第8章 善于利用指针

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

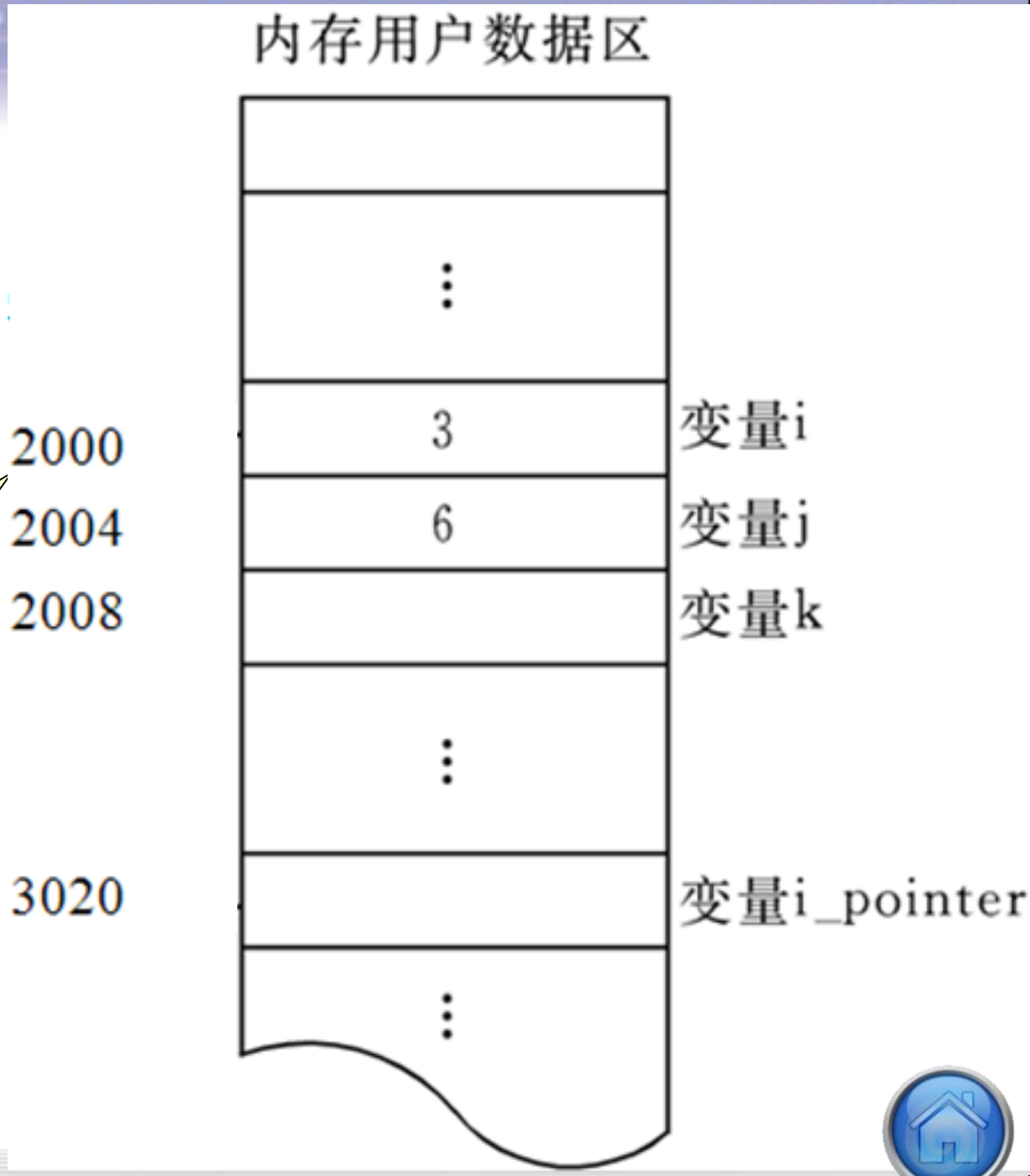
例如：

```
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



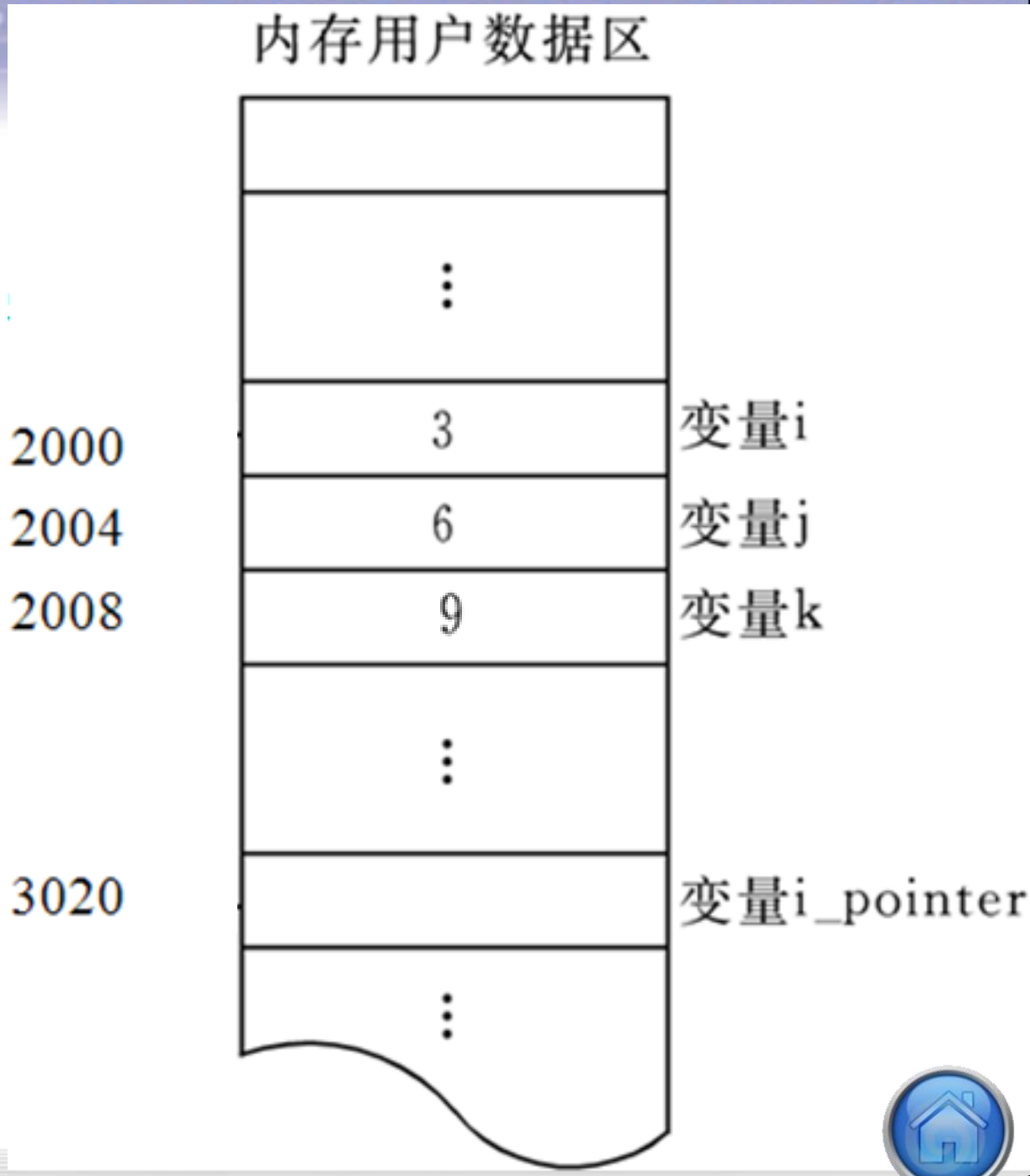
```
int i=3,j=6,k;
k=i+j;
```

从这里取3

从这里取6

将9送到这里

直接存取



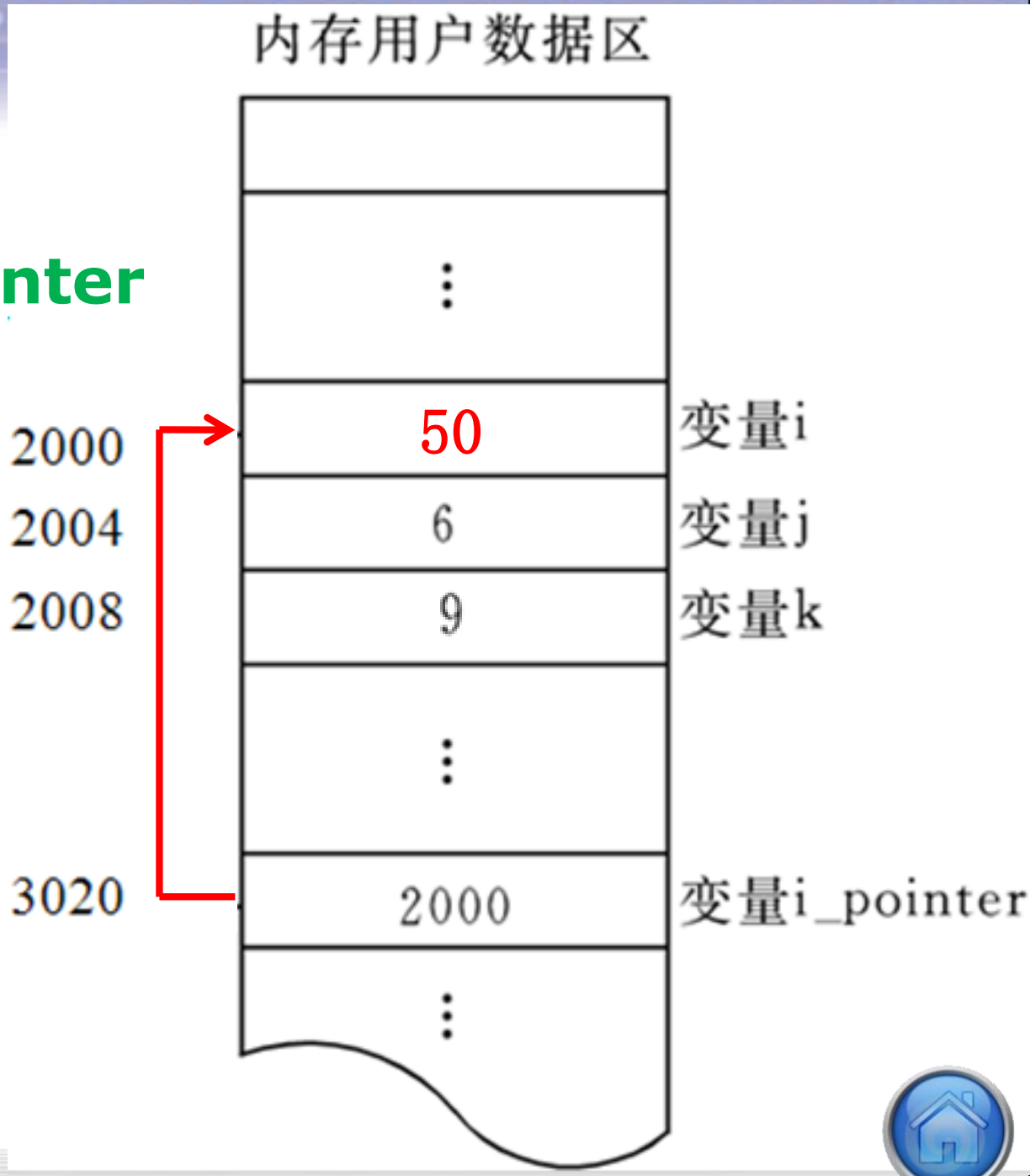
```
int i=3,j=6,k;
```

定义特殊变量i_pointer

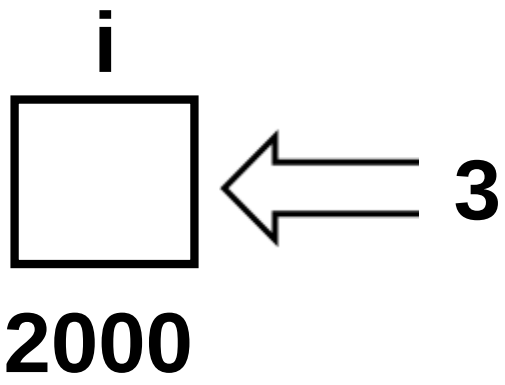
```
i_pointer=&i;
*i_pointer=50;
```

将i的地址存到这里

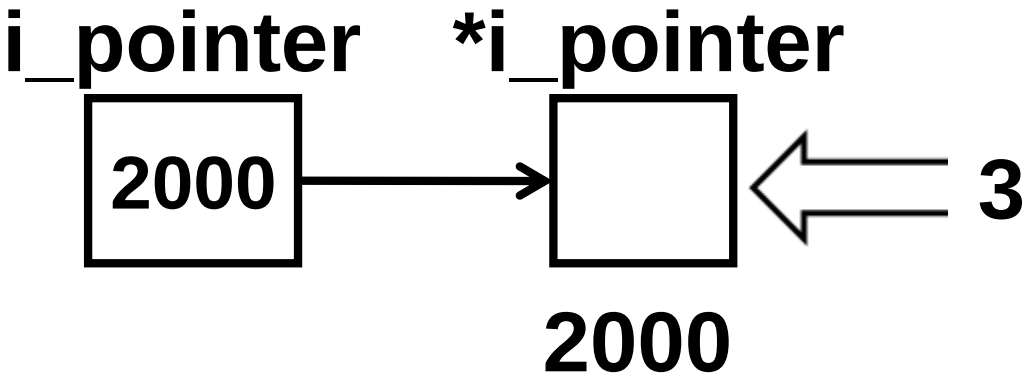
间接存取



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，指針变量的值是地址（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

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_1=3.14;
```

