

第七章

指南

• 7.1 地址和指针的基本概念

7.1地址和指针的概念

为了说清楚什么是指针，必须弄清楚数据在内存中是如何存储的，又是如何读取的。

内存区的每一个字节有一个编号，这就是“**地址**”。如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配内存单元。

1、按变量地址存取变量值的方式称为“**直接访问**”方式

例如：

```
printf ("%d", i);  
scanf ("%d", &i);  
k = i + j;
```

另一种存取变量值的方式称为“**间接访问**”的方式。即，将变量 *i* 的地址存放在另一个变量中。

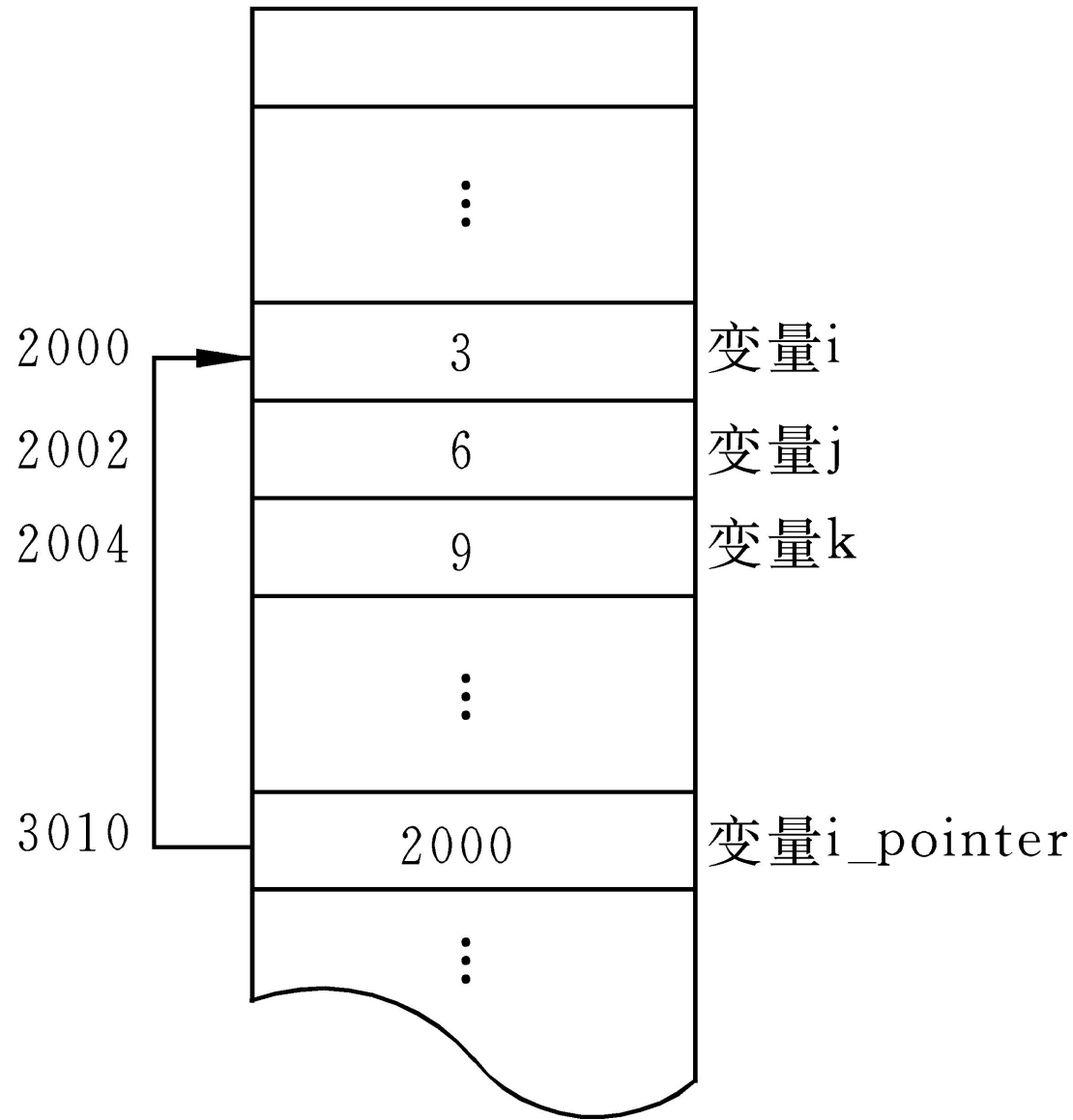
在 C 语言中，指针是一种特殊的变量，它是存放地址的。假设我们定义了一个指针变量 ***i_pointer*** 用来存放整型变量的地址，它被分配地址为(3010)、(3011)的两个字节。可以通过语句：

```
i_pointer = & i ;
```

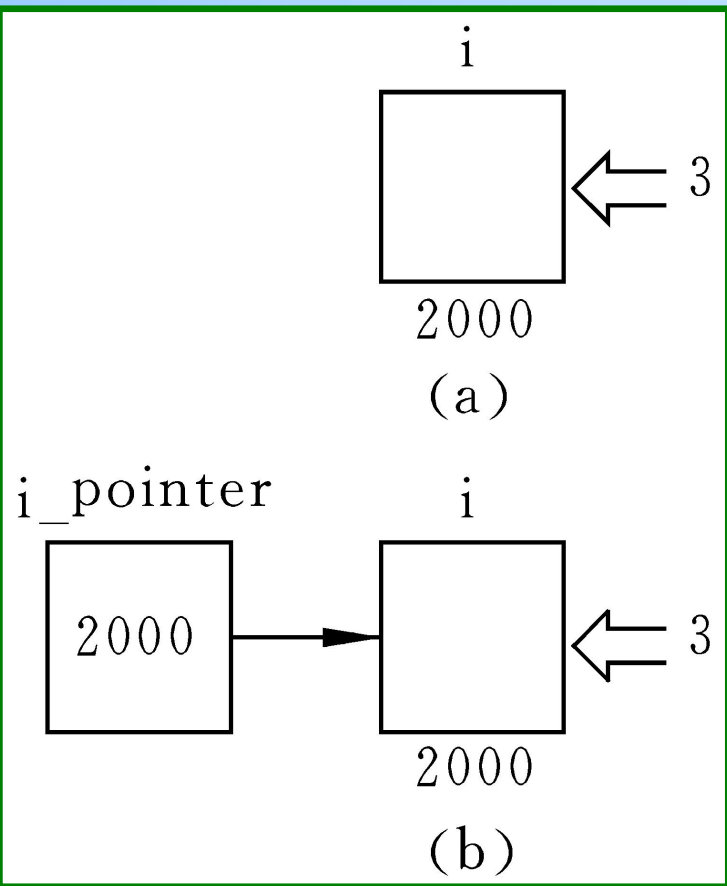
将 *i* 的地址(2000)存放到 *i_pointer* 中。这时， ***i_pointer*** 的值就是(2000)，即变量 *i* 所占用单元的起始地址。

要存取变量 *i* 的值，可以采用间接方式：先找到存放“*i* 的地址”的变量 ***i_pointer***，从中取出 *i* 的地址(2000)，然后到2000、2001 字节取出 *i* 的值（3）。

内存用户数据区



将3送到变量i所标识的单元中



将3送到变量i_pointer所指向的单元中

指针和指针变量的定义：

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”**。

上述的 **i_pointer** 就是一个指针变量。

指针变量的值（即指针变量中存放的值）是地址（即指针）。请区分“指针”和“指针变量”这两个概念。

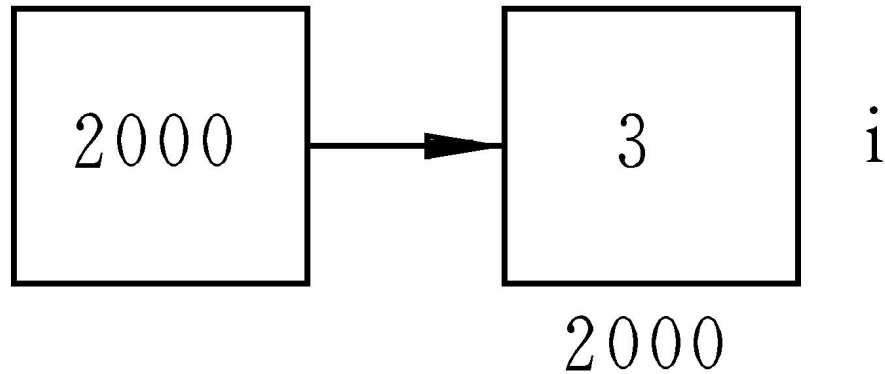
一个变量的地址称为该变量的指针。

一个变量用来存放另一个变量的地址，称为指针变量。

• 7.2 变量的指针和指向变量的指针变量

7.2 变量的指针和指向变量的指针变量

i_pointer * i_pointer



7.2.1 定义一个指针变量

定义指针变量的一般形式为

基类型 *指针变量名;

