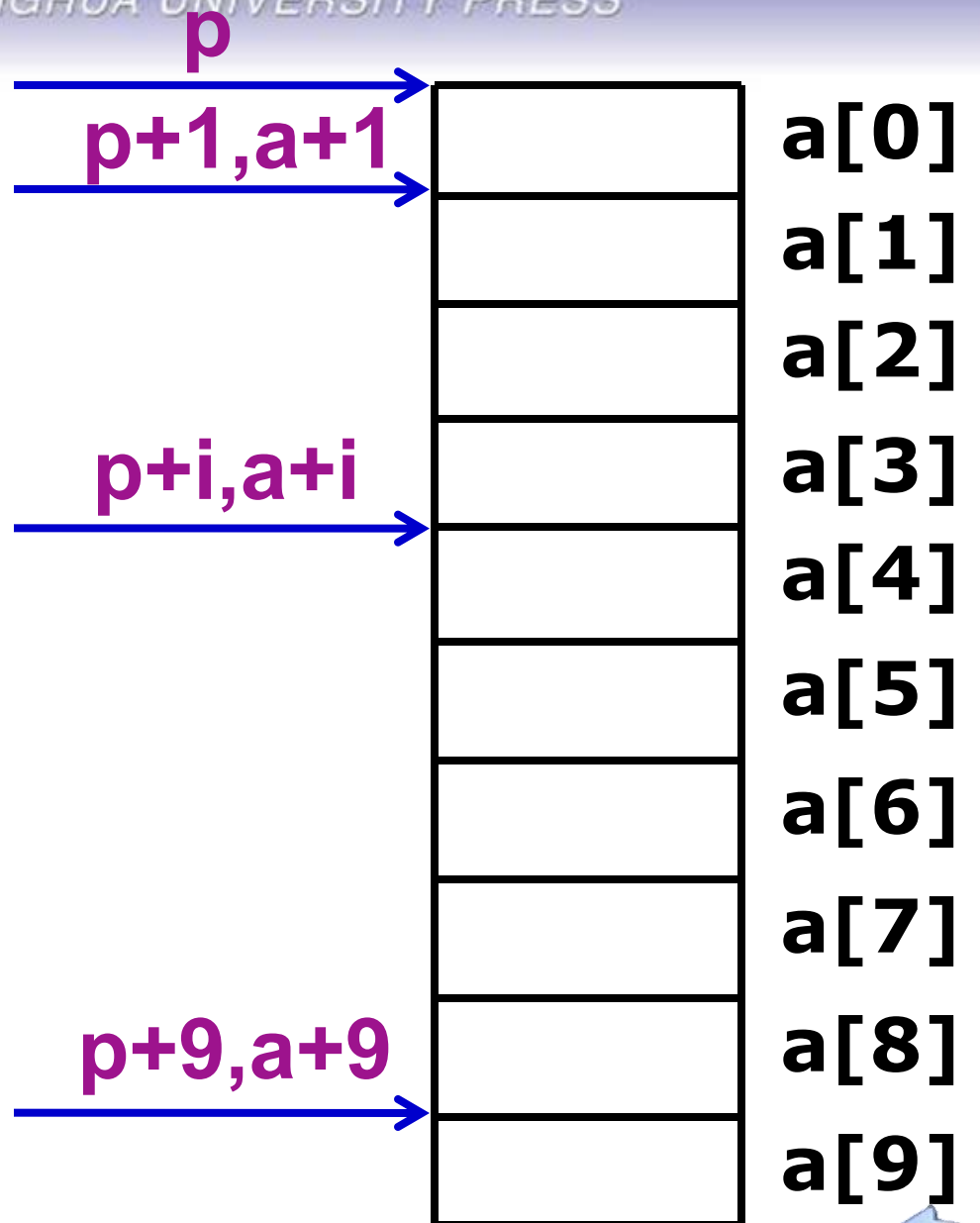
◆**p+1**的值为**2004**

◆**p-1**的值为**1996**

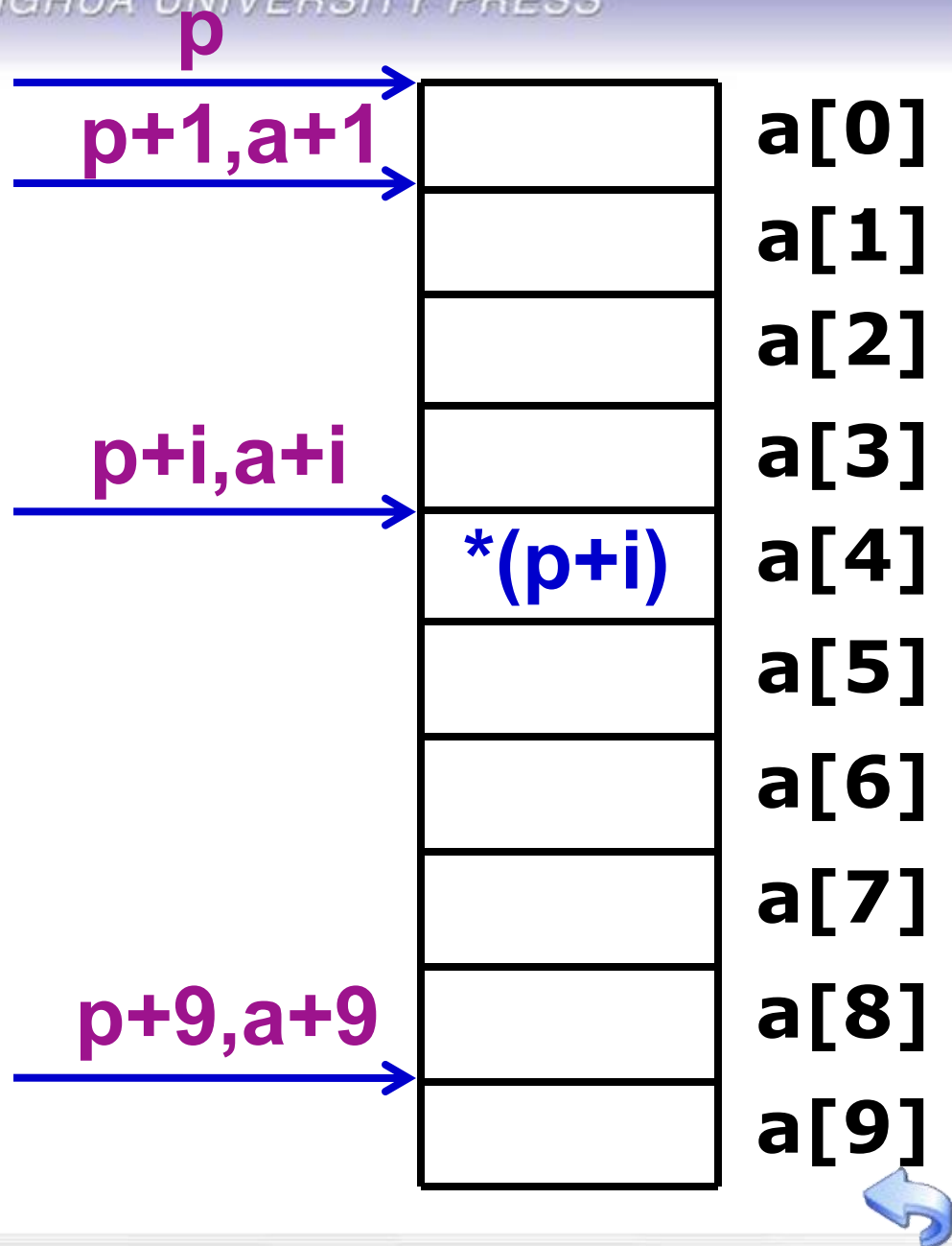
越界



(2) 如果  $p$  的初值为  $\&a[0]$ , 则  $p+i$  和  $a+i$  就是数组元素  $a[i]$  的地址, 或者说, 它们指向  $a$  数组序号为  $i$  的元素



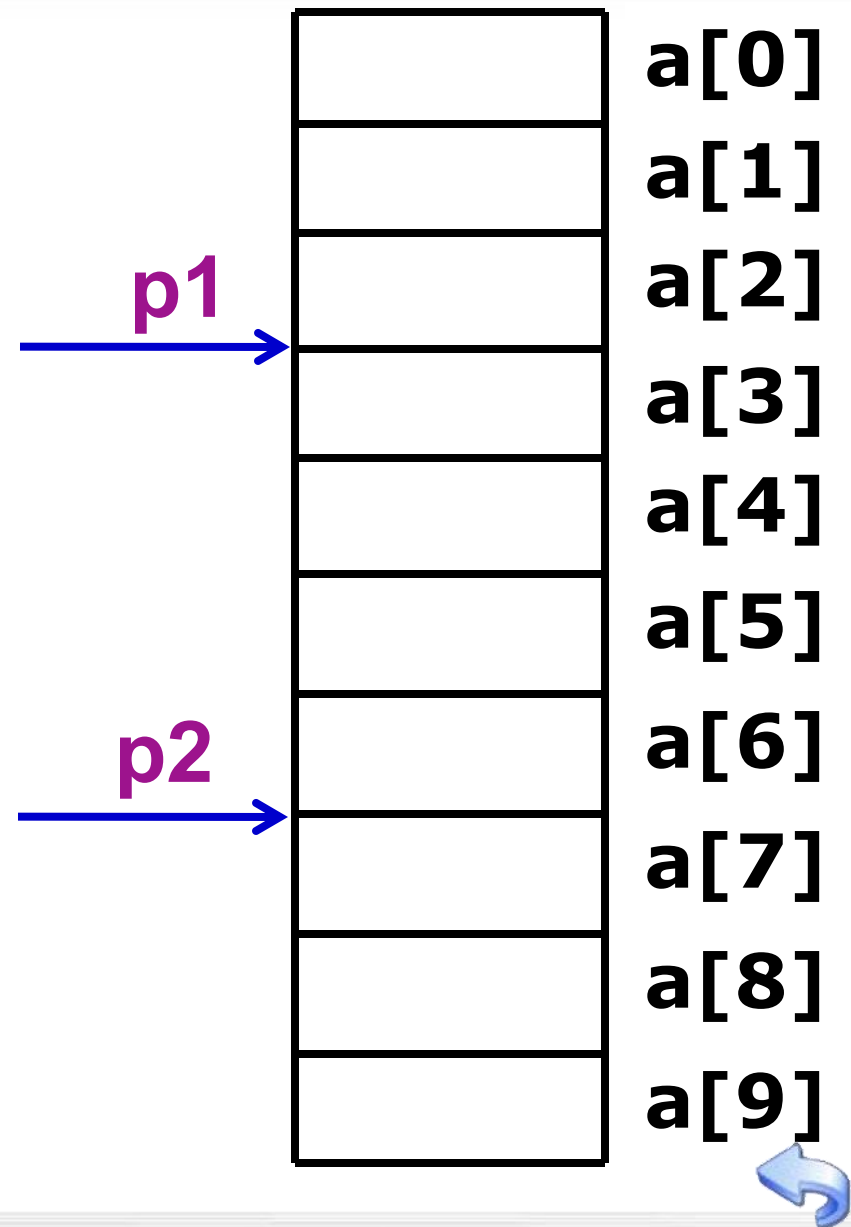
(3)  $*(p+i)$ 或  
 $*(a+i)$ 是 $p+i$   
或 $a+i$ 所指向  
的数组元素，  
即 $a[i]$ 。



(4) 如果指针**p1**和**p2**  
都指向同一数组

**p2-p1**的值是4

不能**p1+p2**



## 8.3.3 通过指针引用数组元素

➤ 引用一个数组元素，可用下面两种方法：

(1) 下标法，如  **$a[i]$**  形式

(2) 指针法，如  **$*(a+i)$**  或  **$*(p+i)$**

其中  **$a$**  是数组名， **$p$**  是指向数组元素的指针变量，其初值  **$p=a$**



## 8.3.3 通过指针引用数组元素

**例8.6** 有一个整型数组**a**，有**10**个元素，要求输出数组中的全部元素。

- 解题思路：引用数组中各元素的值有**3**种方法：**(1)**下标法；**(2)**通过数组名计算数组元素地址，找出元素的值；**(3)**用指针变量指向数组元素
- 分别写出程序，以资比较分析。



**(1) 下标法。**

```
#include <stdio.h>
```

```
int main()
```

```
{ int a[10]; int i;
```

```
    printf("enter 10 integer numbers:\n");
```

```
    for(i=0;i<10;i++) scanf("%d",&a[i]);
```

```
    for(i=0;i<10;i++) printf("%d ",a[i]);
```

```
    printf("%\n");
```

```
    return 0;
```

```
}
```

```
enter 10 integer numbers:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```



**(2) 通过数组名计算数组元素地址，找出元素的值**

```
#include <stdio.h>
```

```
int main()
```

```
{ int a[10]; int i;
```

```
    printf("enter 10 integer numbers:\n");
```

```
    for(i=0;i<10;i++) scanf("%d",&a[i]);
```

```
    for(i=0;i<10;i++)
```

```
        printf("%d ",*(a+i));
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

**scanf("%d",a+i);**



### (3) 用指针变量指向数组元素

```
#include <stdio.h>
```

```
int main()
```

```
{ int a[10]; int *p,i;
```

```
    printf("enter 10 integer numbers:\n");
```

```
    for(i=0;i<10;i++) scanf("%d",&a[i]);
```

```
    for(p=a;p<(a+10);p++)
```

```
        printf("%d ",*p);
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

for(p=a;p<(a+10);p++)

for(p=a;p<(a+10);a++)

printf("%d ",\*a); 错!



## ➤ 3种方法的比较:

### ① 第(1)和第(2)种方法执行效率相同

◆ C编译系统是将 $a[i]$ 转换为 $*(a+i)$ 处理的, 即先计算元素地址。

◆ 因此用第(1)和第(2)种方法找数组元素费时较多。



## ➤ 3种方法的比较:

② 第**(3)**种方法比第**(1)**、第**(2)**种方法快

◆ 用指针变量直接指向元素，不必每次都重新计算地址，像 **$p++$** 这样的自加操作是比较快的

◆ 这种有规律地改变地址值( **$p++$** )能大大提高执行效率



## ➤ 3种方法的比较:

③ 用下标法比较直观，能直接知道是第几个元素。

用地址法或指针变量的方法不直观，难以很快地判断出当前处理的是哪一个元素。



**例8.7** 通过指针变量输出整型数组**a**的**10**个元素。

➤ 解题思路：

用指针变量**p**指向数组元素，通过改变指针变量的值，使**p**先后指向**a[0]**到**a[9]**各元素。



```
#include <stdio.h>
```

```
int main()
```

```
{ int *p,i,a[10];
```

```
    p=a;
```

```
    printf("enter 10 integer numbers:\n");
```

```
    for(i=0;i<10;i++) scanf("%d",p++);
```

```
    for(i=0;i<10;i++,p++)
```

```
        printf("%d ",*p);
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