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：
  - 给指针变量赋值：p=&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);

*p相当于a

以八进制输出a的地址

极少如此使用

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)

指针运算符*的两个含义

1、定义时，表示后续是个指针变量

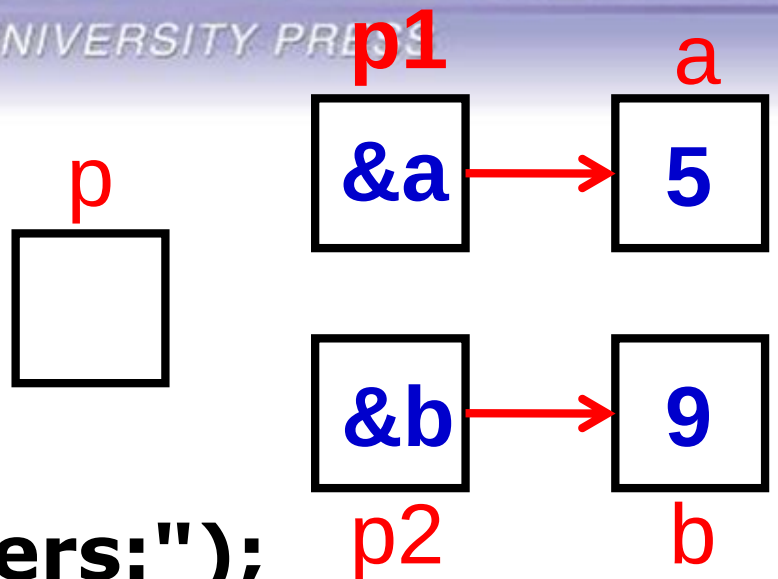
如 **int *pointer_1**

2、定义后，表示指向的内容

如 ***pointer_1 = 5**

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。

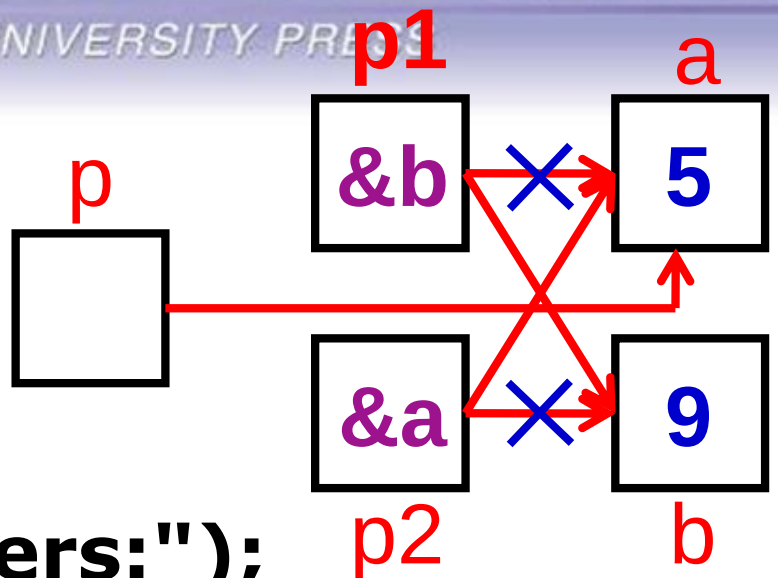
- 解题思路：用指针方法来处理这个问题。不交换整型变量的值，而是交换两个指针变量的值。