下面都是合法的定义：

`float *pointer_3 ;` // `pointer_3` 是指向float型变量的指针变量

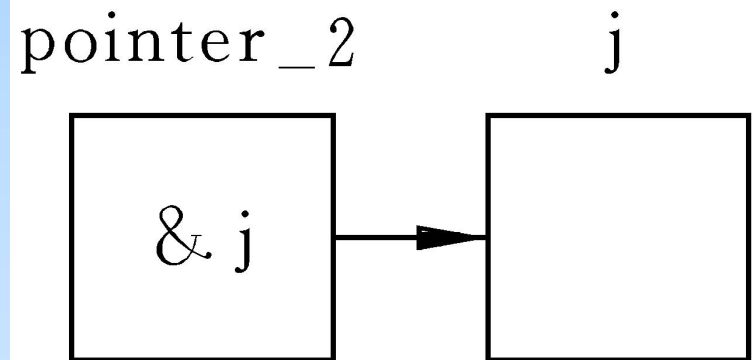
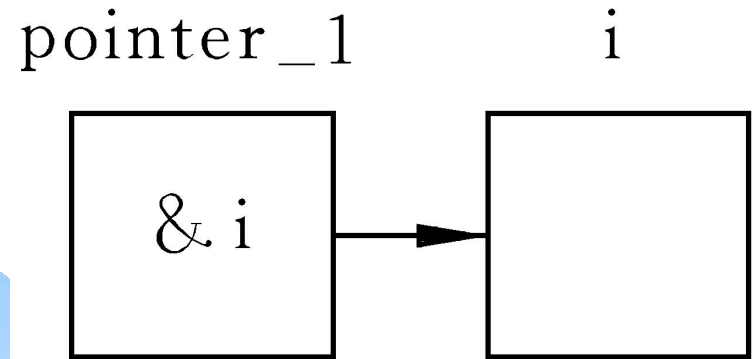
`char *pointer_4 ;` // `pointer_4` 是指向字符型变量的指针变量

可以用赋值语句使一个指针变量得到另一个变量的地址，从而使它指向一个该变量。

如：

`pointer_1 = & i ;`

`pointer_2 = & j ;`



在定义指针变量时要注意两点：

(1) 指针变量前面的 “*” ，表示该变量的类型为指针型变量。

例: `float *pointer_1;`

指针变量名是 `pointer_1` ，而不是 `* pointer_1` 。

(2) 在定义指针变量时必须指定基类型。

需要特别注意的是，只有整型变量的地址才能放到指向整型变量的指针变量中。

下面的赋值是错误的：

```
float a;
```

```
int * pointer_1;
```

```
pointer_1=&a; ❌  /* 将float型变量的地址放到指向整型变量的指
```

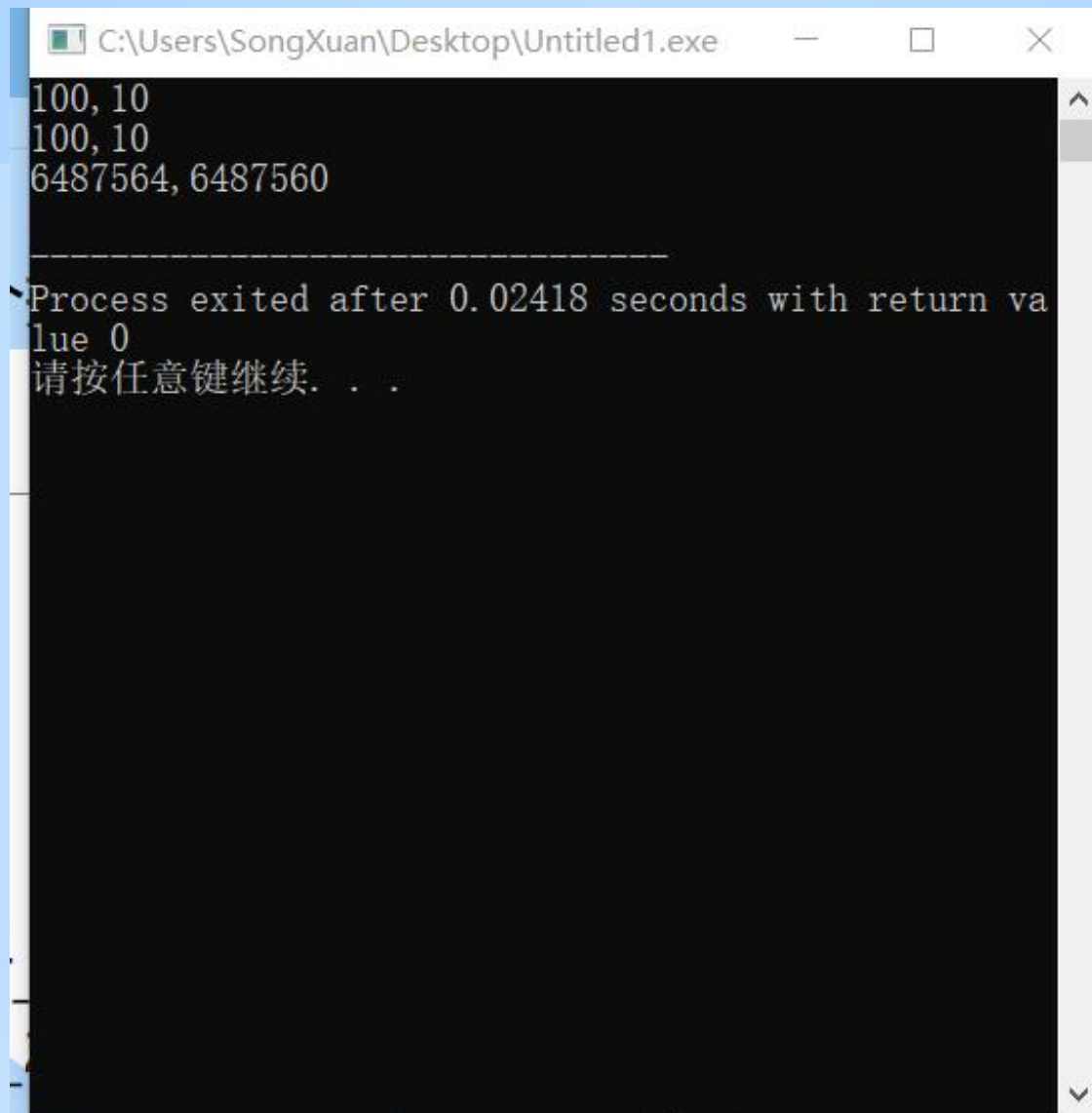
```
针变量中，错误 */
```

7.2.2 指针变量的引用

请牢记，指针变量中只能存放地址（指针），不要将一个整数（或任何其他非地址类型的数据）赋给一个指针变量。

例7.1 通过指针变量访问整型变量

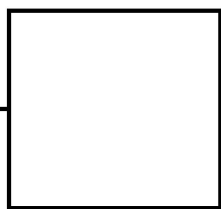
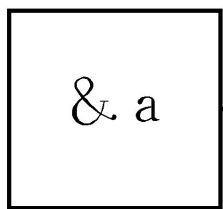
```
#include <stdio.h>
int main ( )
{ int a,b;
  int *pointer_1, *pointer_2;
  a=100;b=10;
  pointer_1=&a; /*把变量 a 的地址赋给pointer_1 */
  pointer_2=&b; /*把变量b的地址赋给pointer_2 */
  printf("%d,%d\n",a,b);
  printf("%d,%d\n",*pointer_1, *pointer_2);
  printf("%d,%d\n",pointer_1, pointer_2);
}
```



```
C:\Users\SongXuan\Desktop\Untitled1.exe
100, 10
100, 10
6487564, 6487560
-----
Process exited after 0.02418 seconds with return value 0
请按任意键继续. . .
```

pointer_1

a

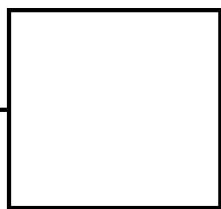
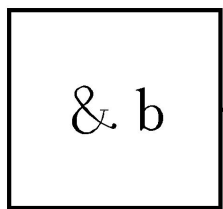


