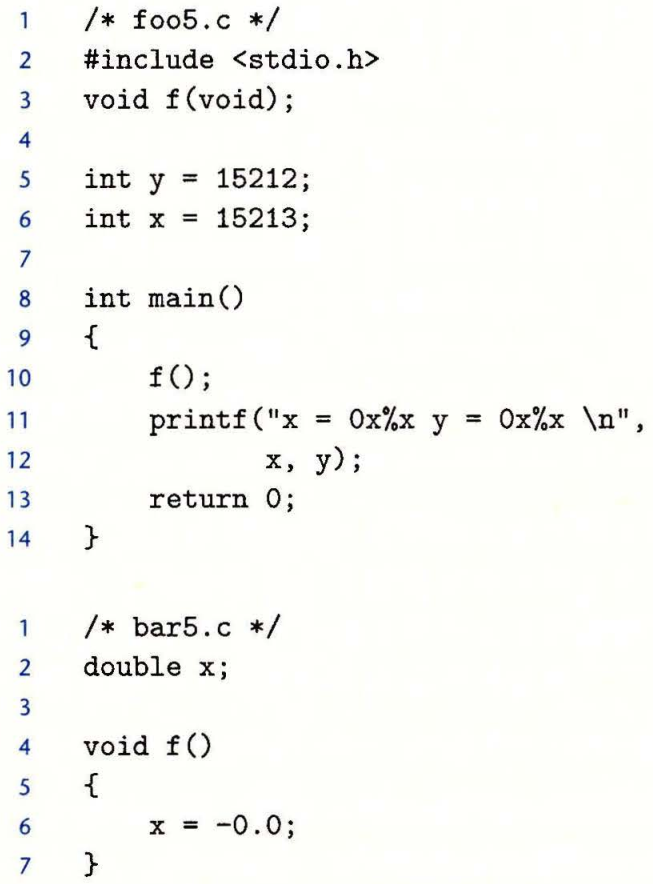
2024秋《计算机硬件基础》

## 09链接与运行 作业

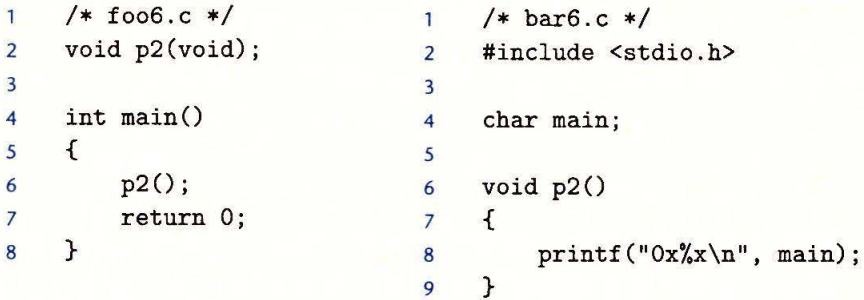
CSAPP 7. 7 不改变任何变量名字，不改变原有赋值语句，修改 7. 6. 1 节中的 bar5.c, 使得 foo5.c 输出 x 和 y 的正确值（也就是整数15213 和 15212 的十六进制表示）。



**答：将 bar5.c 中的 double x 改为 static double x**

**或将 foo5.c 中的 int x 改为 static int x**

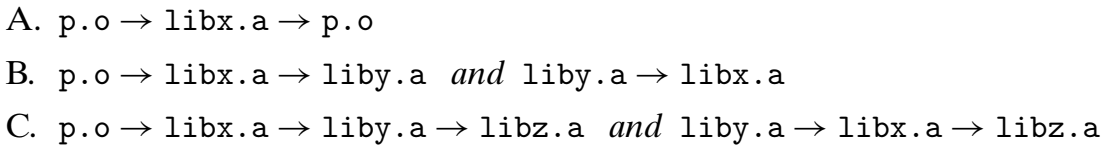
CSAPP 7.9 考虑下面的程序，它由两个目标模块组成：



当在 x86-64 Linux 系统中编译和执行这个程序时，即使函数 p2 不初始化变最 main, 它也能打印字符串 "Ox48\n" 并正常终止。请解释原因。