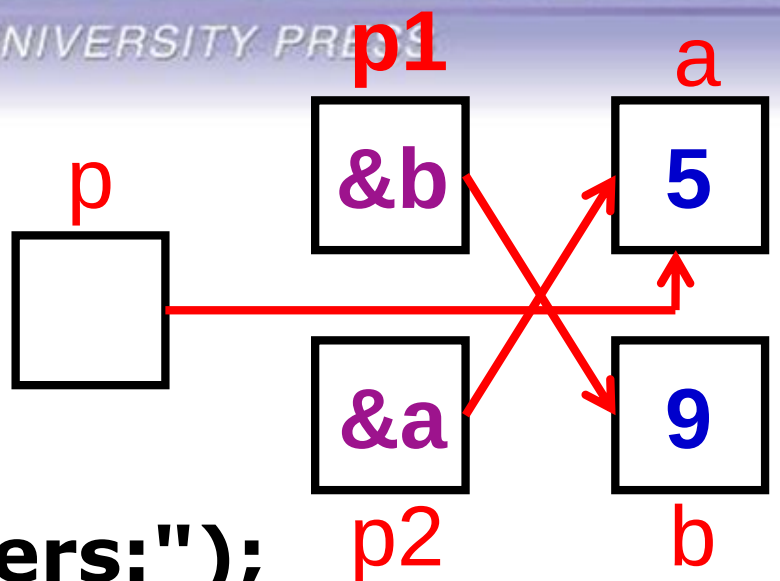

```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);
  p1=&a;   p2=&b;
  if(a<b)
  {  p=p1; p1=p2; p2=p; }
  printf("a=%d,b=%d\n",a,b);
  printf("%d,%d\n",*p1,*p2);
  return 0;
}
```

5,9

成立

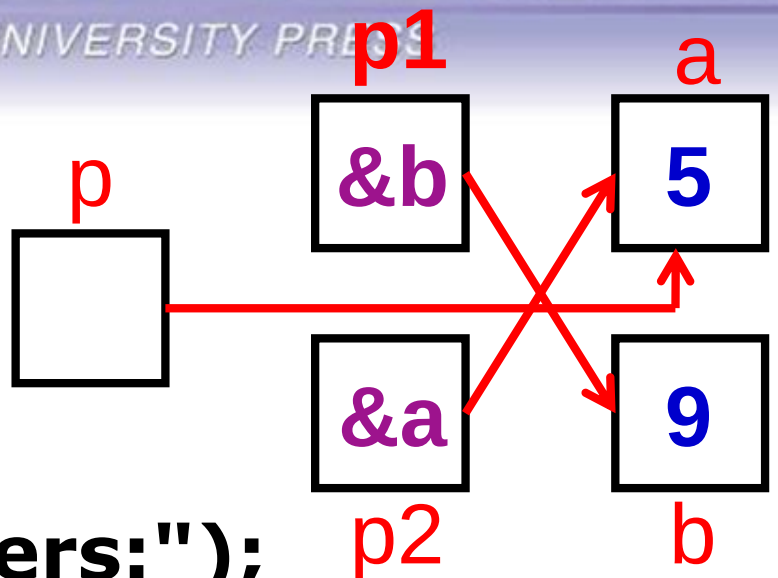



```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);
  p1=&a;   p2=&b;
  if(a<b)
  { p=p1; p1=p2; p2=p; }
  printf("a=%d,b=%d\n",a,b);
  printf("%d,%d\n",*p1,*p2);
  return 0;
}
```




```
#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;?

```
#include <stdio.h>
int main()
{ int *p1,*p2,*p,a,b,t;
  printf("integer numbers:");
  scanf("%d,%d",&a,&b);          5,9
  p1=&a;   p2=&b;
  if(a<b)
  {  t=*p1; *p1=*p2; *p2=t; }
  printf("a=%d,b=%d\n",a,b);   a=9,b=5
  printf("%d,%d\n",*p1,*p2);    9,5
  return 0;
}
```

- 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

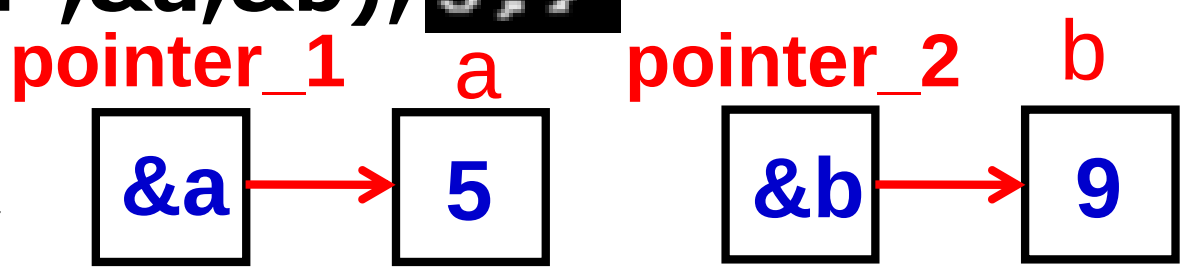
- 只能把地址赋值给指针变量
- 指针变量赋值，注意基类型一致
- 指针变量的赋值表示指针指向的改变
- *p的赋值表示指针指向单元中内容的改变
- 指针变量p未赋值时，其指向的单元不确定，故*p不可赋值

8.2.4 指针变量作为函数参数

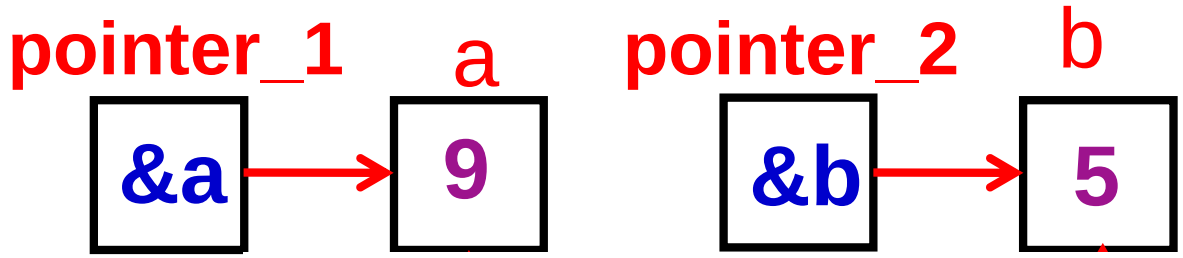
例8.3 题目要求同例8.2，即对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。

- 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_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
```

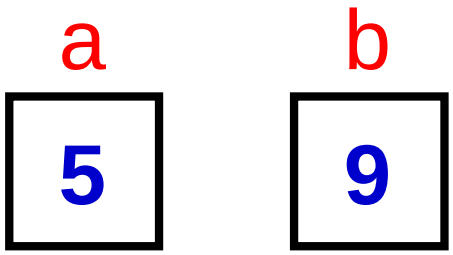
```
void swap(int *p1,int *p2)
{ int temp;
   temp=*p1;
   *p1=*p2;
   *p2=temp;
}
```

