读者在学习处理二维数组的函数中可能不太理解，为何只把数组的行数作为函数的形参，而列数却内置在函数体内。例如，函数定义如下：

```
#define  COLS  4
int  sum2d(int  ar[][COLS],  int  rows)
{
int  r;
int  c;
int  tot  =  0;
for  (r  =  0;  r  <  rows;  r++)
for  (c  =  0;  c  <  COLS;  c++)
tot  +=  ar[r][c];
return  tot;
}
```

假设声明了下列数组：

```
int  array1[5][4];
int  array2[100][4];
int  array3[2][4];
```

可以用sum2d()函数分别计算这些数组的元素之和：

```
tot = sum2d(array1, 5);      // 5×4 数组的元素之和

tot = sum2d(array2, 100);    // 100×4数组的元素之和

tot = sum2d(array3, 2);      // 2×4数组的元素之和
```

sum2d()函数之所以能处理这些数组，是因为这些数组的列数固定为4，而行数被传递给形参rows，　　rows是一个变量。但是如果要计算6×5的数组（即6行5列），就不能使用这个函数，必须重新创建一个CLOS为5的函数。因为C规定，数组的维数必须是常量，不能用变量来代替COLS。

要创建一个能处理任意大小二维数组的函数，比较繁琐（必须把数组作为一维数组传递，然后让函数计算每行的开始处）。而且，这种方法不好处理FORTRAN的子例程，这些子例程都允许在函数调用中指定两个维度。虽然 FORTRAN 是比较老的编程语言，但是在过去的几十年里，数值计算领域的专家已经用FORTRAN开发出许多有用的计算库。C正逐渐替代FORTRAN，如果能直接转换现有的FORTRAN库就好了。

鉴于此，C99新增了变长数组（variable-length array，VLA），允许使用变量表示数组的维度。如下所示：

```
int quarters = 4;

int regions = 5;

double sales[regions][quarters];      // 一个变长数组（VLA）
```

前面提到过，变长数组有一些限制。变长数组必须是自动存储类别，这意味着无论在函数中声明还是作为函数形参声明，都不能使用static或extern存储类别说明符（第12章介绍）。而且，不能在声明中初始化它们。最终，C11把变长数组作为一个可选特性，而不是必须强制实现的特性。

注意 变长数组不能改变大小

变长数组中的“变”不是指可以修改已创建数组的大小。一旦创建了变长

数组，它的大小则保持不变。这里的“变”指的是：在创建数组时，可以使用变量指定数组的维度。

由于变长数组是C语言的新特性，目前完全支持这一特性的编译器不多。下面我们来看一个简单的例子：如何编写一个函数，计算int的二维数组所有元素之和。

首先，要声明一个带二维变长数组参数的函数，如下所示：

```
int sum2d(int rows, int cols, int ar[rows][cols]); // ar是一个变长数组（VLA）
```

注意前两个形参（rows和cols）用作第3个形参二维数组ar的两个维度。因为ar的声明要使用rows和cols，所以在形参列表中必须在声明ar之前先声明这两个形参。因此，下面的原型是错误的：

```
int sum2d(int ar[rows][cols], int rows, int cols); // 无效的顺序
```

C99/C11标准规定，可以省略原型中的形参名，但是在这种情况下，必须用星号来代替省略的维度：

```
int sum2d(int, int, int ar[*][*]); // ar是一个变长数组（VLA），省略了维度形参名
```

其次，该函数的定义如下：

```
int sum2d(int rows, int cols, int ar[rows][cols])
{
int r;
int c;
int tot = 0;
```

```
for (r = 0; r < rows; r++)
for (c = 0; c < cols; c++)
tot += ar[r][c];
return tot;
}
```

该函数除函数头与传统的C函数（程序清单10.17）不同外，还把符号常量COLS替换成变量cols。这是因为在函数头中使用了变长数组。由于用变量代表行数和列数，所以新的sum2d()现在可以处理任意大小的二维int数组，如程序清单10.18所示。但是，该程序要求编译器支持变长数组特性。另外，该程序还演示了以变长数组作为形参的函数既可处理传统C数组，也可处理变长数组。

程序清单**10.18 vararr2d.c**程序

```
//vararr2d.c -- 使用变长数组的函数
#include <stdio.h>
#define ROWS 3
#define COLS 4
int sum2d(int rows, int cols, int ar[rows][cols]);
int main(void)
{
int i, j;
int rs = 3;
```

```
int cs = 10;
int junk[ROWS][COLS] = {
{ 2, 4, 6, 8 },
{ 3, 5, 7, 9 },
{ 12, 10, 8, 6 }
};
int morejunk[ROWS - 1][COLS + 2] = {
{ 20, 30, 40, 50, 60, 70 },
{ 5, 6, 7, 8, 9, 10 }
};
int varr[rs][cs]; // 变长数组（VLA）
for (i = 0; i < rs; i++)
for (j = 0; j < cs; j++)
varr[i][j] = i * j + j;
printf("3x5 array\n");
printf("Sum of all elements = %d\n", sum2d(ROWS, COLS, junk));
printf("2x6 array\n");
printf("Sum of all elements = %d\n", sum2d(ROWS - 1, COLS + 2, morejunk));
```

```
    printf("3x10  VLA\n");
    printf("Sum  of  all  elements  =  %d\n",  sum2d(rs,  cs,  varr));
    return  0;
    }
    // 带变长数组形参的函数
int  sum2d(int  rows,  int  cols,  int  ar[rows][cols])
{
int  r;
int  c;
int  tot  =  0;
for  (r  =  0;  r  <  rows;  r++)
for  (c  =  0;  c  <  cols;  c++)
tot  +=  ar[r][c];
return  tot;
}
```

下面是该程序的输出：

```
3x5  array
Sum  of  all  elements  =  80
2x6  array
```

Sum of all elements = 315

3x10 VLA

Sum of all elements = 270

需要注意的是，在函数定义的形参列表中声明的变长数组并未实际创建数组。和传统的语法类似，变长数组名实际上是一个指针。这说明带变长数组形参的函数实际上是在原始数组中处理数组，因此可以修改传入的数组。下面的代码段指出指针和实际数组是何时声明的：

```
int thing[10][6];
twoset(10,6,thing);
...
}
int thing[10][6];
twoset(10,6,thing);
...
}
void twoset (int n, int m, int ar[n][m]) // ar是一个指向数组（内含m个int类型的值）的指针
{
int temp[n][m];      // temp是一个n×m的int数组
temp[0][0] = 2;      // 设置temp的一个元素为2
ar[0][0] = 2;        // 设置thing[0][0]为2
}
```

如上代码所示调用twoset()时，ar成为指向thing[0]的指针，temp被创建为10×6的数组。因为ar和thing都是指向thing[0]的指针，ar[0][0]与thing[0][0]访问的数据位置相同。

const和数组大小

是否可以在声明数组时使用const变量？

```
const  int  SZ  =  80;
...
double ar[SZ]; // 是否允许？
```

C90标准不允许（也可能允许）。数组的大小必须是给定的整型常量表达式，可以是整型常量组合，如20、sizeof表达式或其他不是const的内容。由于C实现可以扩大整型常量表达式的范围，所以可能会允许使用const，但是这种代码可能无法移植。

C99/C11 标准允许在声明变长数组时使用 const 变量。所以该数组的定义必须是声明在块中的自动存储类别数组。

变长数组还允许动态内存分配，这说明可以在程序运行时指定数组的大小。普通 C数组都是静态内存分配，即在编译时确定数组的大小。由于数组大小是常量，所以编译器在编译时就知道了。第12章将详细介绍动态内存分配。