一、前言

在 C/C++ 语言中，数组类型的变量是不可以直接赋值的。但是如果把数组放在结构体中，然后对结构体变量进行赋值，就可以实现把其中的数组内容进行复制过去。

很多朋友对这个不是特别理解，只是强制记忆，下面我尝试用自己的理解来描述一下，希望对你有所帮助！

二、数组的各种操作

1. 错误代码

```
int a[5] = {1, 2, 3, 4, 5};
int b[5];
b = a;
```

对于上面的赋值语句，编译器会报错 `error: assignment to expression with array type`，即：不能对一个数组类型的变量进行赋值。

那么编译器此时是如何来解释 a 和 b 的？下面会说到这个问题。

> 有一个地方提一下：第一条语句中的 = 操作，不是赋值，而是初始化。
> C/C++ 语法规定在定义变量的时候，是可以使用 操作符 = 来进行初始化操作的。

2. 利用结构体来复制数组

```
typedef struct {
    int arr[5];
} array_wrap;

array_wrap a = {{1, 2, 3, 4, 5}};
array_wrap b;
b = a;
```

这里的赋值操作是针对结构体变量，C 语言标准允许这种行为，是合法的，变量 a 中的所有内容(也就是这个变量占用过的那一块内存空间中的内容)会原样的复制到变量 b 中。

3. 其他复制方式

既然不能直接对数组类型的变量进行赋值，只能寻求其他的替代方式，例如：

1. 利用 memcpy(b, a, sizeof(int) * 5); 复制一整段内存空间中的内容；
2. 利用 for/while 等循环语句，逐个复制数组中每一个元素: b[i] = a[i];

三、语言标准和编译器

C/C++ 只是一门高级语言，是被标准委员会从无到有设计出来的，因此我们编程时需要严格遵守这些规则。

这些规则中，就包括这么一条：只有标量和结构体，才能出现在赋值操作符=的左侧。

但是数组类型并不是一个标量，因此不能对结构体执行赋值操作。

理论上，如果 C/C++ 语言愿意的话，是"可以"对数组直接赋值的(那就要修改语法标准)，只不过标准委员会在经过各种场景的权衡利弊之后，做出了目前这样的规定，这是对各种考虑到的因素进行权衡之后的结果。

也就是说，目前标准中对于数组操作的方式，是利大于弊。

既然标准已经是制定成这样的了，我们就来分析一下编译器是如何来遵循、实现这个标准的。

1. 数组与指针的暧昧关系

很多人都这样记忆：数组名就是数组开始地址的指针。这是不对的，或者说不严谨的。

在 C/C++ 中，数组就是数组，指针就是指针。数组在内存中有确定的空间(每个元素的大小 x 元素个数)。

只不过在表达式中，数组名会“临时的”表示数组中第一个元素的常量指针(前提条件：在没有操作符 sizeof 和 & 的情况下)。

对于下面这段代码，打印结果是相同的：

```
int a[5] = {1, 2, 3, 4, 5};
printf("a = %p \n", a);
printf("&a = %p \n", &a);
```

第一个 printf 中，a 会“临时的”代表指向第一个元素的常量指针。

第二个 printf 中，a 就表示一个数组，与指针没有半毛钱的关系，前面加上取地址符 &，就表示获取这个数组所在的地址，这个地址与第一个元素的地址是重合的。

注意：代码在被编译成二进制文件之后，没有任何变量的概念，全部是用地址来“传递” C/C++ 代码中的变量。

2. 为什么不能对数组变量赋值

有了上面的基础理解就好办了，对于下面的这段代码：

```
int a[5] = {1, 2, 3, 4, 5};
int b[5];
b = a;
```

在赋值语句 `b = a` 中，左侧的 b 是一个数组类型，右侧的 a 被编译器“临时的”代表第一个元素的常量指针，但是数组不是一个标量，不可以放在赋值运算符=的左侧，因此编译器就抱怨：非法！

既然在一个表达式中，数组名被临时的表示第一个元素的常量指针，那么就说明我们不能对数组名本身进行计算，例如：不能进行 a++, a-- 等操作。

例如：下面这的遍历方式是非法的：

```
int a[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++)
{
    // 常量指针，不可以进行递增操作
    printf("a[%d] = %d \n", i, *a++);
}
```

3. 函数形参是数组的情况

考虑下面这个函数：

```
void func(int arr[5])
{
  for (int i = 0; i < 5; ++i)
  {
    printf(*arr++);  // 合法!
  }
}
```

形参 arr 在形式上好像是一个数组，实际上被编译器当做指针，也就是相当于： `void func(int *arr)`，因此，在 printf 打印语句中，可以对 arr 进行递增操作。

PS: 这种场景下都需要额外的传递一个参数，来告知元素的个数。

调用这个函数的代码如下：

```
int a[5] = {1, 2, 3, 4, 5};
fun(a);
```

数组名临时代表第一个元素的常量指针，在传参的时候，形参 arr 的值就是数组中第一个元素的内存地址。

4. 为什么结构体中的数组可以复制

有了前面的语法标准，这个问题似乎不用再讨论了~~

赋值的目的是什么？就是让一块内存空间的内容，与另一块内存空间中的内容完全相同。如果想要完成复制操作，那么就需要知道这块内存空间的大小。

编译器是知道一个结构体变量所占用的空间大小的，所以当复制的时候，类似于 memcpy 一样，把一个结构体变量所占空间按照 byte to byte 的方式复制过去。

5. 参数传递和返回值

> 1. 在调用函数时，实参到形参的传递；
> 2. 函数执行结束后的返回值；

这两个场景中都涉及到变量的赋值问题。

关于参数传递，上面已经说了：编译器是把形参当做普通的指针类型的。

对于函数返回值来说，同样的道理，也不能直接返回一个数组，因为它仅仅是临时性的代表第一个元素的常量指针。

当然，可以利用结构体的可赋值特性，把数组包裹在其中，以此达到复制的效果。

五、总结

记住这两句话：

1.数组就是数组，指针就是指针，它们各不相干。

2.在表达式中，数组名会“临时的”表示数组中第一个元素的常量指针(前提条件：在没有操作符 sizeof 和 & 的情况下)

推荐阅读

1. C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
2. 原来gdb的底层调试原理这么简单
3. 一步步分析-如何用C实现面向对象编程
4. 都说软件架构要分层、分模块，具体应该怎么做(一)
5. 都说软件架构要分层、分模块，具体应该怎么做(二)