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



- 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。

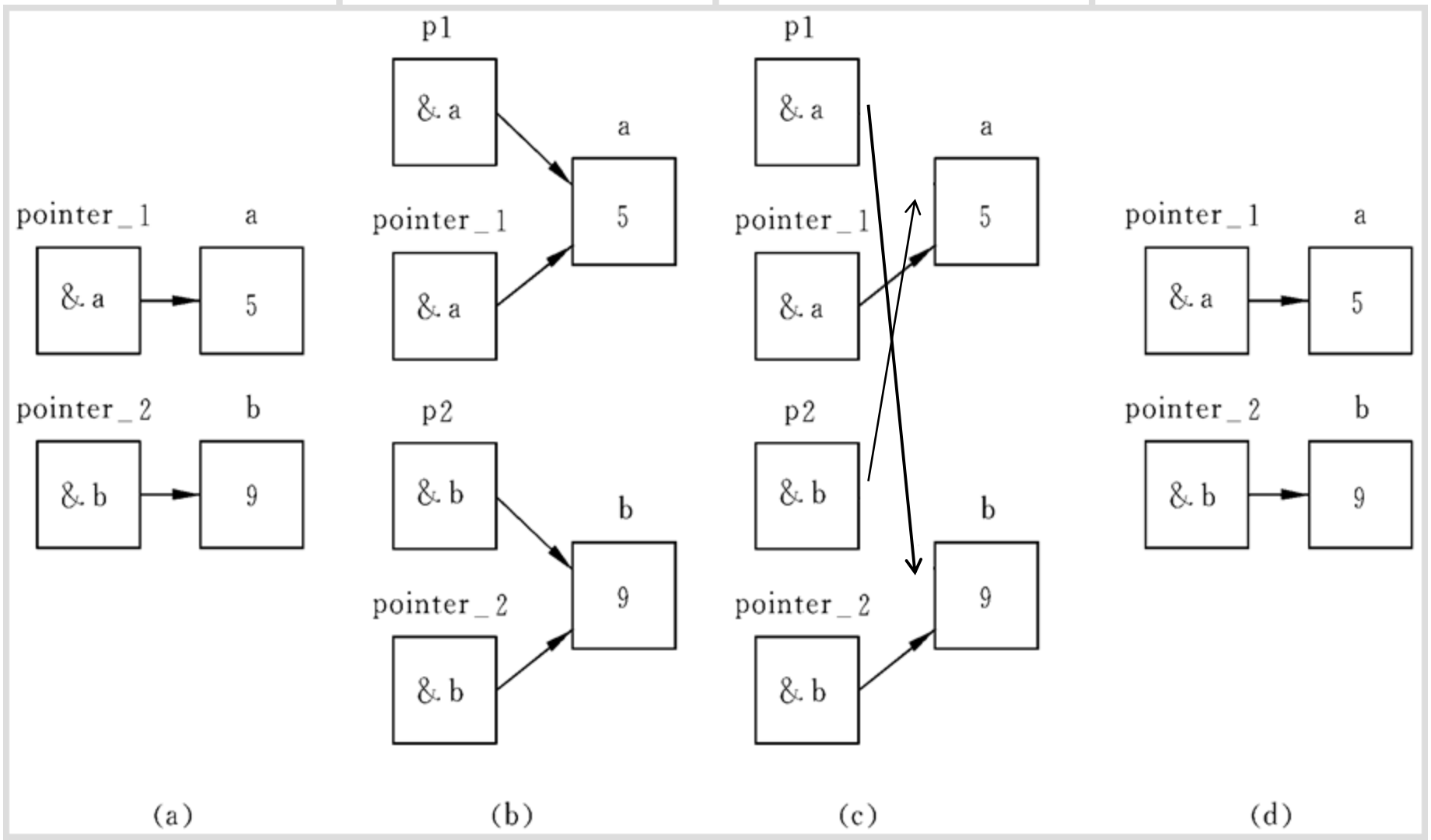
- 解题思路：尝试调用swap函数来实现题目要求。在函数中改变形参(指针变量)的值，希望能由此改变实参(指针变量)的值

```
#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",
   *pointer_1,*pointer_2);
  return 0;
 }  void swap(int *p1,int *p2)
    { int *p;
      p=p1; p1=p2; p2=p;
    }
```

错！！！
只交换形参指向

```
#include <stdio.h>
int main()
 {void swap(int *p1,int *p2);
  int a,b;  int*pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if (a<b)  swap(pointer_1,pointer_2);
  printf("max=%d,min=%d\n",
   *pointer_1,*pointer_2);
  return 0;
 }
```

调用结束后不会改变实际参数的值，
即不改变实参指针的指向


pointer_1
a
&a
5
pointer_2
b
&b
9
(a)
p1
&a
a
5
pointer_1
&a
p2
&b
b
9
pointer_2
&b
(b)
p1
&a
a
5
pointer_1
&a
p2
&b
b
9
pointer_2
&b
(c)
pointer_1
a
&a
5
pointer_2
b
&b
9
(d)

- 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);
  printf("%d,%d,%d\n",a,b,c);
  return 0;
}
```

调用结束后不会改变指针的指向

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

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

交换指针指向的变量值

- 注意：

  1、指针temp未赋值时，不可以对*temp赋值，见P225

  2、指针变量作为函数的参数，其值不可以改变（指针指向不变），但是其指向单元中的内容可以被修改。

  3、注意形参与实参的关系。即point_1与point_2的指向不变，但是其指向单元中的内容发生变化。P224

  4、注意区分4个swap函数的区别，1个对，3个错。

```
void swap（int * p 1， int * p 2）
 {  int  temp;
    temp＝* p 1;
   * p 1 ＝* p 2 ;
   * p 2 ＝temp;
 }
```

第一个swap函数

```
void swap（int * p 1， int * p 2）
 {  int  *temp;
    *temp＝* p 1;
   * p 1 ＝* p 2 ;
   * p 2 ＝temp;
 }
```

第二个swap函数

```
void swap（int  x，int  y）
 {  int  temp;
    temp＝x;
       x＝y;
       y＝temp;
 }
```

第三个swap函数

```
void swap（int * p 1 ， int * p 2 ）
 {  int  *p;
    p＝ p 1;
    p 1 ＝ p 2 ;
    p 2 ＝ p;
 }
```

第四个swap函数

- 函数的返回值最多只能有一个
  若想通过一个函数得到一个以上的结果，该如何处理？

  利用指针，
  如void test(int *p1, int *p2)
  调用后，可通过p1 p2将结果返回

如果想利用指针，得到3个结果

1、void test(int *p1, int *p2 , int *p3)
调用后，可通过p1 p2 p3将结果返回

2、 int test(int *p1, int *p2）

一个函数的返回值，p1 p2两个参数

小　结

指针是什么？

指针变量是什么？

指针变量赋值注意点？

- 不能把整数赋值给指针变量
- 基类型要一致
- 指针变量的赋值、复制表示指针指向的改变。

*p 赋值注意点？

- *p的赋值表示指向内容的修改
- *p赋值时，p要有一个确定的指向
- 注意与定义指针变量时的*区别

