第6章 指　　针

指针是C语言的精华部分，通过利用指针，我们能很好地利用内存资源，使其发挥最大的效率。有了指针技术，我们可以描述复杂的数据结构，对字符串的处理可以更灵活，对数组的处理更方便，使程序的书写简洁，高效，清爽。但由于指针对初学者来说，难于理解和掌握，需要一定的计算机硬件的知识做基础，这就需要多做多练，多上机动手，才能在实践中尽快掌握，成为C的高手。

6.1 指针与指针变量

过去，我们在编程中定义或说明变量，编译系统就为已定义的变量分配相应的内存单元，也就是说，每个变量在内存会有固定的位置，有具体的地址。由于变量的数据类型不同，它所占的内存单元数也不相同。若我们在程序中做定义为：

```
int a=1,  b=2;
float x=3.4,     y=4.5;
double m=3.124;
char ch1='a',    ch2='b';
```

让我们先看一下编译系统是怎样为变量分配内存的。变量a,b是整型变量，在内存各占2个字节；x,y是实型，各占4个字节；m是双精度实型，占8个字节；ch1,ch2是字符型，各占1个字节。由于计算机内存是按字节编址的，设变量的存放从内存2000单元开始存放，则编译系统对变量在内存的安放情况为图6-1所示。

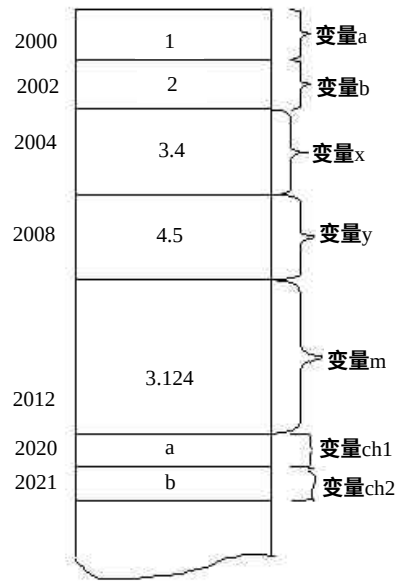

图6-1　不同数据类型的变量在内存中占用的空间

变量在内存中按照数据类型的不同，占内存的大小也不同，都有具体的内存单元地址，如变量 a 在内存的地址是2000，占据两个字节后，变量b的内存地址就为2002，变量m的内存地址为2012等。对内存中变量的访问，过去用scanf("%d%d%f",&a,&b,&x)表示将数据输入变量的地址所指示的内存单元。那么，访问变量，首先应找到其在内存的地址，或者说，一个地址唯一指向一个内存变量，我们称这个地址为变量的指针。如果将变量的地址保存在内存的特定区域，用变量来存放这些地址，这样的变量就是指针变量，通过指针对所指向变量的访问，也就是一种对变量的“间接访问”。

设一组指针变量pa、pb、px、py、pm、pch1、pch2，分别指向上述的变量a、b、x、y、m、ch1、ch2，指针变量也同样被存放在内存，二者的关系如图6-2所示：

在图6-2中，左部所示的内存存放了指针变量的值，该值给出的是所指变量的地址，通过该地址，就可以对右部描述的变量进行访问。如指针变量 pa的值为2000，是变量a在内存的地

