第6章 指针和引用

学习C语言，如果你不能用指针编写有效、正确而灵活的程序，可以认为你没有学好C语言。地址、指针、数组及其相互关系是C语言中最具特色的部分。规范地使用指针，可以使程序更加简洁明了，因此，我们要学会在各种情况下正确地使用指针。

●要求：

1）掌握指针的基本概念、指针变量的定义、引用与运算；

2）熟悉"指向指针的指针"；

3）掌握"指针与数组"、"指针与函数"的相关操作。

指针在C程序中的作用

(1)利用指针能间接引用它所指的对象。

(2)指针能用来描述数据和数据之间的关系，以便构造复杂的数据结构。

(3)利用各种类型指针形参，能使函数增加活力。

(4)指针与数组结合，使引用数组元素的形式更加多样、访问数组元素的手段更加灵活。

(5)熟练正确应用指针能写特别紧凑高效的程序。

6.1 指针基本概念

●地址和指针的概念

- ♦内存按字节编址，每个字节单元都有一个地址。
- ♦程序中定义的任何变量，在编译时都会在内存中分配一个确定的地址单元。
- ♦我们怎样知道机器将某种数据放在内存的什么地方呢？可用求地址运算符&，

例如定义了：int a = 10 ；

则&a就代表变量a在内存中的地址。因为地址运算符&就是取其后变量a的地址。可以用

```
printf("%x\n" , &a);
```

看其地址。注意，这个地址并不是始终不变的，这是由系统来安排的，我们无法预先知道。以后我们所讲的地址都是假设的.

C语言规定：如果变量占用连续的多个字节，则第一个字节的地址就是该变量的地址。例如定义：

```
short a=10;
float b=10;
```

则编译系统给变量分配的内存空间如图6.1所示。从图6.1中可以看出，假设变量a的内存地址为2000，则变量b的内存地址为2002。

程序在引用变量时，首先获得该变量的地址，这还只是变量的首地址，然后还要根据变量的数据类型决定要从首地址开始连续取几个字节来获取变量的值。

若定义如图6.1，现程序要获取变量b的值，则先确定变量首地址为2002，然后由变量b的数据类型float知变量占4个字节，所以从首地址开始连续取4个字节的数据即为变量b的值。

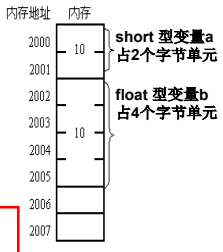


图6.1变量分配的内存示意

● 指针变量和它所指向变量

♦使用一个变量可以直接通过变量名，这种方式称为"直接存取方式"。

♦还可以将变量的地址存入另一“特殊”变量中，然后就可以通过该“特殊”变量来存取变量的值，这种存取变量的方式称为“间接存取方式”。而存放地址的变量就好像存放了一个指针，指向要存取值的变量，故称为“指针变量”。可完整地称为“指向变量的指针变量”。

```
short a=10,b=20,*p=&a;
```

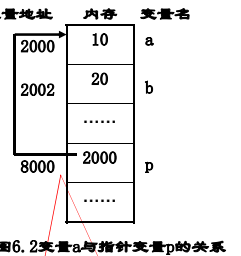


图6.2变量a与指针变量p的关系

这里：a，b是变量，而p是指针变量，存放的是地址

● 指针变量的定义

指针变量也是变量，在使用之前必须先定义。定义时也可对其赋初值。指针变量的定义格式为:

[存储类型] 类型说明符 *pointer[=初值];

表示定义了一个存放某种类型为类型说明符的变量地址的变量，称为pointer。其中，pointer是指针变量的名字，并且必须由字符*作为指针的前导。

由于指针的存储单元是用来存放地址的，该地址必定是某个存储单元的地址。称指针指向该存储单元，并且把指针所指向的存储单元称为指针的目标(对象，object)。简而之，指针变量是专门用来存放变量地址的.

