**编译原理课程实践-语法分析器**

**学 院：**

**专 业：**

**班 级：**

**姓 名：**

**学 号：**

**指导教师：**

1. **实验目的**

BITMiniCC实验的目的是指导和帮助学生通过实践环节深入理解与编译实现有关的形式语言理论基本概念，掌握编译程序构造的一般原理、基本设计方法和主要实现技术，并通过运用自动机理论解决实际问题，从问题定义、分析、建立数学模型和编码的整个实践活动中逐步提高软件设计开发的能力。

此次实验是BITMiniCC实验的语法分析部分，其目的是利用高级语言设计并实现一个语法分析程序，加深对语法分析原理的理解。

1. **实验内容**

**语法分析的任务要求：**

该实验选择 C 语言的一个子集，基于 BIT‐MiniCC 构建 C 语法子集的语法分析器，该语法分析器能够读入 XML 文件形式的属性字符流，进行语法分析并进行错误处理，如果输入正确时输出 XML 形式的语法树，输入不正确时报告语法错误。

需要说明的是，能够分析的输入程序依赖于选用的语法子集，而输出的语法

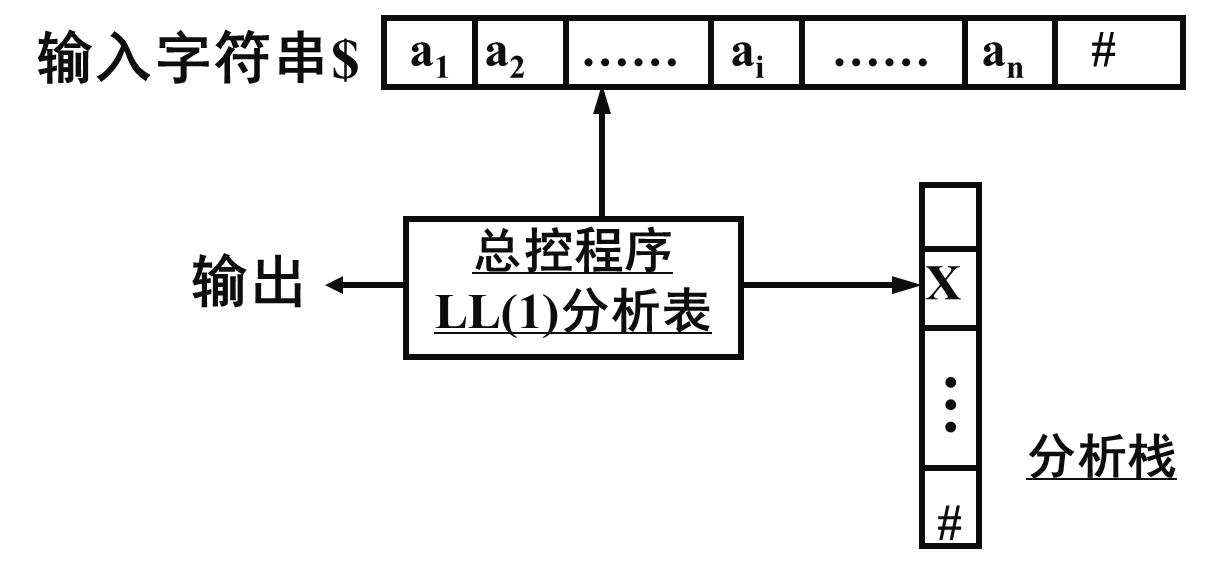
树的结构又与文法的定义密切相关。

文法要求包括全局变量声明，循环语句、分支语句、函数调用语句以及 switch 语句等。要求至少包括局部变量声明语句、赋值语句、返回语句、一种分支语句（if, if‐else, switch 等）和一种循环语句（for, while,do‐while 等）。

1. **实现的具体过程和步骤**

**3.1实验的理论基础**

本次实验采用自上而下的LL（1）分析法。LL（1）分析法的实现是由一个总控程序控制输入字符串在一张LL（1）分析表和一个分析栈上运行而完成语法分析任务的，所以一个LL（1）分析器的逻辑结构如下图所示，由总控程序、LL（1）分析表和分析栈三个部分组成。



