

第 8 章

善于利用指针

指 针

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。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。内存区的每一个字节有一个编号，这就是“地址”。

由于通过地址能找到所需的变量单元，可以说，地址指向该变量单元，将地址形象化地称为“指针”。

C语言中的地址包括位置信息(内存编号，或称纯地址)和它所指向的数据的类型信息，或者说它是“带类型的地址”。

存储单元的地址和存储单元的内容是两个不同的概念。

```
int i=1,j=2,k=3;  
//设int变量占2字节
```

在程序中一般是通过变量名来引用变量的值。

直接按变量名进行的访问，称为“直接访问”方式。还可以采用另一种称为“间接访问”的方式，即将变量的地址存放在另一变量（指针变量）中，然后通过该指针变量来找到对应变量的地址，从而访问变量。

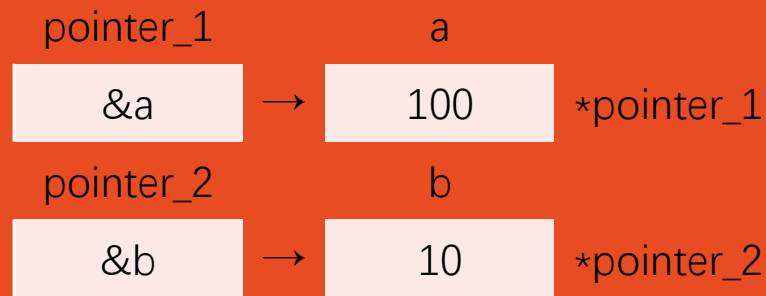
变量名	地址	内容
i	2000	1
	2001	
j	2002	2
	2003	
k	2004	3
	2005	

指针变量

使用指针变量的例子

【例8.1】通过指针变量访问整型变量。

```
#include <stdio.h>
int main()
{
    int a=100,b=10;
    //定义整型变量a,b, 并初始化
    int *pointer_1,*pointer_2;
    //定义指向整型数据的指针变量pointer_1, pointer_2
    pointer_1=&a;    //把变量a的地址赋给指针变量pointer_1
    pointer_2=&b;    //把变量b的地址赋给指针变量pointer_2
    printf("a=%d,b=%d\n",a,b);    //输出变量a和b的值
    printf("*pointer_1=%d,*pointer_2=%d\n",*pointer_1,*pointer_2);
    //输出变量a和b的值
    return 0;
}
```



注意

- 定义指针变量时，左侧应有类型名，否则就不是定义指针变量。

```
C:\WINDOWS\system32\cmd.exe
a=100, b=10
*pointer_1=100, *pointer_2=10
请按任意键继续. . .
```



*pointer_1; //企图定义pointer_1为指针变量。出错



int *pointer_1; //正确，必须指定指针变量的基类型

怎样定义指针变量

类型名 *指针变量名;

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

左端的int是在定义指针变量时必须指定的“**基类型**”。指针变量的基类型用来指定此指针变量可以指向的变量的类型。

前面介绍过基本的数据类型(如int,char, float等), 既然有这些类型的变量, 就可以有指向这些类型变量的指针, 因此, 指针变量是基本数据类型派生出来的类型, 它不能离开基本类型而独立存在。

在定义指针变量时要注意:

- (1) 指针变量前面的“*”表示该变量为指针型变量。指针变量名则不包含“*”。
- (2) 在定义指针变量时必须指定基类型。一个变量的指针的含义包括两个方面, 一是以存储单元编号表示的纯地址 (如编号为2000的字节), 一是它指向的存储单元的数据类型 (如int,char,float等)。
- (3) 如何表示指针类型。指向整型数据的指针类型表示为“int *”, 读作“指向int的指针”或简称“int指针”。
- (4) 指针变量中只能存放地址 (指针), 不要将一个整数赋给一个指针变量。

怎样引用指针变量

- ① 给指针变量赋值。
- ② 引用指针变量指向的变量。
- ③ 引用指针变量的值。

```
int a, *p;  
p=&a;           //把a的地址赋给指针变量p           ①  
printf("%d",*p); //以整数形式输出指针变量p所指向的变量的值，即a的值 ②  
*p=1;          //将整数1赋给p当前所指向的变量，由于p指向变量a，相当于把1赋给a，即a=1 ②  
printf("%o",p); //以八进制形式输出指针变量p的值，由于p指向a，相当于输出a的地址，即&a ③
```

注意

- 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。

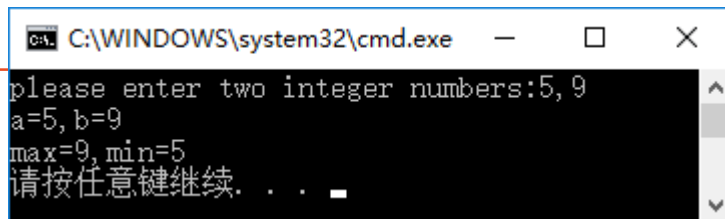
(2) ***** 指针运算符（或称“间接访问”运算符），*p代表指针变量p指向的对象。

怎样引用指针变量

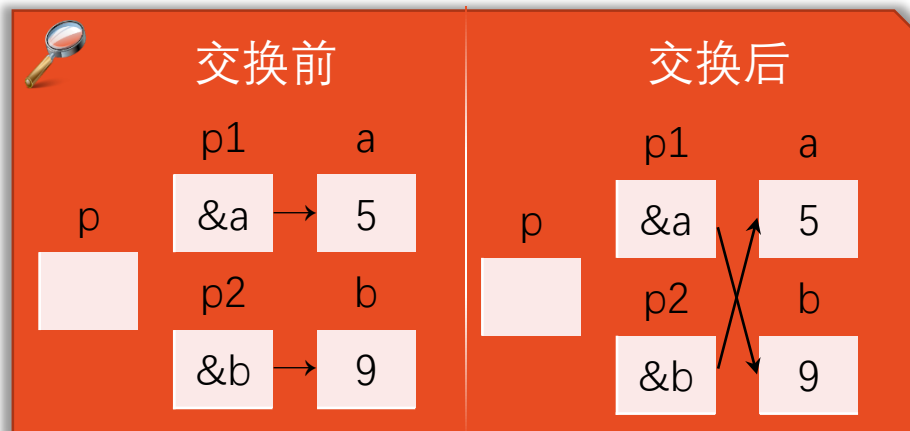
【例8.2】输入a和b两个整数，按先大后小的顺序输出a和b。

解题思路:不交换整型变量的值，而是交换两个指针变量的值（即a和b的地址）。

```
#include <stdio.h>
int main()
{
    int *p1,*p2,*p,a,b;           //p1,p2的类型是int *类型
    printf("please enter two integer numbers:");
    scanf("%d,%d",&a,&b);         //输入两个整数
    p1=&a;                         //使p1指向变量a
    p2=&b;                         //使p2指向变量b
    if(a<b)                       //如果a<b
    {
        p=p1;p1=p2;p2=p;         //使p1与p2的值互换
        printf("a=%d,b=%d\n",a,b); //输出a,b
        printf("max=%d,min=%d\n",*p1,*p2); //输出p1和p2所指向的变量的值
    }
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe - □ ×
please enter two integer numbers:5,9
a=5, b=9
max=9, min=5
请按任意键继续. . .
```



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。
- 实际上，第9行可以改为{p1=&b; p2=&a;}即直接对p1和p2赋以新值，这样可以不必定义中间变量p，使程序更加简练。