址。因此，pa就指向变量a。变量的地址就是指针，存放指针的变量就是指针变量。

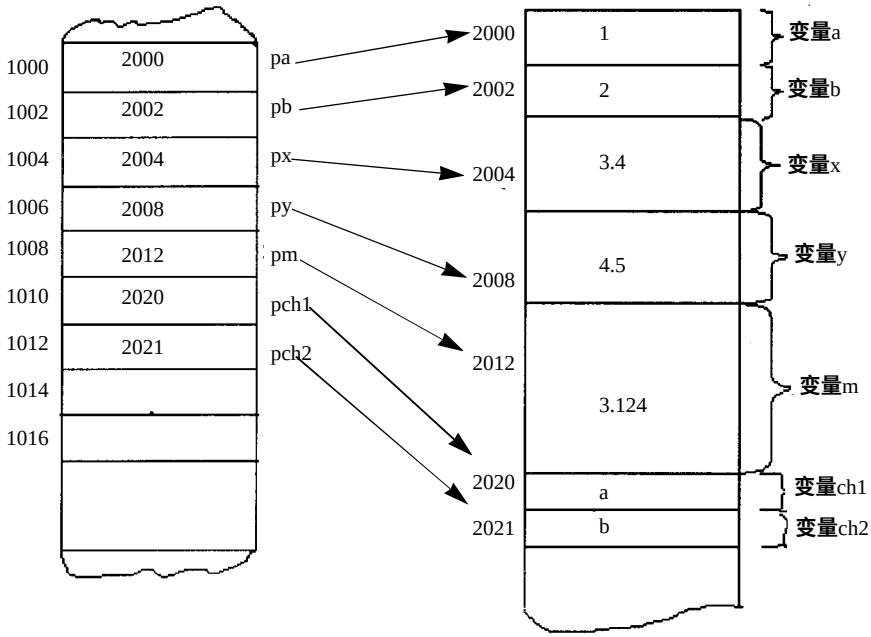

图6-2 指针变量与变量在内存中的关系

6.2 指针变量的定义与引用

6.2.1 指针变量的定义

在C程序中，存放地址的指针变量需专门定义；

```
int *ptr1;
float *ptr2;
char *ptr3;
```

表示定义了三个指针变量ptr1、ptr2、ptr3。ptr1可以指向一个整型变量，ptr2可以指向一个实型变量，ptr3可以指向一个字符型变量，换句话说，ptr1、ptr2、ptr3可以分别存放整型变量的地址、实型变量的地址、字符型变量的地址。

定义了指针变量，我们才可以写入指向某种数据类型的变量的地址，或者说是为指针变量赋初值：

```
int  *ptr1,m= 3;
float  *ptr2, f=4.5;
char  *ptr3, ch='a';
ptr1=&m;
ptr2=&f;
ptr3=&ch;
```

上述赋值语句 ptr1=&m表示将变量m的地址赋给指针变量ptr1，此时 ptr1就指向m。三条赋值语句产生的效果是ptr1指向m；ptr2指向f；ptr3指向ch 。用示意图6-3描述如下：

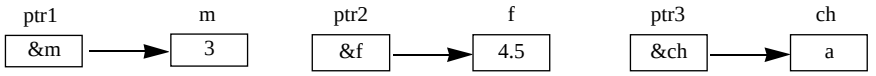

图6-3 赋值语句的效果

需要说明的是，指针变量可以指向任何类型的变量，当定义指针变量时，指针变量的值是随机的，不能确定它具体的指向，必须为其赋值，才有意义。

6.2.2 指针变量的引用

利用指针变量，是提供对变量的一种间接访问形式。对指针变量的引用形式为：

*指针变量

其含义是指针变量所指向的值。

[例6-1] 用指针变量进行输入、输出。

```
main()
{
 int *p,m;
 scanf("%d",&m);
 p=&m;              /*指针p指向变量m*/
 printf("%d",*p);
 /*   p是对指针所指的变量的引用形式,与此m意义相同*/
}
```

运行程序：

```
RUN
 3
 3
```

上述程序可修改为：

```
main()
{
 int *p,m;
 p=&m;
 scanf("%d",p);              /*   p是变量m的地址,可以替换&m*/
 printf("%d", m);
}
```

运行效果完全相同。请思考一下若将程序修改为如下形式：

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

会产生什么样的结果呢？事实上，若定义了变量以及指向该变量的指针为：

```
int a,*p;
```

若p=&a; 则称p指向变量a，或者说p具有了变量a的地址。在以后的程序处理中，凡是可以写&a的地方，就可以替换成指针的表示p，a就可以替换成为*p。

6.3 指针运算符与指针表达式

6.3.1 指针运算符与指针表达式

在C中有两个关于指针的运算符：

- &运算符：取地址运算符，&m即是变量m的地址。
- *运算符：指针运算符，*ptr表示其所指向的变量。

[例6-2] 从键盘输入两个整数，按由大到小的顺序输出。

```
main()
{
 int *p1,*p2,a,b,t;                    /* 定义指针变量与整型变量 */
 scanf("%d,%d",&a,&b);
 p1=&a;                  /* 使指针变量指向整型变量 */
 p2=&b;
 if(*p1<*p2)
   {                                   /* 交换指针变量指向的整型变量 */
    t=*p1;
    *p1=*p2;
    *p2=t;
    }
  printf("%d,%d\n",a,b);
  }
```

