**《编译原理》**

**课程设计说明书**

****

**学 号 1551515**

**姓 名 刘畅**

**专 业 计算机科学与技术**

**授课老师 卫志华**

1. 任务目标

1、掌握使用高级程序语言实现一个一遍完成的、简单语言的编译器的方法。

2、掌握简单的词法分析器、语法分析器、符号表管理、中间代码生成以及目标代码生成的实现方法。

3、掌握将生成代码写入文件的技术。

1. 开发环境

编程语言:Python 2.7、C# 7.0

使用工具: Visual Studio 2017、Pycharm 2018

操作系统:Windows 10,版本17134

运行环境:.Net Framework 4.6.1

1. 程序功能描述

实现了C- -语言的词法分析、对所用文法LR(1)分析表的建立、语法分析、语义分析（包括语义栈、符号表、函数表、中间代码生成以及写入文件，方法为一遍扫描）、MIPS汇编生成及写入文件。

支持源代码保存、文法产生式显示、LR(1)分析表显示、分析过程展示、中间代码及MIPS汇编展示。

1. 程序实现过程
   1. **主要算法与模块**
2. 词法分析：

正则表达式匹配算法:

我借鉴了工具lex的词法分析算法，以正则表达式匹配的方法代替DFA进行词法分析，通过6个正则表达式并行匹配词法元素，每一次将位于开头的最长匹配序列识别为对应的正则表达式所代表的token，从而实现了词法分析。

1. LR(1)语法分析:

求FIRST集算法

由于文法中产生式非终结符的FIRST集存在相互依赖关系，使用递归法会导致计算量的巨大，因此我使用了自下而上求解FIRST集依赖的算法:对于每个产生式左侧非终结符T，若其FIRST依赖于另一非终结符F，则将F加入到其FIRST集中。将所有FIRST集中含有非终结符的非终结符加入到列表Q中，当Q不为空时，对每个元素进行遍历:若其依赖的非终结符不在Q中，则将其替换为它的FIRST集中的终结符，当其FIRST集中依赖的非终结符均被替换后，执行出队操作。达到了自底向上求解依赖关系的效果。

求CLOSURE(I)算法

CLOSURE(I)表示和I中项目可以识别同样活前缀的所有项目的集合。它可以有以下方法得到：

(1)I中的所有项目都属于CLOSURE(I)；

(2)若项目[A→a.Bβ,a]属于CLOSURE(I)，B→ξ是一个产生式，那么，对于FIRST<βa>中的每一个中介符b,如果[β→.ξ,b]原来不在CLOSURE(I)中，则把它加进去；

(3)重复执行步骤（2），直到CLOSURE(I)不再增大为止。

求GO(I,X)算法

GO(I,X)=CLOSURE(J)

其中J={任何形如[A→aX.Β,a]的项目[A→a.X.Β,a]属于I}。

LR(1)项目集族的构建

建立有限状态自动机DFA、哈希表H、项目集队列P，放入初始项目集I0，存入哈希表中并作为DFA的初始状态。取出队首元素I，对于I的每个项目X，求I’=GO(I,X)，若I’不在哈希表中，则将其加入P和H中，并添加为DFA新的状态。为DFA添加一条边(I,X,I’)。循环此操作直到P为空为止，DFA即代表了文法G的LR(1)项目集族，为下一步构造LR(1)分析表提供了依据。

LR(1)预测表的构建

(1)若项目[A→·a, b]属于Ik且GO(Ik,a)＝Ij，a为终结符，则置ACTION[k, a]为“sj”。

(2)若项目[A→·a]属于Ik，则置ACTION[k,a]为“rj”；其中假定A→为文法G的第j个产生式。

(3)若项目[S→S·,#]属于Ik，则置ACTION[k,#]为“acc”。

(4)若GO(Ik，A)＝Ij，则置GOTO[k, A]=j。

(5)分析表中凡不能用规则1至4填入信息的空白栏均填上“出错标志”。

③语义分析

语义栈

LR语义分析是在语法分析的基础上，增加一个语义栈，栈内元素为语义结点。结点类是S属性文法的表示，判别每次语法分析所使用的产生式，实现不同的语义动作，每当规约到特定非终结符时，即可产生中间代码。