指针变量作为函数参数

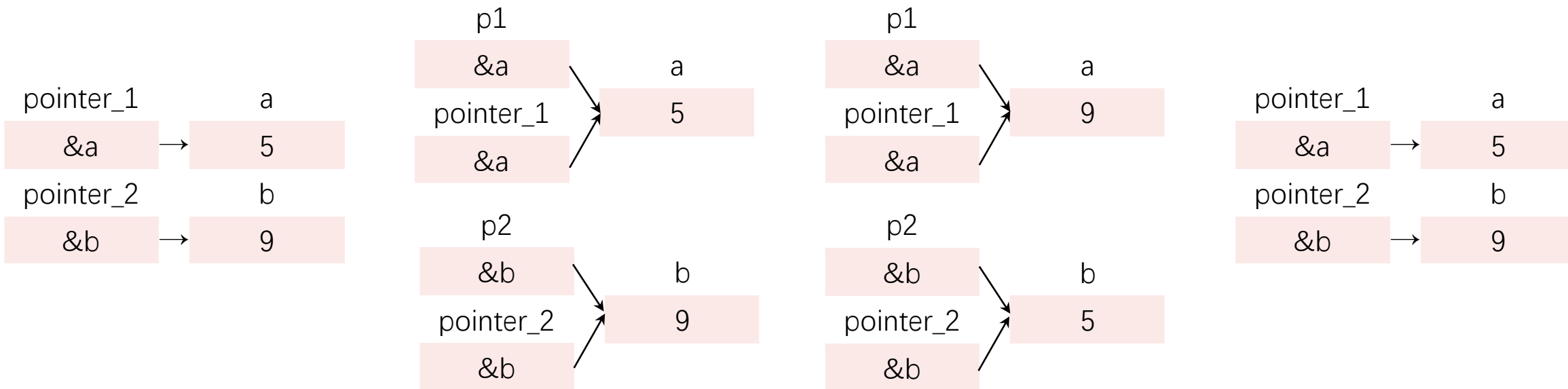
【例8.3】题目要求同例8.2，即对输入的两个整数按大小顺序输出。
现用函数处理，而且用指针类型的数据作函数参数。

```
C:\WINDOWS\system32\cmd.exe
please enter a and b:5,9
max=9,min=5
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{
    void swap(int *p1,int *p2); //对swap函数的声明
    int a,b;
    int *pointer_1,*pointer_2; //定义两个int *型的指针变量
    printf("please enter a and b:");
    scanf("%d,%d",&a,&b); //输入两个整数
    pointer_1=&a; //使pointer_1指向a
    pointer_2=&b; //使pointer_2指向b
    if(a<b) swap(pointer_1,pointer_2); //如果a<b, 调用swap函数
```

```
printf("max=%d,min=%d\n",a,b); //输出结果
return 0;
```


```
void swap(int *p1,int *p2) //定义swap函数
{
    int temp;
    temp=*p1; //使*p1和*p2互换
    *p1=*p2;
    *p2=temp;
} //本例交换a和b的值，而p1和p2的值不变。这恰和例8.2相反
```




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。

```
void swap(int *p1,int *p2)  //定义swap函数
{
    int temp;
    temp=*p1;               //使*p1和*p2互换
    *p1=*p2;
    *p2=temp;
}
```




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



*p1就是a，是整型变量。而*temp是指针变量temp所指向的变量。但由于未给temp赋值，因此temp中并无确定的值(它的值是不可预见的)，所以temp所指向的单元也是不可预见的。所以，对*temp赋值就是向一个未知的存储单元赋值，而这个未知的存储单元中可能存储着一个有用的数据，这样就有可能破坏系统的正常工作状况。

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



在函数调用时，a的值传送给x，b的值传送给。执行完swap函数后，x和y的值是互换了，但并未影响到a和b的值。在函数结束时，变量x和y释放了，main函数中的a和b并未互换。

a	b	a	b
5	9	5	9
↓	↓		
5	9	9	5
x	y	x	y

指针变量作为函数参数

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**。

如果想通过函数调用得到n个要改变的变量，可以这样做：

- 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，将它们的值，也就是相关变量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。
-

指针变量作为函数参数

【例8.4】对输入的两个整数按大小顺序输出。

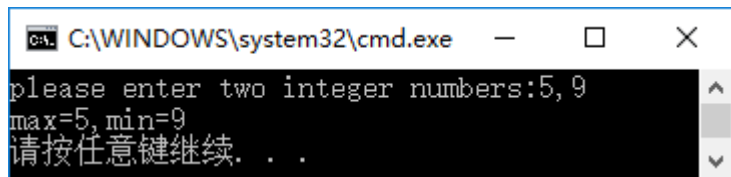
不能企图通过改变指针形参的值而使指针实参的值改变

```
#include <stdio.h>
int main()
{
    void swap(int *p1,int *p2);
    int a,b;
    int *pointer_1,*pointer_2; //pointer_1,pointer_2是int *型变量
    printf("please enter two integer numbers:");
    scanf("%d,%d",&a,&b);
    pointer_1=&a;
    pointer_2=&b;
    if(a<b) swap(pointer_1,pointer_2);
    //调用swap函数，用指针变量作实参
}
```

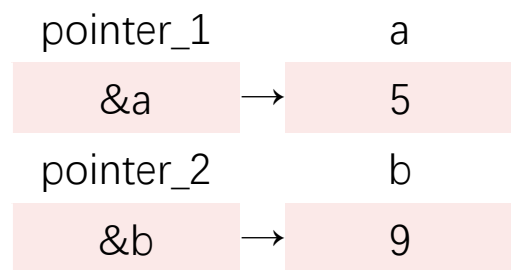
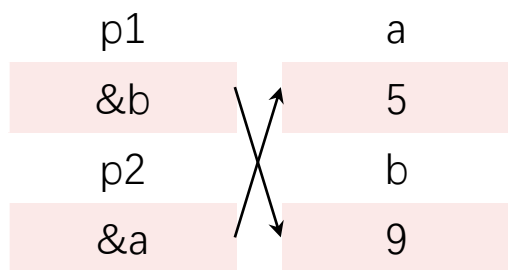
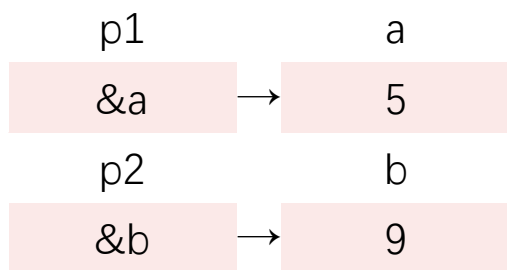
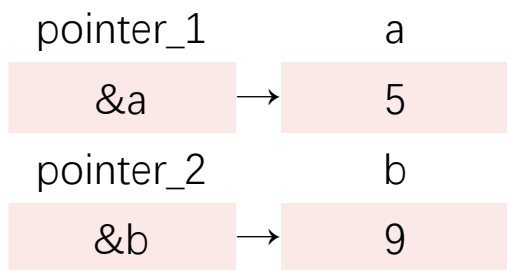
```
printf("max=%d,min=%d\n",*pointer_1,*pointer_2);
return 0;
```

```
void swap(int *p1,int *p2) //形参是指针变量
{
    int *p;
    p=p1;
    p1=p2;
    p2=p;
}
```

//下面3行交换p1和p2的指向



```
C:\WINDOWS\system32\cmd.exe
please enter two integer numbers:5,9
max=5,min=9
请按任意键继续. . .
```