在程序中，当执行赋值操作p1=&a和 p2=&b后，指针实实在在地指向了变量 a与b，这时引用指针*p1与*p2，就代表了变量a与b。

运行程序：

```
RUN
   3,4
4,3
```

在程序运行过程中，指针与所指的变量之间的关系如图 6-4所示：

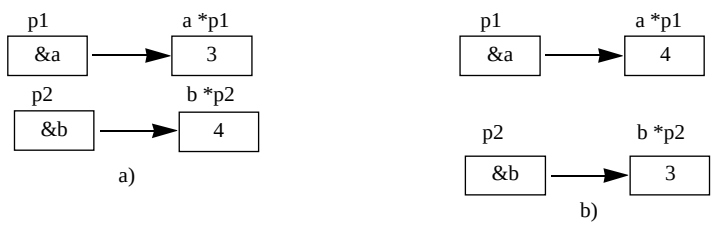

图6-4 程序运行中指针与变量之间的关系

当指针被赋值后，其在内存的安放如 a)，当数据比较后进行交换，这时，指针变量与所指向的变量的关系如 b)所示，在程序的运行过程中，指针变量与所指向的变量其指向始终没变。

下面对程序做修改。

[例6-3]

```
main()
{
 int *p1,*p2,a,b,*t;
 scanf("%d,%d",&a,&b);
 p1=&a;
 p2=&b;
 if(*p1<*p2)
   {       /* 指针交换指向 */
    t=p1;
    p1=p2;
    p2=t;
    }
  printf("%d,%d\n",*p1,*p2);
  }
```

程序的运行结果完全相同，但程序在运行过程中，实际存放在内存中的数据没有移动，而是将指向该变量的指针交换了指向。其示意如图 6-5：

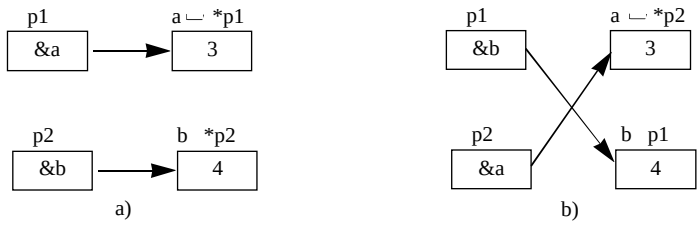

图6-5 修改后的程序在运行中指针与变量之间的关系

当指针交换指向后，p1和p2由原来指向的变量a和b改变为指向变量b和a，这样一来，*p1就表示变量b，而*p2就表示变量a。在上述程序中，无论在何时，只要指针与所指向的变量满足p=&a；我们就可以对变量 a 以指针的形式来表示。此时p等效于&a，*p等效于变量a 。

6.3.2 指针变量作函数的参数

函数的参数可以是我们在前面学过的简单数据类型，也可以是指针类型。使用指针类型做函数的参数，实际向函数传递的是变量的地址。由于子程序中获得了所传递变量的地址，在该地址空间的数据当子程序调用结束后被物理地保留下来。

[例6-4] 利用指针变量作为函数的参数，用子程序的方法再次实现上述功能。

```
main()
{
 void chang();   /* 函数声明 */
 int *p1,*p2,a,b,*t;
 scanf("%d,%d",&a,&b);
 p1=&a;
 p2=&b;
 chang(p1,p2);       /* 子程序调用 */
  printf("%d,%d\n",*p1,*p2);
```

```
 return 0;
 }
void chang(int *pt1,int *pt2)
{        /*子程序实现将两数值调整为由大到小*/
 int t;
 if (*pt1<*pt2)/*交换内存变量的值*/
 {
  t=*pt1;  *pt1=*pt2; *pt2=t;}
  return;
  }
```

由于在调用子程序时，实际参数是指针变量，形式参数也是指针变量，实参与形参相结合，传值调用将指针变量传递给形式参数pt1和pt2。但此时传值传递的是变量地址，使得在子程序中pt1和pt2具有了p1和p2的值，指向了与调用程序相同的内存变量，并对其在内存存放的数据进行了交换，其效果与[例6-2]相同。

思考下面的程序，是否也能达到相同的效果呢？

```
main()
{
 void chang();
 int *p1,*p2,a,b,*t;
 scanf("%d,%d",&a,&b);
 p1=&a;
 p2=&b;
 chang(p1,p2);
  printf("%d,%d\n",*p1,*p2);
}
 void chang(int *pt1,int *pt2)
 {
  int *t;
  if (*pt1<*pt2)
  {
   t=pt1;  pt1=pt2;    pt2=t;
}
   return;
   }
```