符号表和函数表

每次规约识别出一个新的标识符，都会将其加入符号表中，符号的信息包括标识符、中间变量名、类型、占用空间、内存偏移量、作用的函数等。而当规约到函数定义的时候，则将函数名、形参列表、代号加入函数表。

中间代码生成

分为赋值语句、算术运算语句、函数调用语句、循环语句、选择语句、跳出语句、函数定义语句等。

* + 1. MIPS汇编生成

寄存器分配算法

使用了简单的寄存器分配思路，用一张中间变量表记录中间变量的位置：在内存中还是在寄存器中，用一张寄存器表记录8-26号寄存器的占用情况。当有空闲寄存器时，任意分配；当寄存器全部用完时，执行清理操作，取出一个下文出现次数最少的中间变量放入内存，释放其占用的寄存器。

函数调用的翻译

借鉴了8086汇编的子程序调用方法，将内存偏移量为0x8000的地址设置sp和fp栈指针的起始地址,fp栈向高地址增长，sp栈向低地址增长。fp栈用于传递函数调用的形参（我并没有使用a0-a4寄存器传递参数）；sp栈用于处理函数调用时，保护ra与形参。当需要进行函数调用时，首先将当前函数形参与ra压入sp栈，然后查询函数表，将待传递的形参依次压入fp栈。函数调用返回后，从sp栈中推出ra与当前函数形参。

* 1. **基本框图**

语法分析器

词法分析器

语义分析器

中间代码生成

MIPS翻译器

提供Token

确定语法动作

执行语义动作

文本输入

中间代码输出

MIPS输出

已达Accept状态

* 1. **功能函数**

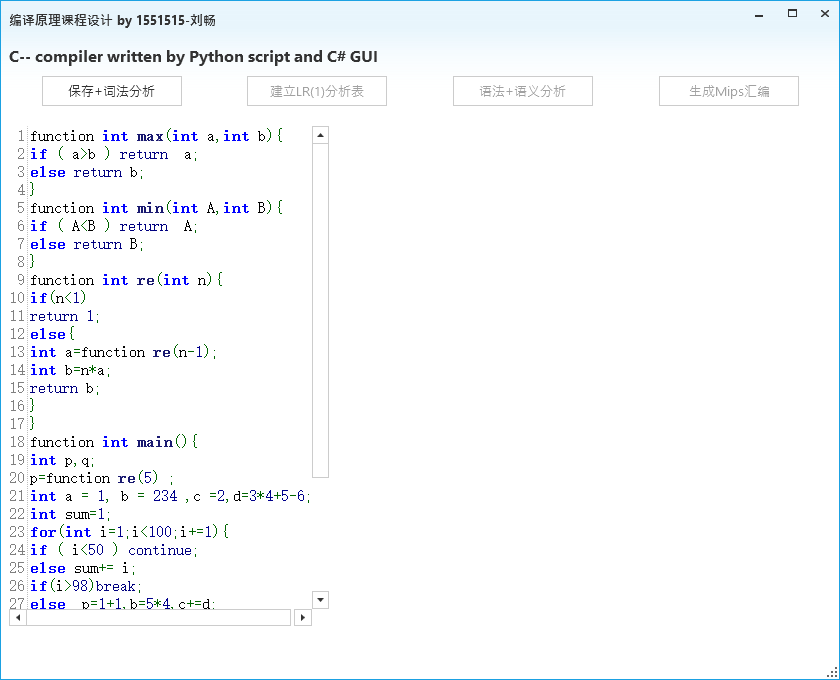
1. LexcialAnalyzer.py文件

|  |  |
| --- | --- |
| 函数定义 | 注释说明 |
| **def** RemoveComments(data) | 对读取的字符串去除注释 |
| **def** Scan(line) | 经行一次扫描，返回得到的token以及剩余的字符串 |
| **def** ScanLine(line) | 对一行进行重复扫描，获得一组token |
| **def** main(path) | 读取文件，返回其中识别的全部token |