**3.2设计C语言的文法子集**

本实验的C语言文法包括变量的声明语句、赋值语句、简单的运算语句、返回语句、分支语句（if-else）、循环语句（for）。

C语言if-else和for循环的文法

if语句

<if\_statement>::=<KW\_IF><TK\_OPENPA><expression>

<TK\_CLOSEPA><statement>[<KW\_ELSE><statement>]

for语句

<for\_statement>::=<KW\_FOR><TK\_OPENPA><expression\_statement>

<expression\_statement><expression><TK\_CLOSEPA><statement>

根据C语言if-else和for循环的文法和老师给出的文法子集以及网络资料后设计出自己的C语言子集的文法如下：

<程序>-----------><函数>

<函数>-----------><函数声明><函数序列>

<函数序列>-------><函数>| 空

<函数声明>-------><数据类型><标识符>左括号<参数>右括号<函数体>

<参数>-----------><参数声明><参数声明序列>| 空

<声明>-------><数据类型> <标识符>

<参数序列>--->逗号 <参数声明><参数序列>| 空

<函数体>---------> 左大括号 <陈述序列> 右大括号

<陈述序列>-------><陈述语句><陈述序列>| 空

<陈述语句>-------><表达式语句>|<返回语句>|<for循环语句>|<if语句>

<表达式语句>-----><表达式>分号

<表达式>---------><修饰符><表达式列表>|<参数声明序列>

<修饰符>--------->左括号<表达式>右括号 | 标识符 | 常量

<表达式列表>----->等于号<修饰符><表达式> | 加号 <修饰符><表达式> |

减号 <修饰符><表达式>| 大于号 <修饰符><表达式>|小于号 <修饰符><表达式>

<参数声明>-------><声明> <参数声明序列>|空

<参数声明序列>--->逗号 标识符 <参数声明序列> | 空

<返回语句>------->返回关键字 <表达式语句>

<for循环语句>--->for 左括号 <表达式语句> <表达式语句> <表达式语句> 右括号 左大括号 <陈述序列> 右大括号

<if语句>-------->if 左括号 <表达式语句> 右括号 左大括号 <陈述序列>右大括号 <else-if语句>

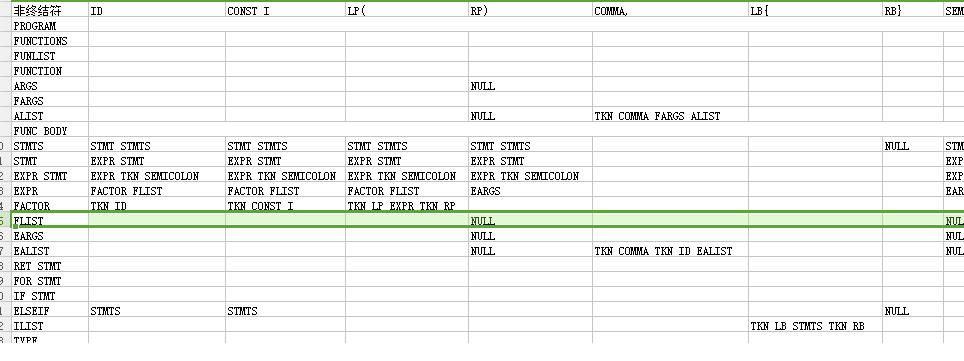
<else-if语句>--->else <if语句序列> | 空

<if语句序列>----><if语句>| 左大括号 <陈述序列> 右大括号

<数据类型>------>int|float|bool |double

**3.3根据文法求出LL（1）分析表**

根据上述文法验证文法的正确性后求出每个非终结符相应的FIRST集合或FOLLOW集合，并根据FIRST集合或FOLLOW集合求出LL（1）分析表。LL（1）分析表的部分截图如下图所示，完整的LL（1）分析表保存在提交文件的doc目录下的LL（1）分析表.excel文件中。



表中最左侧的一列为非终结符，表中第一行为终结符，表中的每一项为产生式的右部。

**3.3 输入字符串与LL（1）分析栈**

本实验中的输入字符串为从上一级词法分析的生成的XML文件中读取出的属性字流，属性字流包括单词所在行数、单词值、单词类型等信息，由每一个单词构成LL（1）分析的输入字符串。LL（1）分析栈即为从文法开始符号PROGRAM起，根据LL（1）分析表及输入字符串栈顶元素选择适当的产生式，根据产生式进行相应的出栈和入栈操作来进行分析。当LL（1）分析栈栈顶元素与输入字符串栈顶元素都为非终结符且相等时，栈顶元素同时出栈，不相等时则出错。当LL（1）分析栈栈顶元素与输入字符串栈顶元素都为“#”时分析完成，acc。

