

08 | 指针系列（二）：记住，指针变量也是变量

2020-01-23 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 16:45 大小 13.44M



你好，我是胡光，咱们又见面了，上节课中，我们介绍了结构体相关的基础知识，也介绍了指针变量，并且教给你了最重要的一句话“指针变量也是变量”。这句话的意思在于告诉你，所有你对变量的理解，都可以放到指针变量上，例如：变量有类型，变量有大小，变量里面的值支持某些操作等等。今天呢，我们就来详细地聊一下指针变量。

任务回顾

在正式开始之前，我们先来回顾一下上节课的任务内容：



上节课我们说，如果给我们如下 Data 结构体类型，这个类型中有两个整型数据字段 x，y：

[复制代码](#)

```
1 struct Data {  
2     int x, y;  
3 } a[2];
```

那么请用尽可能多得形式，替换下面代码中 `&a[1].x` 的部分，使得代码效果不变：

[复制代码](#)

```
1 struct Data *p = a;  
2 printf("%p", &a[1].x);
```

你会看到，如上代码中，就是输出 `a[1].x` 的地址值。

通过上节的学习，你现在已经掌握了关于结构体的相关知识，也初步地接触了“指针变量也是变量”的这个概念，今天就让我们再深入了解指针变量吧。

必知必会，查缺补漏

1. 深入理解：指针变量的类型

还记得我们是如何定义 `p` 变量的么？代码语句是：

[复制代码](#)

```
1 int *p
```

之前我们介绍了，语句中的 `*` 代表 `p` 变量是一个指针变量，而 `int` 的作用是什么呢？只是用来说明 `p` 是一个指向整型存储区的指针变量么？其实 `int` 更大的作用，就是用来解决我们上面提到的那个问题，根据 `p` 变量中的内容，我们可以找到一个存储区的首地址，然后再根据 `p` 的类型，就可以确定要取几个字节中的内容了。

下面给你举个例子：

[复制代码](#)

```
1 int a = 0x61626364;  
2 int *p = &a;  
3 char *q = (char *)&a;
```

```
4 printf("%x %c\n", *p, *q);
```

这段上面代码中，p 和 q 同时指向了 a 变量的存储区。而取值 p 和取值 q 的结果，却截然不同。这是因为，取值 p 时，程序会从 p 所指向的首地址开始，取 4 个字节的内容作为数据内容进行解析，而取值 q 的时候，则是取 1 个字节的内容，作为数据内容进行解析。

