# 编译原理最终实验

——PL/0编译器

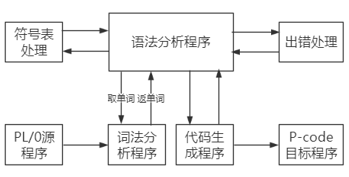
**16211115\_刘天一**

1. 实验目的：

编码实现一个针对PL/0文法的编译器，输入一段PL/0文法的代码，编译器能够将其编译，并输出相应的目标代码Pcode。在代码有语法错误时，编译器能够调用错误处理，指出错误发生的位置并跳过出错部分继续编译。本编译器使用Java语言进行编写，有较良好的图形界面。

1. 实验步骤：
2. 编译器结构

本编译器采用一次扫描，以语法分析部分为核心，语法分析部分调用词法分析器取得单词，在递归子程序执行的过程中间穿插语义分析，进行符号表的填写和Pcode的生成。遇到错误时，调用错误处理打印错误信息。

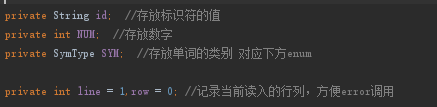


1. 词法分析器 lexica

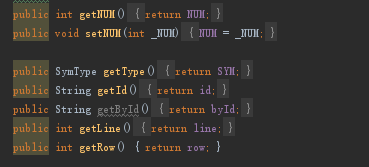
PL/0文法中的字符可分为以下几类

|  |  |
| --- | --- |
| 保留字 | begin,end,if,then,else,const,procedure,var,  do,while,odd,call,read,write,repeat,until |
| 算数运算符 | + | - | \* | / |
| 比较运算符 | <> | < | <= | > | >= | = |
| 赋值符 | := | = |
| 标识符 | Identifier |
| 常数 | 整数 |
| 界符 | , | . | ; | ( | ) |

Lexica使用getSYM()方法来取得下一个token的所有属性，保存在以下几个字段中



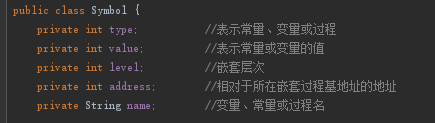
有相应的get方法可以得到属性。



getSYM()调用getch()方法，该方法读取下一个单字符，过滤空格缩进和换行，反复调用getch方法得到一个完整单词后判断该单词的类别，保留字保存在String数组key[]中，用于判断单词是否为保留字。

1. 符号表管理

单个符号储存结构由Symbol类实现，结构定义如下



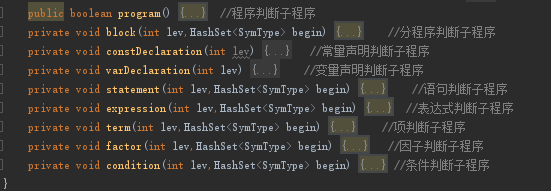
整个符号表由AllSymbol类来实现，



其中enterConst,enterVar,enterProc分别用来将常量，变量，过程添加入符号表。getSymbol方法找到id为name的符号表项并将该项返回。

1. 语法分析器Syntax

编译器才用了递归子程序法进行语法分析，为语法成分分别编写一个分析子程序，根据当前getType返回的符号可以选择相应的子程序进行语法分析。



对于if then [else]，while do和repeat until语句，要生成跳转指令，采用地址返填技术。目标代码生成模式如下：

1. if <condition> then <statement>[else]

<condition>

JPC addr1

<statement>

addr1:[else]

1. while <condition> do <statement>

addr2:<condition>

JPC addr3

<statement>

JPC addr2

addr3:

1. repeat <statement> until <condition>

addr4:<statement>

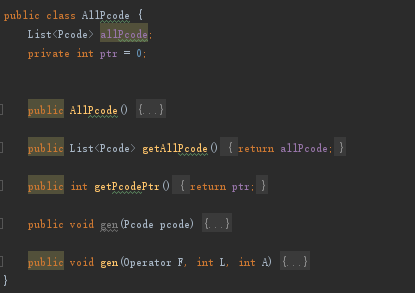
<condition>

JPC addr4

1. Pcode生成

Pcode指令格式形如F L A，其中F表示操作码，可用枚举变量表示；L表示层次差 （标识符引用层减去定义层）；A具体含义不同。Pcode单个指令在Pcode类中实现。

