形参数组与实参数组共占同一段内存单元，在调用函数期间，如果改变了形参数组的值，也就是改变了实参数组的值，在主调函数中可以利用这些已改变的实参数组的值。这种形象化的方法对初学者是有好处的。专业人员往往喜欢用指针变量作形参，程序显得比较专业和高效。

实参与形参的结合，有以下4种形式：

实 参	形 参	
数组名	数组名	（如例5.7）
数组名	指针变量	（如例6.6）
指针变量	数组名	
指针变量	指针变量	

前两种形式已经学过了，请读者把例6.6改写成后两种形式。

在此基础上，还要说明一个问题：实参数组名a代表一个固定的地址，或者说是指针型常量，因此要改变a的值是不可能的。如

```
a++;                                    //语法错误,a是常量,不能改变
```

而形参数组名是指针变量，并不是一个固定的地址值。它的值是可以改变的。在函数调用开始时，它接收了实参数组首元素的地址，但在函数执行期间，它可以再被赋值。如

```
f(array[],int n)
  {cout<<array;                         //输出array[0]的值
   array=array+3;                       //指针变量array的值改变了,指向array[3]
   cout<<*arr<<endl;                    //输出array[3]的值
  }
```

6.3.3 多维数组与指针

用指针变量可以指向一维数组中的元素，也可以指向多维数组中的元素。但在概念上和使用上，多维数组的指针比一维数组的指针要复杂一些。

1. 多维数组元素的地址

设有一个二维数组a，它有3行4列。它的定义为

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,18,21,23}};
```

a是一个数组名。a数组包含3行，即3个元素：a[0]，a[1]，a[2]。而每一元素又是一个一维数组，它包含4个元素（即4个列元素），例如，a[0]所代表的一维数组又包含4个元素：a[0][0]，a[0][1]，a[0][2]，a[0][3]，见图6.14。可以认为二维数组是“数组的数组”，即数组a是由3个一维数组所组成的。

从二维数组的角度来看，a代表二维数组首元素的地址，现在的首元素不是一个整型变量，而是由4个整型元素所组成的一维数组，因此a代表的是首行的起始地址（即第0行的起始地址，&a[0]），a+1代表a[1]行

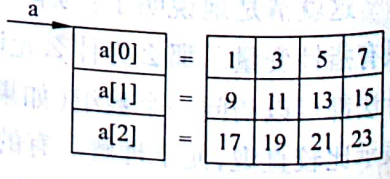

图 6.14

的首地址，即 &a[1]。如果二维数组的首行的起始地址为2000，则 a+1 所代表的地址为2016，因为 a+4×4=2016，a+2 代表 a[2]的起始地址，它的值是2032，见图6.15。

a[0]，a[1]，a[2]既然是一维数组名，而C++又规定了数组名代表数组首元素地址，因此a[0]代表一维数组a[0]中0列元素的地址，即 &a[0][0]。a[1]的值是 &a[1][0]，a[2]的值是 &a[2][0]。

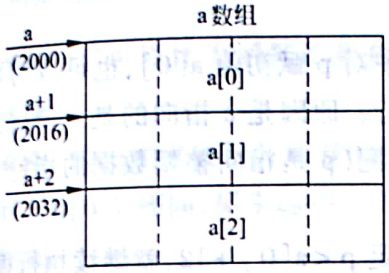

图 6.15

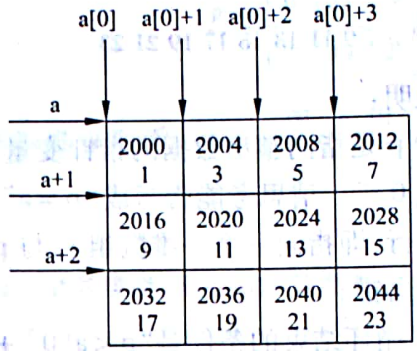

图 6.16

请思考：0行1列元素的地址怎么表示？当然可以直接写为 &a[0][1]，也可以用指针法表示。a[0]为一维数组名，该一维数组中序号为1的元素的地址显然可以用 a[0]+1 来表示，见图6.16。此时“a[0]+1”中的1代表1个列元素的字节数，即4个字节。这里a[0]的值是2000，a[0]+1 的值是2004（而不是2016）。这是因为现在是在一维数组范围内讨论问题的，正如有一个一维数组 arr，arr+1 是其1号元素 arr[1]的地址一样。

由此可知：a[0]+0，a[0]+1，a[0]+2，a[0]+3 分别是 a[0][0]，a[0][1]，a[0][2]，a[0][3]元素的地址（即 &a[0][0]，&a[0][1]，&a[0][2]，&a[0][3]）。如图6.16所示。

进一步分析，欲得到a[0][1]的值，用地址法怎么表示呢？既然 a[0]+1 是a[0][1]元素的地址，那么，*(a[0]+1) 就是a[0][1]元素的值。而a[0]又是和 *(a+0)无条件等价的，因此也可以用 *(*(a+0)+1)表示 a[0][1]元素的值。依此类推，*(a[i]+j)或 *(*(a+i)+j)是a[i][j]的值。

2. 指向多维数组元素的指针变量

在了解上面的概念后，可以用指针变量指向多维数组的元素。

(1) 指向数组元素的指针变量

例6.7 输出二维数组各元素的值。

这里采用的方法是用基类型为整型的指针变量先后指向各元素，逐个输出它们的值。