* pointer_1

作用是使pointer_1指向a

pointer_2

b



* pointer_2

作用是使pointer_2指向b

对 “&” 和 “*” 运算符说明：

如果已执行了语句 `pointer_1 = & a ;`

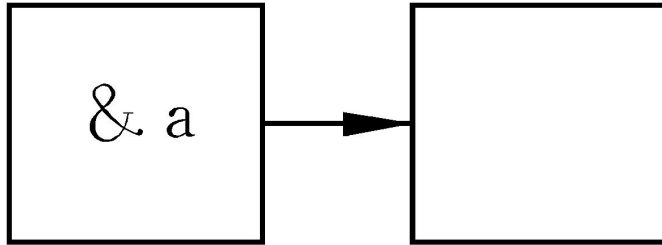
(1) `&* pointer_1` 的含义是什么？

“&” 和 “*” 两个运算符的优先级别相同，但按自右而左方向结合，因此先进行 `* pointer_1` 的运算，它就是变量 `a`，再执行 `&` 运算。因此，`&* pointer_1` 与 `& a` 相同，即变量 `a` 的地址。

如果有 `pointer_2 = &* pointer_1`；它的作用是将 `& a`（`a` 的地址）赋给 `pointer_2`，如果 `pointer_2` 原来指向 `b`，经过重新赋值后它已不再指向 `b` 了，而指向了 `a`。

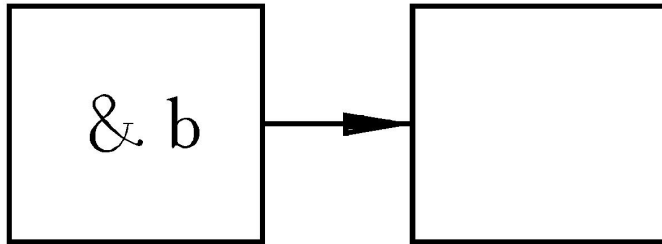
pointer_1

a



pointer_2

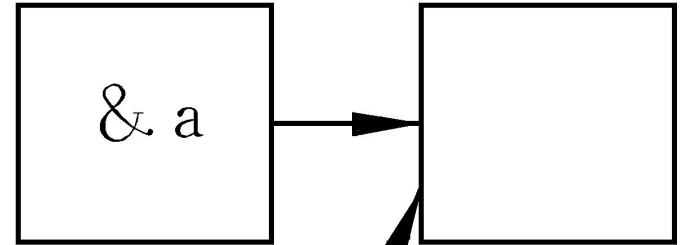
b



(a)

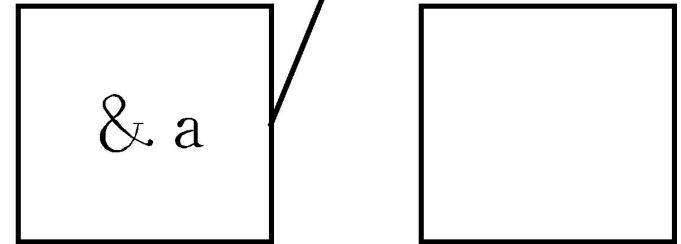
pointer_1

a



pointer_2

b



(b)

(2) $*\&a$ 的含义是什么？先进行 $\&a$ 运算，得 a 的地址，再进行 $*$ 运算。即 $\&a$ 所指向的变量，也就是变量 a 。 $*\&a$ 和 $*pointer_1$ 的作用是一样的，它们都等价于变量 a 。即 $*\&a$ 与 a 等价。

(3) $(*pointer_1)++$ 相当于 $a++$ 。注意括号是必要的，如果没有括号，就成为了 $*pointer_1++$ ，从附录可知： $++$ 和 $*$ 为同一优先级别，而结合方向为自右而左，因此它相当于 $*(pointer_1++)$ 。由于 $++$ 在 $pointer_1$ 的右侧，是“后加”，因此先对 $pointer_1$ 的原值进行 $*$ 运算，得到 a 的值，然后使 $pointer_1$ 的值改变，这样 $pointer_1$ 不再指向 a 了。

例7.2 输入a和b两个整数，按先大后小的顺序输出a和b。

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

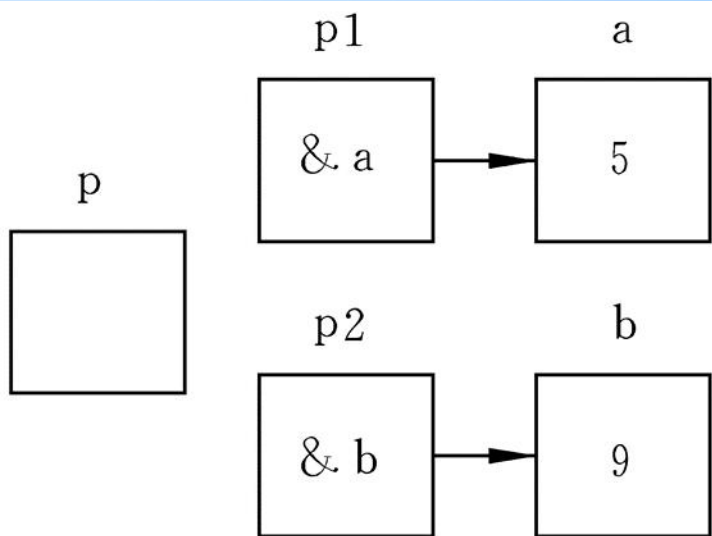
运行情况如下：

5, 9 ✓

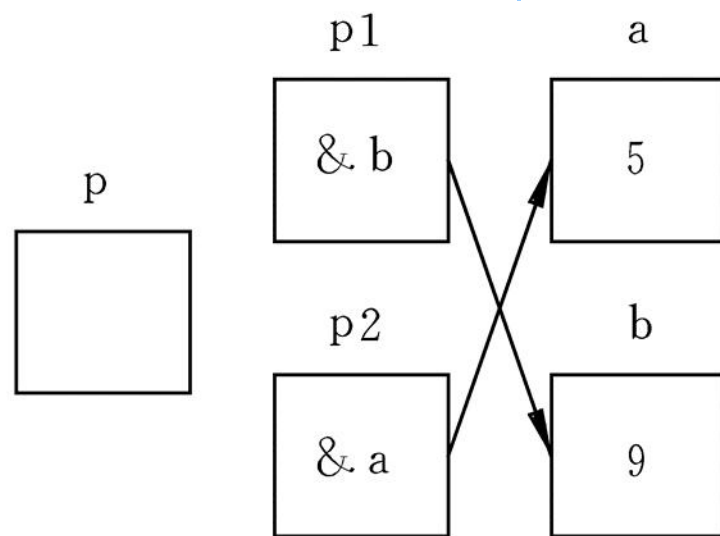
$a = 5, b = 9$

$\max = 9, \min = 5$

当输入 $a = 5, b = 9$ 时，由于 $a < b$ ，将 $p1$ 和 $p2$ 交换。交换前的情况见图 (a)，交换后见图 (b)。



(a)



