**申优文章**

**15231204 周扬**

本c0编译器采用了自顶向下分析词法语法，利用语法制导翻译技术进行语义分析。其中主要的文件以及其中包含的内容功能如下：程序的入口main.cpp，程序的语法分析部分以及错误处理程序lexical.cpp，程序的语法分析及语法制导的语义分析、中间代码生成：syntax.cpp，符号表管理：symboltable.cpp，生成中间代码的子函数：middlecode.cpp，中间代码生成：mips.cpp，中间代码优化：optimize.cpp。下面我将依据自己搭建程序的过程与心得，谈一谈构建编译器各过程的重点难点。

在词法分析的阶段，书上的算法和方法十分详尽：先依据底层单词的语法，画出单词分割的状态图，再依据状态图进行程序的实现。而在实现中，需要注意的一点便是每一次切出一个单词，字符指针都停在单词的最后一个字符上，这是一种约定，使得词法程序不易出错，这与语法分析程序中，每次进入子函数之前下一个单词约定一定读入了单词缓冲是一个道理。同时，再词法分析阶段就要根据语法建立初步的错误处理程序，而由于错误处理程序报错要清晰明了，所以可以建立一个行缓冲，用于存放当前编译所在行，若出现错误则可直接输出行缓冲；同时要在设立行数变量，遇到换行符的时候行数变量加一，在出错时同时输出。

而在语法分析阶段，基本与词法分析相似，也是由图来知道程序的编写。而愈发程序涉及到了左递归与回溯的问题，在我们的文法中，没有左递归，但是有少量的回溯，比如说变量定义和函数定义，就会出现FIRST集重合的情况。在我们的理论课学习中，一般采用的是改写文法来消除回溯问题，但在我们的文法中却容易出现打乱语义连贯的断语的问题，会为之后的语义分析造成不利影响，加之发现文法中出现回溯的地方最多只需要预读3个单词，便可对语法分支进行明确的判断，代价小而且易维护，因此在我的系统中采用了预读字符的方法。

在语义分析阶段，在之前的语法分析程序的基础上，插入语义分析的子程序，符号表、错误处理和由语法制导翻译，在确定了产生式后进行相应的语法分析，产生相应的中间代码。而在语义分析阶段，最重要的是中间代码的设计，要综合考虑到之后的目标代码生成和源语法。同时，在语义分析阶段，涉及到的综合参数传递的问题，例如在表达式的分析需要向上传递名、类型等参数，在这里的处理我采用的是利用全局变量进行传递，而因为是全局变量，所以要注意全局变量的保护，防止错误参数的传递，因此也有约定：在子函数跳出前才对全局变量赋值，而回到上层函数后立刻取出全局变量的值，将全局变量的有效使用范围进行压缩。

在符号表管理方面，利用vector类存储符号表，且由于c0文法中不涉及动态数组分配的内容，数组的定义方括号内必须为数字，因此数组的长度在编译的第一遍便可以确定，因此在存储管理中我放弃了数组模板，将数组的长度直接记录在符号表中。而在一个函数编译第一遍结束后，由函数本层的符号表，计算出函数的运行栈深度反填回符号表，再删除该层函数的局部变量的符号表信息，这也是与语义分析相结合的部分。

在目标代码生成阶段，我认为最重要也是最难的点就是函数的调用问题，即如何分配函数运行栈，这个问题与变量的存储和函数的递归调用直接挂钩，因此是目标代码正确运行的基础。在这一方面，采取了先由符号表中的函数深度为函数分配栈的大小，在根据符号表中函数的局部变量和临时变量的插入顺序对函数栈进行顺序分配，在变量被使用时，先查符号表计算出相对地址，加上函数的栈地址便可准确定位变量的位置。而这里有一个需要注意的点，那边是全局变量和自己定义的局部变量的区别。若采取在开头MIPS程序数据段定义全局变量，该全局变量的数组是地址向上顺序索引，而自己分配栈则是地址向下顺序索引，若弄混了则会出现栈地址越界导致程序错误的情况。而在子函数调用时，需要在栈中保存上层函数的寄存器信息，在函数返回的时候再从栈中取出。可以说，弄明白了栈的使用方式，目标代码生成的大部分问题也就解决了。

最后代码优化的部分，本系统实现了dag图消除共共子表达式，全局寄存器分配优化。其中全局寄存器分配采用的是数据流活跃变量分析后进行图着色。这一部分树上的算法实际上十分的详细，但在转化为实际程序的时候还是有几个需要注意的点。

在dag图消除公共子表达式这一部分，实现过程中书中并没有强调的是在消除了公共子表达式后，需要保证尚未优化的中间代码和已经优化后的表达式中间代码的相对顺序，因此需要设置记录优化前表达式中间代码的长度的变量和记录优化后表达式中间代码的长度，在最后依据这两个变量，将其他中间代码相应前移，这样便可以保证先对顺序而不会造成错乱。

在全局寄存器分配，需要首先划分基本快，计算出各个基本块的def和use集，随之计算in，out集。而在构建冲突的时候，依据计算的in，out集，通过二维数组来保存两个变量之间的冲突关系，即冲突图，且冲突图需要留存副本，在最后分配时还需要用到。在分配时，依照冲突图先计算每一个变量和其他变量的冲突总数，因为冲突总数影响到冲突图中节点的消去。