你如果运行上述代码，大概率你会看到输出内容是：

```
1 61626364 d
```

复制代码

小概率会看到输出内容是：

```
1 61626364 a
```

复制代码

这个原因和“大端机”“小端机”有关，关于这个问题，你要是有兴趣的话，可以自行查阅相关资料。下面的图中呢，就是以“小端机”为例，说明的 p 和 q 取值的问题：

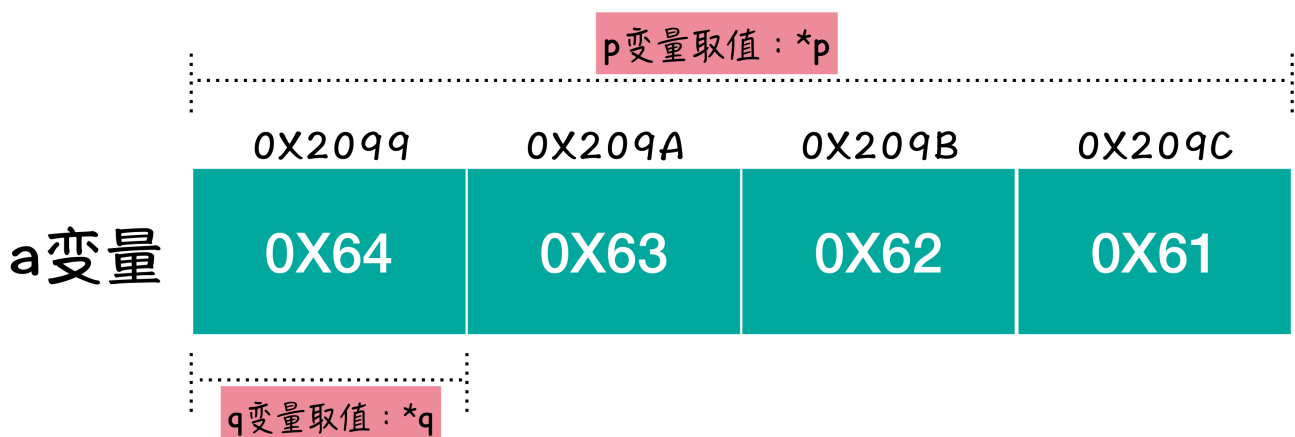


图1：指针变量取值示意图

如图所示，p 变量对应了 a 变量整个存储区中的内容，所以输出取值 p 和 a 原本存储内容相同。而 q 变量由于是字符型指针变量，只能从首地址取到 1 个字节的内容，取到的就是 64，这里的 64 注意可是 16 进制的数字，对应到 10 进制数字就是 100，而 %c 是输出一个字符，数字 100 对应的字符就是英文小写字母 ‘d’ 。

实际上，我们看到的任何字符，在底层都对应了一个具体的数字。常用的有字符 'a'，对应的是 97，字符 'b'，对应的是 98，以此类推，还有数字 '0' 是 48，数字 '1' 是 49，后面的对应规律类似，我们管这个对应规则叫做 ASCII 编码。

指针变量的类型，除了用来确定取值时，确定覆盖存储区的大小以外，还有其他作用。想一想，整型支持加减乘除操作，而我们所谓的地址类型的值，也可以在其上面做加减的操作，你可以试着运行下面的代码：

复制代码

```
1 int a, *p = &a;
2 char *q = &a;
3 printf("%p %p", p, q);
4 printf("%p %p", p + 1, q + 1);
```

代码中，定义了三个变量，其中一个整型变量 a，两个指针变量 p 和 q，其中 p 是整型指针变量，q 是字符型指针变量。然后分别输出 p 和 q，以及 p + 1 和 q + 1 的值以作对比。

如果你运行上面的程序，你会看到，p 和 q 的值是相同的，都是 a 变量的首地址，但是 p + 1 和 q + 1 的值却不同。如果你仔细观察会发现，p + 1 的地址值与 a 的地址之间差了 4 个字节，而 q + 1 的地址值与 a 的地址之间只差了 1 个字节。

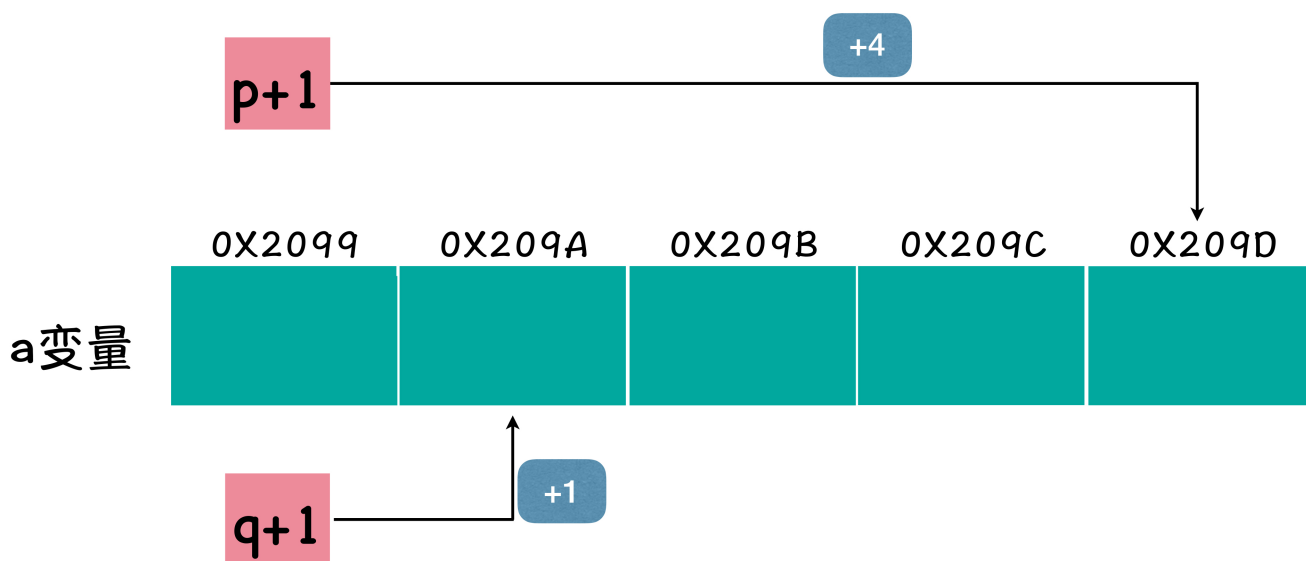


图2：地址加法操作结果

通过上图，你就可以更清晰的看到，由于 p 是整型指针，所以 $p + 1$ 的计算结果，是向后跳了一个整型，相当于从第一个整型的首地址，跳到第二个整型的首地址；而由于 q 是字符型指针，所以 $q + 1$ 的计算结果，就是向后跳了一个字符型。

