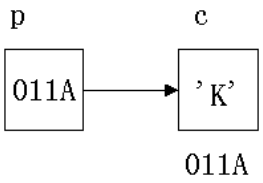
10　指针

指针是C语言中广泛使用的一种数据类型。运用指针编程是C语言最主要的风格之一。利用指针变量可以表示各种数据结构；能很方便地使用数组和字符串；并能象汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了C语言的功能。学习指针是学习C语言中最重要的一环，能否正确理解和使用指针是我们是否掌握C语言的一个标志。同时，指针也是C语言中最为困难的一部分，在学习中除了要正确理解基本概念，还必须要多编程，上机调试。只要作到这些，指针也是不难掌握的。

1.1 地址指针的基本概念

在计算机中，所有的数据都是存放在存储器中的。一般把存储器中的一个字节称为一个内存单元，不同的数据类型所占用的内存单元数不等，如整型量占 2 个单元，字符量占 1 个单元等，在前面已有详细的介绍。为了正确地访问这些内存单元，必须为每个内存单元编上号。根据一个内存单元的编号即可准确地找到该内存单元。内存单元的编号也叫做地址。

既然根据内存单元的编号或地址就可以找到所需的内存单元，所以通常也把这个地址称为指针。 内存单元的指针和内存单元的内容是两个不同的概念。 可以用一个通俗的例子来说明它们之间的关系。我们到银行去存取款时， 银行工作人员将根据我们的帐号去找我们的存款单， 找到之后在存单上写入存款、取款的金额。在这里，帐号就是存单的指针， 存款数是存单的内容。对于一个内存单元来说，单元的地址即为指针，其中存放的数据才是该单元的内容。在C语言中，允许用一个变量来存放指针，这种变量称为指针变量。因此，一个指针变量的值就是某个内存单元的地址或称为某内存单元的指针。



图中，设有字符变量C，其内容为"K"(ASCII 码为十进制数 75)，C 占用了 011A 号单元(地址用十六进数表示)。设有指针变量 P，内容为 011A，这种情况我们称为 P 指向变量 C，或说 P 是指向变量 C 的指针。

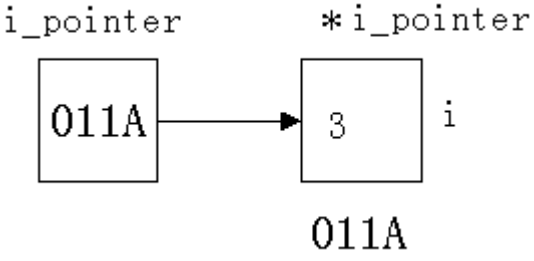
严格地说，一个指针是一个地址，是一个常量。而一个指针变量却可以被赋予不同的指针值，是变量。但常把指针变量简称为指针。为了避免混淆，我们中约定："指针"是指地址，是常量，"指针变量"是指取值为地址的变量。定义指针的目的是为了通过指针去访问内存单元。

既然指针变量的值是一个地址，那么这个地址不仅可以是变量的地址，也可以是其它数据结构的地址。在一个指针变量中存放一个数组或一个函数的首地址有何意义呢？ 因为数组或函数都是连续存放的。通过访问指针变量取得了数组或函数的首地址，也就找到了该数组或函数。这样一来，凡是出现数组，函数的地方都可以用一个指针变量来表示，只要该指针变量中赋予数组或函数的首地址即可。这样做，将会使程序的概念十分清楚，程序本身也精练，高效。在C语言中，一种数据类型或数据结构往往都占有一组连续的内存单元。 用"地址"这个概念并不能很好地描述一种数据类型或数据结构，而"指针"虽然实际上也是一个地址，但它却是一个数据结构的首地址，它是"指向"一个数据结构的，因而概念更为清楚，表示更为明确。 这也是引入"指针"概念的一个重要原因。

1.1 变量的指针和指向变量的指针变量

变量的**指针**就是变量的**地址**。存放变量**地址**的变量是**指针变量**。即在C语言中，允许用一个变量来存放指针，这种变量称为指针变量。因此，一个指针变量的值就是某个变量的地址或称为某变量的指针。

为了表示指针变量和它所指向的变量之间的关系，在程序中用"*"符号表示"指向"，例如，i_pointer 代表指针变量，而*i_pointer 是 i_pointer 所指向的变量。



因此，下面两个语句作用相同：

```
i=3;
*i_pointer=3;
```

第二个语句的含义是将 3 赋给指针变量 i_pointer 所指向的变量。

1.1.1 定义一个指针变量

对指针变量的定义包括三个内容：

(1) 指针类型说明，即定义变量为一个指针变量；

(2) 指针变量名；

(3) 变量值(指针)所指向的变量的数据类型。

其一般形式为：

```
类型说明符　*变量名;
```

其中，*表示这是一个指针变量，变量名即为定义的指针变量名，类型说明符表示本指针变量所指向的变量的数据类型。

例如：　　int *p1;

表示 p1 是一个指针变量，它的值是某个整型变量的地址。或者说 p1 指向一个整型变量 。至于 p1 究竟指向哪一个整型变量，应由向 p1 赋予的地址来决定。

再如：

```
int *p2;          /*p2 是指向整型变量的指针变量*/
float *p3;        /*p3 是指向浮点变量的指针变量*/
char *p4;         /*p4 是指向字符变量的指针变量*/
```

应该注意的是，一个指针变量只能指向同类型的变量，如 P3 只能指向浮点变量，不能时而指向一个浮点变量，时而又指向一个字符变量。

1.1.1 指针变量的引用

指针变量同普通变量一样，使用之前不仅要定义说明，而且必须赋予具体的值。未经赋值的指针变量不能使用，否则将造成系统混乱，甚至死机。指针变量的赋值只能赋予地址，决不能赋予任何其它数据，否则将引起错误。在 C 语言中，变量的地址是由编译系统分配的，对用户完全透明，用户不知道变量的具体地址。

两个有关的运算符：

1)　&:取地址运算符。

2)　*：指针运算符（或称"间接访问"运算符）。

C 语言中提供了地址运算符&来表示变量的地址。

其一般形式为：

```
&变量名;
```

如&a 表示变量 a 的地址，&b 表示变量 b 的地址。变量本身必须预先说明。

设有指向整型变量的指针变量 p，如要把整型变量 a 的地址赋予 p 可以有以下两种方式：

(1) 指针变量初始化的方法

```
int a;
int *p=&a;
```

(2) 赋值语句的方法

```
int a;
int *p;
p=&a;
```

不允许把一个数赋予指针变量，故下面的赋值是错误的：

```
int *p;
p=1000;
```

被赋值的指针变量前不能再加“*”说明符，如写为*p=&a 也是错误的。

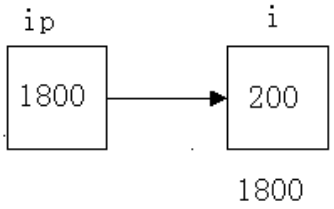
假设:

```
  int i=200, x;
  int *ip;
```

我们定义了两个整型变量 i,x,还定义了一个指向整型数的指针变量 ip。i,x 中可存放整数,而 ip 中只能存放整型变量的地址。我们可以把 i 的地址赋给 ip:

```
ip=&i;
```

此时指针变量 ip 指向整型变量 i,假设变量 i 的地址为 1800,这个赋值可形象理解为下图所示的联系。



以后我们便可以通过指针变量 ip 间接访问变量 i,例如:

```
x=*ip;
```

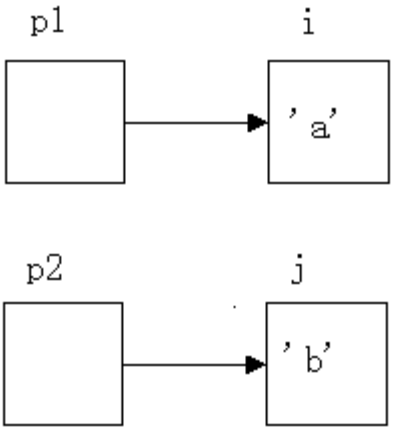
运算符*访问以 ip 为地址的存贮区域,而 ip 中存放的是变量 i 的地址,因此,*ip 访问的是地址为 1800 的存贮区域(因为是整数,实际上是从 1800 开始的两个字节),它就是 i 所占用的存贮区域, 所以上面的赋值表达式等价于

```
x=i;
```

另外,指针变量和一般变量一样,存放在它们之中的值是可以改变的,也就是说可以改变它们的指向,假设

```
  int i,j,*p1,*p2;
  i='a';
  j='b';
  p1=&i;
  p2=&j;
```

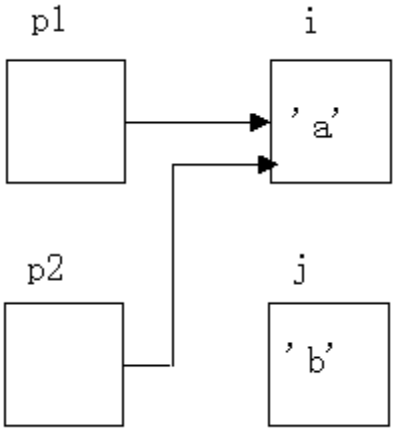
则建立如下图所示的联系:



