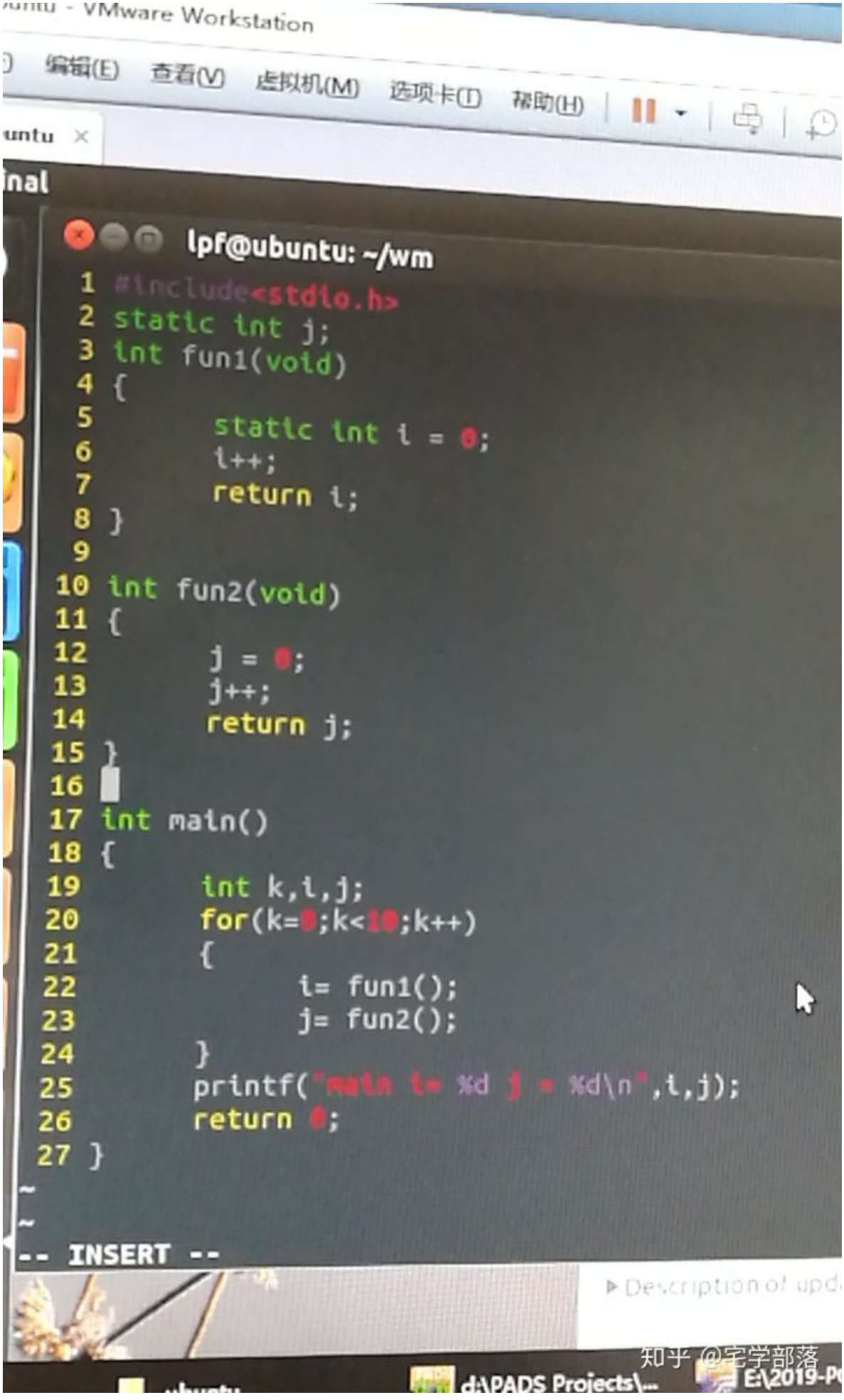
跟涛哥一起学嵌入式 21：一个static关键字引发的思考

文档说明	作者	日期
来自微信公众号：宅学部落(armLinuxfun)	王利涛	2019.10.22
嵌入式视频教程淘宝店：https://wanglitao.taobao.com/		
联系微信：brotau(宅学部落)		