所有的Pcode序列结构由AllPcode类实现，具体方法见代码

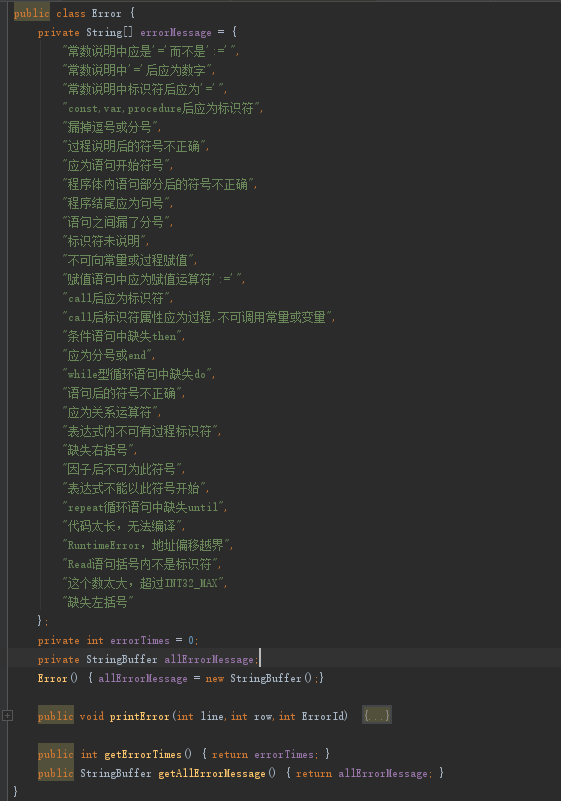


|  |  |
| --- | --- |
| 指令 | 含义 |
| LIT 0, a | 取常量a放到数据栈栈顶 |
| OPR 0, a | 执行运算，a表示执行何种运算(+ - \* /) |
| LOD l, a | 取变量放到数据栈栈顶(相对地址为a,层次差为l) |
| STO l, a | 将数据栈栈顶内容存入变量(相对地址为a,层次差为l) |
| CAL l, a | 调用过程(入口指令地址为a,层次差为l) |
| INT 0, a | 数据栈栈顶指针增加a |
| JMP 0, a | 无条件转移到指令地址a |
| JPC 0, a | 条件转移到指令地址a |

1. 错误处理

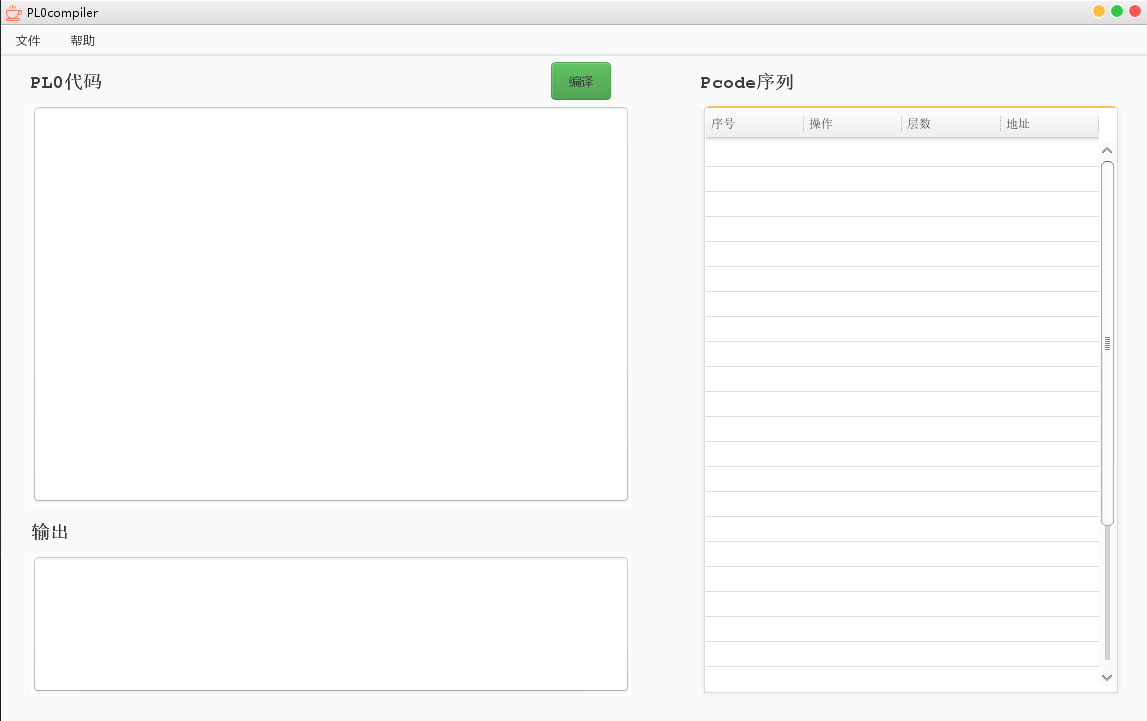
当编译器发现源程序中的错误时，报告错误的类型以及发生的行列，而后跳过错误继续执行编译。对于简单的错误，如赋值符号:=写错，忘记分号等，错误处理可以使得错误被略过，不影响后边的编译。对于其他的错误，采取跳读的方式来处理。跳过一段程序，保证下方的程序段能够正常编译。错误信息放在StringBuffer allErrorMessage中，遇到错误就append进StringBuffer。