这时赋值表达式:

p2=p1

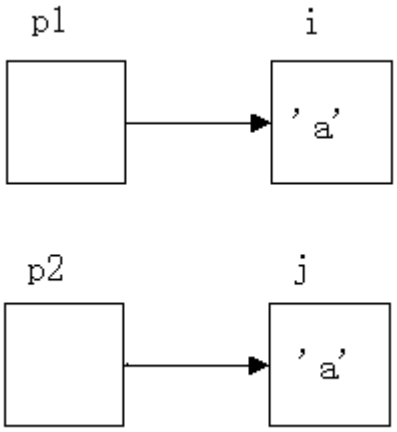
就使 p2 与 p1 指向同一对象 i,此时*p2 就等价于 i,而不是 j,图所示:



如果执行如下表达式:

*p2=*p1;

则表示把 p1 指向的内容赋给 p2 所指的区域, 此时就变成图所示



通过指针访问它所指向的一个变量是以间接访问的形式进行的,所以比直接访问一个变量要费时间,而且不直观,因为通过指针要访问哪一个变量,取决于指针的值(即指向),例如

"*p2=*p1;"实际上就是"j=i;",前者不仅速度慢而且目的不明。但由于指针是变量,我们可以通过改变它们的指向,以间接访问不同的变量,这给程序员带来灵活性,也使程序代码编写得更为简洁和有效。

指针变量可出现在表达式中, 设

int x,y，*px=&x;

指针变量 px 指向整数 x,则*px 可出现在 x 能出现的任何地方。例如:

y=*px+5;　/*表示把 x 的内容加 5 并赋给 y*/

y=++*px;　/*px 的内容加上 1 之后赋给 y，++*px 相当于++(*px)*/

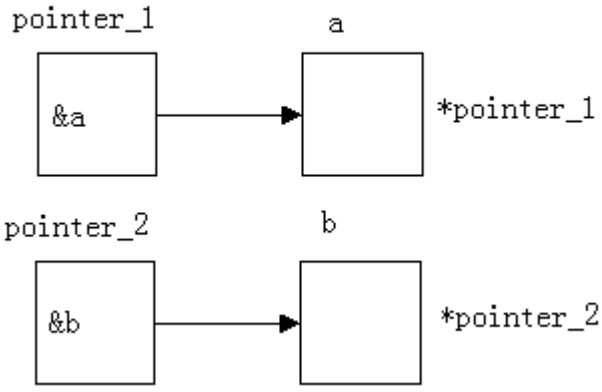
y=*px++;　/*相当于 y=*px; px++*/

【例 10.1】

```
main()
{ int a,b;
  int *pointer_1, *pointer_2;
  a=100;b=10;
  pointer_1=&a;
  pointer_2=&b;
  printf("%d,%d\n",a,b);
  printf("%d,%d\n",*pointer_1, *pointer_2);
}
```

对程序的说明：

1) 在开头处虽然定义了两个指针变量 pointer_1 和 pointer_2，担它们并未指向任何一个整型变量。只是提供两个指针变量，规定它们可以指向整型变量。程序第 5、6 行的作用就是使 pointer_1 指向 a，pointer_2 指向 b。



2) 最后一行的*pointer_1 和*pointer_2 就是变量 a 和 b。最后两个 printf 函数作用是相同的。

3) 程序中有两处出现*pointer_1 和*pointer_2，请区分它们的不同含义。

4) 程序第 5、6 行的"pointer_1=&a"和 "pointer_2=&b"不能写成"*pointer_1=&a"和"*pointer_2=&b"。

请对下面再的关于“&"和"*"的问题进行考虑：

1) 如果已经执行了“pointer_1=&a; "语句，则&*pointer_1 是什么含义?

2)　*&a 含义是什么？
3)　(pointer_1)++和 pointer_1++的区别？

【例 10.2】输入 a 和 b 两个整数，按先大后小的顺序输出 a 和 b。

```
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("\na=%d,b=%d\n",a,b);
  printf("max=%d,min=%d\n",*p1, *p2);
}
```

1.1.1 指针变量作为函数参数

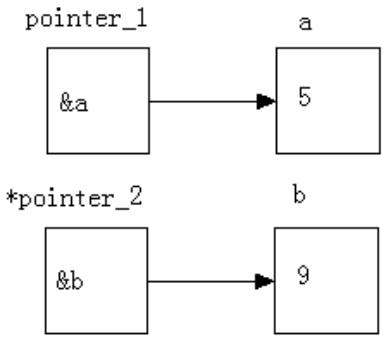
函数的参数不仅可以是整型、实型、字符型等数据，还可以是指针类型。它的作用是将一个变量的地址传送到另一个函数中。

【例 10.3】题目同例 10.2，即输入的两个整数按大小顺序输出。今用函数处理，而且用指针类型的数据作函数参数。

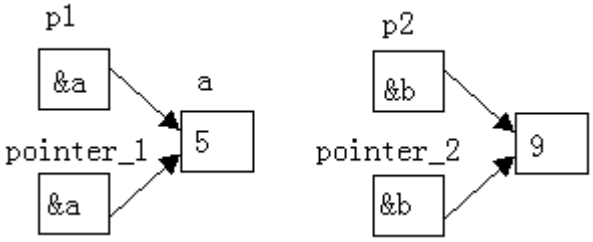
```
swap(int *p1,int *p2)
{int temp;
 temp=*p1;
 *p1=*p2;
 *p2=temp;
}
main()
{
  int a,b;
  int *pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;pointer_2=&b;
  if(a<b) swap(pointer_1,pointer_2);
  printf("\n%d,%d\n",a,b);
  }
```

对程序的说明：

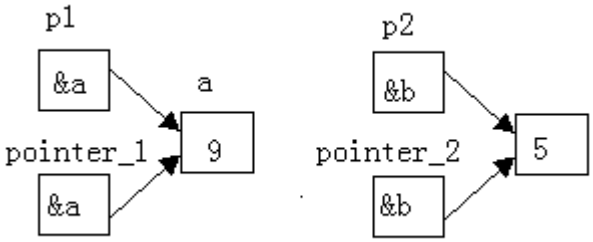
swap 是用户定义的函数，它的作用是交换两个变量（a 和 b）的值。swap 函数的形参 p1、p2 是指针变量。程序运行时，先执行 main 函数，输入 a 和 b 的值。然后将 a 和 b 的地址分别赋给指针变量 pointer_1 和 pointer_2，使 pointer_1 指向 a，pointer_2 指向 b。



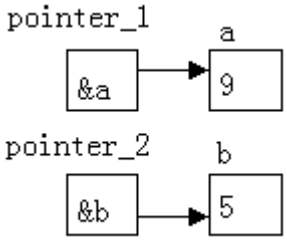
接着执行 if 语句，由于 a〈b，因此执行 swap 函数。注意实参 pointer_1 和 pointer_2 是指针变量，在函数调用时，将实参变量的值传递给形参变量。采取的依然是"值传递"方式。因此虚实结合后形参 p1 的值为&a，p2 的值为&b。这时 p1 和 pointer_1 指向变量 a，p2 和 pointer_2 指向变量 b。



接着执行执行 swap 函数的函数体使*p1 和*p2 的值互换，也就是使 a 和 b 的值互换。



函数调用结束后，p1 和 p2 不复存在（已释放）如图。



最后在 main 函数中输出的 a 和 b 的值是已经过交换的值。

请注意交换*p1 和*p2 的值是如何实现的。请找出下列程序段的错误：

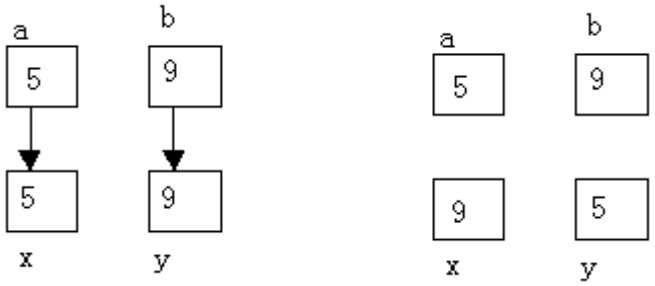
```
swap(int *p1,int *p2)
{int *temp;
 *temp=*p1;       /*此语句有问题*/
 *p1=*p2;
 *p2=temp;
```

```
}
```