指针变量作为函数参数

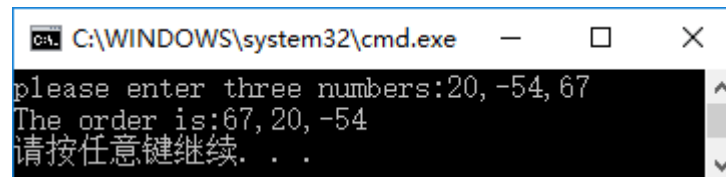
【例8.5】输入3个整数a,b,c，要求按由大到小的顺序将它们输出。用函数实现。

```
#include <stdio.h>
int main()
{
    void exchange(int *q1, int *q2, int *q3); //函数声明
    int a,b,c,*p1,*p2,*p3;
    printf("please enter three numbers:");
    scanf("%d,%d,%d",&a,&b,&c);
    p1=&a;p2=&b;p3=&c;
    exchange(p1,p2,p3);
    printf("The order is:%d,%d,%d\n",a,b,c);
    return 0;
}

void exchange(int *q1, int *q2, int *q3) //将3个变量的值交换的函数
```

```
{
    void swap(int *pt1, int *pt2); //函数声明
    if(*q1<*q2) swap(q1,q2); //如果a<b, 交换a和b的值
    if(*q1<*q3) swap(q1,q3); //如果a<c, 交换a和c的值
    if(*q2<*q3) swap(q2,q3); //如果b<c, 交换b和c的值
}

void swap(int *pt1, int *pt2) //交换2个变量的值的函数
{
    int temp;
    temp=*pt1; //交换*pt1和*pt2变量的值
    *pt1=*pt2;
    *pt2=temp;
}
```



```
C:\WINDOWS\system32\cmd.exe
please enter three numbers:20, -54, 67
The order is:67, 20, -54
请按任意键继续. . .
```

通过指针引用数组

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）。

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};    //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
p=&a[0];                                  //把a[0]元素的地址赋给指针变量p
```

p	&a[0]	→	1	a[0]
			3	
			5	
			7	
			9	
			11	
			13	
			15	
			17	
			19	

引用数组元素可以用下标法，也可以用指针法，即通过指向数组元素的指针找到所需的元素。

```
p=&a[0];    //p的值是a[0]的地址  ≡  p=a;    //p的值是数组a首元素(即a[0])的地址
```

注意

- 程序中的数组名不代表整个数组，只代表数组首元素的地址。

在定义指针变量时可以对它初始化：

```
int *p;
p=&a[0];    //不应写成*p=&a[0];  ≡  int *p=&a[0];  ≡  int *p=a;
```

在引用数组元素时指针的运算

在指针已指向一个数组元素时，可以对指针进行以下运算：

- 加一个整数(用+或+=)，如 $p+1$ ，表示指向同一数组中的下一个元素；
- 减一个整数(用-或-=)，如 $p-1$ ，表示指向同一数组中的上一个元素；
- 自加运算，如 $p++$ ， $++p$ ；
- 自减运算，如 $p--$ ， $--p$ 。

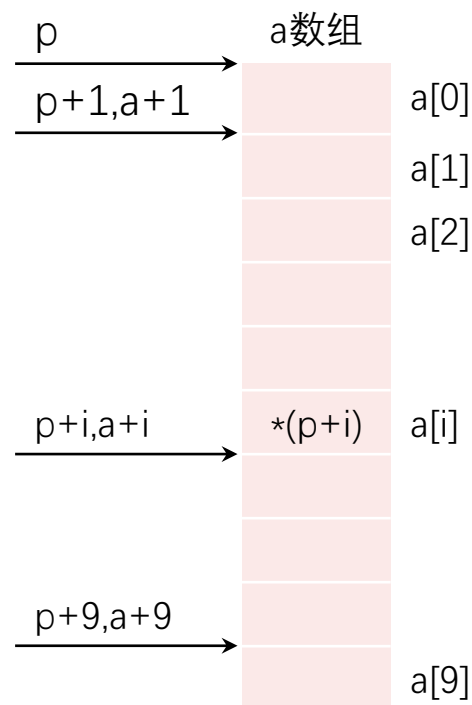
两个指针相减，如 p_1-p_2 (只有 p_1 和 p_2 都指向同一数组中的元素时才有意义)，结果是两个地址之差除以数组元素的长度。注意：两个地址不能相加，如 p_1+p_2 是无实际意义的。

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。

$*(p+i)$ 或 $*(a+i)$ 是 $p+i$ 或 $a+i$ 所指向的数组元素，即 $a[i]$ 。 $[]$ 实际上是变址运算符，即将 $a[i]$ 按 $a+i$ 计算地址，然后找出此地址单元中的值。

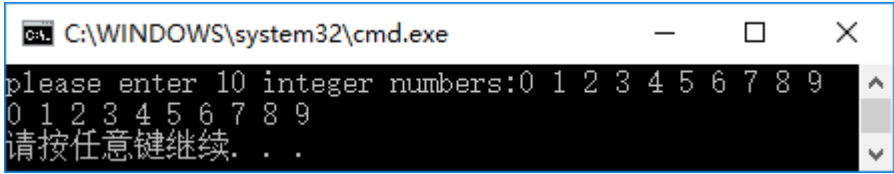
注意

- 执行 $p+1$ 时并不是将 p 的值(地址)简单地加1，而是根据定义的基本类型加上一个数组元素所占用的字节数。



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。



```
C:\WINDOWS\system32\cmd.exe
please enter 10 integer numbers:0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
请按任意键继续...
```

①下标法

```
#include <stdio.h>
int main()
{
    int a[10];
    int i;
    printf("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    //数组元素用数组名和下标表示
    printf("\n");
    return 0;
}
```

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("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    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找到该元素
    printf("\n");
    return 0;
}
```

③用指针变量指向数组元素

```
#include <stdio.h>
int main()
{
    int a[10];
    int *p,i;
    printf("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    for(p=a;p<(a+10);p++)
        printf("%d ",*p);
    //用指针指向当前的数组元素
    printf("\n");
    return 0;
}
```

第(1)和第(2)种方法执行效率是相同的。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。

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。

通过指针引用数组元素

用下标法比较直观，能直接知道是第几个元素。适合初学者使用。

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。单用指针变量的方法进行控制，可使程序简洁、高效。

注意

- 在使用指针变量指向数组元素时，有以下几个问题要注意：

(1) 可以通过改变指针变量的值指向不同的元素。

如果不用p变化的方法而用数组名a变化的方法（例如，用a++）行不行呢？

```
for(p=a;a<(p+10);a++)  
printf("%d",*a);
```



因为数组名a代表数组首元素的地址，它是一个指针型常量，它的值在程序运行期间是固定不变的。既然a是常量，所以a++是无法实现的。

(2) 要注意指针变量的当前值。

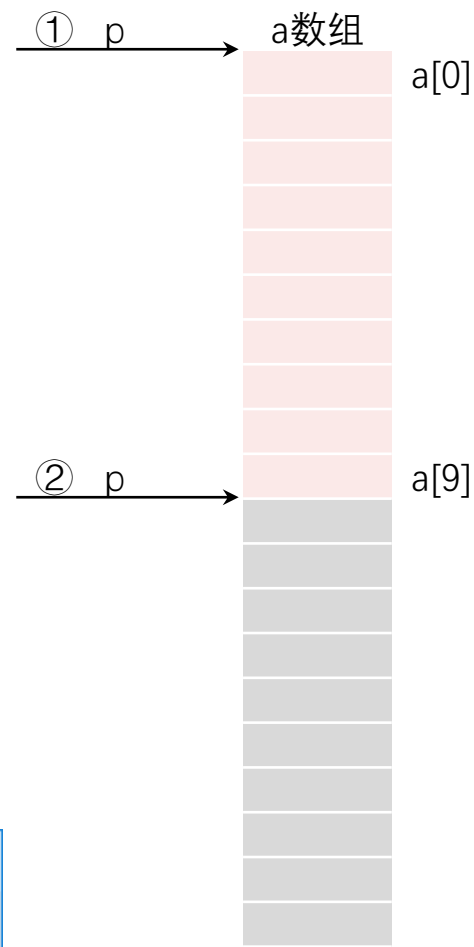
通过指针引用数组元素

【例8.7】通过指针变量输出整型数组a的10个元素。

```
#include <stdio.h>
int main()
{   int *p,i,a[10];
    p=a;           //p指向a[0]      ①
    printf("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",p++); //输入10个整数给a[0]~a[9]
    for(i=0;i<10;i++,p++)
        printf("%d ",*p); //想输出a[0]~a[9] ②
    printf("\n");
    return 0;
}
```



```
#include <stdio.h>
int main()
{   int i,a[10],*p=a; //p的初值是a, p指向a[0]
    printf("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",p++);
    p=a;           //重新使p指向a[0]
    for(i=0;i<10;i++,p++)
        printf("%d ",*p);
    printf("\n");
    return 0;
}
```



