第 6 章　指针

指针是 C 语言中的一个重要概念，是其最具特色的语言部分，也是 C 语言的精华，同时也是 C 语言的难点。指针可以有效地表示复杂的数据结构；能动态分配内存；方便地使用字符串；有效而方便地使用数组；在调用函数时能获得 1 个以上的结果；能直接处理内存单元的地址等。

指针的概念比较复杂，使用也比较灵活，因此初学时常会出错。希望读者在学习本章内容时，多思考、多比较、多练习，在实践应用中逐步掌握它。

6.1　地址和指针

6.1.1　地址和指针的概念

1. 内存地址——内存中存储单元的编号

内存是计算机的一个重要组成部分，用于存放那些“正在”使用的数据和“正在”执行的程序。内存单元的基本单位是字节，为了方便对内存的访问，每一个内存单元都有一个编号，就像门牌号一样，这个编号就是内存的地址。C 程序中的每一个变量，C 程序的每一个函数，在内存中都会对应一定的内存单元。

注意：内存单元的地址与内存单元中的数据是两个完全不同的概念。

2. 变量的地址——系统分配给变量的内存单元的起始地址

如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配一个或若干个存储单元。不同的数据类型分配不同字节的存储空间。例如，一般微机使用的 C 系统对整型变量分配 2 个字节，对浮点型变量分配 4 个字节，对字符型变量分配 1 个字节，

而每一字节都有一个地址，一般把每个数据的首字节地址称为该数据的地址。

假如有如下定义：

```
int i,j,k;
```

编译时系统分配 2000 和 2001 两个字节给变量 i，2002 和 2003 两个字节给变量 j，2004 和 2005 两个字节给变量 k，如图 6-1 所示。

3. 变量值的存取——通过变量在内存中的地址进行

在程序中一般是通过变量名来对内存单元进行存取操作的。其实程序经过编译以后已经将变量名转换为变量的地址，对变量值的存取都是通过地址进行的。

系统执行“scanf("%d",&i);”和“printf("i=%d\n",i);”存取变量 i 值的方式可以有两种：

（1）直接访问——直接利用变量的地址进行存取。

①对于语句“printf("%d",i);”它是这样执行的：根据变量名与地址的对应关系（这个对应关系是在编译时确定的），找到变量 i 的地址 2000，然后从由 2000 开始的两个字节中取出数据（即变量的值），把它输出。这里输出的值为 3。

②系统对语句“scanf("%d",&i);”在执行时，如果从键盘输入 3，表示要把 3 送到变量 i 中，实际上是把 3 送到地址为 2000 开始的整型存储单元中。

如果有语句

```
k=i+j;
```

则从 2000 和 2001 字节取出 i 的值（3），再从 2002 和 2003 字节取出 j 的值（假设为 6），将它们相加后再将其和（9）送到 k 所占用的 2004 和 2005 字节单元中。这种按变量地址存取变量值的方式称为“直接访问”方式。

（2）间接访问——通过另外一个变量访问该变量的值。C 语言规定，可以在程序中定义整型变量、浮点型变量、字符变量等，也可以定义这样一种特殊的变量，它是存放地址的。假设定义了一个变量 i_pointer，用来存放整型变量的地址，它被分配为 3010 和 3011 两个字节。可以通过下面的语句将 i 的起始地址（2000）存放到 i_pointer 中。

```
i_pointer=&i;
```

这时，i_pointer 的值就是 2000，即变量 i 所占用单元的起始地址。要存取变量 i 的值，也可以采用间接方式：先找到存放“i 的地址”的变量 i_pointer，从中取出 i 的地址（2000），然后到 2000 和 2001 字节取出 i 的值（3），见图 6-1。

打个比方，为了开一个 A 抽屉，有两种办法，一种是将 A 钥匙带在身上，需要时直接找出 A 钥匙打开 A 抽屉，取出所需的东西。另一种办法是：为安全起见，将 A 钥匙放到另一抽屉 B 中锁起来。如果需要打开 A 抽屉，就要先找出 B 钥匙，打开 B 抽屉，取出 A 钥匙，再打开 A 抽屉，取出 A 抽屉中之物，这就是“间接访问”。

（3）两种访问方式的比较。图 6-2（a）表示直接访问，已经知道变量 i 的地址，根据此地址直接对变量 i 的存储单元进行存取访问（图示把数值 3 存放到 i 中）。图 6-2（b）表示间接访问，先找到存放变量 i 地址的变量 i_pointer，从其中得到变量 i 的地址，再据此找到变量 i 的存储单元，然后对它进行存取访问。

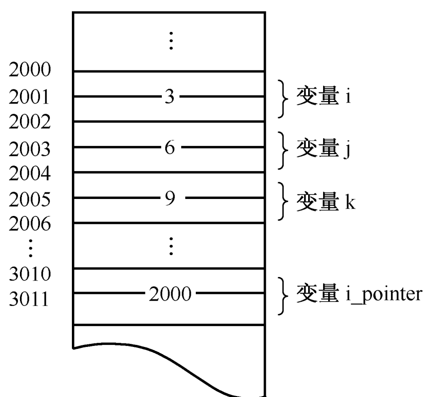


图 6-1　变量存储

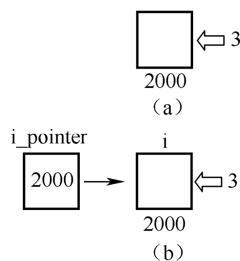


图 6-2　间接访问

为了表示将数值 3 送到变量中，可以有两种表达方法：

（1）将 3 送到变量 i 所标识的单元中，见图 6-2（a）。

（2）将 3 送到变量 i_pointer 所指向的单元（即 i 所标识的单元）中，见图 6-2（b）。

所谓指向就是通过地址来体现的。假设 i_pointer 中的值是变量 i 的地址（2000），这样就

在 i_pointer 和变量 i 之间建立起一种联系，即通过 i_pointer 能知道 i 的地址，从而找到变量 i 的内存单元。图 6-2 中以箭头表示这种“指向”关系。

由于通过地址能找到所需的变量单元，我们可以说，地址指向该变量单元。打个比方，一个房间的门口挂了一个房间号 2008，这个 2008 就是房间的地址，或者说，2008“指向”该房间。因此在 C 语言中，将地址形象化地称为“指针”，意思是通过它能找到以它为地址的内存单元（例如根据地址 2000 就能找到变量 i 的存储单元，从而读取其中的值）。

一个变量的地址称为该变量的“指针”。例如，地址 2000 是变量 i 的指针。如果有一个变量专门用来存放另一个变量的地址（即指针），则它称为“指针变量”，指针变量就是地址变量（存放地址的变量）。上述的 i_pointer 就是一个指针变量。指针变量的值（即指针变量中存放的值）是地址（即指针）。

6.1.2 指向变量的指针变量

1．指针与指针变量

（1）指针，即地址，一个变量的地址称为该变量的指针。通过变量的指针能够找到该变量。

（2）指针变量，专门用于存储其他变量地址的变量。指针变量 i_pointer 的值就是变量 i 的地址。

（3）为表示指针变量和它指向的变量之间的关系，用指针运算符“*”表示。

例如，指针变量 i_pointer 与它所指向的变量 i 的关系，表示为：*i_pointer，即*i_pointer 等价于变量 i。

因此，下面两个语句的作用相同：

```
i=3;              /*将 3 直接赋给变量 i*/
i_pointer=&i;     /*使 i_pointer 指向 i*/
*i_pointer=3;     /*将 3 赋给指针变量 i_pointer 所指向的变量*/
```

2．指针变量的定义与应用

与一般变量的定义相比，除变量名前多了一个星号“*”（指针变量的定义标识符）外，其余一样：

```
数据类型  *指针变量[,指针变量 2……];
```

例如：int x=10, *p,y;

这里定义了两个整型变量 x 和 y，一个指针变量 p。

【例 6.1】指针变量的定义。

```
#include "stdio.h"
void main()
{
    int num_int=12,*p_int;       /*定义一个指向 int 型数据的指针变量 p_int*/
    float num_f=3.14,*p_f;       /*定义一个指向 float 型数据的指针变量 p_f*/
    char num_ch='p',*p_ch;       /*定义一个指向 char 型数据的指针变量 p_ch*/
    p_int=&num_int;              /*取变量 num_int 的地址，赋值给 p_int*/
    p_f=&num_f;                  /*取变量 num_f 的地址，赋值给 p_f*/
    p_ch=&num_ch;                /*取变量 num_ch 的地址，赋值给 p_ch*/
    printf("num_int=%d,*p_int=%d\n",num_int,*p_int);
```

```
        printf("num_f=%d,*p_f=%d\n",num_f,*p_f);
        printf("num_ch=%d,*p_ch=%d\n",num_ch,*p_ch);
    }
```

程序的运行结果为：

```
num_int=12,*p_int=12
num_f=3.14,*p_f=3.14
num_ch=p,*p_ch=p
```

注意：本例中的第 4 到第 6 行的指针变量 p_int、p_f、p_ch 开始时，并未指向某个具体的变量（称指针是悬空的）。使用悬空指针很容易破坏系统，导致系统瘫痪。

