

做好闭环（二）：函数是压缩的数组，数组是展开的函数

2020-02-01 胡光

人人都能学会的编程入门课

[进入课程 >](#)



讲述：胡光

时长 17:23 大小 13.94M



你好，我是胡光。

不知不觉，我们已经学完了语言基础篇的全部内容。其实还有很多东西想给你讲，可限于篇幅，所以咱们整个语言基础篇中的内容，都是那些，我认为你自学容易忽视的，容易学错的知识点。有道是，授之以鱼，不如授之以渔，我也相信只要你跟着课程学习，一定会感觉到自己收获到了“渔具”。如果能引发你的主动思考，进而触类旁通，举一反三，那这场学习过程就更加有意义啦。



我也非常高兴，看到很多同学都在紧跟着专栏更新节奏，坚持学习。每每在专栏上线的第一时间，这些同学就给我留言，提出自己的疑惑。大部分留言，我都在相对应的文章中回复过

了，而对于文章中的思考题呢，由于要给你充足的思考时间，所以我选择在今天这样一篇文章中，给你进行一一的解答。

看一看我的参考答案，和你的思考结果之间，有什么不同吧。也欢迎你在留言区中，给出一些你感兴趣的题目的思考结果，我希望我们能在这个过程中，碰撞出更多智慧的火花。

函数：自己动手实现低配版 `scanf` 函数

在这一节里面呢 [🔗 《函数：自己动手实现低配版 `scanf` 函数》](#)，我们讲了函数的基本概念，明确了“实参”和“形参”两个概念，并且知道了函数传参的过程，就是“实参”给“形参”赋值的过程。


还有，我们介绍了“传入参数”和“传出参数”两个概念，弄懂这两个概念，对于设计一个函数来说，还是非常重要的。“传入参数”是从外部，传入到函数内部，影响函数内部执行逻辑的参数，“传出参数”呢，就是由函数内部，传出到函数外部的参数，一般是以传送地址为主要形式。

最后呢，我留了两个开放性的思考题，我选择其中一个你可能会不知所措的题目，来讲解一下如何完成这个题目。下面就看看我的答案吧。

思考题（1）：数组和函数的思考

关于这个问题呢，我们首先来具象化一下，我们设想一种具体的问题情况，比如说：`arr` 数组里面，`arr[i]` 存储的是 $2 * i$ 的值，你可以认为是第 i 个偶数的值；`func` 函数的功能呢， $\text{func}(x) = 2 * x$ ，就是计算得到第 x 个偶数的值。如下述示意代码所示：

```
1 int arr[100] = {0, 2, 4, 6, 8, 10, ...};
2 int func(int x) {
3     return 2 * x;
4 }
```

 复制代码

解析这个示例代码，我们先从数组 `arr` 和函数 `func` 的不同处开始说起。

很明显，两者的本质不一样，`arr` 是数组，对于代码 `arr[36]`，相当于访问数组下标 36 的位置中存储的值；而 `func` 是函数，对于代码 `func(36)` 来说，也会得到一个整型值，但是

这个整型值，却是由 func 的函数逻辑代码计算得到的。简单来说，就是对于 arr 中的值进行访问，是一个静态的过程，而对于 func 函数得到返回值的过程来说，是一个动态计算的过程。

我们再来看看两者的时间和空间效率，也就是代码运行速度以及所需要的存储空间之间的比较。

关于时间效率方面，通常来说是数组访问速度要比函数计算速度快很多。

而空间效率的话，函数通常要比数组节省很多存储空间，就像 func 函数的值，是动态计算得到的，通常情况下，不管我们代码中执行 func(100) 还是 func(10000)，我们不需要修改函数的代码。但对于 arr 数组来说，当我们需要访问 arr[100] 的时候，数组最起码要有 101 个元素空间，而当我们想要访问 arr[10000] 的时候，数组最起码要有 10001 个元素空间。总的来说，就是函数比数组更加节省空间，数组比函数呢，得到结果的速度更快。

