**YACC项目说明文档**

**学 校：华南师范大学**

**学 院：计算机学院**

**班 级：2015级软件工程4班**

**学 号：20152100027**

**姓 名：詹萍**

**课 程：编译原理**

目录

[一、题目分析 3](#_Toc503727742)

[1、使用Java语言实现一个简单的Yacc 3](#_Toc503727743)

[2、输入的文法应符合BNF语法定义 3](#_Toc503727744)

[3、分析输入的单词流是否符合文法 3](#_Toc503727745)

[二、实现思路 3](#_Toc503727746)

[1、输入BNF文法 3](#_Toc503727747)

[（1）选择存储结构 3](#_Toc503727748)

[（2）输入处理 4](#_Toc503727749)

[2、求解First集 4](#_Toc503727750)

[（1）First集的存储 4](#_Toc503727751)

[（2）求解First集的算法 4](#_Toc503727752)

[3、求解Follow集 5](#_Toc503727753)

[（1）Follow集的存储 5](#_Toc503727754)

[（2）求解Follow集的算法 5](#_Toc503727755)

[4、构造预测分析表 6](#_Toc503727756)

[（1）预测分析表的存储 6](#_Toc503727757)

[（2）求解预测分析表的算法 7](#_Toc503727758)

[5、输入单词流 7](#_Toc503727759)

[6、语法分析 7](#_Toc503727760)

[（1）所需的存储 7](#_Toc503727761)

[（2）非递归下降分解的算法 8](#_Toc503727762)

[三、代码结构 8](#_Toc503727763)

[1、代码整体框架 8](#_Toc503727764)

[2、类Yacc的属性 9](#_Toc503727765)

[2、类Yacc的主要方法 9](#_Toc503727766)

[四、实验小结 10](#_Toc503727767)

# 一、题目分析

## 1、使用Java语言实现一个简单的Yacc

yacc是一个经典的生成语法分析器的工具。

yacc的输入是BNF表达的语法规则以及语法规约的处理代码，Yacc输出的是基于表格驱动的编译器，包含输入的语法规约的处理代码部分。

本题要求使用LL(1)文法实现语法分析。

## 2、输入的文法应符合BNF语法定义

程序读取一个BNF语法定义文件，然后构造一个Java语言的LL(1)分析器（如果输入BNF定义的文法不是LL(1)文法则报错）。

BNF文法有如下基础要求：

1. 没有出现在任何产生式左端的<…>或者出现在双引号"..."内的是终结符。
2. 在产生式左端出现过的<…>是非终结符。
3. ""代表空单词ε。

## 3、分析输入的单词流是否符合文法

这个LL(1)分析器能够判断一个输入的单词流是否符合BNF文法。如果单词流符合文法，输出“yes”，否则输出“no”。

即对于一个输入的单词流，这个LL(1)分析器应该给出一个结果“yes”或者“no”。

# 二、实现思路

## 1、输入BNF文法

### （1）选择存储结构

在输入BNF文法这里需要存储的有非终结符号、终结符号以及产生式。

为了方便之后的存取，我采用了ArrayList分别对非终结符号和终结符号进行存储。这个存储结构的好处在于数据是按照存储的顺序排列的，方便之后构造预测分析表，只需要根据某个非终结符号和终结符号在ArrayList中的位置即可其知道对应的表项的位置。

对于产生式的存储，我看到网上很多方法都是采用ArrayList二维数组进行存储，但由于我在进行输入处理的同时也将产生式存储起来，因此没办法马上知道产生式的长度，而二维数组的建立需事先确定长度。因此我采用了ArrayList嵌套ArrayList的方法（ArrayList<ArrayList<String>>），避开了对长度的求解。