```
#include <iostream>
using namespace std;
int main( )
{ int a[3][4] = {1,3,5,7,9,11,13,15,17,19,21,23};
  int *p;                          //p是基类型为整型的指针变量
```

```
    for(p=a[0];p<a[0]+12;p++)
      cout<<*p<<" ";
    cout<<endl;
    return 0;
}
```

运行结果如下：

```
1 3 5 7 9 11 13 15 17 19 21 23
```

说明：

① p是指向整型数据的指针变量，在for语句中对p赋初值a[0]，也可以写成“p=&a[0][0]”。请思考能否写成“p=a”？结论是不行。原因是a指向的是a[0]（二维数组的首行，即指向一个一维数组），与p的类型不匹配（p是指向整型数据的指针变量），编译时出错。

② 循环结束的条件是“p<a[0]+12”，只要满足p<a[0]+12，就继续执行循环体。也可以将结束条件写成“p<&a[0][0]+12”，但不能写成“p<a+12”。因为执行“p<a[0]+12”或“p<&a[0][0]+12”时，其中的12代表12个整型元素（48个字节），而“p<a+12”中的12是代表12个一维数组，每个一维数组有4个整型元素（16个字节），a+12表示从数组开头向后移动12×16=192个字节。

③ 执行“cout<<*p;”输出p当前所指的列元素的值，然后执行p++，使p指向下一个列元素。

（2）指向由m个元素组成的一维数组的指针变量

可以定义一个指针变量，它不是指向一个整型元素，而是指向一个包含m个元素的一维数组。这时，如果指针变量p先指向a[0]（即p=&a[0]），则p+1不是指向a[0][1]，而是指向a[1]，p的增值以一维数组的长度为单位，见图6.17。

例6.8 输出二维数组任一行任一列元素的值。

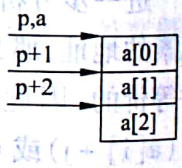

图 6.17

```
#include <iostream>
using namespace std;
int main( )
{ int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int (*p)[4],i,j;
  cin>>i>>j;
  p=a;
  cout<<*(*(p+i)+j)<<endl;
  return 0;
}
```

运行情况如下：

```
2 3↙          （本行为键盘输入，表示要求输出a[2][3]的值）
23            （输出a[2][3]的值23）
```

程序第5行中“int (*p)[4]”表示p是一个指针变量，它指向包含4个整型元素的

一维数组。注意 * p 两侧的括号不可缺少，如果写成 * p[4]，由于方括号[ ]运算级别高，因此 p 先与[4]结合，是数组，然后再与前面的 * 结合，* p[4]是指针数组(见6.7节)。

由于执行了"p = a"，使 p 指向 a[0]。因此 p + 2 是二维数组 a 中序号为 2 的行的起始地址(由于 p 是指向一维数组的指针变量，因此 p 加 1，就指向下一个一维数组)，见图 6.18。 * (p + 2) + 3 是 a 数组 2 行 3 列元素地址。 * (* (p + 2) + 3)是 a[2][3]的值。

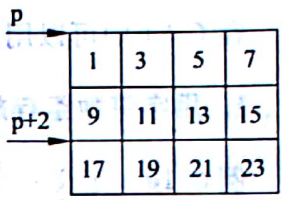

图 6.18

3. 用指向数组的指针作函数参数

一维数组名可以作为函数参数传递，多维数组名也可作函数参数传递。

例 6.9 输出二维数组各元素的值。

题目与例 6.7 相同，但本题用一个函数实现输出，用多维数组名作函数参数。

程序如下：

```
#include <iostream>
using namespace std;
int main( )
{ void output(int ( * p)[4]);              //函数声明
  int a[3][4] = {1,3,5,7,9,11,13,15,17,19,21,23};
  output(a);                               //多维数组名作函数参数
  return 0;
}

void output(int ( * p)[4])                 //形参是指向一维数组的指针变量
{ int i,j;
  for(i = 0;i < 3;i ++ )
    for(j = 0;j < 4;j ++ )
      cout << * ( * (p + i) + j) <<" ";
  cout <<endl;
}
```

运行情况如下：

```
1  3  5  7  9  11  13  15  17  19  21  23
```

a 是二维数组名，a 指向 a[0]。把 a 作为函数 output 的实参，把 a[0]的地址传给形参 p。请注意应当怎样声明形参 p 的类型，"int (* p)[4]"表示指针变量 p 指向的是含有 4 个整型元素的一维数组。不能把 p 声明为指向整型数据的指针变量，如

```
void output(int * p)                       // output 函数首部
```

是不对的，因为 p 指向的类型与实参 a 指向的类型不匹配(实参 a 指向 a[0]，是指向一维数组)，编译时出错。