取地址运算的格式：＆变量

例 6.1 中的＆num_int、＆num_f、＆num_ch 的结果分别为变量 num_int、num_f、num_ch 的地址。

注意：指针变量只能存放指针（地址），且只能是相同类型变量的地址。

例如：int x=10, *p,y;

```
P=&x;        /*取变量 x 的地址赋给指针变量 p*/
y=*p;        /**p 表示取指针变量 p 所指单元的内容，即变量 x 的值，则 y=10*/
```

此例中第一个语句和第三个语句都出现了“*p”，但意义是不同的 。这是因为“*”在类型说明和在取值运算中的含义是不同的。在第一个语句中的“*p”表示将变量 p 说明为指针变量，用“*”以区别于一般变量，这里是说明指针变量 p。而在第三个语句中的“*p”是使用指针变量 p，此时“*”是运算符，表示取指针所指向的内容，即对 p 进行间接存取运算，取变量 x 的值，如图 6-3 所示。

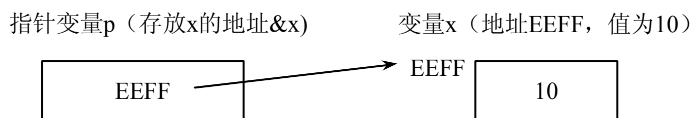


图 6-3　指针变量 p 与整型变量 x 的关系

在 C 语言中，如果 int x=10, y,*px;且有 px=＆x；则变量 x 与指向变量 x 的指针 px 之间有以下等价关系：

```
y=x;等价于 y=*px;                /*y=指针变量 px 的内容*/
x++;等价于(*px)++;               /*对指针变量 px 的内容加 1*/
y=x+5 等价于 y=(*px)+5;          /*这里的括号是不能省略的*/
y=x;等价于 y=*(&x);              /*y=变量 x 地址的内容*/
```

C 语言中用 NULL 表示空指针。若有语句：

```
p=NULL;
```

则表示指针 p 为空，没有指向任何对象。

若有语句：

```
if(p==NULL)
{……}
```

其含义是：判断指针 p 是否为空，若指针 p 为空，则表达式成立。

【例 6.2】输入两个整数，按升序（从小到大排序）输出。

```
#include "stdio.h"
void main()
{
   int num1,num2;
   int *num1_p=&num1,*num2_p=&num2,*pointer;
   printf("input the first number:");
   scanf("%d",num1_p);
   printf("input the second number:");
   scanf("%d",num2_p);
   printf("num1=%d,num2=%d\n",num1,num2);
   fi(*unm1_p>*num2_p)      /*如果 num1>num2，则交换指针*/
   {pointer=numi_p;num1_p=num2_p;num2_p=pointer;}
    printf("min=%d,max=%d\n",*num1_p,*num2_p);
}
```

程序运行情况如下：

```
input the first number:9↙
input the second number:6↙
num1=9,num2=6
min=6,max=9
```

6.2　指针与数组

一个变量有一个地址，一个数组包含若干个元素，每个数组元素都在内存中占有存储单元，它们都有相应的地址。指针变量既然可以指向变量，也可以指向数组和数组元素。所谓数组的指针是指数组的起始地址，数组元素的指针就是数组元素的地址。

引用数组元素可以用下标法（如 a[3]），也可以用指针的方法，即通过指向数组元素的指针找到所需的元素。使用指针法能使目标程序质量更高。

6.2.1　指向数组元素的指针

指向数组元素的指针变量的定义，与指向普通变量的指针变量的定义方法一样。

例如，

```
int array[10];    （定义 array 为包含 10 个整型变量的数组）
int *pointer;     （定义 pointer 为指向整型变量的指针变量）
```

应当注意，如果数组为 int 型，则指针变量也应该为 int 型。

下面是对指针变量的赋值：

```
pointer=&array[0];
```

把 array[0]元素的地址赋给指针变量 pointer。也就是说，pointer 指向 array 数组的第 0 号（第 1 个）元素。

C 语言规定数组名代表数组的首地址，也就是第一个元素的地址。因此，下面两个语句等价。

```
pointer=&array[0];
pointer=array;
```

注意：数组名不代表整个数组，上述“pointer=array;”的作用是“把 array 数组的第一个元素的地址赋给指针变量 pointer”而不是“把 array 数组各元素的值赋给 pointer”。

在定义指针变量时可以对它赋初值。

```
int *pointer=&array[0];
```

它等效于：

```
int *pointer;
pointer=&array[0];
```

当然定义时也可以写成：

```
int *pointer=array;
```

它的作用是把 array 数组的第一个元素的地址赋给指针变量 pointer。

6.2.2　通过指针引用数组元素

1．通过指针引用一维数组中的元素

如果有 int array[10],*pointer=array;则：

- pointer+i 和 array+i 都是数组元素 array[i]的地址。
- *(pointer+i)和*(array+i)就是数组元素 array[i]。
- 指向数组的指针变量，也可将其看作数组名，因而可按下标法来使用。例如，pointer[i]等价于*(pointer+i)。

注意：pointer+1 指向数组的下一个元素，而不是简单地使指针变量 pointer 的值+1。其实际变化为 pointer+1*size（size 为一个元素占有的字节数）。

例如，假设指针变量 pointer 的当前值为 3000，则 pointer+1 为 3000+1*2=3002，而不是 3001。

【例 6.3】使用指向数组的指针变量来引用数组元素。

```
#include "stdio.h"
#include "ctype.h"
main()
{
  int array[10],*pointer=array,i;
  printf("Input 10numbers:");
  for(i=0;i<10;i++)
     scanf("%d",pointer+i);          /*使用指针变量来输入数组元素的值*/
  printf("array[10]:");
  for(i=0;i<10;i++)
     printf("%d  ",*(pointer+i));    /*使用指向数组的指针变量输出数组*/
   printf("\n");
}
```

程序的运行情况如下：

```
Input 10 numbers: 0 1 2 3 4 5 6 7 8 9 ↙
Array[10]: 0  1   2   3   4   5   6   7   8   9
```

程序说明：程序中的两个 for 语句等价于下面的程序段：

```
for(i=0;i<10;i++,pointer++)
   scanf("%d",pointer);
printf("array[10]:");
  pointer=array;
  for(i=0;i<10;i++,pointer++)
    printf("%d  ",*pointer);
```

如果去掉“pointer=array;”行，程序运行的结果如何？请读者思考后上机验证。对于本例中的循环变量 i，也可以不用 i 来做循环控制变量，程序应该如何修改？请读者思考。

说明：

（1）指针变量的值是可以改变的，所以必须知道其当前值，否则容易出错。

（2）指向数组的指针变量，可以指向数组以后的内存单元，但是没有实际意义。

（3）对指向数组的指针变量（px 和 py）进行算术运算和关系运算的含义如下：

可以进行以下几种算术运算：px±n,px++/++px,px--/--px,px-py。

px±n：将指针当前位置向前（+n）或后退（-n）n 个数据单位，而不是 n 个字节。显然 px++/++px,px--/--px 是 px±n 的特例（n=1）。

px-py：两指针之间的数据个数，而不是指针的地址之差。

（4）关系运算：表示两个指针所指地址之间、位置的前后关系：前者为小，后者为大。例如，如果指针 px 所指地址在指针 py 所指地址之前，则 px<py 的值为 1。

2. 通过指针引用二维数组中的元素

在 C 语言中，二维数组是按行优先的规律转换为一维线性数组存放在内存中的，因此，可以通过指针访问二维数组中的元素。

如果有：`int a[M][N];`

则将二维数组中的元素 a[i][j]转换为一维线性地址的一般公式是：

线性地址=a+i×M+j

其中：a 为数组的首地址，M 和 N 分别为二维数组行和列的元素个数。

若有：
```
int a[4][3],*p;
P=&a[0][0];
```

则二维数组 a 的数据元素在内存中的存储顺序及地址关系如图 6-4 所示。

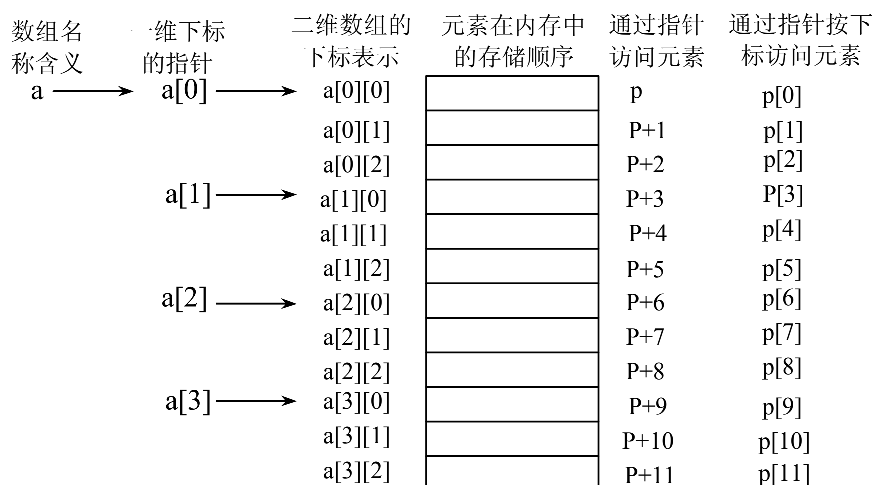


图 6-4 二维数组的元素在内存中的存储顺序及地址关系

这里，a 表示二维数组的首地址，a[0]表示第 1 行元素的起始地址，a[1]表示第 2 行元素的起始地址，a[2]和 a[3]分别表示第 3 行和第 4 行元素的起始地址。