### （2）输入处理

对于文法符号，在输入时需要判别获取的符号是非终结符号、终结符号还是空串。

对于每一条产生式，将其左侧的符号加入到非终结符号集（判别是否重复加入），将其右侧的所有符号加入到终结符号集。输入结束后，再将非终结符号以及空串从终结符号集删去。

对于产生式，一条产生式对应于一个ArrayList<ArrayList<String>>外层的项。

举个例子：对于产生式<E>::=<T><A>，存储之后为<E> <T> <A>。

## 2、求解First集

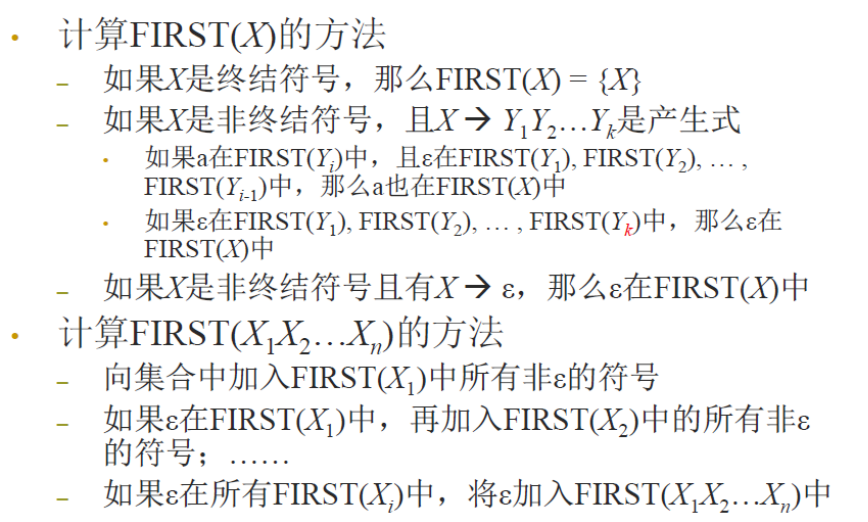
### （1）First集的存储

采用哈希表（HashMap<String,HashSet<String>>）进行存储，其中key值是非终结符号，value值是一个哈希集合，保存该非终结符号的First集。

采用键值对的存储方便找到非终结符号对应的First集，采用HashSet存储First集可以避免符号重复的情况。

### （2）求解First集的算法

根据龙书讲到的如下算法：



在程序实现时，依次求解每个非终结符号的First集。

对于某个非终结符号，先要找到其对应的所有产生式，再依次处理每一条产生式。

对于非终结符号<E>的每一条产生式，有如下情况：

1. 遇到的符号是空串""，将其加入到First集中。
2. 遇到的符号是终结符号<a>，将其加入到First集中，并结束对本条产生式的求解。
3. 遇到的符号是非终结符号<A>，需要递归求解<A>的First集，并将这个First集加入到<E>的First集中。

如果<A>的First集中不存在空串""，则可以结束对本条产生式的求解；如果<A>的First集中存在空串""，则需要对下一个符号进行上述①~③的判断。

## 3、求解Follow集

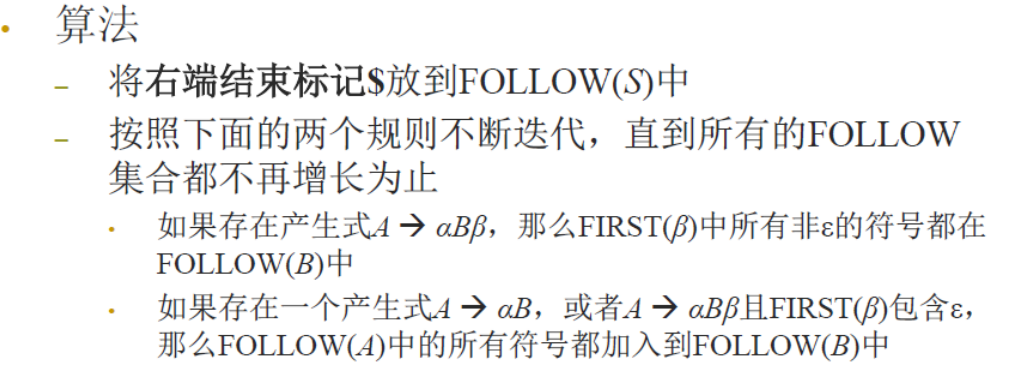
### （1）Follow集的存储

同样采用哈希表（HashMap<String,HashSet<String>>）进行存储，其中key值是非终结符号，value值是一个哈希集合，保存该非终结符号的Follow集。

采用键值对的存储方便找到非终结符号对应的Follow集，采用HashSet存储Follow集可以避免符号重复的情况。

### （2）求解Follow集的算法

根据龙书讲到的如下算法：



在程序实现时，先将$加入到起始符号的Follow集中，再依次处理每一条产生式。只处理产生式右侧的非终结符号。

计算Follow集需要进行多轮计算，直到Follow集没有变化为止。

我采用了整型数组来记录每个非终结符号的Follow集的长度，当所有Follow集长度都没有变化时说明求解结束。

对于一条产生式，左侧的非终结符号为<E>,产生式右侧的非终结符号<A>，有如下情况：

1. <A>是产生式的最后一个符号。

将<E>的Follow集加入到<A>的Follow集中。

1. <A>是产生式的中间的符号。

如果<A>后面是空串""，则考虑下一个符号。

如果<A>后面是终结符号<a>，将其加入到<A>的Follow集中，并结束本轮对<A>的Follow集的求解。

如果<A>后面是非终结符号<B>，将<B>的First集除去空串""以外的符号加入到<A>的Follow集中。如果<B>是产生式的最后一个符号，并且<B>中存在空串""，则将<E>的Follow集加入到<A>的Follow集中。