请考虑下面的函数能否实现实现 a 和 b 互换。

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

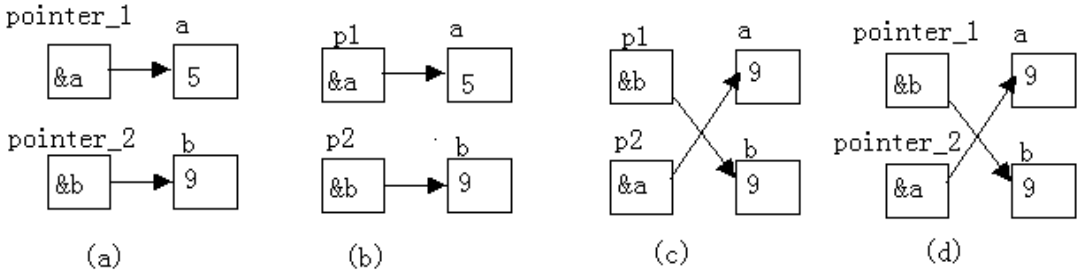
如果在 main 函数中用"swap(a,b);"调用 swap 函数，会有什么结果呢？请看下图所示 。



【例 10.4】请注意，不能企图通过改变指针形参的值而使指针实参的值改变。

```
swap(int *p1,int *p2)
{int *p;
 p=p1;
 p1=p2;
 p2=p;
}
main()
{
  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("\n%d,%d\n",*pointer_1,*pointer_2);
  }
```

其中的问题在于不能实现如图所示的第四步（d）。



【例 10.5】输入 a、b、c3 个整数，按大小顺序输出。

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

1.1.1 指针变量几个问题的进一步说明

指针变量可以进行某些运算，但其运算的种类是有限的。它只能进行赋值运算和部分算术运算及关系运算。

1. 指针运算符

1) 取地址运算符&:取地址运算符&是单目运算符，其结合性为自右至左，其功能是取变量的地址。在 scanf 函数及前面介绍指针变量赋值中，我们已经了解并使用了&运算符。

2) 取内容运算符*:取内容运算符*是单目运算符，其结合性为自右至左，用来表示指针变量所指的变量。在*运算符之后跟的变量必须是指针变量。

需要注意的是指针运算符*和指针变量说明中的指针说明符*不是一回事。在指针变量说明中，“*”是类型说明符，表示其后的变量是指针类型。而表达式中出现的“*”则是一个运算符用以表示指针变量所指的变量。

【例 10.6】

```
main(){
  int a=5,*p=&a;
  printf ("%d",*p);
}
```

表示指针变量 p 取得了整型变量 a 的地址。printf("%d",*p)语句表示输出变量 a 的值。

2. 指针变量的运算

1) 赋值运算：指针变量的赋值运算有以下几种形式。

① 指针变量初始化赋值，前面已作介绍。

② 把一个变量的地址赋予指向相同数据类型的指针变量。

例如：

```
int a,*pa;
pa=&a;     /*把整型变量 a 的地址赋予整型指针变量 pa*/
```

③ 把一个指针变量的值赋予指向相同类型变量的另一个指针变量。

如：

```
int a,*pa=&a,*pb;
pb=pa;     /*把 a 的地址赋予指针变量 pb*/
```

由于 pa,pb 均为指向整型变量的指针变量，因此可以相互赋值。

④ 把数组的首地址赋予指向数组的指针变量。

例如：

```
int a[5],*pa;
pa=a;
```

(数组名表示数组的首地址，故可赋予指向数组的指针变量 pa)

也可写为：

```
pa=&a[0];   /*数组第一个元素的地址也是整个数组的首地址，也可赋予 pa*/
```

当然也可采取初始化赋值的方法：

```
int a[5],*pa=a;
```

⑤ 把字符串的首地址赋予指向字符类型的指针变量。

例如：

```
char *pc;
pc="C Language";
```

或用初始化赋值的方法写为：

```
char *pc="C Language";
```

这里应说明的是并不是把整个字符串装入指针变量，而是把存放该字符串的字符数组的首地址装入指针变量。在后面还将详细介绍。

⑥ 把函数的入口地址赋予指向函数的指针变量。

例如：

```
int (*pf)();
pf=f;      /*f 为函数名*/
```

2) 加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n,pa-n,pa++,++pa,pa--,--pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址。而不是在原地址基础上加 1。例如：

```
int a[5],*pa;
pa=a;       /*pa 指向数组 a，也是指向 a[0]*/
pa=pa+2;    /*pa 指向 a[2]，即 pa 的值为&pa[2]*/
```

指针变量的加减运算只能对数组指针变量进行，对指向其它类型变量的指针变量作加减运算是毫无意义的。

3) 两个指针变量之间的运算：只有指向同一数组的两个指针变量之间才能进行运算，否则运算毫无意义。

① 两指针变量相减：两指针变量相减所得之差是两个指针所指数组元素之间相差的元素个数。实际上是两个指针值(地址)相减之差再除以该数组元素的长度(字节数)。例如 pf1 和 pf2 是指向同一浮点数组的两个指针变量，设 pf1 的值为 2010H，pf2 的值为 2000H，而浮点数组每个元素占 4 个字节，所以 pf1-pf2 的结果为(2000H-2010H)/4=4,表示 pf1 和 pf2 之间相差 4 个元素。两个指针变量不能进行加法运算。例如，pf1+pf2 是什么意思呢?毫无实际意义。

② 两指针变量进行关系运算：指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的关系。

例如：

pf1==pf2 表示 pf1 和 pf2 指向同一数组元素；

pf1>pf2 表示 pf1 处于高地址位置；

pf1<pf2 表示 pf2 处于低地址位置。

指针变量还可以与 0 比较。

设 p 为指针变量，则 p==0 表明 p 是空指针，它不指向任何变量；

p!=0 表示 p 不是空指针。

空指针是由对指针变量赋予 0 值而得到的。

例如：

```
#define NULL 0
int *p=NULL;
```

对指针变量赋 0 值和不赋值是不同的。指针变量未赋值时，可以是任意值，是不能使用的。否则将造成意外错误。而指针变量赋 0 值后，则可以使用，只是它不指向具体的变量而已。

【例 10.7】

```
main(){
  int a=10,b=20,s,t,*pa,*pb; /*说明 pa,pb 为整型指针变量*/
  pa=&a;                     /*给指针变量 pa 赋值，pa 指向变量 a*/
  pb=&b;                     /*给指针变量 pb 赋值，pb 指向变量 b*/
  s=*pa+*pb;                 /*求 a+b 之和,(*pa 就是 a,*pb 就是 b)*/
  t=*pa**pb;                 /*本行是求 a*b 之积*/
  printf("a=%d\nb=%d\na+b=%d\na*b=%d\n",a,b,a+b,a*b);
  printf("s=%d\nt=%d\n",s,t);
}
```

【例 10.8】

```
main(){
  int a,b,c,*pmax,*pmin;              /*pmax,pmin 为整型指针变量*/
  printf("input three numbers:\n");   /*输入提示*/
  scanf("%d%d%d",&a,&b,&c);           /*输入三个数字*/
  if(a>b){                            /*如果第一个数字大于第二个数字... */
    pmax=&a;                          /*指针变量赋值*/
    pmin=&b;}                         /*指针变量赋值*/
```

```
  else{
    pmax=&b;                          /*指针变量赋值*/
    pmin=&a;}                         /*指针变量赋值*/
  if(c>*pmax) pmax=&c;                /*判断并赋值*/
  if(c<*pmin) pmin=&c;                /*判断并赋值*/
    printf("max=%d\nmin=%d\n",*pmax,*pmin); /*输出结果*/
}
```

1.1 数组指针和指向数组的指针变量

一个变量有一个地址，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。所谓数组的指针是指数组的起始地址，数组元素的指针是数组元素的地址。

1.1.1 指向数组元素的指针

一个数组是由连续的一块内存单元组成的。数组名就是这块连续内存单元的首地址。一个数组也是由各个数组元素(下标变量)组成的。每个数组元素按其类型不同占有几个连续的内存单元。一个数组元素的首地址也是指它所占有的几个内存单元的首地址。

定义一个指向数组元素的指针变量的方法，与以前介绍的指针变量相同。

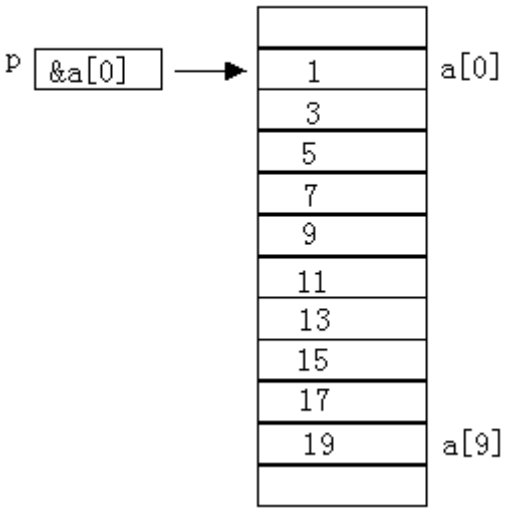
例如：

```
int a[10];     /*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*/
```

应当注意，因为数组为 int 型，所以指针变量也应为指向 int 型的指针变量。下面是对指针变量赋值：

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

把 a[0]元素的地址赋给指针变量 p。也就是说，p 指向 a 数组的第 0 号元素。



C 语言规定，数组名代表数组的首地址，也就是第 0 号元素的地址。因此，下面两个语句等价：

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

在定义指针变量时可以赋给初值：

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

它等效于：

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

当然定义时也可以写成：

```
int *p=a;
```

从图中我们可以看出有以下关系：

p,a,&a[0]均指向同一单元，它们是数组 a 的首地址，也是 0 号元素 a[0]的首地址。应该说明的是 p 是变量，而 a,&a[0]都是常量。在编程时应予以注意。

