


在这个例子中，我们看到全局符号 `main` 定义的条目，它是一个位于 `.text` 节中偏移量为 0 (即 `value` 值) 处的 24 字节函数。其后跟着的是全局符号 `array` 的定义，它是一个位于 `.data` 节中偏移量为 0 处的 8 字节目标。最后一个条目来自对外部符号 `sum` 的引用。`READELF` 用一个整数索引来标识每个节。Ndx=1 表示 `.text` 节，而 Ndx=3 表示 `.data` 节。

 **练习题 7.1** 这个题目针对图 7-5 中的 `m.o` 和 `swap.o` 模块。对于每个在 `swap.o` 中定义或引用的符号，请指出它是否在模块 `swap.o` 中的 `.symtab` 节中有一个符号表条目。如果是，请指出定义该符号的模块 (`swap.o` 或者 `m.o`)、符号类型 (局部、全局或者外部) 以及它在模块中被分配到的节 (`.text`、`.data`、`.bss` 或 `COMMON`)。

符号	.symtab 条目?	符号类型	在哪个模块中定义	节
<code>buf</code>				
<code>bufp0</code>				
<code>bufp1</code>				
<code>swap</code>				
<code>temp</code>				

<pre> 1 void swap(); 2 3 int buf[2] = {1, 2}; 4 5 int main() 6 { 7 swap(); 8 return 0; 9 } </pre> <p style="text-align: right; margin-right: 20px;"><i>code/link/m.c</i></p>	<pre> 1 extern int buf[]; 2 3 int *bufp0 = &buf[0]; 4 int *bufp1; 5 6 void swap() 7 { 8 int temp; 9 10 bufp1 = &buf[1]; 11 temp = *bufp0; 12 *bufp0 = *bufp1; 13 *bufp1 = temp; 14 } </pre> <p style="text-align: right; margin-right: 20px;"><i>code/link/swap.c</i></p>
a) <code>m.c</code>	b) <code>swap.c</code>

图 7-5 练习题 7.1 的示例程序

7.6 符号解析

链接器解析符号引用的方法是将每个引用与它输入的可重定位目标文件的符号表中的一个确定的符号定义关联起来。对那些和引用定义在相同模块中的局部符号的引用，符号解析是非常简单明了的。编译器只允许每个模块中每个局部符号有一个定义。静态局部变量也会有本地链接器符号，编译器还要确保它们拥有唯一的名字。

不过，对全局符号的引用解析就棘手得多。当编译器遇到一个不是在当前模块中定义的符号 (变量或函数名) 时，会假设该符号是在其他某个模块中定义的，生成一个链接器符号表条目，并把它交给链接器处理。如果链接器在它的任何输入模块中都找不到这个被引用符号的定义，就输出一条 (通常很难阅读的) 错误信息并终止。比如，如果我们试着在一