例如:short i; //定义基型变量i
short *ip ;//定义基型指针
short **p;//定义基型指向指针的指针变量p

ip=&i;如果变量i是指针ip的目标，称指针ip指向变量i，简称ip指向i。我们用带箭头的直线从指针ip指向其目标i(如图)。

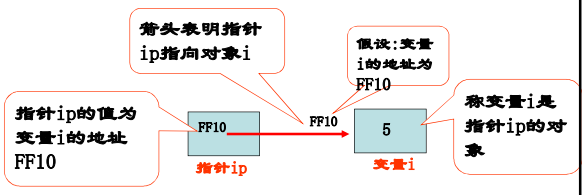


图6.3. 指针变量与它所指变量关系图

在定义指针变量时，应注意以下几点：

(1)类型说明符表示该指针变量所指向的变量的数据类型。如int、float、double、char等。

(2)定义指针变量时，指针变量名前必须有一个“*”号， 表示定义的变量是指针变量,否则就变成了普通整型变量。

(3)指针变量在定义时允许对其赋初值。如:

double c,*p=&c; //令指针p初始化为变量c的地址

需要特别指出的是：这里是用&c对p初始化，而不是对*p初始化。初始化后指针变量p中存放的是实型变量c的地址。

●给指针变量赋值

(1) 指针变量名=&变量名;

作用：

使指针指向一个对象。

♦通过求地址运算符(&)把一个变量的地址赋给指针变量。

“&”是求地址运算符，该运算符为单目运算符，用于求变量的地址，且该变量必须为内存变量。

例如： int a=8,b,*pa,pb;
pa=&a; pb=&b;

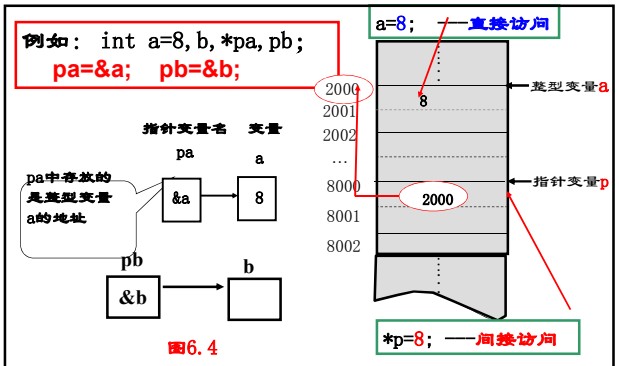


图6.4

当一个指针（如pa）获得某个变量（如a）的地址时，我们称这个指针指向该变量，即使该变量成为指针的对象。假设变量a的地址为2000，则当pa指向a后，pa的值就等于2000。

也可以在定义指针变量的同时赋初值，例如：

int a=8,*p=&a;

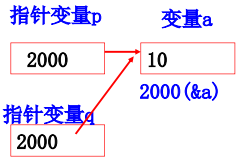
和前面表述的一样，这里是用&a对p初始化,而不是对*p初始化。初始化后指针变量p中存放的是整型变量a的地址。

另外，读者一定还记得在前面调用scanf函数时，其函数调用格式中，输入数据所对应的各变量名之前都必须加运算符& ，这就是我们所说的求地址运算符。scanf函数把从终端读入的数据依次放入这些地址所代表的存储单元中，也就是说scanf函数要求输入项是地址值。因此接上例，scanf(“%d%d”,&a,&b) ；语句和scanf(“%d%d” ，pa，pb)；语句是等价的，都是将终端输入的整型数据存入到变量a和变量b所在的存储单元中。

(2) 同类型指针变量之间可以直接赋值。

可以把指针变量的值赋给指针变量，但一定要确保这两个指针变量的类型是相同的。

例如:float x;
int a=10,*p,*q;
p=&a; /*方式1*/
q=p; /*方式2*/



执行以上语句后，使指针变量*q*也存放了变量*a*的地址，也就是说指针变量*p*和*q*同时指向了变量*a*。如执行 pa=&x ；语句，则是绝对错误的。为什么？请读者想一想！

(3) 给指针变量赋“空”值

为了使指针变量有一确定的数值，除了给指针变量赋一地址值外，当指针变量没有指向的对象时，也可以给指针变量赋NULL值，此值为空值。

例如：int *p ； p=NULL ；指针p 

表示指针变量 p 为空指针，暂不指向任何变量。

对于静态或全局的指针变量，如果在定义时未给它指定初值，系统自动给它指定初值为0，让它暂时是一个空指针。使指针的值等于零(或者等于空)，称为指针清零。

NULL是在stdio.h头文件中定义的预定义符，因此在使用NULL时，应该在程序的前面出现预定义命令行：#include “stdio.h”。

NULL的代码值为0，所以语句p=NULL ;等价于：p=0 ； 都是表示指针变量p是一个空指针，没有指向任何对象。

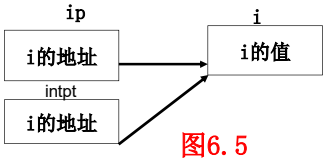
给指针变量赋值举例：

```
int i = 100, j, *ip, *intpt;
float f, *fp;
以下都是不正确的赋值
ip = 100; /* 指针变量不能赋整数值*/
intpt = j;/*指针变量不能赋整型变量的值*/
fp = &i;//能指向float型变量,不能指向int型变量
fp = ip;//不同类型指针变量不能相互赋值
```