数组指针变量说明的一般形式为：

类型说明符　*指针变量名；

其中类型说明符表示所指数组的类型。从一般形式可以看出指向数组的指针变量和指向普通变量的指针变量的说明是相同的。

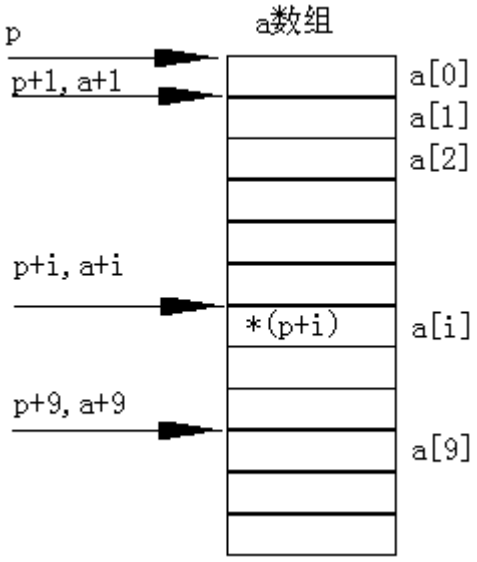
1.1.1 通过指针引用数组元素

C 语言规定：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。

引入指针变量后，就可以用两种方法来访问数组元素了。

如果 p 的初值为&a[0],则：

1) p+i 和 a+i 就是 a[i]的地址，或者说它们指向 a 数组的第 i 个元素。



2) *(p+i)或*(a+i)就是 p+i 或 a+i 所指向的数组元素，即 a[i]。例如，*(p+5)或*(a+5)就是 a[5]。

3) 指向数组的指针变量也可以带下标，如 p[i]与*(p+i)等价。

根据以上叙述，引用一个数组元素可以用：

1) 下标法，即用 a[i]形式访问数组元素。在前面介绍数组时都是采用这种方法。
2) 指针法，即采用*(a+i)或*(p+i)形式，用间接访问的方法来访问数组元素，其中 a 是数组名，p 是指向数组的指针变量，其处值 p=a。

【例 10.9】输出数组中的全部元素。（下标法）

```
main(){
  int a[10],i;
  for(i=0;i<10;i++)
    a[i]=i;
  for(i=0;i<5;i++)
    printf("a[%d]=%d\n",i,a[i]);
}
```

【例 10.10】输出数组中的全部元素。（通过数组名计算元素的地址，找出元素的值）

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

【例 10.11】输出数组中的全部元素。（用指针变量指向元素）

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

【例 10.12】

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

几个注意的问题：

1) 指针变量可以实现本身的值的改变。如 p++是合法的；而 a++是错误的。因为 a 是数组名，它是数组的首地址，是常量。

2) 要注意指针变量的当前值。请看下面的程序。

【例 10.13】找出错误。

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

【例 10.14】改正。

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

3) 从上例可以看出，虽然定义数组时指定它包含 10 个元素，但指针变量可以指到数组以后的内存单元，系统并不认为非法。

4) *p++，由于++和*同优先级，结合方向自右而左，等价于*(p++)。

5) *(p++)与*(++p)作用不同。若 p 的初值为 a，则*(p++)等价 a[0]，*(++p)等价 a[1]。

6) (*p)++表示 p 所指向的元素值加 1。

7) 如果 p 当前指向 a 数组中的第 i 个元素，则

*(p--)相当于 a[i--]；

*(++p)相当于 a[++i]；

*(--p)相当于 a[--i]。

1.1.1 数组名作函数参数

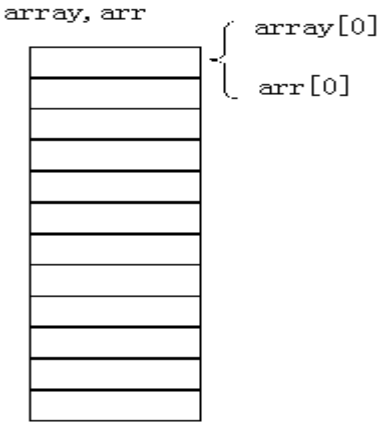
数组名可以作函数的实参和形参。如：

```
main()
{int array[10];
   ……
   ……
```

```
    f(array,10);
        ……
        ……
}

f(int arr[],int n);
{
        ……
        ……
}
```

array 为实参数组名，arr 为形参数组名。在学习指针变量之后就更容易理解这个问题了。数组名就是数组的首地址，实参向形参传送数组名实际上就是传送数组的地址，形参得到该地址后也指向同一数组。这就好象同一件物品有两个彼此不同的名称一样。



同样，指针变量的值也是地址，数组指针变量的值即为数组的首地址，当然也可作为函数的参数使用。

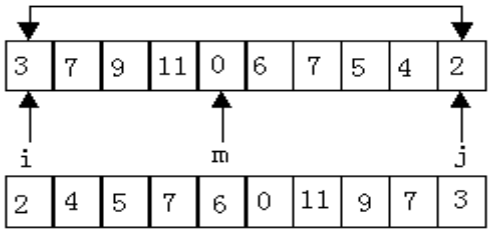
【例 10.15】

```
float aver(float *pa);
main(){
  float sco[5],av,*sp;
  int i;
  sp=sco;
  printf("\ninput 5 scores:\n");
  for(i=0;i<5;i++) scanf("%f",&sco[i]);
  av=aver(sp);
  printf("average score is %5.2f",av);
}
float aver(float *pa)
{
  int i;
  float av,s=0;
  for(i=0;i<5;i++) s=s+*pa++;
  av=s/5;
  return av;
```

}

【例 10.16】将数组 a 中的 n 个整数按相反顺序存放。

算法为：将 a[0]与 a[n-1]对换，再 a[1]与 a[n-2] 对换……,直到将 a[(n-1/2)]与 a[n-int((n-1)/2)]对换。今用循环处理此问题，设两个"位置指示变量"i 和 j，i 的初值为 0,j 的初值为 n-1。将 a[i]与 a[j]交换，然后使 i 的值加 1，j 的值减 1，再将 a[i]与 a[j]交换，直到 i=(n-1)/2 为止，如图所示。



程序如下：

```
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;
}
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 benn inverted:\n");
 for(i=0;i<10;i++)
  printf("%d,",a[i]);
 printf("\n");
}
```

对此程序可以作一些改动。将函数 inv 中的形参 x 改成指针变量。

【例 10.17】对例 10.16 可以作一些改动。将函数 inv 中的形参 x 改成指针变量。

程序如下：

```
void inv(int *x,int n)   /*形参x为指针变量*/
{
 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;}
 return;
}
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 benn inverted:\n");
 for(i=0;i<10;i++)
  printf("%d,",a[i]);
 printf("\n");
}
```

运行情况与前一程序相同。

【例 10.18】从 0 个数中找出其中最大值和最小值。

调用一个函数只能得到一个返回值，今用全局变量在函数之间“传递”数据。程序如下：

```
int max,min;       /*全局变量*/
void max_min_value(int array[],int n)
{int *p,*array_end;
 array_end=array+n;
 max=min=*array;
 for(p=array+1;p<array_end;p++)
   if(*p>max)max=*p;
   else if (*p<min)min=*p;
 return;
}
main()
{int i,number[10];
 printf("enter 10 integer umbers:\n");
 for(i=0;i<10;i++)
   scanf("%d",&number[i]);
 max_min_value(number,10);
 printf("\nmax=%d,min=%d\n",max,min);
 }
```

说明：

1)　在函数 max_min_value 中求出的最大值和最小值放在 max 和 min 中。由于它们是全局，

因此在主函数中可以直接使用。

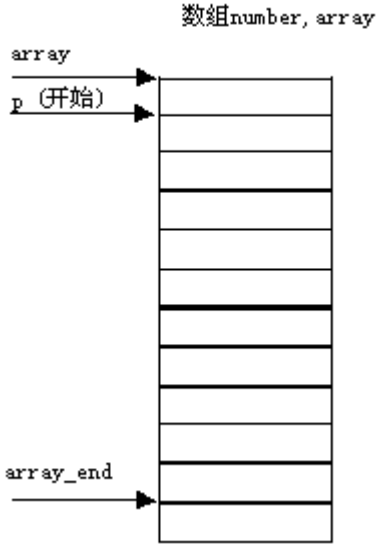
2) 函数 max_min_value 中的语句：

max=min=*array;

array 是数组名，它接收从实参传来的数组 numuber 的首地址。

array 相当于（&array[0]）。上述语句与 max=min=array[0];等价。

3) 在执行 for 循环时，p 的初值为 array+1，也就是使 p 指向 array[1]。以后每次执行 p++，使 p 指向下一个元素。每次将*p 和 max 与 min 比较。将大者放入 max，小者放 min。



