10.4 向函数传递一个一维数组

在C语言中，任何一维数组均可以作为函数的实参。形参被改写为指向数组第一个元素的指针，所以需要一个约定来提示数组的长度。一般有两个基本方法：

- 增加一个额外的参数，表示元素的数目（argc就是起这个作用）。
- 赋予数组最后一个元素一个特殊的值，提示它是数组的尾部（字符串结尾的'\0'字符就是起这个作用）。这个特殊值必须不会作为正常的元素值在数组中出现。

二维数组的情况要复杂一些，数组被改写为指向数组第一行的指针。现在需要两个约定，其中一个用于提示每行的结束，另一个用于提示所有行的结束。提示单行结束可以使用一维数组所用的两种方法，提示所有行结束也可以这样。我们所接收的是一个指向数组第一个元素的指针。每次当对指针执行自增操作时，指针就指向数组中下一行的起始位置，但怎么知道指针到达了数组的最后一行呢？我们可以增加一个额外的行，行内所有元素的值都是不可能在数组正常元素中出现的，能够提示数组超出了范围。当对指针进行自增操作时，要对它进行检查，看看它是否到达了那一行。另一种方法是，定义一个额外的参数，提示数组的行数。

10.5 使用指针向函数传递一个多维数组

使用上一节所描述的笨拙方法，可以解决标记数组范围这个难题。但是还存在一个问题，就是如何在函数内部声明一个二维数组参数，这才是真正的麻烦所在。C语言没有办法表达"这个数组的边界在不同的调用中可以变化"这个概念。C编译器必须要知道数组的边界，以便为下标引用产生正确的代码。从技术上说，也可以在运行时处理才知道数组的边界，而且很多其他语言就是这样做的，但这种做法违背了C语言的设计理念。

我们能够采取的最好方法就是放弃传递二维数组，把array[x][y]这样的形式改写为一个一维数组array[x+1]，它的元素类型是指向array[y]的指针。这样就改变了问题的性质，而改变后的问题是我们已经解决了的。在数组最后的那个元素array[x+1]里存储一个NULL指针，提示数组的结束。

在C语言中，没有办法向函数传递一个普通的多维数组

这是因为我们需要知道每一维的长度，以便为地址运算提供正确的单位长度。在C语言中，我们没有办法在实参和形参之间交流这种数据（它在每次调用时会改变）。因此，你必须提供除了最左边一维以外的所有维的长度。这样就把实参限制为除最左边一维外所有维都必须与形参匹配的数组。

```
invert_in_place(int a[][3][5]);
```

用下面两种方法调用都可以：

```
int b[10][3][5];  invert_in_place(b);
int b[999][3][5]; invert_in_place(b);
```

但像下面这样任意的三维数组：

```
int fails1[10][5][5];  invert_in_place(fails1); /* 无法通过编译 */
int fails2[999][3][6]; invert_in_place(fails2); /* 无法通过编译 */
```

却是无法通过编译器这一关的。

二维或更多维的数组无法在C语言中用作一般形式的参数。你无法向函数传递一个普通的多维数组。可以向函数传递预先确定长度的特殊数组，但这个方法并不能满足一般情况。最显而易见的方法是声明一个像下面这样的原型：

10.5.1 方法 1

```
my_function(int my_array[10][20]);
```

尽管这是最简单的方法，但同时也是作用最小的。因为它迫使函数只处理 10 行 20 列的 int 型数组。我们想要的是一个确定更为普通的多维数组形参的方法，使函数能够操作任意长度的数组。注意，多维数组最主要的一维的长度（最左边一维）不必显式写明。所有的函数都必须知道数组其他维的确切长度和数组的基地址。有了这些信息，它就可以一次“跳过”一个完整的行，到达下一行。

10.5.2 方法 2

我们可以合法地省略第一维的长度，像下面这样声明多维数组：

```
my_function(int my_array[][20]);
```

但这样做法仍不够充分，因为每一行都必须正好是 20 个整数的长度。函数也可以类似地声明为：

```
my_function(int (*my_array)[20]);
```

参数列表中(* my_array)周围的括号是绝对需要的，这样可以确保它被翻译为一个指向 20 个元素的 int 数组的指针，而不是一个 20 个 int 指针元素的数组。同样，我们对最右边一维的长度必须为 20 感觉不快。

10.5.3 方法 3

我们可以采取的第三种方法是放弃二维数组，把它的结构改为一个 Iliffe 向量。也就是说，创建一个一维数组，数组中的元素是指向其他东西的指针。回想一下 main()函数的两个参数，我们已经习惯了看到 char * argv[];的形式，有时也能看到 char ** argv;这样的形式，它能提醒我们怎样分析这个声明。可以简单地传递一个指向数组参数的第一个元素的指针，如下所示（用于二维数组）：

```
my_function(char **my_array);
```

注意：只有把二维数组改为一个指向向量的指针数组的前提下才可以这样做！

Iliffe 向量这种数据结构的美感在于：它允许任意的字符串指针数组传递给函数，但必须是指针数组，而且必须是指向字符串的指针数组。这是因为字符串和指针都有一个显式的越界值（分别为 NUL 和 NULL），可以作为结束标记。至于其他类型，并没有一种类似的通用且可靠的值，所以并没有一种内置的方法知道何时到达数组某一维的结束位置。即使是指向字符串的指针数组，通常也需要一个计数参数 argc，记录字符串的数量。

10.5.4 方法 4

我们可以采取的最后一种方法也是放弃多维数组的形式，提供自己的下标方式。当 Groucho Marx 评论“如果你把酸果蔓煮成苹果酱那样，它们尝起来会比大黄更像李子”时，他脑子里想的肯定就是这种错综复杂的迂回方法。

```
char_array[row_size * i + j] = ...
```

这很容易误入歧途，而且会让你困惑，如果可以手工做这些事情，为什么还需要使用编译器呢？

总之，如果多维数组各维的长度都是一个完全相同的固定值，那么把它传递给一个函数毫无问题。如果情况更普通一些，也更常见一些，就是作为函数的参数的数组的长度是任意的，我们用下面的方法进行进一步的分析：

- 一维数组——没有问题，但需要包括一个计数值或者是一个能够标识越界位置的结束符。被调用的函数无法检测数组参数的边界。正因为如此，gets()函数存在安全漏洞，从而导致了 Internet 蠕虫的产生。
- 二维数组——不能直接传递给函数，但可以把矩阵改写为一个一维的 Iliffe 向量，并使用相同的下标表示方法。对于字符串来说，这样做是可以的，对于其他类型，需要增加一个记数值或者能够标识越界位置的结束符。同样，它依赖于调用函数和被调用函数之间的约定。
- 三维或更多维的数组—— 都无法使用。必须把它分解为几个维数更少的数组。

对多维数组作为参数传递的支持缺乏是 C 语言存在的一个内在限制。这使得用 C 语言编写某些特定类型的程序非常困难（如数值分析算法）。