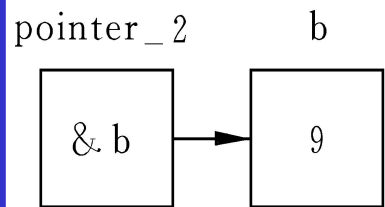
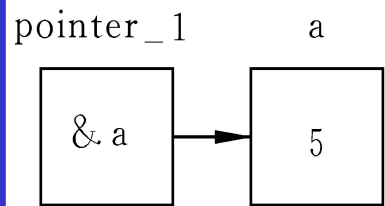
(b)

7.2.3 指针变量作为函数参数

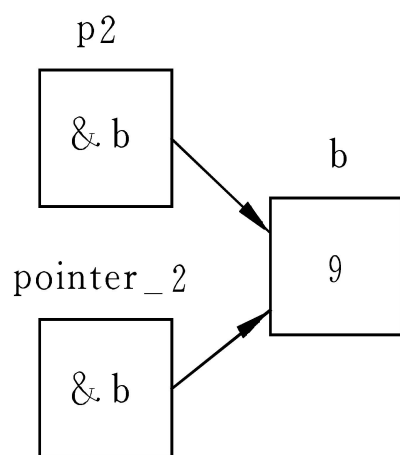
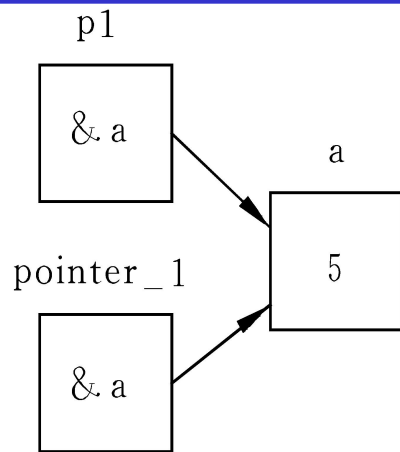
例7.3 对输入的两个整数按大小顺序输出

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

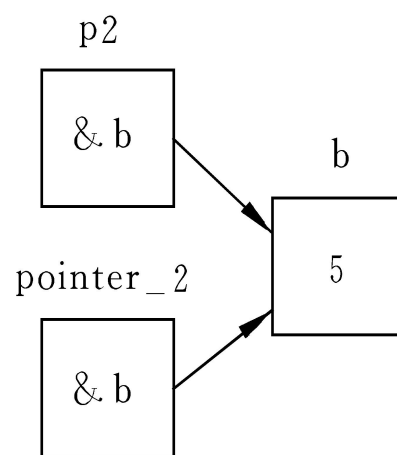
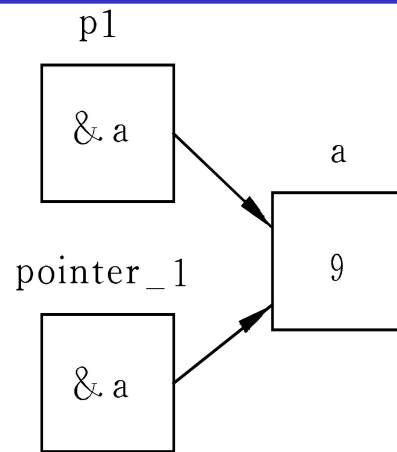
void swap(int *pt1, int *pt2)
{
    int temp;
    temp=*pt1;
    *pt1=*pt2;
    *pt2=temp;
}
```



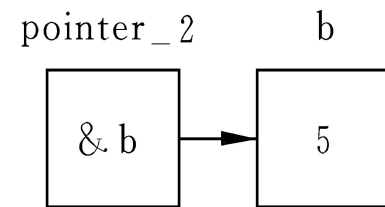
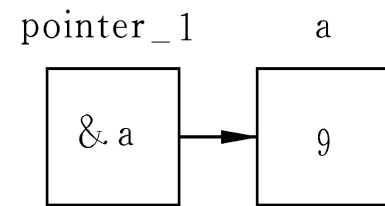
(a)



(b)



(c)



(d)

例7.4 输入 a 、 b 、 c 3个整数，按大小顺序输出

```
#include <stdio.h>

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

int main()
{
    int  a,b,c,*p1,*p2,*p3;
    scanf("%d%d%d",&a, &b, &c);
    p1=&a;p2=&b;p3=&c;
    exchange (p1,p2,p3);
    printf("\n%d,%d,%d\n",a,b,c);
}
```

```
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;
```

```
}
```

• 7.3 数组与指针

7.3 数组与指针

数组元素的指针就是数组元素的地址。

一个变量有地址，一个数组包含若干元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。

指针变量既然可以指向变量，当然也可以指向数组元素（把某一元素的地址放到一个指针变量中）。

7.3.1 指向数组元素的指针

定义一个指向数组元素的指针变量的方法，与以前介绍的指向变量的指针变量相同。

例如：

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

(定义a为包含10个整型数据的数组)

```
int *p; (定义p为指向整型变量的指针变量)
```

应当注意，如果数组为int型,则指针变量的基类型亦应为int型。

对该指针变量赋值：

p = &a[0];

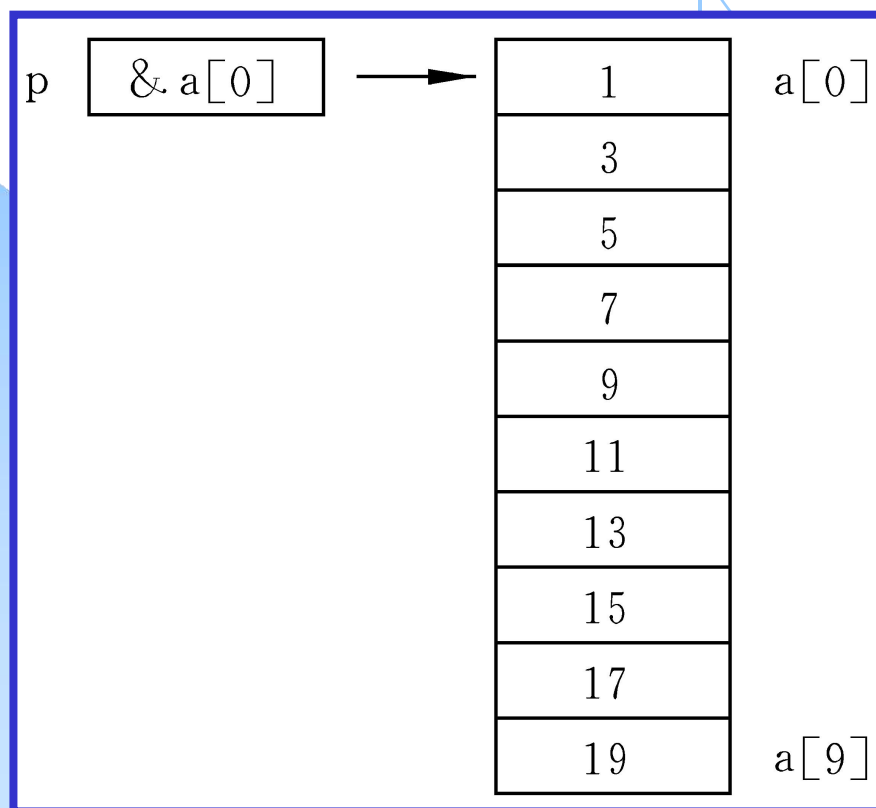
把**a[0]**元素的地址赋给指针变量 p。

即：使 p 指向 a 数组的第 0 号元素，如图：

数组名代表数组首元素地址，
因此下面的语句是等价的。

p = &a[0];

p = a;



7.3.1在引用数组元素时指针的运算

对数值型数据进行算术运算（加、减、乘、除）的目的和意义很清楚。

指针型数据的算术运算的含义是什么？

指针即地址

对地址的加、减、乘、除的意义何在？

7.3.1在引用数组元素时指针的运算

当指针指向数组元素时，可以通过对指针的加减运算实现指针指向的移动。

在指针已经指向一个数组元素时，可以对指针进行一下运算。

加一个整数 (+, 或者 +=), $p+1$

减一个整数 (-, 或者 -=), $p-1$;

自加运算, 如 $p++$, $++p$

自减运算, 如 $p--$, $--p$

两个指针相减, 如 $p1-p2$ (只有同一数组时才有意义)

(1) 如果指针变量已经指向数组的某一元素，则：

$p+1$ 指向同一数组的下一元素；

$p-1$ 指向同一数组的上一个元素；

注意： $p+1$ 并不是将 p 的值简单 $+1$ ，而是加上一个数组元素所占的字节数。如：

数组元素是 `float` 型，每个元素占 4 个字节，则 $p+1$ 所代表的是 $p+1 \times d$, d 为一个数组元素所占的字节数。

这就是定义指针变量时，必须定义基类型的原因。

(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，或者说指向a数组序号为i的元素。

(3) $*(p+i)$ 或者 $*(a+i)$

是 $p+i$ 或者 $a+i$ 所指向的数组元素，即 $a[i]$ 。如：

$*(p+5)$ 或者 $*(a+5)$ 就是 $a[5]$

即 $*(p+5)$ ， $*(a+5)$ 和 $a[5]$ 这三者是等价的。

实际上，在编译时，对数组元素 $a[i]$ 就是按照 $*(a+i)$ 处理的，即按数组首元素地址加上相对应位偏移量得到要找的元素的地址，然后找出单元中的内容。

(4) 如果p1和p2都指向同一数组中的元素，如执行 $p2 - p1$ ，结果是 $p2 - p1$ 的值（两个地址之差）除以数组元素的长度。

p2 指向 a[5]，地址为2020

p1指向 a[3]，地址为2012

所以 $p2 - p1$ 结果为 $(2020 - 2012) / 4 = 2$ 。

该结果的意思是p2和p1所指向元素之间差2个元素。

注：两个地址不能相加，即 $p1 + p2$ 没有实际意义。

7.3.2 通过指针引用数组元素

引用一个数组元素，可以用：

(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 $p = a$ 。

定义指针变量时可以对其进行初始化，如：

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

它等效于：