4) 函数 max_min_value 的形参 array 可以改为指针变量类型。实参也可以不用数组名，而用指针变量传递地址。

【例 10.19】程序可改为：

```
int max,min;       /*全局变量*/
void max_min_value(int *array,int n)
{int *p,*array_end;
 array_end=array+n;
 max=min=*array;
 for(p=array+1;p<array_end;p++)
   if(*p>max)max=*p;
   else if (*p<min)min=*p;
 return;
}
main()
{int i,number[10],*p;
 p=number;              /*使p指向number数组*/

 printf("enter 10 integer umbers:\n");

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

```
 printf("\nmax=%d,min=%d\n",max,min);
 }
```

归纳起来，如果有一个实参数组，想在函数中改变此数组的元素的值，实参与形参的对应关系有以下 4 种：

1)　形参和实参都是数组名。

```
main()                                  f(int x[],int n)
{int a[10];                             {
  ……                                       ……
 f(a,10)                                }
  ……
}
```

a 和 x 指的是同一组数组。

2)　实用数组，形参用指针变量。

```
main()                                  f(int *x,int n)
{int a[10];                             {
  ……                                       ……
 f(a,10)                                }
  ……
}
```

3)　实参、型参都用指针变量。

4)　实参为指针变量，型参为数组名。

【例 10.20】用实参指针变量改写将 n 个整数按相反顺序存放。

```
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;
}
main()
{int i,arr[10]={3,7,9,11,0,6,7,5,4,2},*p;
 p=arr;
 printf("The original array:\n");
 for(i=0;i<10;i++,p++)
   printf("%d,",*p);
 printf("\n");
 p=arr;
 inv(p,10);
 printf("The array has benn inverted:\n");
 for(p=arr;p<arr+10;p++)
  printf("%d,",*p);
```

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

注意：main 函数中的指针变量 p 是有确定值的。即如果用指针变作实参，必须现使指针变量有确定值，指向一个已定义的数组。

【例 10.21】用选择法对 10 个整数排序。

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

说明：函数 sort 用数组名作为形参，也可改为用指针变量，这时函数的首部可以改为：sort(int *x,int n) 其他可一律不改。

1.1.1 指向多维数组的指针和指针变量

本小节以二维数组为例介绍多维数组的指针变量。

1. 多维数组的地址

   设有整型二维数组 a[3][4]如下：

   0　1　2　3

   4　5　6　7

   8　9　10　11

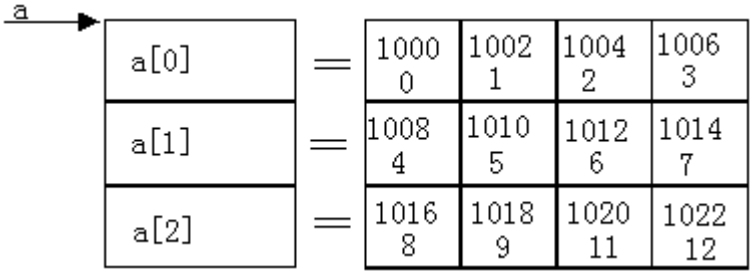
   它的定义为：

   int a[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}}

设数组 a 的首地址为 1000，各下标变量的首地址及其值如图所示。

1000 0	1002 1	1004 2	1006 3
1008 4	1010 5	1012 6	1014 7
1016 8	1018 9	1020 11	1022 12

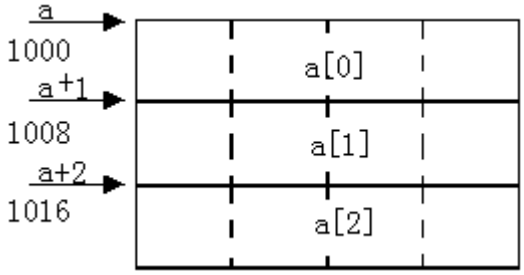
前面介绍过，C 语言允许把一个二维数组分解为多个一维数组来处理。因此数组 a 可分解为三个一维数组，即 a[0]，a[1]，a[2]。每一个一维数组又含有四个元素。



例如 a[0]数组，含有 a[0][0]，a[0][1]，a[0][2]，a[0][3]四个元素。

数组及数组元素的地址表示如下：

从二维数组的角度来看，a 是二维数组名，a 代表整个二维数组的首地址，也是二维数组 0 行的首地址，等于 1000。a+1 代表第一行的首地址，等于 1008。如图：



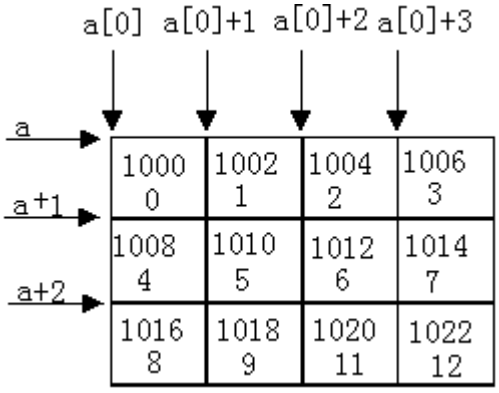
a[0]是第一个一维数组的数组名和首地址，因此也为 1000。*(a+0)或*a 是与 a[0]等效的， 它表示一维数组 a[0]0 号元素的首地址，也为 1000。&a[0][0]是二维数组 a 的 0 行 0 列元素首地址，同样是 1000。因此，a，a[0]，*(a+0)，*a，&a[0][0]是相等的。

同理，a+1 是二维数组 1 行的首地址，等于 1008。a[1]是第二个一维数组的数组名和首地址，因此也为 1008。&a[1][0]是二维数组 a 的 1 行 0 列元素地址，也是 1008。因此 a+1,a[1],*(a+1),&a[1][0]是等同的。

由此可得出：a+i，a[i]，*(a+i)，&a[i][0]是等同的。

此外，&a[i]和 a[i]也是等同的。因为在二维数组中不能把&a[i]理解为元素 a[i]的地址，不存在元素 a[i]。C 语言规定，它是一种地址计算方法，表示数组 a 第 i 行首地址。由此，我们得出：a[i]，&a[i]，*(a+i)和 a+i 也都是等同的。

另外，a[0]也可以看成是 a[0]+0，是一维数组 a[0]的 0 号元素的首地址，而 a[0]+1 则是 a[0]的 1 号元素首地址，由此可得出 a[i]+j 则是一维数组 a[i]的 j 号元素首地址，它等于&a[i][j]。



由 a[i]=*(a+i)得 a[i]+j=*(a+i)+j。由于*(a+i)+j 是二维数组 a 的 i 行 j 列元素的首地址，所以，该元素的值等于*(*(a+i)+j)。

【例 10.22】

```
main(){
    int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
    printf("%d,",a);
    printf("%d,",*a);
    printf("%d,",a[0]);
    printf("%d,",&a[0]);
    printf("%d\n",&a[0][0]);
    printf("%d,",a+1);
    printf("%d,",*(a+1));
    printf("%d,",a[1]);
    printf("%d,",&a[1]);
    printf("%d\n",&a[1][0]);
    printf("%d,",a+2);
    printf("%d,",*(a+2));
    printf("%d,",a[2]);
    printf("%d,",&a[2]);
    printf("%d\n",&a[2][0]);
    printf("%d,",a[1]+1);
    printf("%d\n",*(a+1)+1);
    printf("%d,%d\n",*(a[1]+1),*(*(a+1)+1));
}
```

2.　指向多维数组的指针变量

把二维数组 a 分解为一维数组 a[0],a[1],a[2]之后，设 p 为指向二维数组的指针变量。可定义为：

```
int (*p)[4]
```

它表示 p 是一个指针变量，它指向包含 4 个元素的一维数组。若指向第一个一维数组 a[0]，其值等于 a,a[0]，或&a[0][0]等。而 p+i 则指向一维数组 a[i]。从前面的分析可得出*(p+i)+j 是二维数组 i 行 j 列的元素的地址，而*(*(p+i)+j)则是 i 行 j 列元素的值。二维数组指针变量说明的一般形式为：

类型说明符　(*指针变量名)[长度]

其中“类型说明符”为所指数组的数据类型。“*”表示其后的变量是指针类型。“长度”表示二维数组分解为多个一维数组时，一维数组的长度，也就是二维数组的列数。应注意“(*指针变量名)”两边的括号不可少，如缺少括号则表示是指针数组(本章后面介绍)，意义就完全不同了。

【例 10.23】

