数组和指针绝不等价，数组是另外一种类型

通过前面的讲解，相信很多读者都会认为数组和指针是等价的，数组名表示数组的首地址。不幸的是，这是一种非常危险的想法，并不完全正确，前面我们将数组和指针等价起来是为了方便大家理解（在大多数情况下数组名确实可以当做指针使用），不至于被指针难倒，这节请大家放弃这种观念，我将会颠覆你的认知。

数组和指针不等价的一个典型案例就是求数组的长度，这个时候只能使用数组名，不能使用数组指针，前面我们已经强调过了，这里不妨再来演示一下：

```
#include <stdio.h>

int main(){
    int a[6] = {0, 1, 2, 3, 4, 5};
    int *p = a;
    int len_a = sizeof(a) / sizeof(int);
    int len_p = sizeof(p) / sizeof(int);
    printf("len_a = %d, len_p = %d\n", len_a, len_p);
    return 0;
}
```

运行结果：

```
len_a = 6, len_p = 1
```

数组是一系列数据的集合，没有开始和结束标志，p 仅仅是一个指向 int 类型的指针，编译器不知道它指向的是一个整数还是一堆整数，对 p 使用 sizeof 求得的是指针变量本身的长度。也就是说，编译器并没有把 p 和数组关联起来，p 仅仅是一个指针变量，不管它指向哪里，sizeof 求得的永远是它本身所占用的字节数。

站在编译器的角度讲，变量名、数组名都是一种符号，它们最终都要和数据绑定起来。变量名用来指代一份数据，数组名用来指代一组数据（数据集合），它们都是有类型的，以

便推断出所指代的数据的长度。

对，数组也有类型，这是很多读者没有意识到的，大部分 C 语言书籍对这一点也含糊其辞！我们可以将 int、float、char 等理解为基本类型，将数组理解为由基本类型派生得到的稍微复杂一些的类型。sizeof 就是根据符号的类型来计算长度的。

对于数组 a，它的类型是 `int [6]`，表示这是一个拥有 6 个 int 数据的集合，1 个 int 的长度为 4，6 个 int 的长度为 4×6 = 24，sizeof 很容易求得。

对于指针变量 p，它的类型是 `int *`，在 32 位环境下长度为 4，在 64 位环境下长度为 8。

归根结底，a 和 p 这两个符号的类型不同，指代的数据也不同，它们不是一码事，sizeof 是根据符号类型来求长度的，a 和 p 的类型不同，求得的长度自然也不一样。

对于二维数组，也是类似的道理，例如 `int a[3][3]={1, 2, 3, 4, 5, 6, 7, 8, 9};`，它的类型是 `int [3][3]`，长度是 4×3×3 = 36，读者可以亲自测试。

站在哲学的高度看问题

编程语言的目的是为了将计算机指令（机器语言）抽象成人类能够理解的自然语言，让程序员能够更加容易地管理和操作各种计算机资源，这些计算机资源最终表现为编程语言中的各种符号和语法规则。

整数、小数、数组、指针等不同类型的数据都是对内存的抽象，它们的名字用来指代不同的内存块，程序员在编码过程中不需要直接面对内存，使用这些名字将更加方便。

编译器在编译过程中会创建一张专门的表格用来保存名字以及名字对应的数据类型、地址、作用域等信息，sizeof 是一个操作符，不是函数，使用 sizeof 时可以从这张表格中查询到符号的长度。

与普通变量名相比，数组名既有一般性也有特殊性：一般性表现在数组名也用来指代特定的内存块，也有类型和长度；特殊性表现在数组名有时候会转换为一个指针，而不是它所指代的数据本身的值。