**3.4java语言的具体实现说明**

实现语言 java JDK 1.8

**3.4.1用到的外部jar包**

本实验对于xml文件的解析通过外部jar包jdom-2.0.6、org.w3c.dom、org.xml.sax进行解析。

**3.4.2主要的class及数据结构简要说明**

**Class：**

1. Word.java 主要用于保存从xml文件中读取的属性字流中单词的各种属性
2. AnalyseNode.java 主要用于表示LL（1）分析栈中的元素即保存终结符和非终结符的相关信息。
3. State.java 主要定义了22个AnalyseNode类型的非终结符以及终结符并初始化。
4. Errorinformaintion.java 用于保存在分析过程中的错误信息，错误信息包含错误类型、错误的个数、错误单词、错误所在的行数。
5. Maincontrol.java 此类为LL（1）分析的主控程序，负责更新LL（1）分析栈和输入字符栈的内容，并以LL（1）分析表为基础进行语法分析，若分析成功，输出XML格式的分析树，若出错则给出错误提示。
6. MiniCCParser.java(implements IMiniCCParser) 此类重写了run方法，并调用Maincontrol中的方法进行LL（1）语法分析。

**数据结构：**

实验中输入字符串用ArrayList<Word>实现，LL（1）分析栈用Stack<AnalyseNode>实现，错误信息的存储用ArrayList<Errorinformaintion>实现。

**四、运行效果截图**

**4.1配置文件**

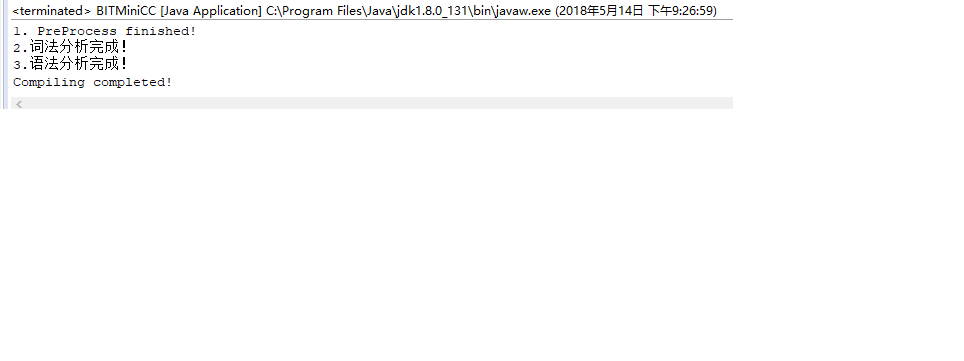
将文件中的config.xml配置文件的词法分析部分的路径进行修改



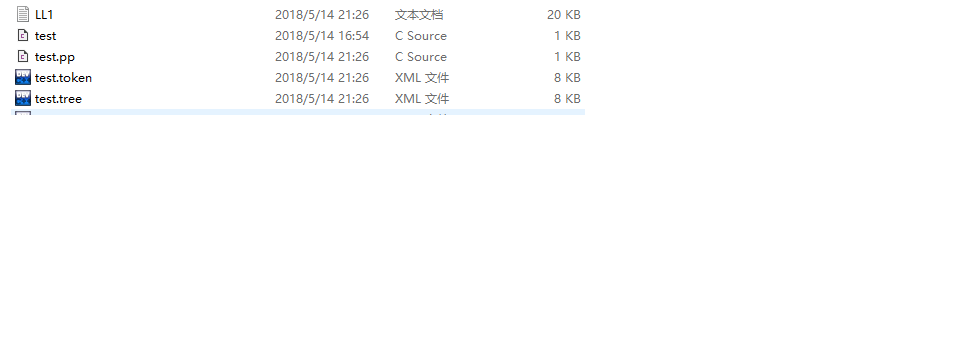
**4.2测试运行**

在eclipse内运行结果：

第三行中文输出：3.语法分析完成！ 为自己实现的语法分析器输出结果。



在input文件夹中可看到生成的test.tree.xml文件以及分析过程中产生的LL1.txt



**test.pp.c的测试代码**：

int main ( ) {

int i ;

int a ;

for ( i = 0 ; i != 10 ; i = i + 1 )

{

a = a + 1 ;

}

if ( a )

{

a = 0 ;