```
int *p;
```

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

例7.5 输出数组中的全部元素

假设有一个整型数组 a ,
有10个元素。要输出各
元素的值有三种方法:

(1)下标法

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

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

通过数组名和元素序号计算
元素地址，再找到该元素

(3) 用指针变量指向数组元素

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

例7.6 通过指针变量输出 a 数组的10个元素。

有人编写出以下程序：

```
#include <stdio.h>

int main()
{  int *p,i,a[10];
    p=a;
    for(i=0;i<10;i++ )
        scanf("%d",p++);
    printf("\n");
    for(i=0;i<10;i++,p++ )
        printf("%d",*p);
}
```

希望把输入的10个数打印出来，但是运行的结果却是：

1 2 3 4 5 6 7 8 9 0✓

22153 234 0 0 30036 25202 11631 8259 8237 28483

显然输出的数值并不是 a 数组中各元素的值，
本质问题在于指针没有回到数组的起始地址

解决办法，在第二个for循环前加赋值语句：**p = a ;**

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

- 假设p开始时指向数组a的首元素，即 $p=a$

(1) $p++$;

$*p$ 得到的是 $a[1]$ 的值

(2) $*p++$;

$*$ 和 $++$ 优先级相同，从右向左结合。

它等价于 $*(p++)$

(3) $*(p++)$ 和 $*(++p)$

$*(p++)$ 先取 p 的值，然后使 p 加1，结果是 $a[0]$ 的值。

$*(++p)$ 先使 $p+1$ ，然后再取 $*p$ ，结果是 $a[1]$ 的值。

- 假设p开始时指向数组a的首元素，即 $p=a$

(4) $++ (*p)$;

p 所指向的元素值加1。即元素 $a[0]$ 的值加1，而不是指针p的值加1。

(5) 如果p当前指向a数组中第i个元素 $a[i]$, 则:

- * $(p--)$ 相当于 $a[i--]$, 先对p继续进行*运算, 再使p自减;
- * $(++p)$ p相当于 $a[++i]$, 先使p自加, 再进行*运算;
- * $(--p)$ p相当于 $a[--i]$, 先使p自减, 再进行*运算;

7.3.3 用数组名作函数参数

用数组名作函数的参数，如：

```
f(int arr[],int n); //函数声明
```

```
int main()
```

```
{
```

```
    int array[10];
```

```
        |
```

```
    f(array,10); //函数调用
```

```
        |
```

```
}
```

```
void f(int arr[ ],int n)
```

```
{
```

```
    |
```

```
}
```

f (int arr[], int n)

在编译时是将arr按指针变量处理的，即：

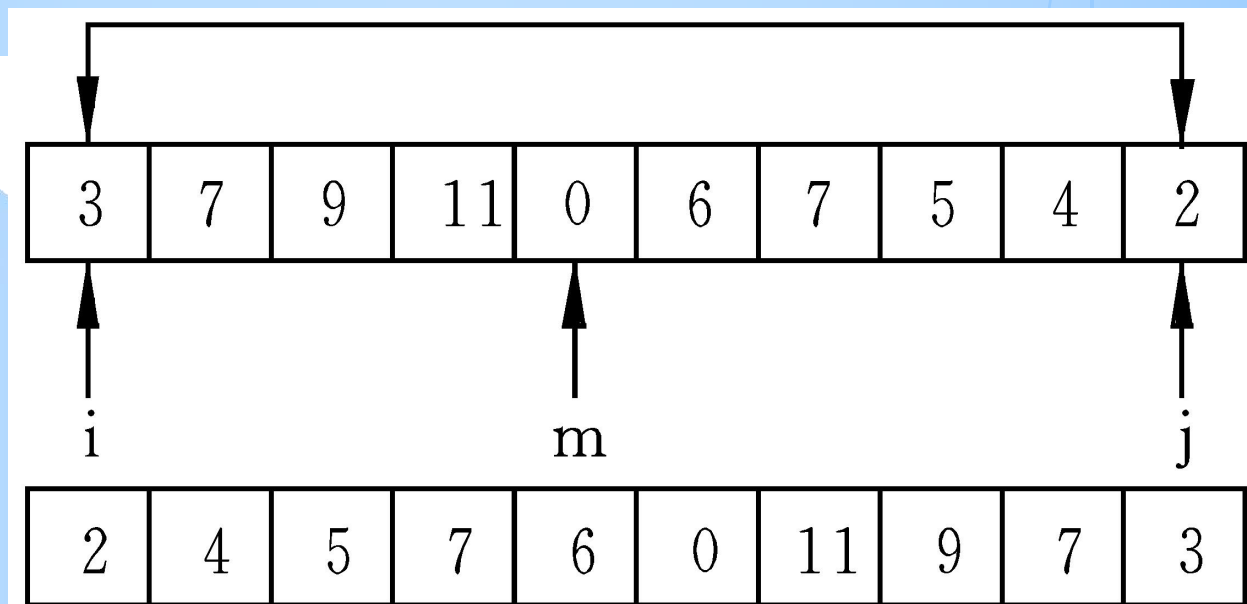
将函数f首部写成

f (int *arr, int n)

以上两种写法是等价的。

需要说明的是：C语言调用函数时虚实结合的方法都是采用“值传递”方式，当用**变量名作为函数参数**时传递的是变量的值，当用**数组名作为函数参数**时，由于数组名代表的是数组首元素地址，因此传递的值是地址，所以要求形参为指针变量。

例7.7 将数组 a 中 n 个整数按相反顺序存放.



解题思路: $a[0]$ 和 $a[n-1]$ 兑换,

$a[1]$ 和 $a[n-2]$

直到 $a[\text{int}(n-1)/2]$ 和 $a[\text{int}(n-1)/2-1]$ 对换。

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

```
void inv(int x[ ],int n);
```

```
int main()
```

```
{
```

```
    int i,a[10]={3,7,9,11,0, 6,7,5,4,2};
```

```
    printf("The original array:\n");
```

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

```
        printf ("%d,",a[i]);
```

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

```
    inv (a,10);
```

```
    printf("The array has been inverted:\n");
```

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

```
        printf ("%d,",a[i]);
```

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

```
}
```

```
void inv(int x[ ],int n)  /*形参x是数组名*/
```

```
{  
    int temp,i,j,m=(n-1)/2;
```

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