重新执行  
**p=a;**

退出循环时**p**指向**a[9]**  
后面的存储单元

因此执行此  
循环出问题



## 8.3.4 用数组名作函数参数

- 用数组名作函数参数时，因为实参数组名代表该数组首元素的地址，形参应该是一个指针变量
- **C**编译都是将形参数组名作为指针变量来处理的

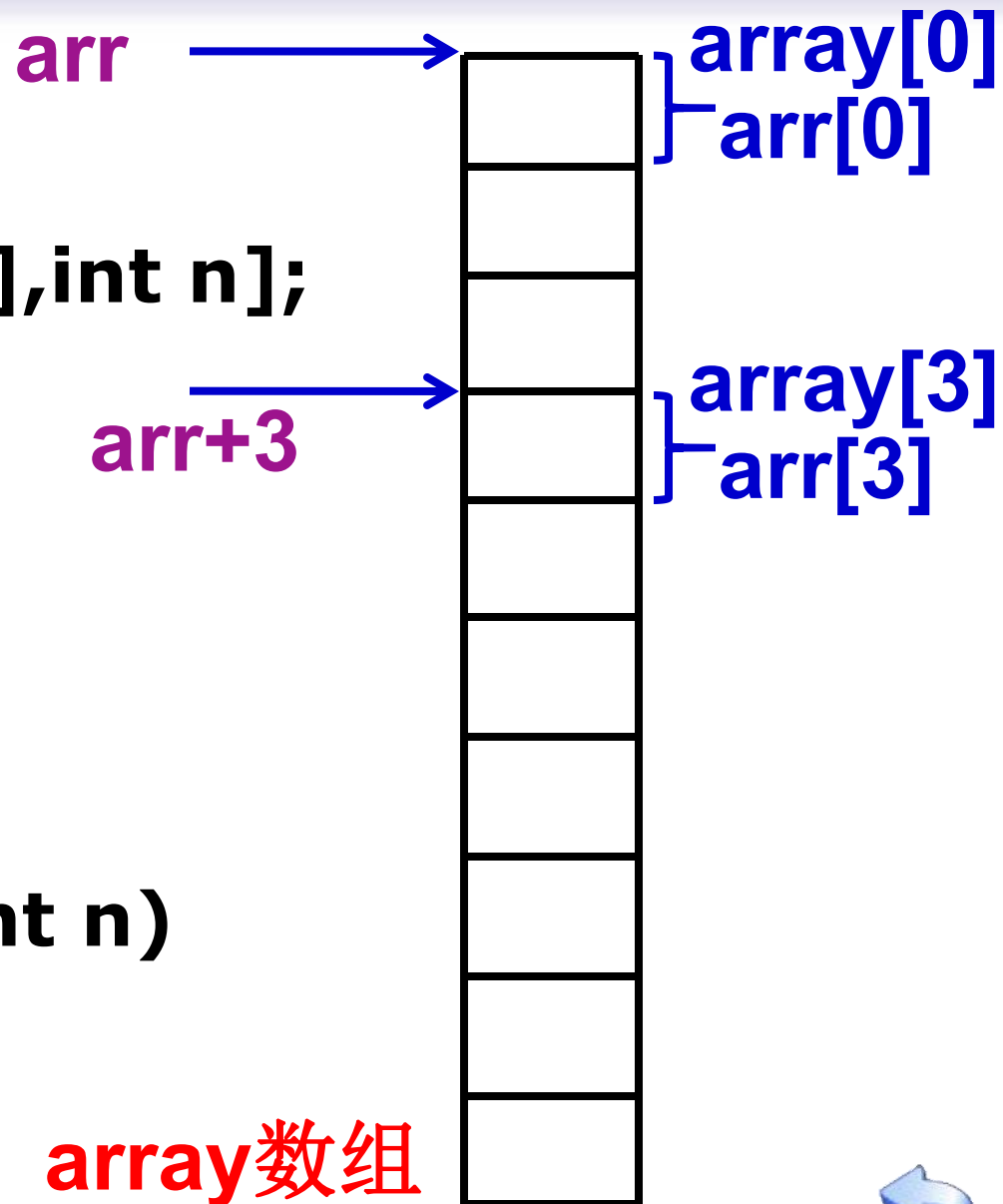


```
int main()  
{ void fun(int arr[],int n);  
  int array[10];    :  
  :  
  fun (array,10);  
  return 0;  
}  
void fun(int arr[ ],int n)  
{ : }
```

**fun(int \*arr,int n)**



```
int main()
{ void fun(int arr[],int n);
  int array[10];
  :
  fun (array,10);
  return 0;
}
void fun(int *arr,int n)
{ : }
```



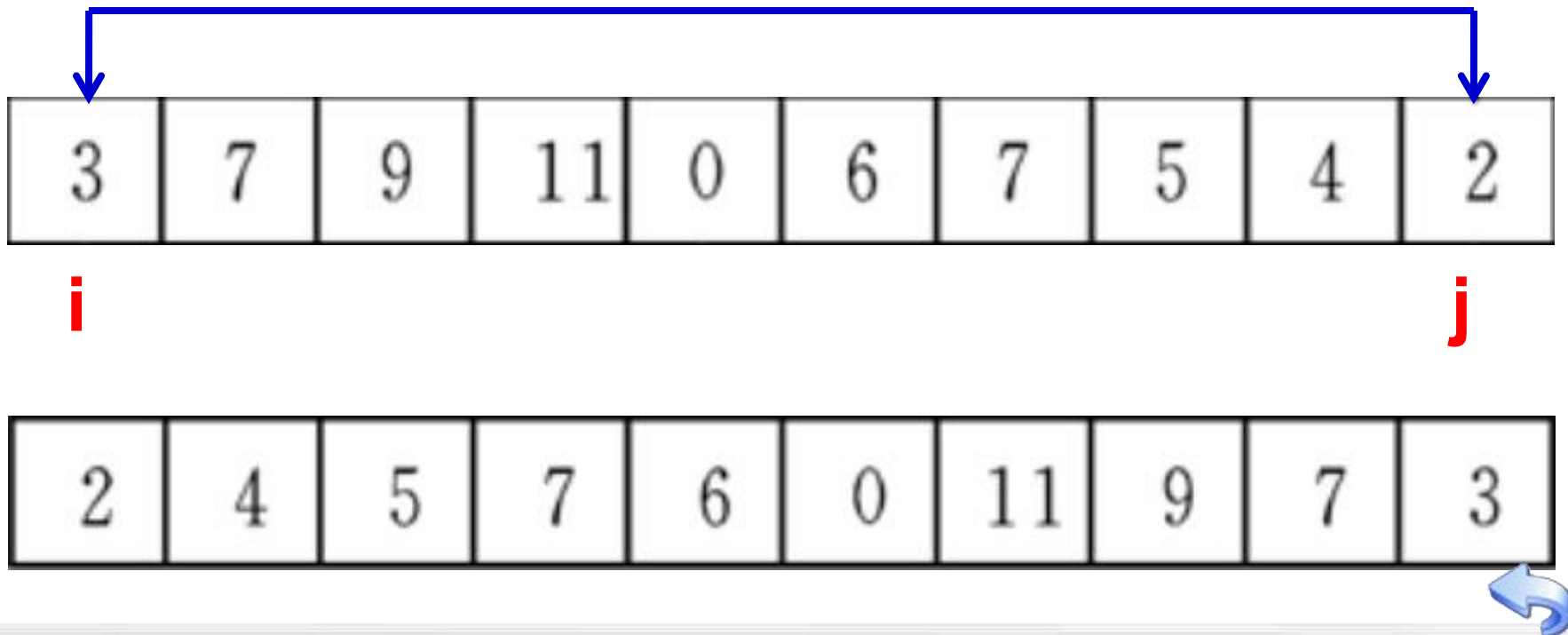
- 实参数组名是指针常量，但形参数组名是按指针变量处理
- 在函数调用进行虚实结合后，它的值就是实参数组首元素的地址
- 在函数执行期间，形参数组可以再被赋值

```
void fun (arr[ ],int n)  
{ printf("%d\n", *arr);  
  arr=arr+3;  
  printf("%d\n", *arr);  
}
```