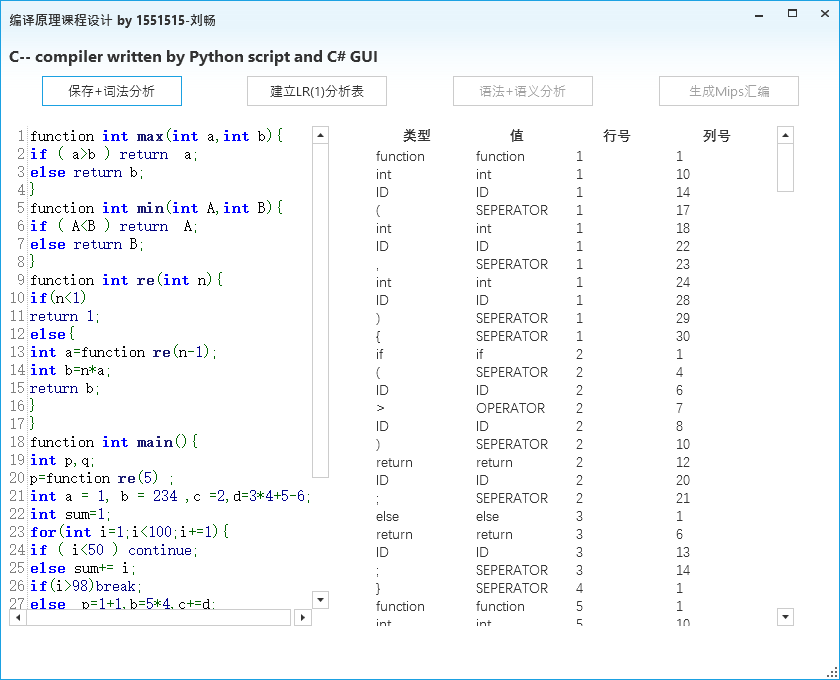
1. MipsGenerate.py文件

|  |  |
| --- | --- |
| 函数定义 | 注释说明 |
| **def** FindSymbol(name, function) | 根据所在函数及标识符，找到符号表中的符号 |
| **def** UpdateSymbolTable(symbol) | 更新或者插入符号表 |
| **def** FindFunctionByName(name) | 根据函数名找到函数表中的函数 |
| **def** UpdateFunctionTable(symbol) | 更新或者插入函数表 |
| **def** ReadGrammer(file) | 读取LR(1)文法 |
| **def** AddDotToproductions(production) | 对产生式加点 |
| **def** GenerateDotedproductions() | 获得所有加点的产生式 |
| **def** FindProduction(NT) | 在所有加点的产生式中，找到其中左侧非终结符为NT的 |
| **def** CLOSURE(productions) | 求一个项目集的CLOSURE |
| **def** GO(I,item) | 求一个项目集对于item的GO |
| **def** GetFirstSet(symbol) | 初步获取First集 |
| **def** MakeUpFirst() | 补全Fisrt集 |
| **def** GenerateFirst() | 产生First集 |
| **def** GenerateDFA() | 构造LR(1)项目集规范族的DFA |
| **def** SearchGoToState(I,target) | 查找GO(I,X)所到达的项目集 |
| **def** GenerateTable() | 生成LR(1)分析表 |
| **def** PrintTable() | 将LR(1)分析表写入文件 |
| **def** MakeAnalyse() | 进行语法分析 |
| **def** NewLabel() | 生成一个新的lable |
| **def** NewFunction() | 生成一个新的函数lable |
| **def** NewTemp() | 生成一个新的中间变量lable |
| **def** SemanticAnalysis() | 语义动作子程序 |
| **def** PrintIntermediateCode(codes) | 将中间代码写入文件 |
| **def** GetReg(temp,codes) | 为中间变量申请一个寄存器 |
| **def** FreeReg(codes) | 释放寄存器占用 |
| **def** GenerateMips(codes) | 翻译中间代码为MIPS |