C:\WINDOWS\system32\cmd.exe

```
please enter 10 integer numbers:0 1 2 3 4 5 6 7 8 9
-858993460 -858993460 2 -858993460 -858993460 5240960 -858993460
请按任意键继续. . .
```

C:\WINDOWS\system32\cmd.exe

```
please enter 10 integer numbers:0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
请按任意键继续. . .
```

通过指针引用数组元素

- (1) 从例8.7可以看到，虽然定义数组时指定它包含10个元素，并用指针变量p指向某一数组元素，但是实际上指针变量p可以指向数组以后的存储单元，结果不可预期，应避免出现这样的情况。
- (2) 指向数组元素的指针变量也可以带下标，如p[i]。p[i]被处理成*(p+i)，如果p是指向一个整型数组元素a[0]，则p[i]代表a[i]。但是必须弄清楚p的当前值是什么？如果当前p指向a[3]，则p[2]并不代表a[2]，而是a[3+2]，即a[5]。
- (3) 利用指针引用数组元素，比较方便灵活，有不少技巧。请分析下面几种情况：

设p开始时指向数组a的首元素（即p=a）：

① p++; //使p指向下一元素a[1]
*p; //得到下一个元素a[1]的值

② *p++; /*由于++和*同优先级，结合方向自右而左，因此它等价于*(p++)。先引用p的值，实现*p的运算，然后再使p自增1*/

③ *(p++); //先取*p值，然后使p加1
*(++p); //先使p加1，再取*p

④ ++(*p); /*表示p所指向的元素值加1，如果p=a，则相当于++a[0]，若a[0]的值为3，则a[0]的值为4。注意：是元素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进行“”运算，再使p自减

(++p) //相当于a[++i]，先使p自加，再进行“”运算

(--p) //相当于a[--i]，先使p自减，再进行“”运算

用数组名作函数参数

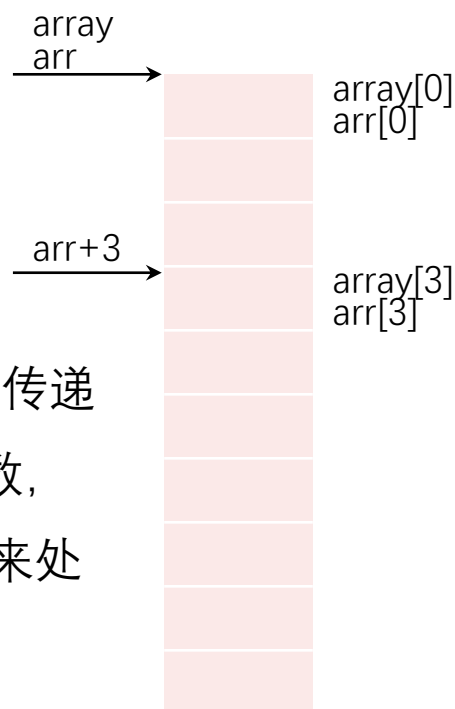
```
int main()
{
    void fun(int arr[], int n); //对fun函数的声明
    int array[10];             //定义array数组
    :
    fun(array,10);             //用数组名作函数的参数
    return 0;
}

void fun(int arr[], int n)     //定义fun函数
{
    :
}
```

array是实参数组名，arr为形参数组名。当用数组名作参数时，如果形参数组中各元素的值发生变化，实参数组元素的值随之变化。

≡

```
void fun(int *arr, int n)     //定义fun函数
{
    :
}
```



在该函数被调用时，系统会在fun函数中建立一个指针变量arr，用来存放从主调函数传递过来的实参数组首元素的地址。如果在fun函数中用运算符sizeof测定arr所占的字节数，可以发现sizeof(arr)的值为4(用Visual C++时)。这就证明了系统是把arr作为指针变量来处理的(指针变量在Visual C++中占4个字节)。

当arr接收了实参数组的首元素地址后，arr就指向实参数组首元素，也就是指向array[0]。

用数组名作函数参数

以变量名和数组名作为函数参数的比较

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能改变实参变量的值	能改变实参数组的值

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。

注意

- 实参数组名代表一个固定的地址，或者说是指针常量，但形参数组名并不是一个固定的地址，而是按指针变量处理。

在函数调用进行虚实结合后，形参的值就是实参数组首元素的地址。

在函数执行期间，它可以再被赋值。

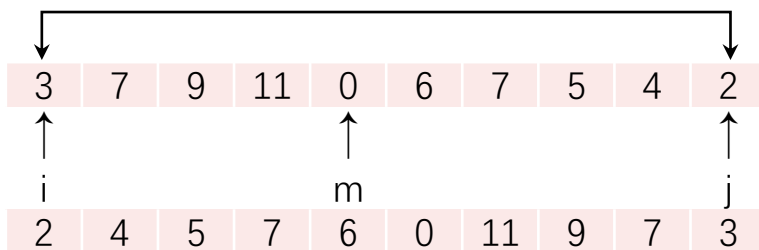
```
void fun (arr[ ],int n)
{   printf("%d\n", *arr);    //输出array[0]的值
    arr=arr+3;               //形参数组名可以被赋值
    printf("%d\n", *arr);    //输出array[3]的值
}
```

用数组名作函数参数

【例8.8】将数组a中n个整数按相反顺序存放。

```
#include <stdio.h>
int main()
{
    void inv(int x[],int n);    //inv函数声明
    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]);    //输出未交换时数组各元素的值
    printf("\n");
    inv(a,10);                  //调用inv函数, 进行交换
    printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);    //输出交换后数组各元素的值
    printf("\n");
    return 0;
}

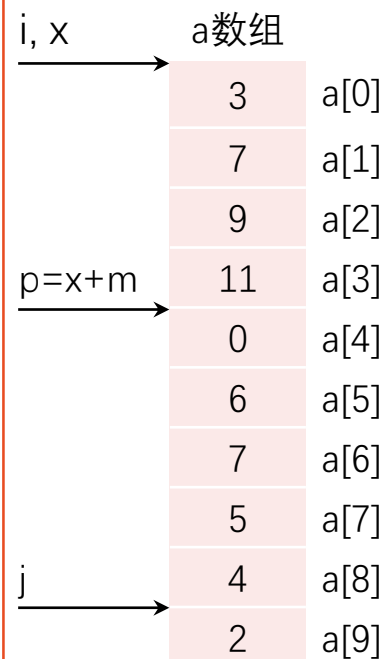
void inv(int x[],int n)        //形参x是数组名
{
    int temp,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {
        j=n-1-i;
        temp=x[i]; x[i]=x[j]; x[j]=temp;    //把x[i]和x[j]交换
    }
    return;
}
```



```
C:\WINDOWS\system32\cmd.exe
The original array:
3 7 9 11 0 6 7 5 4 2
The array has been inverted:
2 4 5 7 6 0 11 9 7 3
请按任意键继续. . .
```

```
#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};
    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");
    return 0;
}

void inv(int *x,int n)        //形参x是指针变量
{
    int *p,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;    //i与*j交换
    }
    return;
}
```



用数组名作函数参数

如果有一个实参数组，要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

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

int f(int x[], int n)
{
    ⋮
}
```

②

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

int f(int *x, int n)
{
    ⋮
}
```