今天有个学员问了一个C语言的static静态变量的细节问题，以前自己也没怎么注意过，感觉挺有意思，就跟大家分享下。


VMware Workstation
编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
lpf@ubuntu: ~/wm
1 #include<stdio.h>
2 static int j;
3 int fun1(void)
4 {
5 static int i = 0;
6 i++;
7 return i;
8 }
9
10 int fun2(void)
11 {
12 j = 0;
13 j++;
14 return j;
15 }
16
17 int main()
18 {
19 int k,i,j;
20 for(k=0;k<10;k++)
21 {
22 i= fun1();
23 j= fun2();
24 }
25 printf("main i= %d j = %d\n",i,j);
26 return 0;
27 }
~
~
-- INSERT --
d:\PADS Projects\...
E:\2019-P
知乎 @宅学部落

在上面的程序中，分别定义了一个静态变量 i 和全局变量 j，然后在main函数中循环调用10次后，再分别打印 i 和 j 的值，不用纠结地去想，答案分别是 i=10, j=1。

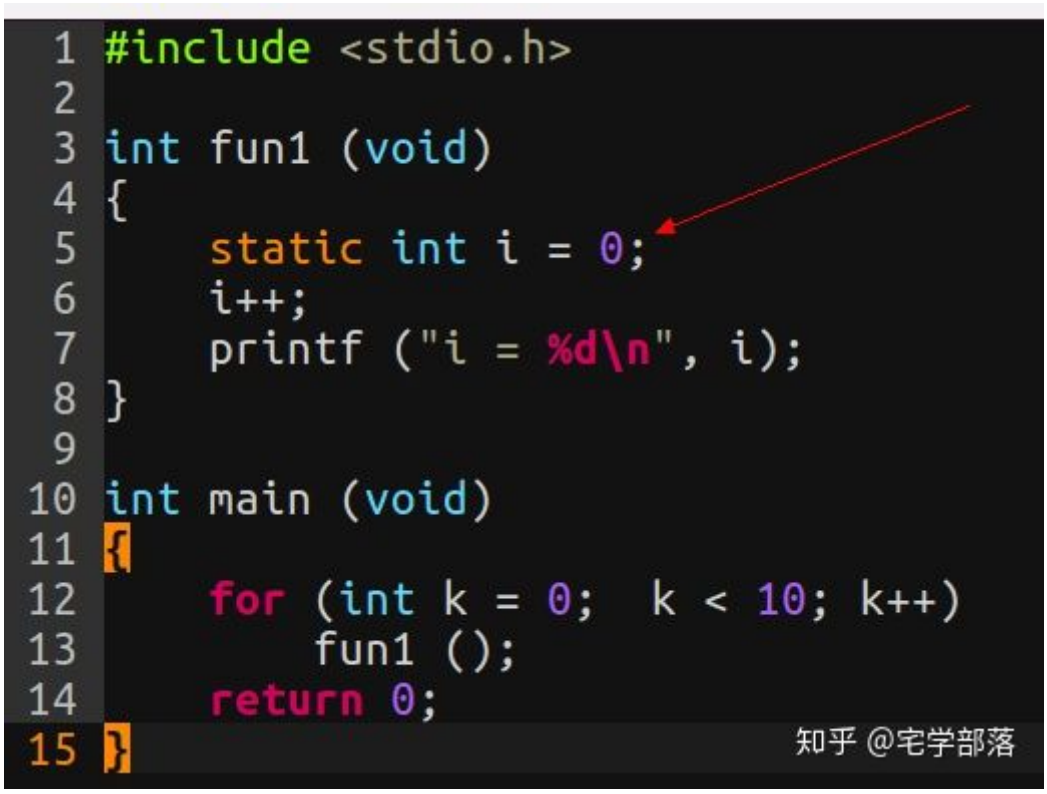
然后问题就来了：为什么i 的值等于10，而 j 的值等于1呢？

全局变量 j 的值等于1，这个很好理解：我们每次调用 fun2 函数时，都会将全局变量 j 重新赋值为0，然后来个 ++ 操作，所以 j 的值在调用10次之后依然是 1。