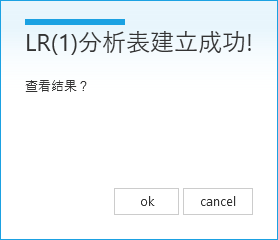
1. 执行界面与运行结果



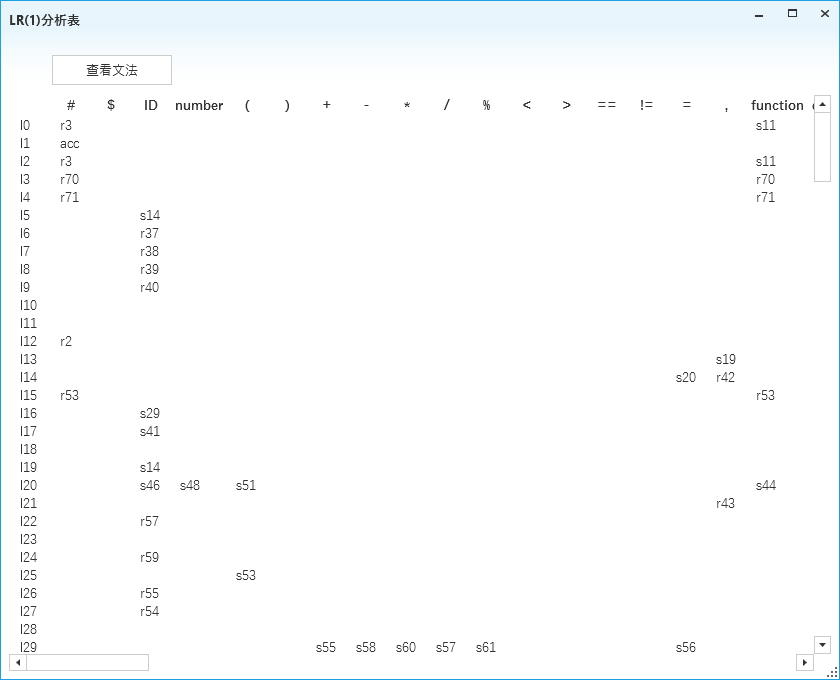
程序初始状态，可在左侧代码编辑框内写入C—代码



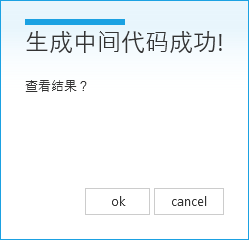
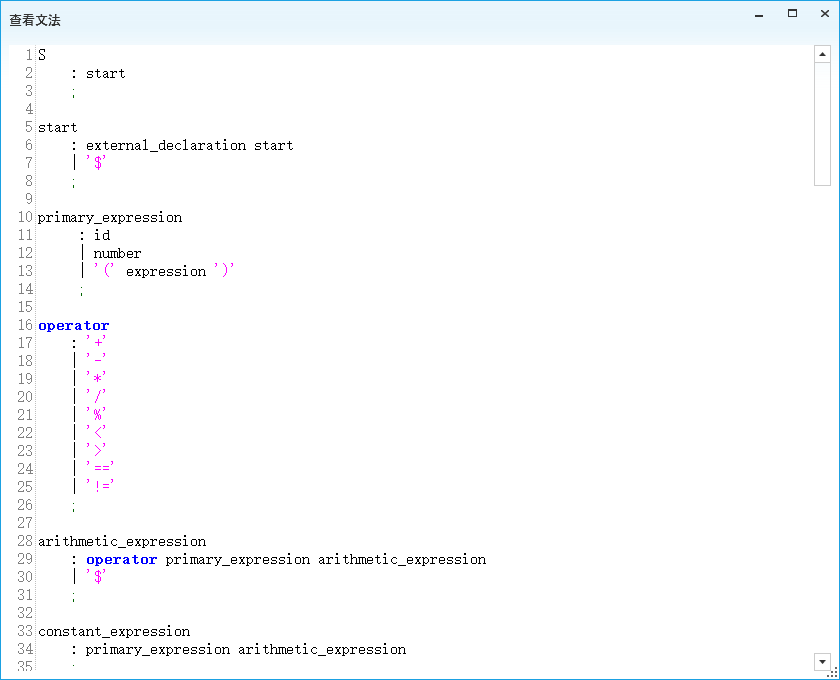
点击**词法分析**按钮，会在右侧显示词法分析的结果，同时“建立LR(1)分析表”按钮会变为可用状态。