这样，你就可以明白了吧？如果一个浮点型的指针变量加 1，就会向后跳一个浮点型。这就是**指针变量类型的第二个作用：在加法或者减法时，确定增加或者减少的地址长度。**

2. 指针变量与数组

理解了指针类型的作用以后，我们再回到“指针变量也是变量”这句话上，指针变量所存储的值，就是地址。在之前的学习中，还有什么与地址相关的概念呢？你一定会想起数组这个概念。对，数组名代表了数组中第一个元素的首地址，也是整个数组的首地址，既然是地址，那就可以用指针变量来存储。

下面，我就跟你说几个之前没有告诉你，但却很有趣的事情。

假设有一个整型数组 arr ，如何表示第二个元素的地址呢？是不是 $\&arr[1]$ ？如果 arr 也代表了整个数组的首地址，同时把这个首地址存储在一个整型指针变量 p 中，那么用这个指针变量如何表示第二个元素的地址呢？

根据上面的学习，应该是 $p + 1$ 。那如何表示 $arr[n]$ 元素的地址呢？稍加思索，你就应该知道就是 $p + n$ 。所以我们现在知道了，在程序中， $\&arr[n]$ 等价于 $p + n$ ，当然也等价于 $arr + n$ ，聪明的你别犯糊涂，一定要注意，参与运算的是值，不是变量名！


既然 p 中存储了一个地址，可以参与加法运算，那么 arr 实际上也代表了一个地址，也可以参与加法运算。地址才是参与运算的值，指针只是存储地址值的变量，只是一个容器。所以，不是指针支持加减法操作，而是地址这种类型的值，支持加减法操作。

在这里，我们回头看数组名称后面的那对方括号，如果我告诉你这也是一个运算符，你会想到什么？请注意认真看下面这一段合理化的猜想推理：

如果那对方括号代表了运算符，而运算符本质上是作用在值上面，也就是说，当我们写 $arr[1]$ 的时候，方括号运算符前面看似放着一个数组名，实际上放了一个地址，放了一个数组的首地址，因为 arr 就是数组的首地址，还是那句话：地址才是参与运算的值。也就是

说，当我们把数组的首地址，存储在一个指针变量中以后，这个指针变量配合上方括号运算符，也可以达到相同的效果！

为了让你更清楚的理解，准备了如下演示代码：

 复制代码

```
1 int arr[100] = {1, 2, 3, 4};  
2 int *p = arr;  
3 printf("%d %d\n", arr[1], p[1]);
```

代码中，我们定义了一个整型数组 `arr`，然后将数组的首地址赋值给了一个整型指针变量 `p`，最后分别输出 `arr[1]` 和 `p[1]` 的值，你将看到输出的是同一个值，都是数组中第二个元素的值。

最后，我用一张图给你展示了指针与数组的几个程序代码层面的等价关系，在实际编程过程中，重点是需要分析，相关的指针操作后，对应的到底是哪个元素，对应的是这个元素的首地址，还是这个元素的值。

`int arr[100], *p=arr;`

①: **`p`**  **`arr`**

②: **`p[1]`**  **`arr[1]`**

③: **`p + 5`**  **`arr + 5`**

图3：指针与数组的等价表示

从上图的等价表示中，你可能会自己推导出另外一种等价表示 $*(p + 5)$ 等于 `arr[5]`。我希望你重视等价表示的学习，因为所谓等价表示，就是在写程序的时候，多种等价表示，写哪一种都一样。这就造成了，不同的编码习惯，会用不同的符号来完成程序，如果你不理解这些等价的表示方法，很有可能在看别人程序的过程中，就会出现看不懂的现象。

3. 指针变量的大小

最后，我们再回到“指针变量也是变量”这句话上。只要是变量，就占据一定的存储空间，那一个指针变量占多少个字节的存储空间呢？

在回答这个问题之前，我先问你另一个问题，请你思考一下：是整型指针变量占用的存储空间大，还是字符型指针变量占用的存储空间大？我们想想啊，一种数据类型占用多少存储空间跟什么有关系？和存储的值有关系啊。当你想存储一个 32 位整数的时候，就必须要用 4 个字节，不能用 2 个字节，也不能用 3 个字节，这都是不够的。