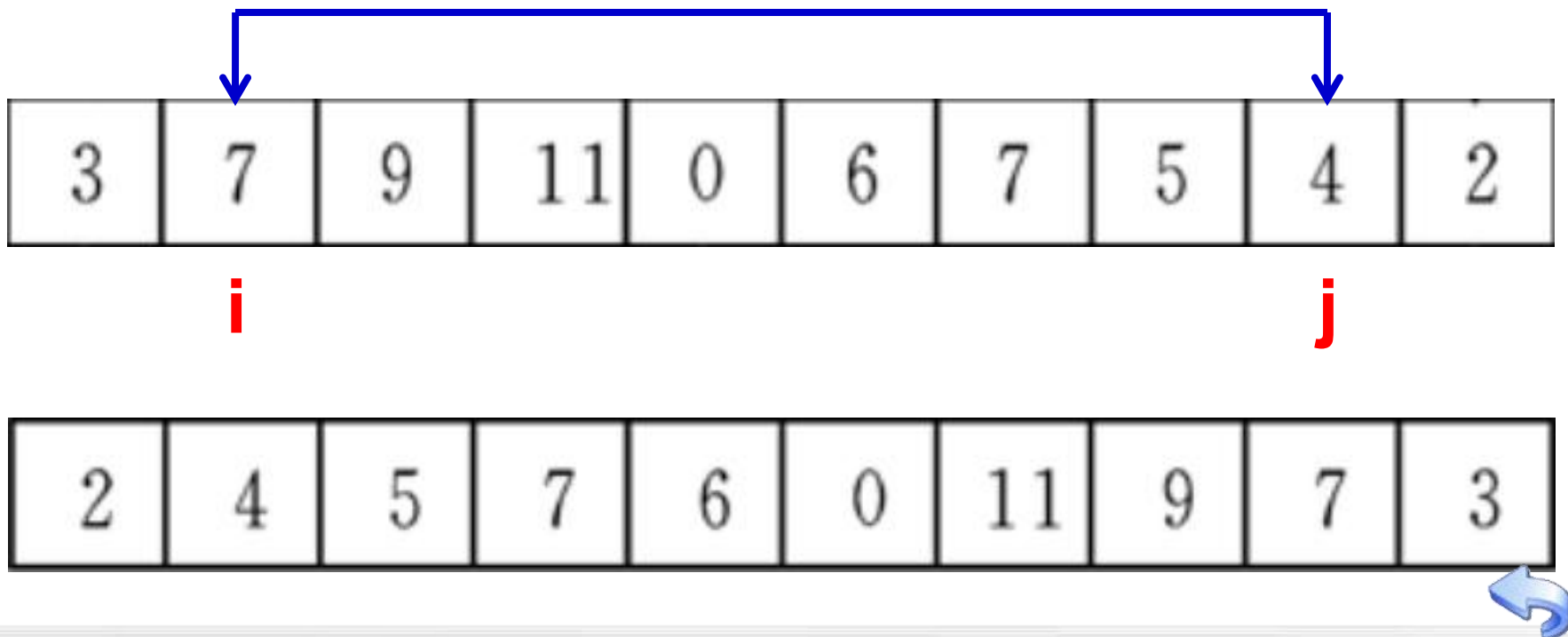
**例8.8** 将数组**a**中**n**个整数按相反顺序存放

➤ 解题思路：将**a[0]**与**a[n-1]**对换，.....  
将**a[4]**与**a[5]**对换。



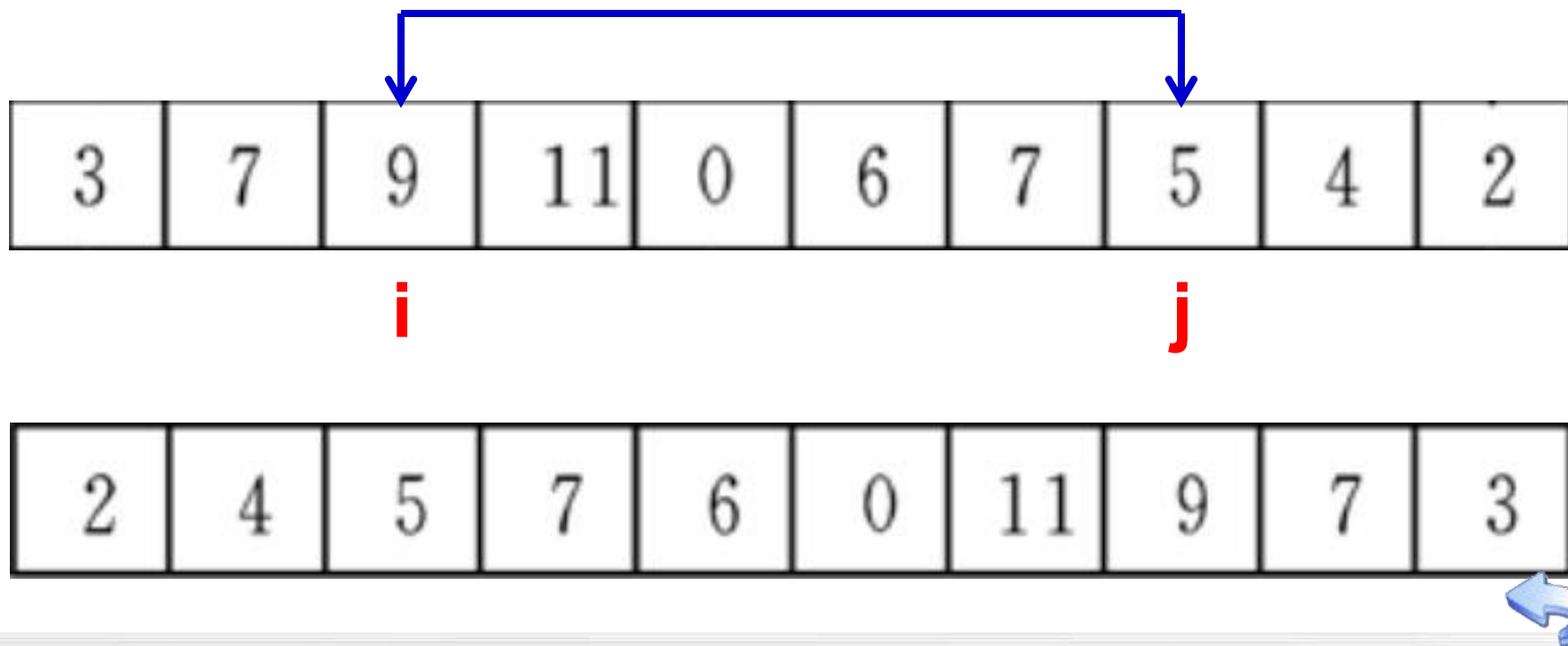
**例8.8** 将数组**a**中**n**个整数按相反顺序存放

➤ 解题思路：将**a[0]**与**a[n-1]**对换，.....  
将**a[4]**与**a[5]**对换。



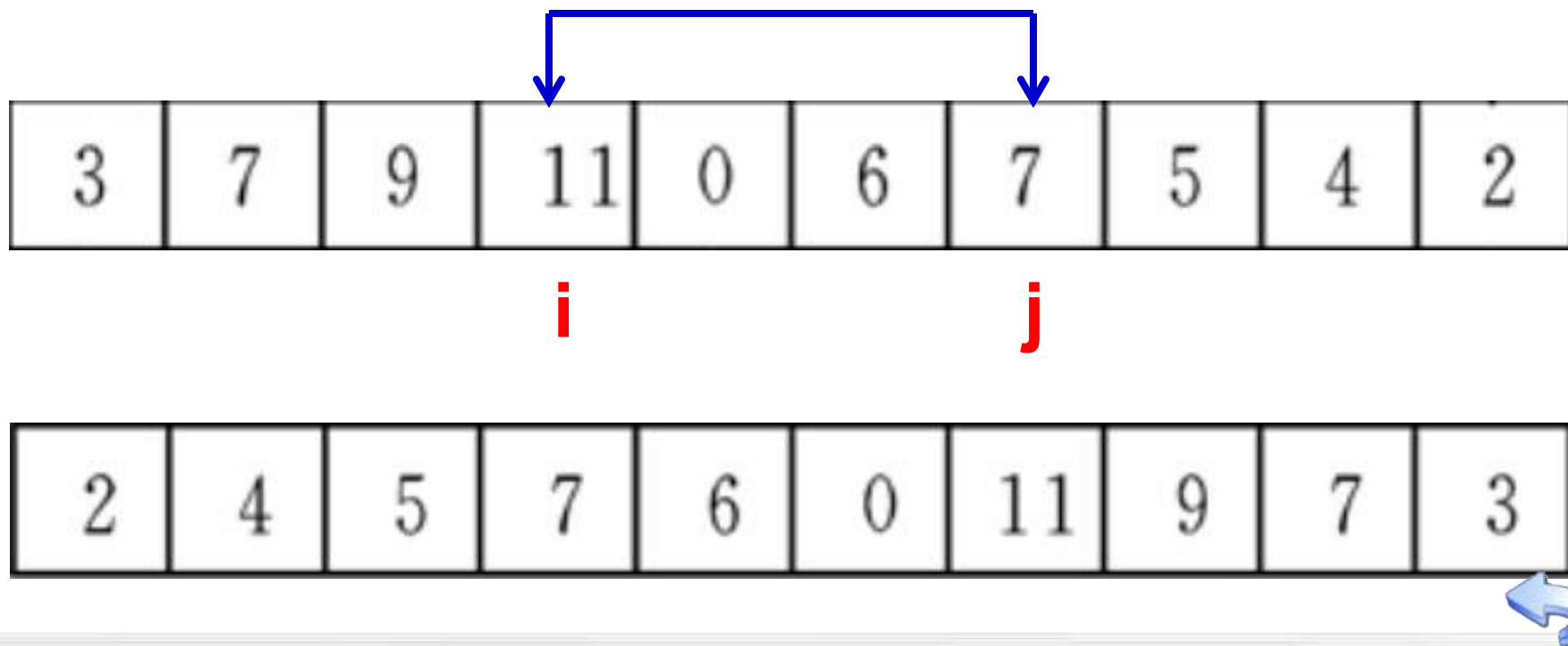
**例8.8** 将数组**a**中**n**个整数按相反顺序存放

➤ 解题思路：将**a[0]**与**a[n-1]**对换，.....  
将**a[4]**与**a[5]**对换。



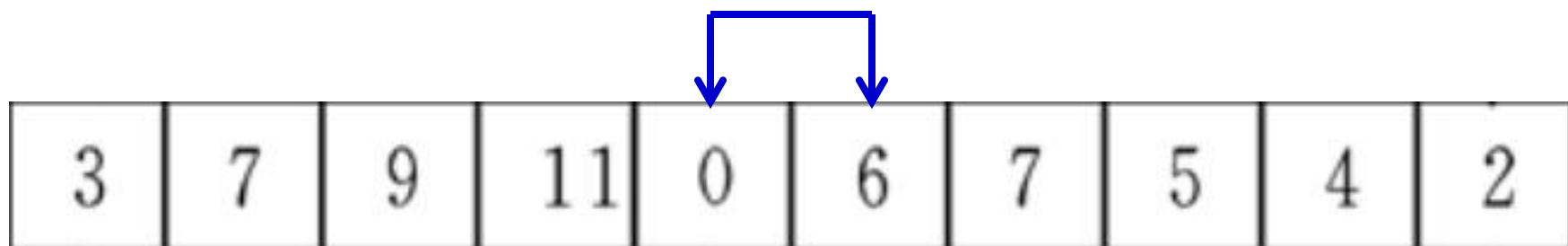
**例8.8** 将数组**a**中**n**个整数按相反顺序存放

➤ 解题思路：将**a[0]**与**a[n-1]**对换，.....  
将**a[4]**与**a[5]**对换。



**例8.8** 将数组**a**中**n**个整数按相反顺序存放

➤ 解题思路：将**a[0]**与**a[n-1]**对换，.....  
将**a[4]**与**a[5]**对换。



**i**   **j**



```
#include <stdio.h>  
int main()  
{ void inv(int x[ ],int n);  
  int i, a[10]={3,7,9,11,0,6,7,5,4,2};  
  for(i=0;i<10;i++) printf("%d ",a[i]);  
  printf("\n");  
  inv(a,10);  
  for(i=0;i<10;i++) printf("%d ",a[i]);  
  printf("\n");  
  return 0;  
}
```



```
void inv(int x[ ],int n)  
{ int temp,i,j,m=(n-1)/2;  
  for(i=0;i<=m;i++)  
  { j=n-1-i;  
    temp=x[i];x[i]=x[j];x[j]=temp;  
  }  
}
```

优化

```
void inv(int x[ ],int n)  
{ int temp,*i,*j;  
  i=x; j=x+n-1;  
  for( ; i<j; i++,j--)  
  { temp=*i; *i=*j; *j=temp; }  
}
```



**例8.9** 改写例8.8，用指针变量作实参。

```
#include <stdio.h>
```

```
int main()
```

```
{ void inv(int *x,int n);
```

```
int i, arr[10], *p=arr;
```

```
for(i=0;i<10;i++,p++)
```

```
    scanf("%d",p);
```

```
inv(p,10);
```

```
for(p=arr;p<arr+10;p++)
```

```
    printf("%d ",*p);
```

```
printf("\n");
```

```
return 0;
```

```
}
```

不可少!!!



## 例8.10 用指针方法对**10**个整数按由大到小顺序排序。

### ➤ 解题思路:

- ◆ 在主函数中定义数组**a**存放**10**个整数，定义 **int \***型指针变量**p**指向**a[0]**
- ◆ 定义函数**sort**使数组**a**中的元素按由大到小的顺序排列
- ◆ 在主函数中调用**sort**函数，用指针**p**作实参
- ◆ 用选择法进行排序



```
#include <stdio.h>  
int main()  
{ void sort(int x[ ],int n);  
  int i,*p,a[10];  
  p=a;  
  for(i=0;i<10;i++) scanf("%d",p++);  
  p=a;  
  sort(p,10);  
  for(p=a,i=0;i<10;i++)  
  { printf("%d ",*p);  p++;  }  
  printf("\n");  
  return 0;  
}
```



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

|     |    |    |     |     |    |     |   |     |     |
|-----|----|----|-----|-----|----|-----|---|-----|-----|
| 12  | 34 | 5  | 689 | -43 | 56 | -21 | 0 | 24  | 65  |
| 689 | 65 | 56 | 34  | 24  | 12 | 5   | 0 | -21 | -43 |



## 8.3.5 通过指针引用多维数组

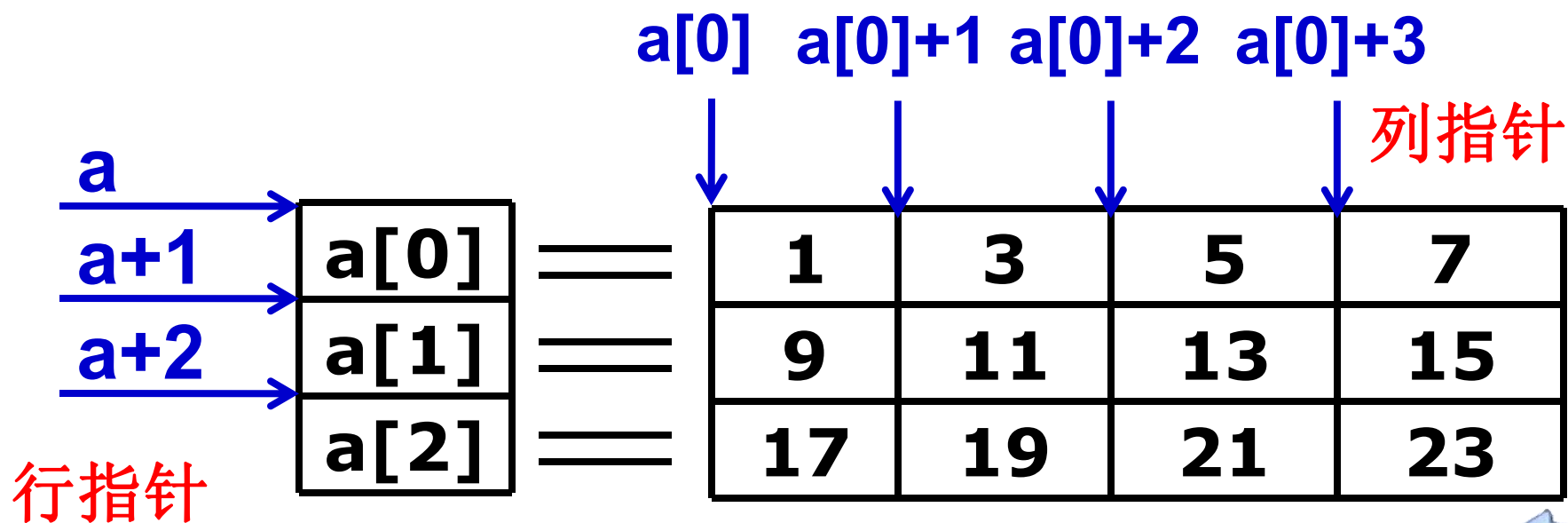
- 指针变量可以指向一维数组中的元素，也可以指向多维数组中的元素。但在概念上和使用方法上，多维数组的指针比一维数组的指针要复杂一些。