```
main(){
    int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
    int(*p)[4];
    int i,j;
    p=a;
    for(i=0;i<3;i++)
    {for(j=0;j<4;j++) printf("%2d  ",*(*(p+i)+j));
    printf("\n");}
}
```

1.1 字符串的指针指向字符串的针指变量

1.1.1 字符串的表示形式

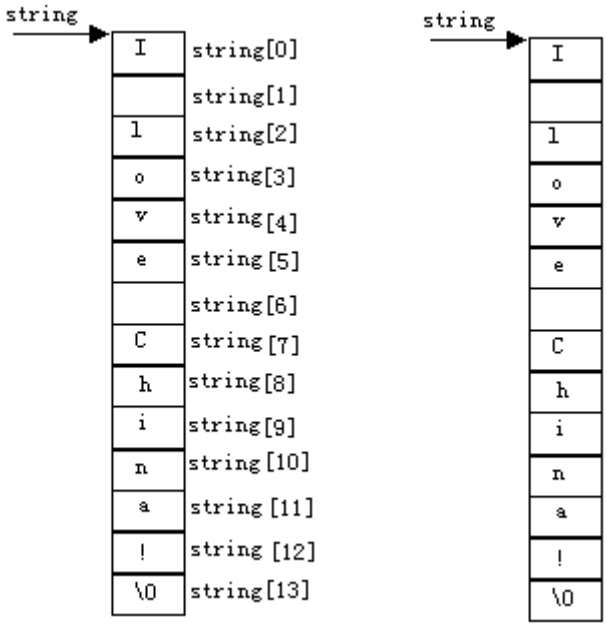
在 C 语言中，可以用两种方法访问一个字符串。

1)　用字符数组存放一个字符串，然后输出该字符串。

【例 10.24】

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

说明：和前面介绍的数组属性一样，string 是数组名，它代表字符数组的首地址。



2)　用字符串指针指向一个字符串。

【例 10.25】

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

字符串指针变量的定义说明与指向字符变量的指针变量说明是相同的。只能按对指针变量的赋值不同来区别。对指向字符变量的指针变量应赋予该字符变量的地址。

如：

```
char c,*p=&c;
```

表示 p 是一个指向字符变量 c 的指针变量。

而：

```
char *s="C Language";
```

则表示 s 是一个指向字符串的指针变量。把字符串的首地址赋予 s。

上例中，首先定义 string 是一个字符指针变量，然后把字符串的首地址赋予 string(应写出整个字符串，以便编译系统把该串装入连续的一块内存单元)，并把首地址送入 string。程序中的：

```
char *ps="C Language";
```

等效于：

```
char *ps;
ps="C Language";
```

【例 10.26】输出字符串中 n 个字符后的所有字符。

```
main(){
  char *ps="this is a book";
  int n=10;
  ps=ps+n;
```

```
  printf("%s\n",ps);
}
```

运行结果为：

book

在程序中对 ps 初始化时，即把字符串首地址赋予 ps，当 ps= ps+10 之后，ps 指向字符"b"，因此输出为"book"。

【例 10.27】在输入的字符串中查找有无'k'字符。

```
main(){
  char st[20],*ps;
  int i;
  printf("input a string:\n");
  ps=st;
  scanf("%s",ps);
  for(i=0;ps[i]!='\0';i++)
    if(ps[i]=='k'){
       printf("there is a 'k' in the string\n");
       break;
    }
  if(ps[i]=='\0') printf("There is no 'k' in the string\n");
}
```

【例 10.28】本例是将指针变量指向一个格式字符串，用在 printf 函数中，用于输出二维数组的各种地址表示的值。但在 printf 语句中用指针变量 PF 代替了格式串。 这也是程序中常用的方法。

```
main(){
  static int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
  char *PF;
  PF="%d,%d,%d,%d,%d\n";
  printf(PF,a,*a,a[0],&a[0],&a[0][0]);
  printf(PF,a+1,*(a+1),a[1],&a[1],&a[1][0]);
  printf(PF,a+2,*(a+2),a[2],&a[2],&a[2][0]);
  printf("%d,%d\n",a[1]+1,*(a+1)+1);
  printf("%d,%d\n",*(a[1]+1),*(*(a+1)+1));
}
```

【例 10.29】本例是把字符串指针作为函数参数的使用。要求把一个字符串的内容复制到另一个字符串中，并且不能使用 strcpy 函数。函数 cprstr 的形参为两个字符指针变量。pss 指向源字符串，pds 指向目标字符串。注意表达式：(*pds=*pss)!=`\0'的用法。

```
cpystr(char *pss,char *pds){
  while((*pds=*pss)!='\0'){
```

```
      pds++;
      pss++; }
 }
main(){
  char *pa="CHINA",b[10],*pb;
  pb=b;
  cpystr(pa,pb);
  printf("string a=%s\nstring b=%s\n",pa,pb);
}
```

在本例中，程序完成了两项工作：一是把 pss 指向的源字符串复制到 pds 所指向的目标字符串中，二是判断所复制的字符是否为`\0'，若是则表明源字符串结束，不再循环。否则，pds 和 pss 都加 1，指向下一字符。在主函数中，以指针变量 pa,pb 为实参，分别取得确定值后调用 cprstr 函数。由于采用的指针变量 pa 和 pss,pb 和 pds 均指向同一字符串，因此在主函数和 cprstr 函数中均可使用这些字符串。也可以把 cprstr 函数简化为以下形式：

```
    cprstr(char *pss,char*pds)
      {while ((*pds++=*pss++)!=`\0');}
```

即把指针的移动和赋值合并在一个语句中。 进一步分析还可发现`\0'的 ASCⅡ码为 0，对于 while 语句只看表达式的值为非 0 就循环，为 0 则结束循环，因此也可省去"!=`\0'"这一判断部分，而写为以下形式：

```
      cprstr (char *pss,char *pds)
             {while (*pdss++=*pss++);}
```

表达式的意义可解释为，源字符向目标字符赋值，移动指针，若所赋值为非 0 则循环，否则结束循环。这样使程序更加简洁。

【例 10.30】简化后的程序如下所示。

```
cpystr(char *pss,char *pds){
    while(*pds++=*pss++);
}
main(){
  char *pa="CHINA",b[10],*pb;
  pb=b;
  cpystr(pa,pb);
  printf("string a=%s\nstring b=%s\n",pa,pb);
}
```

1.1.1 使用字符串指针变量与字符数组的区别

用字符数组和字符指针变量都可实现字符串的存储和运算。但是两者是有区别的。在使用时应注意以下几个问题：

1. 字符串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

由于若干个数组元素组成的，它可用来存放整个字符串。

2. 对字符串指针方式

char *ps="C Language";

可以写为：

```
char *ps;
ps="C Language";
```

而对数组方式：

```
static char st[]={"C Language"};
```

不能写为：

```
char st[20];
st={"C Language"};
```

而只能对字符数组的各元素逐个赋值。

从以上几点可以看出字符串指针变量与字符数组在使用时的区别，同时也可看出使用指针变量更加方便。

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 *ps="C Langage";
```

或者

```
char *ps;
ps="C Language";
```

都是合法的。

1.1 函数指针变量

在C语言中，一个函数总是占用一段连续的内存区，而函数名就是该函数所占内存区的首地址。我们可以把函数的这个首地址(或称入口地址)赋予一个指针变量，使该指针变量指向该函数。然后通过指针变量就可以找到并调用这个函数。我们把这种指向函数的指针变量称为"函数指针变量"。

函数指针变量定义的一般形式为：

类型说明符　(*指针变量名)();

其中“类型说明符”表示被指函数的返回值的类型。“(* 指针变量名)"表示"*"后面的变量是定义的指针变量。最后的空括号表示指针变量所指的是一个函数。

例如：

```
int (*pf)();
```

表示 pf 是一个指向函数入口的指针变量，该函数的返回值(函数值)是整型。

【例 10.31】本例用来说明用指针形式实现对函数调用的方法。

```
int max(int a,int b){
  if(a>b)return a;
  else return b;
}
main(){
  int max(int a,int b);
  int(*pmax)();
```

```
  int x,y,z;
  pmax=max;
  printf("input two numbers:\n");
  scanf("%d%d",&x,&y);
  z=(*pmax)(x,y);
  printf("maxmum=%d",z);
}
```

从上述程序可以看出用，函数指针变量形式调用函数的步骤如下：

1) 先定义函数指针变量，如后一程序中第 9 行 int (*pmax)();定义 pmax 为函数指针变量。
2) 把被调函数的入口地址(函数名)赋予该函数指针变量，如程序中第 11 行 pmax=max;
3) 用函数指针变量形式调用函数，如程序第 14 行 z=(*pmax)(x,y);
4) 调用函数的一般形式为：

(*指针变量名)（实参表)

使用函数指针变量还应注意以下两点：

a) 函数指针变量不能进行算术运算，这是与数组指针变量不同的。数组指针变量加减一个整数可使指针移动指向后面或前面的数组元素，而函数指针的移动是毫无意义的。
b) 函数调用中"(*指针变量名)"的两边的括号不可少，其中的*不应该理解为求值运算，在此处它只是一种表示符号。