程序运行结束，并未达到预期的结果，输出与输入完全相同。其原因是对子程序来说，函数内部进行指针相互交换指向，而在内存存放的数据并未移动，子程序调用结束后，main()函数中p1和p2保持原指向，结果与输入相同。

6.4 指针与数组

变量在内存存放是有地址的，数组在内存存放也同样具有地址。对数组来说，数组名就是数组在内存安放的首地址。指针变量是用于存放变量的地址，可以指向变量，当然也可存放数组的首址或数组元素的地址，这就是说，指针变量可以指向数组或数组元素，对数组而言，数组和数组元素的引用，也同样可以使用指针变量。下面就分别介绍指针与不同类型的数组。

6.4.1 指针与一维数组

假设我们定义一个一维数组，该数组在内存会有系统分配的一个存储空间，其数组的名字就是数组在内存的首地址。若再定义一个指针变量，并将数组的首址传给指针变量，则该指针就指向了这个一维数组。我们说数组名是数组的首地址，也就是数组的指针。而定义的指针变量就是指向该数组的指针变量。对一维数组的引用，既可以用传统的数组元素的下标法，也可使用指针的表示方法。

int a[10] , *ptr; /* 定义数组与指针变量*/

做赋值操作：ptr=a; 或 ptr=&a[0];

则ptr就得到了数组的首址。其中，a是数组的首地址，&a[0]是数组元素a[0]的地址，由于a[0]的地址就是数组的首地址，所以，两条赋值操作效果完全相同。指针变量 ptr就是指向数组a的指针变量。

若ptr指向了一维数组，现在看一下C规定指针对数组的表示方法：

1) ptr+n与a+n表示数组元素a[n]的地址，即&a[n] 。对整个a数组来说，共有10个元素，n的取值为0～9，则数组元素的地址就可以表示为 ptr+0～ptr+9或a+0～a+9，与&a[0] ～&a[9]保持一致。

2) 知道了数组元素的地址表示方法， *(ptr+n)和*(a+n）就表示为数组的各元素即等效于a[n]。

3) 指向数组的指针变量也可用数组的下标形式表示为 ptr[n]，其效果相当于*(ptr+n)。

[例6-5] /*以下标法输入输出数组各元素。

下面从键盘输入10个数，以数组的不同引用形式输出数组各元素的值。

```
# include <stdio.h>
main()
{
 int n,a[10],*ptr=a;
 for(n=0;n<=9;n++)
   scanf("%d",&a[n]);
 printf("1------output! \n");
 for(n=0;n<=9;n++)
 printf("%4d",a[n]);
 printf("\n");
 }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 0
1------output!
1 2 3 4 5 6 7 8 9 0
```

[例6-6] 采用指针变量表示的地址法输入输出数组各元素。

```
# include<stdio.h>
main()
{
 int n,a[10],*ptr=a;          /*定义时对指针变量初始化*/
```

```
 for(n=0;n<=9;n++)
   scanf("%d",ptr+n);
 printf("2------output! \n");
 for(n=0;n<=9;n++)
   printf("%4d",*(ptr+n));
 printf("\n");
 }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 0
2------output!
1 2 3 4 5 6 7 8 9 0
```

[例6-7] **采用数组名表示的地址法输入输出数组各元素。**

```
main()
{
 int n,a[10],*ptr=a;
 for(n=0;n<=9;n++)
   scanf("%d",a+n);
 printf("3------output! \n");
 for(n=0;n<=9;n++)
 printf("%4d",*(a+n));
 printf("\n");
 }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 0
3------output!
1 2 3 4 5 6 7 8 9 0
```

[例6-8] **用指针表示的下标法输入输出数组各元素。**

```
main()
{
 int n,a[10],*ptr=a;
 for(n=0;n<=9;n++)
   scanf("%d",&ptr[n]);
 printf("4------output! \n");

 for(n=0;n<=9;n++)
    printf("%4d",ptr[n]);
 printf("\n");
 }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 0
4----output!
```

```
1 2 3 4 5 6 7 8 9 0
```

[例6-9] **利用指针法输入输出数组各元素。**

```
main()
{
 int n,a[10],*ptr=a;
 for(n=0;n<=9;n++)
   scanf("%d",ptr++);
 printf("5------output! \n");
 ptr=a;     /*指针变量重新指向数组首址*/
 for(n=0;n<=9;n++)
    printf("%4d",*ptr++);
 printf("\n");
 }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 0
5-----output!
1 2 3 4 5 6 7 8 9 0
```