## 8.3.5 通过指针引用多维数组

### 1. 多维数组元素的地址

```
int a[3][4]={ {1,3,5,7},  
              {9,11,13,15}, {17,19,21,23}};
```

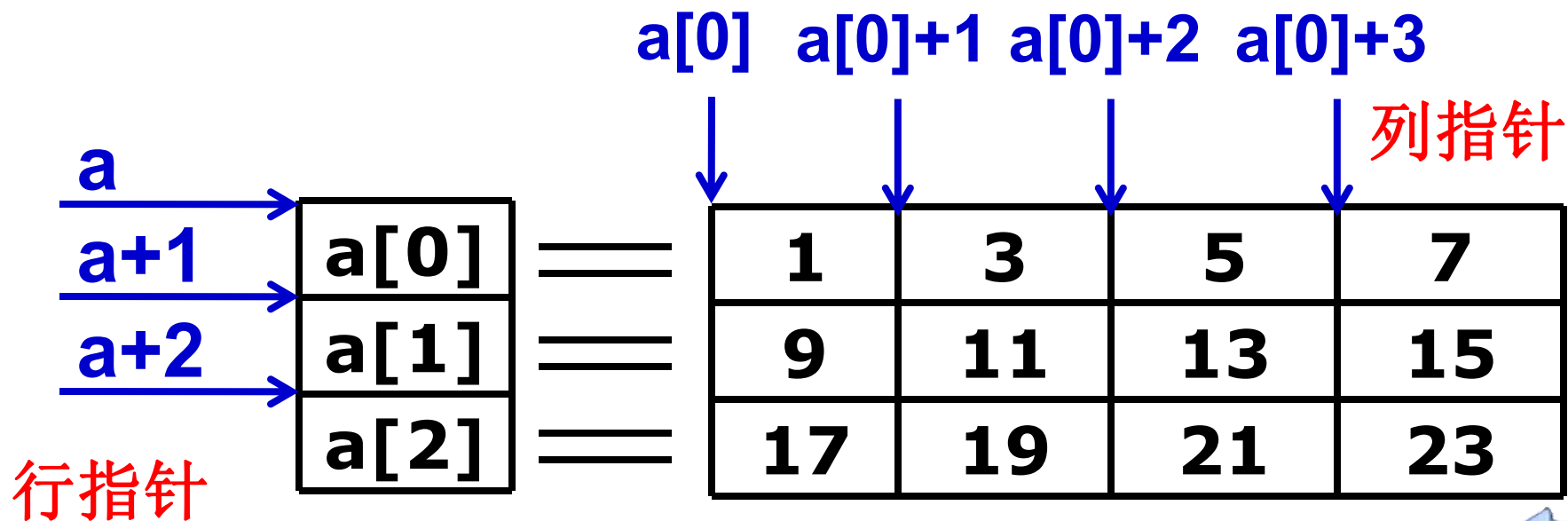


行指针每加1，走一行

**a**代表第**0**行首地址

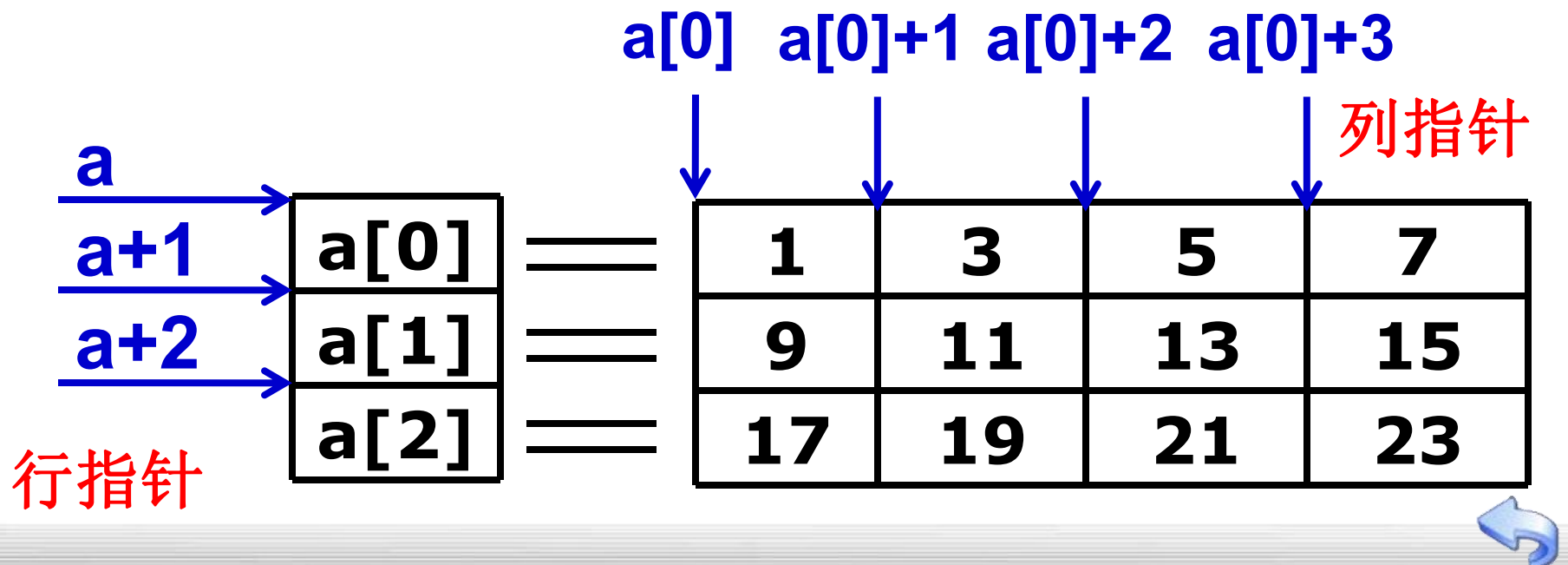
**a+1**代表第**1**行首地址

**a+2**代表第**2**行首地址



$a+i$  代表行号为  $i$  的行首地址（按行变化）

$*(a+i)$  代表什么？ 相当于  $a[i]$



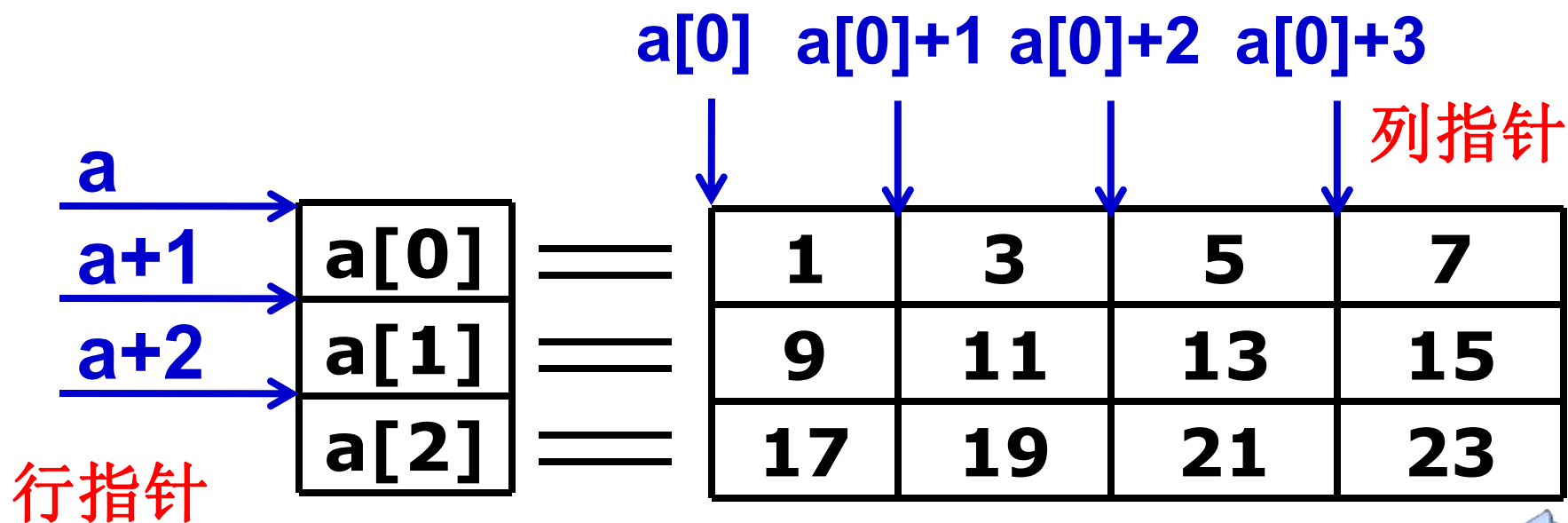
列指针每加1，走一列

$a[0]$ 代表 $a[0][0]$ 的地址

$a[0]+1$ 代表 $a[0][1]$ 的地址

$a[0]+2$ 代表 $a[0][2]$ 的地址

$a[0]+3$ 代表 $a[0][3]$ 的地址

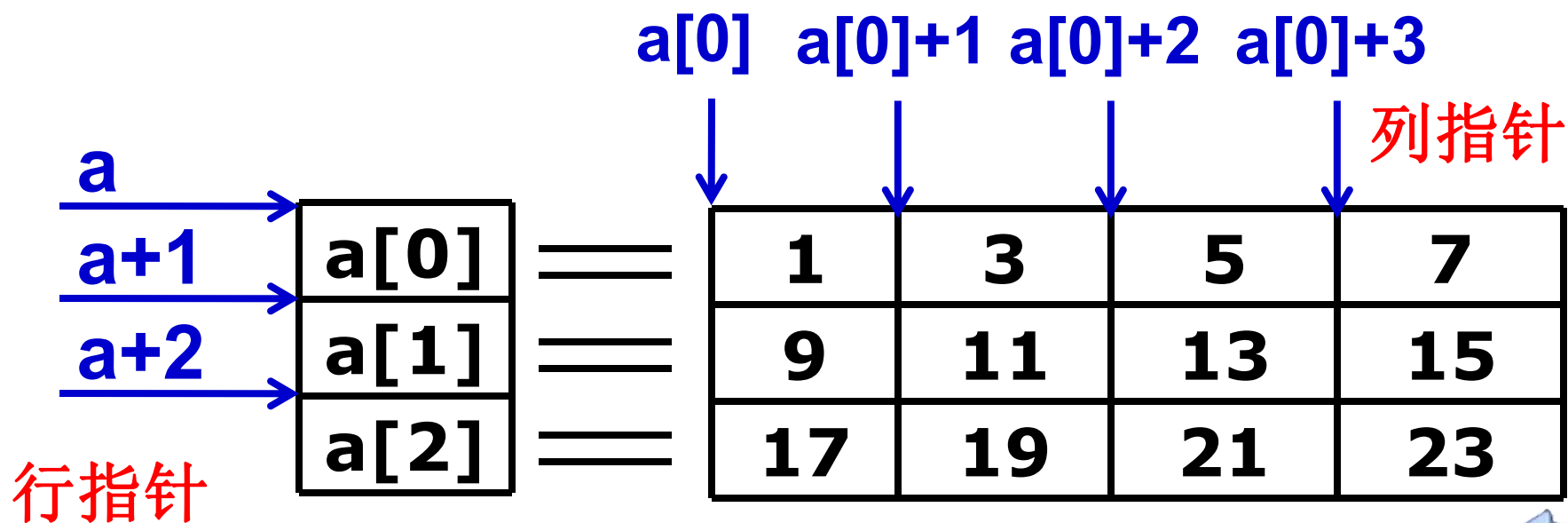


**$a[1]$** 代表谁的地址？

**$a[1]+1$** 代表谁的地址？

**$a[1]+2$** 代表谁的地址？

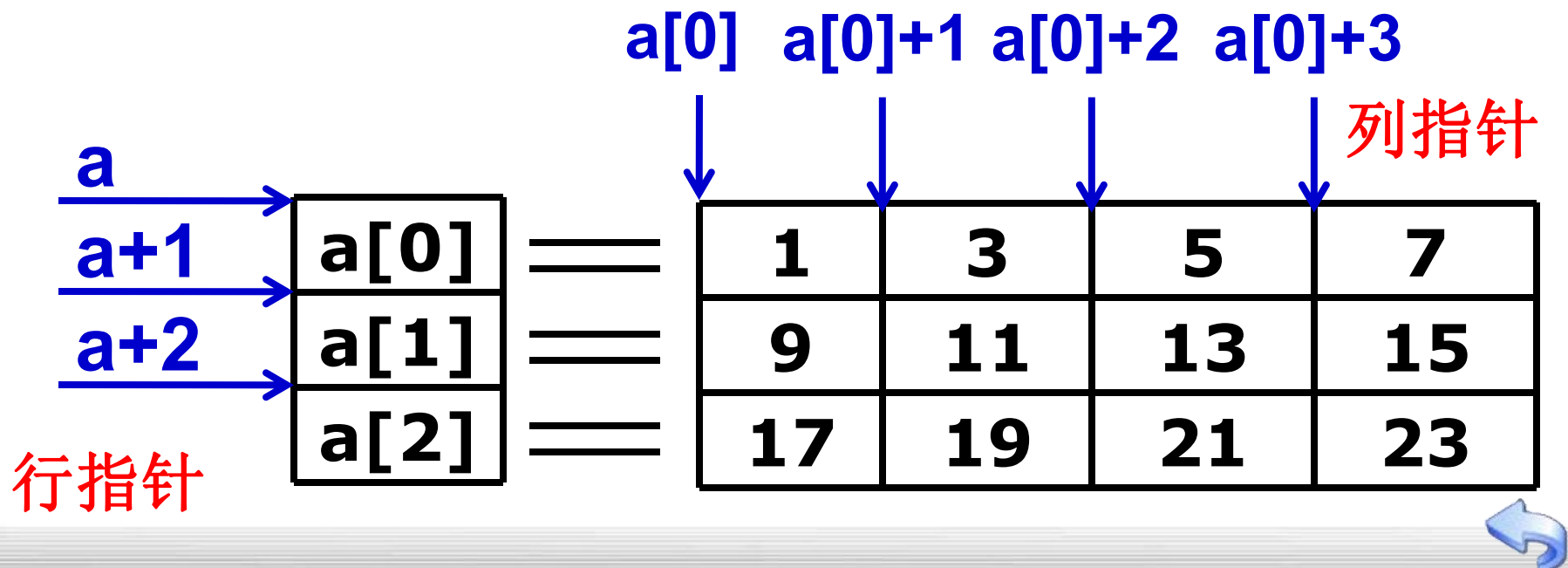
**$a[1]+3$** 代表谁的地址？



$a[i]+j$  代表谁的地址？ 代表  $a[i][j]$  的地址

$*(a[i]+j)$  代表什么？ 代表元素  $a[i][j]$

$*(*(a+i)+j)$  代表什么？ 与  $*(a[i]+j)$  等价



## 例8.11 二维数组的有关数据(地址和值)

```
#include <stdio.h>
```

```
int main()
```

```
{ int a[3][4]={1,3,5,7,9,11,13,15,  
               17,19,21,23};
```



```
printf("%d,%d\n",a,*a);  
printf("%d,%d\n",a[0],*(a+0));  
printf("%d,%d\n",&a[0],&a[0][0]);  
printf("%d,%d\n",a[1],a+1);  
printf("%d,%d\n",1245008,1245008*(a+1)+0);  
printf("%d,%d\n",1245008,1245008);  
printf("%d,%d\n",1245008,1245008);  
printf("%d,%d\n",a[1][0],*(*a+1)+0);  
printf("%d,%d\n",*a[2],*(*a+2)+0);  
return 0;  
}
```



```
printf("%d,%d\n",a,*a);
printf("%d,%d\n",a[0],*(a+0));
printf("%d,%d\n",&a[0],&a[0][0]);
printf("%d,%d\n",a[1],a+1);
printf("%d,%d\n",&a[1][0],*(a+1)+0);
printf("%d,%d\n",a[2],*(a+2));
printf("%d,%d\n",&a[2],a+2);
printf("%d,%d\n",a[1],*(a+1)+0));
printf("%d,%d\n",a[2],a+2)+0));
return 0;
}
```

```
1245024,1245024
1245024,1245024
1245040,1245040
1245040,1245040
```



```
printf("%d,%d\n",a,*a);  
printf("%d,%d\n",a[0],*(a+0));  
printf("%d,%d\n",&a[0],&a[0][0]);  
printf("%d,%d\n",a[1],a+1);  
printf("%d,%d\n",&a[1][0],*(a+1)+0);  
printf("%d,%d\n",a[2],*(a+2));  
printf("%d,%d\n",&a[2],a+2);  
printf("%d,%d\n",a[1][0],*(*a+1)+0);  
printf("%d,%d\n",*a[2],*(*a+2)+0);  
return 0;  
}
```

```
9,9  
17,17
```



## 2. 指向多维数组元素的指针变量

### (1) 指向数组元素的指针变量

**例8.12** 有一个 $3 \times 4$ 的二维数组，要求用指向元素的指针变量输出二维数组各元素的值。



## ➤ 解题思路:

- ◆ 二维数组的元素是整型的，它相当于整型变量，可以用**int\***型指针变量指向它
- ◆ 二维数组的元素在内存中是按行顺序存放的，即存放完序号为**0**的行中的全部元素后，接着存放序号为**1**的行中的全部元素，依此类推
- ◆ 因此可以用一个指向整型元素的指针变量，依次指向各个元素



```
#include <stdio.h>
```

```
int m
```

逐个访问各元素时常用此类指针

```
{ int a[3] {1,3,5,7,9,11,13,15,  
          17,19,21,23};
```

```
int *p;
```

```
for(p=a[0];p<a[0]+12;p++)
```

```
{ if((p-a[0])%4==0) printf("\n");
```

```
  printf("%4d",*p);
```

控制换行

```
}
```

```
printf("\n");
```

```
return 0;
```

```
}
```

|    |    |    |    |
|----|----|----|----|
| 1  | 3  | 5  | 7  |
| 9  | 11 | 13 | 15 |
| 17 | 19 | 21 | 23 |



## (2) 指向由 $m$ 个元素组成的一维数组的指针变量

**例8.13** 输出二维数组任一行任一列元素的值。

- 解题思路：假设仍然用例**8.12**程序中的二维数组，例**8.12**中定义的指针变量是指向变量或数组元素的，现在改用指向一维数组的指针变量。



```
#include <stdio.h>
```

```
int main()
```

```
{int a[3][4] = {1, 3, 5, 7, 9, 11, 13, 15,  
               17, 19, 21, 23};
```

行指针

```
int (*p)[4], i, j;
```

```
p = a;
```

```
printf("enter row and colum:");
```

```
scanf("%d,%d",&i,&j);
```

```
printf("a[%d,%d]=%d\n",
```

a[i][j]

```
i, j, *(*(p+i)+j));
```

```
return 0;
```

```
}
```

```
enter row and colum:1,2  
a[1,2]=13
```



### 3. 用指向数组的指针作函数参数

- 一维数组名可以作为函数参数，多维数组名也可作函数参数。
- 用指针变量作形参，以接受实参数组名传递来的地址。
- 可以有两种方法：
  - ① 用指向变量的指针变量
  - ② 用指向一维数组的指针变量



**例8.14** 有一个班，**3**个学生，各学**4**门课，计算总平均分数以及第**n**个学生的成绩。

- 解题思路：这个题目是很简单的。本例用指向数组的指针作函数参数。用函数 **average** 求总平均成绩，用函数 **search** 找出并输出第**i**个学生的成绩。



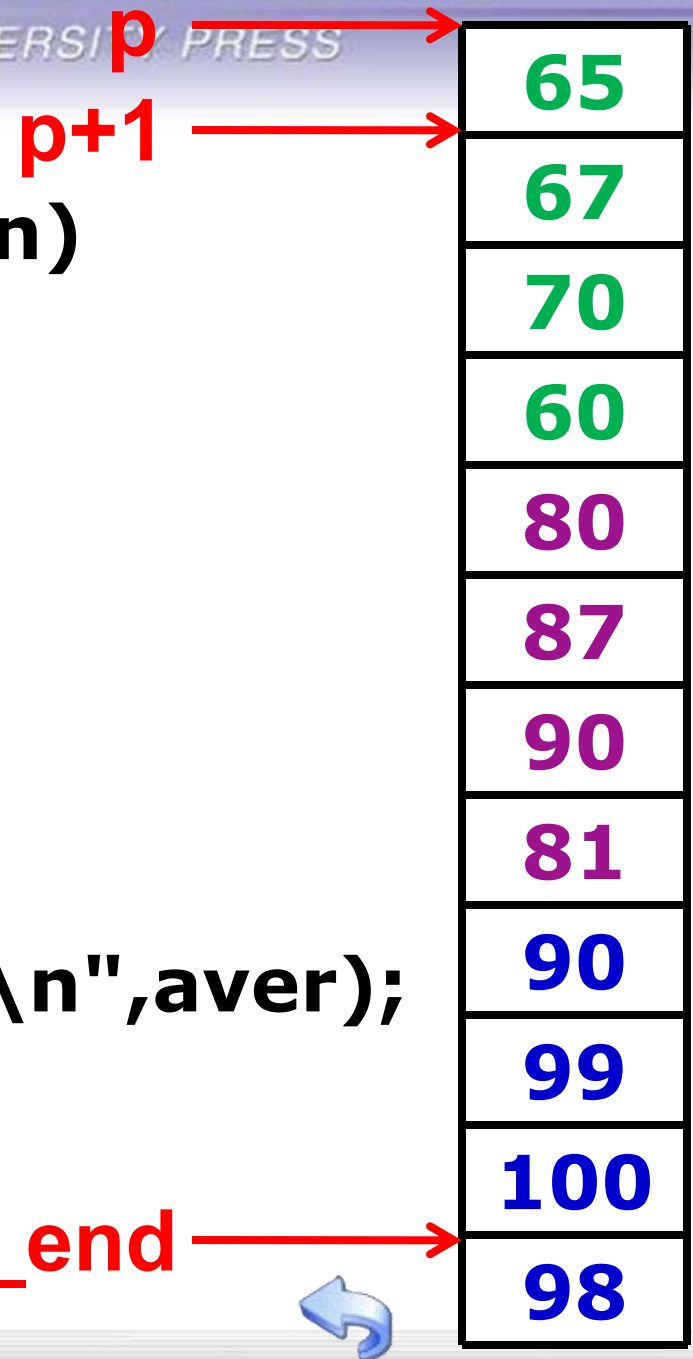
```
#include <stdio.h>
int main()
{ void average(float *p,int n);
  void search(float (*p)[4],int n);
  float score[3][4]={ {65,67,70,60},
                      {80,87,90,81},{90,99,100,98}};
  average(*score,12);
  search(score,2);
  return 0;
}
```