即一个指针变量只能指向同一个类型的变量。因为，定义什么样的类型指针变量，该指针变量只能存放什么样类型变量的地址，两者必须一致，否则就可能出现了张冠李戴的错误现象。

而以下都是正确的赋值：

```
ip = &i ;     /* 使ip指向i */
intpt = ip ; //使intpt指向ip所指变量
```



同类型的指针变量可以相互赋值；

图6.5

```
fp  = &f ; /*使fp指向f*/
ip = NULL ;//使ip不再指向任何变量
```

●指针变量的引用

(1) 通过指针或地址引用一个存储单元

当指针变量中存放了一个确切的地址值时，通过指针可以用“间接运算符”（*）来引用该地址所代表的存储单元。即利用指针变量，提供对变量的一种间接访问形式。

(a)在赋值号右边由“*”运算符和指针变量组成的表达式，代表指针所指存储单元的内容。

如代码： int i=8,j,*ip ;
ip = &i;
j = *ip;

第二条语句是把变量i的地址赋给了指针变量ip，第三条语句是把指针变量ip所指向的变量i存储单元的值8赋给变量j。

上述赋值等价于： j = i;

“*”号在这里是一个“间接运算符”，它为单目运算符，与运算对象自右至左结合，且运算对象必须为一个地址对象。

例如： j=*&i ;

该语句中“&”运算符求出变量*i*的地址，“*”运算符取变量*i*地址中的值8赋给变量*j*。

(b) 在赋值号左边由“*”和指针变量组成的表达式，代表指针所指的存储单元

例如： int *p , k=0 ;
p=&k ;
*p=150 ;

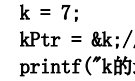
以上第三条语句是把整数150存入变量*k*中。

*p=*p +1 ; 或 *p+=1 ;

以上语句是获取指针变量*p*所指向的存储单元*k*中的值150，然后加1再放入指针变量*p*所指向的存储单元*k*中，此时变量*k*中存放的数值为151。

【例6.1】说明指针变量与它所指变量之间关系的示意程序

```
#include <stdio.h>
int main()
{   int k; //一个整型变量
    int *kPtr; //一个整型指针变量
    k = 7;
    kPtr = &k;//使kPtr指向变量k。
    printf("k的地址是 %x\nkPtr的值是 %x",&k, kPtr);
    printf("\nk的值是 %d\n*kPtr的值是 %d", k ,*kPtr);
    printf("\n以下表明运算符 * 和 & 是互逆的:
\n&*kPtr = %x\n*&kPtr = %x\n",&*kPtr,*&kPtr);
    return 0;
}
```



输出结果：

```
k的地址是 12ff7c
kPtr的值是 12ff7c
k的值是 7
*kPtr的值是 7
以下表明运算符 * 和 & 是互逆的:
&*kPtr = 12ff7c
*&kPtr = 12ff7c
```

记住对变量的两种访问方式：

对变量的两种访问方式：

① 直接访问：按变量地址存取变量值

② 间接访问：通过存放变量地址的变量去访问变量

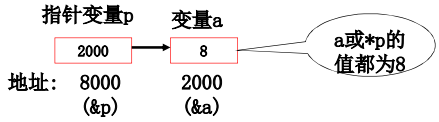


图6.6

为正确理解和使用指针变量，必须注意以下几点：

(1)当定义局部指针变量时，如果未给它指定初值，则它的值是不确定的。程序在使用它们时，应首先给它们赋值，让指针变量确定指向某个变量。

例如： short *pa;

表示只是定义了一个指针，它还没有对象。

例如：printf("%d\n", *pa);

表示访问非法，将是致命的错误，也是初学者易犯的错误。

以下语句正确：

```
short a=3, *pa=&a;
*pa = a + 6; //a=a+6
```

(2) 指针变量只能取变量的地址值

不能将任何其他非地址值赋给指针变量。

例如:int i, *ip = &i; //ip= i;

*ip = i;

通过某个指向变量i的指针变量ip间接引用变量i与直接按变量名i引用变量i，效果是相同的，凡能直接按变量名可引用的地方，也可以用指向它的某个指针变量间接引用它。例如，有,则凡变量i能使用的地方，*ip一样能用。

(3)单目运算符*、&、++和--是从右向左结合的。注意分清运算对象是指针变量、还是指针变量所指对象。

例如，有变量定义

int i=200, j, *ip = &i ;

代码 j = ++*ip;//j=?