数组元素 a[i][j]的存储地址是：&a[0][0]+i*n+j。

可以说，a 和 a[0]是数组元素 a[0][0]的地址，也是第 1 行的起始地址。a+1 和 a[1]是数组元素 a[1][0]的地址，也是第 2 行的起始地址。

由于 a 是二维数组，经过两次下标运算[]之后才能访问到数组元素。所以根据 C 语言的地址计算方法，a 要经过两次*操作后才能访问到数组元素。

【例 6.4】给定某年某月某日，将其转换成这一年的第几天并输出。

算法分析：给定的月是 i，则将 1，2，3，…，i-1 月的各月的天数累加，再加上指定的日。但对于闰年，二月的天数为 29 天，因此还要判断给定的年是否为闰年，为了实现这一算法，需要设置一张每月天数的列表，给出每月的天数，考虑闰年非闰年的情况，此表可以设置成一个两行 13 列的二维数组，其中第 1 行对应的每列（设 1～12 列有效）元素是平年各月的天数，第 2 行对应的是闰年各月的天数。程序中使用指针作为函数 day_of_year 的形式参数。

```
#include "stdio.h"
#include "ctype.h"
main()
{
   Static int day_tab[2][13]={{0,31,28,31,30,31,30,31,31,30,31,30,31},
   {0,31,29,31,30,31,30,31,31, 30,31,30,31}};
   int y,m,d;
   scanf("%d%d%d",&y,&m,&d );
   printf("%d\n",day_of_year(day_tab,y,m,d));    /*实参为二维数组名*/
}
day_of_year(day_tab,year,month,day)
int *day_tab;                                    /*形式参数为指针*/
int year,month,day;
{
   int i,j;
   i=(year%4==0&&year%100!=0)||year%400==0;
   for(j=1;j<month;j++)
   day+=*(day_tab+i*13+j);  /*day_tab+i*13+j;对二维数组中元素进行地址变换*/
   return(day);
}
```

由于 C 语言对于二维数组中的元素在内存中是按行存放的，所以在函数 day_of_year 中要使用公式 day_tab+i*13+j 计算 main 函数的 day_tab 中元素对应的地址。

6.2.3　数组名作为函数参数

数组名作为函数的形参和实参。如：

```
main()
{
   int array[10];
   f(array,10);
}
f(arr,n)
```

```
int arr[],n;
{
  …
}
```

array 为实参数组名，arr 为形参数组名。前面已经介绍，当数组名作为实参时，如果形参数组中各元素的值发生了变化，实参数组元素的值也随之变化。这是为什么？在学习本节内容后，对此问题就容易理解了。

先看数组元素作实参的情况。例如：

```
swap(a[1],a[2]);
```

如果已定义 swap(x,y)函数的作用是将两个形参的值交换。则上述函数调用的作用是使形参变量 x 和 y 交换。用数组元素作实参的情况与变量作实参的情况一样，是“值传递”方式，单向传递数据，此时 a[1]和 a[2]的值并不改变。

再看用数组名作为函数参数的情况。前面已经介绍，数组名代表数组的首地址。因此，用数组名作实参，在调用函数时实际上是把数组的首地址传给形参（注意，不是把数组的值传给形参）。这样实参数组与形参数组共同占用同一段内存，如图 6-5 所示。

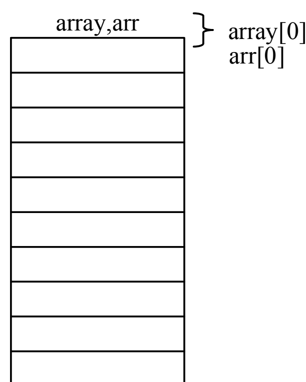


图 6-5　实参数组与形参数组在内存中的关系

由图可知，实参数组和形参数组各元素之间并不存在“值传递”，在函数调用前，形参数组并不占用内存，在函数调用的时候，形参数组并没有另外开辟新的存储单元，而是以实参数组的首地址作为形参数组的首地址，这样实参数组 array 的第 1 个元素和形参数组 arr 的第 1 个元素共占一个内存单元。同理，array[1]与 arr[1]共占一个内存单元。如果在函数调用过程中使形参数组 arr 的元素值发生变化也就使得实参数组的元素值发生了变化。

注意，在调用函数后，实参数组的元素值可能会发生改变，这种值的变化实际上并不是从形参传回实参的，而是由于形参和实参共享同一段内存造成的。前面已经介绍，为了便于理解，我们曾说数组作为函数参数的时候，参数的虚实结合是“双向传递”的。现在我们知道，实际上并不是将形参的值传回实参，数据的传递仍然是单向（把实参数组首地址传给形参数组），函数调用结束后，实参数组各元素所在单元的内容已经改变，当然在主调函数中可以利用这些已经改变的值。

【例 6.5】将数组 a 中 n 个整数按照相反顺序存放。

算法分析：将 a[0]与 a[n-1]对换，再将 a[1]与 a[n-2]对换，……，直到将 a[(n-1)/2]与 a[n-int((n-1)/2)]对换。现在用循环处理这个问题，设两个“位置指示变量”i 和 j，i 的初始值为 0，j 的初始值为 n-1。将 a[i]和 a[j]交换，然后使 i 的值加 1，j 的值减 1，再将 a[i]和 a[j]对换，直到 i=(n-1)/2 为止。

```
# include "stdio.h"
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;
}

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

运行情况如下：

```
3,1,9,11,0,6,7,5,4,2
2,4,5,7,6,0,11,9,1,3
```

主函数中数组名为 a，赋以各元素初值。函数 inv 中的形参数组名为 x，在 inv 函数中不具体定义数组元素的个数，元素个数由形参变量 n 传入（本例中对应的实参值为 10）。这样做可以增加函数的灵活性。即不必要求函数 inv 中的形参数组 x 和 main 函数中的实参数组 a 长度相同。如果在 main 函数中有函数调用语句“inv(a,10)”，表示要求对 a 数组的前 10 个元素按照要求进行颠倒排列。如果改为“inv(a,5)”，则表示将数组 a 的前 5 个元素按照要求进行颠倒排列，此时，函数 inv 只处理 5 个数组元素。函数 inv 中的 m 是 i 的上限，当 i<m 时，循环继续执行，当 i>m 时，循环过程结束。例如，若 n=10，则 m=4。最后一次 a[i]与 a[j]的交换是 a[4]与 a[5]的交换。

对这个程序可以做一些改动。将函数 inv 中的形参 x 改为指针变量。实参为数组名 a，即数组 a 的首地址，传递给形参指针变量 x，x 就指向 a[0]。x+m 是 a[m]元素的地址。设 i 和 j 以及 p 都是指针变量，用它们指向有关元素。i 的初始值为 x，j 的初始值为 x+n-1，见图 6-6。使*i 和*j 交换就是使 a[i]与 a[j]交换。

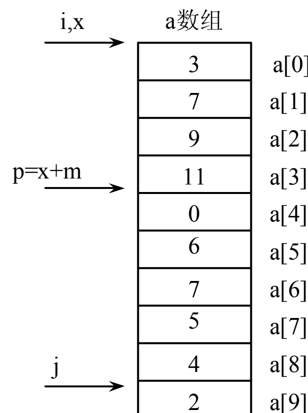


图 6-6　指针变量与数组的关系

【例 6.6】程序如下：

```
#include "stdio.h"
void inv(int *x,int n)
{int p,temp,*i,*j,m=(n-1)/2;
i=x;j=x+n-1;p=x+m;
for(;i<=p;i++,j--)
```

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

程序的运行情况与前一段程序相同。

【例 6.7】从 10 个数中找出其中最大值和最小值。

算法分析：本题要求不改变数组元素的值，只须找出该数组中的最大值与最小值，而函数只能得到一个返回值，现在用全局变量在函数之间传递数据。

程序如下：

```
#include "stdio.h"
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];
for(i=0;i<10;i++)
 scanf("%d",&number[i]);
max_min_value(nember,10);
printf("\nmax=%d,min=%d\n",max,min);
}
```

运行结果如下：

```
-5 6 7 67 54 89 0 34 -23 19↙
max=89,min=-23
```

在函数 max_min_value 中求出最大值和最小值放在 max 和 min 中。由于它们是全局变量，因此在主函数中可以直接使用。

函数 max_min_value 中的语句：