**score[0][0]的地址**



```
void average(float *p,int n)
{ float *p_end;
  float sum=0,aver;
  p_end=p+n-1;
  for( ;p<=p_end; p++)
    sum=sum+(*p);
  aver=sum/n;
  printf("average=%5.2f\n",aver);
}
```

average=82.25



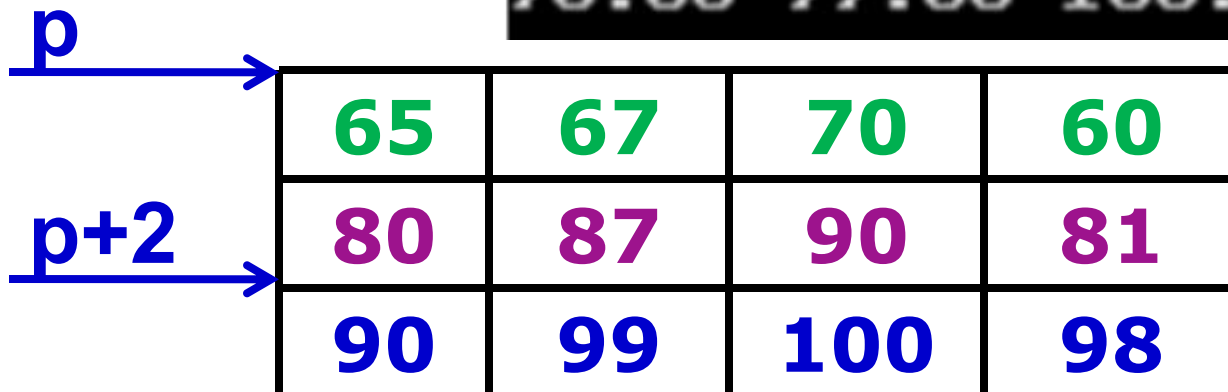
```
#include <stdio.h>
int main()
{ void average(float *p,int n);
  void search(float (*p)[4],int n);
  float score[3][4]={ {65,67,70,60},
                      {80,87,90,81},{90,99,100,98}};
  average(*score,12);
  search(score,2);
  return 0;
}
```

二维数组首行地址



```
void search(float (*p)[4],int n)
{ int i;
  printf("The score of No.%d are:\n",n);
  for(i=0;i<4;i++)
    printf("%5.2f ",*(*(p+n)+i));
  printf("\n");
}
```

The score of No.2 are:  
90.00 99.00 100.00 98.00



|           |           |            |           |
|-----------|-----------|------------|-----------|
| <b>65</b> | <b>67</b> | <b>70</b>  | <b>60</b> |
| <b>80</b> | <b>87</b> | <b>90</b>  | <b>81</b> |
| <b>90</b> | <b>99</b> | <b>100</b> | <b>98</b> |



**例8.15** 在上题基础上，查找有一门以上课程不及格的学生，输出他们的全部课程的成绩。

- **解题思路：**在主函数中定义二维数组 **score**，定义 **search** 函数实现输出一门以上课程不及格的学生们的全部课程的成绩，形参 **p** 的类型是 **float(\*)[4]**。在调用 **search** 函数时，用 **score** 作为实参，把 **score[0]** 的地址传给形参 **p**。



```
#include <stdio.h>  
int main()  
{ void search(float (*p)[4],int n);  
  float score[3][4]={ {65,57,70,60},  
    {58,87,90,81},{90,99,100,98}};  
  search(score,3);  
  return 0;  
}
```



```
void search(float (*p)[4],int n)
{ int i,j,flag;
  for(j=0;j<n;j++)
  { flag=0;
    for(i=0;i<4;i++)
      if(*(*(p+j)+i)<60) flag=1;
    if(flag==1)
    { printf("No.%d fails\n",j+1);
      for(i=0;i<4;i++)
        printf("%5.1f ",*(*(p+j)+i));
      printf("\n");
    }
  }
}
```

No.1 fails

65.0 57.0 70.0 60.0

No.2 fails

58.0 87.0 90.0 81.0

→  
p

|    |    |     |    |
|----|----|-----|----|
| 65 | 57 | 70  | 60 |
| 58 | 87 | 90  | 81 |
| 90 | 99 | 100 | 98 |

```
void search(float (*p)[4],int n)
```

```
{ int i,j,flag;
```

```
  for(j=0;j<n;j++)
```

```
  { flag=0;
```

```
    for(i=0;i<4
```

```
      if(*(*(p+j)+i)<60) flag=1;
```

```
    if(flag==1)
```

```
    { printf("No.%d fails\n",j+1);
```

```
      for(i=0;i<4;i++)
```

```
        printf("%5.1f ",*(*(p+j)+i));
```

```
      printf("\n");
```

```
    }
```

```
  }
```

```
}
```

若有不及格，则输出

发现不及格，赋1

不用**flag**，而用**break**  
语句如何改程序？



## 8.4 通过指针引用字符串

8.4.1 字符串的引用方式

8.4.2 字符指针作函数参数

8.4.3 使用字符指针变量和字符数组的比较



## 8.4.1 字符串的引用方式

- 字符串是存放在字符数组中的。引用一个字符串，可以用以下两种方法。
  - (1) 用字符数组存放一个字符串，可以通过数组名和格式声明“**%s**”输出该字符串，也可以通过数组名和下标引用字符串中一个字符。
  - (2) 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。



**例8.16** 定义一个字符数组，在其中存放字符串“**I love China!**”，输出该字符串和第**8**个字符。

- **解题思路：**定义字符数组**string**，对它初始化，由于在初始化时字符的个数是确定的，因此可不必指定数组的长度。用数组名**string**和输出格式**%s**可以输出整个字符串。用数组名和下标可以引用任一数组元素。



```
#include <stdio.h>
int main()  string↓           ↓ string+7
{ char string[]="I love China!";
  printf("%s\n",string);
  printf("%c\n",string[7]);
  return 0;
}
```

```
I love China!
C
```



**例8.17** 通过字符指针变量输出一个字符串。

➤ **解题思路：**可以不定义字符数组，只定义一个字符指针变量，用它指向字符串常量中的字符。通过字符指针变量输出该字符串。



```
#include <stdio.h>
int main()
{ char *string="I love China!";
  printf("%s\n",string);
  return 0;
}
```

**char \*string;**  
**string=" I love China!";**

**I love China!**



```
#include <stdio.h>
int main() string↓
{ char *string="I love China!";
  printf("%s\n",string);
  string="I am a student.";
  printf("%s\n",string);
  return 0;
}
```



```
#include <stdio.h>
int main()
{ char *string="I love China!";
  printf("%s\n",string);
  string="I am a student.";
  printf("%s\n",string);
  return 0;
}
```

```
I love China!
I am a student.
```



**例8.18** 将字符串**a**复制为字符串**b**，然后输出字符串**b**。

- 解题思路：定义两个字符数组**a**和**b**，用“**I am a student.**”对**a**数组初始化。将**a**数组中的字符逐个复制到**b**数组中。可以用不同的方法引用并输出字符数组元素，今用地址法算出各元素的值。



```
#include <stdio.h>
```

```
int main()
```

```
{ char a[ ]="I am a student.",b[20];
```

```
int i;
```

```
for(i=0;*(b+i)!='\0';i++)
```

```
    printf("string b is:%s\n",b);
```

```
*(b+i)='\0';
```

```
printf("string a is:%s\n",a);
```

```
printf("string b is:");
```

```
for(i=0;b[i]!='\0';i++)
```

```
    printf("%c",b[i]);
```

```
printf("\n");
```

```
return 0;
```

```
}
```

```
string a is:I am a student.  
string b is:I am a student.
```



**例8.19** 用指针变量来处理**例8.18**问题。

➤ **解题思路：** 定义两个指针变量**p1**和**p2**，分别指向字符数组**a**和**b**。改变指针变量**p1**和**p2**的值，使它们顺序指向数组中的各元素，进行对应元素的复制。



```
#include <stdio.h>
int main()
{char a[]="I am a boy.",b[20],*p1,*p2;
  p1=a; p2=b;
  for( ; *p1!='\0'; p1++,p2++)
    *p2=*p1;
  *p2='\0';
  printf("string a is:%s\n",a);
  printf("string b is:%s\n",b);
  return 0;
}
```

```
string a is:I am a student.
string b is:I am a student.
```



## 8.4.2 字符指针作函数参数

- 如果想把一个字符串从一个函数“传递”到另一个函数，可以用地址传递的办法，即用字符数组名作参数，也可以用字符指针变量作参数。
- 在被调用的函数中可以改变字符串的内容
- 在主调函数中可以引用改变后的字符串。



## 8.4.2 字符指针作函数参数

例**8.20** 用函数调用实现字符串的复制。

- 解题思路：定义一个函数**copy\_string**用来实现字符串复制的功能，在主函数中调用此函数，函数的形参和实参可以分别用字符数组名或字符指针变量。分别编程，以供分析比较。



# (1) 用字符数组名作为函数参数

```
#include <stdio.h>
```

```
int main()
```

```
{void copy_string(char from[],char to[]);
```

```
char a[]="I am a teacher.";
```

```
char b[]="you are a student.";
```

```
printf("a=%s\nb=%s\n",a,b);
```

```
printf("copy string a to string b:\n");
```

```
copy_string(a,b);
```

```
printf("a=%s\nb=%s\n",a,b);
```

```
return 0;
```

```
}
```



```
void copy_string(char from[], char to[])  
{ int i=0;  
  while(from[i]!='\0')  
  { to[i]=from[i];  
    i++;  
  }  
  to[i]='\0';  
}
```

```
a=I am a teacher.  
b=You are a student.  
copy string a to string b:  
a=I am a teacher.  
b=I am a teacher.
```



## (2)用字符型指针变量作实参

- **copy\_string**不变，在**main**函数中定义字符指针变量**from**和**to**，分别指向两个字符数组**a,b**。
- 仅需要修改主函数代码



```
#include <stdio.h>
int main()
{void copy_string(char from[], char to[]);
  char a[]="I am a teacher.";
  char b[]="you are a student.";
  char *from=a,*to=b;
  printf("a=%s\nb=%s\n",a,b);
  printf("\ncopy string a to string b:\n");
  copy_string(from,to);
  printf("a=%s\nb=%s\n",a,b);
  return 0;
}
```



## **(3)用字符指针变量作形参和实参**



```
#include <stdio.h>
int main()
{void copy_string(char *from, char *to);
  char *a="I am a teacher.";
  char b[]="You are a student.";
  char *p=b;
  printf("a=%s\nb=%s\n",a,b);
  printf("\ncopy string a to string b:\n");
  copy_string(a,p);
  printf("a=%s\nb=%s\n",a,b);
  return 0;
}
```



```
void copy_string(char *from, char *to)  
{ for( ;*from!='\0'; from++,to++)  
    { *to=*from; }  
    *to='\0';  
}
```

函数体有多种简化写法，请见主教材



## 8.4.3 使用字符指针变量和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

**(1)** 字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址（字符串第**1**个字符的地址），决不是将字符串放到字符指针变量中。



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(2) 赋值方式。可以对字符指针变量赋值，但不能对数组名赋值。

**char \*a; a="I love China!"; 对**

**char str[14];str[0]='I'; 对**

**char str[14]; str="I love China!"; 错**



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(3) 初始化的含义

**char \*a="I love China! ";**与

**char \*a; a="I love China! ";**等价



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(3) 初始化的含义

**char str[14] = "I love China! ";**与

**char str[14];**

**str[] = "I love China! ";** 不等价



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

### (4) 存储单元的内容

编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

### (4) 存储单元的内容

**char \*a; scanf("%s",a);**    错

**char \*a,str[10];**

**a=str;**

**scanf ("%s",a);**    对



## 8.4.3 使用字符指针变量和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

**(5)** 指针变量的值是可以改变的，而数组名代表一个固定的值(数组首元素的地址)，不能改变。



例8.21 改变

不能改为

**char a[]="I love China!";**

**#include <**

**int main()**

**{ char \*a="I love China!";**

**a=a+7;**

**printf("%s\n",a);**

**return 0;**

**}**

China!



## 8.4.3 使用字符指针变量和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

**(6)** 字符数组中各元素的值是可以改变的，但字符指针变量指向的字符串常量中的内容是不可以被取代的。

```
char a[]="House",*b=" House";  
a[2]='r';      对
```



## 8.4.3 使用字符指针变量和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

**(6)** 字符数组中各元素的值是可以改变的，但字符指针变量指向的字符串常量中的内容是不可以被取代的。

```
char a[]="House",*b="House";  
b[2]='r';           错
```



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

### (7) 引用数组元数

对字符数组可以用下标法和地址法引用数组元素（ **$a[5]$** ,  **$*(a+5)$** ）。如果字符指针变量 **$p=a$** ，则也可以用指针变量带下标的形式和地址法引用（ **$p[5]$** ,  **$*(p+5)$** ）。



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

**char \*a="I love China!";**

则**a[5]**的值是第**6**个字符，即字母'**e**'



## 8.4.3 使用字符指针变量和字符数组的比较

➤ 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

(8) 用指针变量指向一个格式字符串，可以用它代替**printf**函数中的格式字符串。



## 8.4.3 使用字符指针变量和字符数组的比较

- 用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要有以下几点。

```
char *format;
```

```
format="a=%d,b=%f\n";
```

```
printf(format,a,b);
```

相当于

```
printf("a=%d,b=%f\n",a,b);
```



## 8.5 指向函数的指针

8.5.1 什么是函数指针

8.5.2 用函数指针变量调用函数

8.5.3 怎样定义和使用指向函数的指针变量

8.5.4 用指向函数的指针作函数参数



## 8.5.1 什么是函数指针

- 如果在程序中定义了一个函数，在编译时，编译系统为函数代码分配一段存储空间，这段存储空间的起始地址，称为这个函数的指针。



## 8.5.1 什么是函数指针

- 可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量指向该函数。例如：

**int (\*p)(int,int);**

定义**p**是指向函数的指针变量，它可以指向类型为整型且有两个整型参数的函数。**p**的类型用**int (\*)(int,int)**表示



## 8.5.2 用函数指针变量调用函数

例8.22 用函数求整数**a**和**b**中的大者。

- 解题思路：定义一个函数**max**，实现求两个整数中的大者。在主函数调用**max**函数，除了可以通过函数名调用外，还可以通过指向函数的指针变量来实现。分别编程并作比较。



## (1) 通过函数名调用函数

```
#include <stdio.h>
```

```
int main()
```

```
{ int max(int,int);
```

```
    int a,b,c;
```

```
    printf("please enter a and b:");
```

```
    scanf("%d,%d",&a,&b);
```

```
    c=max(a,b);
```

```
    printf("%d,%d,max=%d\n",a,b,c);
```

```
    return 0;
```

```
}
```



```
int max(int x,int y)  
{ int z;  
    if(x>y) z=x;  
    else z=y;  
    return(z);  
}
```