```
    {
```

```
        j=n-1-i;
```

```
        temp=x[i];
```

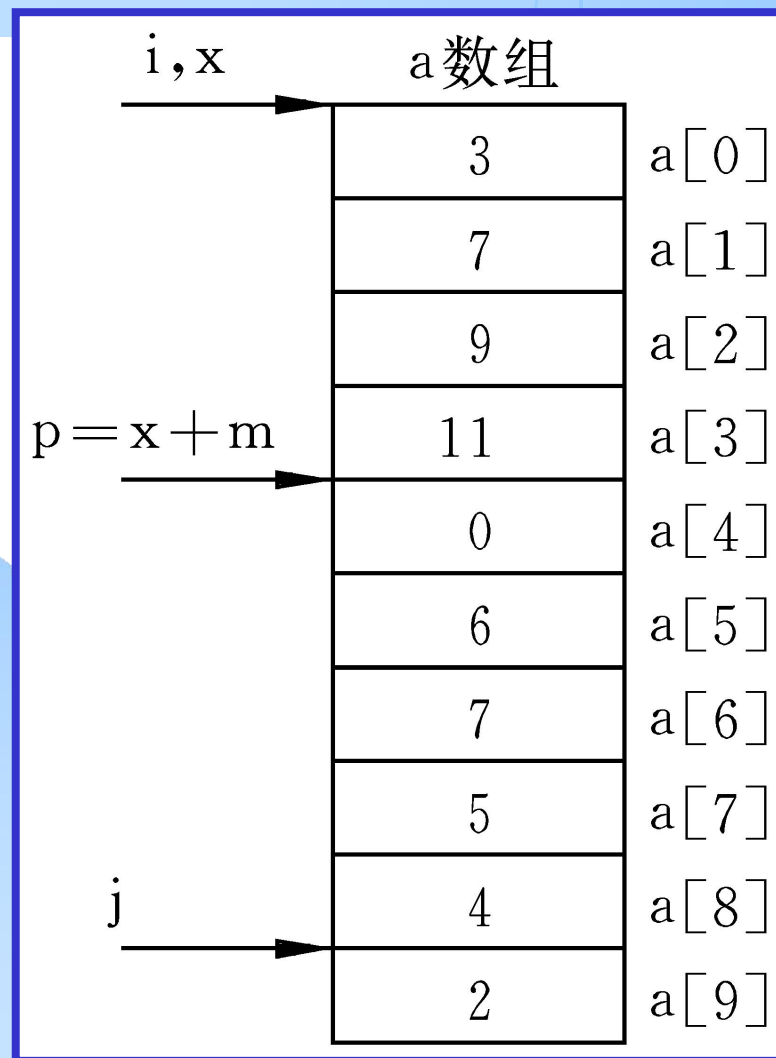
```
        x[i]=x[j];
```

```
        x[j]=temp;
```

```
    }
```

```
    //return;
```

```
}
```



运行情况如下：

The original array:

3 , 7 , 9 , 1 1 , 0 , 6 , 7 , 5 , 4 , 2

The array has been inverted:

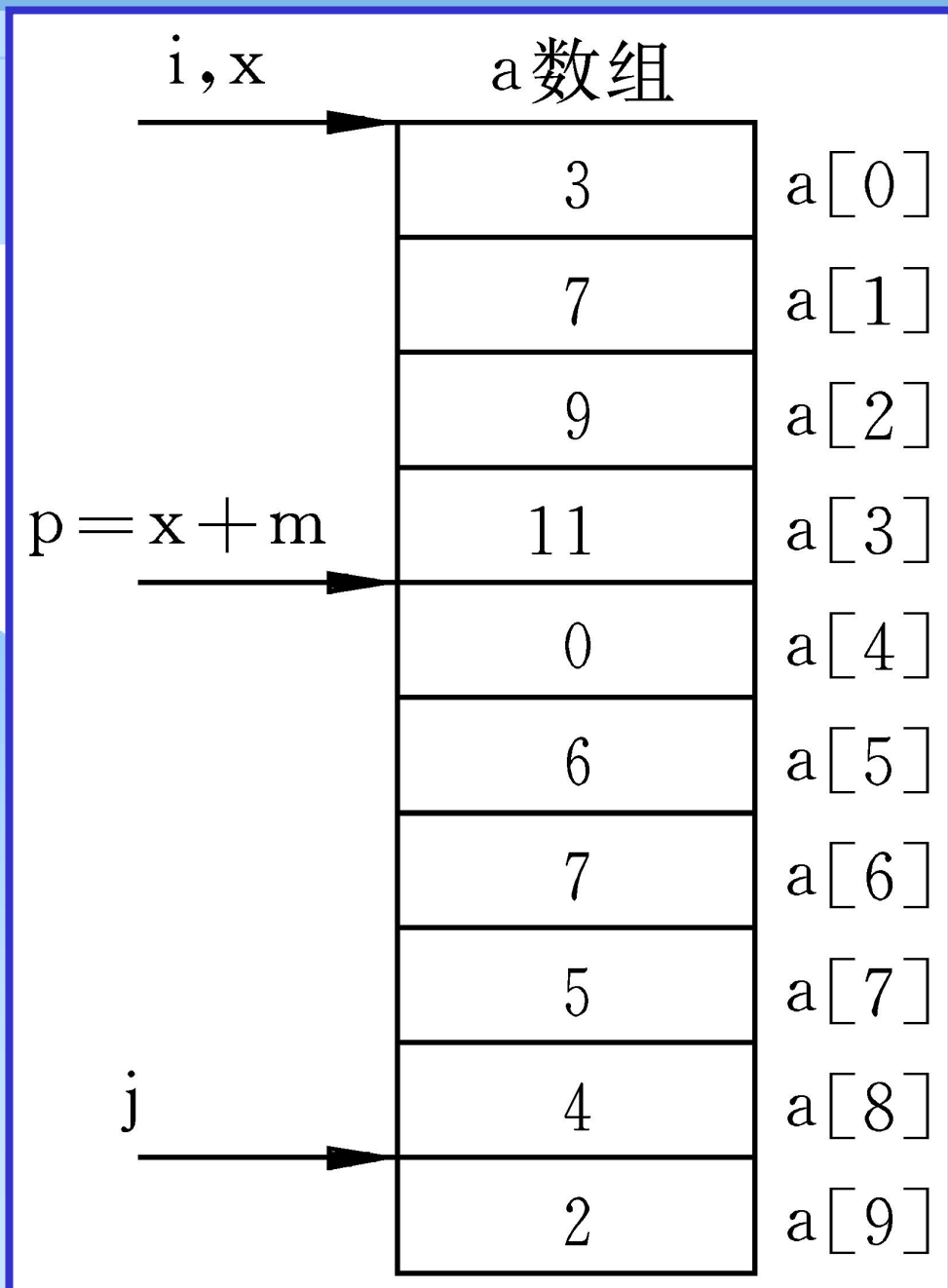
2 , 4 , 5 , 7 , 6 , 0 , 1 1 , 9 , 7 , 3

将inv中的形参x改成指针变量。

相应的实参仍为数组名a，即数组a首元素的地址，将它传给形参指针变量x，这时x就指向a[0]。

$x+m$ 是 $a[m]$ 元素的地址。

设i和j以及p都是指针变量，用它指向有关元素。i的初值为x，j的初值是 $x+n-1$ ，见右图：



对程序进行改动，将函数inv
中的形参 x 改成指针变量。

```
#include <stdio.h>

void inv(int *x,int n);

int main()
{
    int i,a[10]={3,12,9,11,0, 6,7,5,4,8};
    printf("The original array:\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");

    inv (a,10);

    printf("The array has been in verted:\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");
}
```

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