```
max=min=*array;
```

array 是形参数组名，它接收从实参数组 number 传来的 number 的首地址。array 是形参数组的首地址，*array 相当于*(array+0)，即 array[0]。上述语句与下面语句等价：

```
max=min=array[0];
```

如图 6-7 所示。在执行 for 循环时，p 的初始值为 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，将大者存放在 max 中，小者存放在 min 中，效果相同。

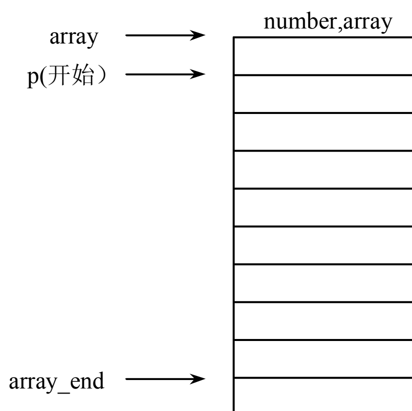


图 6-7　形参实参为指针变量

与上例相似，函数 max_min_value 的形参 array 可以改为指针变量类型，即将该函数中的定义部分

```
void max_min_value(int array[],int n)
```

改为

```
void max_min_value(int *array,int n)
```

效果相同。

实参也可以不用数组名，而用指针变量传递地址。程序可改为：

【例 6.8】

```
#include "stdio.h"
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;
   for(i=0;i<10;i++)
     scanf("%d",p);
   for(p=number,i=0;i<10;i++,p++)
```

```
    printf("%d",*p);
  p=number;
  max_min_value(p,10);
  printf("\nmax=%d,min=%d\n",max,min);
}
```

数组名作为形参时，接收实参数组的起始地址；作为实参时，将数组的起始地址传递给形参数组。

引入指向数组的指针变量后，数组及指向数组的指针变量作函数参数时，可有 4 种等价形式（本质上是一种，即指针数据作函数参数）：

- 形参和实参都用数组名。
- 形参和实参都用指针变量。
- 形参用指针变量、实参用数组名。
- 形参用数组名、实参用指针变量。

6.2.4 指针与字符数组

字符串在内存中的起始地址称为字符串的指针，可以定义一个字符指针变量指向一个字符串。

1. 字符串的表示和引用

在 C 语言中，既可以用字符数组表示字符串，也可以用字符指针变量来表示；引用时，既可以逐个字符引用，也可以整体引用。

（1）逐个引用。

【例 6.9】使用字符指针变量表示和引用字符串。

```
#include "stdio.h"
main()
{
   char *string="I love Beijing.";
   for(;*string !='\0';string++)
   printf("%c",*string);
   printf("\n");
}
```

程序运行结果：

```
I love Beijing
```

对于语句“char *string="I love Beijing";”，其作用是初始化字符指针变量 string，用串常量“I love Beijing.”的地址（由系统自动开辟、存储串常量的内存块的首地址）给 string 赋初值。

该语句也可以分成如下两条语句：

```
char *string;
string="I love Beijing.";
```

注意：字符指针变量 string 中，仅存储串常量的地址，而串常量的内容（即字符串本身）是存储在由系统自动开辟的内存块中，并在串尾添加一个结束标志'\0'。

（2）整体引用。

【例 6.10】采用整体引用的办法改写例 6.9。

```
#include "stdio.h"
main()
{
   char *string="I love Beijing.";
   printf("%s",*string);
}
```

对于语句“printf("%s",*string);”是通过指向字符串的指针变量 string，整体引用它所指向的字符串的原理：系统首先输出 string 指向的第一个字符，然后 string 自动加 1，使之指向下一个字符；重复上述过程，直到遇到字符串结束标志。

注意：其他类型的数组是不能用数组名来一次性输出它的全部元素的，只能逐个元素输出。

例如：

```
int array[10]={……};
……
 printf("%d\n",array);   /*这种用法是非法的*/
……
```

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 6.11】将字符串 a 复制到字符串 b 中。

```
#include "stdio.h"
main()
{
   char a[]="Hello World!",b[20];
   int i;
   for(i=0;*(a+1)!='\0';i++)
      *(b+i)=*(a+i);
      *(b+i)= '\0';
   printf("string a is :%s\n",a);
   printf("string b is:");
   for(i=0;b[i]!= '\0';i++)
      printf("%c",b[i]);
   printf("\n");
}
```

程序运行结果为：

```
string a is : Hello World!
string b is : Hello World!
```

程序中的 a 和 b 都定义为字符数组，可以用地址方法表示数组元素。在 for 语句中，先检查 a[i]是否为'\0'（a[i]是以*(a+i)形式表示的）。如果不等于'\0'，表示字符串尚未处理完，就将 a[i]的值赋给 b[i]，即复制一个字符。在 for 循环中将 a 串全部复制给了 b 串，还应将'\0'复制过去，故有“*(b=1)= '\0';”。

此时的 i 值是字符串有效字符的个数 n 加 1。第二个 for 循环中用下标法表示数组元素（即一个字符）。

也可以设指针变量，用其值的改变来指向字符串中不同的字符。

【例 6.12】用指针变量来处理例 6.11 的问题。

```
#include "stdio.h"
main()
```

```
{ char a[]="Hello World!",b[20],*p1,*p2;
  int i;
  p1=a;p2=b;
  for(;*p1!='\0';p1++,p2++)
    *p2=*p1;
    *p2='\0';
  printf("string a is :%s\n",a);
  printf("string b is:");
   for(i=0;b[i]!= '\0';i++)
     printf("%c",b[i]);
     printf("\n");
}
```

p1、p2是指针变量，它指向字符型数据。先使p1和p2的值分别为字符串a和b的首地址。*p1最初为'H'，赋值语句“*p2=*p1”的作用是将'H'（a串中第1个字符）赋给p2所指向的元素，即b[1]。然后p1和p2分别加1，指向其下面一个元素，直到*p1的值为'\0'止。注意p1和p2 的值是不断改变的，程序必须保证使p1和p2 同步移动。

2. 字符指针变量与字符数组的比较

虽然用字符指针变量和字符数组都能实现字符串的存储与处理，但二者是有区别的，不能混为一谈。

（1）存储内容不同。字符指针变量中存储的是字符串的首地址，而字符数组中存储的是字符串本身（数组的每个元素都存放一个字符）。

（2）赋值的方式不同。对于字符指针变量，可采用下面的赋值语句赋值：

```
char *pointer;
pointer="This is a example.";
```

而字符数组虽然可以在定义时初始化，但不能用赋值语句整体赋值。下面的语句是非法的：

```
char char_array[20];
char_array="This is a example.";        /*非法的用法*/
```

（3）指针变量的值是可以改变的，字符指针变量也不例外。

【例6.13】利用指针字符变量输出一串字符。

```
main()
{
  char *a="I love  China!";
  a=a+7;
  printf("%s",a);
}
```

程序运行的结果为：

```
China!
```

指针变量a的值可以变化，输出字符串时从a当时所指向的单元开始输出各个字符，直到遇'\0'为止。而数组名代表数组的起始地址，是一个常量，而常量是不能改变的。下面是错误的：

```
char str[]={"I love  China!"};
str=str+7;
printf("%s",str);
```

若定义一个指针变量，使它指向一个字符串后，可以用下标形式引用指针变量所指的字符串中的字符。

【例 6.14】

```
#include "stdio.h"
main()
{
  char *a="I LOVE CHINA!";
  int i;
  printf("The sixth character is %c\n",a[5]);
  for(i=0;a[i]!='\0';i++)
  printf ("%c",a[i]);
}
```

程序运行的结果为：

```
The sixth character is E
I LOVE CHINA!
```

程序中虽然并未定义数组 a，但字符串在内存中是以字符数组的形式存放的。a[5]按*(a+5)执行，即从 a 当前指向的元素下移 5 个元素的位置，取出其单元中的值。

3. 字符串指针作为函数参数

【例 6.15】用函数调用方式，实现字符串的复制。

```
#include "stdio.h"
void string_copy(char*str_form,char *str_to)
{ int i=0;
  for(;(*(str_to+i)=*(str_form+i))!='\0';i++);     /*循环体为空语句*/
}
main()
{
  char array_str1[20]="I am a teacher.";
  char array_str2[20];
  string_copy(array_str1,array_str2);              /*数组名作为实参*/
  printf("array_str2=%s\n",array_str2);
}
```

程序的运行结果：

