7.6

符号	swap.o.symtab条目?	符号类型	定义符号的模块	节
buf	√	外部	m.o	.data
bufp0	√	全局	swap.o	.data
bufp1	√	局部	swap.o	.bss
swap	√	全局	swap.o	.text
temp	×	×	×	×
incr	√	局部	swap.o	.text
count	√	局部	swap.o	.bss

7.8 在此题中，REF(x.i)→DEF(x.k)表示链接器将任意对模块 i 中符号 x 的引用与模块 k 中符号 x 的定义相关联。在下面每个例子中，用这种符号来说明链接器是如何解析在每个模块中有多重定义的引用的。如果出现链接时错误(规则 1)，写“错误”。如果链接器从定义中任意选择一个(规则 3)，那么写“未知”。

A.
```
/* Module 1 */
int main()
{
}
```
```
/* Module 2 */
static int main=1;
int p2()
{
}
```
(a) REF(main.1) → DEF(main.1)

(b) REF(main.2) → DEF(main.2)

B.
```
/* Module 1 */
int x;   weak
void main()
{
}
```
```
/* Module 2 */
double x;  weak
int p2()
{
}
```
(a) REF(x.1) → DEF(未知)

(b) REF(x.2) → DEF(未知)

C.
```
/* Module 1 */
int x=1; strong
void main()
{
}
```
```
/* Module 2 */
double x=1.0;  strong
int p2()
{
```
(a) REF(x.1) → DEF(错误)

(b) REF(x.2) → DEF(错误)

考虑下面的程序，它由两个目标模块组成：

```
1  /* foo6.c */
2  void p2(void);
3
4  int main()
5  {
6      p2();
7      return 0;
8  }
```
```
1  /* bar6.c */
2  #include <stdio.h>
3
4  char main;
5
6  void p2()
7  {
8      printf("0x%x\n", main);
9  }
```

因为foo6.c中的main()是强符号，而bar6.c中的main是弱符号，所以链接器选择第一个模块中的强符号main用以在p2()函数中打印字符时输出第一个模块中main的地址

当在 x86-64 Linux 系统中编译和执行这个程序时，即使函数 p2 不初始化变量 main，它也能打印字符串“0x48\n”并正常终止。你能解释这一点吗？

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+0xe>
```

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
```

PC相对

ADDR(r.symbol) + r.addend - refaddr.

0x400505 - 0x4 - (0x4004d0 + 0xa

0x4004f8 - 0x4 - (0x4004e0 +

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？ 0xa.

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？ 0x22.