先是*ip，间接引用ip所指向的变量(变量i)，然后对变量i的自增运算，并将运算结果赋值给变量j。

++*ip,++(*ip) 表示ip指向的元素值加1。

ip++,ip+=1; ip指向下一个元素。

*ip++, *(ip++) 先取*ip，再使ip加1。

ip-- ,(ip--) 先取*ip，再使ip减1。

*(--ip)； 先使ip减1，再取*ip。

*(++ip); 先使ip加1，再取*ip。

由于“++”在指针变量ip的前面，属于前置运算，出现在表达式中时遵循“先加后用”的使用规则.

例如: j = *ip++; 或 j=*(ip++);

因为“*”运算符和“++”运算符同优先级，而结合方向为“自右至左”（右结合性），即它相当于*(ip++)。而“++”在指针变量p的后面，属于后置运算，出现在表达式中时遵循“先用后加”的使用规则.

即表达式*ip++的值与*ip相同，并在求出表达式值的同时，ip增加了1个单位。相当于代码

j = *ip; ip++;

经上述表达式计算后，ip不再指向变量i。

这种情况常用在指针指向数组元素的情况，在引用数组某元素之后，自动指向数组的下一个元素。

代码 j = (*ip)++ ;

则是先间接引用ip所指向的变量，取这个变量的值赋值给j，并让该变量自增。

指针变量的引用小结

假设: int a=8,*p=&a;

&a	表示变量a的地址；假设&a的地址为:2000,a=8;
p	指针变量p的值，p=&a
*p	表示指针变量p指向的变量*p=a=8
*&a	相当于*(&a)，有*(&a)＝*p＝a=8
&*p	相当于&(*p)，有&*p=&(*p)=&a=2000
&p	表示指针变量p的地址，若有&p=8000
*&p	相当于*(&p)，有*(&p)=2000
&*a	写法有误，因* 后只能接指针变量

6.2 指向数组元素的指针

- 数组在内存中占用连续的存储空间，数组名代表的是数组的首地址。
- 可定义一个指针变量，通过赋值或函数调用参数传递的方式，把数组名或数组的第1个元素的地址赋值给该指针变量，该指针变量就指向了该数组首元素。
- 当一个指针指向数组首元素后，对数组所有元素的访问，既可以使用数组下标，也可以使用指针。

例如：int a[7], *p,*q;

p=a;

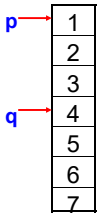
表示：把a[0]元素的地址赋给指针变量p，即p是指向数组a的首地址。

q = &a[3] ;

表示：把a[1]元素的地址赋给指针变量 p，即p是指向数组a的下标为3的元素的指针。

好处：

1. 为引用数组元素提供了一种新的途径；
2. 比用下标引用数组元素更灵活和简洁，因为指针有一定的运算能力。



例如:

```
#include <stdio.h>
void main()
{  short a[]={1,2,3,4,5,6,7};
   short  i,*p=a;  //*p=&a[0];
   for(i=0;i<7;i++)
       printf("%4d",*p++);
  printf("\n");
}
```

使用指针时，应尽量避免指针访问越界。

在上例循环执行后，p已经越过数组的范围，如图所示，这时它所指向的单元的值是不确定的，但编译器不能发现该问题，避免指针访问越界是程序员自己的责任。

在定义的同时为指针变量赋初值

例1：

```
int  a[10];
int  *p = &a[0];
int  *q = &a[8];
```

例2：

```
int  b[2][5];
int  *p=&b[0][0];
int  *q=&b[1][3];
```

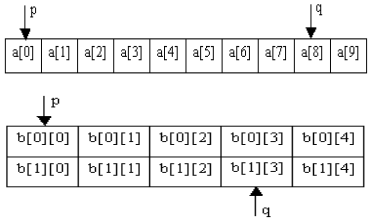


图6.7

对指向数组元素的指针允许作有限的运算：

例如： int *p, *q, a[10];

p = &a[1];

q = &a[5];