```
I am a teacher.
```

“for(;(*(str_to+i)=*(str_form+i))!='\0\;i++);”语句的执行过程为：首先将源串中的当前字符复制到目标串中；然后判断该字符（即赋值表达式的值）是否为结束标志。如果不是，则相对位置变量增 1，以便复制下一个字符；如果是结束标志，则结束循环。其特点是：先复制、后判断，循环结束前，结束标志已经复制。

在 C 语言中，用复制运算符而不是赋值语句来实现复制操作，能给某些处理带来很大的灵活性，该语句（实现字符串的复制）的用法就是最好的例证。

6.2.5　数组指针

上面介绍的指针都是指向某一类型的变量的，例如可以使一个指针 p 指向某一数组元素。如果 p 的值增 1，则指向下一个元素，这些指针在处理一维数组时非常方便，但在处理二维数

组时就不合适了。如果定义一个指针p，让它指向一个包含n个元素的一维数组，且p的增值以一维数组的长度为单位，此时，如果指针p指向二维数组的某一行，则p+1就指向了该二维数组的下一行。在C++中，这样的指针称为数组指针，使用数组指针可以很方便地处理二维数组。数组指针的说明形式如下：

```
存储类型  数据类型 （*指针名）[元素个数]
```

例如：在程序中定义一个数组指针：

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

它表明指针p指向的存储空间有4个整型元素，即数组指针p指向一个一维数组，p的值就是该一维数组的首地址。

在使用数组指针时，有两点一定要注意：

（1）*p两侧的括号一定不能漏掉，如果写成*p[4]的形式，由于[]的运算级别高，因此p先和[4]结合，是数组，然后再与前面的*结合，*p[4]是指针数组。

（2）p是一个行指针，它只能指向一个包含n个元素的一维数组，不能指向一维数组中的元素，如果要访问一维数组中的某个元素，如第j个元素，可用(*p)[j]表示。

【例6.16】用数组指针处理二维数组。

```
#include "stdio.h"
void main()
{
   int a[2][3]={1,3,5,7,9,11};
   int (*p)[3];
   int i,j;
   for(i=0;i<2;i++)
   {for(j=0;j<3;j++)
      printf("%2d",(*p)[i]);
      p++;
   }
}
```

程序的运行结果为：

```
1 3 5 7 9 11
```

6.3 指针与函数

我们知道在函数之间可以传递一般变量的值，也可以传递地址（指针）。函数与指针之间有着密切的关系，它包含3种含义：指针作为函数的参数，函数的返回值为指针以及指向函数的指针。

6.3.1 指针作函数参数

指针变量，既可以做函数的形参，也可以作函数的实参。指针变量作实参时，与普通变量一样，也是“值传递”，即将指针变量的值（一个地址）传递给被调用函数的形参（必须是一个指针变量）

注意：被调用函数不能改变实参指针变量的值，但可以改变实参指针变量所指向的变量的值。

【例 6.17】使用函数调用方式改写例 6.2，要求实参为指针变量。

```
#include "stdio.h"
void exchange(int *pointer1,int *pointer2)
{ int temp;
  temp=*pointer1;
  *pointer1=*pointer2;
  *pointer2=temp;
}
void main()
{
  int num1,num2;        /*定义并初始化指针变量 num1_p 和 num2_p*/
  int *num1_p=&num1,*num2_p=&num2;
  printf("Input the first number:");
     scanf("%d",num1_p);
  printf("Input the second number:");
     scanf("%d",num2_p);
  printf("num1=%d,num2=%d\n",num1,num2);
  if(*num1_p>*num2_p)            /*即 num1>num2*/
  exchange(num1_p,num2_p);       /*指针变量作实参*/
  printf("min=%d,max=%d\n",num1,num2);  /*输出排序后的 num1 和 num2 的值*/
}
```

程序的运行情况如下：

```
Input the first number:9↙
Input the second number:6↙
num1=9,num2=6
min=6,max=9
```

形参指针变量 pointer1（指向变量 num1）和形参指针变量 pointer2（指向变量 num2），在函数调用开始时才分配存储空间，函数调用结束后立即被释放。

虽然被调用函数不能改变实参指针变量的值，但可以改变它们所指向的变量的值。

为了利用被调用函数改变的变量值，应该使用指针（或指针变量）作函数实参。其机制为：在执行被调用函数时，使形参指针变量所指向的变量的值发生变化；函数调用结束后，通过不变的实参指针（或实参指针变量）将变化的值保留下来。

【例 6.18】输入 3 个整数，按降序（从大到小的顺序）输出。要求使用变量的指针作函数调用的实参来实现。

```
#include "stdio.h"
void exchange(int *pointer1,int *pointer2)
{ int temp;
  temp=*pointer1;
  *pointer1=*pointer2;
  *pointer2=temp;
}
void main()
{
  int num1,num2,num3;
```

```
    /*从键盘上输入 3 个整数*/
    printf("Input the first number:");
      scanf("%d",num1);
    printf("Input the second number:");
      scanf("%d",num2);
    printf("Input the third number:");
      scanf("%d",num3);
    printf("num1=%d,num2=%d,num3=%d",num1,num2,num3)
    /*排序*/
    if(num1<num2)
        exchange(&num1,&num2);
    if(num1<num3)
        exchange(&num1,&num3);
    if(num2<num3)
        exchange(&num2,&num3);
    /*输出排序结果*/
    printf("排序结果:%d,%d,%d\n",num1,num2,num3);
}
```

程序的运行情况如下：

```
Input the first number:9↙
Input the second number:6↙
Input the third number:12↙
num1=9,num2=6,num3=12
排序结果：12，9，6
```

6.3.2 函数指针

1. 函数指针的概念

一个函数在编译时被分配了一个入口地址，这个地址就称为该函数的指针。可以用一个指针变量指向一个函数，然后通过该指针变量调用此函数。

【例 6.19】求 a 和 b 中的大者。

```
#include "stdio.h"
void main()
{ int max();
  int(*p)();
  int a,b,c;
  p=max;
  scanf("%d,%d",&a,&b);
     c=(*p)(a,b);
  printf("a=%d,b=%d,max=%d",a,b,c);
}
int max(int x,int y)
{
  int z;
  if (x>y)
      z=x;
  else
```

```
        z=y;
      return (z);
    }
```

其中“int(*p)();”语句是说明 p 是一个指向函数的指针变量，此函数带回整型的返回值。注意*p 两侧的括号不能省略，表示 p 先与*结合，是指针变量，然后再与后面的()结合，表示此指针变量指向函数，这个函数值（即函数返回的值）是整型的。如果写成“int*p”，则由于()优先级高于*，它就成了说明一个函数了，这个函数的返回值是指向整型变量的指针。

赋值语句“p=max;”的作用是：将函数 max 的入口地址赋给指针变量 p。和数组名代表数组起始地址一样，函数名代表该函数的入口地址。这时，p 就是指向函数 max 的指针变量，也就是 p 和 max 都指向函数的开头，见图 6-8。调用*p 就是调用函数 max。

注意，p 是指向函数的指针变量，它只能指向函数的入口而不能指向函数中间的某一条指令，也不能用*(p+1)来表示函数的下一条语句。

在 main 函数中有一条赋值语句 c=(*p)(a,b);就是用指针形式实现函数的调用。

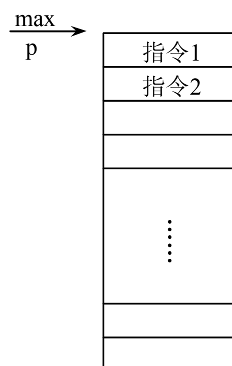


图 6-8　用指针实现函数的调用

2. 指向函数的指针变量

（1）定义格式。

```
函数类型(*指针变量)();
```

注意：“(*指针变量)”外的括号不能缺，否则就成了返回指针值的函数。

例如

```
int(*fp)();    /*fp 为指向 int 函数的指针变量*/
```

（2）赋值。函数名代表该函数的入口地址。因此，可用函数名给指向函数的指针变量赋值。

```
指向函数的指针变量=[&]函数名;
```

注意：函数名后不能带括号和参数；函数名前的“&”符号是可选的。

（3）调用格式。

```
(*函数指针变量)([实参表])
```

函数名作实参时，因为要缺省括号和参数，造成编译器无法判断它是一个变量还是一个函数，所以必须加以说明。

注意：对指向函数的指针变量，诸如 p+i、p++/p--等运算是没有意义的。

3. 指向函数的指针变量作函数参数

指向函数的指针变量的常用用途之一就是将函数指针作参数传递到其他函数。

【例 6.20】设一个函数 process，在调用它时，每次实现不同的功能。输入 a 和 b 两个数，第一次调用 process 时找出 a 和 b 中大者，第二次找出其中小者，第三次求 a 和 b 之和。程序如下：

```
#include"stdio.h"
main()
{ int max(int,int);         /*函数声明*/
  int min(int,int);         /*函数声明*/
```

