C 语言教程之 指针

指针简介

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。

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， 在第二章中已有详细的介绍。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。 但在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。

指针变量的类型说明

对指针变量的类型说明包括三个内容：
(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：

staic int *p2; /*p2 是指向静态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。

指针变量的赋值

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。 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 变示变量 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

指针变量的运算

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。

```
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。本语句表示输出变量 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

```
main(){
int a=10,b=20,s,t,*pa,*pb;
pa=&a;
pb=&b;
s=*pa+*pb;
t=*pa**pb;
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);
}
```

.....
说明 pa,pb 为整型指针变量
给指针变量 pa 赋值，pa 指向变量 a。
给指针变量 pb 赋值，pb 指向变量 b。
本行的意义是求 a+b 之和，(*pa 就是 a，*pb 就是 b)。
本行是求 a*b 之积。
输出结果。
输出结果。
.....

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。

```
main(){
int a,b,c,*pmax,*pmin;
printf("input three numbers:\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b){
pmax=&a;
pmin=&b;}
else{
pmax=&b;
pmin=&a;}
if(c>*pmax) pmax=&c;
if(c<*pmin) pmin=&c;
printf("max=%d\nmin=%d\n",*pmax,*pmin);
}
```

.....

pmax,pmin 为整型指针变量。

输入提示。

输入三个数字。

如果第一个数字大于第二个数字. . .

指针变量赋值

指针变量赋值

指针变量赋值

指针变量赋值

判断并赋值

判断并赋值

输出结果

.....

数组指针变量的说明和使用

指向数组的指针变量称为数组指针变量。 在讨论数组指针变量的说明和使用之前，我们先明确几个关系。

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。 一个指针变量既可以指向一个数组，也可以指向一个数组元素， 可把数组名或第一个元素的地址赋予它。如要使指针变量指向第 i 号元素可以把 i 元素的首地址赋予它或把数组名加 i 赋予它。

设有实数组 a，指向 a 的指针变量为 pa，从图 6.3 中我们可以看出有以下关系：
pa,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。
pa+1,a+1,&a[1]均指向 1 号元素 a[1]。类推可知 a+i,a+i,&a[i]
指向 i 号元素 a[i]。应该说明的是 pa 是变量，而 a,&a[i]都是常量。在编程时应予以注意。

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

主函数
定义一个整型数组和一个整型变量
循环语句
给数组赋值
打印每一个数组的值
.....
输出换行
.....
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。
引入指针变量后，就可以用两种方法来访问数组元素了。
第一种方法为下标法，即用 a[i]形式访问数组元素。 在第四章中介绍数组时都是采用这种方法。
第二种方法为指针法，即采用*(pa+i)形式，用间接访问的方法来访问数组元素。

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

主函数
定义整型数组和指针
将指针 pa 指向数组 a

循环
将变量 i 的值赋给由指针 pa 指向的 a[]的数组单元
将指针 pa 指向 a[]的下一个单元
.....
指针 pa 重新取得数组 a 的首地址
循环
用数组方式输出数组 a 中的所有元素
将指针 pa 指向 a[]的下一个单元
.....
.....
下面，另举一例，该例与上例本意相同，但是实现方式不同。

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

主函数
定义整型数组和指针，并使指针指向数组 a
循环
将变量 i 的值赋给由指针 pa 指向的 a[]的数组单元
用指针输出数组 a 中的所有元素，同时指针 pa 指向 a[]的下一个单元
.....
.....

数组名和数组指针变量作函数参数

在第五章中曾经介绍过用数组名作函数的实参和形参的问题。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。

```
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;
}
```

指向多维数组的指针变量

本小节以二维数组为例介绍多维数组的指针变量。

一、多维数组地址的表示方法
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

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。在第四章中介绍过，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：a 是二维数组名，也是二维数组 0 行的首地址，等于 1000。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。该元素的值等于*(*(a+i)+j)。

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

```
}
```

二、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，它指向二维数组 a 或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。

```
[Explain]main(){
static 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));
}
```

'Expain 字符串指针变量的说明和使用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。

请看下面一例。

```
main(){
char *ps;
ps="C Language";
printf("%s",ps);
}
```

运行结果为：

C Language

上例中，首先定义 ps 是一个字符指针变量， 然后把字符串的首地址赋予 ps(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 ps。程序中的： char *ps;ps="C Language";等效于：char *ps="C Language";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"。

```
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");
}
```

本例是在输入的字符串中查找有无‘k’字符。 下面这个例子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));
}
```

在下例是讲解，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。表达式：

```
(*pds=*pss)!=`\0'
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);
}
```

在上例中，程序完成了两项工作：一是把 pss 指向的源字符复制到 pds 所指向的目标字符中，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 ASC II 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。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);
}
```

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. 对字符数组作初始化赋值，必须采用外部类型或静态类型，如： static char st[]={“C Language”};而对字符串指针变量则无此限制，如： char *ps="C Language";

3. 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。

函数指针变量

在C语言中规定，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。
下面通过例子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); 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。

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。下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。

```
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。指针数组的说明与使用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。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。图 6—6 表示了这种关系。

```
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];
main(){
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 6.20 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?quot;Monday".....。

指针数组也可以用作函数参数。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。

指针数组作指针型函数的参数