说完二者的不同以后，我们再来看看二者的相同之处。

站在使用者的角度来看，当你盯着 arr[100] 和 func(100) 这两段代码看的时候，你没觉得这两个代码的异常的相似？func 和 arr 就是名字不一样，如果这个时候我将 func 后面的小括号换成中括号，你是不是就会觉得 func 是一个数组？

对！你可能发现了，在使用者看来，func(100) 和 arr[100] 的作用是完全一样的，区别可能只是中括号和小括号的区别。你不觉得站在使用者的角度，考虑这个问题很有趣么？本质区别很大的两个东西，一个函数，一个数组，突然发现它俩的区别根本没有那么大。

简单来说，就是在数学里，函数做的事情就是“映射”，传入一个值，传出一个值。在程序中也不例外，函数做的事情，就是从传入值到传出值的映射。而数组做的事情呢，其实是从下标到存储值的映射。你会发现，数组和函数做的事情，本质上都是映射！

最后，我来总结一下，这个总结讲对你日后的程序设计思维有巨大的帮助，这句话就是“**函数是压缩的数组，数组是展开的函数**”，也就是说当你可以用数组进行程序的时候，你也可以使用某个能够完成相同映射功能的函数来进行替代。

二者在程序设计方面的差别，就在于时间和空间的使用效率，数组在时间效率方面占优势，函数在空间效率方面占优势。当你理解了这些事情以后，你就可以更好的理解某些资料里面经常讲的“**时间换空间**”或者“**空间换时间**”的概念了。你现在可以简单的理解成为是数组思维和函数思维之间的互相转换。

预处理命令：必须掌握的“黑魔法”，让编译器帮你写代码

关于预处理命令这个知识点，我们用了两节课的篇幅来讲解，[🔗《预处理命令（上）：必须掌握的“黑魔法”，让编译器帮你写代码》](#)和[🔗《预处理命令（下）：必须掌握的“黑魔法”，让编译器帮你写代码》](#)。其中讲了两种使用比较多的预处理命令，宏定义和条件编译。并且强调了，宏定义就是做简单替换，条件编译做的事情，就是代码剪裁，根据条件是否成立，决定哪段代码最终留在“待编译源码”中。

其中，用户 @一步 问到：有没有什么办法可以看到预处理阶段后的待编译源代码的内容？这个应该是很多小伙伴的共同问题吧，在这里我就来讲一下。

在 Linux/Mac 的编程环境下呢，操作比较简单，原本的程序编译命令是 gcc 加源文件名，如果你想看到待编译源码的内容，你只需要在中间加一个 -E 编译选项即可，例如：gcc -E test.c。如果你用的是集成开发环境，那你就需要自己搜索解决办法了，你可以搜索关键词如：XXX 下如何查看宏展开内容。XXX 就代表了你的集成开发环境。

对于课后的思考题，这里必须为用户 @Geek_Andy_Lee00 和用户 @Aaren Shan 的回答点赞。答案虽然不是很完美，可我想说，答案不重要，重要的是思考过程。下面就来看看我给出的参考答案吧。

思考题：没有 Bug 的 MAX 宏

就像之前所说的，对于这个问题呢，能否满分通过，是不重要的，重要的是你在解决这个问题过程中遇到的一个又一个 Bug，以及你对于这些 Bug 的思考过程。下面我就将带你一步一步地解决，这个问题中，你可能遇到的几个典型的 Bug，以及解决办法。

首先，让我们先对样例输出的每一行编上序号，如下所示：


 复制代码

```
1 ①: MAX(2, 3) = 3
2 ②: 5 + MAX(2, 3) = 8
```

```
3 ③: MAX(2, MAX(3, 4)) = 4
4 ④: MAX(2, 3 > 4 ? 3 : 4) = 4
5 ⑤: MAX(a++, 5) = 6
6 ⑥: a = 7
```