③

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

int f(int *x, int n)
{
    ⋮
}
```

④

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

int f(int x[], int n)
{
    ⋮
}
```

用数组名作函数参数

【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);    //inv函数声明
    int i,arr[10],*p=arr;    //指针变量p指向arr[0]
    printf("The original array:\n");
    for(i=0;i<10;i++,p++)
        scanf("%d",p);    //输入arr数组的元素
    printf("\n");
    p=arr;    //指针变量p重新指向arr[0]
    inv(p,10);    //调用inv函数，实参p是指针变量
    printf("The array has been inverted:\n");
    for(p=arr;p<arr+10;p++)
        printf("%d ",*p);
    printf("\n");
    return 0;
}

void inv(int *x,int n)    //定义inv函数，形参x是指针变量
{
    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;
}
```

```
#include <stdio.h>
int main()
{
    void inv(int *x,int n);    //inv函数声明
    int i,*arr;    //指针变量arr未指向数组元素
    printf("The original array:\n");
    for(i=0;i<10;i++)
        scanf("%d",arr+i);
    printf("\n");
    inv(arr,10);    //调用inv函数，实参arr是指针变量，但无指向
    printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
        printf("%d ",*(arr+i));
    printf("\n");
    return 0;
}
```

注意

- 如果用指针变量作实参，必须先使指针变量有确定值，指向一个已定义的对象。

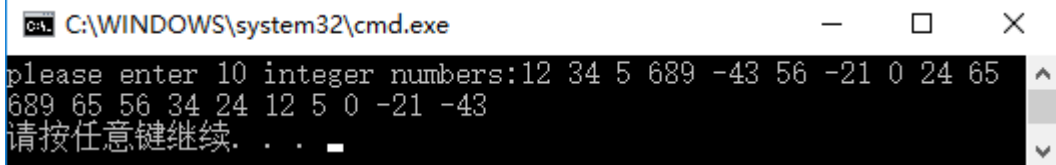
用数组名作函数参数

【例8.10】用指针方法对10个整数按由大到小顺序排序。（选择排序法）

```
#include <stdio.h>
int main()
{
    void sort(int x[],int n); //sort函数声明
    int i,*p,a[10];
    p=a; //指针变量p指向a[0]
    printf("please enter 10 integer numbers:");
    for(i=0;i<10;i++)
        scanf("%d",p++); //输入10个整数
    p=a; //指针变量p重新指向a[0]
    sort(p,10); //调用sort函数
    for(p=a,i=0;i<10;i++)
    {
        printf("%d ",*p); //输出排序后的10个数组元素
        p++;
    }
    printf("\n");
    return 0;
}
```

```
void sort(int x[],int n) //定义sort函数，x是形参数组名
{
    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;
        }
    }
}
```

```
void sort(int *x,int n) //形参x是指针变量
{
    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; //*(x+j)就是x[j]，其他亦然
        if(k!=i)
        {
            t=*(x+i); *(x+i)=*(x+k); *(x+k)=t;
        }
    }
}
```

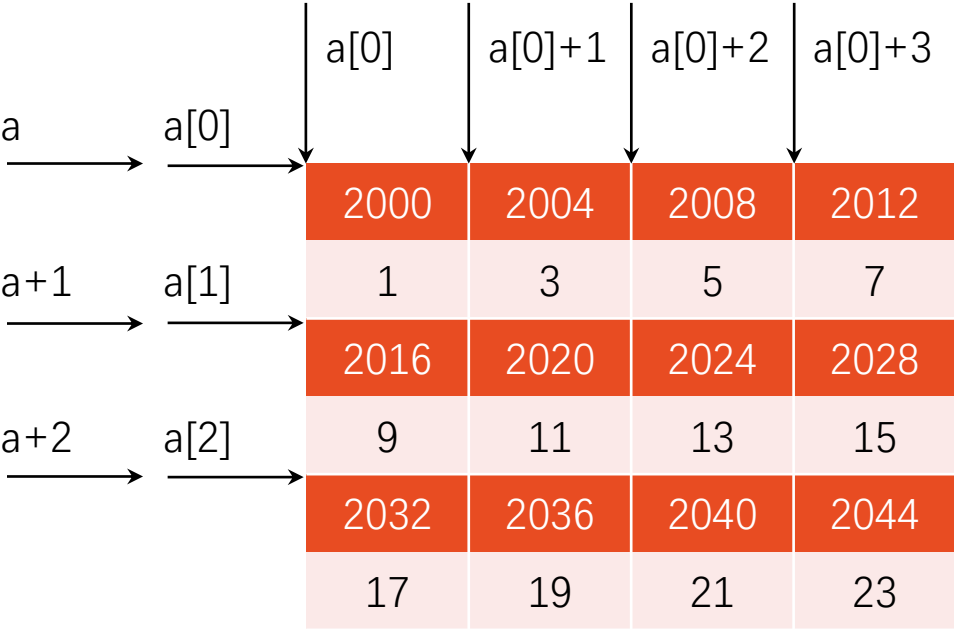


```
C:\WINDOWS\system32\cmd.exe
please enter 10 integer numbers:12 34 5 689 -43 56 -21 0 24 65
689 65 56 34 24 12 5 0 -21 -43
请按任意键继续. . .
```

***通过指针引用多维数组**

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}};
```



表示形式	含义	地址
a	二维数组名，指向一维数组a[0]，即0行起始地址	2000
a[0], *(a+0), *a	0行0列元素地址	2000
a+1, &a[1]	指向第1行起始地址	2016
a[1], *(a+1)	1行0列元素a[1][0]的地址	2016
a[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2]的地址	2024
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a[1][2]的值	是元素值，为13

多维数组元素的地址

C语言的地址信息中既包含位置信息(如内存编号2000), 还包含它所指向的数据的类型信息。

a[0]是一维数组名, 它是一维数组中起始元素的地址, a是二维数组名, 它是二维数组的首行起始地址, 二者的纯地址是相同的, 即2000, 但它们的基类型不同, 即它们指向的数据的类型不同, 前者是整型数据, 后者是一维数组。

如果用一个指针变量pt来指向此一维数组, 应当这样定义:

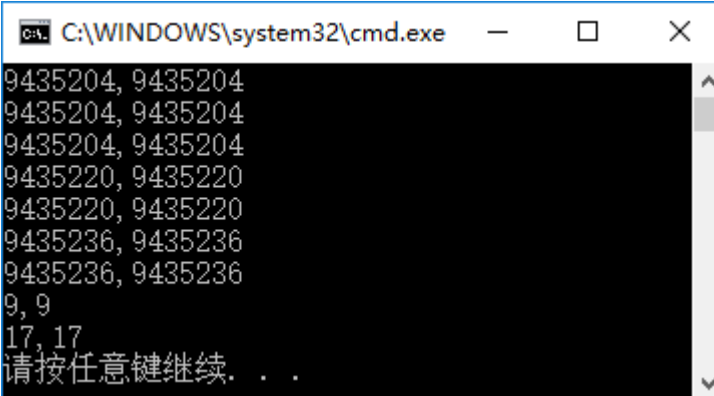
```
int (*pt)[4];
```

```
//表示pt指向由4个整型元素组成的一维数组, 此时指针变量pt的基类型是由4个整型元素组成的一维数组
```