(1)当两个指针指向同一个数组的元素时，这两个指针可以作关系比较

(< , <=, ==, >, >=, !=)。

若 p==q 表示p，q指向数组的同一个元素；

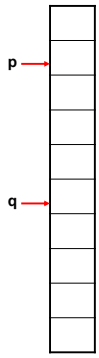
p<q 表示p所指向的数组元素的下标小于q所指向的数组元素的下标。

p>q 表示p所指向的数组元素的下标大于q所指向的数组元素的下标。

若两个指针要做减法运算,要求这两个指针必须指向同一个数组,运算的结果是它们所指向数组元素下标相差的整数.

p-q; 结果为整数-4

q-p; 结果为整数4



(2) 指向数组元素的指针可与整数进行加减运算，使指针指向在数组元素之间前后移动

由于数组元素在内存中顺序连续存放，数组名代表的是数组的首地址。因此表达式a+1为a[1]的地址，a+2为a[2]的地址。

当用一个指针指向数组首元素后，对数组所有元素的访问，既可以使用数组下标，也可以使用指针.

若指针 p = &a[0]; 则表达式 p+n 的值为 a[n]的地址。或者说，p+n的值为指向a[n]的指针值。

注意： 在使用指针过程中，通过移动指针（p）来实现对不同数据单元的访问操作。

p+1不是将p的值和1简单地相加。对不同的类型，移动的单位长度不同。单位长度一般是指针所指向的变量的数据类型长度。

如果数组元素是短整型（short），p+1表示p的地址加2；如果数组元素是实型（float），或整型（int），p+1表示p的地址加4；如果数组元素是字符型，p+1表示p的地址加1。

(3) 引用数组元素的方法

(1)当指针变量p指向一维数组a的第一个元素a[0] ，如p=&a[0]；数组的第i+1个数组元素a[i]有如下写法：

- 下标法，如a[i]
- 数组名，如 *(a+i)表示a[i]。
- 指针变量，如 *(p+i)，或p[i]。

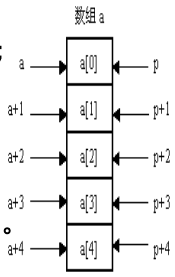


图6.8

同理:a[i]的地址也对应有如下写法：

&a[i]， &p[i]，a+i ，(p+i)

例如：

```
int a[100], b[100], *p, *q;
for(p = a; p<a+100;)scanf("%d",p++);
/*利用指针遍历数组，输入数组的全部元素*/
for(p=a;p<=&a[99];p++)
    printf("*p = %d\t", *p);
/*利用指针遍历数组，输出数组的全部元素*/
printf("\n");
for(p = a, q = b; p < a + 100;)*q++=*p++;
/* 利用指针将已知数组复制到另一个数组*/
```

(2) 指向一维数组非首元素的指针

假设指针p指向一维数组a的第3个元素a[2]，则：如右图所示。

a) p[1]： 使p指向下一个元素a[3]。(p+1)

b) p[-1]： 使p指向前一个元素a[1]。(p-1)

同理:

a) p[i]： 使p指向元素a[2+i]。 (p+i)

d) p[-i]：使p指向元素a[2-i] (p-i)

由上可知：当指针变量p指向一维数组a的第n个元素a[n]以后，数组的第i个数组元素a[i]有如下写法：

a[i] ,p[n±i],*(a+i),*(p+n±i)

如:p=&a[4];则p[i]引用的是a[4+i],

则: p[-2],引用的是a[2].

