参考编译器介绍

我参考的是PL0编译器。

其编译过程采用一趟的扫描方式。以语法、语义分析程序为核心，词法分析和代码生成作为一个过程。当语法分析需要读单词时就会调用词法分析程序。而当语法、语义分析全部正确时，就会调用代码生成程序生成目标代码。

词法分析

词法分析程序能够读入使用PL0语言书写的源程序，其作用为读源程序，输出单词符号串。具有错误分析的功能，可以给出词法错误提示。

语法分析

语法分析的功能是判断和识别由词法分析输出的单词符号串是否符合文法规则。语法分析采用自顶向下的递归下降分析法，对于每个非终结符，都有对应的子程序。从读入的第一个单词开始，当遇到非终结符时就调用相应的处理子程序。若一个PL0语言的单词序列在整个的分析过程中，都能得到匹配则语法分析正确。

符号表管理

符号表中保存三类：1.常量 2.变量 3.过程

常量：如遇常量定义，则将常量写入符号表中，相对地址为-1；

变量：如遇变量定义，则将变量写入符号表中，相对地址为pardr（每个子程序的相对地址从3开始依次累加）；

过程：先保存过程中以变量形式所调用的参数，最后保存地址名，地址依次累加；

代码生成

写语句：将输入的值存入栈顶，再将栈顶元素压入待写入的变量中。

读语句：将待读的变量或常量读出至栈顶。

条件判断语句：对栈顶元素提出进行比较，将比较结果压入栈。

计算语句：对栈顶元素和次栈顶元素取出进行操作，结果压入栈。

编译器总体设计

编译器的工作流程为：词法分析、语法分析、错误处理、中间代码生成、目标代码生成。

词法分析、语法分析将源代码拆分并生成语法树。中间代码生成的步骤为遍历语法树，根据语法树的不同成分生成不同的中间代码，再将中间代码生成目标代码。

--parse 递归下降进行语法分析的文件夹

--quaternion 中间代码四元式的文件夹

--table 符号表、别名表各类表的文件夹

--code 生成中间代码的文件夹

--actuator 生成目标代码的文件夹

主函数在Compiler.java中

词法分析设计

从源代码文件中一行一行读入源代码，记录行数，为错误处理提供帮助。

词法分析过程的本质是根据终结符构造一个有限状态机，实际实现过程中并不需要真正构造出一个DFA，只需要模拟DFA进行贪心匹配即可，用以下方式进行分隔：

- 空格、换行符
- 字母数字下划线和其他字符

对于注释需要特殊处理：

- 单行注释 从 // 开始直至 \n
- 多行注释 从 /* 开始直至 */

整体的状态转移图如下：

语法分析设计

语法分析是递归下降分析语法成分，同时建立语法树的过程。

我在一开始为了偷懒，只进行了递归下降分析语法成分的部分，没有在分析的过程中建立语法树。进行到后面的部分时才补上建树的部分，因此建树的部分设计不太合理。

语法树的节点仅有一个Node类，不同语法成分通过Node类中的类型属性进行区分，以文法的规则构建语法树节点的父子关系。

文法中存在左递归的规则，这对于递归下降分析法来说是不合法的，因此我对文法进行了更改，改掉了左递归文法。

```
LOrExp -->  LAndExp (|| LAndExp)*
LAndExp --> EqExp (&& EqExp)*
EqExp --> RelExp ( (== | !=) RelExp )*
RelExp --> AddExp ( (< | > | <= | >=) AddExp )*
AddExp --> MulExp ( (+ | -) MulExp)*
MulExp --> UnaryExp ( (* | / | %) UnaryExp )*
```