**答：foo6.c中的main作为强符号被定义，而bar6.c中的main定义为弱符号，会被解析到foo6.c中的main上，所以在p2中调用main与bar6.c中的变量main是否初始化无关**

CSAPP 7.10 a 和 b 表示当前路径中的目标模块或静态库，而 a→b 表示 a 依赖于 b, 也就是说 a 引用了一个 b 定义的符号。对于下面的每个场景，给出使得静态链接器能够解析所有符号引用的最小的命令行（即含有最少数量的目标文件和库参数的命令）。



**答：**

**A: gcc p.0 libx.a**

**B: gcc p.o libx.a liby.a libx.a**

**C: gcc p.o liibx.a liby.a libx.a libz.a**

CSAPP 7.12 考虑目标文件 m.o 中对函数 swap 的调用（作业题 7. 6）。

**9: e8 00 00 00 00 callq e<main+Oxe> swap()**

具有如下重定位条目：

r.offset = 0xa

r.symbol = swap

r.type = R\_X86\_64\_PC32

r.addend = -4

A. 假设链接器将 m.o 中的 .text 重定位到地址 0x4004e0, 把 swap 重定位到地址 0x4004f8 。那么callq 指令中对 swap 的重定位引用的值应该是什么？

B. 假设链接器将 m.o 中的 .text 重定位到地址 0x4004d0, 把 swap 重定位到地址 0x400500 。那么callq 指令中对 swap 的重定位引用的值应该是什么？

**答：**

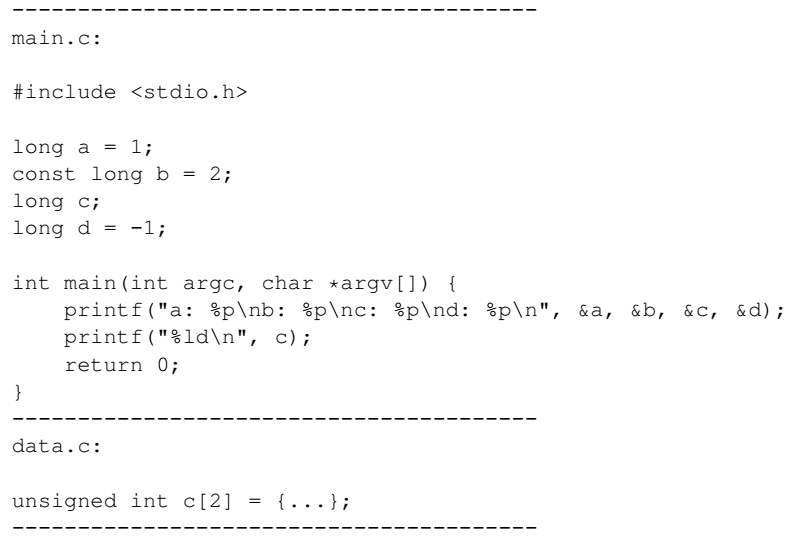
**A：当前指令地址：0x4004e0 + r.offset = 0x4004ea**

**下一条指令地址：0x4004ea + 5 = 0x4004ef**

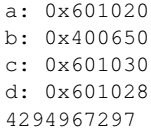
**偏移量：0x4004f8 + (-4) - 0x4004ea = 0x5**

**B：同理可得重定位引用的值为0x1d**

一位程序员编译运行下面两个文件：



看到了下面的输出：



注：省略了%p输出的前导零

他原以为这些变量会一个一个按顺序分布，很明显他错了。请分析发生了什么。

1. 可执行文件的符号表中，由main.c 生成的符号有几个？
2. main.c中定义的强符号有哪些，弱符号有哪些？
3. b的地址为什么离其它变量那么远？
4. 为什么c的地址在d后面？
5. 最后一行的输出4294967297是变量的值还是地址？为什么？

**答：**

1. **5个，a,b,c,d,main**
2. **强：a,b,d,main 弱：c**
3. **因为b是一个const常量，存储在在.rodata段中，不在.data段中**
4. **因为main.c中的c为弱符号，被data.c中的c强符号定义，打印的时候输出的是强符号的地址，所以在d的后边**
5. **地址，因为c是一个数组的指针，其指向一个地址空间的头部**