在程序中要注意 *ptr++**所表示的含义。** *ptr**表示指针所指向的变量；** ptr++**表示指针所指向的变量地址加**1**个变量所占字节数，具体地说，若指向整型变量，则指针值加** 2**，若指向实型，则加**4**，依此类推。而** printf(“ %4d ”,*ptr++)**中，** *ptr++**所起作用为先输出指针指向的变量的值，然后指针变量加**1**。循环结束后，指针变量指向如图** 6-6**所示：**

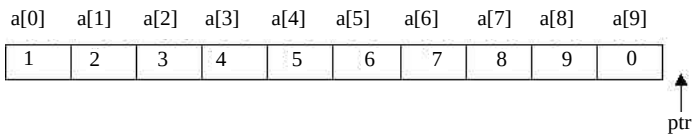

图6-6 例6-9中循环结束后的指 针变量

指针变量的值在循环结束后，指向数组的尾部的后面。假设元素 a[9]**的地址为**1000,**整型占**2**字节，则**ptr**的值就为**1002**。请思考下面的程序段：**

```
main()
{
 int n,a[10],*ptr=a;
 for(n=0;n<=9;n++)
   scanf("%d",ptr++);
 printf("4------output! \n");
 for(n=0;n<=9;n++)
    printf("%4d",*ptr++);
 printf("\n");
 }
```

程序与例6-9**相比，只少了赋值语句**ptr=a**；程序的运行结果还相同吗？**

6.4.2 指针与二维数组

定义一个二维数组：

```
int a[3][4];
```

表示二维数组有三行四列共12个元素，在内存中按行存放，存放形式为图6-7：

其中a是二维数组的首地址， &a[0][0]既可以看作数组0行0列的首地址，同样还可以看作是二维数组的首地址， a[0]是第0行的首地址，当然也是数组的首地址。同理 a[n]就是第n行的首址；&a[n][m]就是数组元素a[n][m]的地址。

既然二维数组每行的首地址都可以用 a[n]来表示，我们就可以把二维数组看成是由 n行一维数组构成，将每行的首地址传递给指针变量，行中的其余元素均可以由指针来表示。下面的图6-8给出了指针与二维数组的关系：

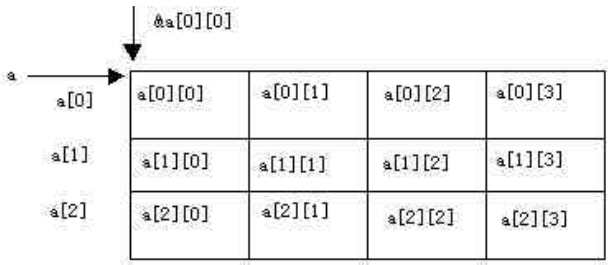

图6-7 二维数组在内存中的存放

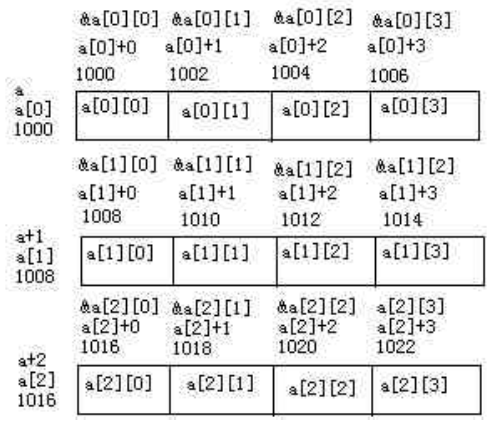

图6-8 指针与二维数组的关系

我们定义的二维数组其元素类型为整型，每个元素在内存占两个字节，若假定二维数组从1000单元开始存放，则以按行存放的原则，数组元素在内存的存放地址为 1000～1022。

用地址法来表示数组各元素的地址。对元素 a[1][2]，&a[1][2]是其地址，a[1]+2也是其地址。分析a[1]+1与a[1]+2的地址关系，它们地址的差并非整数 1，而是一个数组元素的所占位置2，原因是每个数组元素占两个字节。