```
    int add(int,int);         /*函数声明*/
    int a,b;
    scanf("%d,%d",&a,&b);
    printf("max=");
    processs(a,b,max);
    printf("min=");
    process(a,b,min);
    printf("sum=");
    process(a,b,add);
}
max(int x,int y)              /*函数定义*/
{ int z;
    if(x>y)z=x;
    else z=y;
    return(z);
}
min(int x,int y)              /*函数定义*/
{ int z;
    if(x<y)z=x;
    else z=y;
    return(z);
}
add(int x,int y)              /*函数定义*/
{ int z;
    z=x+y;
    return(z);
}
 process(int x,int y,int(*fun)(int,int))  /*函数定义 int(*fun)(int,int)*/
/*表示 fun 是指向函数的指针，该函数是一个整型函数，有两个整型形参*/
{ int result;
   result=(*fun)(x,y);
   printf("%d\n",result);
}
```

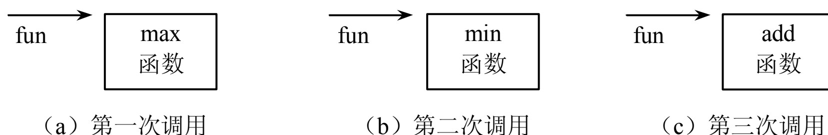
程序运行结果如下：

```
3,9↙
max=9
min=3
sum=12
```

max、min 和 sum 是已定义的 3 个函数，分别用来实现求大数、求小数和求和的功能。在 main 函数中第一次调用 process 函数时，除了将 a 和 b 作为实参将两个数传给 process 的形参 x、y 外，还将函数名 max 作为实参将其入口地址传送给 process 函数中的形参，指向函数的指针变量 fun，如图 6-9（a）所示。

这时，process 函数中的(*fun)(x,y)相当于 max(x,y)，执行 process 可以输出 a 和 b 中的大者。在 main 函数第二次调用时，改以函数名 min 作实参，此时 process 函数的形参 fun 指向函数 min，如图 6-9（b）所示，在 process 函数中的函数调用(*fun)(x,y)相当于 min(x,y)。同理，第三次调用 process 函数时，情况见图 6-9（c），(*fun)(x,y)相当于 add(x,y)。从本例可以清楚地

看到，不论执行 max、min 或 add，函数 process 一点都没有改动，只是在调用 process 函数时实参函数名改变而已。这就增加了函数使用的灵活性，可以编一个通用的函数来实现各种专用的功能。



（a）第一次调用　（b）第二次调用　（c）第三次调用

图 6-9　函数调用示意图

6.3.3　指针函数

指针函数是指返回值为指针的函数，函数的返回值可以是不同类型的指针。

指针函数定义格式如下：

```
函数类型    *函数名([形参表])
```

例如：int *pr(x,y);

pr 是函数名，调用它以后能得到一个指向整型数据的指针（地址）。x、y 是函数 pr 的形参。请注意，在*pr 两侧没有括号，在 pr 的两侧分别为*运算符和( )运算符。而( )的优先级高于*，因此 a 先与( )结合。显然这是函数形式。这个函数前面有一个*，表示此函数是指针型函数（函数值是指针）。最前面的 int 表示返回的指针指向整型变量。对初学 C 语言者来说，这种定义形式可能不大习惯，容易弄错，用时要十分小心。

【例 6.21】某数理化三项竞赛训练组有 3 个人，找出其中至少有一项成绩不合格者。要求使用指针函数实现。

```
int  *seek(int (*pnt_row)[3])
{ int i=0,*pnt_col;          /*定义一个(列)指针变量pnt_col*/
   pnt_col=*(pnt_row+1);     /*使pnt_col指向下一行之首(作标志用)*/
   for(;i<3;i++)
     if(*(*pnt_row+i)<60)    /*某项成绩不合格*/
        {pnt_col=*pnt_row;   /*使pnt_col指向本行之首*/
           break;            /*退出循环*/
         }
   return(pnt_col);
}
/*主函数main()*/
main()
{ int grade[3][3]={{55,65,75},{65,75,85},{75,80,90}};
   int i,j,*pointer;              /*定义一个(列)指针变量pointer*/
   for(i=0;i<3;i++)               /*控制每个学生*/
   { pointer=seek(grade+i);       /*用行指针作实参，调用seek()函数*/
      if(pointer==*(grade+i))     /*该学生至少有一项成绩不合格*/
        {/*输出该学生的序号和各项成绩*/
         printf("No. %d grade list: ",i+1);
         for(j=0;j<3;j++)
```

```
            printf("%d",*(pointer+j));
            printf("\n");
        }
    }
}
```

程序运行结果：

```
No.1 grade list: 55  65  75
```

程序说明如下：

（1）主函数中的 pointer=seek(grade+i);语句调用 seek()函数时，将实参 grade+i（行指针）的值复制到形参 pnt_row（行指针变量）中，使形参 pnt_row 指向 grade 数组的第 i 行。

（2）在指针函数 seek()中：

1）pnt_col=*(pnt_row+1);语句中的*(pnt_row+1)将行指针转换为列指针，指向 grade 数组的第 i+1 行第 0 列，并赋值给（列）指针变量 pnt_col。

2）if(*(*pnt_row+i)<60)行中的 pnt_row 是一个行指针，指向数组 grade 的第 i 行；*pnt_row 使指针由行转换为列，指向数组 grade 的第 i 行第 0 列；*pnt_row+j 的值还是一个指针，指向数组的第 i 行第 j 列；*(*pnt_row+j)是一个数据（数组元素 grade[i][j]的值）。

6.4　多级指针与指针数组

6.4.1　多级指针

1. 概念

在前面的叙述中，一个指针变量可以指向一个相应数据类型的数据，例如：

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

使指针 p 指向 a，则指针 p 所指向的变量*p 就是要处理的数据变量 a。如果同时存在另一个指针 pp，并且把指针 p 的地址赋予指针变量 pp，即：pp=＆p;，则 pp 就指向指针 p，这时指针 pp 所指向的变量*pp 就是指针 p，pp 就是一个二级指针。

2. 定义格式

```
数据类型 **指针变量[,**指针变量 2,…];
```

【例 6.22】用二级指针处理多个字符串。

```
#include "stdio.h"
main()
{
  static char *name[]={"Follow me","BASIC","FORTRAN","Great Wall",
  "Computerdesign"};
  char**p;
  int i;
  for(i=0;i<5;i++)
  { p=name+i;
    printf("%s\n",*p);
  }
}
```

程序运行结果如下：

```
Follow me
BASIC
FORTRAN
Great Wall
Computerdesign
```

指针数组的元素也可以不指向字符串，而指向整型数据或浮点型数据等，例如：

```
int   a[5]={1,3,5,7,9};
int   *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
int   **p;
```

此时为了得到数据“5”，可以先使 p=num+2，然后输出**p。注意*p 是 p 间接指向的对象地址。而**p 是 p 间接指向的对象的值。

6.4.2　指针数组

1．概念

数组的每个元素都是一个指针数据。指针数组比较适合用于指向多个字符串，使字符串处理更加方便、灵活。

2．定义格式

```
数据类型 *数组名[元素个数]
```

例如：int *pa[5];

表示定义一个由 5 个指针变量构成的指针数组，数组中的每个数组元素——指针，都指向一个整数，其结构如图 6-10 所示。

注意“int*pa[5]”与“int(*pb)[5]”的区别。int(*pb)[5]表示定义了一个指向数组的指针 pb，pb 指向的数组是一维的，大小为 5 的整型数组，其结构如图 6-11 所示。

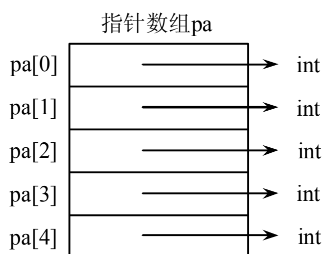


图 6-10　指针数组 pa

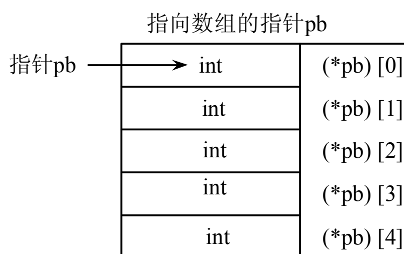


图 6-11　指针数组 pb

【例 6.23】有若干计算机图书，请按字母顺序从小到大输出书名。使用排序函数完成排序，在主函数中进行输入输出。

```
#include"stdio.h"
void sort(char *name[],int count)
{ char*temp_p;
  int i,j,min;
  /*使用选择法排序*/
  for(i=0;i<count-1;i++)                /*外循环：控制选择次数*/
```

```
        {min=i;                                /*预置本次最小串的位置*/
         for(j=i+1;j<count;j++)                /*内循环：选出本次的最小串*/
           if(strcmp(name[min],name[j])>0)     /*存在更小的串*/
             min=j;                            /*保存之*/
        if(min!=i)                             /*存在更小的串，交换位置*/
          {temp_p=name[i];name[min];name[min]=temp_p;}
        }
    }
    /*主函数main()*/
    main()
    { char*name[5]={"BASIC","FORTRAN","PASCAL","C","FoxBASE"};
       int i=0;
       sort(name,5);         /*使用字符指针数组名作实参，调用排序函数sort()*/
       /*输出排序结果*/
       for(;i<5;i++)
        printf("%s\n",name[i]);
      }
```

程序运行结果：

