从0开始的扩充C0文法编译器设计

本文旨在指导如何完成编译技术课程设计，实现一个扩展C0文法编译器（含优化）。

**第0周 编译器源代码阅读**

这是编译课设开始的前奏，需要阅读一份PL/0文法编译器源代码，并写出注释。一开始可能会无从下手，此时就需要参考编译教材中的附录部分，这个部分详细介绍了编译器的总体流程和每个函数的作用。有了这份指南，读代码就会变得容易得多，再加上Pascal语言易于理解的特性，我们就能够跟着代码大致走一遍编译的流程，而写注释的问题也就迎刃而解。

需要注意的是，尽管这次任务只是写注释，却是不可轻视的。趁早阅读代码，能够在心里大致画一个编译器的“蓝图”，到后期自己实现时就不会太茫然；而后期实现时，碰到不懂如何处理的地方，也可以以此为参考。

**第1周 文法解读文档**

在编译课设的开始，每个人都会抽到一份自己的文法，而本周就需要对文法进行解读，并编写一个覆盖详尽的测试程序。这周的工作是**相当重要**的，对文法解读的详尽与否与后期编译器的BUG数量有很强的相关性。我们需要仔细阅读文法的每一条规则，整理出其作用、限定条件和产生句子。这其中的细节是需要着重关注的，例如：标识符是否区分大小写？循环语句是先执行一次还是先进行判断？程序各个部分是否有强制的顺序要求？数组的下标是从0开始还是从1开始？······注意的是，本课程的C0文法与大家孰知的C语言文法在一些规则上是有明显的不同的，若没有仔细挖掘文法中的细节，后期生成的程序可能会产生难以发现的BUG。

在此基础上，需要写一个覆盖了文法所有分支的测试程序，这对之后每一步的测试都是有帮助的。好的测试程序能够帮你发现编译器的很多BUG。

**第2周 设计文档与词法分析程序**

第二周，我们需要完成编译器设计文档与词法分析程序。设计文档要求我们先将顶层设计好，也就是编译器的总体“框架”。有人可能会觉得奇怪，编译器都还没开始写呢，怎么就要求把程序结构、函数调用关系、存储方案、四元式等等都设计出来呢？实际上，这样做是有很大好处的：一方面，编译器设计是一个比较长的流程，总共持续约两个月，这之间不可避免的会有各种考试和课设过程，从一开始就设计好，才能保证不会出现前后设计不一致的现象产生；另一方面，先设计再编码的工作模式更有利于发现隐藏的BUG，而你的设计也会随着接下来每一周的迭代而逐渐趋于完善。因此，不妨花上一定时间，仔细设计自己的编译器。

本周，还需要完成一个词法分析程序，这部分内容相对容易，按照书本或课件中的流程来做即可。完成之后，就可以用上周编写的测试程序来验证，逐个单词对照输出结果，检查自己的程序正确性。

**第3周 语法分析程序**

本周，我们要在上周实现的词法分析程序的基础上，完成语法分析程序。语法分析程序从词法分析程序中获取单词，并使用递归子程序法分析每个语法成分，在控制台中输出成分信息。

在这里，我们就需要注意到文法的左递归问题。例如，我抽到的文法有这个部分：

＜有返回值函数定义＞  ::=  ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’|＜声明头部＞‘{’＜复合语句＞‘}’  //第一种选择为有参数的情况，第二种选择为无参数的情况

＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞  
＜变量定义＞  ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ )}

那么，当我们读到int和标识符的时候，就无从判断这是一个有返回值的函数定义还是变量定义，出现了左递归问题，需要改写文法或是进行跳读。

我选择的解决方案是进行跳读，即再读下一个单词，若是左小括号或左大括号，说明这是一个函数定义，否则是变量定义。

在语法分析完成后，若时间充裕，可以加紧进行符号表管理和四元式生成，避免后期过于紧张。

**第4周 代码生成程序**

在完成了语法分析程序之后，我们需要编写代码生成程序。这之间其实有两个步骤：从语法分析中生成中间代码，根据中间代码生成目标代码。看起来需要在一周时间完成稍显紧张，但实际上，若前期工作扎实，也是可以完成的。

在四元式生成过程中，我们需要符号表的支持，符号表记录了当前程序的变量、常量和函数的信息。当我们识别出某个语法成分时，就能根据该成分信息生成四元式。需要注意条件语句和switch语句的跳转标签问题。

四元式与MIPS汇编在形式上是比较相近的。将数据段（包含全局变量、全局常量）填写好后就可以进行代码段的生成。我们根据每条四元式的运算符，生成对应的MIPS汇编码。

**第5周 目标代码执行**

本周进行的实际上还是上周的工作，我们需要保证生成的目标代码能够在Mars模拟器中正确运行。这其实是一个不断发现错误并修正的过程，需要多多编写测试程序。重点需要测试的地方大致有以下几处：

* 函数递归调用：可编写诸如斐波那契数列、阶乘、组合数等包含递归的数学函数来验证
* 数组元素存取：可编写排序算法、复杂数组存取等程序来验证
* 复杂表达式计算：一条表达式中包含可能的所有成分、表达式在语句中等
* 条件语句嵌套：多重条件语句混合，考察跳转的正确性

以上是较容易出错的地方，编写测试程序时可以参考。

至此，一个能够生成正确源代码对应程序的编译程序已经完成。

**第6周至后期 错误处理和优化**

编译课设已过半，我们需要继续完善编译器并完成优化部分。

首先是错误处理部分。错误处理的要求是尽可能减小错误所影响的范围，并能够继续编译，发现后期的错误。这部分的重点是划清每种错误的跳读符号和同步符号，避免由于一个错误导致编译器进入死循环的状态。

优化部分需要完成以下优化：

* 基本块内部的公共子表达式删除（DAG图）
* 全局寄存器分配（引用计数或着色算法）
* 数据流分析
* 代码生成时合理利用临时寄存器
* 其他优化

其中，DAG图的生成需要注意多个父节点的情况，此时代码生成的顺序可能会对值产生影响。全局寄存器分配若使用着色算法，需要数据流分析的支持，这部分需要较多时间进行调试，建议及早完成。其他还有一些原理简单、易于实现但效果较好的优化，例如常数合并、窥孔优化等，实现这些优化是较好的选择。

至此，我们的扩充C0文法编译器就完成了。