对0行首地址与1行首地址a与a+1来说，地址的差同样也并非整数 1，是一行，四个元素占的字节数8。

由于数组元素在内存的连续存放。给指向整型变量的指针传递数组的首地址，则该指针指向二维数组。

```
int  *ptr, a[3][4];
```

若赋值： ptr=a；**则用**ptr++ **就能访问数组的各元素。**

[例6-10] **用地址法输入输出二维数组各元素。**

```
# include <stdio.h>
main()
{
 int a[3][4];
 int i,j;
 for(i=0;i<3;i++)
    for(j=0;j<4;j++)
       scanf("%d",a[i]+j);          /*地址法*/
 for(i=0;i<3;i++)
    {
    for(j=0;j<4;j++)
       printf("%4d",*(a[i]+j)); /* *(a[i]是地址法所表示的数组元素*/
    printf("\n");
    }
}
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 10 11 12
1  2     3     4
5  6     7     8
9  10    11    12
```

[例6-11] **用指针法输入输出二维数组各元素。**

```
# include<stdio.h>
main()
{
 int a[3][4],*ptr;
 int i,j;
 ptr=a[0];
 for(i=0;i<3;i++)
    for(j=0;j<4;j++)
        scanf("%d",ptr++);          /*指针的表示方法*/
 ptr=a[0];
 for(i=0;i<3;i++)
    {
    for(j=0;j<4;j++)
        printf("%4d",*ptr++);
    printf("\n");
    }
}
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 10 11 12
1  2     3     4
```

```
5  6     7     8
9  10    11    12
```

对指针法而言，程序可以把二维数组看作展开的一维数组：

```
main()
{
 int a[3][4],*ptr;
 int i,j;
 ptr=a[0];
 for(i=0;i<3;i++)
    for(j=0;j<4;j++)
          scanf("%d",ptr++);        /*指针的表示方法*/
 ptr=a[0];
 for(i=0;i<12  ;i++)
          printf("%4d",*ptr++);
    printf("\n");
       }
```

运行程序：

```
RUN
1 2 3 4 5 6 7 8 9 10 11 12
1   2     3     4     5     6     7     8     9     10    11    12
```

6.4.3 数组指针作函数的参数

学习了指向一维和二维数组指针变量的定义和正确引用后，我们现在学习用指针变量作函数的参数。

[例6-12] 调用子程序，实现求解一维数组中的最大元素。

我们首先假设一维数组中下标为0的元素是最大和用指针变量指向该元素。后续元素与该元素一一比较，若找到更大的元素，就替换。子程序的形式参数为一维数组，实际参数是指向一维数组的指针。

```
# include <stdio.h>
main()
{
 int sub_max();          /*函数声明*/
 int n,a[10],*ptr=a;           /*定义变量，并使指针指向数组*/
 int max;
 for(n=0;n<=i-1;n++)     /*输入数据*/
    scanf("%d",&a[n]);
 max=sub_max(ptr,10);    /*函数调用，其实参是指针*/
 printf("max=%d\n",max);
}
 int sub_max(b,i)        /*函数定义，其形参为数组*/
   int b[],i;
 {
     int temp,j;
     temp=b[0];
```

```
    for(j=1;j<=9;j++)
        if(temp<b[j]) temp=b[j];
    return temp;
  }
```

程序的main()函数部分，定义数组 a 共有10个元素，由于将其首地址传给了 ptr，则指针变量 ptr 就指向了数组，调用子程序，再将此地址传递给子程序的形式参数 b，这样一来，b 数组在内存与 a 数组具有相同地址，即在内存完全重合。在子程序中对数组 b 的操作，与操作数组 a 意义相同。其内存中虚实结合的示意如图 6-9所示。

main()函数完成数据的输入，调用子程序并输出运行结果。 sub_max()函数完成对数组元素找最大的过程。在子程序内数组元素的表示采用下标法。运行程序：

```
RUN
1 3 5 7 9 2 4 6 8 0
max=9
```

[例6-13] 上述程序也可采用指针变量作子程序的形式参数。

```
# include <stdio.h>
main()
{
 int sub_max();
 int n,a[10],*ptr=a;
 int max;
  for(n=0;n<=9;n++)
    scanf("%d",&a[n]);
  max=sub_max(ptr,10);
  printf("max=%d\n",max);
 }
 int sub_max(b,i)        /*形式参数为指针变量*/
 int *b,i;
 {
  int temp,j;
  temp=b[0];             /*数组元素指针的下标法表示*/
  for(j=1;j<=i-1;j++)
     if(temp<b[j]) temp=b[j];
  return temp;
  }
```

在子程序中，形式参数是指针，调用程序的实际参数 ptr为指向一维数组a的指针，虚实结合，子程序的形式参数b得到ptr的值，指向了内存的一维数组。数组元素采用下标法表示，即一维数组的头指针为b，数组元素可以用b[j]表示。其内存中虚实参数的结合如图 6-10所示。

