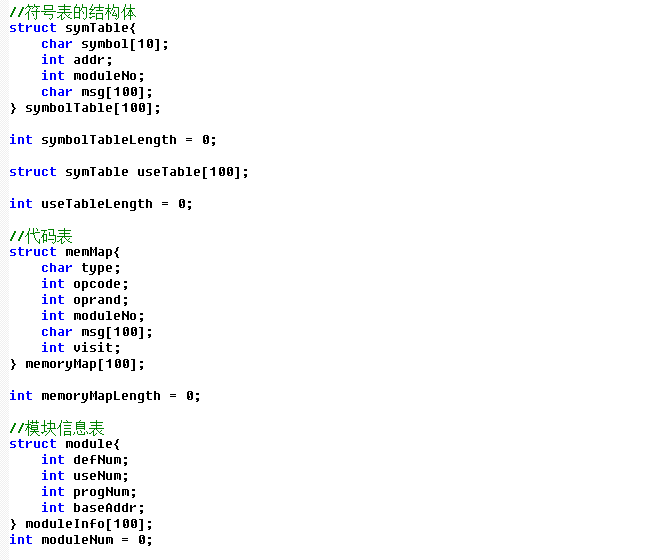
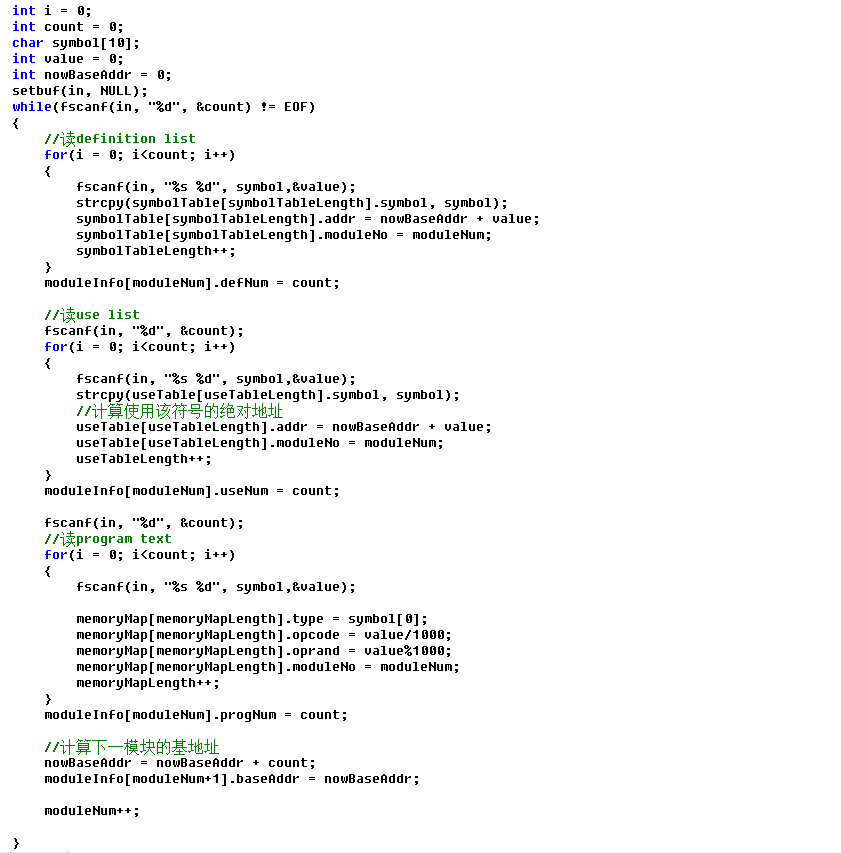
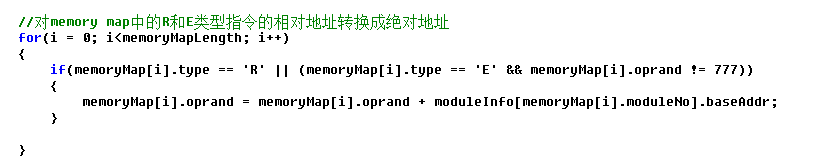
# 链接器的实验

