**简单c0编译器分解**

**——编译课设指导**

1. **前言**

两个月前阅读pascal代码的时候，我可以说是略微迷茫，认为理论和实践相差甚远，尽管书本知识已经学习了大半部分，对整体还是没有足够的把握，认为这一课设任务非常艰巨。

　　两个月后，我实现了课设要求，并额外做了一些不硬性要求的代码优化算法，整个过程说起来也没有想象中那么艰巨，但受益良多，因此希望能够将些许经验写下来以供后人参考。

　　首先还是要对编译器的整体结构进行宏观把握。将编译器拆分成若干个小目标，我们就可以很清楚地描述出实现的流程和计划。书上对编译器组成的描述是：词法分析、语法分析、语义分析、代码生成、代码优化、出错处理和符号表处理，这七个部分构成了一个完整的编译器。而实际上，课设的审核共分为如下几个阶段：编译器源代码阅读与分析（与编译器编写无关）、文法解读作业（这个也不是实际的编码）、词法分析和设计文档（对应书上的词法分析）、语法分析程序（对应书上的语法分析和语义分析）、代码生成程序（代码生成，其中涉及到中间代码的生成和目标代码的生成）、目标代码执行、测试程序考核（一、二、三、四）。

　　虽然时间线已经在课设开始便已定好，各阶段工作也似乎不紧不慢，但各部分工作量完全不一致，我个人建议需要在开始有一个清晰的规划，在什么时间节点完成哪些工作。尽量提前完成任务，以备不时之需。

　　下面我按照自己的经验对整个工作进行剖分，按照流程执行，可以说是一定程度上起到提高效率的作用。

**二、工作流程**

**工作一：词法分析+错误处理**

　　首先是词法分析，其实这一部分算是最简单的部分，有pascal编译器作为参照，并且测试也很容易。之所以将错误处理也提前加入计划，是因为在词法分析时就已经遇到了几类词法编译错误，如字符串引号缺失等等，因此可以先把错误处理的接口写好，并加入在词法分析时遇到的错误。

实践过程中重点首先是将词法分析完善，根据pascal编译器的启示，建立自己的词法分析器，要注意的地方在词法分析到文件末尾时的处理，需要仔细思考；其次是错误处理部分代码需要具有易扩展的特点，例如将错误处理部分看作一个错误处理器，具有随时随地在各函数中收集记录错误，在分析完毕后集中打印错误的功能，需要自定义可扩展的数据结构，并写好灵活的函数接口。

**工作二：语法语义分析+符号表管理+存储管理+中间代码生成**

在完成词法分析器和错误处理接口后，接下来的任务首先是设计文档，再是语法分析考核，因此在这一段时间里ddl感并没有那么重，而这也是提前写程序的好机会。

首先在设计文档阶段本来就需要考虑这四部分的内容，虽然看上去一共有四个部分，内容比较繁重，但实际上符号表管理+存储管理均是接口性内容，真正要花时间写并调试的是语法语义分析和中间代码生成部分，因此需要先把符号表管理、存储管理内容想明白。

对于符号表管理，需要注意的地方在于，我们思考问题不能局限于理论课知识以及前次作业pascal编译器的内容，要从本质出发。符号表的功能是什么？符号表设计需要考虑什么？例如，对比扩充c0文法和pascal文法，很明显二者不是一类语言，因此符号表的设计是否可以采取更加简便的方式？其次，还需要考虑应该采取什么样的数据结构进行符号表实现，C++中是否有高级的数据结构类型可供使用？在这一阶段进行了好的设计，能为后续带来非常大的方便。

对于存储管理，可能会让学生感到疑惑，需要怎么去设计？设计需要考虑什么地方？同理，还是需要抓住问题的本质，存储管理在我们的课程设计中，目标是给目标代码—mips指令提供运行环境以及相关规定，因此在设计运行栈时，应该结合mips汇编程序的特点去想，可以参考计组学习时p2的内容。需要注意的地方有，在c0文法中的变量(局部变量、全局变量、临时变量)，在运行栈中是否需要有空间？除此之外，对于函数的相关信息，对于c0文法来说，有哪些是需要保存的？最后，对于一条计算类中间代码，翻译成mips汇编的整个流程是什么？是否需要取值、存值等等。最后，建议大家参考mips体系结构相关知识，同时也可参考mips运行栈规范。

对于语法语义分析，实际上就是语法制导翻译的直接实现，在这里需要提醒的是，尽可能花时间在语法分析过程就一并把中间代码生成，这样可以减少返工的风险。因此首先对中间代码进行设计，然后写好中间代码接口。