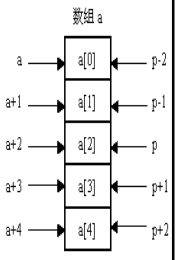


图6.9

同理:a[i]的地址也对应有如下写法：

&a[i]， &p[n±i]，a+i ，(p+n±i)

在实际应用中，一般是定义指向一维数组“首元素”的指针来对数组进行操作。

再次提醒注意：定义指向数组元素的指针变量，其指针类型应与数组元素类型相同。

例如定义：

```
float  a[10];
int  *p = &a[0];
```

中的第二个语句就是错误的，因为定义的指针p是一个整型指针，它是指向整型数据的，而数组元素是实型数据，类型不相同。所以使用时千万要小心。

用数组名与指针的区别：

使用指针访问数组元素，应注意的问题是：

若指针p指向数组a首元素，虽然p+i与a+i、*(p+i)与*(a+i)意义相同，但仍应注意p与a的区别。

- a代表数组的首地址，是常量，不变的，例如语句 for(p=a; a<(p+10); a++) printf("%d", *a);

企图通过语句a++来改变a的值是不合法的。

- p是一个指针变量，可指向数组中的任何元素，例如p++，故要注意指针变量的当前值。

● 通过指针引用二维数组元素举例

当p指向二维数组的首元素后，p+1将指向数组第2个元素，p+2将指向数组第3个元素，……，依此类推。例如定义：

```
int  a[2][5];
int  *p=&a[0][0];
```

后，p指向二维数组a的首元素，如图6.10所示。若将数组元素用p表示出来，则如图6.11所示。

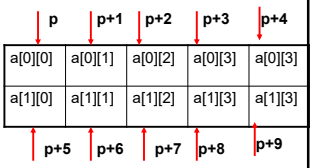
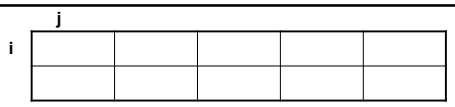


图6.10

*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)
*(p+5)	*(p+6)	*(p+7)	*(p+8)	*(p+9)

图6.11用指针变量p表示的二维数组元素



对图进行观察，可知：

a[i][j]的地址为p+i*5+j，

a[i][j]可表示为*(p+i*5+j)。

由此推出一般性的结论如下：

假设指针变量p已经指向共有M行N列的数组A的首元素，则

a[i][j]的地址为p+i*N+j，

a[i][j]可表示为*(p+i*N+j)。

其中　0≤i<M(行数)，

0≤j<N(列数)。

例：　用指针法求二维数组的最大值。

程序清单如下：

```
#include <stdio.h>
#define M 2
#define N 5
void  main()
{
    int a[M][N], max, i, j;
    int *p=&a[0][0]; /*通过赋初值使p指向a数组首元素*/
    printf("请输入数组中各元素的值：\n");
     for(i=0;i<M;i++)
        for(j=0;j<N;j++)
            scanf("%d",p++);
    /*通过p++依次引用各数组元素地址*/
```

```
max=a[0][0];          /*首先认为第一元素的值是最大值*/
p=&a[0][0];           /*通过赋值使p指向a数组首元素*/
for(i=0;i<M;i++)
    for(j=0;j<N;j++)
        if(max<*(p+i*N+j)) /*max与各数组元素进行比较*/
            max=*(p+i*N+j); /*max总是存放比较后的大者*/

printf("数组中的最大值为：%d\n",max);/*输出最大值*/
}
```

指针与字符串

当字符串存放在数组中，其处理方法与一维数组的处理方法基本一致，可以先定义一个字符类型的指针变量，通过赋初值或赋值的方式让指针变量指向字符数组首元素。然后就可以使用这个指针变量处理单个字符，也可以用它一次性地处理整个字符串，此种方法不再详细讨论。本节主要讨论利用字符类型的指针变量处理字符串常量的方法。

● 1．使指针变量指向“字符串”的方法

将字符类型的指针变量指向字符串有两种方法：

(1) 在定义的同时给指针变量赋初值。例如：

```
void main()
{
    char  *string="I love  China! ";
    printf("%s\n",string);
}
```

(2) 给指针变量赋值。例如：

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

上述两种方式，实际上都是把字符串的第一个字符string的地址赋给指针变量

● 2. 指向字符串常量的指针变量的使用

(1) 把字符串当作整体来处理

这种方式通常用在字符串的的输入和输出中，格式如下：

1. 输入字符串：
   (a) scanf("%s",指针变量);
   (b) gets(指针变量);
2. 输出字符串：
   (a) printf("%s",指针变量);
   (b) puts(指针变量);

(2) 处理字符串中的单个字符

若指针变量已经指向了字符串常量，则用指针变量表示的第i个字符为：*（指针变量+i）

●3.使用字符指针变量与字符数组的区别

(1) 存储内容不同。

字符指针变量本身是一个变量，用于存放字符串的首地址。而字符串本身是存放在以该首地址为首的一块连续的内存空间中并以'\0'作为串的结束。字符数组是由若干个数组元素组成的，每个元素中放一个字符。