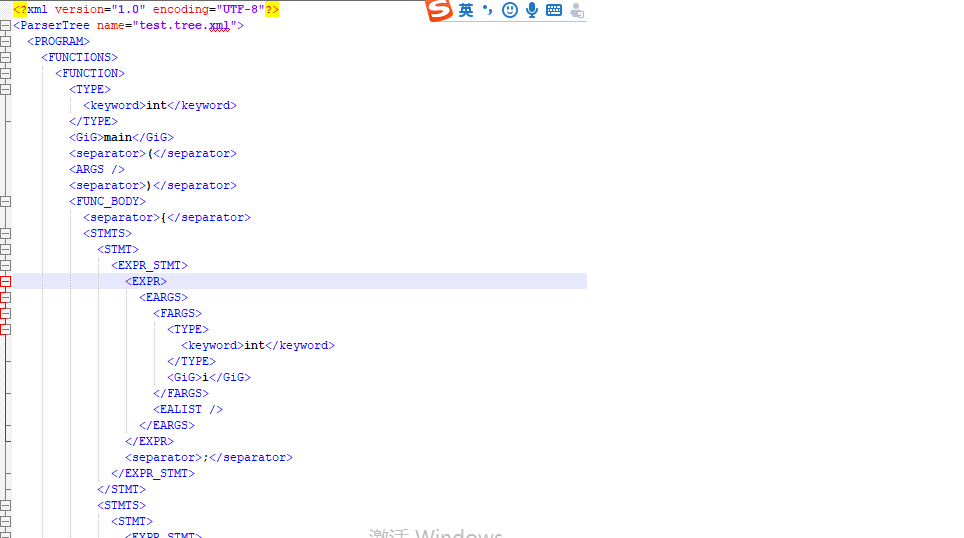
}

a = 0 ;

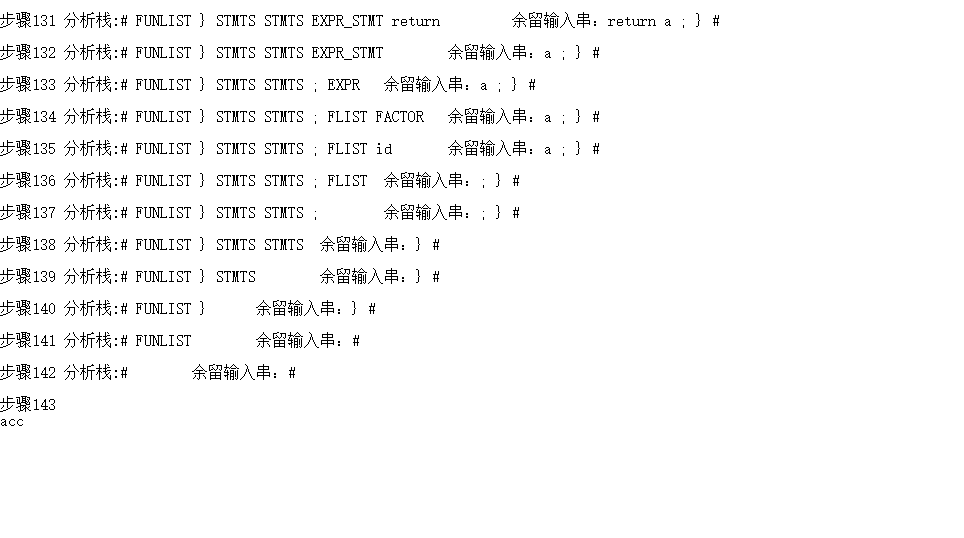
return a ;

}

**生成的语法分析树部分内容截图：**



生成的LL1分析表：



将C语言代码修改出语法错误，语法分析程序显示的错误信息如下：



C语言代码中有一个变量未定义错误和一个语法错误。

由于乐学网站对于上传的文件大小有限制，故删除了run文件夹中的BITMiniCC.jar以及lib文件夹中的jar包。

**五、实验心得体会**

本次实验为语法分析器的实现，通过实验掌握了语法分析的基本知识，加深了对语法分析原理的理解。此次实验是一次从理论上到实践的尝试，我选择的是自上而下的LL（1）分析法。在编译原理课程的第四章学习完LL（1）分析方法的相关内容时对于LL（1）分析法的FIRST子集以及FOLLOW子集的作用以及求法理解的不够透彻，求解不够熟练。在实验中我根据老师参考文法设计C语言的文法子集，对于文法的左递归，最左公因子问题有了更深入的认识。在设计好文法后就到了LL（1）分析方法的最核心的部分即LL（1）分析表的构造，通过对每个非终结符求FIRST子集以及FOLLOW子集后得出了一个很大的LL（1）分析表，LL（1）分析表的求解以及实现是整个程序中工作量最大的部分，也是对于LL(1)分析方法以及语法分析理解初见深入的一个过程。

实验使用了BIT-MINICC框架，此框架对于编译的各个步骤的中间文件都有严格的限制。语法分析的输入文件为词法分析的.xml格式的属性字流，输出位.xml格式的语法树。在词法分析实验中我采用了最原始的read和write对.xml文件进行读写，工作量很大，所以在本实验中由于java有很多的开源jar包，我利用了一些专门解析.xml文件的jar包对.xml文件进行了解析，不仅降低了工作量还是思路更加的清晰明了。

本次实验让我对C语言从其文法层面上有了更深层次的认识，对于编译原理课程中的理论知识有了更深层次的认识，实现了理论到实践的尝试。此外还锻炼了自己利用java语言的编程能力，提升了我对于java语言的掌握程度。在实验过程中感受到了BIT-MINICC框架的强大、便利对我学习编译原理有很大的帮助。