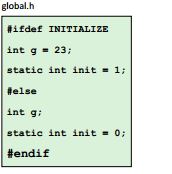
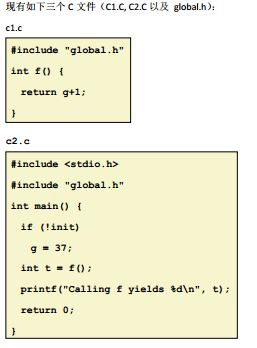
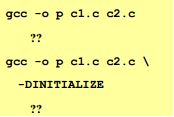
第七章家庭作业

计科1502 201507010206 黄郭斌

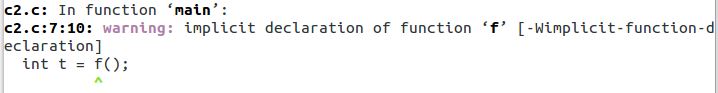
说明：写得有点长，讲了完整的过程。很多内容和书上差不多，对问题的解释只看加粗的字就行了。



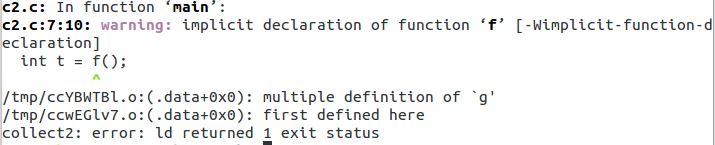
1. ***如下两个命令行的输出分别是什么？ 并解释为什么会出现那样的输出结果。***



使用 gcc -o p c1.c c2.c 编译链接两个源文件时，编译正常通过，但是gcc会提示一个warning，如下图：



使用 gcc -o p c1.c c2.c -DINITIALIZE编译链接两个源文件时，编译正常通过，链接出错， gcc会提示一个warning和一个错误提示，如下图：



**分析：**

这是因为在main函数中引用了函数f，但是并没有声明该函数的原型，因为编译的过程是对单个文件进行的，编译的时候gcc无法看到c2.c中给出的函数f的定义，但是编译器会根据调用的特点聪明地猜测出这个符号是一个函数，并且根据其实参的特点猜测其形参类型，并动态地生成相应的汇编代码（主要是函数栈帧的参数准备），并且引出一个strong type的外部符号(函数符号)输出给汇编器。

汇编器根据编译器输出的信息构造好符号表，其中的一个条目就记录了这个符号f，并标明了这个符号是一个外部符号（undefined in this file），名字（f）等内容，但是有趣的是，对于外部符号，汇编器产生的可重定位目标文件的符号表（放在ELF格式中的.text section）中却没有关于这个符号类型的任何信息（课本P453）。

于此同时，汇编器在检测到这个引用的目标位置不在文件内部，因此会将原本是存放目标地址的地方放入一个由系统决定的值（一般如果目标是一个全局数据这个值为0，这个值在链接过程中会被修改），并在生成的可重定位目标文件的重定位条目（放在ELF格式中的.rel section，特别地，这里由于是对外部函数的引用，因此其放在.rel.text中，注意，虽然符号表中没有记录这个符号的类型信息，但是编译器的输出文件中包含了这个符号的类型信息）中详细地记录这个未被赋上正确的目标地址的引用的位置（在相应的section的偏移量），引用的符号（好在链接过程中通过符号表找到处在另一个文件中的正确位置）和重定位的类型（如何引用目标地址，是PC relative还是absolute？）

在另外一个源文件（c2.c）的编译和汇编也发生了类似的事情，但是与c1.c不同的是，由于这里符号f不是外部符号而是一个全局符号，符号表中记录了这个符号的类型，地址（节偏移），处于哪个section等等，而由于这个符号不是一个引用，因此在重定位条目里面自然没有关于f 的记录。