运行程序：

```
RUN
1 3 5 7 9 2 4 6 8 0
max=9
```

[例6-14] 上述程序的子程序中，数组元素还可以用指针表示。

```
# include <stdio.h>
main()
```

```
{
 int sub_max();
 int n,a[10],*ptr=a;
 int max;
  for(n=0;n<=9;n++)
     scanf("%d",&a[n]);
  max=sub_max(ptr,10);
  printf("max=%d\n",max);
  }
 int sub_max(b,i)/* 子程序定义*/
 int *b,i;
 {
  int temp,j;
  temp=*b++;
  for(j=1;j<=i-1;j++)
     if(temp<*b) temp=*b++;
  return temp;
  }
```

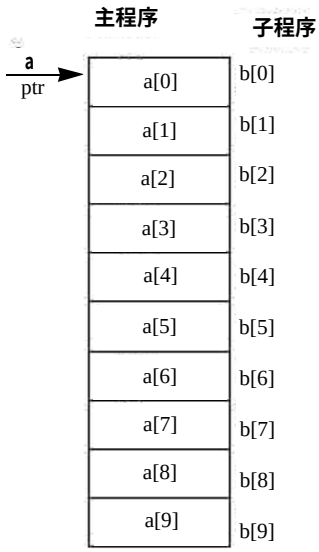

图6-9 例6-12程序在内存中虚实结合示意图

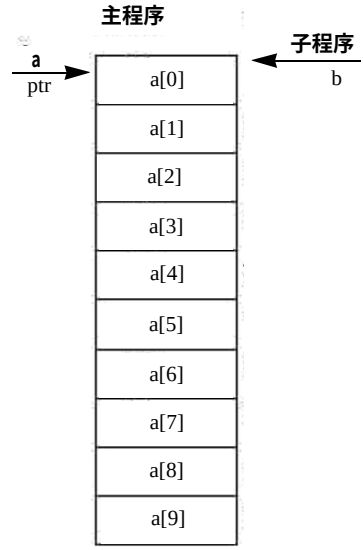

图6-10 例6-13程序在内存中虚实结合示意图

在程序中，赋值语句temp=*b++；可以分解为：temp=*b；b++；两句，先作temp=*b；后作b++；程序的运行结果与上述完全相同。

对上面的程序作修改，在子程序中不仅找最大元素，同时还要将元素的下标记录下来。

```
# include <stdio.h>
main()
{
 int *max();/*   函数声明*/
 int n,a[10],*s,i;
 for(i=0;i<10;i++)/*    输入数据*/
```

```
  scanf("%d",a+i);
s=max(a,10);        /*函数调用*/
printf("max=%d,index=%d\n",*s,s-a);
}
int *max(a,n)   /*定义返回指针的函数*/
int *a,n;
{
int *p,*t;      /*p用于跟踪数组，t用于记录最大值元素的地址*/
for(p=a,t=a;p-a<n;p++)
   if(*p>*t) t=p;
return t;
}
```

在max（）函数中，用p-a<n来控制循环结束，a是数组首地址，p用于跟踪数组元素的地址，p-a正好是所跟踪元素相对数组头的距离，或者说是所跟踪元素相对数组头的元素个数，所以在main()中，最大元素的下标就是该元素的地址与数组头的差，即s-a。运行程序：

```
RUN
1 3 5 7 9 2 4 6 8 0
max=9,index=4
```

[例6-15] 用指向数组的指针变量实现一维数组的由小到大的冒泡排序。编写三个函数用于输入数据、数据排序、数据输出。

在第5章的例题中，我们介绍过选择法排序及算法，此例再介绍冒泡排序算法。为了将一组n个无序的数整理成由小到大的顺序，将其放入一维数组a[0]、a[1]...a[n-1]。冒泡算法如下：（开序）

① 相邻的数组元素依次进行两两比较，即a[0]与a[1]比、a[1]与a[2]比...a[n-2]与a[n-1]比，通过交换保证数组的相邻两个元素前者小，后者大。此次完全的两两比较，能免实现a[n-1]成为数组中最大。