1.1 指针型函数

前面我们介绍过，所谓函数类型是指函数返回值的类型。在C语言中允许一个函数的返回值是一个指针(即地址)，这种返回指针值的函数称为指针型函数。

定义指针型函数的一般形式为：

```
类型说明符 *函数名(形参表)
{
    ……          /*函数体*/
}
```

其中函数名之前加了“*”号表明这是一个指针型函数，即返回值是一个指针。类型说明符表示了返回的指针值所指向的数据类型。

如：

```
int *ap(int x,int y)
{
  ……        /*函数体*/
}
```

表示 ap 是一个返回指针值的指针型函数，它返回的指针指向一个整型变量。

【例 10.32】本程序是通过指针函数，输入一个 1～7 之间的整数，输出对应的星期名。

```
main(){
  int i;
  char *day_name(int n);
  printf("input Day No:\n");
```

```
  scanf("%d",&i);
  if(i<0) exit(1);
  printf("Day No:%2d-->%s\n",i,day_name(i));
}
char *day_name(int n){
  static char *name[]={ "Illegal day",
                        "Monday",
                        "Tuesday",
                        "Wednesday",
                        "Thursday",
                        "Friday",
                        "Saturday",
                        "Sunday"};
  return((n<1||n>7) ? name[0] : name[n]);
}
```

本例中定义了一个指针型函数 day_name，它的返回值指向一个字符串。该函数中定义了一个静态指针数组 name。name 数组初始化赋值为八个字符串，分别表示各个星期名及出错提示。形参 n 表示与星期名所对应的整数。在主函数中，把输入的整数 i 作为实参，在 printf 语句中调用 day_name 函数并把 i 值传送给形参 n。day_name 函数中的 return 语句包含一个条件表达式，n 值若大于 7 或小于 1 则把 name[0]指针返回主函数输出出错提示字符串"Illegal day"。否则返回主函数输出对应的星期名。主函数中的第 7 行是个条件语句，其语义是，如输入为负数(i<0)则中止程序运行退出程序。exit 是一个库函数，exit(1)表示发生错误后退出程序，exit(0)表示正常退出。

应该特别注意的是函数指针变量和指针型函数这两者在写法和意义上的区别。如 int(*p)()和 int *p()是两个完全不同的量。

int (*p)()是一个变量说明，说明 p 是一个指向函数入口的指针变量，该函数的返回值是整型量，(*p)的两边的括号不能少。

int *p()则不是变量说明而是函数说明，说明 p 是一个指针型函数，其返回值是一个指向整型量的指针，*p 两边没有括号。作为函数说明，在括号内最好写入形式参数，这样便于与变量说明区别。

对于指针型函数定义，int *p()只是函数头部分，一般还应该有函数体部分。

1.1 指针数组和指向指针的指针

1.1.1 指针数组的概念

一个数组的元素值为指针则是指针数组。 指针数组是一组有序的指针的集合。 指针数组的所有元素都必须是具有相同存储类型和指向相同数据类型的指针变量。

指针数组说明的一般形式为：

类型说明符 *数组名[数组长度]

其中类型说明符为指针值所指向的变量的类型。

例如：

```
int *pa[3]
```

表示 pa 是一个指针数组，它有三个数组元素，每个元素值都是一个指针，指向整型变量。

【例 10.33】通常可用一个指针数组来指向一个二维数组。指针数组中的每个元素被赋予二维数组每一行的首地址，因此也可理解为指向一个一维数组。

```
main(){
  int a[3][3]={1,2,3,4,5,6,7,8,9};
  int *pa[3]={a[0],a[1],a[2]};
  int *p=a[0];
  int i;
  for(i=0;i<3;i++)
      printf("%d,%d,%d\n",a[i][2-i],*a[i],*(*(a+i)+i));
  for(i=0;i<3;i++)
      printf("%d,%d,%d\n",*pa[i],p[i],*(p+i));
}
```

本例程序中，pa 是一个指针数组，三个元素分别指向二维数组 a 的各行。然后用循环语句输出指定的数组元素。其中*a[i]表示 i 行 0 列元素值；*(*(a+i)+i)表示 i 行 i 列的元素值；*pa[i]表示 i 行 0 列元素值；由于 p 与 a[0]相同，故 p[i]表示 0 行 i 列的值；*(p+i)表示 0 行 i 列的值。读者可仔细领会元素值的各种不同的表示方法。

应该注意指针数组和二维数组指针变量的区别。这两者虽然都可用来表示二维数组，但是其表示方法和意义是不同的。

二维数组指针变量是单个的变量，其一般形式中"(*指针变量名)"两边的括号不可少。而指针数组类型表示的是多个指针(一组有序指针)在一般形式中"*指针数组名"两边不能有括号。

例如：

```
int (*p)[3];
```

表示一个指向二维数组的指针变量。该二维数组的列数为 3 或分解为一维数组的长度为 3。

```
int *p[3]
```

表示 p 是一个指针数组，有三个下标变量 p[0]，p[1]，p[2]均为指针变量。

指针数组也常用来表示一组字符串，这时指针数组的每个元素被赋予一个字符串的首地址。指向字符串的指针数组的初始化更为简单。例如在例 10.32 中即采用指针数组来表示一组字符串。其初始化赋值为：

```
char *name[]={"Illagal day",
              "Monday",
              "Tuesday",
              "Wednesday",
              "Thursday",
              "Friday",
              "Saturday",
              "Sunday"};
```

完成这个初始化赋值之后，name[0]即指向字符串"Illegal day"，name[1]指向

"Monday"......。

指针数组也可以用作函数参数。

【例 10.34】指针数组作指针型函数的参数。在本例主函数中，定义了一个指针数组 name，并对 name 作了初始化赋值。其每个元素都指向一个字符串。然后又以 name 作为实参调用指针型函数 day_name，在调用时把数组名 name 赋予形参变量 name，输入的整数 i 作为第二个实参赋予形参 n。在 day_ name 函数中定义了两个指针变量 pp1 和 pp2，pp1 被赋予 name[0]的值(即*name)，pp2 被赋予 name[n]的值即*(name+ n)。由条件表达式决定返回 pp1 或 pp2 指针给主函数中的指针变量 ps。最后输出 i 和 ps 的值。

```
main(){
  static char *name[]={ "Illegal day",
                        "Monday",
                        "Tuesday",
                        "Wednesday",
                        "Thursday",
                        "Friday",
                        "Saturday",
                        "Sunday"};
  char *ps;
  int i;
  char *day_name(char *name[],int n);
  printf("input Day No:\n");
  scanf("%d",&i);
  if(i<0) exit(1);
  ps=day_name(name,i);
  printf("Day No:%2d-->%s\n",i,ps);
}
char *day_name(char *name[],int n)
{
  char *pp1,*pp2;
  pp1=*name;
  pp2=*(name+n);
  return((n<1||n>7)? pp1:pp2);
}
```

【例 10.35】输入 5 个国名并按字母顺序排列后输出。现编程如下：

```
#include"string.h"
main(){
  void sort(char *name[],int n);
  void print(char *name[],int n);
  static char *name[]={ "CHINA","AMERICA","AUSTRALIA",
                        "FRANCE","GERMAN"};
  int n=5;
  sort(name,n);
```

```
  print(name,n);
}
void sort(char *name[],int n){
  char *pt;
  int i,j,k;
  for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++)
          if(strcmp(name[k],name[j])>0) k=j;
      if(k!=i){
          pt=name[i];
          name[i]=name[k];
          name[k]=pt;
      }
  }
}
void print(char *name[],int n){
  int i;
  for (i=0;i<n;i++) printf("%s\n",name[i]);
}
```

说明：

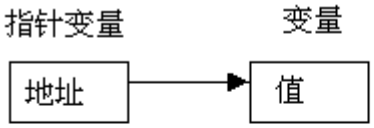
在以前的例子中采用了普通的排序方法，逐个比较之后交换字符串的位置。交换字符串的物理位置是通过字符串复制函数完成的。反复的交换将使程序执行的速度很慢，同时由于各字符串(国名)的长度不同，又增加了存储管理的负担。用指针数组能很好地解决这些问题。把所有的字符串存放在一个数组中，把这些字符数组的首地址放在一个指针数组中，当需要交换两个字符串时，只须交换指针数组相应两元素的内容(地址)即可，而不必交换字符串本身。

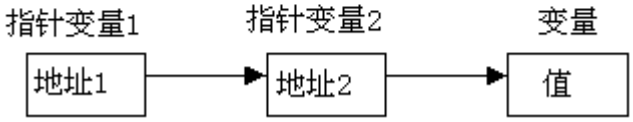
本程序定义了两个函数，一个名为 sort 完成排序，其形参为指针数组 name，即为待排序的各字符串数组的指针。形参 n 为字符串的个数。另一个函数名为 print，用于排序后字符串的输出，其形参与 sort 的形参相同。主函数 main 中，定义了指针数组 name 并作了初始化赋值。然后分别调用 sort 函数和 print 函数完成排序和输出。值得说明的是在 sort 函数中，对两个字符串比较，采用了 strcmp 函数，strcmp 函数允许参与比较的字符串以指针方式出现。name[k]和 name[j]均为指针，因此是合法的。字符串比较后需要交换时，只交换指针数组元素的值，而不交换具体的字符串，这样将大大减少时间的开销，提高了运行效率。