```
BASIC
C
FORTRAN
FoxBASE
PASCAL
```

程序说明如下：

（1）实参对形参的值传递如下：

```
                sort( name  ,      5);
                        ↓          ↓
       void sort(char*name[],int count)
```

（2）字符串的比较只能使用 strcmp()函数。形参字符指针数组 name 的每个元素，都是一个指向字符串的指针，所以有 strcmp(name[min]，name[j])。

（3）程序中没有使用二维字符数组，而是采用指针数组 name。可以看到指针数组比二维字符数组有明显的优点：一是指针数组中每个元素所指的字符串不必限制在相同的字符长度，二是访问指针数组中的一个元素是用指针间接进行的，效率比下标方式要高。

6.4.3 main 函数的参数

指针数组的一个重要应用是作为 main 函数的形参。在前面讲述的程序中，main 函数的第一行全部写成了以下形式：

```
main()
```

括号中为空，表示没有参数。实际上 main 函数是可以带参数的，其一般形式为：

```
main(argc,argv)
int argc;                /*argc表示命令行参数个数*/
char *argv[];            /*argv是指向命令行参数的指针数组*/
```

argc 和 argv 是 main 函数的形式参数。

在操作系统下运行 C 程序时，可以以命令行参数形式，向 main 函数传递参数。命令行参数的一般形式是：

运行文件名　参数 1　参数 2　……　参数 n

运行文件名和参数之间、各个参数之间要用一个空格分隔。

argc 表示命令行参数个数（包括运行文件名），argv 是指向命令行参数的数组指针。指针 argv[0]指向的字符串是运行文件名，argv[1]指向的字符串是命令行参数 1，argv[2]指向的字符串是命令行参数 2，……。

【例 6.24】下列文件的运行文件的文件名为 TEST1，按数组方式引用命令行的参数。

```
/*程序功能：main 函数的参数应用实例*/
#include"stdio.h"
main(argc,argv)
int argc;
char*argv[];
{ int i;
  printf("argc=%d\n",argc);        /*输出参数 argc 的值*/
  for(i=0;i<argc;i++)
     printf("%s\n",argv [i]);      /*按数组方式引用命令行的参数*/
}
```

设在操作系统提示符下运行程序，为了运行程序，输入的命令行参数为：

```
TEST1 IBM-PC COMPUTER
```

则执行程序后输出结果为：

```
argc=3
TEST1
IBM-PC
COMPUTER
```

这样利用指针数组作为主函数 main 的形式参数，可以很方便地实现 main 函数与操作系统的通信。

6.5　动态内存分配与指向它的指针变量

6.5.1　什么是内存的动态分配

前面介绍过全局变量和局部变量，全局变量是分配在内存中的静态存储区域的，非静态的局部变量（包括形参）是分配在内存的动态存储区域的，这个存储区是一个称为栈（stack）的区域。除此之外，C 语言允许建立内存动态分配区域，以存放一些临时的数据，这些数据不必在程序的声明部分定义，也不必等到函数结束时才释放，而是需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为堆（heap）区。可以根据需要，向系统申请所需大小的空间。由于未在声明部分定义它们为变量或数组，因此不能通过变量名或数组名去引用这些数据，只能通过指针来引用。

6.5.2　怎样建立内存的动态分配

对内存的动态分配是通过系统提供的库函数来实现的，主要有 malloc，calloc，free 和 realloc

这 4 个函数。

1. malloc 函数

其函数的原型为：

```
void *malloc(unsigned int size);
```

其作用是在内存的动态存储区中分配一个长度为 size 的连续空间。此函数的值（即“返回值”）是分配区域的起始地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的开头位置。如：

```
Malloc(100);     /*开辟 100 个字节的临时分配域，函数值为其第 1 个字节的地址*/
```

注意指针的基类型为 void，即不指向任何类型的数据，只提供一个地址。如果此函数未能成功地执行（例如，内存空间不足），则返回空指针（NULL）。

2. calloc 函数

其函数的原型为：

```
void *calloc(unsigend n,unsigend size);
```

其作用是在内存的动态存储区域中分配 n 个长度为 size 的连续空间。函数返回一个指向分配域起始位置的指针；如果分配不成功，返回 NULL。

用 calloc 函数可以为一维数组开辟动态存储空间，n 为数组元素的个数，每个元素长度为 size。这就是动态数组。如：

```
p=calloc(50,4);    /*开辟 50×4 个字节的临时分配域，把起始地址赋给指针变量 p*/
```

3. free 函数

其函数的原型为：

```
void free(void *p);
```

其作用是释放由指针变量 p 指向的动态空间，使这部分空间能被其他变量使用。p 是最近一次调用 calloc 或 malloc 函数时返回的值。free 函数无返回值。如：

```
free(p);     /*释放指针变量 p 指向的已经分配的动态空间*/
```

4. realloc 函数

其函数的原型为：

```
void *realloc(void *p,unsigend int size);
```

如果已经通过 malloc 函数或 calloc 函数获得了临时分配域，想改变其大小，可以用 realloc 函数重新分配。将 p 指向的临时分配域大小改变为 size 大小。如：

```
realloc(p,50);  /*将 p 所指向的已分配的动态空间改为 50 个字节*/
```

如果重新分配不成功，返回 NULL。

以上 4 个函数的声明在 stdlib.h 头文件中，在用到这些函数的时候应当用#include<stdlib.h>命令把 stdlib.h 头文件包含到程序文件中。

说明：以前的 C 版本提供的 malloc 和 calloc 函数得到的是指向字符型数据的指针，即其原型为:

```
char *malloc(unsigend int size)
```

但实际上，这些空间并不一定是用来存放字符的，如果程序中把它们用来存放整数，则要进行强制类型转换，如:

```
Int*pt;
pt=(int*)malloc(100);
```

pt 是基类型为整型的指针变量，而 malloc(100)返回的是基类型为字符型的指针，二者不一致，先把 malloc(100)返回的地址强制转换成基类型为整型的地址，然后赋值给 pt，通过 pt 对区域进行存取操作。

ANSI C 新标准把 malloc、calloc 和 realloc 函数的基类型改为 void 类型，即不能确定它指向哪一种具体的类型数据。表示用来指向一个抽象类型的数据，即仅提供一个地址。显然这样的指针是不能直接指向确定的数据的。在使用该地址时可以如上面那样先对它们进行强制类型转换，把它转换为任何其他指针类型。要说明的是类型转换只是产生了一个临时的中间值赋给 pt，并没有改变 malloc 函数本身的类型。

6.6　综合实训

【例 6.25】用选择法对 10 个整数进行排序。

```
#include"stdio.h"
main()
{ int*p,i,a[10];                /*定义一个指向 int 型数据的指针变量 p*/
   p=a;                         /*取数组 a 的首地址，赋值给 p*/
   for(i=0;i<10;i++)
   scanf("%d",p++);             /*通过输入函数给数组元素赋值*/
   p=a;                         /*使指针重新指向数组首地址*/
   sort(p,10);                  /*调用排序函数*/
   for(p=a,i=0;i<10;i++)
   {  printf("%4d",*p);p++;}    /*输出排好序的数组*/
}
sort(int x[10],int n)           /*排序函数*/
{ int i,j,k,t;
   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;}
}
```

运行结果如下：

```
  6   9   10  66  87  -4  -25   0   7  11↙
-25  -4   0   6   7   9   10   11  66  87
```

【例 6.26】有若干个学生的成绩（每个学生有 4 门课程）。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数来实现。

```
#include"stdio.h"
main()
{  float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
   float   *search(float(*pointer)[4],int n);
    float*p;
    int i,m;
    printf("enter the number of student.");
    scanf("%d",&m);
```

```
        printf("The scores of No.%d are: \n",m);
        p=search(score,m);
           for(i=0;i<4;i++)
        printf("%5.2f\t",*(p+i));
     }
     float   *search(float(*pointer)[4],int n)
     { float  *pt;
       pt=*(pointer+n);
       return(pt);
     }
```

运算结果如下：

```
enter the number of student;1<CR>
The scores of No. 1 are :
56.00      89.00       67.00      88.00
```

学生序号是从 0 号算起的。函数 search 被定义为指针型函数，其形参 pointer 是指向包含 4 个元素的一维数组的指针变量。pointer+1 指向 score 数组第 1 行。

*(pointer+1)指向第 1 行第 0 列元素。

pt 是指针变量，它指向实型变量（而不是指向一维数组）。main 函数调用 search 函数，将 score 数组的首地址传给 pointer（注意 score 也是指向行的指针，而不是指向列元素的指针）。m 是要查找的学生序号。调用 search 函数后，得到一个地址（指向第 m 个学生第 0 门课程），赋给 p。然后将此学生的 4 门课的成绩打印出来。*(p+i)表示此学生第 i 门课的成绩。

注意指针变量 p、pt 和 pointer 的区别。如果将 search 函数中的语句 pt=*(pointer+n);改为：pt=(*pointer+n);，则运行结果为：