而对于静态变量 i，当我们10次调用fun1时，fun1函数体内的语句static int i = 0; 会不会每次都会将 i 初始化为0？答案是不会。

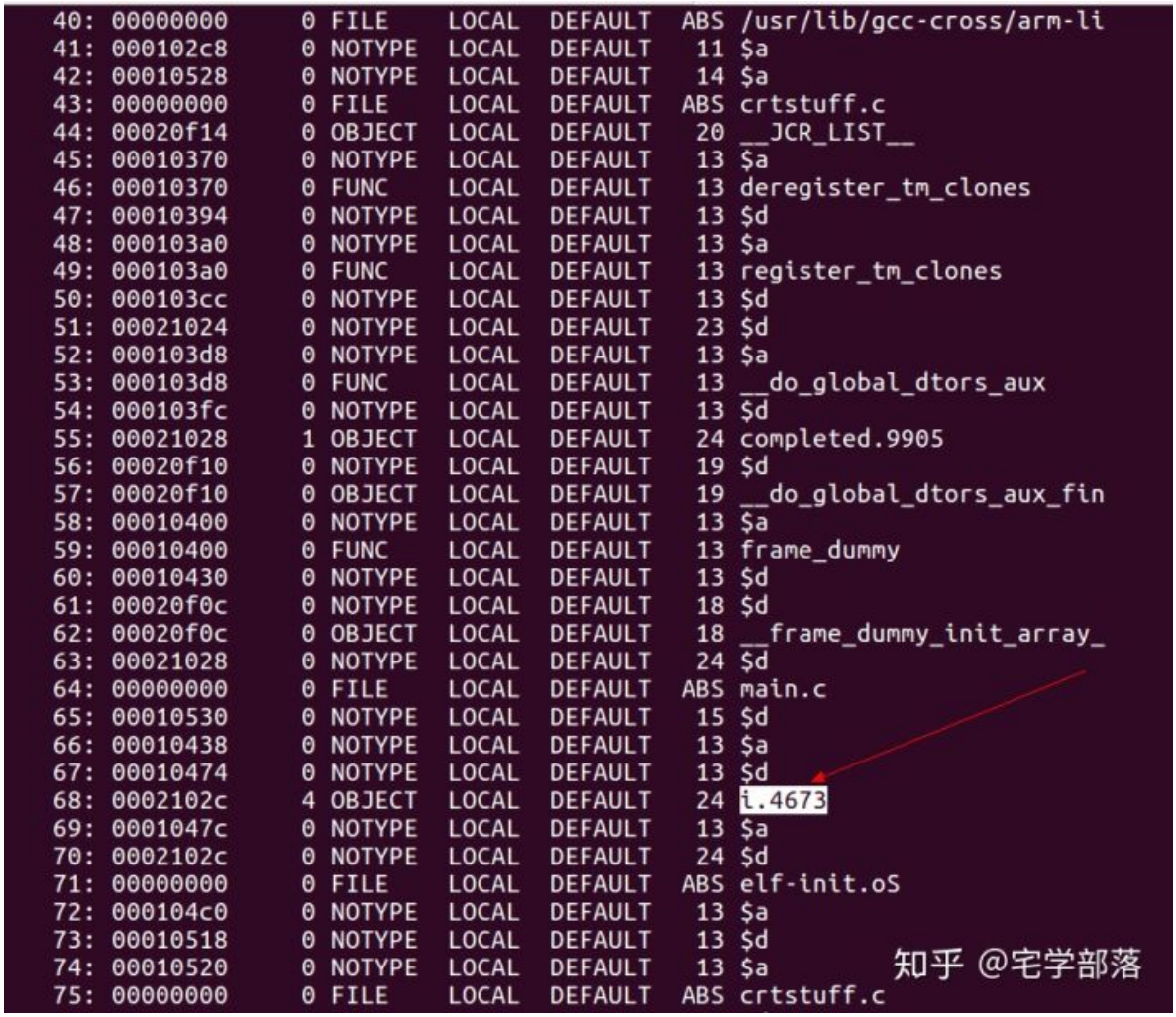
这里就要涉及到普通局部变量和静态变量的区别了：普通局部变量是定义在函数体内的变量，是在栈中存储的。我们可以通过栈指针来访问和修改它，当函数退出时，栈销毁，普通局部变量也就灰飞烟灭，生命周期结束。而静态变量则不同，当一个局部变量使用static修饰时，我们可以改变这个局部变量的存储方式从栈中迁移到数据段或BSS段中，升级为静态变量，但是这个静态变量的作用域不变，仍然由大括号{}决定。