多维数组元素的地址

【例8.11】输出二维数组的有关数据(地址和元素的值)。

```
#include <stdio.h>
int main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    printf("%d,%d\n",a,*a);           //0行起始地址和0行0列元素地址
    printf("%d,%d\n",a[0],*(a+0));    //0行0列元素地址
    printf("%d,%d\n",&a[0],&a[0][0]); //0行起始地址和0行0列元素地址
    printf("%d,%d\n",a[1],a+1);       //1行0列元素地址和1行起始地址
    printf("%d,%d\n",&a[1][0],*(a+1)+0); //1行0列元素地址
    printf("%d,%d\n",a[2],*(a+2));    //2行0列元素地址
    printf("%d,%d\n",&a[2],a+2);      //2行起始地址
    printf("%d,%d\n",a[1][0],*(*(a+1)+0)); //1行0列元素的值
    printf("%d,%d\n",*a[2],*(*(a+2)+0)); //2行0列元素的值
    return 0;
}
```

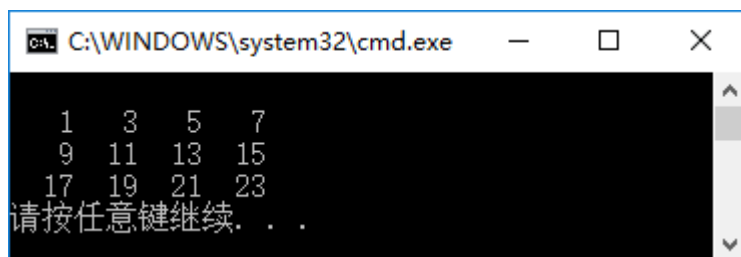


```
C:\WINDOWS\system32\cmd.exe
9435204, 9435204
9435204, 9435204
9435204, 9435204
9435220, 9435220
9435220, 9435220
9435236, 9435236
9435236, 9435236
9, 9
17, 17
请按任意键继续. . .
```

指向数组元素的指针变量

【例8.12】有一个3×4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。

```
#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 *型指针变量
    for(p=a[0];p<a[0]+12;p++)        //使p依次指向下一个元素
    {   if((p-a[0])%4==0) printf("\n"); //p移动4次后换行
        printf("%4d",*p);            //输出p指向的元素的值
    }
    printf("\n");
    return 0;
}
```

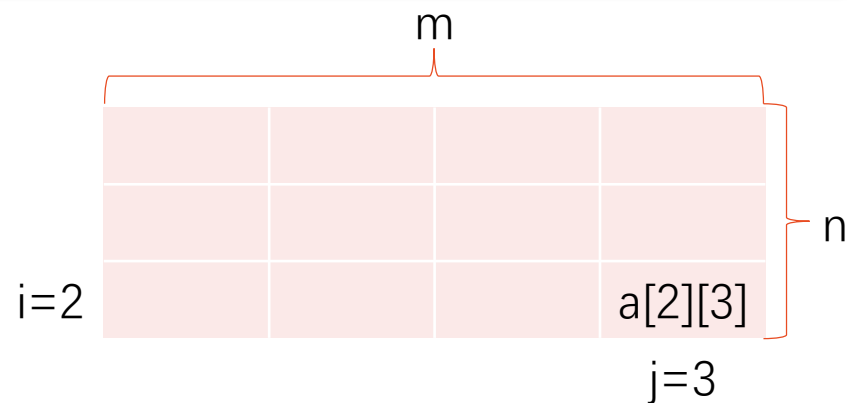


```
C:\WINDOWS\system32\cmd.exe
1   3   5   7
9  11  13  15
17 19 21 23
请按任意键继续...
```



p是一个int *型(指向整型数据)的指针变量，它可以指向一般的整型变量，也可以指向整型的数组元素。每次使p值加1，使p指向下一元素。

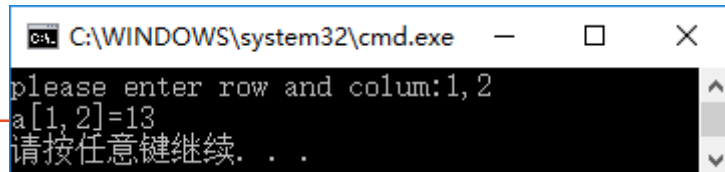
如果要输出某个指定的数值元素（例如a[2][3]），则应事先计算该元素在数组中的相对位置（即相对于数组起始位置的相对位移量）。计算a[i][j]在数组中的相对位置的计算公式为： $i*m + j$ ，其中，m为二维数组的列数（二维数组大小为n×m）。



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。

```
#include <stdio.h>
int main()
{
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};    //定义二维数组a并初始化
    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);    //输入要求输出的元素的行列号
    printf("a[%d,%d]=%d\n",i,j,*(p+i)+j);    //输出a[i][j]的值
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
please enter row and colum:1,2
a[1,2]=13
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{
    int a[4]={1,3,5,7};    //定义一维数组a, 包含4个元素
    int (*p)[4];    //定义指向包含4个元素的一维数组的指针变量中
    p=&a;    //使p指向一维数组
    printf("%d\n",(*p)[3]);    //输出a[3], 输出整数7
    return 0;
}
```



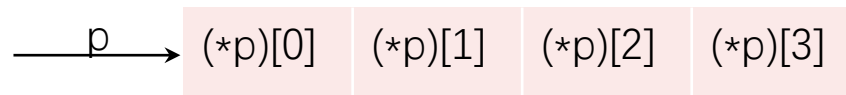
比较:

① int a[4]; (a有4个元素, 每个元素为整型)

② int (*p)[4];

第②种形式表示(*p)有4个元素, 每个元素为整型。也就是p所指的对象是有4个整型元素的数组, 即p是指向一维数组的指针, 见图8.24。应该记住, 此时p只能指向一个包含4个元素的一维数组, 不能指向一维数组中的某一元素。p的值是该一维数组的起始地址。虽然这个地址(指纯地址)与该一维数组首元素的地址相同, 但它们的基类型是不同的。

*p (数组)



指向由m个元素组成的一维数组的指针变量

要注意指针变量的类型，从“`int (*p)[4];`”可以看到，`p`的类型不是`int *`型，而是`int (*)[4]`型，`p`被定义为指向一维整型数组的指针变量，一维数组有4个元素，因此`p`的基类型是一维数组，其长度是16字节。“`*(p+2)+3`”括号中的2是以`p`的基类型(一维整型数组)的长度为单位的，即`p`每加1，地址就增加16个字节（4个元素，每个元素4个字节），而“`*(p+2)+3`”括号外的数字3，不是以`p`的基类型的长度为单位的。由于经过`*(p+2)`的运算，得到`a[2]`，即`&a[2][0]`，它已经转化为指向列元素的指针了，因此加3是以元素的长度为单位的，加3就是加 (3×4) 个字节。虽然`p+2`和`*(p+2)`具有相同的值，但由于它们所指向的对象的长度不同，因此`(p+2)+3`和`*(p+2)+3`的值就不相同了。

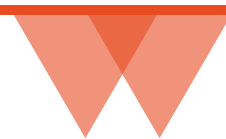


用指向数组的指针作函数参数

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。



用指向数组的指针作函数参数

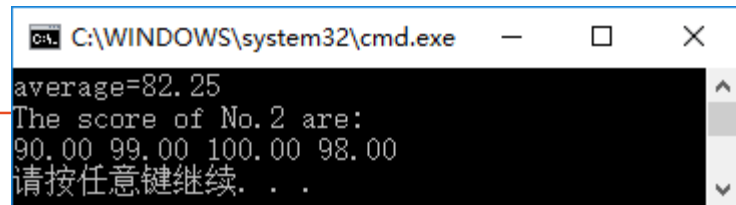
【例8.14】有一个班，3个学生，各学4门课，计算总平均分数以及第n个学生的成绩。

```
#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);           //求12个分数的平均分
    search(score,2);              //求序号为2的学生的成绩
    return 0;
}

void average(float *p,int n)      //定义求平均成绩的函数
{
    float *p_end;
    float sum=0,aver;
    p_end=p+n-1;
    //n的值为12时，p_end的值是p+11，指向最后一个元素
```

```
    for(;p<=p_end;p++)
        sum=sum+(*p);
    aver=sum/n;
    printf("average=%5.2f\n",aver);
}

void search(float (*p)[4],int n)
//p是指向具有4个元素的一维数组的指针
{
    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");
}
```



```
C:\WINDOWS\system32\cmd.exe
average=82.25
The score of No.2 are:
90.00 99.00 100.00 98.00
请按任意键继续. . .
```