```
enter the number of student: 1
The score of No. 1 are:
70.00      80.00     90.00     56.00
```

得到的不是第一个学生的成绩，而是二维数组中 a[0][1]开始的 4 个元素的值。

习题六

一、选择题

1．以下程序的输出结果是（　）。

```
# include"stdio.h"
main()
{
   int a=25,*p=&a;
   printf("%d",++*p);
}
```

A．23　　B．24　　C．25　　D．26

2．设有如下定义：

```
char *aa[2]={"abcd","ABCD"};
```

则下列说法中正确的是（　）。

A．aa 数组元素的值分别是"abcd"和"ABCD"

B．aa 是指针变量，它指向含有两个数组元素的字符型数组

C．aa 数组的两个元素分别存放的是含有 4 个字符的一维字符数组的首地址

D．aa 数组的两个元素中各自存放了字符'a'和'A'的地址

3．以下程序的输出结果是（　）。

```
#include"stdio.h"
main()
{int a[]={1,2,3,4,5,6,7,8,9,10,11,12};
int *p=a+5,*q=NULL;
*q=*(p+5);
printf("%d%d",*p,*q);
}
```

A．编译时报错　　B．6 6　　C．6 11　　D．5 5

4．设有如下程序段：

```
char str[]="Hellow";
char *ptr=str;
```

其中*(ptr+6)的值为（　）。

A．'0'　　B．'\0'　　C．6　　D．'0'的地址

5．以下程序输出的结果是（　）。

```
#include "stdio.h"
#include "string.h"
main()
{  char *p1="abc",*p2="ABC",str[50]="xyz";
   strcpy(str+2,p1);
   strcat(str,p2);
   puts(str);
}
```

A．xyzabcABC　　B．xABCabc

C．yzabcABC　　D．xyabcABC

6．对于类型相同的两个指针变量，它们之间不能进行的运算是（　）。

A．<　　B．=　　C．+　　D．-

7．以下程序的输出结果是（　）。

```
#include "stdio.h"
main()
{ int a[]={2,4,6,8,10},y=1,x,*p;
   p=&a[1];
   for(x=0;x<3;x++)
   y+=*(p+x);
   printf("%d",y);
}
```

A．17　　B．18　　C．19　　D．20

8．执行以下程序段后，m 的值为（　）。

```
int a[2][3]={{1,2,3},{4,5,6}};
int m,*p;
p=&a[0][0];
m=(*p)*(*(p+2))*(*(p+4));
```

A．6　　B．18　　C．15　　D．20

9．以下程序的输出结果是（　）。

```
#include "stdio.h"
#include"string.h"
main()
{ char str[][20]={"Hello","Beijing"},*p=str;
  printf("%d\n",strlen(p+20));
}
```

A．0　　B．5　　C．7　　D．20

10．以下程序的输出结果是（　）。

```
#include "stdio.h"
main()
{ char a[]="programming",b[]="language";
  char *p1=a,*p2=b;
  int i;
  for(i=0;i<=7;i++)
  if(*(p1+i)==*(p2+i))printf("%c",*(p1+i));
}
```

A．gm　　B．rg　　C．or　　D．ga

11．库函数 strcpy()用以复制字符串。若有以下定义和语句：

```
char str1[]="string",str2[8],*str3,*str4="string";
```

则对库函数 strcpy()的调用不正确的是（　）。

A．strcpy(str1,"HELLO1");　　B．strcpy(str2,"HELLO2");

C．strcpy(str3,"HELLO3");　　D．strcpy(str4,"HELLO4");

12．以下程序的输出结果是（　）。

```
#include"stdio.h"
main()
{ char *p[5]={"ABCD","EF","GHI","JKL","MNOP"};
  char **q=p;
  int i;
  for(i=0;i<=4;i++)printf("%s",q[i]);
}
```

A．ABCDEFGHIJKL　　B．ABCD

C．ABCDEFGHIJKMNOP　　D．AEJM

13．若有定义语句“int (*p)[M];”其中标识符 p 表示的是（　）。

A．M 个指向整型变量的指针

B．指向 M 个整型变量的函数指针

C．一个指向具有 M 个整型元素的一维数组的指针

D．具有 M 个指针元素的一维数组，每个元素都是指向整型变量的指针

14．若有定义语句“int (*p)();”其中标识符 p 表示的是（　）。

A．指向整型变量的指针　　B．指向整型函数的指针

C．返回整型指针的函数　　D．返回整型变量的函数

15．以下程序段的输出结果是（　）。

```
char *s="abcde";
s+=2;
printf("%s",s);
```

A．cde　　B．字符'c'

C．字符'c'的地址　　D．无确定的输出结果

16．main()函数中参数表示不合法的是（　）。

A．main(int a,char*c[])　　B．main(int arc,char **arv)

C．main(int arge,char *argv)　　D．main(int argv,char *argc[])

17．以下程序的输出结果是（　）。

```
#include"stdio.h"
#include"string.h"
main()
{
  char arr[2][4];
  strcpy(arr[0],"you");
  strcpy(arr[1],"me");
  arr[0][3]='&';
  puts(arr[0]);
}
```

A．you&me　　B．you　　C．me　　D．err

18．以下程序的输出结果是（　）。

```
#include"stdio.h"
void func(int *a,int b[]){b[0]=*a+6;}
main()
{ int a,b[5];
  a=0;
  b[0]=3;
  func(&a,b);
  printf("%d",b[0]);
}
```

A．6　　B．7　　C．8　　D．9

19．若有以下说明和定义：

```
int fun(int *c);
main(){int (*a)()=fun,*b(),w[10],c;…}
```

在必要的赋值之后，对 fun()函数正确调用的表达式是（　）。

A．a(w)　　B．(*a)(&c)　　C．b(w)　　D．fun(b)

20．以下程序的输出结果是（　）。

```
#include"stdio.h"
void fun(int *a,int *b)
```

```
{ int *k;
  k=a;a=b;b=k;}
main()
{ int a=3,b=6,*x=&a,*y=&b;
  fun(x,y);
  printf("%d  %d",a,b);
}
```

A．6 3　　B．3 6　　C．编译出错　　D．0 0

二、填空题

1．对于变量 x，其在内存中的地址可以写成___（1）___；对于数组 y[10]，其首地址可以写成___（2）___或___（3）___；对于数组元素 y[3]，其地址可以写成___（4）___或___（5）___。

2．设有定义语句“int k,*p1=&k,*p2;”，能完成表达式“p2=&k;”功能的表达式可以写成________。

3．以下程序的输出结果是________。

```
#include"stdio.h"
main()
{ int a[5]={2,4,6,8,10},*p,**k;
  p=a;
  k=&p;
  printf("%d  ",*(p++));
  printf("%d",**k);
}
```

4．以下程序的输出结果是________。

```
#include"stdio.h"
main()
{ int a[3]={2,4,6},*prt,x=8,y,z;
  prt=&a[0];
  for(y=0;y<3;y++)
  z=(*(prt+y)<x)?*(prt+y):x;
  printf("%d",z);
}
```

5．以下程序的输出结果是________。

```
char s[20]="goodgood",*sp=s;
sp=sp+2;
sp="to";
printf("%s",s);
```

6．以下程序的输出结果是________。

```
#include"stdio.h"
main()
{
 char b[]="ABCDEFG",*chp=&b[7];
 while(--chp>=&b[0])
```

```
    printf("%c",*chp);
  }
```

7．以下程序输出数组 a 中的最大元素，并由指针 s 指向该元素。请填空。

```
#include"stdio.h"
main()
{
   int a[10]={6,7,2,9,1,10,5,8,4,3},8p,*s;
   for(p=a,s=a;p-a<10;p++)
     if(________) s=p;
   printf(%d",*s);
}
```

8．设有如下定义语句，则表达式“**p2”的值是___（1）___，表达式“*(*p2+1)”的值是___（2）___。

```
int x[3]={1,2,3},*p1=x,**p2=&p1;
```

三、编程题

1．设计一个 select()函数，在 N 行 M 列的二维数组中选出一个最大值作为函数的返回值，并通过形参传回此最大值所在的行下标。

2．设计一个程序，输入一个十进制正整数，输出其对应的十六进制数。

3．设计一个函数，判断一个字符串是否为回文，即顺读和逆读都是一样的字符串，若是，返回 1；否则返回 0。

4．设计一个函数，模拟字符串比较函数 strcmp(str1,str2)。

5．有 n 个整数，使其前面各数顺序向后移 m 个位置，最后面的 m 个数变成最前面的 m 个数。设计一个 move()函数实现上述功能，在 main()函数中输入 n 个数，并输出调整后的 n 个数。

6．设计一个程序，输入一个字符串，内有数字字符和非数字字符，将其中连续的数字字符作为一个整数。统计字符串中共有多少个这样的整数，并输出这些数。例如，字符串"a123x45?6h7890"中共有 4 个所求的整数：123、45、6、7890。

7．设计一个程序，输入 n 为偶数时，调用函数求 1/2+1/4+⋯+1/n；输入 n 为奇数时，调用函数求 1/1+1/3+⋯1/n。要求通过指向函数的指针完成。