```
int a,b,c,d；
int *p1 =&a；
int *p2；  p2 = &b;
int  *p3；*p3 = &c;  错
int  *p4；*p4 =d；   错
*p1=5;  *p2=d;
```

8.3通过指针引用数组

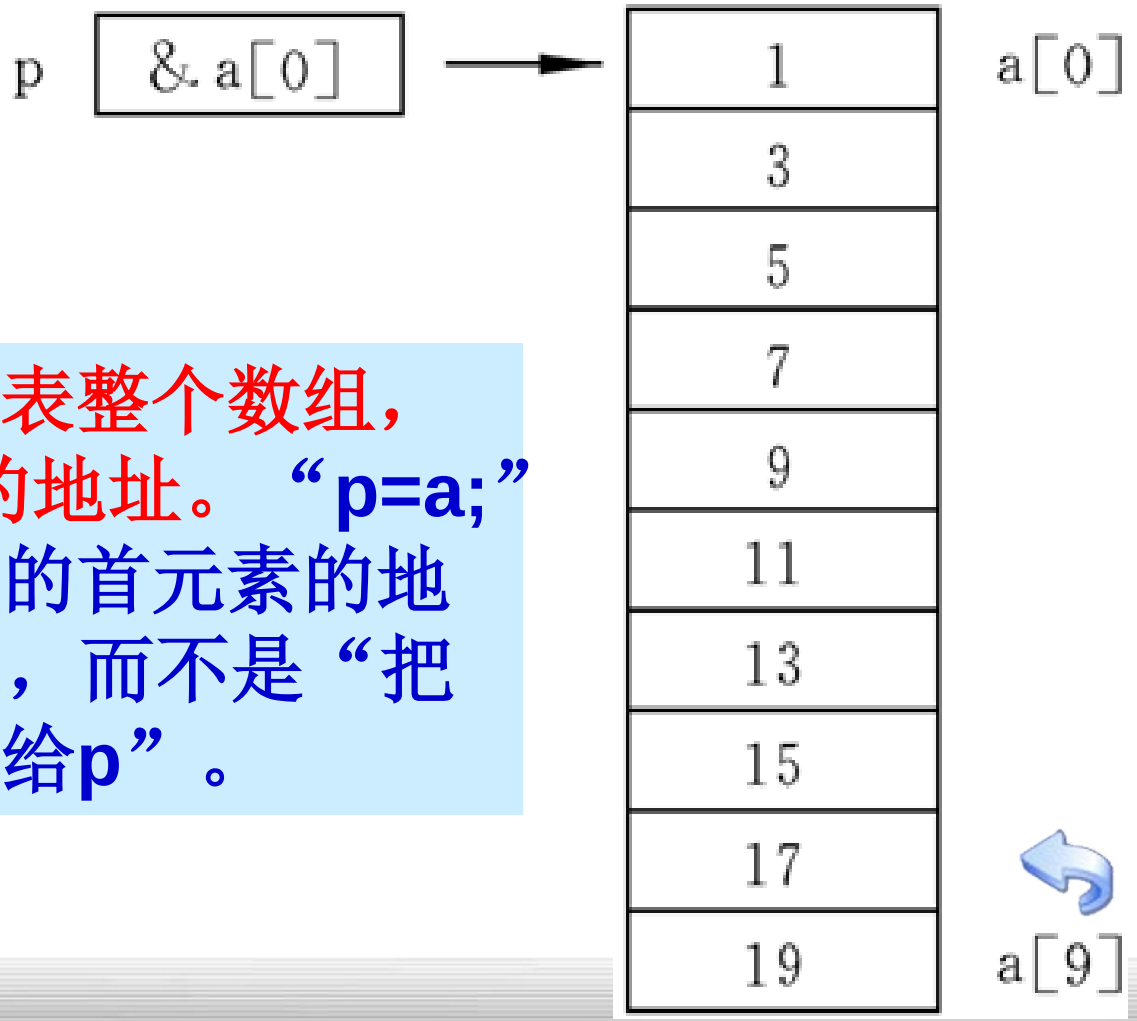
8.3.1 数组元素的指针

- 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

- 可以用一个指针变量指向一个数组元素

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

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



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”。

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。

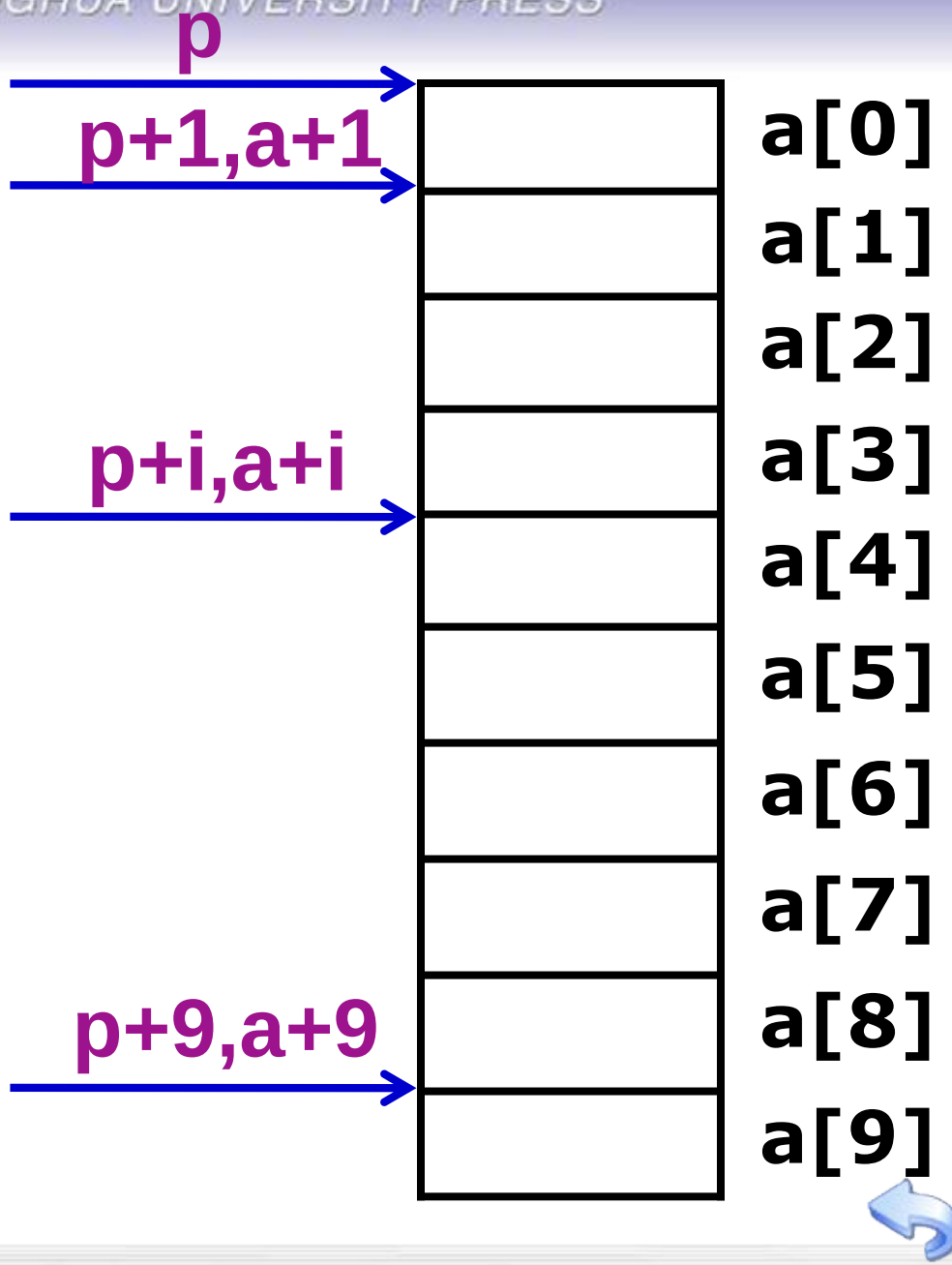
```
float a[10],*p=a;
```

假设a[0]的地址为2000，则