对于中间代码，大家在设计时需要灵活设计，避免给自己找麻烦，例如，对于临时变量表示，可以将名字统一命名为#XX，由于源程序中标识符限定了字符的范围，这样就可以将临时变量与局部变量进行区分，在后续工作中可减少许多工作量。

**工作三：目标代码生成：无优化**

合理存储分配

　　在生成无优化的目标代码时，有两种选择，一种是不对寄存器进行分配，即对于每一变量的使用，如计算类代码，先lw从栈中取出变量值到临时寄存器，进行运算、结果临时寄存器中的值sw存回栈中，即暴力生成，这样在本阶段不涉及优化的时候能够减轻负担；另一种则是进行寄存器分配。我个人支持第一种，首先第一种不涉及寄存器分配，正确性可以得到很好的保证；其次，在本阶段就进行寄存器分配，比较耗时费力，且容易出错，毕竟基础的编译器并没有完成，贸然优化可能事倍功半。

存储寻址

另一个重要的问题是存储寻址问题，即变量存储的地址怎么设计，字符串地址怎么设计等等；对于字符串，很自然的想法是在.data声明后用标签一个一个标识，用.asciiz声明值。这样做的好处是可以简化printf语句转换工作；对于变量存储地址，首先思考mips存储结构，考虑三个寄存器$gp, $fp, $sp各自的功能；然后再将变量分类，与这些寄存器进行对应。这样思考即可设计一个初步的运行栈，不过可以提示的是，是不是这三个寄存器均需要使用到呢？

函数调用

第三个问题是函数调用，对于非函数调用语句，一条一条翻译即可，而对于函数相关语句，由于我们需要维护运行栈，因此需要和栈结构联系起来，有哪些需要维护的量？寄存器值是否需要保存？栈指针如何移动？调用参数怎么传递等等

**工作四：代码优化——常数优化+公共子表达式删除**

接下来进入到代码优化部分，虽然从此阶段开始到期末才会考核优化内容，但中间时间并不是非常宽裕，并且有其他课程的任务，所以还是需要push自己抓紧时间完成优化部分。

首先是常数优化，按照书上算法按部就班实现即可。

其次是公共子表达式删除，这里实际上和书上算法有很大不同，首先书上算法可以说都是计算类指令，而我们的文法显然不只有这些语法成分，因此需要重新考虑。简单的做法是将基本块范围缩小为表达式范围。除此之外，可以参考鲸书上对子表达式的处理。需要注意的是，公共的定义，何时停止向前查看？需要好好斟酌

**工作五：活跃变量分析/定义使用链分析+复写传播+死代码删除**

对于数据流分析，主要有两种方法，一是活跃变量，二是定义使用链分析，实际上mips寄存器足够用，两种方法在本次课设区别并不大，但我还是建议两种都实现一遍，毕竟是为了学习而学。

在实现了数据流分析后，可以进行复写传播和死代码删除，对于后续不使用的变量定义点，可以删去，对于无意义代码，可以删去。

**工作六：循环优化+代码内联**

这里进入到了额外的拓展部分。

对于循环优化，一是如何标识循环块；二是如何确定循环不变量；三是如何执行代码外提操作。

对于代码内联，实际上这一部分对优化的效果提升是最显著的。但适用范围不大，主要是能对足够简单的函数进行内联，但是一旦可内联，则可以显著减少函数调用带来的开销。

**工作七：全局寄存器分配+临时寄存器分配**

全局寄存器分配使用的是书上的图染色算法，重点是冲突图的建立；我使用的是定义使用链分析，因此需要对网-网进行冲突分析，这里需要额外注意，容易踩坑。

临时寄存器分配使用临时寄存器池进行管理，可以对书上算法进行一定拓展，如同样可对临时变量做定义使用链分析。

**工作八：目标代码生成——优化**

终于到了尾声，但这里仍然需要注意一些问题，诸如指令的选取，可以进行多处窥孔优化。

**三、总结**

　　总体来说，这一次课设受益良多，但仍然有可以做得更好的地方。所以还是需要严格按计划执行，不要拖慢进度。总而言之，我认为自己在这次课程设计中在诸多方面得到了提升，一是代码能力，包括代码组织、架构、调试能力；而是程序设计、算法实现能力；三是积累了工程项目开发经验；四是提高了时间管理能力。我认为非常有意义，同时也希望自己的意见能够给学弟学妹带来一定的帮助！