1. 读取文件信息并处理
2. 建立符号表、代码表以及模块信息表的结构体
3. 初始化符号表、使用表、代码表和模块信息表

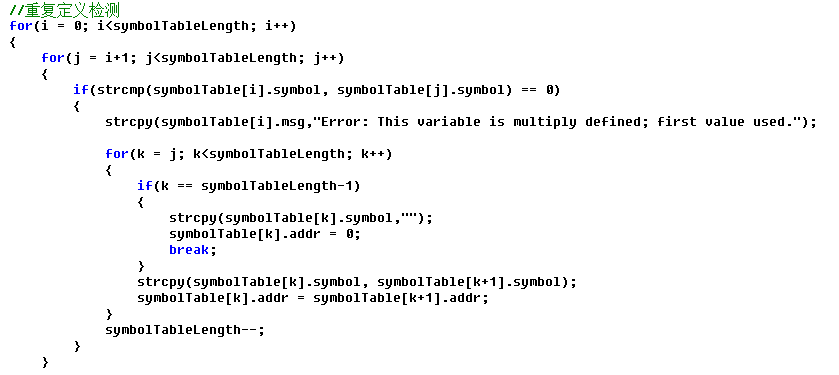


1. process\_one()函数中，进行第一遍处理，得到文件的输入信息
2. 
3. 指令分析
4. 对各指令进行处理，将E、R指令的地址进行转换，变为绝对地址



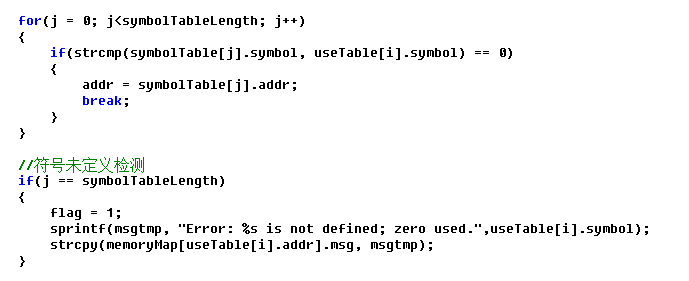
1. 从使用表中的地址引入，来处理E类型，并输出（涉及错误检测，详见代码文件）
2. 7中错误检测、处理并输出
3. 符号重复定义

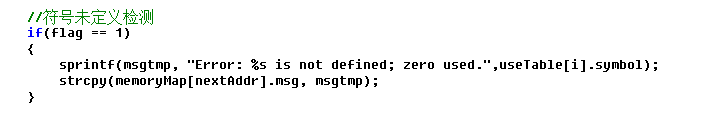
采用一个双重循环来解决，如果发现重定义的符号则将其删除，并写入错误信息，代码片段如下：



1. 符号未定义

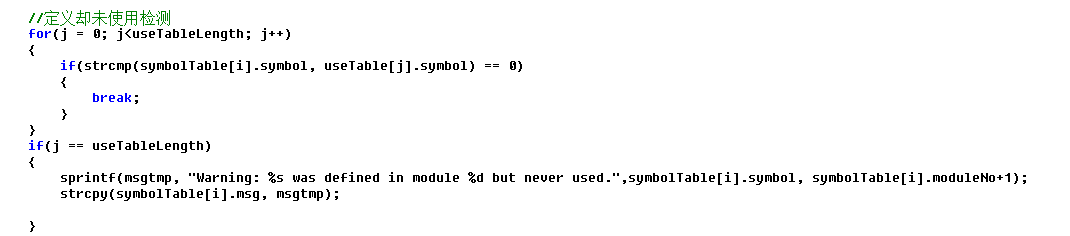
分为两个部分进行，处理E的循环里面对符号表进行遍历，如果找不到该符号，就将标志位赋值为1，并写入和输入错误信息，之后，由于，头部符号未定义，每条指令都要显示错误信息，故在遍历代码表时找到flag标志位为1的，并输出错误的信息，代码片段如下：