究竟是哪一种类型的指针占的存储空间大呢？答案是：一样大。为什么呢？就是因为，无论是什么类型的指针，存储的值都是某个字节的地址，而在一个系统中，无论是哪个字节的地址，二进制数据长度都是一样的。所以，无论什么类型的指针，所需要存储的值的底层表示长度是一样的，那么所占用的存储空间也当然是一样的了！

有句话描述的非常形象“类型就是指针变量的职业”。什么意思呢？我们知道现实生活中，有些人做保安，有些人做工程师，还有些人当艺术家，可不管你做什么，你无法改变的是你作为人的生理结构。所以放到指针变量的概念里，那就是不管什么类型的指针，指针所改变不了的是其占用空间的存储大小，因为不管是什么类型的指针，存储的都是无差别的地址信息。

任务参考答案

至此，我们终于准备完了所有的基础知识，下面就让我们回到最开始的那个任务吧。对于这个任务，如果我们要是想写的话，至少能写出 20 种以上的答案。这里，我会选出两种比较有代表性的、比较有趣的做法分享给你。

1. 间接引用

首先来看第一种：


```
1 struct Data *p = a;  
2 printf("%p", &((a + 1)->x));
```

这里用到了一个之前提到过，可是没有讲到的运算符，减号大于号（->），组合起来，我们叫做“间接引用”运算符，作用可以和“直接引用”运算符对比。

例如：a 是一个结构体变量，a 中有一个字段叫做 x，由 a 去找到 x，这个过程比较直接，我们就用 a.x 来表示。可如果 p 是一个指针，指向 a 变量，如果要是由 p 去找到 x，这个过程就是个间接的过程，所以我们就使用 p->x。简单来说，就是：是结构体变量引用字段，就直接引用，如果是指针想引用字段，就是间接引用。

在这个第一种做法中，直接用 a + 1 定位到第二个结构体元素的首地址，然后间接引用 x 字段，最后再对 x 字段取地址，那么得到的和原任务中所输出的地址是一样的。

2. 巧妙使用指针类型

再来看一下第二种：

```
1 struct Data *p = a;  
2 printf("%p", &(a[0].y) + 1);
```

这个第二种做法就有点儿意思了。首先，它先定位到 a[0] 元素中 y 字段的首地址，然后对 y 字段取地址，这个时候，由于 y 字段是整型，所以取到的地址类型就是整型地址，之后再对这个整型地址执行 +1 操作，得到的也是 a[1].x 的首地址。

按照之前所学，画出内存中的存储示意图，你就会得到下面这张图的具体情况：

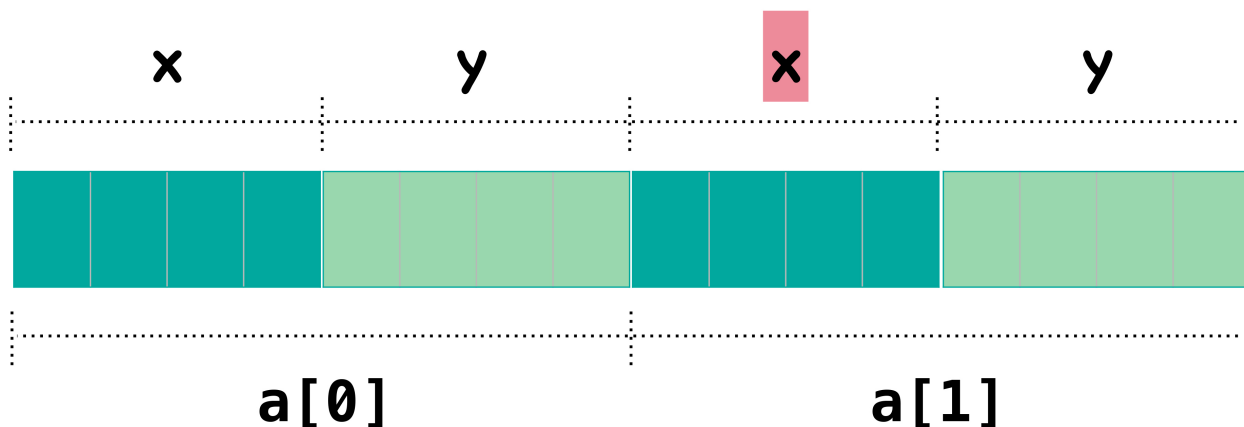


图4: a数组内存结构示意图

第二种方法巧妙的利用了地址类型这个知识点，通过整型地址加法操作结合对于内存存储结构的知识，综合运用以上两个知识点，最终定位 `a[1].x` 变量的地址。如果你可以独立想出这个方案，那我真的是要给你点赞的！