错误处理由Error类来实现

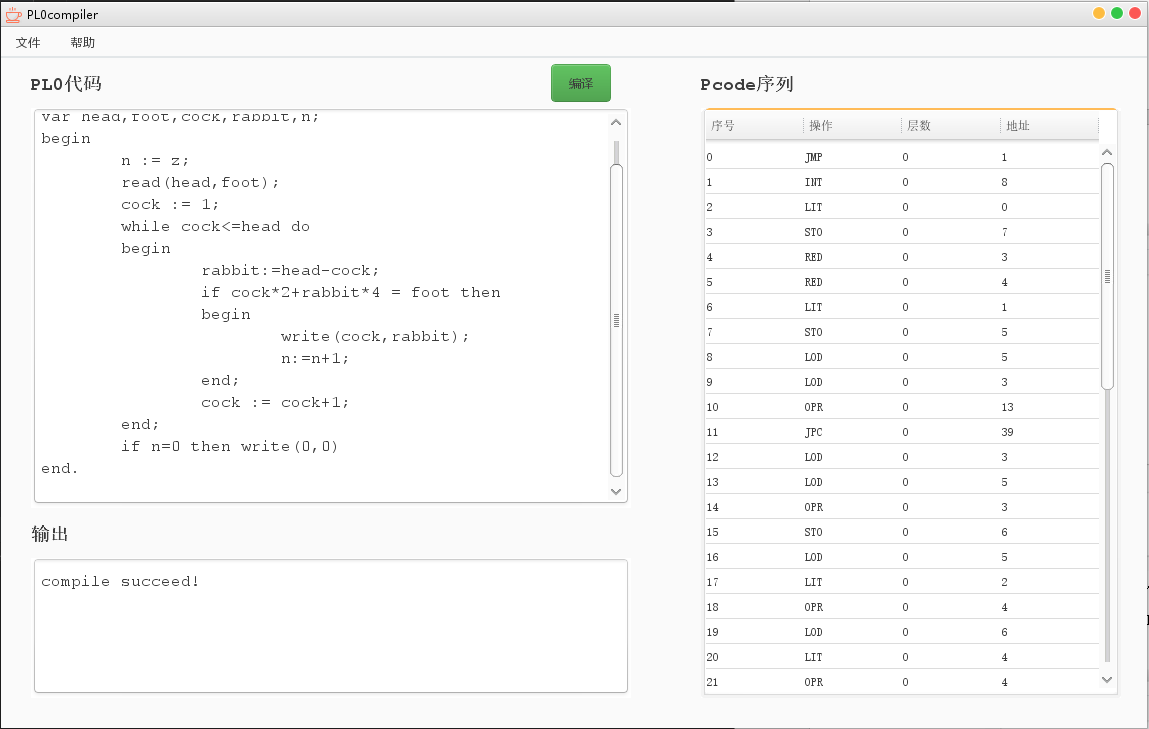


1. GUI界面

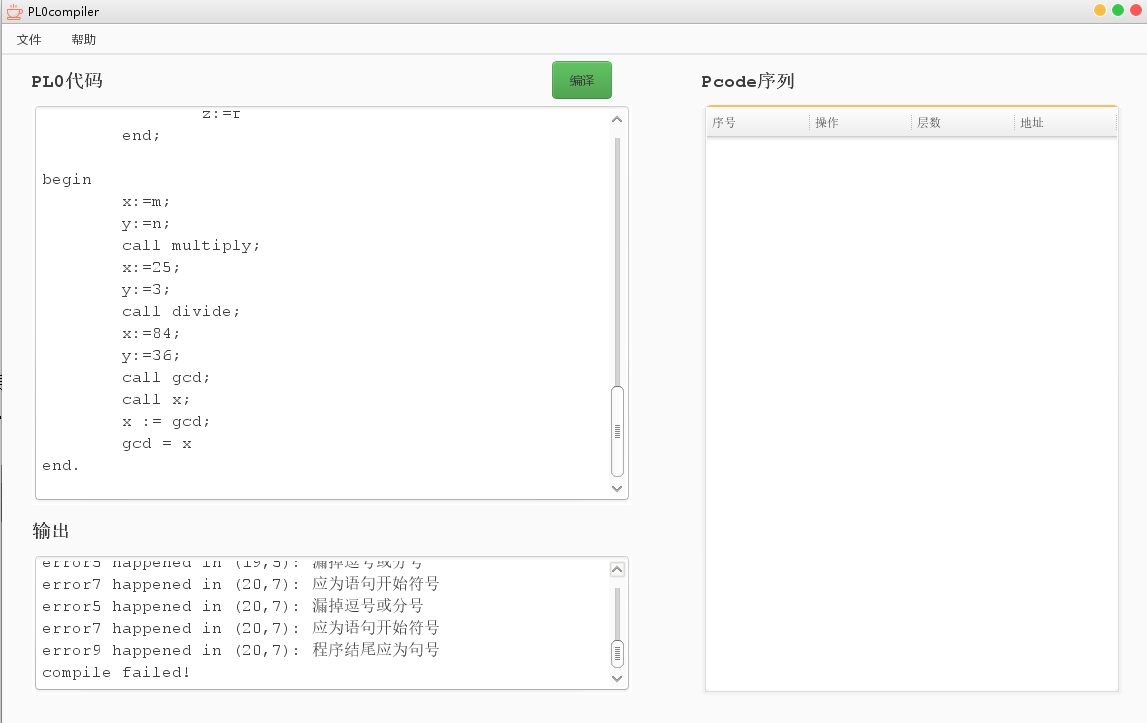
编译器的界面使用java.swing，如图



在代码框内输入代码并保存到本地，或是从文件中选择本地的代码文档即可开始编译，编译成功会提示compile succeed!并在右方给出Pcode序列。如图



编译失败则会提示并在输出框中给出尽可能多的错误提示，如图



1. 编译系统文件说明

|  |  |
| --- | --- |
| 文件名 | 文件描述 |
| AllPcode.java | 实现Pcode序列的结构 |
| AllSymbol.java | 实现整个符号表的结构 |
| Console.java | 整个编译器的起点 |
| Error.java | 保存错误信息和错误数量 |
| Lexica.java | 词法分析器，调用getSYM()读取token |
| MyCompiler.java | UI界面的配置 |
| Operator.java | Enum形式保存了Pcode动作 |