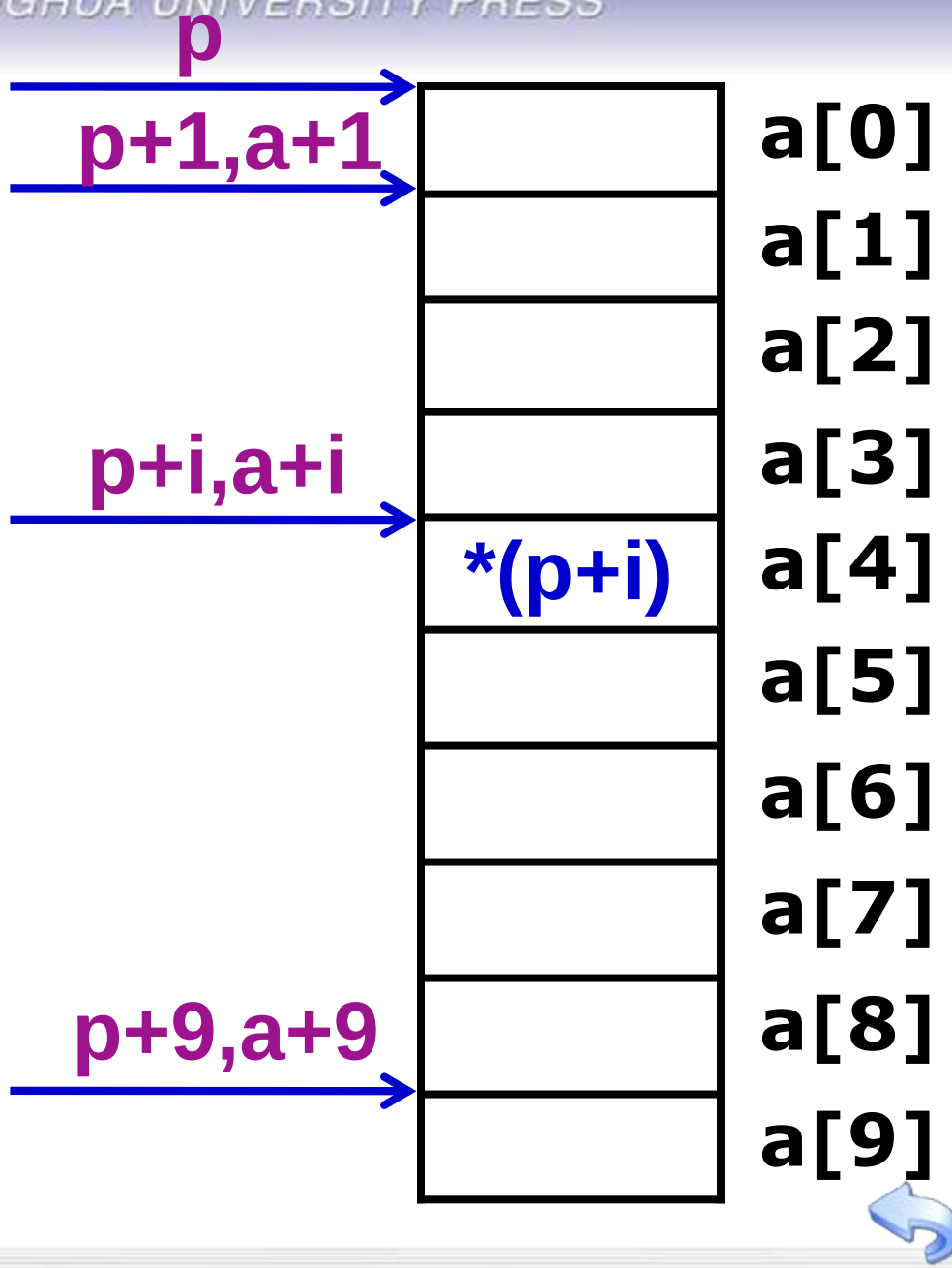
- p的值为2000
- p+1的值为2004
- P-1的值为1996

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



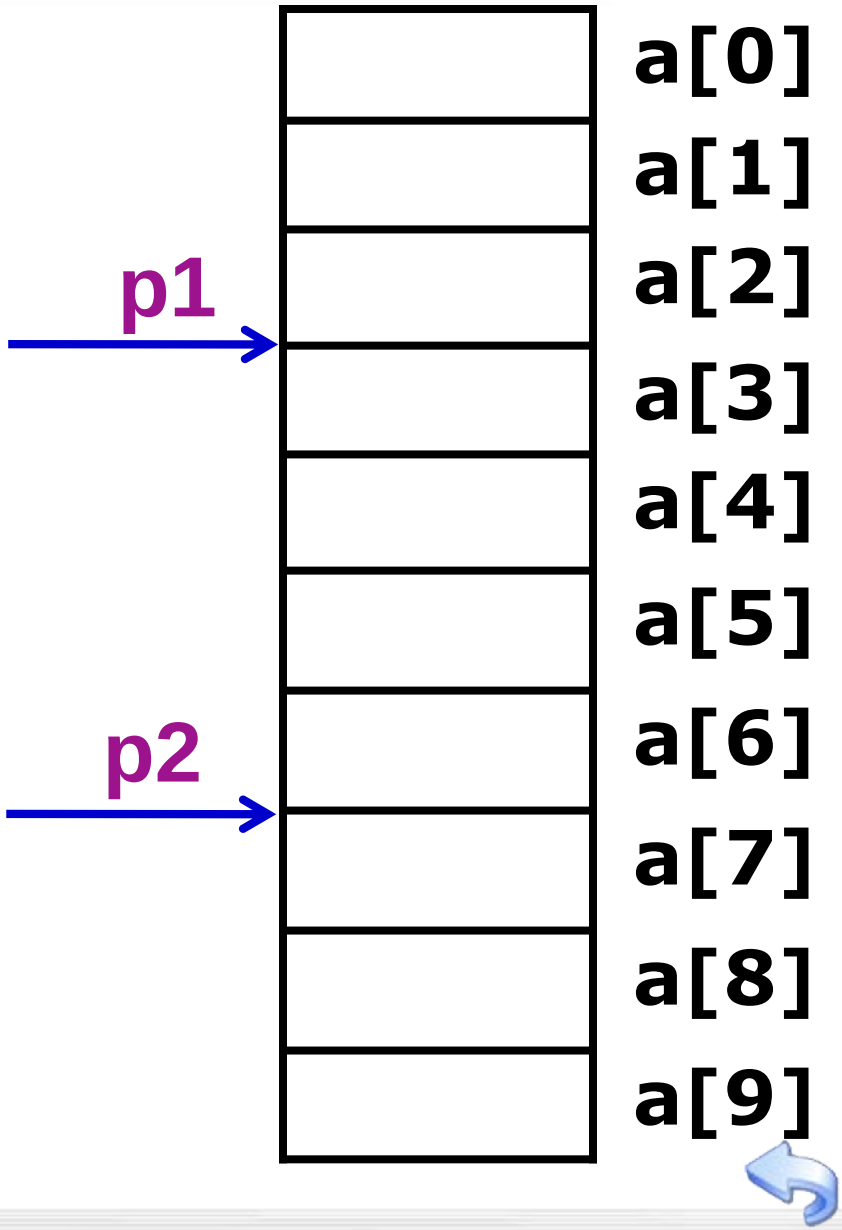
(3) *(p+i)或*(a+i)是p+i或a+i所指向的数组元素，即a[i]。



(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**

数组名表示数组的首地址，是常量

p++ p-- p+=2 a++ a-- a+=2 错

a-2 错 a+1 a+2 对

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[
   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]
的存储单元

因此执行此
循环出问题

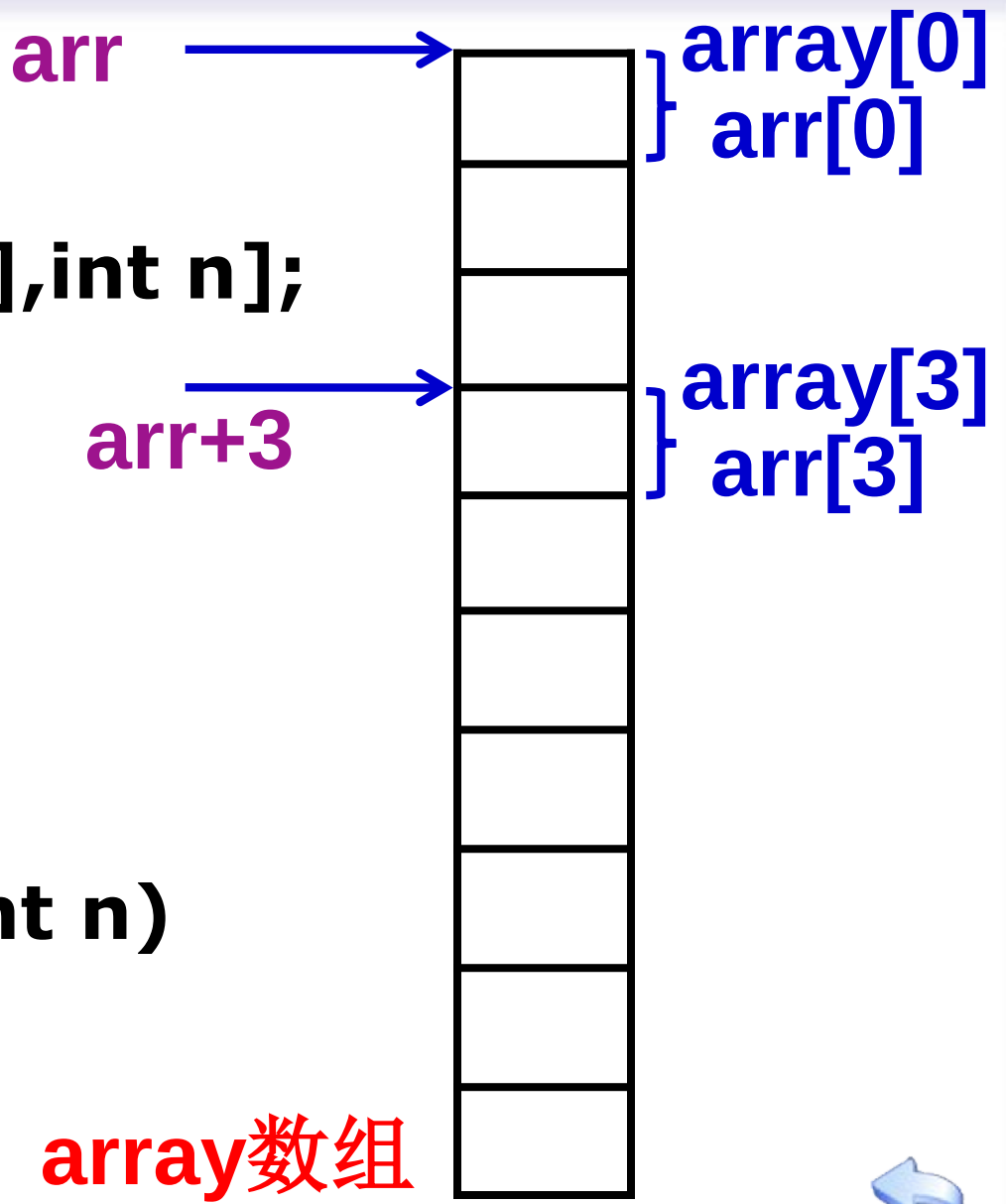
8.3.4 用数组名作函数参数

- 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)
{ ┆ }
```