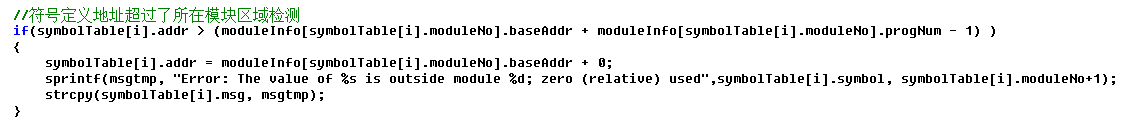
1. 符号定义未使用

遍历符号表中的每一个内容，去使用表中找，如果能找到，那么就break，遍历的次数永远都是小于使用表的长度的，如果等于了该长度说明未找到，并设置警告信息，代码片段如下：



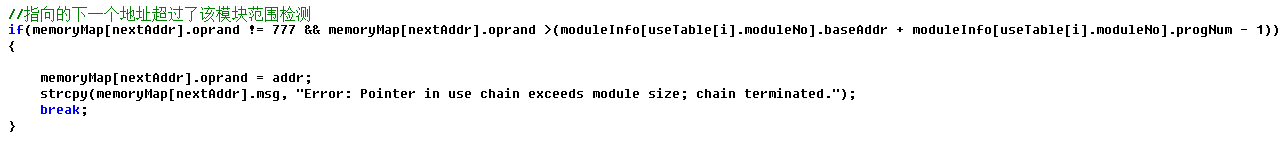
1. 符号定义的地址超过所在模块区域

遍历符号表内容，如果地址超过其所在模块地址范围进行比较，如果超出就设置错误信息，代码片段如下：



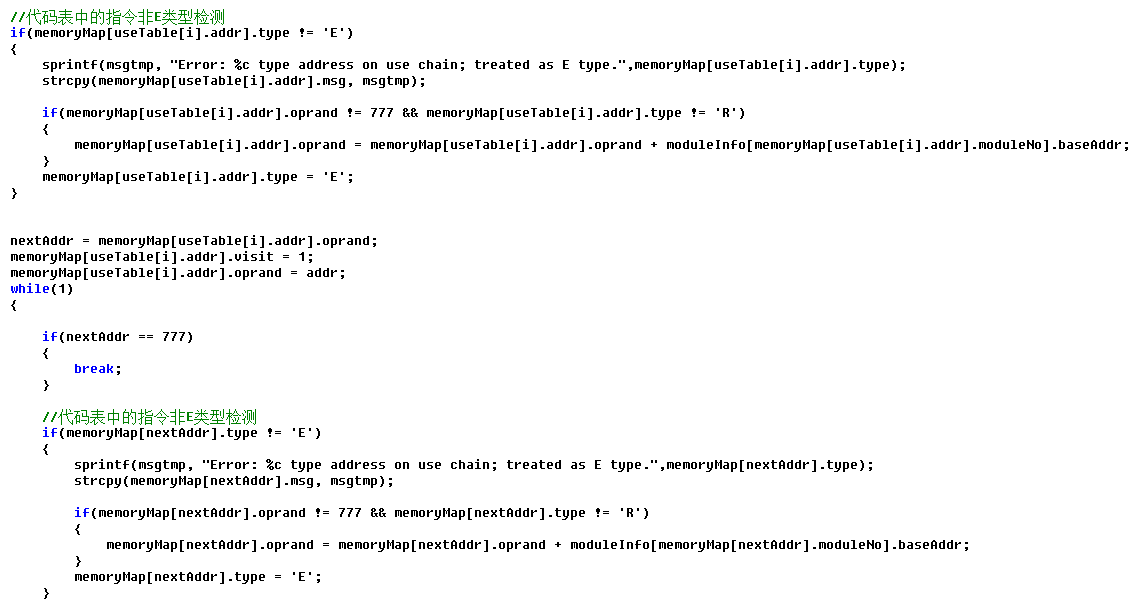
1. 指向的下一个地址超过该模块的范围

同样是在遍历符号表的过程中，如果发现下一个地址非777的地址，超过模块的范围，就设置错误信息，代码片段如下：



1. 使用表的符号但指令不是E类型

两处地方进行检测，一是从use表中得到链表头部地址时，发现所指地址并非E类型时，将其作为E类型来对待，并设置错误信息，二是在遍历链表的循环过程中，进行检测，代码片段如下：



1. E类型的地址不在表中

每次遍历时都需要将visit字段设置为1，处理完E类型之后，遍历代码表，找到未访问的E类型指令，并将其设为I类型，并写入报错信息