(2)通过指针变量访问函数所返回的函数值

```
#include <stdio.h>
```

```
int main()
```

```
{ int max(int,int);
```

```
  int (*p)(int,int); int a,b,c;
```

```
  p=max;
```

```
  printf("please enter a,b,c:");
```

```
  scanf("%d,%d,%d",&a,&b,&c);
```

```
  c=(*p)(a,b);
```

```
  printf("\nd, %d, max=%d\n",a,b,c);
```

```
  return 0;
```

```
}
```

只能指向函数返回值  
为整型且有两个  
整型参数的函数

必须先指向，若写成  
**p=max(a,b);** 错



## 8.5.3 怎样定义和使用指向函数的指针变量

➤ 定义指向函数的指针变量的一般形式为  
数据类型 (\*指针变量名)(函数参数表列);

如 **int (\*p)(int,int);**

**p=max;** 对

**p=max(a,b);** 错

**p+n,p++,p--**等运算无意义



**例8.23** 输入两个整数，然后让用户选择**1**或**2**，选**1**时调用**max**函数，输出二者中的大数，选**2**时调用**min**函数，输出二者中的小数。

- 解题思路：定义两个函数**max**和**min**，分别用来求大数和小数。在主函数中根据用户输入的数字**1**或**2**，使指针变量指向**max**函数或**min**函数。



```
#include <stdio.h>
```

```
int main()
```

```
{int max(int,int);  int min(int x,int y);
```

```
  int (*p)(int,int);  int a,b,c,n;
```

```
  scanf("%d,%d",&a,&b);
```

```
  scanf("%d",&n);
```

```
  if (n==1) p=max;
```

```
  else if (n==2) p=min;
```

```
  c=(*p)(a,b);  只看此行看不出调用哪函数
```

```
  printf("a=%d,b=%d\n",a,b);
```

```
  if (n==1) printf("max=%d\n",c);
```

```
  else printf("min=%d\n",c);
```

```
  return 0;
```

```
}
```



```
int max(int x,int y)  
{ int z;  
  if(x>y) z=x;  
  else   z=y;  
  return(z);  
}  
int min(int x,int y)  
{ int z;  
  if(x<y) z=x;  
  else   z=y;  
  return(z);  
}
```



## 8.5.4 用指向函数的指针作函数参数

- 指向函数的指针变量的一个重要用途是把函数的地址作为参数传递到其他函数
- 指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够和被调用的函数中使用实参函数



## 8.5.4 用指向函数的指针作函数参数

.....

```
int main()
```

```
{ ..... fun(f1,f2) ..... }
```

```
void fun(int (*x1)(int),int (*x2)(int,int))
```

```
{ int a,b,i=3,j=5;
```

```
    a=(*x1)(i); 相当于a=f1(i);
```

```
    b=(*x2)(i,j); 相当于b=f2(i,j);
```

```
}
```



**例8.24** 有两个整数**a**和**b**，由用户输入**1,2**或**3**。如输入**1**，程序就给出**a**和**b**中大者，输入**2**，就给出**a**和**b**中小者，输入**3**，则求**a**与**b**之和。

- 解题思路：与例**8.23**相似，但现在用一个函数**fun**来实现以上功能。



```
#include <stdio.h>
int main()
{void fun(int x,int y, int (*p)(int,int));
 int max(int,int); int min(int,int);
 int add(int,int);  int a=34,b=-21,n;
 printf("please choose 1,2 or 3:");
 scanf("%d",&n);
 if (n==1)    fun(a,b,max);
 else if (n==2) fun(a,b,min);
 else if (n==3) fun(a,b,add);
 return 0;
}
```



```
int fun(int x,int y,int (*p)(int,int))
```

```
{ int resout;
```

```
    resout=(*p)(x,y);
```

```
    printf("%d\n",resout);
```

```
}
```

```
int max(int x,int y)
```

```
{ int z;
```

```
    if(x>y) z=x;
```

```
    else z=y;
```

```
    printf("max=" );
```

```
    return(z);
```

```
}
```

输入的选项为**1**时

相当于**max(x,y)**



```
int fun(int x,int y,int (*p)(int,int))
```

```
{ int resout;
```

```
    resout=(*p)(x,y);
```

```
    printf("%d\n",resout);
```

```
}
```

```
int max(int x,int y)
```

```
{ int z;
```

```
    if(x>y) z=x;
```

```
    else z=y;
```

```
    printf("max=" );
```

```
    return(z);
```

```
}
```

输入的选项为**2**时

相当于**min(x,y)**



```
int fun(int x,int y,int (*p)(int,int))
```

```
{ int result;
```

```
    result=(*p)(x,y);
```

```
    printf("%d\n",result);
```

```
}
```

```
int max(int x,int y)
```

```
{ int z;
```

```
    if(x>y) z=x;
```

```
    else z=y;
```

```
    printf("max=" );
```

```
    return(z);
```

```
}
```

输入的选项为**3**时

相当于**add(x,y)**



```
int min(int x,int y)  
{ int z;  
    if(x<y) z=x;  
    else z=y;  
    printf("min=");  
    return(z);  
}  
  
int add(int x,int y)  
{ int z;  
    z=x+y;  
    printf("sum=");  
    return(z);  
}
```



## 8.6 返回指针值的函数

- 一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已
- 定义返回指针值的函数的一般形式为  
类型名 \*函数名(参数表列);



**例8.25**有**a**个学生，每个学生有**b**门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数实现。



## 解题思路:

- ◆ 定义二维数组**score**存放成绩
- ◆ 定义输出某学生全部成绩的函数**search**，它是返回指针的函数，形参是行指针和整型
- ◆ 主函数将**score**和要找的学号**k**传递给形参
- ◆ 函数的返回值是**&score[k][0]**(**k**号学生的序号为**0**的课程地址)
- ◆ 在主函数中输出该生的全部成绩



```
#include <stdio.h>
```

```
int main()
```

```
{float score[ ][4]={ {60,70,80,90},  
    {56,89,67,88},{34,78,90,66}};
```

```
float *search(float (*pointer)[4],int n);
```

```
float *p; int i,k;
```

```
scanf("%d",&k);
```

```
printf("The scores of No.%d are:\n",k);
```

```
p=search(score,k); 返回k号学生课程首地址
```

```
for(i=0;i<4;i++)
```

```
    printf("%5.2f\t",*(p+i));
```

```
printf("\n");
```

```
return 0;
```

```
}
```



```
float *search(float (*pointer)[4],int n)
{ float *pt;
  pt=*(pointer+n);
  return(pt);
}
```



**例8.26**对例8.25中的学生，找出其中有不及格的课程的学生及其学生号。



## ➤ 解题思路:

- ◆ 在例**8.25**程序基础上修改。
- ◆ **main**函数不是只调用一次**search**函数，而是先后调用**3**次**search**函数，其中检查**3**个学生有无不及格的课程，如果有，就返回该学生的**0**号课程的地址**&score[i][0]**，否则返回**NULL**
- ◆ 在**main**函数中检查返回值，输出有不及格学生**4**门课的成绩



.....

```
float *search(float (*pointer)[4]);  
float *p; int i,j;  
for(i=0;i<3;i++)  
{ p=search(score+i);  
  if(p==*(score+i)) 相当于if(p!=NULL)  
  { printf("No.%d score:",i);  
    for(j=0;j<4;j++)  
      printf("%5.2f ",*(p+j));  
    printf("\n");  
  }  
}
```

.....



```
float *search(float (*pointer)[4])  
{ int i=0;  
  float *pt;  
  pt=NULL;  
  for( ;i<4;i++)  
    if(*(*pointer+i)<60)  
      pt=*pointer;  
  return(pt);  
}
```



## 8.7 指针数组和多重指针

8.7.1 什么是指针数组

8.7.2 指向指针数据的指针

8.7.3 指针数组作main函数的形参



## 8.7.1 什么是指针数组

➤ 一个数组，若其元素均为指针类型数据，称为指针数组，也就是说，指针数组中的每一个元素都存放一个地址，相当于一个指针变量。

➤ 定义一维指针数组的一般形式为

类型名 \* 数组名 [数组长度];

**int \*p[4];**



## 8.7.1 什么是指针数组

- 指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活
- 可以分别定义一些字符串，然后用指针数组中的元素分别指向各字符串
- 由于各字符串长度一般是不相等的，所以比用二维数组节省内存单元



## 8.7.1 什么是指针数组

**例8.27** 将若干字符串按字母顺序（由小到大）输出。

- **解题思路：** 定义一个指针数组，用各字符串对它进行初始化，然后用选择法排序，但不是移动字符串，而是改变指针数组的各元素的指向。



```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main()
```

```
{void sort(char *name[ ],int n);
```

```
void print(char *name[ ],int n);
```

```
char *name[ ]={"Follow","Great",  
               "FORTRAN","Computer"};
```

```
int n=4;
```

```
sort(name,n);
```

```
print(name,n);
```

```
return 0;
```

```
}
```

|   |   |   |   |   |   |    |
|---|---|---|---|---|---|----|
| F | o | l | l | o | w | \0 |
|---|---|---|---|---|---|----|

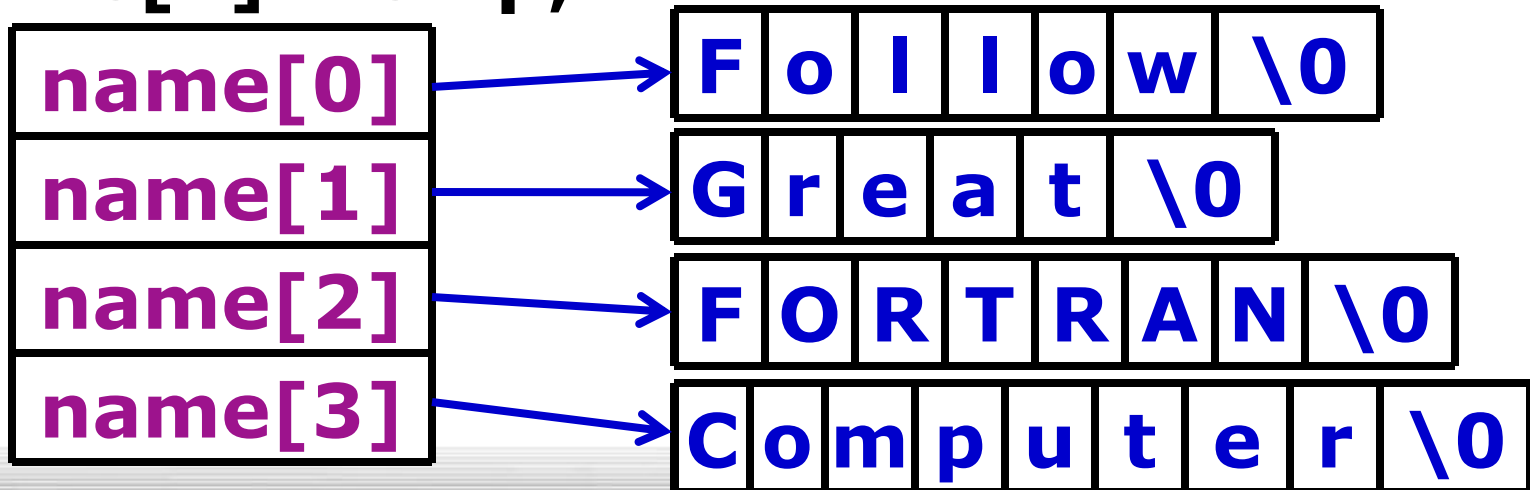
|   |   |   |   |   |    |
|---|---|---|---|---|----|
| G | r | e | a | t | \0 |
|---|---|---|---|---|----|

|   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|----|
| F | O | R | T | R | A | N | \0 |
|---|---|---|---|---|---|---|----|

|   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|----|
| C | o | m | p | u | t | e | r | \0 |
|---|---|---|---|---|---|---|---|----|



```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```



```
void sort(char *name[ ],int n)
```

```
{ char *temp; int i,j,k;
```

```
  for (i=0;i<n-1;i++)
```

```
  { k=i;
```

**i=0时**

**执行后k变为3**

```
    for (j=i+1;j<n;j++)
```

```
      if(strcmp(name[k],name[j])>0) k=j;
```

```
  if (k!=i)
```

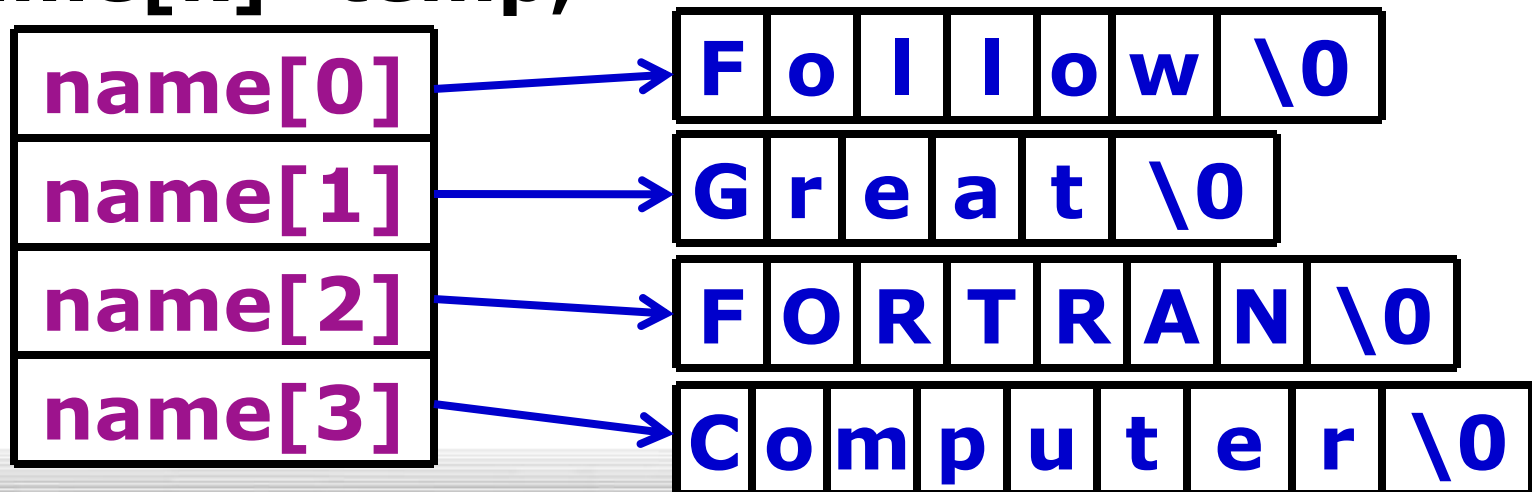
```
  { temp=name[i]; name[i]=name[k];
```

