编译课设难点分析

1. 词法分析

这部分比较简单，而且书上有案例，需要注意的是此处可能有时要进行回退，比如分析一个标识符或数字时。我直接从文件中用getc函数一个一个字符读取，使用了fseek函数进行回退，难点在于，fseek函数是将文件读写指针回退一个字符，getc函数每次调用时返回文件读写指针当前指向的字符，并将指针指向下一个字符（除非当前已经是最后一个字符），因此如果在读到文件字符时，又多读了一个字符，此时文件读写指针指向的依旧是最后一个字符，这时如果使用fseek函数进行回退，指针将指向倒数第二个字符，而非我们希望的最后一个字符，这可能导致程序死循环。

1. 语法分析

该部分较为简单，主要的难点是变量定义和函数定义及主函数的前几个符号可能相同，只要预读就行了。先读一个符号，可能读到int、char、void，如果是main函数，再读一个符号应该是main，如果是变量定义，再读2个符号，读到的应该是’[‘ ‘,’ ‘;’ ，三者之一。如果是普通函数定义，再读两个符号，读到的应该是’(‘。这样就处理了这个难点。

1. 语义分析及中间代码生成

这一部分的难点在于需要进行大量的查询符号表、以及报错。

1. 生成目标代码

这一部分的难点在于传参及函数调用，我设计了一个参数栈，一旦传参，就将参数地址压栈，遇到函数调用，就将参数弹栈直到个数符合，而遇到函数调用时，我先开辟一段新的函数运行栈给被调用函数，再将实参地址的值取出，并且存到形参的地址中，再储存返回地址到运行栈上。在函数的开头，需要先保存上一个函数运行栈的栈顶地址，遇到return语句，再取出返回地址到$31，用jr指令跳回。

1. 符号表管理

我采用的是书上的无序符号表，而我的经历告诉我，这不是一个很好的方案。首先是可维护性较差，符号表一旦建立，再向其中插入表项十分困难，以至于我最后在目标代码生成时重写了一遍，一个函数一个符号表。这里的难点我觉得是符号表中一定要保留足够多的标识符信息，最好连同标识的相对地址偏移也算出来，这样在生成中间代码时可以省去不少麻烦，而我将标识符分成了十种类型0-int,1-char ,2-const int, 3-const char,4-int数组,5-char数组,6-返回int的函数,7-返回char的函数,8-返回void的函数,9-int型参数,10-char型参数，我觉得分的类型越细越好，标识符的信息越完善。

1. 错误处理

错误处理最重要的一点是能够准确的报错，其次是报错之后不能终止程序，要继续进行编译。第一点要求很容易实现，但是第二点要求就意味着出错后需要进行相应的跳读，跳读不好可能出现跳过了出错代码或者误报错误或者使得程序陷入死循环，这是错误处理的难点。

为了跳读的准确，我没有根据错误类型的不同来进行跳读，而是根据错误发生位置的不同进行不同的跳读。因为同一种类型的错误，可能需要不同的跳读。比如缺失’;’，在分析变量定义和分析语句时，都有可能出现这个错误，但是前者应该跳读到下一个变量定义或函数定义或某种语句的开头，后者只要跳到某个语句的开头或函数结尾，它们所需的跳读不相同。因此，我设计了一个变参数函数void skip(int arg\_num,...)用来跳读，它的第一个参数指的是后面还有多少个参数。

在词法分析、语法分析、语义分析阶段，一旦出错，就先进行报错，接着就在下一行编译程序代码调用skip函数。如语句缺失’;’，就调用skip(10,ID,IFSYM,FORSYM,SWITCHSYM,SCANFSYM,RETURNSYM,PRINTFSYM,SEMICOLON,LBRACE,RBRACE)，以跳到下一个语句开头或者函数结尾。