注意

- 实参与形参如果是指针类型，应当注意它们的基类型必须一致。不应把int *型的指针(即数组元素的地址)传给int (*)[4] 型(指向一维数组)的指针变量，反之亦然。

用指向数组的指针作函数参数

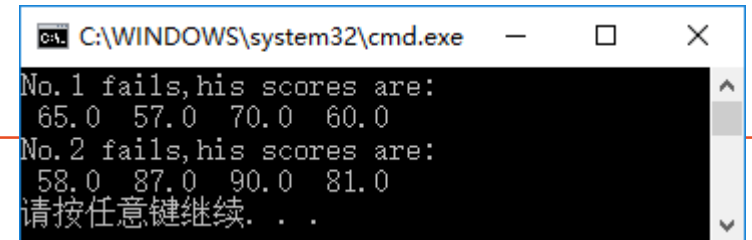
【例8.15】在例8.14的基础上，查找有一门以上课程不及格的学生，输出他们的全部课程的成绩。

```
#include <stdio.h>
int main()
{
    void search(float (*p)[4],int n);    //函数声明
    float score[3][4]={{65,57,70,60},{58,87,90,81},{90,99,100,98}};
    //定义二维数组函数score
    search(score,3);                    //调用search函数
    return 0;
}

void search(float (*p)[4],int n)
//形参p是指向包含4个float型元素的一维数组的指针变量
{
    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;
        //(*(*(p+j)+i)就是score[j][i])
        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));
            //输出(*(*(p+j)+i)就是输出score[j][i]的值
            printf("\n");
        }
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
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
请按任意键继续...
```

通过指针引用字符串

字符串的引用方式

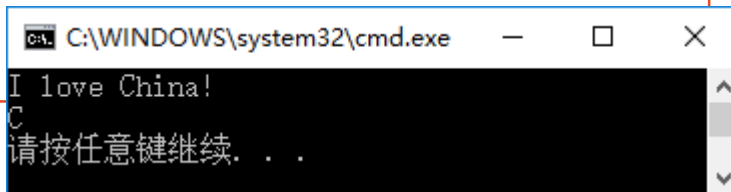
- (1) 用字符数组存放一个字符串，可以通过数组名和下标引用字符串中一个字符，也可以通过数组名和格式声明“%s”输出该字符串。
 - (2) 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。
-

字符串的引用方式

【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。

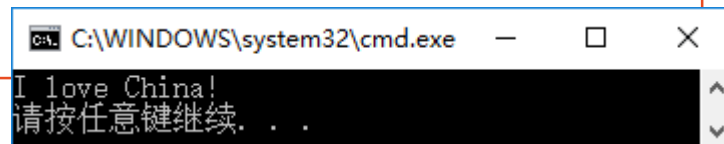
【例8.17】通过字符指针变量输出一个字符串。

```
#include <stdio.h>
int main()
{
    char string[]="I love China!"; //定义字符数组string
    printf("%s\n",string);          //用%s格式声明输出string，可以输出整个字符串
    printf("%c\n",string[7]);       //用%c格式输出一个字符数组元素
    return 0;
}
```

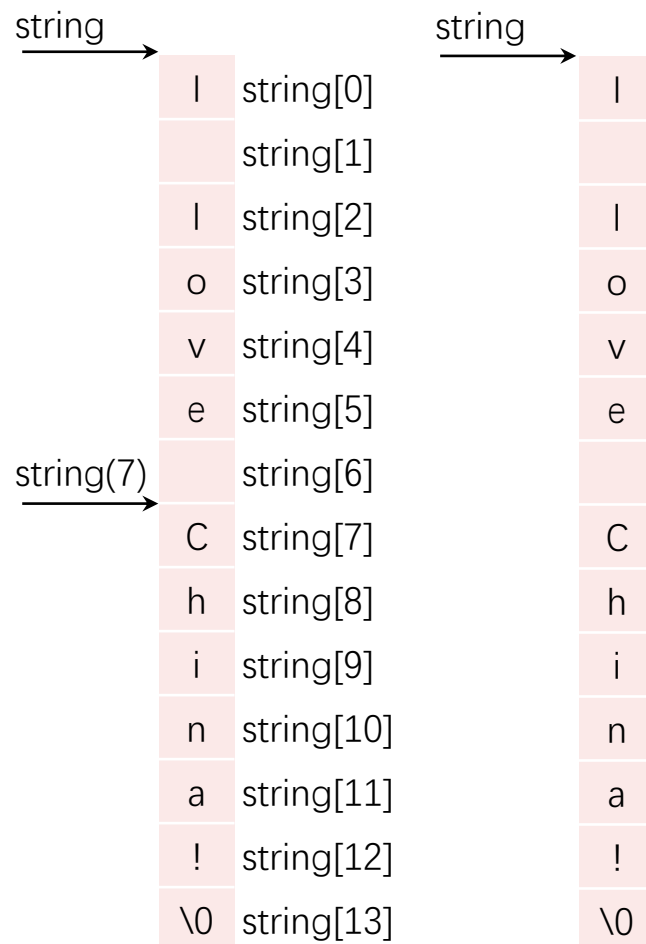


```
C:\WINDOWS\system32\cmd.exe
I love China!
C
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{
    char *string="I love China!"; //定义字符指针变量string并初始化
    printf("%s\n",string);        //输出字符串
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
I love China!
请按任意键继续. . .
```



字符串的引用方式

在C语言中只有字符变量，没有字符串变量。

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

≡

```
char *string;    //定义一个char *型变量  
string="I love China!";  
//把字符串第1个元素的地址赋给字符指针变量string
```

注意

- string被定义为一个指针变量，基类型为字符型。它只能指向一个字符类型数据，而不能同时指向多个字符数据，更不是把"I love China!"这些字符存放到string中（指针变量只能存放地址），也不是把字符串赋给*string。只是把"I love China!"的第1个字符的地址赋给指针变量string。

可以对指针变量进行再赋值，如：

```
string="I am a student.";    //对指针变量string重新赋值
```

可以通过字符指针变量输出它所指向的字符串，如：

```
printf("%s\n",string); //%s可对字符串进行整体的输入输出
```

%s是输出字符串时所用的格式符，在输出项中给出字符指针变量名string，则系统会输出string所指向的字符串第1个字符，然后自动使string加1，使之指向下一个字符，再输出该字符……如此直到遇到字符串结束标志'\0'为止。注意，在内存中，字符串的最后被自动加了一个'\0'。