```
    name[k]=temp;
```

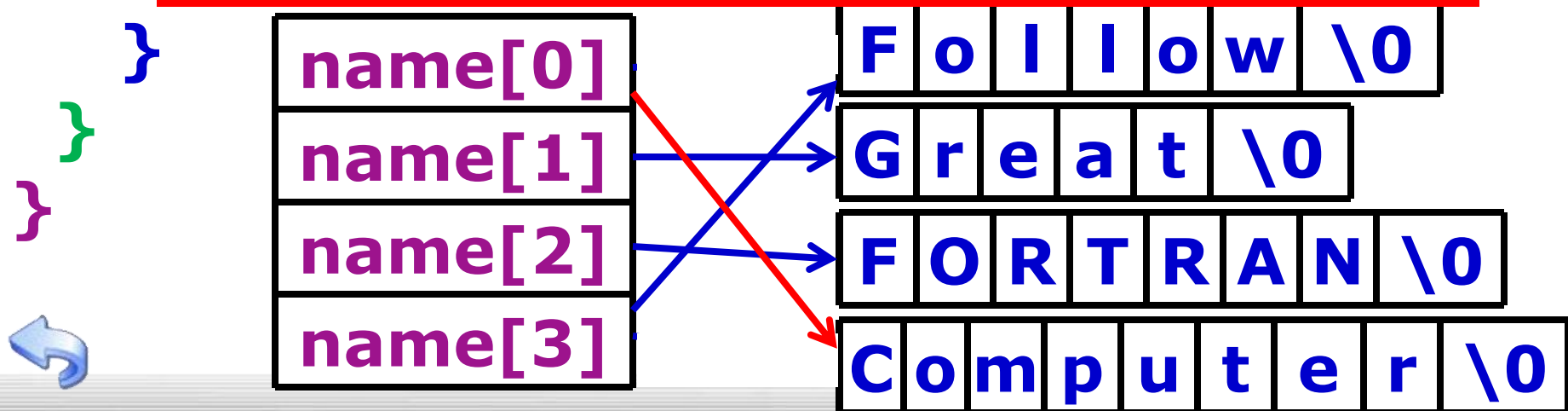
```
  }
```

```
}
```

```
}
```



```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
```



```
void sort(char *name[ ],int n)
```

```
{ char *temp; int i,j,k;
```

```
  for (i=0;i<n-1;i++)
```

```
  { k=i; i=1时 执行后k变为2
```

```
    for (j=i+1;j<n;j++)
```

```
      if(strcmp(name[k],name[j])>0) k=j;
```

```
    if (k!=i)
```

```
    { temp=name[i]; name[i]=name[k];
```

```
      name[k]=temp;
```

```
    }
```

```
  }
```

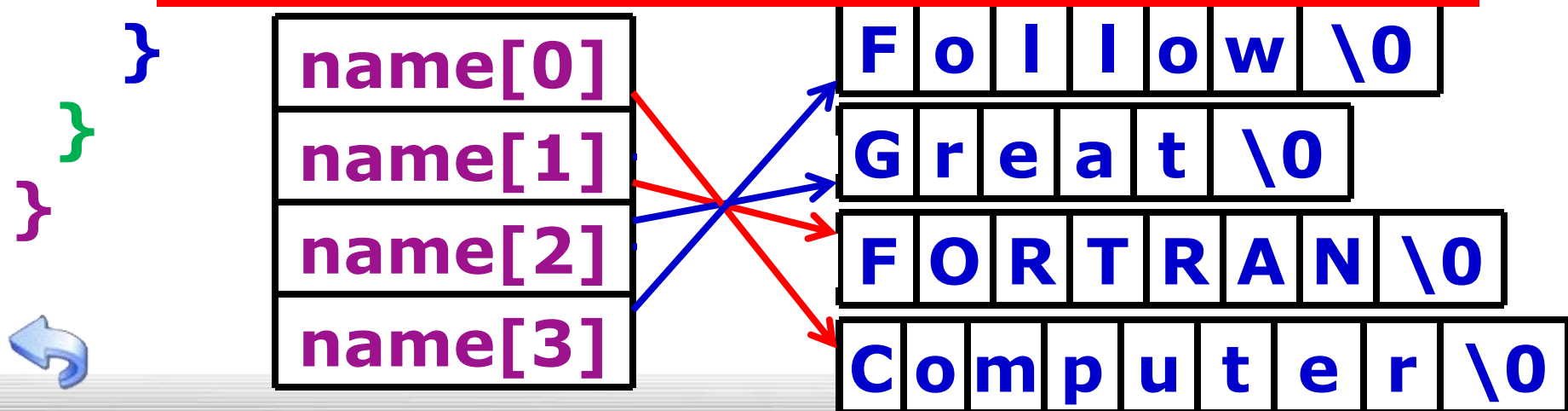
```
}
```



|         |
|---------|
| name[0] |
| name[1] |
| name[2] |
| name[3] |

|   |   |   |   |   |    |        |
|---|---|---|---|---|----|--------|
| F | o | l | l | o | w  | \0     |
| G | r | e | a | t | \0 |        |
| F | O | R | T | R | A  | N \0   |
| C | o | m | p | u | t  | e r \0 |

```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
```



```
void sort(char *name[ ],int n)
```

```
{ char *temp; int i,j,k;
```

```
  for (i=0;i<n-1;i++)
```

```
  { k=i; i=2时 执行后k变为3
```

```
    for (j=i+1;j<n;j++)
```

```
      if(strcmp(name[k],name[j])>0) k=j;
```

```
    if (k!=i)
```

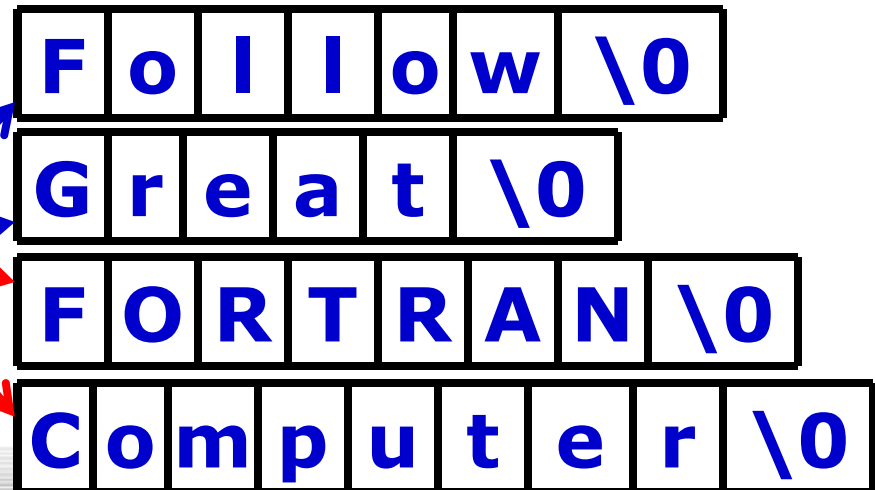
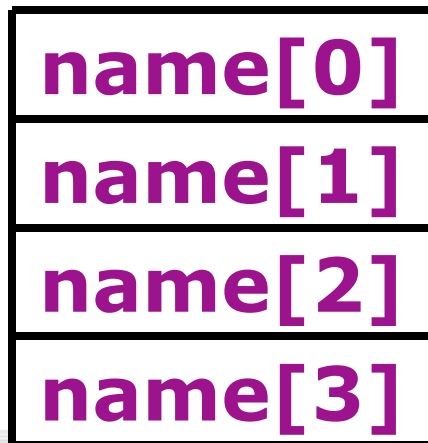
```
    { temp=name[i]; name[i]=name[k];
```

```
      name[k]=temp;
```

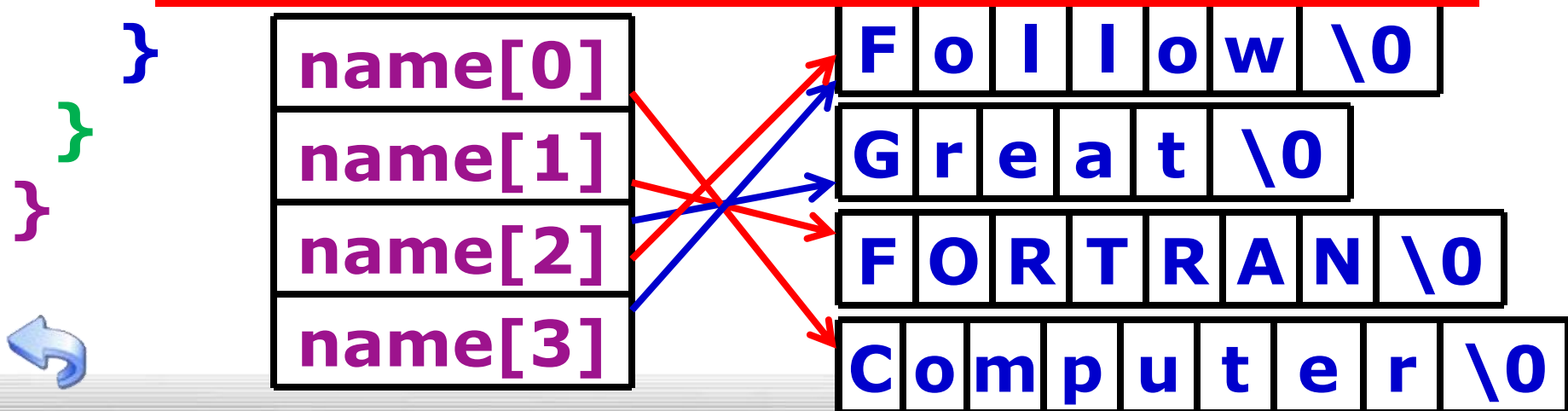
```
    }
```

```
  }
```

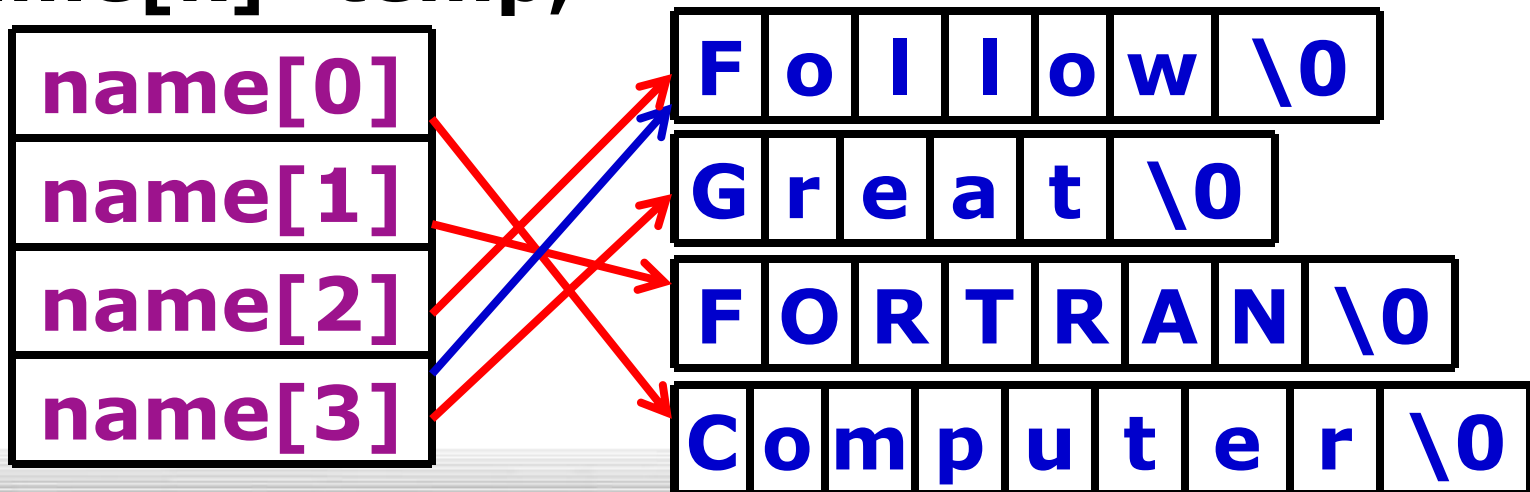
```
}
```



```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
```



```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```



```
void sort(char *name[ ],int n)
{char *temp; int i,j,k;
  for (i=0;i<n-1;i++)
  { k=i;
    for (j=i+1;j<n;j++)
      if(strcmp(name[k],name[j])>0) k=j;
    if (k!=i)
    { temp=name[i]; name[i]=name[k];
      name[k]=temp;
    }
  }
}
```

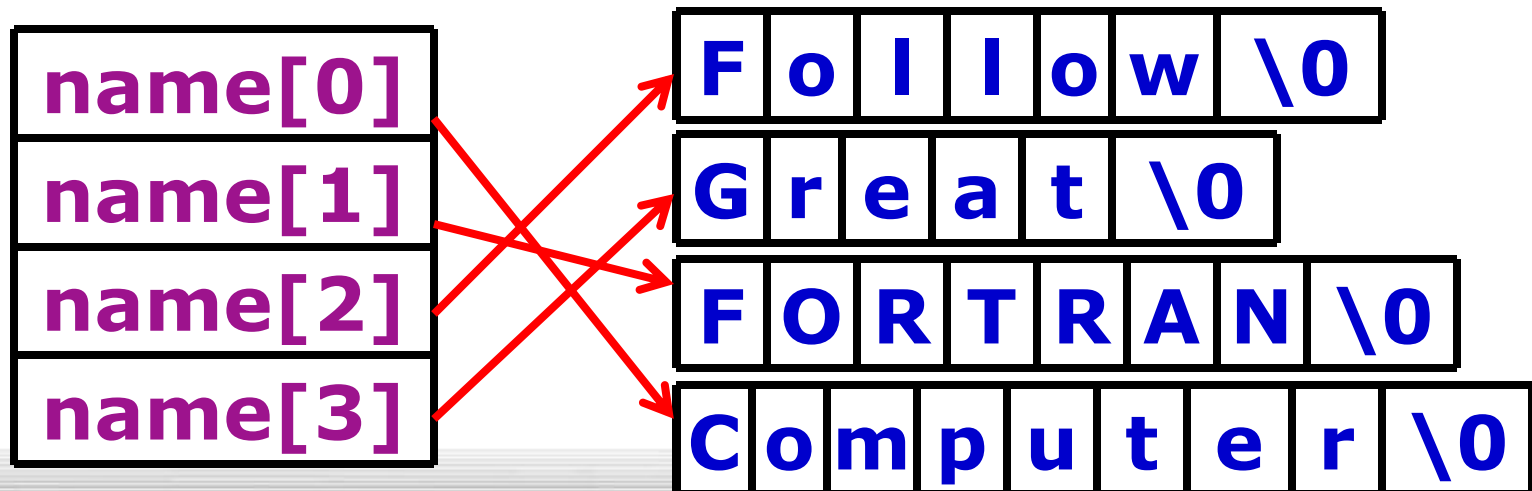
name[0]  
name[1]  
name[2]  
name[3]

|   |   |   |   |   |    |        |
|---|---|---|---|---|----|--------|
| F | o | l | l | o | w  | \0     |
| G | r | e | a | t | \0 |        |
| F | O | R | T | R | A  | N \0   |
| C | o | m | p | u | t  | e r \0 |



```
void print(char *name[ ],int n)
{ int i;
  for(i=0;i<n;i++)
    printf("%s\n",name[i]);
}
```