② 余下n-1个元素，按照上述原则进行完全两两比较，使a[n-2]成为余下n-1个元素中最大。

③ 进行共计n-1趟完全的两两比较，使全部数据整理有序。

下面给出一趟排序的处理过程：

原始数据　　3 8 2 5

第一次相邻元素比：　3 8 2 5

第二次相邻元素比：　3 2 8 5

第三次相邻元素比：　3 **2 5 8**

4个元素进行3次两两比较，得到一个最大元素。若相邻元素表示为a[j]和a[j+1]，用指针变量P指向数组，则相邻元素表示为*(P+j)和*(P+j+1)程序实现如下：

```
# include<stdio.h>
#define  N 10
main()
{
 void input();              /*函数声明*/
```

```
 void sort();
 void output();
 int a[N],*p;                    /*定义一维数组和指针变量*/
 input(a,N);                     /*数据输入函数调用，实参a是数组名*/
 p=a;                            /*指针变量指向数组的首地址*/
 sort(p,N);                      /*排序，实参p是指针变量*/
 output(p,N);                    /*输出，实参p是指针变量*/
}
void input(arr,n)                /*无需返回值的输入数据函数定义,形参arr是数组*/
int arr[],n;
{
 int i;
 printf("input data:\n");
 for(i=0;i<n;i++)                /*采用传统的下标法*/
    scanf("%d",&arr[i]);
 }

void sort(ptr,n)                 /*冒泡排序，形参ptr是指针变量*/
int *ptr,n;
{
int i,j,t;
for(i=0;i<n-1;i++)
for(j=0;j<n-1-i;j++)
   if (*(ptr+j)>*(ptr+j+1))相临两个元素进行比较*/
      {
      t=*(ptr+j);      /*两个元素进行交换*/
      *(ptr+j)=*(ptr+j+1);
      *(ptr+j+1)=t;
      }
}
void output(arr,n)               /*数据输出*/
int arr[],n;
{
int i,*ptr=arr;                  /*利用指针指向数组的首地址*/
printf("output data:\n");
for(;ptr-arr<n;ptr++)            /*输出数组的n个元素*/
 printf("%4d",*ptr);
printf("\n");
}
```

运行程序：

```
RUN
3   5   7   9   3   23   43   2   1   10
1   2   3   3   5   7    9  10  23  43
```

由于C程序的函数调用是采用传值调用，即实际参数与形式参数相结合时，实参将值传给形式参数，所以当我们利用函数来处理数组时，如果需要对数组在子程序中修改，只能传递数组的地址，进行传地址的调用，在内存相同的地址区间进行数据的修改。在实际的应用中，如果需要利用子程序对数组进行处理，函数的调用利用指向数组（一维或多维）的指针作参数，无论是实参还是形参共有下面四种情况：

	实 参	形 参
1	数组名	数组名
2	数组名	指针变量
3	指针变量	数组名
4	指针变量	指针变量

在函数的调用时，实参与形参的结合要注意所传递的地址具体指向什么对象，是数组的首址，还是数组元素的地址，这一点很重要。

[例6-16] 用指向二维数组的指针作函数的参数，实现对二维数组的按行相加。

```
# include <stdio.h>
#define M 3
#define N 4
main()
{
 float a[M][N];
 float score1,score2,score3, *pa=a[0]指针变量pa指向二维数组*/
   /* score1,score2,scor分别记录三行的数据相加*/
 int i,j;
 void fun();
 for(i=0;i<M;i++)
  for(j=0;j<N; j++)                    /*二维数组的数据输入*/
     scanf("%f",&a[i][j]);
 fun(pa,&score1,&score2,&score3);
/*函数调用，不仅传递数组首地址，还要传递变量的地址*/
  printf("%.2f,%.2f,%.2f\n",score1,score2,score3);
 }
 void fun(b,p1,p2,p3)
 float b[ ][N],*p1,*p2,*p3;
 {int i,j;
 *p1=*p2=*p3=0;
 for(i=0;i<M;i++)
 for(j=0;j<N;j++)
  {
  if(i==0) *p1=*p1+b[i][j];           /*第0行的数据相加*/
  if(i==1) *p2=*p2+b[i][j];           /*第1行的数据相加*/
  if(i==2) *p3=*p3+b[i][j];           /*第2行的数据相加*/
  }
  }
```

程序中与形式参数p1、p2和p3相对应的是实际参数&score1、&score2和&score3，其实际含义为p1=&score1等，即将变量的地址传递给指针变量达到按行相加。运行程序，

```
RUN
1 2 3 4
3 4 5 6
5 6 7 8
10.00,18.00,26.00
```

[例6-17] 求解二维数组中的最大值及该值在二维数组中的位置。