**注意到，在预处理器处理源文件的时候，由于未定义INITIALIZE，预处理器会插入：**

**int g;**

**static int init = 0;**

**分别到两个源文件中，并输出处理结果到.i文件中，而g未被赋初值，故为weak type symbol。这两个变量在编译的时候也会分配合适的空间，并引出链接器符号给汇编器。**

**在链接的阶段，链接器看到两个同名的全局符号，由于这两个符号都是弱类型的，因此随便选择一个符号，而本地符号同名则无所谓，两者都会被分别记录。**

在符号解析过程中，由于这两个目标文件引用外部库只用C标准库，而它在链接过程中是默认被加入的，因此符号解析不会出现问题。

重定位的过程中，先是进行了section的重定位，注意到由于之前每个符号表中符号的地址和每个重定位条目记录的目标地址都是根据节偏移的偏移量，因此这里对于section的重定并不会导致条目中地址的改变。之后，依据重定位条目，对.text .data .bss中未被赋上正确地址的目标附上运行时的地址（根据重定位条目中的symbol找到新的symbol table中对应的条目，读取其所在的节，偏移，并根据重定位之后的该起始位置计算出这个symbol定义的地址，若为absolute reference，则直接将这个地址写入到引用的地址（引用地址在重定位条目中），若为PC relative reference，则根据定义地址 和 引用地址之后一条指令的PC值，两者相减得出地址只差写入引用的位置）。

**在这个问题中，由于链接器选择了两个课重定位文件中定义的同名的全局变量之一，其地址被作为任何对该符号的引用的目标地址，因此尽管编译器为两个课重定位目标文件都开辟的全局空间，但是真正被使用的只有其中一个，而这显然和作者定义这两个全局变量的初衷相背离。但是编译却通过了。**

**而在链接过程中加上了-DINITIALIZE命令，意思声明编译时刻已经预定义了INITIALIZE，预处理器会插入：**

**int g = 23;**

**static int init = 0;**

**分别到两个源文件中，并输出处理结果到.i文件中，而g被赋初值，故为strong type symbol。因此在链接的阶段，链接器看到两个同名的全局符号，而且这两个符号都是强类型的，因此报错（multiple definition of g）。**

注意到，链接器对外部符号的情况知之甚少，对于函数，只是简单地将引用（就是函数调用）的地方指向函数定义的地方（这就是重定义做的事情），而对函数的定义到底是怎样的，它是看不见的。在这个例子中，在引用到这个函数f的时候，编译器根据其实参的特点猜测其形参类型，并动态地生成相应的汇编代码（准备函数栈参数），因此不管函数调用有没有出错，在编译的时候都不会出错。而且有趣的是，由于链接器能力有限，链接器也不会发现错误。直到运行时，才会出现runtime-error。

因此，虽然在这个例子中调用是正确了，但是依然是很危险的，因为一旦调用出错，编译器不会发现任何错误的信息。这就体现了函数声明的重要所在。

对于全局变量，当要引用到外部的变量的时候，编译器就没有那么聪明了，必须要显式的声明要引用的外部符号的类型，才能通过编译。

这些声明都应该是放在头文件中，从某种意义上说，头文件是函数声明和函数调用之间的一份合同，保证编译器拥有在独自编译各个源文件的情况下有能力检查函数调用的正确性，以防止函数调用的错误，避免因此形成的runtime-error。

在两个文件中定义了两个同名全局的变量，这在现实生活中极有可能是写代码的时候的失误，下面是一些防范此类问题的办法：

初衷1.该全局变量仅在本文件内使用

这种情况下，我们最好为它加上static关键字，以确保不为被其他文件使用。

初衷2. 该全局变量会在外部文件中使用

这种情况下，在定义的时候一定要为它赋上初值，这样链接的时候如果发现了与其他文件中定义的全局变量同名，好及时报错提醒你修改变量的命名，而防止出现如上面这种实际链接之后的结果和初衷不同的情况。而在其他文件中使用这个变量的时候，要加上extern 的变量声明语句。

1. ***交换编译顺序（比如*** *gcc -o p c2.c c1.c****） 对结果有影响吗？如果没有，为什么？ 如果有，为什么？（提示：从第七章 链接 的符号强弱的知识点考虑解答）***

**没有影响，编译时各自独立的，与顺序无关。命令行语句的顺序不同，可能会在链接过程中导致符号解析的情况有所不同，但是这是在与静态库进行链接的时候才会发生的情况，具体来说就是如果定义一个符号的库出现在引用这个符号的目标文件之前，那么引用就不能被解析。**

**库中目标文件加入到实际的可执行文件中的特点是，在从左到右扫描命令行语句的过程中，一旦被认为不需要被加，这个库中的目标文件在之后也永远不可能被加入了，哪怕后面出现了需要这个库中的符号的引用（除去在后面的命令行中又添上这个库的情况）。**

**而这里涉及到的库只是在链接过程中默认被加入的c标准库，而不管顺序如何c1.o c2.o均会被linker加入到最终的可执行文件中去，在一趟扫描完后，是否存在未解析的符号与顺序无关。**