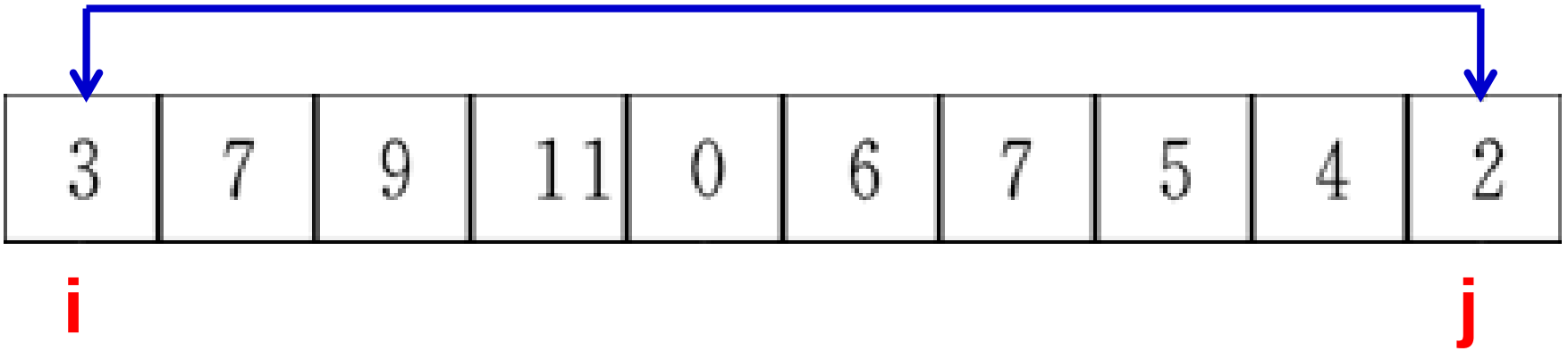


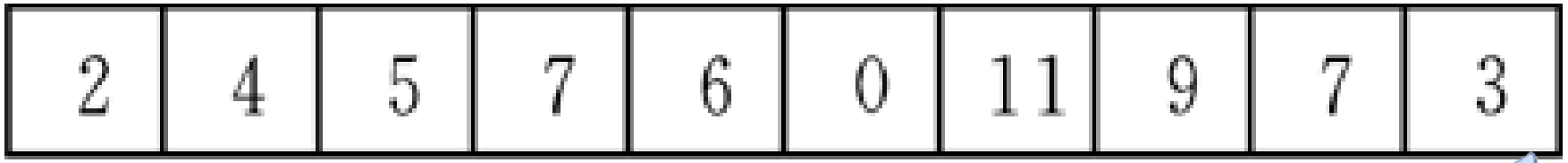
- 实参数组名是指针常量，但形参数组名是按指针变量处理
- 在函数调用进行虚实结合后，它的值就是实参数组首元素的地址
- 在函数执行期间，形参数组可以再被赋值

```
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

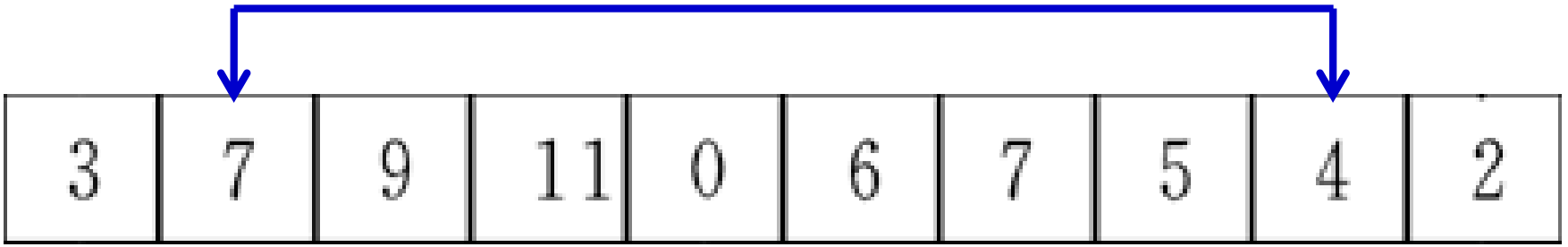
- 解题思路：将a[0]与a[n-1]对换，……
  将a[4]与a[5]对换。

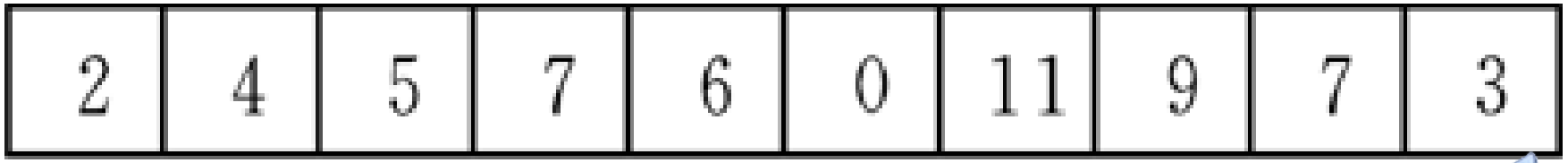




例8.8 将数组a中n个整数按相反顺序存放

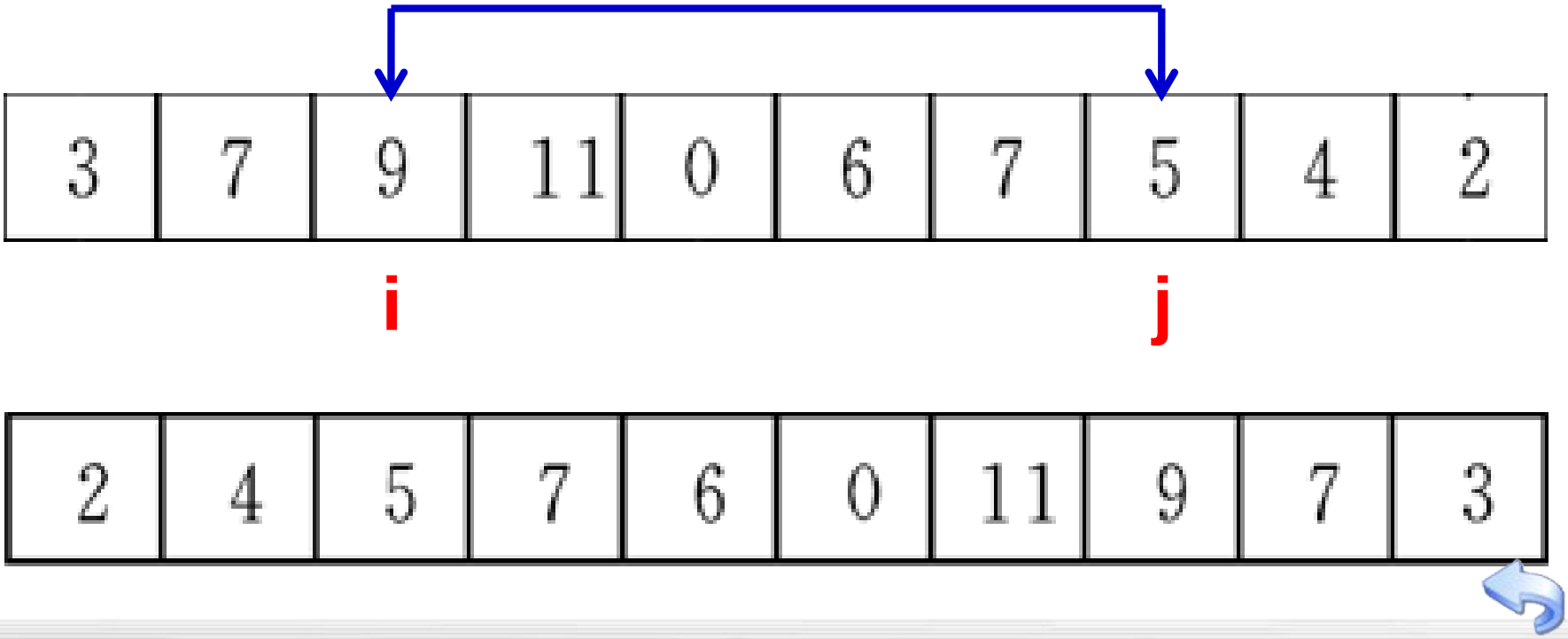
- 解题思路：将a[0]与a[n-1]对换，……
  将a[4]与a[5]对换。





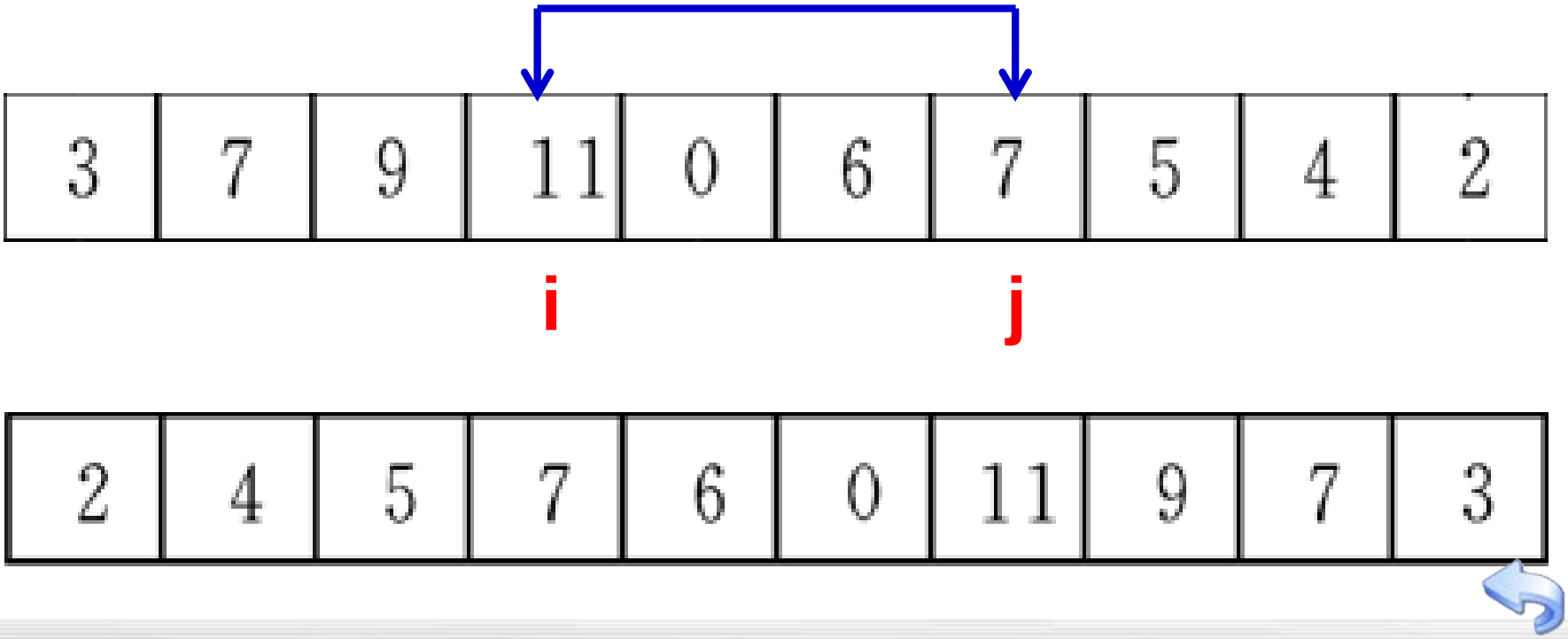
例8.8 将数组a中n个整数按相反顺序存放

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



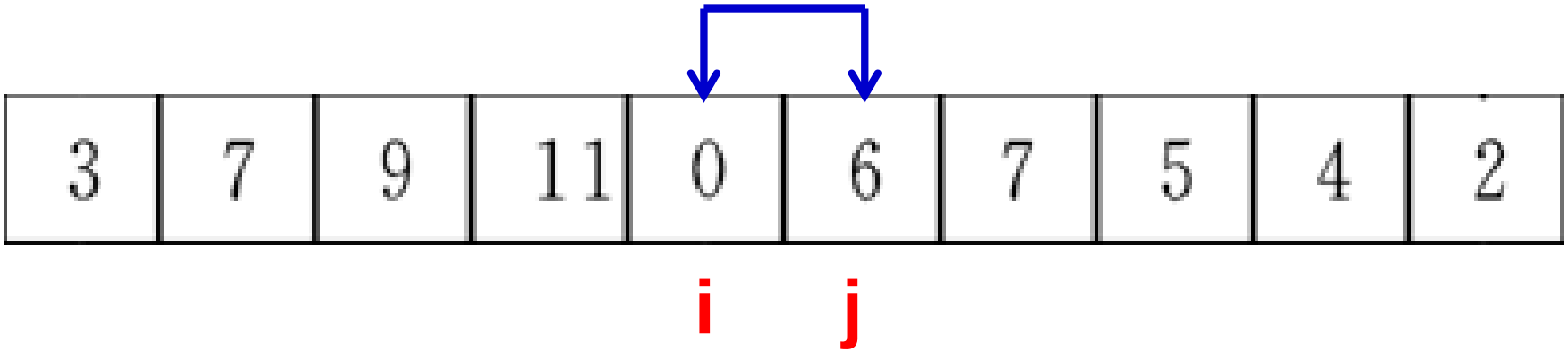
例8.8 将数组a中n个整数按相反顺序存放

- 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。