上面的方案中，都在用原数组 `a` 去定位 `a[1].x` 变量的地址，你可以使用 `p` 指针，完成相同的操作么？欢迎把你的答案写在留言区，让我也欣赏一下你的思维方式。记住，这个问题，至少能写出来 20 种以上的等价表示形式。

课程小结

今天我们终于讲完了指针部分，这一部分的知识，再回过头来看，虽然各种各样的知识点，可我想让你记住的还是那一句话：“指针变量也是变量”。

而在今天的学习中，我希望你记住的重点，有以下三点：

1. 指针的类型，决定了指针取值时所取的字节数量。
2. 指针的类型，决定了指针加减法过程中，所跨越的字节数量。
3. 无论是什么类型的指针，大小都相等，因为地址信息是统一规格的。

好了，今天就到了这里了，我是胡光，我们下次见！

人人都能学会的编程入门课

>>> 每天 10 分钟，轻松学编程

胡光

原百度高级算法研发工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 指针系列（一）：记住，指针变量也是变量

下一篇 09 | 函数：自己动手实现低配版 scanf 函数

精选留言 (4)

写留言



徐洲更

2020-01-23

同样也有@大牛凯一样疑惑，不过不只是针对指针，而是所有数据类型。C语言是如何存放类型信息呢？对于`int a`而言，使用`a="123"`将字符串赋值给整型是会出问题的。这种类型错误的底层原理是啥呢？是不是C语言会划定一些区域，用来存放不同类型的变量呢？

展开

作者回复: 类型信息会被转换成相关的汇编代码。这个当中涉及到两件事情：一个是类型检查，这个是在编译阶段就做完了，另一个就是具体的程序运行，而运行阶段就已经没有了类型信息。也就是说，你所谓的出错，是在编译阶段报的错误。这个问题，你可以往后看，看到预处理命令一节的时候，可能就会认识的更清晰了。

1

2

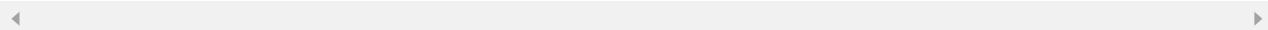


大牛凯

2020-01-23

老师好，请教一个问题，所有指针如果都是无差别存储地址的话，那拿到一个指针如何判断它的类型呢？整数型和字符型指针中，“整数”和“字符”这两个类型是存放在哪的呢？

作者回复: 这个信息已经转换成了相关的汇编代码，你想想，指针的类型是不是只有在加减运算和取值操作的时候有用？那么转换成汇编的时候，只需要对这两个操作，做针对性的转换即可。



💬 1

👍 2



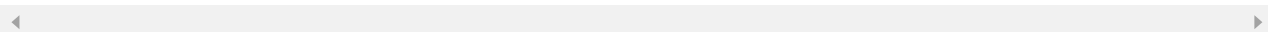
潮汐

2020-01-31

更清楚理解关于c语言中的数组，指针，内存空间(字节，存储单元)的关系，还有他们的一些运算的原理。老师的讲解思路清晰连贯，跟下来学到非常多，点赞！

展开 ∨

作者回复: d(^_^o)



💬

👍



Geek_Andy_Lee00

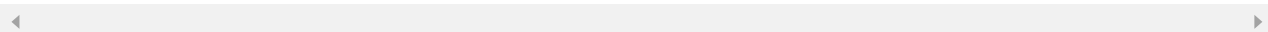
2020-01-24

参照老师的参考**答案和等价**关系，我能想到的替换类型还有：

- 1) 将参考**答案中的a替换为p;
- 2) $\&(* (p+1).x)$;
- 3) $\&(a[0].x)+2$ 或者 $\&(a[1].y)-1$;
- 4) $\&(*p.y)+1...$

展开 ∨

作者回复: 第四个有错误，直接引用运算符. 的优先级要高于 取值运算符 *，也就是说先算右边的 p.y，这个是不对的。更多的关于运算符优先级的内容，你可以上网自行搜索。



💬

👍