(2) 赋值方式不同。

对字符指针变量，可采用下面方法赋值：

如：

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

而对字符数组只能对各个元素逐个赋值，不能用以下办法对字符数组赋值：

```
char s[20];
s="I love  China!";
```

(3) 指针变量的值可以改变；而数组名代表的是该数组的首地址，是常量,它的值是不能改变的。

例

```
#include <stdio.h>
void main()
{
  char *p="Gold medal";
  p=p+5;✓
  printf("%s",p);
}
```

例

```
#include <stdio.h>
void main()
{ char a[]="Gold medal";
  a=a+5;✘
  printf("%s",a);
}
```

(4) 内存分配有区别：

- ♦如果定义了一个字符数组，在编译时自动为它分配内存单元，它有确定的地址。
- ♦指针在定义时，若没有人为的给它分配内存或使它指向确定的内存单元，则该指针指向的内存位置是不确定的。(这点也是初学者常犯的错误，请通过后面的课深入掌握)

```
#include <stdio.h>
void main()
{  char p[10];
   scanf("%s",p);✓
   printf("%s", p);
}
```

```
#include <stdio.h>
void main()
{  char *p;
   scanf("%s",p);✘
   printf("%s", p);
}
```

?

前面说过，当一个指针变量在未取得确定地址前使用是危险的，容易引起错误。但是对指针变量直接赋值是可以的。因为C系统对指针变量赋值时要给以确定的地址。因此，

```
char *cp1, *cp2 = "I am a string";
/*用字符串常量初始化字符指针变量或者*/
cp1 = "Another string";
/*用字符串常量赋值给字符指针变量*/
```

都是合法的。

使字符指针变量cp2指向字符串常量

"I am a string"的第一个字符I，使cp1指向字符串常量"Another string "的第一个字符A。以后就可通过cp2或cp1分别访问这两个字符串常量中的其它字符。

例如，*cp2或cp2[0]就是‘I’，*(cp1+3)或cp1[3]就是字符‘t’。

要特别强调指出，企图通过指针变量修改存于常量区中的字符串常量是不允许的。

例如:

将cp2[0]中'I'改成'y',将会出现致命的错误.

例如，调用库函数strlen()求字符串常量的长度：

strlen("I am a string.")

该函数调用的结果是14，表示这个字符串常量由14个有效字符组成。

如果s是一个字符数组，其中已存有字符串，cp是一个字符指针变量，它指向某个字符串的首字符:

```
char s[] = "I am a string.",
     *cp = "Another string.";
```

则代码： printf("%s\n",s);

输出存于字符数组s中的字符串。

而代码： printf("%s\n",cp);

输出字符指针变量cp所指向的字符串。

【例6.2】字符串复制：实现将字符数组s2中的字符串复制到字符数组s1中。

根据前面介绍的知识，用指针指向数组元素后，访问某个数组元素时有四种方法，我们因此可以写出四种实现代码。

(1)用数组名下标来访问数组元素。

```
#include <stdio.h>
void main( )
{
     int i=0;
     char s1[20],s2[20]={"How are you!"};
     for(;(s1[i]=s2[i]) !='\0';i++);
     puts(s1);
}
```

由于字符串结束符‘\0’的值为0，上述测试当前复制字符是不是字符串结束符的代码中，“!='\0'”是多余的，字符串复制更简洁的写法是：

for(;s1[i]=s2[i];i++);

也可用:while(s1[i]=s2[i])i++;

(2)用指针移位或指针名下标来访问数组元素。

```
#include <stdio.h>
void main( )
{    int i=0;
     char s1[20],s2[20]={"How are you!"};
     char *from=s2,*to=s1;
     while(*to++=*from++);
 /*whie(to[i]=from[i])i++;指针名接下标*/
     puts(s1);
}
```

(3)用指针名加偏移量计算出的地址来访问数组元素

```
#include <stdio.h>
void  main( )
{    int i=0;
     char s1[20],s2[20]={"How are
you!"};
     char *to=s1,*from=s2;
     while(*(to+i)=*(from+i))i++;
     puts(s1);
}
```

(4)用数组名加偏移量计算出的地址来访问数组元素

```
#include <stdio.h>
void   main( )
{
     int i=0;
     char s1[20],s2[20]={"How are you!"};
     while(*(s1+i)=*(s2+i))i++;
     puts(s1);
}
```