```
{
```

```
    int *p,m,temp,*i,*j, m=(n-1)/2;
```

```
    i=x;
```

```
    j=x+n-1;
```

```
    p=x+m;
```

```
    for(;i<=p;i++,j--)
```

```
    {
```

```
        temp=*i;
```

```
        *i=*j;
```

```
        *j=temp;
```

```
    }
```

```
    return;
```

```
}
```

总之：如果有一个实参数组，想在函数中改变此数组中的元素的值，实参与形参的对应关系有以下 4 种情况：

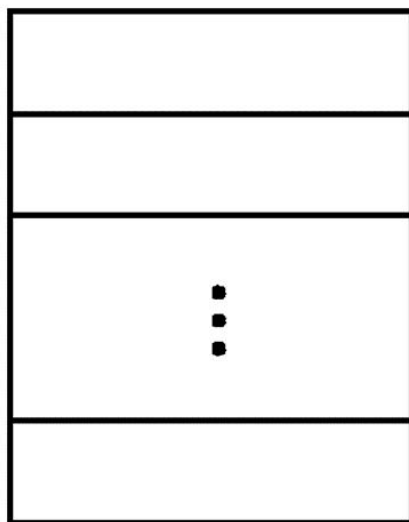
- (1) 形参和实参都用数组名；
- (2) 实参用数组名，形参用指针变量；
- (3) 实参形参都用指针变量；
- (4) 实参为指针变量，形参为数组名。

(1) 形参和实参都用数组名，如：

```
int main ()  
{ int a[10];  
  ...  
  f(a,10);  
}
```

```
int f(int x[],int n)  
{  
  ...  
}
```

数组a, x



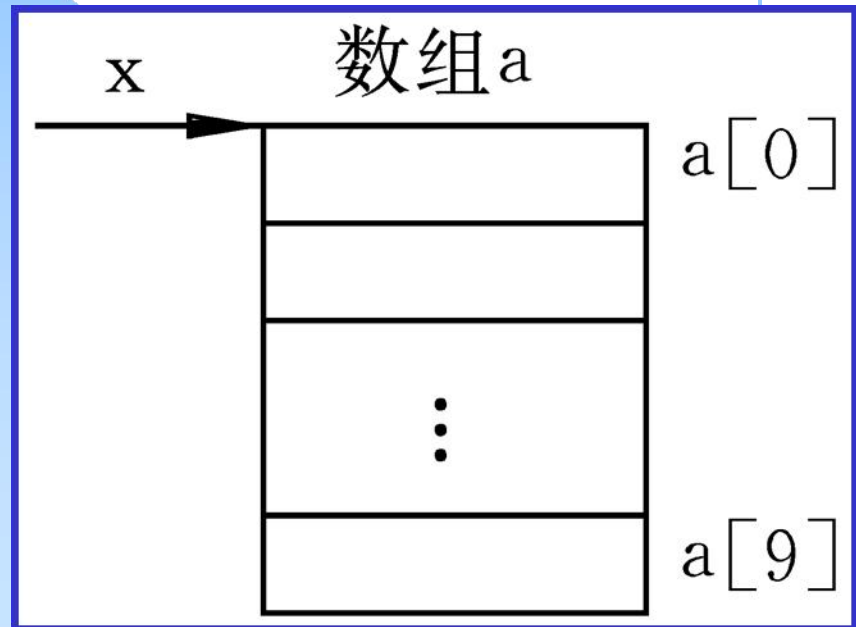
$a[0], x[0]$

$a[9], x[9]$

(2) 实参用数组名，形参用指针变量。如：

```
int main ()  
{int a[10];  
    ...  
    f(a,10);  
}
```

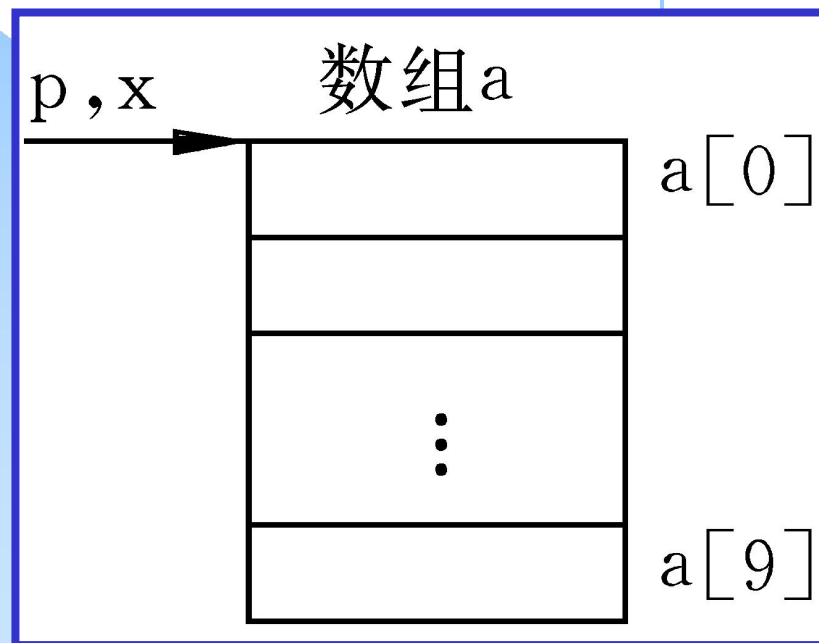
```
int f (int *x, int n)  
{  
    ...  
}
```



(3)实参形参都用指针变量。例如:

```
int main ()  
{int a [10] , *p=a;  
    ...  
    f (p, 10) ;  
}
```

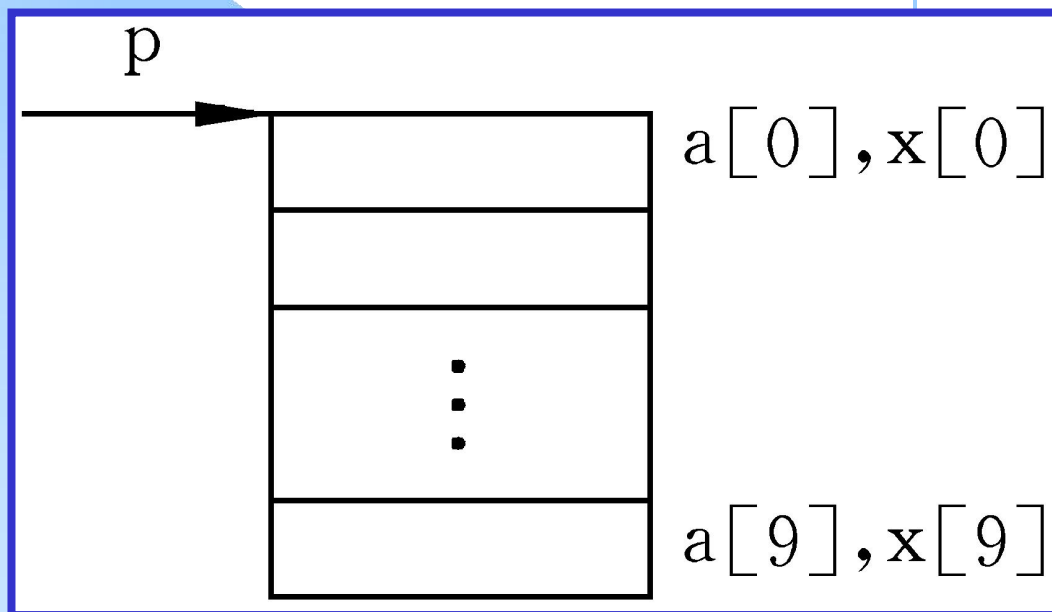
```
int f (int *x , int n)  
{  
    ...  
}
```



(4) 实参为指针变量，形参为数组名。如：

```
int main()  
{int a[10],*p=a;  
    ...  
    f(p,10);  
}
```

```
int f(int x[],int n)  
{  
    ...  
}
```



例7.8 用实参指针变量改写例7.7

```
#include <stdio.h>
void inv(int *x,int n);
int main()
{
    int i,arr[10],*p=arr;
    printf("The original array:\n ");
    for(i=0;i<10;i++,p++)
        scanf("%d",p);
    printf("\n");
    p=arr;
    inv(p,10); /* 实参为指针变量 */
    printf("The array has been inverted :\n");
    for(p=arr;p<arr+10;p++)
        printf("%d",*p);
    printf("\n");
}
```



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

例7.9 用选择法对10个整数按由大到小顺序排序

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