我们先来实现一个最简单的 MAX 宏，如下所示：

```
1 #define MAX(a, b) a > b ? a : b
```

 复制代码


如上所示，MAX 宏的实现，利用了三目运算符，问号冒号表达式， $a > b$ 条件如果成立，表达式的值等于 a 的值，否则等于 b 的值。看似没问题，但如果你要是运行代码，你会发现，程序的输出可能会如下所示：

```
1 MAX(2, 3) = 3
2 ✕ 5 + MAX(2, 3) = 2
3 ✕ MAX(2, MAX(3, 4)) = 2
4 ✕ MAX(2, 3 > 4 ? 3 : 4) = 2
5 ✕ MAX(a++, 5) = 7
6 ✕ a = 8
```

 复制代码


你会发现，这种实现，只有第一行是对的，其余几行都是错的。我们就来首先分析一下第 3 行到底是什么错误。按照宏展开的替换原则，最外层的 MAX 宏会被替换成： $2 > \text{MAX}(3, 4) ? 2 : \text{MAX}(3, 4)$ 。然后我们再将里面的 $\text{MAX}(3, 4)$ 宏展开，就变成了：

```
1 2 > 3 > 4 ? 3 : 4 ? 2 : 3 > 4 ? 3 : 4
```

 复制代码

这段表达式代码，看着有点儿乱，别急，我来帮你分析。首先我们从左向右看，先分离出来第一个问号冒号表达式的结构：

```
1 (2 > 3 > 4) ? (3) : (4 ? 2 : 3 > 4 ? 3 : 4)
```

 复制代码

我们看到在这个里面，第一部分 $2 > 3 > 4$ 是条件；第二部分 3 是在表达式为真时候的返回值；第三部分，是包含两个问号冒号表达式结构的式子。我们继续对第三部分进行拆解：

 复制代码

```
1 (4) ? (2) : (3 > 4 ? 3 : 4)
```


继续拆解后，我们发现，第一部分 4 是条件；第二部分的 2 是表达式为真时的返回值；第三部分，就是一个单独的问号冒号表达式。拆解到现在为止，已经不需要再继续拆解了。

要想理解原表达式，我们需要先了解 $2 > 3 > 4$ 这个“惨无人道”的表达式的值。这个表达式，从左向右执行，首先执行 $2 > 3$ 这个条件表达式的判断。之前我们讲过，条件表达式的值，只有 1 和 0，那么 $2 > 3$ 这个表达式的值，显然是 0，之后其实是在做 $0 > 4$ 的判断，结果也是 0。

所以 $2 > 3 > 4$ 这个表达式的值，就是 0，也就是假值，代表条件不成立，之后的事情，就是转到了两个问号冒号表达式的部分，剩下的事情，你自己就可以理解，最终原表达式的值为什么是 2 了。

理解了原表达式值计算的原理以后，下面让我们来分析一下，为什么会出现这种问题。本质原因，就在于我们实现的宏中，参数 a, b 原本都是独立的表达式部分，而我们却简单的把它们放到问号冒号表达式中，导致展开以后的内容前后连接到一起后，改变了原本我们想要的计算顺序。

所以针对这种情况，我们在实现宏的时候，可以给每个参数部分，都加上一个括号，就变成了如下所示的实现方式：

 复制代码

```
1 #define MAX(a, b) (a) > (b) ? (a) : (b)
```

至此，你就会得到如下的输出：

 复制代码


```
1 MAX(2, 3) = 3
2 5 + MAX(2, 3) = 2
```



```
3 MAX(2, MAX(3, 4)) = 4
4 MAX(2, 3 > 4 ? 3 : 4) = 4
5 ✕ MAX(a++, 5) = 7
6 ✕ a = 8
```