比如下面这段程序，我们定义了一个静态变量 i：



使用ARM交叉编译器编译，然后使用readelf 文件查看其符号表：

```
$ arm-linux-gnueabi-gcc  main.c
$ readelf -s  a.out
```



我们会看到一个叫 i.4673 的变量保存在BSS段中，这就说明了，当我们使用static修饰一个局部变量时，它的存储方式会发生变化，有栈中迁移到数据段或BSS段中。

还需要注意的是，当我们使用static修饰一个局部变量时，如果我们不初始化，默认值是0；而普通局部变量如果不初始化，默认值则是一个随机值。

如果我们使用static定义一个静态变量时对其进行初始化，这个初始化语句只有第一次执行才有效。这也解释了为什么我们多次调用fun1时，i 的值不会重新初始化为0，而是保存上一次函数退出时的值。我们接着再看一个例子：

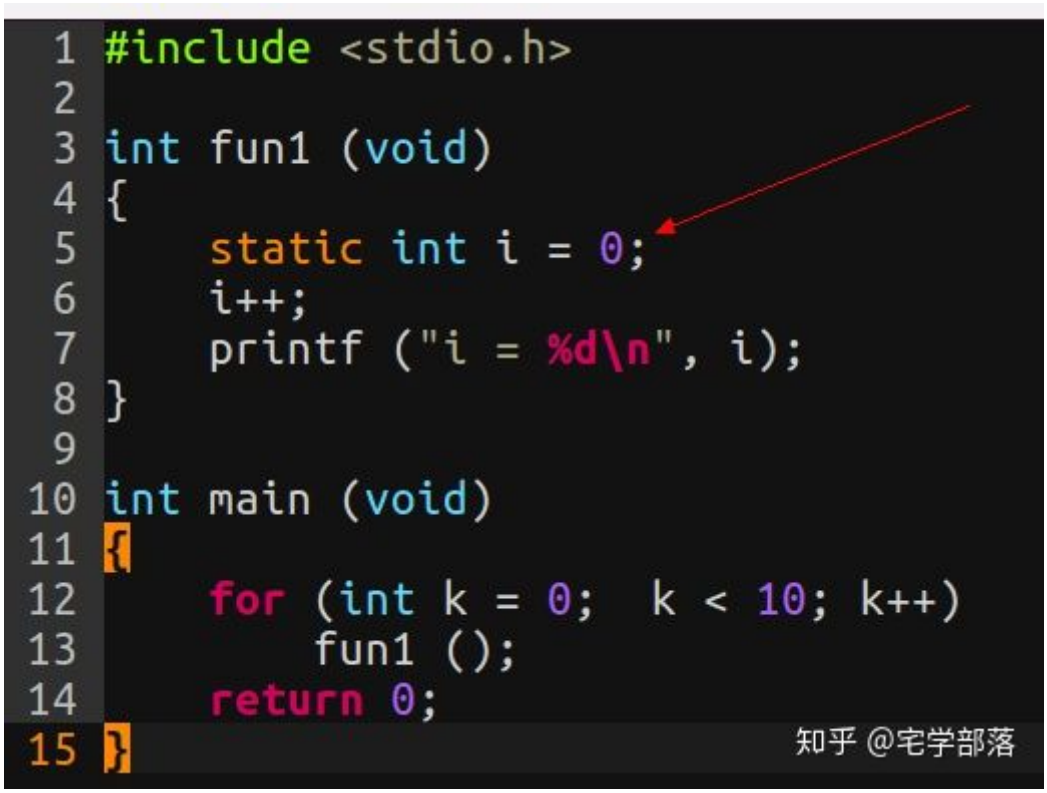
在上面的程序中，我们在调用fun1时，使用一个变量 arg 来给静态变量 i 进行初始化。编译这个程序，你会发现编译错误：

```
error: initializer element is not constant
static int i = arg;
```

这是另外一个需要注意的地方：static静态变量初始化语句需要使用常量进行初始化。

为什么static修饰的局部变量需要常量才能初始化呢？其实这个也很好理解：static修饰的静态变量，是存储在数据段或BSS段中的，这两个段中的变量在编译阶段就要给它们分配存储空间，然后初始化。这跟函数内的局部变量在运行时才给它们分配存储空间是不同的。在编译阶段，因为数据段或BSS段的变量需要一个确定的值来初始化(要么是0，要么是指定的常量值)，当static静态变量也要保存到这块区域时，因此必须也要用一个常量来初始化。