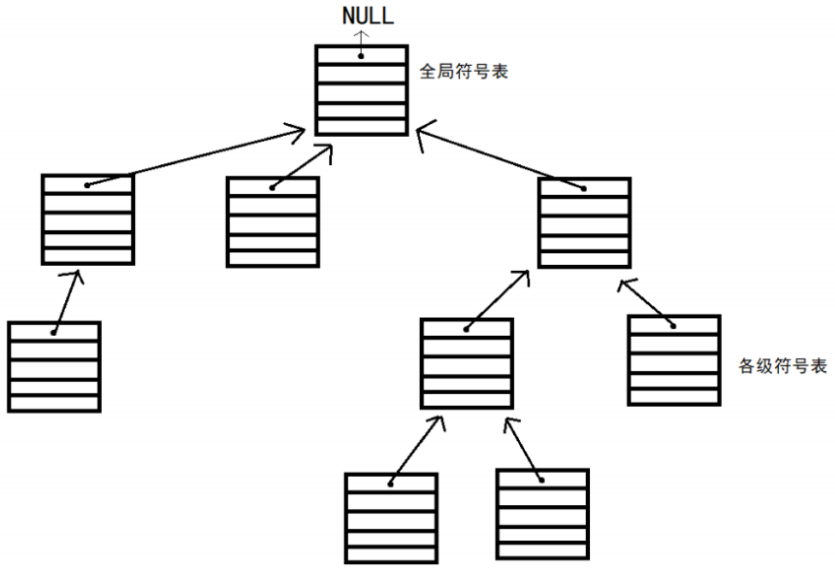
错误处理设计

符号表

错误处理部分，符号表是重点。

我参考编译实验参考手册，设计了单向树结构的符号表。每一级有一张独立的符号表，其父节点直接指向上一级符号表，全局符号表的父节点指向null。这样的设计可以保证在每张符号表上可以查询到其本身及其所有的祖先表。

加入新符号时，仅查询本表中是否有重名变量。而查询符号时，从本表开始逐级向上查询，找到最近的一个即可，很好地满足了内部符号覆盖外部符号的要求。



- FormatString中含非法符号：对每一处FormatString作检查，当检测到非法符号时，即记录错误
- 变量重定义、变量未定义便使用：按照上述建立符号表。
- 函数调用参数个数不匹配：在函数定义时，为函数记录该函数各个参数的类型。在函数调用时对参数进行计数，后对比个数。
- 函数调用参数类型不匹配：参数类型仅有：int、一维数组、二维数组三种情况，且仅需考虑数组维度是否匹配，在传参时对比函数记录中的相应信息。
- 无返回值函数存在return语句：由于void类型函数允许存在return ； 当在void函数中检测到return后应继续判断return后是否存在Exp
- 有返回值函数缺少return语句：直接对函数block的最后一句进行检测，若最后一句不为return或是return，但return后无Exp，即报错
- 修改常量数值：每当对LVal进行赋值时，检查该标识符是否为Const常量。
- 缺少分号、小括号、中括号：在语法分析过程中可顺便处理，当检测到缺少符号时，即记录错误，并补充，以免影响之后的语法分析。
- printf参数不匹配：对FormatString中的%d 和 其后的Exp分别计数，进行比对。
- 非循环体存在break语句或continue语句：设置一个全局变量存储当前是否处于循环体中，若在非循环体中出现break或continue，即记录错误。

最终将记录的错误按行号排序，依次输出即可。

代码生成设计

中间代码我采用的是自行设计的四元式，对于每一种中间代码，都有其处理方式

相加 add

计算两个值的相加结果，MIPS指令中选用add

相减 sub

计算两个值的相减结果，MIPS指令选用sub

相乘 mul

计算两个值的相乘结果，MIPS指令选用mul

相除 div

计算两个值的相除结果，MIPS指令选用div与mflo

取模 mod

计算两个值取模的结果，MIPS指令选用div

输入 getin

读入一个整数，系统调用

输出整数 printInterger

输出一个整数，系统调用

输出字符串 printString

输出一个字符串，系统调用

非 notLogic

对一个值取非，即取其是否为0的反值，MIPS指令选用seq

数组取值 getValue

根据索引从数组中取值，MIPS指令选用lw

跳转 jump

无条件跳转至标签处，MIPS指令选用jump

条件为真时跳转 jumpWhenTrue

当条件变量不为0时跳转至标签处，MIPS指令选用bnez

条件为假时跳转 jumpWhenFalse

当条件变量为0时跳转至标签处，MIPS指令选用beqz

传入参数 pushParam

调用函数时，传入参数。按变量的顺序，sw存入相应的地址处

函数调用 callFunction

调用函数，MIPS指令选用jal

函数返回 return

若函数有返回值，则将返回值存入v0寄存器，从内存中恢复现场，最后 jr $31返回被调用处

内存管理

全局变量与字符串存储在data区。

每个活动记录都以gp寄存器为栈底，fp寄存器为栈顶，向高地址增长。普通变量存储在栈中，编译过程中产生的临时变量存储在fp寄存器的高地址处。

调用和返回函数时需要维护和恢复gp寄存器和fp寄存器。

代码优化设计

#####