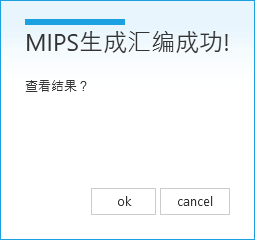
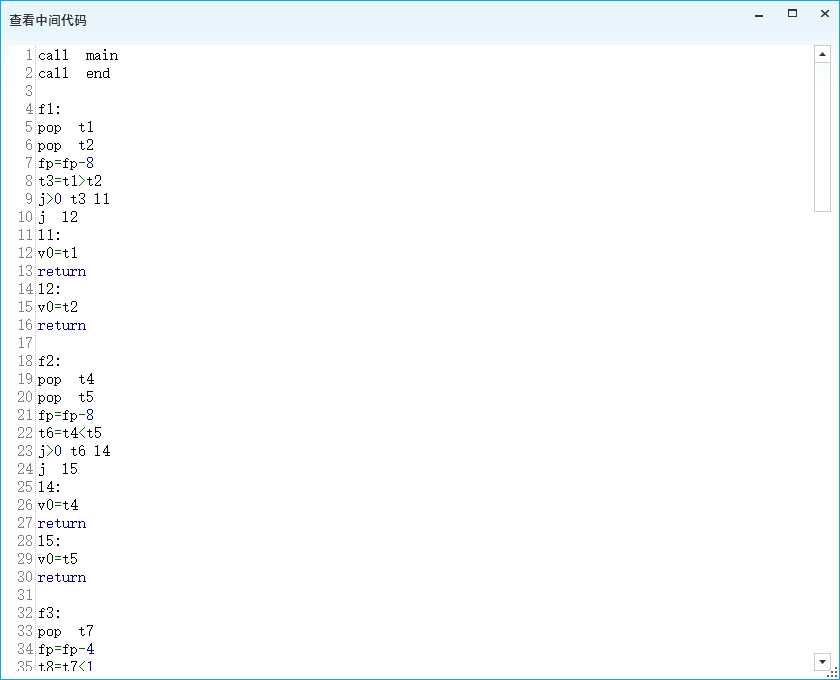
点击**建立LR(1)分析表**按钮会进行LR(1)分析表的建立，成功后点击对话框的ok按钮，会打开LR(1)分析表查看窗口



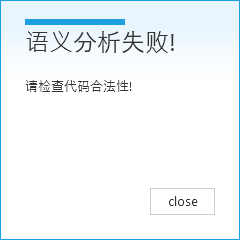
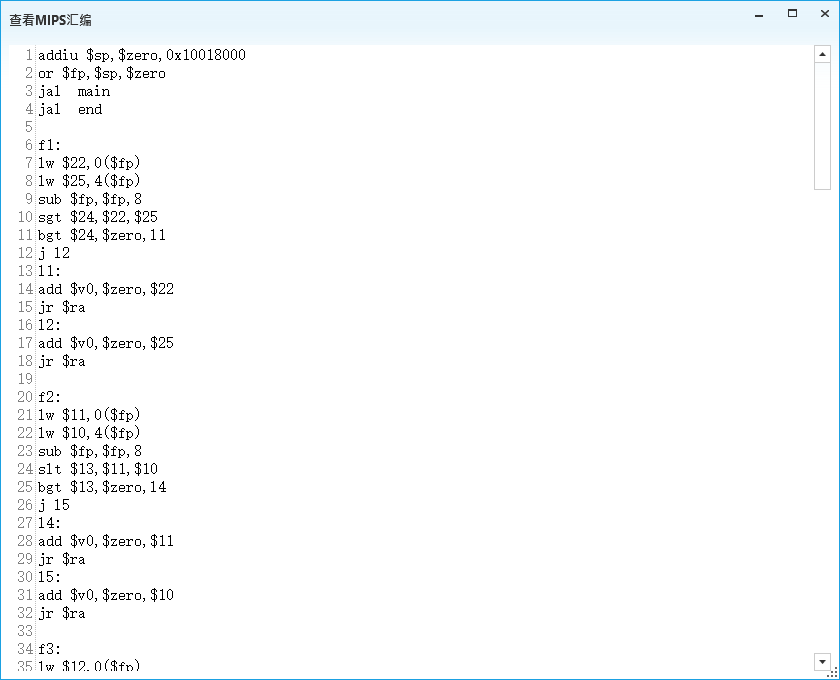
点击**查看文法**按钮，会打开文法查看窗口