Computer  
FORTRAN  
Follow  
Great



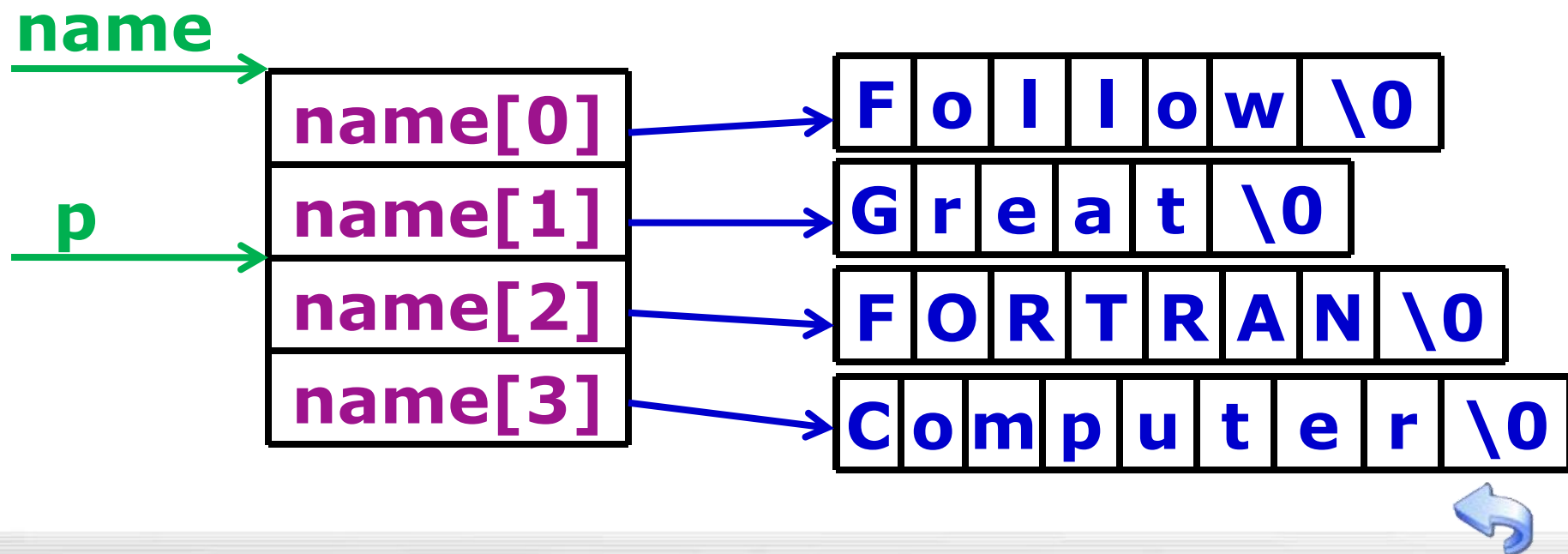
```
void print(char *name[ ],int n)  
{ int i;  
    for(i=0;i<n;i++)  
        printf("%s\n",name[i]);  
}
```

```
void print(char *name[ ],int n)  
{ int i=0; char *p;  
    p=name[0];  
    while(i<n)  
        { p=*(name+i++);  
            printf("%s\n",p);  
        }  
}
```



## 8.7.2 指向指针数据的指针

- 在了解了指针数组的基础上，需要了解指向指针数据的指针变量，简称为指向指针的指针。



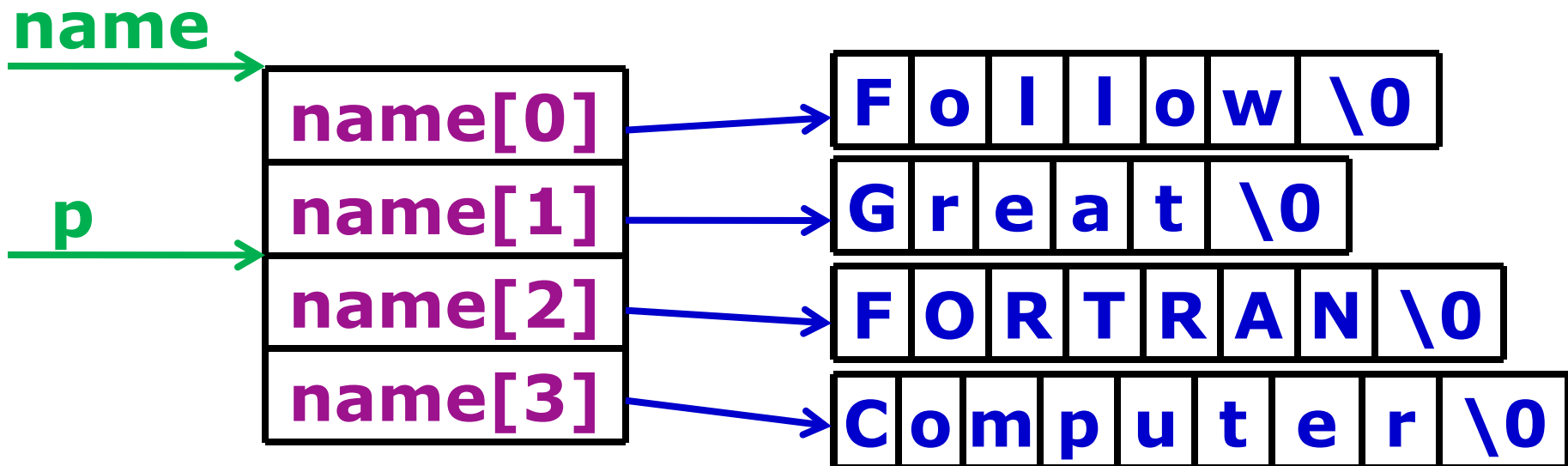
**例8.28** 使用指向指针数据的指针变量。

```
char *name[]={“Follow”,“Great”,  
               “FORTRAN”,“Computer”};
```

```
char **p;  int i;
```

```
for(i=0;i<5;i++)
```

```
{ p=name+i;  printf(“%s\n”,*p); }
```

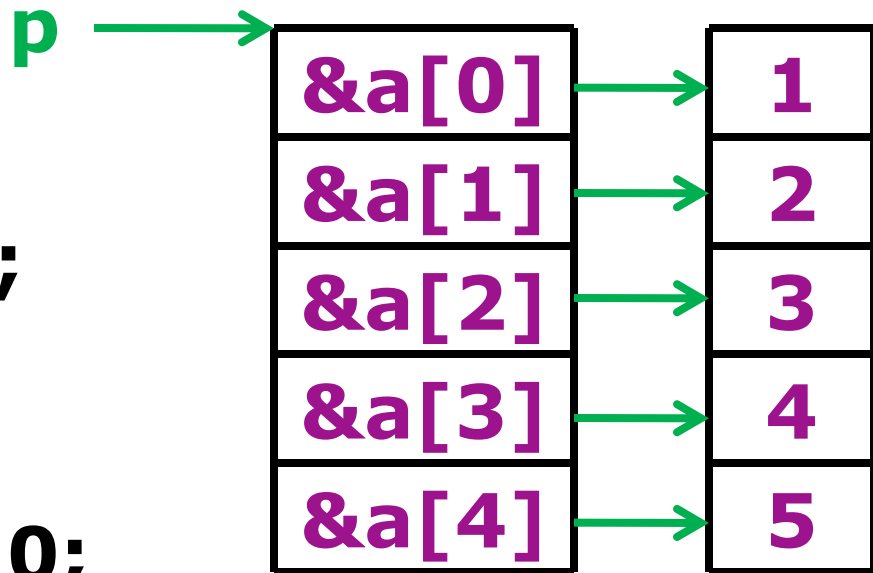


**例8.29** 有一个指针数组，其元素分别指向一个整型数组的元素，用指向指针数据的指针变量，输出整型数组各元素的值。



```
#include <stdio.h>
int main()
{int a[5]={1,3,5,7,9};
  int *num[5]={&a[0],&a[1],&a[2],
              &a[3],&a[4]};

  int **p,i;
  p=num;
  for(i=0;i<5;i++)
  { printf("%d ",**p);
    p++;
  }
  printf("\n"); return 0;
}
```



## 8.7.3 指针数组作main函数的形参

- 指针数组的一个重要应用是作为**main**函数的形参。在以往的程序中，**main**函数的第一行一般写成以下形式：

**int main()** 或 **int main(void)**

- 表示**main**函数没有参数，调用**main**函数时不必给出实参。
- 这是一般程序常采用的形式。



## 8.7.3 指针数组作main函数的形参

- 实际上，在某些情况下，**main**函数可以有参数，例如：

**int main(int argc,char \*argv[])**

其中，**argc**和**argv**就是**main**函数的形参，它们是程序的“命令行参数”。

- **argv**是**\*char**指针数组，数组中每一个元素(其值为指针)指向命令行中的一个字符串。



## 8.7.3 指针数组作main函数的形参

- 通常**main**函数和其他函数组成一个文件模块，有一个文件名。
- 对这个文件进行编译和连接，得到可执行文件（后缀为**.exe**）。用户执行这个可执行文件，操作系统就调用**main**函数，然后由**main**函数调用其他函数，从而完成程序的功能。

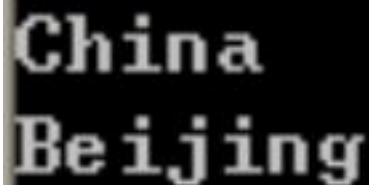


## 8.7.3 指针数组作main函数的形参

- **main**函数的形参是从哪里传递给它们的呢？
- 显然形参的值不可能在程序中得到。
- **main**函数是操作系统调用的，实参只能由操作系统给出。



```
#include <stdio.h>
int main(int argc, char *argv[])
{ while(argc > 1)
  { ++argv;
    printf("%s\n", *argv);
    --argc;
  }
  return 0;
}
```



```
China
Beijing
```

在VC++环境下编译、连接后，  
“工程” — “设置” — “调  
试” — “程序变量”中输入  
“China Beijing”，再运行就  
可得到结果



## 8.8 动态内存分配与指向它的指针变量

8.8.1 什么是内存的动态分配

8.8.2 怎样建立内存的动态分配

8.8.3 void指针类型



## 8.8.1 什么是内存的动态分配

- 非静态的局部变量是分配在内存中的动态存储区的，这个存储区是一个称为**栈**的区域
- **C**语言还允许建立内存动态分配区域，以存放一些临时用的数据，这些数据需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为**堆**区



## 8.8.2 怎样建立内存的动态分配

- 对内存的动态分配是通过系统提供的库函数来实现的，主要有**malloc**，**calloc**，**free**，**realloc**这4个函数。



## 8.8.2 怎样建立内存的动态分配

### 1. **malloc**函数

➤ 其函数原型为

**void \*malloc(unsigned int size);**

- ◆ 其作用是在内存的动态存储区中分配一个长度为**size**的连续空间
- ◆ 函数的值是为所分配区域的第一个字节的地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的开头位置



## 8.8.2 怎样建立内存的动态分配

**malloc(100);**

- ◆开辟**100**字节的临时分配域，函数值为其第**1**个字节的地址
- 注意指针的基类型为**void**，即不指向任何类型的数据，只提供一个地址
- 如果此函数未能成功地执行（例如内存空间不足），则返回空指针(**NULL**)



## 8.8.2 怎样建立内存的动态分配

### 2. calloc函数

➤ 其函数原型为

**void \*calloc(unsigned n,unsigned size);**

➤ 其作用是在内存的动态存储区中分配**n**个长度为**size**的连续空间，这个空间一般比较大，足以保存一个数组。



## 8.8.2 怎样建立内存的动态分配

- 用**calloc**函数可以为二维数组开辟动态存储空间，**n**为数组元素个数，每个元素长度为**size**。这就是动态数组。函数返回指向所分配域的起始位置的指针；如果分配不成功，返回**NULL**。如：

```
p=calloc(50,4);
```

开辟**50×4**个字节的临时分配域，把起始地址赋给指针变量**p**



## 8.8.2 怎样建立内存的动态分配

### 3. free函数

➤ 其函数原型为

**void free(void \*p);**

➤ 其作用是释放指针变量 p 所指向的动态空间，使这部分空间能重新被其他变量使用。p 应是最近一次调用 **calloc** 或 **malloc** 函数时得到的函数返回值。



## 8.8.2 怎样建立内存的动态分配

**free(p);**

- 释放指针变量 p 所指向的已分配的动态空间
- **free**函数无返回值



## 8.8.2 怎样建立内存的动态分配

### 4. realloc函数

➤ 其函数原型为

**void \*realloc(void \*p,unsigned int size);**

➤ 如果已经通过**malloc**函数或**calloc**函数获得了动态空间，想改变其大小，可以用**realloc**函数重新分配。



## 8.8.2 怎样建立内存的动态分配

- 用**realloc**函数将**p**所指向的动态空间的大小改变为**size**。**p**的值不变。如果重分配不成功，返回**NULL**。如

**realloc(p,50);**

将**p**所指向的已分配的动态空间改为**50**字节



## 8.8.2 怎样建立内存的动态分配

- 以上**4**个函数的声明在**stdlib.h**头文件中，在用到这些函数时应当用“**#include <stdlib.h>**”指令把**stdlib.h**头文件包含到程序文件中。



## 8.8.3 void指针类型

**例8.30** 建立动态数组，输入**5**个学生的成绩，另外用一个函数检查其中有无低于**60**分的，输出不合格的成绩。



## 8.8.3 void指针类型

- 解题思路：用**malloc**函数开辟一个动态自由区域，用来存**5**个学生的成绩，会得到这个动态域第一个字节的地址，它的基类型是**void**型。用一个基类型为**int**的指针变量**p**来指向动态数组的各元素，并输出它们的值。但必须先把**malloc**函数返回的**void**指针转换为整型指针，然后赋给**p1**



```
#include <stdio.h>  
#include <stdlib.h>  
int main()  
{ void check(int *);  
  int *p1,i;  
  p1=(int *)malloc(5*sizeof(int));  
  for(i=0;i<5;i++)  
    scanf("%d",p1+i);  
  check(p1);  
  return 0;  
}
```



```
void check(int *p)
{ int i;
  printf("They are fail:");
  for(i=0;i<5;i++)
    if (p[i]<60)
      printf("%d ",p[i]);
  printf("\n");
}
```

```
67 98 59 78 57
They are fail:59 57
```



## 8.9 有关指针的小结

1. 首先要准确地弄清楚指针的含义。指针就是地址，凡是出现“指针”的地方，都可以用“地址”代替，例如，变量的指针就是变量的地址，指针变量就是地址变量
- 要区别指针和指针变量。指针就是地址本身，而指针变量是用来存放地址的变量。



## 8.9 有关指针的小结

**2.** 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。但应注意：只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。



## 8.9有关指针的小结

**void \***指针是一种特殊的指针，不指向任何类型的数据，如果需要用此地址指向某类型的数据，应先对地址进行类型转换。可以在程序中进行显式的类型转换，也可以由编译系统自动进行隐式转换。无论用哪种转换，读者必须了解要进行类型转换



## 8.9 有关指针的小结

**3.** 要深入掌握在对数组的操作中怎样正确地使用指针，搞清楚指针的指向。一维数组名代表数组首元素的地址



## 8.9有关指针的小结

```
int *p,a[10];  
p=a;
```

- ◆ **p**是指向**int**类型的指针变量，**p**只能指向数组中的元素，而不是指向整个数组。在进行赋值时一定要先确定赋值号两侧的类型是否相同，是否允许赋值。
- ◆ 对“**p=a;**”，准确地说应该是：**p**指向**a**数组的首元素



## 8.9 有关指针的小结

**4.**有关指针变量的定义形式的归纳比较，见主教材中表**8.4**。



## 8.9 有关指针的小结

### 5. 指针运算

#### (1) 指针变量加（减）一个整数

例如： $p++$ ,  $p--$ ,  $p+i$ ,  $p-i$ ,  $p+=i$ ,  $p-=i$  等  
均是指针变量加（减）一个整数。

- 将该指针变量的原值（是一个地址）和它指向的变量所占用的存储单元的字节数相加（减）。



## 8.9 有关指针的小结

### 5. 指针运算

#### (2) 指针变量赋值

- 将一个变量地址赋给一个指针变量
- 不应把一个整数赋给指针变量



## 8.9 有关指针的小结

### 5. 指针运算

#### (3) 两个指针变量可以相减

- 如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数



## 8.9有关指针的小结

### 5. 指针运算

#### (4) 两个指针变量比较

- 若两个指针指向同一个数组的元素，则可以进行比较
- 指向前面的元素的指针变量“小于”指向后面元素的指针变量
- 如果**p1**和**p2**不指向同一数组则比较无意义



## 8.9有关指针的小结

### 5. 指针运算

**(5)** 指针变量可以有空值，即该指针变量不指向任何变量，可以这样表示：

**p=NULL;**