在这份输出中，第 1 行、第 3 行、第 4 行均已正确。如果你自己，仿照我上面说的方式对第二行内容的输出，加以分析，你一定可以知道如何修正第 2 行的结果错误。如果你努力以后，还是想不到的话，可以参考用户 @Aaren Shan 的留言。这样做以后呢，你程序的输出，就会变成如下输出：

```
1 MAX(2, 3) = 3
2 5 + MAX(2, 3) = 8
3 MAX(2, MAX(3, 4)) = 4
4 MAX(2, 3 > 4 ? 3 : 4) = 4
5 ✕ MAX(a++, 5) = 7
6 ✕ a = 8
```

 复制代码

其中还是有两行是错误的，你如果试着展开第 5 行的宏，你会得到如下的代码：

```
1 a++ > 5 ? a++ : 5
```

 复制代码

在这行代码中，如果 `a++` 表达式的值真的大于 5 的话，那么 `a++` 就会被执行两次。而原本使用者的意图，是执行一次 `a++`，如何让 `a++` 只执行一次呢？这需要用到之前我跟你提过的 `__typeof` 相关的技巧了，下面是我给你准备的参考代码：

```
1 #define MAX(a, b) ({ \
2     __typeof(a) __a = (a), __b = (b); \
3     __a > __b ? __a : __b; \
4 })
```

 复制代码

在这段代码中，我们定义了两个中间变量，`__a` 和 `__b` 用来存储宏参数中 `a` 和 `b` 部分的原本的值，之后判断大小的部分呢，我们使用新的变量 `__a` 和 `__b` 即可。

这段代码中，我们看到了，无论是 a 部分，还是 b 部分的表达式，只被使用了一次，也就保证了只被计算了一次。而这个里面，用小括号括起来了大括号，实际作用是把一个代码段，变成一个表达式。根据任何表达式都有值的特性，这个代码段表达式的值，等于其中最后一行代码表达式的值，也就是等于最后那行问号冒号表达式的值。

第 6 行的错误，其实和第 5 行的一样，解决了第 5 行的错误，这一行的 Bug 也就解了。

至此，我们就几乎完美地解决了 MAX 宏的问题了。通过这个问题，你会看到，预处理命令虽然强大，可你需要拥有掌握这种强大的力量。这份力量，包括了你的基础知识储量，还包括了你严谨的思维逻辑。

想要掌握强大，必先变得强大，记住，一步一个脚印，才是最快、最靠谱的成长路线，学习过程中，没有捷径。

好了今天的思考题答疑就结束了，如果你还有什么不清楚的，或者有更好的想法的，欢迎告诉我，我们留言区见！



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。

精选留言 (4)

写留言



1900

2020-02-01

@Aaren Shan 的方法我早就想到了，是不行的，我后面试了如下方法，还是不行。

```
#define MAX(a, b) { \
    __typeof(a) c = a; \
    __typeof(b) d = b; \
    c > d ? c : d; \...
```

展开 ▾

作者回复: 你把你的这个宏展开一下，代码会是 int c=a的内容，int d=b的内容，c>d...，在同一行中用逗号表达式链接了两个定义语句，这存在语法歧义，这是不对的。

1

1



潮汐

2020-02-02

很多问题还是无法自己独立思考出来，需要参照下面留言和老师的答疑，才能发现，原来应该这样啊！

针对宏命令的问题，看了老师的答疑，对，宏定义只是做简单的替换，有更深刻的理解了！

1

1



潮汐

2020-02-02

老师，按照@1900的写法，写成一行是不是一定行不通（编译不通过），如果宏替换内容包含多行的逻辑，是不是一定要换成多行的写法，用;结尾每一行？

展开 ▾

1

1



柒~龟虽寿!

2020-02-02

着急！还请大家答复我，上次课log那个bug，如何消除，我把helloworld后面加上一个参数，如"huv"就行了，但估计不是这个意思。

展开 ▾

作者回复: 其实只需要在将第二个printf语句改写成: printf(frm, ##args); 即可。