点击**语法+语义分析**按钮，会进行中间代码生成，成功后点击ok按钮，会打开分析过程查看窗口和中间代码查看窗口



点击**生成MIPS汇编**按钮，会进行MIPS翻译，成功后点击ok按钮，会打开MIPS汇编查看窗口，至此，完成了C—语言的翻译



当语法分析或者语义分析识别到错误时，弹出错误窗口，提醒检查代码合法性

1. 问题和解决的方法、心得体会

在课程设计的过程中，我遇到了极大的困难。总共历时一个月，才完成了一个基本的编译器。原计划完成的很多特性都没有实现：

* 1. 除了int类型，double、float、char等均未实现
  2. 未实现运算符优先级
  3. 未实现数组
  4. 未实现布尔表达式
  5. break与continue实现不完美
  6. 语义分析不检查错误

主要因为基础功能的实现耗费了大量的时间，没有空余来完成原定的特性，可以说一个支持众多语法特性的编译器不能一蹴而就。

困难在编译器的每个阶段均有发生，其中语法分析最难完成。首先是要找到可用的语法产生式，我借鉴了[ANSC C语言 yacc文法产生式](http://www.quut.com/c/ANSI-C-grammar-y-1995.html)，在此基础上对其进行了大量的删减与改造。在此过程中，我首先采用了LR(0)语法分析，但是由于出现了非常多的shift-reduce冲突，我谋求使用SLR分析法，但是在求Follw集的过程中，出现了循环依赖的现象。只能采用LR(1) ，最终得到了一个可用的LR(1)语法分析器。在构造LR(1)分析表的过程中，CLOSURE算法非常难以实现，这是LR(1)分析法最大的难点。而在分析过程中，对于形如NT->’$’的推出空的产生式，语法动作应该为规约，但是状态栈应该增加一个状态，那就是GO(I,X)所到达的状态，这是书本上所没有的。

在语义分析的过程中，最大的难点是如何实现一遍扫描。根据教科书的指导，LR分析需要扩充语法分析器，增加一个语义栈，这样就不需要构造语法树，而是在生成中间代码的同时，将语义栈结点出栈，利用S属性文法传递生成的中间代码

。而中间代码的生成，则是分为不同种类语句的翻译，其中闭包表达式（算术运算语句，声明语句，赋值语句，语句块闭包）的翻译由于必须一遍扫描+语义栈，我只能定义一个stack属性文法，每当出现闭包规约的时候，就把被规约的token加入到结点的栈中，直到能确定闭包结束时，再将token翻译为中间代码。

而break和continue由于上下文联系非常强，并没有找到很完美的翻译方法，我的处理是这样的：每当规约到break或continue，保留一个占位符，等待二次翻译。当规约到循环语句时，检查属性文法中传递来的中间代码中有没有占位符，识别为break或者continue后，结合自身翻译为j next或者j begin。

MIPS汇编代码最大的难题是函数调用的翻译，在前面已经说过解决的思路了，就不再多言。

通过此次编译原理课程设计我非常深入地理解了从形式语言到汇编代码的编译过程，获得了很多知识。