```
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);
}
```

下例要求输入 5 个国名并按字母顺序排列后输出。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。程序中定义了两个函数，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。
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]);
}
```

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 (argc,argv)
int argv;
char *argv[];或写成：
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:\>E6 24 BASIC dbase FORTRAN 由于文件名 E6 24 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 6.8 所示：

```
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 dBASE FORTRAN
则运行结果为：

```
BASIC
dBASE
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。

下例的命令行中有两个参数，第二个参数 20 即为输入的 n 值。在程序中*++argv 的值

为字符串“20”，然后用函数"atoi"把它换为整型作为 while 语句中的循环控制变量，输出 20 个偶数。

```
#include"stdlib.h"
main(int argc,char*argv[]){
int a=0,n;
n=atoi(*++argv);
while(n--) printf("%d ",a++*2);
}
```

本程序是从 0 开始输出 n 个偶数。指向指针的指针变量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， 简称间访。由于指针变量直接指向变量，所以称为单级间访。 而如果通过指向指针的指针变量来访问变量则构成了二级或多级间访。在 C 语言程序中，对间访的级数并未明确限制， 但是间访级数太多时不容易理解解，也容易出错，因此，一般很少超过二级间访。 指向指针的指针变量说明的一般形式为：

类型说明符** 指针变量名；

例如： int ** pp; 表示 pp 是一个指针变量，它指向另一个指针变量， 而这个指针变量指向一个整型量。下面举一个例子来说明这种关系。

```
main(){
int x,*p,**pp;
x=10;
p=&x;
pp=&p;
printf("x=%d\n",**pp);
}
```

上例程序中 p 是一个指针变量，指向整型量 x；pp 也是一个指针变量， 它指向指针变量 p。通过 pp 变量访问 x 的写法是**pp。程序最后输出 x 的值为 10。通过上例，读者可以学习指向指针的指针变量的说明和使用方法。

下述程序中首先定义说明了指针数组 ps 并作了初始化赋值。 又说明了 pps 是一个指向指针的指针变量。在 5 次循环中， pps 分别取得了 ps[0]，ps[1]，ps[2]，ps[3]，ps[4]的地址值(如图 6.10 所示)。再通过这些地址即可找到该字符串。

```
main(){
static char *ps[]={ "BASIC","DBASE","C","FORTRAN",
"PASCAL"};
char **pps;
int i;
for(i=0;i<5;i++){
pps=ps+i;
printf("%s\n",*pps);
}
}
```

本程序是用指向指针的指针变量编程，输出多个字符串。

小结

1. 指针是C语言中一个重要的组成部分，使用指针编程有以下优点：
(1)提高程序的编译效率和执行速度。
(2)通过指针可使用主调函数和被调函数之间共享变量或数据结构，便于实现双向数据通讯。
(3)可以实现动态的存储分配。
(4)便于表示各种数据结构，编写高质量的程序。

2. 指针的运算
(1)取地址运算符&：求变量的地址
(2)取内容运算符*：表示指针所指的变量
(3)赋值运算
·把变量地址赋予指针变量
·同类型指针变量相互赋值
·把数组，字符串的首地址赋予指针变量
·把函数入口地址赋予指针变量
(4)加减运算
对指向数组，字符串的指针变量可以进行加减运算，如 p+n,p-n,p++,p--等。对指向同一数组的两个指针变量可以相减。对指向其它类型的指针变量作加减运算是无意义的。
(5)关系运算
指向同一数组的两个指针变量之间可以进行大于、小于、 等于比较运算。指针可与 0 比较，p==0 表示 p 为空指针。

3. 与指针有关的各种说明和意义见下表。

int *p;	p 为指向整型量的指针变量
int *p[n];	p 为指针数组，由 n 个指向整型量的指针元素组成。
int (*p)[n];	p 为指向整型二维数组的指针变量，二维数组的列数为 n
int *p()	p 为返回指针值的函数，该指针指向整型量
int (*p)()	p 为指向函数的指针，该函数返回整型量
int **p	p 为一个指向另一指针的指针变量，该指针指向一个整型量。

4. 有关指针的说明很多是由指针，数组，函数说明组合而成的。
但并不是可以任意组合，例如数组不能由函数组成，即数组元素不能是一个函数；函数也不能返回一个数组或返回另一个函数。例如
int a[5]()；就是错误的。

5. 关于括号
在解释组合说明符时， 标识符右边的方括号和圆括号优先于标识符左边的“*”号，而方括号和圆括号以相同的优先级从左到右结合。但可以用圆括号改变约定的结合顺序。

6. 阅读组合说明符的规则是“从里向外”。
从标识符开始，先看它右边有无方括号或园括号，如有则先作出解释，再看左边有无*号。 如果在任何时候遇到了闭括号，则在继续之前必须用相同的规则处理括号内的内容。例如：
int*(*(*a())[10]

↑ ↑↑↑↑↑↑
7 6 4 2 1 3 5
上面给出了由内向外的阅读顺序，下面来解释它：
(1)标识符 a 被说明为；
(2)一个指针变量，它指向；
(3)一个函数，它返回；
(4)一个指针，该指针指向；
(5)一个有 10 个元素的数组，其类型为；
(6)指针型，它指向；
(7)int 型数据。
因此 a 是一个函数指针变量，该函数返回的一个指针值又指向一个指针数组，该指针数组的元素指向整型量。