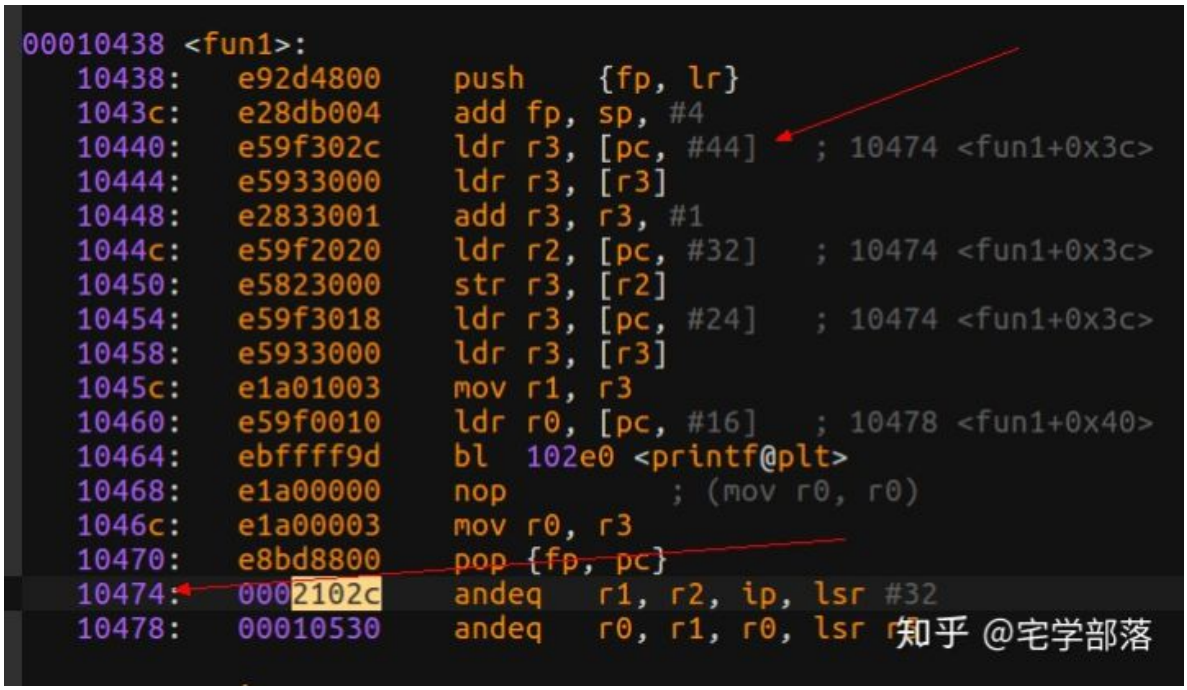
以上就是我们使用static关键字去修饰一个静态变量时，需要注意的一些细节。接下来我们就要思考了：当我们在函数体内去定义一个静态变量时，编译器到底是如何处理它的，或者说生成的指令代码到底是什么样的？我们以下面的代码为例：



交叉编译上面的程序，然后再反汇编，生成汇编代码：

```
$ arm-linux-gnueabi-gcc  main.c
$ arm-linux-gnueabi-objdump -D a.out > 1.s
```

分析生成的汇编文件1.s，找到fun1函数的实现：



分析fun1的反汇编代码，我们可以看到，当我们多次调用fun1函数时，并没有每次都将变量 i 赋值为0，在函数体内压根就没有这样的指令，而是一上来就对 i 做++操作，i 变量存储在0002102C这个地址，而这个地址在哪里呢？在我们的BSS段空间内：

通过以上分析，我们可以得出结论：当我们在一个函数体内使用static定义一个静态局部变量时，在编译阶段，遇到static int i = 0;这样的语句，编译器会将该变量存储在数据段或BSS段中。而且这个初始化语句只有一次有效，阅后即焚。当我们多次调用fun1时，我们在函数体内并没有找到这条语句的汇编指令，这说明编译器在首次编译后，然后就可能把它当作一个声明语句来处理了。

C语言博大精深，任何一个细节细细品味，都能牵涉出一系列自己想不到的知识来，进而能不断更新和完善我们的知识体系。感谢这位学员的问题，让我们对C语言的语法理解又加深了一层。


ARM