【例6.3】将字符串s中的某种字符去掉，假设要去掉的字符与字符变量c中的字符相同。

一边考察字符，一边复制不去掉的字符来实现。引入字符指针p和q，其中p指向当前正要考察的字符，若它所指的字符与c中的字符不相同，则应将它复制到新字符串中。否则，该字符不被复制，也就从新的字符串中去掉了；q指向下一个用于存储复制字符的存储位置。每次复制一个字符后，q增加1。每次考察了一个字符后，p就增1。

```
for(p = q = s; *p; p++)
    if (*p != c)  *q++ = *p; /* 复制 */
*q = '\0';     /* 重新构成字符串 */
```

6.3　指针与函数

指针与函数的结合使编程更为灵活。

- 指针可以用作函数的参数(传递地址值)
- 函数也可以返回一个指针类型的值，
- 甚至可以定义指向函数的指针。

- 1．函数形参为指针，实参为地址表达式

因为地址表达式的值是一个地址值，所以函数调用时，实参地址表达式传递给形参指针变量的值是地址，故参数传递后实参与对应的形参指向同一个单元，函数体内对形参的任何操作就相当于是对实参的操作。

【例6.5】说明指针形参用法的示意程序。

程序中的函数swap()的功能是交换两个整型变量的值，函数swap()设置了两个整型指针形参。

在函数swap()的体中，用指针形参间接引用它们所指向的变量。

调用函数swap()时，提供的两个实参必须是要交换值的两个变量的指针，而不是变量的值。

```
#include <stdio.h>
int main()
{  int a=1,b=2;  void swap(int *,int *);
   printf("调用swap函数之前：a = %d\tb = %d\n", a, b);
   swap(&a,&b);/*以变量的指针为实参，而不是变量的值 */
   printf("调用swap函数之后：a = %d\tb = %d\n", a, b);
   return 0;
}
void swap(int *pu, int *pv)/* 函数设置两个指针形参 */
{  int t;
   t = *pu; /*函数体通过指针形参，间接引用和改变调用环境中的变量*/
   *pu=*pv;
   *pv = t;
}
```

调用 swap 函数之前：　a = 1　　b = 2

调用 swap 函数之后：　a = 2　　b = 1

&a　　&b

【例6.6】对于非指针类型形参，实参向形参传值，函数不能改变实参变量值的示意程序。

```
#include <stdio.h>
void paws(int u, int v)
{  int t = u;
   u = v;   v = t;
   printf("在函数 paws 中：u = %d\tv = %d\n", u, v);
}
int main()
{  int x = 1, y = 2;
   paws(x, y);
   printf("在主函数 main 中：x = %d\ty = %d\n", x, y);
   return 0;
}
```

在函数 paws 中：u = 2　　v = 1

在主函数 main 中：x = 1　y = 2

例子说明，当实参向形参单向传递值，函数执行时，形参值的改变不会影响实参变量。

希望函数能按需要改变由实参指定的变量，需要在三个方面协调一致。

(1)函数应设置指针形参；

(2)函数体必须通过指针形参间接引用变量；

(3)调用函数时，必须以希望改变值的变量的指针为实参。

/*用指针编写函数，该函数的功能是删除两个整型变量的最大公因子*/

```
void delgcd(int *a,int *b)
{   int x,y,r;
    x=r=*a;
    y=*b;
    do{
       x=y;
       y=r;
       r=x%y;
     }while(r);
    *a/=y;//从*a消去最大最大公因子
    *b/=y;//从*b消去最大最大公因子
}
```

```
void main()
{ int u=15,v=24;
  delgcd(&u,&v);
  printf("u=%d,v=%d\n",u,v);
}
```

读程序，回答程序的输出结果。

```
#include <stdio.h>
void f1(int x, int y)    { int t = x; x = y; y = t;    }
void f2(int *x, int *y)  { int t = *x; *x = *y; *y = t;}
int main()
{  int x = 1, y = 2;    int *xpt = &x, *ypt = &y;
   printf("First: x = %d\ty = %d\n", x, y);   f1(x, y);
   printf("After call f1(): x = %d\ty = %d\n", x, y);
   x = 1; y = 2;    f2(&x, &y);
   printf("After call f2(): x = %d\ty = %d\n", x, y);
}
```