```
#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;
    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; }
  }
}
```

if (*(x+j)>*(x+k)) k=j;

{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
```

指针与数组

str[i] 等价 *(str+i)

P235

p++与(++p)区别

*p++与++(*p)区别

数组与指针

形参	实参
数组	数组
指针	指针
数组	指针
指针	数组

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

特别注意：字符串赋值、复制的方法

最容易出错的地方

```
char str1[10],str2[]="China";  对
str1="China";                  错
str1= str2;                    错
str1[0]=str2[0];               对
str1[1]='a';                   对
str1[1]=str2;                  错
strcpy(str1,str2);             对
strcpy(str1, "China");         对
```

定义初始化时，可用 = 其余不行

char c[6];

scanf(”%s”,c);　China↙

系统自动在**China**后面加一个**’\0’**

scanf(”%s”,c[0]); 错误！

注意：**%s** 对应数组名

scanf(”%c”,&c[0]); 对

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()
{ char string[]="I love China!";
   printf("%s\n",string);
   printf("%c\n",string[7]);
   return 0;
}
```

string ↓ ↓ string+7

```
I love China!
C
```

字符串常量：

1、放在双引号里面，如

“China”“Hello”

2、整个引号的内容作为指向该字符串存储位置的指针（即地址）

3、为常量，其值不可以被修改。

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;
}
```

string

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

```
I love China!
```

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

```
#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;
}
```

```
#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;
}
```

string

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

char *string="I love China!";
string="I am a student.";

string 为指针变量，指针变量的赋值注意什么？表示什么？

string 为指针变量，指针变量的赋值:
只能把地址赋值，基类型要一致。
指向的改变

- **char *str1="I love China!";** 对
- **char *str2;**
  str2="I love XMU"; 对
- **char *str3;**
  ***str3="I am a student";** 错
- **char a[80]="I am a student."** 对
- **char b[80]**
  b="I am a teacher." 错
- **char c[80]**
  strcpy(c,"hello"); 对

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];
    *(b+i)=*(a+i);
  *(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;
}
```

很重要，添加字符串结束标记

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

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

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

*(b+i)的赋值表示什么意思？

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
  student.",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;
}
```

注意：形参名与实参名相同时，需区分清楚

```
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.
```

(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

对 copy_string 函数还可作简化

1、将copy_string函数改写为

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

2、 copy_string函数的函数体还可改为

```
{
  while（（*to＋＋＝*from＋＋）！＝'\0'）;
}
```

3、copy_string函数的函数体还可写成

```
{
  while（*from！＝'\0'）
    *to＋＋＝*from＋＋;
    *to＝'\0';
}
```

4、上面的while语句还可以进一步简化为下面的while语句：

```
while（*to++=*from++）;
```

它与下面语句等价：

```
while（（*to++=*from++）!='\0'）;
```

将*from赋给*to，如果赋值后的*to值等于'\0'，则循环终止（'\0'已赋给*to）

5、函数体中while语句也可以改用for语句：

```
for（；（*to++=*from++）!=0；）;
```

或

```
for（；*to++=*from++；）;
```

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

很重要！

8.4.3 使用字符指针变量和字符数组的比较

char *a; scanf(“%s”,a); 错

char *a,str[10];
a=str;
scanf (“%s”,a); 对

scanf(“%s”,a);
表示对a指向的内容进行存储。
即 *a 的赋值，a需要有确定的指向

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，是常量)，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);
```

字符指针变量和字符数组的比较

- 指向方向的改变、指向内容的改变

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

指向方向的改变：　a+=2;　　对

　　b+=2;　　错

指向内容的改变：　a[0]='Y';　错

　　b[0]='H';　对

```
char *a=”I love China!”;
char b[50]=”hello”;
printf(“%s”,a);
printf(“%s”,b);
strcpy(b,a);
printf(“%c”,*a);
printf(“%c”,b[1]);
strcpy(a,b);
```

错

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);

```
#include <stdio.h>
int main()
{ float score[ ]={60,70,80,98,89,67,78};
  float  *fun(float pointer[],int n);
  float  *p;
  p=fun(score,7);
  printf("number=%5.2f\n",*p);
  return 0;
}
```

```
float  *fun(float pointer[ ],int n)
{  int i;
   float *p,*pt;
   p=pt=pointer;
   for(i=1;i<n;i++)
   {
      p++;
      if(*p>*pt)
      {
         pt=p;
      }
   }
   return(pt);
}
```

求最大值，并把其指针返回

指针变量的赋值

- 变量的地址，如 **p=&a;**
- 其他指针的值，如**p=p1;**
- 数组名，如 **int a[10], int *p=a;**
- 函数返回值，如 **p= fun(i,k);**

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

- 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**函数可以为一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**recalloc**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，另外用一个函放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;
```