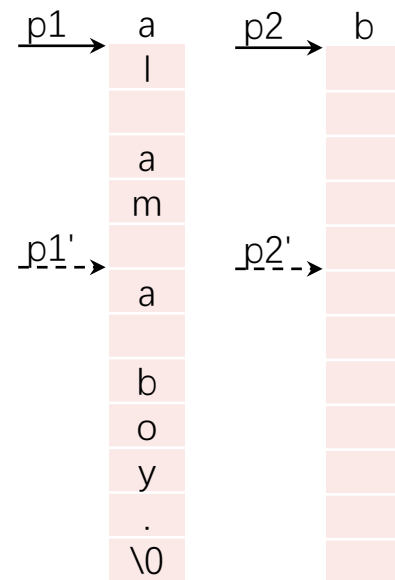
字符串的引用方式

对字符串中字符的存取，可以用下标方法，也可以用指针方法。

【例8.18】将字符串a复制为字符串b，然后输出字符串b。 【例8.19】用指针变量来处理例8.18问题。

```
#include <stdio.h>
int main()
{
    char a[]="I am a student.",b[20]; //定义字符数组
    int i;
    for(i=0;*(a+i)!='\0';i++)
        *(b+i)=*(a+i); //将a[i]的值赋给b[i]
    *(b+i)='\0'; //在b数组的有效字符之后加'\0'
    printf("string a is:%s\n",a); //输出a数组中全部有效字符
    printf("string b is:");
    for(i=0;b[i]!='\0';i++)
        printf("%c",b[i]); //逐个输出b数组中全部有效字符
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char a[]="I am a boy.",b[20],*p1,*p2;
    p1=a;p2=b;
    //p1,p2分别指向a数组和b数组中的第一个元素
    for(;*p1!='\0';p1++,p2++) //p1,p2每次自加1
        *p2=*p1;
    //将p1所指向的元素的值赋给p2所指向的元素
    *p2='\0'; //在复制全部有效字符后加'\0'
    printf("string a is:%s\n",a); //输出a数组中的字符
    printf("string b is:%s\n",b); //输出b数组中的字符
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
string a is:I am a student.
string b is:I am a student.
请按任意键继续. . .
```


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。

(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("string a=%s\nstring b=%s\n",a,b);
    printf("copy string a to string b:\n");
    copy_string(a,b);      //用字符数组名作为函数实参
    printf("\nstring a=%s\nstring b=%s\n",a,b);
    return 0;
}

void copy_string(char from[], char to[])    //形参为字符数组
{
    int i=0;
    while(from[i]!='\0')
    {
        to[i]=from[i]; i++;
    }
    to[i]='\0';
}
```

a,p
from →

a
I
a
m
a
t
e
a
c
h
e
r
.
\0

a
to →

b
Y
o
u
a
r
e
a
s
t
u
d
e
n
t
.
\0

I
a
m
a
t
e
a
c
h
e
r
.
\0
t
.
\0

```
C:\WINDOWS\system32\cmd.exe
string a=I am a teacher.
string b=You are a student.
copy string a to string b:

string a=I am a teacher.
string b=I am a teacher.
请按任意键继续. . .
```

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。

(2) 用字符型指针变量作实参

```
#include <stdio.h>
int main()
{
    void copy_string(char from[], char to[]);    //函数声明
    char a[]="I am a teacher.";                //定义字符数组a并初始化
    char b[]="You are a student.";              //定义字符数组b并初始化
    char *from=a,*to=b;                        //from指向a数组首元素, to指向b数组首元素
    printf("string a=%s\nstring b=%s\n",a,b);
    printf("copy string a to string b:\n");
    copy_string(from,to);                      //实参为字符指针变量
    printf("\nstring a=%s\nstring b=%s\n",a,b);
    return 0;
}
void copy_string(char from[], char to[])        //形参为字符数组
{
    int i=0;
    while(from[i]!='\0')
    {
        to[i]=from[i]; i++;
    }
    to[i]='\0';
}
```



指针变量from的值是a数组首元素的地址，指针变量to的值是b数组首元素的地址。它们作为实参，把a数组首元素的地址和b数组首元素的地址传递给形参数组名from和to(它们实质上也是指针变量)。其他与程序(1)相同。

```
C:\WINDOWS\system32\cmd.exe
string a=I am a teacher.
string b=You are a student.
copy string a to string b:

string a=I am a teacher.
string b=I am a teacher.
请按任意键继续. . .
```

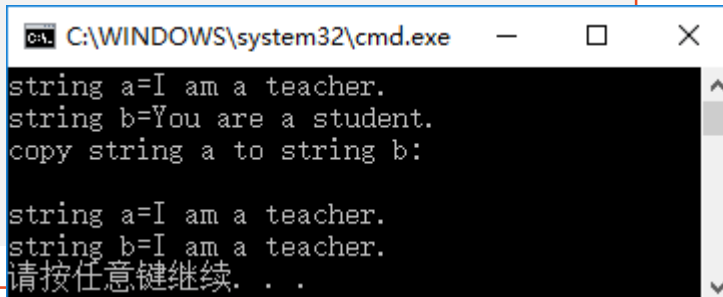
字符指针作函数参数

【例8.20】用函数调用实现字符串的复制。

(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.";      //a是char*型指针变量
    char b[]="You are a student.";   //b是字符数组
    char *p=b;                      //使指针变量p指向b数组首元素
    printf("string a=%s\nstring b=%s\n",a,b);  //输出a串和b串
    printf("copy string a to string b:\n");
    copy_string(a,p); //调用copy_string函数, 实参为指针变量
    printf("\nstring a=%s\nstring b=%s\n",a,b); //输出改变后的a串和b串
    return 0;
}

void copy_string(char *from, char *to) //定义函数, 形参为字符指针变量
{
    for(;*from!='\0';from++,to++)
    {
        *to=*from;
        *to='\0';
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
string a=I am a teacher.
string b=You are a student.
copy string a to string b:

string a=I am a teacher.
string b=I am a teacher.
请按任意键继续. . .
```

改进

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

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

```
void copy_string(char *from, char *to)
{
    while(*from!='\0')
    //或while(*from), 因为'\0'的ASCII码为0
    {
        *to++=*from++;
        *to='\0';
    }
}
```

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

```
void copy_string (char from[], char to[])
{
    char *p1, *p2;
    p1=from;p2=to;
    while((*p2++=*p1++)!='\0');
}
```

字符指针作函数参数

字符指针作为函数参数时，实参与形参的类型有以下几种对应关系：

实参	形参
字符数组名	字符数组名
字符数组名	字符指针变量
字符指针变量	字符指针变量
字符指针变量	字符数组名

使用字符指针变量和字符数组的比较

(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)，绝不是将字符串放到字符指针变量中。

(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。(数组名是常量)

(3) 初始化的含义。

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

≡

```
char *a;
```

```
a="I love China!";
```

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

≠

```
char str[14];
```

```
str[]="I love China!";
```



(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(Visual C++为指针变量分配4个字节)。

```
char *a;  
scanf("%s",a);
```



```
char *a,str[10];  
a=str; scanf("%s",a);
```



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

(6) 字符数组中各元素的值是可以改变的(可以对它们再赋值)，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(不能对它们再赋值)。

```
char a[]="House";  
a[2]='r';
```



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



(7) 引用数组元素。对字符数组可以用下标法(用数组名和下标)引用一个数组元素(如a[5])，也可以用地址法(如*(a+5))引用数组元素a[5]。如果定义了字符指针变量p，并使它指向数组a的首元素，则可以用指针变量带下标的形式引用数组元素(如p[5])，同样，可以用地址法(如*(p+5))引用数组元素a[5]。

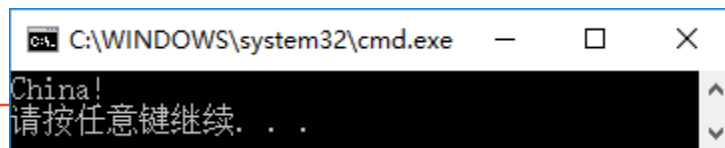
```
char *format="a=%d,b=%f\n";  
printf(format,a,b);
```

(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。

使用字符指针变量和字符数组的比较

【例8.21】改变指针变量的值。

```
#include <stdio.h>
int main()
{
    char *a="I love China!";
    a=a+7;           //改变指针变量的值，即改变指针变量的指向
    printf("%s\n",a); //输出从a指向的字符开始的字符串
    return 0;
}
```



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



指针变量a的值是可以变化的。printf函数输出字符串时，从指针变量a当时所指向的元素开始，逐个输出各个字符，直到遇'\0'为止。而数组名虽然代表地址，但它是常量，它的值是不能改变的。

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