```
void sort(int x[ ],int n);
```

```
int main()
```

```
{
```

```
    int *p,i,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++;}
```

```
}
```

```
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;        }
```

```
    }
```

```
}
```

程序思路

7.3.4 多维数组与指针

用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。但多维数组的指针比一维数组的指针要复杂一些。

1. 多维数组元素的地址

多维数组的性质：可以认为二维数组是“数组的数组”，例：

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

则二维数组a是由3个一维数组所组成的。设二维数组的首行的首地址为2000，则

a	a[0]	=	1	3	5	7
	a[1]	=	9	11	13	15
	a[2]	=	17	19	21	23

注：

- 1) a是二维数组名；
- 2) a数组包含了3个行元素，即a[0],a[1],a[2]；
- 3) 每一个行元素又是一个一维数组，它包含4个元素（4个列元素）

因此二维数组可以看做是**数组的数组**。

a					
a[0]	=	1	3	5	7
a[1]	=	9	11	13	15
a[2]	=	17	19	21	23

a	a 数组		
(2000)		a[0]	
a + 1		a[1]	
(2008)			
a + 2		a[2]	
(2016)			

从二维数组的角度来看：行表示

- 1) **a**代表的首元素不是一个简单的整型元素，而是4个整型元素所组成的一维数组，因此**a**代表首行的起始地址。a[0][0]
- 2) a+1指向a[1]，a[1]行的起始地址, a[1][0];
a+2指向a[2]，a[2]行的起始地址, a[2][0];

	$a[0]$	$a[0]+1$	$a[0]+2$	$a[0]+3$
a	2000 1	2002 3	2004 5	2006 7
$a+1$	2008 9	2010 11	2012 13	2014 15
$a+2$	2016 17	2018 19	2020 21	2022 23

从二维数组的角度来看：列表示

$a[0]+\mathbf{0}$:表示该一维数组中序列号为0的元素的地址, $a[0][0]$

$a[0]+\mathbf{1}$:表示该一维数组中序列号为1的元素的地址, $a[0][1]$

$a[0]+\mathbf{2}$:表示该一维数组中序列号为2的元素的地址, $a[0][2]$

$a[0]+\mathbf{3}$:表示该一维数组中序列号为3的元素的地址, $a[0][3]$

二维数组a的有关指针

表 示 形 式	含 义	地 址
a	二维数组名，指向一维数组a[0]，即0行首地址	2000
a[0] , *(a+0) , *a	0行0列元素地址	2000
a+1, &a [1]	1行首地址	2008
a[1], *(a+1)	1行0列元素a[1][0]的地址	2008
a[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2] 的地址	2012
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a [1] [2] 的值	元素值为 13

表示形式	含 义	地 址
a	二维数组名,指向一维数组 a[0],即 0 行首地址	2000
a[0], *(a+0), *a	0 行 0 列元素地址	2000
a+1, &a[1]	1 行首地址	2008
a[1], *(a+1)	1 行 0 列元素 a[1][0]的地址	2008
a[1]+2, *(a+1)+2, &a[1][2]	1 行 2 列元素 a[1][2]的地址	2012
*(a[1]+2), *(*(a+1)+2), a[1][2]	1 行 2 列元素 a[1][2]的值	元素值为 13

例7.1 0 输出二维数组有关的值

```
#include <stdio.h>

#define FROMAT "%d %d\n"

int main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    printf(FROMAT,a,*a);
    printf(FROMAT,a[0],*(a+0));
    printf(FROMAT,&a[0],&a[0][0]);
    printf(FROMAT,a[1],a+1);
    printf(FROMAT,&a[1][0],*(a+1)+0);
    printf(FROMAT,a[2],*(a+2));
    printf(FROMAT,&a[2],a+2);
    printf(FROMAT,a[1][0],*(*(a+1)+0));
}
```

某一次运行结果如下：

1 5 8 , 1 5 8	(0行首地址和0行0列元素地址)
1 5 8 , 1 5 8	(0行0列元素地址)
1 5 8 , 1 5 8	(0行0首地址和0行0列元素地址)
1 6 6 , 1 6 6	(1行0列元素地址和1行首地址)
1 6 6 , 1 6 6	(1行0列元素地址)
1 7 4 , 1 7 4	(2行0列元素地址)
1 7 4 , 1 7 4	(2行首地址)
9 , 9	(1行0列元素的值)

2. 指向多维数组元素的指针变量

在了解上面的概念后，可以用指针变量指向多维数组的元素。

(1) 指向数组元素的指针变量

例7.12 用指针变量输出二维数组元素的值

解题思路：二维数组的所有元素都是整型的，它相当于整型变量，可以用int *型的指针变量指向它。

二维数组中的各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。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 main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int *p; //p是int *型的指针变量
    for(p=a[0];p<a[0]+12;p++)
    { if((p-a[0])%4==0)
        printf("\n");
        printf("%4d",*p); }
}
```

运行结果如下：

1	3	5	7
9	11	13	15
19	21	23	

可将程序最后三个语句改为

```
printf("addr=%o, value = %2d\n",p,*p);
```

在devc++环境下某一次运行时输出如下：

addr=30377154, value = 1

addr=30377160, value = 3

addr=30377164, value = 5

addr=30377170, value = 7

addr=30377174, value = 9

addr=30377200, value = 11

addr=30377204, value = 13

addr=30377210, value = 15

addr=30377214, value = 17

addr=30377220, value = 19

addr=30377224, value = 21

addr=30377230, value = 23

(2) 指向由m个元素组成的一维数组的指针变量

- 上例中的指针变量p是用`int *p`，它指向整型数据，`p+1`所指向的元素是p所指向的列元素的下一个元素。
- 新方法：使p不指向整型变量，而是指向一个包含m个元素的一维数组。此时，如果p先指向`a[0]`(即`p=&a[0]`)，则`p+1`不是指向`a[0][1]`,而是指向`a[1]`，p的增值是以一维数组的长度为单位。

例7.13 输出二维数组任一行任一列元素的值

```
#include <stdio.h>
int main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int (*p)[4],i,j; //指针变量p指向包含4个整型元素的一维数组
    p=a;             //p指向二维数组的0行
    printf( "please enter row and colum: ")
    scanf( "%d, %d",&i,&j);
    printf( "a[%d,%d]=%d\n",i,j,*(*(p+i)+j));
}
```

运行结果：
1,2 (本行为键盘输入)
a [1 , 2] = 1 3

释疑

int a[4]: a有4个元素, 每个元素为整型

int (*p) [4]: (*p) 有4个元素, 每个元素为整型。

可以看出p的类型不是int *, 而是**int (*) [4]**型。

a[i][j]的地址是: * (p+i) +j

*** (p+2) +3** 中的**2**是以p的基类型 (一维数组) 长度为单位的, 即p每增加1, 地址就增加16个字节 (4个元素, 每个元素4个字节) 。 **3**是以元素的长度为单位的, 加3就是加 (3×4) 个字节。

3. 用指向数组的指针作函数参数

例10.13 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。

该题目只是为了说明用指向数组的指针作函数参数而举的例子。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>
```

```
void average(float *p,int n);
```

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

```
int main()
```

```
{
```

```
float score[3][4]={{65,67,70,60},{80,87,90,81},{90,99,100,98}};
```

```
average(*score,12);
```

```
/*求12个分数的平均分*/
```

```
search(score,2);
```

```
/*求序号为 2 的学生的成绩*/
```

```
}
```

```
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);
```

```
}
```

`/*p是指向具有4个元素的一维数组的指针*/`

`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));`

`}`

程序运行结果如下：

average=82.25

the score of No. 2 are:

90.00 99.00 100.00 98.00

例7.14 在上题基础上，查找有一门以上课程不及格的学生，打印出他们的全部课程的成绩。

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

    search (score, 3) ;
}
```

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

```
{
```

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

//j 表示学生号, i表示课程号

```
    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,his scores are:\n",j+1);
```

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

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

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

```
        }
```

```
    }
```

```
}
```

程序运行结果如下：

No.1 fails,his scores are:

65.0 57.0 70.0 60.0

No.2 fails,his scores are:

58.0 87.0 90.0 81.0

Questions & Answers