1.1.1 指向指针的指针

如果一个指针变量存放的又是另一个指针变量的地址，则称这个指针变量为指向指针的指针变量。

在前面已经介绍过，通过指针访问变量称为间接访问。由于指针变量直接指向变量，所以称为“单级间址”。而如果通过指向指针的指针变量来访问变量则构成“二级间址”。





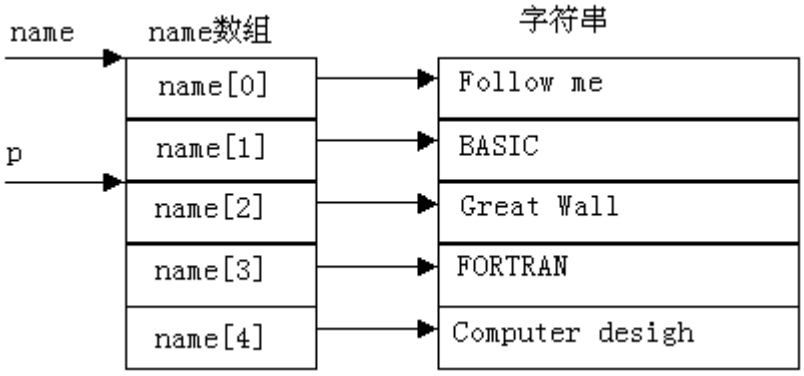
从下图可以看到，name 是一个指针数组，它的每一个元素是一个指针型数据，其值为地址。Name 是一个数据，它的每一个元素都有相应的地址。数组名 name 代表该指针数组的首地址。name+1 是 mane[i]的地址。name+1 就是指向指针型数据的指针（地址）。还可以设置一个指针变量 p，使它指向指针数组元素。P 就是指向指针型数据的指针变量。

怎样定义一个指向指针型数据的指针变量呢？如下：

```
char **p;
```

p 前面有两个*号,相当于*(*p)。显然*p 是指针变量的定义形式，如果没有最前面的*，那就是定义了一个指向字符数据的指针变量。现在它前面又有一个*号，表示指针变量 p 是指向一个字符指针型变量的。*p 就是 p 所指向的另一个指针变量。

从下图可以看到，name 是一个指针数组，它的每一个元素是一个指针型数据，其值为地址。name 是一个数组，它的每一个元素都有相应的地址。数组名 name 代表该指针数组的首地址。name+1 是 mane[i]的地址。name+1 就是指向指针型数据的指针（地址）。还可以设置一个指针变量 p，使它指向指针数组元素。P 就是指向指针型数据的指针变量。



如果有：

```
p=name+2;
printf("%o\n",*p);
printf("%s\n",*p);
```

则，第一个 printf 函数语句输出 name[2]的值（它是一个地址），第二个 printf 函数语句以字符串形式（%s）输出字符串"Great Wall"。

【例 10.36】使用指向指针的指针。

```
main()
{char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer desighn"};
 char **p;
 int i;
 for(i=0;i<5;i++)
   {p=name+i;
```

```
    printf("%s\n",*p);
   }
}
```

说明：

p 是指向指针的指针变量。

【例 10.37】一个指针数组的元素指向数据的简单例子。

```
main()
{static int a[5]={1,3,5,7,9};
 int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
 int **p,i;
 p=num;
 for(i=0;i<5;i++)
   {printf("%d\t",**p);p++;}
}
```

说明：

指针数组的元素只能存放地址。

1.1.1 main 函数的参数

前面介绍的 main 函数都是不带参数的。因此 main 后的括号都是空括号。实际上，main 函数可以带参数，这个参数可以认为是 main 函数的形式参数。C 语言规定 main 函数的参数只能有两个，习惯上这两个参数写为 argc 和 argv。因此，main 函数的函数头可写为：

```
main (argc,argv)
```

C 语言还规定 argc(第一个形参)必须是整型变量,argv(第二个形参)必须是指向字符串的指针数组。加上形参说明后，main 函数的函数头应写为：

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

由于 main 函数不能被其它函数调用，因此不可能在程序内部取得实际值。那么，在何处把实参值赋予 main 函数的形参呢？实际上,main 函数的参数值是从操作系统命令行上获得的。当我们要运行一个可执行文件时，在 DOS 提示符下键入文件名，再输入实际参数即可把这些实参传送到 main 的形参中去。

DOS 提示符下命令行的一般形式为：

```
C:\>可执行文件名　参数　参数……;
```

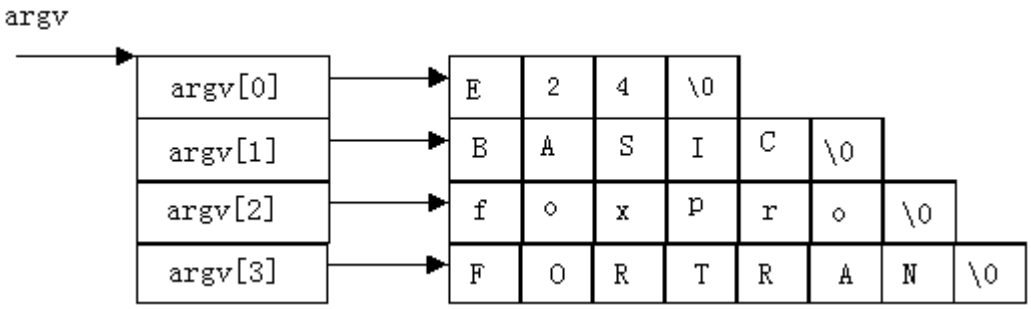
但是应该特别注意的是,main 的两个形参和命令行中的参数在位置上不是一一对应的。因为,main 的形参只有二个，而命令行中的参数个数原则上未加限制。argc 参数表示了命令行中参数的个数(注意：文件名本身也算一个参数)，argc 的值是在输入命令行时由系统按实际参数的个数自动赋予的。

例如有命令行为：

```
C:\>E24  BASIC  foxpro  FORTRAN
```

由于文件名 E24 本身也算一个参数，所以共有 4 个参数，因此 argc 取得的值为 4。argv 参数是字符串指针数组,其各元素值为命令行中各字符串(参数均按字符串处理)的首地址。指

针数组的长度即为参数个数。数组元素初值由系统自动赋予。其表示如图所示：



【例 10.38】

```
main(int argc,char *argv){
  while(argc-->1)
      printf("%s\n",*++argv);
}
```

本例是显示命令行中输入的参数。如果上例的可执行文件名为 e24.exe，存放在 A 驱动器的盘内。因此输入的命令行为：

C:\>a:e24 BASIC foxpro FORTRAN

则运行结果为：

BASIC
foxpro
FORTRAN

该行共有 4 个参数，执行 main 时，argc 的初值即为 4。argv 的 4 个元素分为 4 个字符串的首地址。执行 while 语句，每循环一次 argv 值减 1，当 argv 等于 1 时停止循环，共循环三次，因此共可输出三个参数。在 printf 函数中，由于打印项*++argv 是先加 1 再打印，故第一次打印的是 argv[1]所指的字符串 BASIC。第二、三次循环分别打印后二个字符串。而参数 e24 是文件名，不必输出。

1.1 有关指针的数据类型和指针运算的小结

1.1.1 有关指针的数据类型的小结

定义	含　义
int i;	定义整型变量 i
int *p	p 为指向整型数据的指针变量
int a[n];	定义整型数组 a，它有 n 个元素
int *p[n];	定义指针数组 p，它由 n 个指向整型数据的指针元素组成
int (*p)[n];	p 为指向含 n 个元素的一维数组的指针变量
int f();	f 为带回整型函数值的函数
int *p();	p 为带回一个指针的函数，该指针指向整型数据
int (*p)();	p 为指向函数的指针，该函数返回一个整型值
int **p;	P 是一个指针变量，它指向一个指向整型数据的指针变量

1.1.1 指针运算的小结

现把全部指针运算列出如下：

1) 指针变量加（减）一个整数：

例如：p++、p--、p+i、p-i、p+=i、p-=i

一个指针变量加（减）一个整数并不是简单地将原值加（减）一个整数，而是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内存单元字节数加（减）。

2) 指针变量赋值：将一个变量的地址赋给一个指针变量。

p=&a;　(将变量 a 的地址赋给 p)

p=array;　(将数组 array 的首地址赋给 p)

p=&array[i];　(将数组 array 第 i 个元素的地址赋给 p)

p=max;　(max 为已定义的函数，将 max 的入口地址赋给 p)

p1=p2;　(p1 和 p2 都是指针变量，将 p2 的值赋给 p1)

注意：不能如下：

p=1000;

3) 指针变量可以有空值，即该指针变量不指向任何变量：

p=NULL;

4) 两个指针变量可以相减：如果两个指针变量指向同一个数组的元素，则两个指针变量值之差是两个指针之间的元素个数。

5) 两个指针变量比较：如果两个指针变量指向同一个数组的元素，则两个指针变量可以进行比较。指向前面的元素的指针变量“小于” 指向后面的元素的指针变量。

1.1.1 void 指针类型

ANSI 新标准增加了一种“void”指针类型，即可以定义一个指针变量，但不指定它是指向哪一种类型数据。