| Pcode.java | 实现了单条Pcode指令的结构 |
| Symbol.java | 实现了符号表中单条项目的结构 |
| Syntax.java | 编译器的核心语法分析器 |

1. 错误样例说明

|  |  |
| --- | --- |
| 样例编号 | 样例描述 |
| 1 | 教材P309例14.1 求解鸡兔同笼问题 |
| 2 | 教材P310例14.2 求最大公约数和最小公倍数 |
| 3 | 教材P310例14.3 打印素数表 |
| 4 | 教材P320例14.7.2 语法错误样例 |
| 5 | if then else测试样例 |
| 6 | 提供的测试样例 |
| 7 | 提供的测试样例 |
| 8 | 提供的测试样例 |

1. 实验感想

在写编译器之前我还以为实现一个编译器很简单，因为上课讲过的内容都听懂了，真正开始写代码的时候才感到无从下手。编译器是一个整体，语法分析的部分要调用词法分析、语义分析还要穿插着符号表的管理以及错误处理。如果不能完整的考虑完整个编译器的实现就直接开始想到哪就写到哪是不行的。所以就只能再把各个部分和他们之间如何配合都仔细想清楚。

所以完成了编译器的实现收获也是很大的，复习了一学期的课程内容，也锻炼了自己的代码能力。