符号	swap.o.symtab条目?	符号类型	定义符号的模块	节
buf				
bufp0				
bufp1				
swap				
temp				
incr				
count				

*7.7　不改变任何变量名字，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)。

*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 */          /* Module 2 */
int main()              static int main=1[
{                       int p2()
}                       {
                        }
```

(a) REF(main.1) → DEF(_____.____)

(b) REF(main.2) → DEF(_____.____)

B.
```
/* Module 1 */          /* Module 2 */
int x;                  double x;
void main()             int p2()
{                       {
}                       }
```

(a) REF(x.1) → DEF(_____.____)

(b) REF(x.2) → DEF(_____.____)

C.
```
/* Module 1 */          /* Module 2 */
int x=1;                double x=1.0;
void main()             int p2()
{                       {
}                       }
```

(a) REF(x.1) → DEF(_____.____)

(b) REF(x.2) → DEF(_____.____)

*7.9　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  }
```

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？

**7.10　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)。