## 4、构造预测分析表

### （1）预测分析表的存储

采用二维整型(int[][])数组进行存储，先将预测分析表的表项都初始化为-1（代表error）。

基于前面使用ArrayList存储非终结符号和终结符号集（可以通过下标获取对应的值），对于分析预测表的某一个表项，我们可以通过下标知道对应的非终结符和终结符分别是什么。

基于产生式也是使用ArrayList存储的，因此可以使用整型数组存储预测分析表，只需要存储产生式的序号即可，对于error的表项，保持值为-1即可。

### （2）求解预测分析表的算法

根据龙书讲到的如下算法：



逐一处理每一条产生式，求取产生式右边部分的First集。在程序实现时，需要对右侧的首个符号求取First集，再判断是否需要处理下一个符号（当前符号可能为空时需要处理下一个符号）。

产生式左侧为<E>，对于右侧的符号，有如下情况：

1. 遇到空串""，处理下一个符号。
2. 遇到终结符号<a>，代表产生式右边的First集即为{<a>}，则将对应的产生式的序号填写到Table[<E>][<a>]中。
3. 遇到非终结符号<A>，获取<A>的First集，将对应的产生式的序号填写到<A>的除去空串""以外的First集对应的表项中。
4. 在上述①~③处理之后，需要判断当前处理的符号是否为产生式最后一个符号，空串""是否可能存在其中，如果存在，说明产生式右侧可能为空。则需要将对应的产生式的序号填写到<E>的Follow集对应的表项中。

当发现已经是非-1的表项重复填入时，说明该文法不适合用LL(1)分析。

## 5、输入单词流

对于输入的单词流，由于一个符号占一行，因此不需要再分割。同样使用ArrayList进行存储。

需要做的处理就是剔除其中的空串""，并且在末尾添加$。

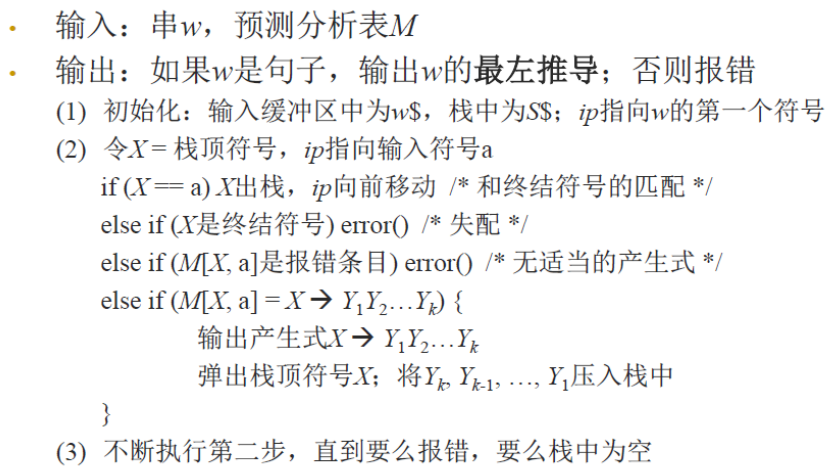
## 6、语法分析

### （1）所需的存储

进行语法分析需要一个语法分析栈，栈顶存放当前进行语法分析的符号。

### （2）非递归下降分解的算法

根据龙书讲到的如下算法：



初始化时依次将$和起始符号<S>压入栈中。并用一个指针来保存对输入单词流的处理进度。

当栈不为空时，执行如下循环：

1. 栈顶元素是终结符号并且是输入单词流当前处理对象，弹出栈顶元素，单词流指针后移。
2. 栈顶元素是其他终结符号，说明该单词流不符合该文法，分析结束。
3. 栈顶元素是非终结符，但该非终结符与当前输入单词对应的表项是-1，即说明该单词流不符合该文法，分析结束。
4. 栈顶元素是非终结符，并且对应的表项不为-1，弹出栈顶元素，将对应的产生式的右侧的非空串""符号从右到左（注意顺序）依次压入栈中。

直到栈中为空，如果没有报错，说明分析成功，该单词流符合文法。

# 三、代码结构

## 1、代码整体框架

程序中主要是实现了一个叫Yacc的类。

类的属性即为存储起始符号、非终结符号、终结符号、产生式、First集、Follow集、预测分析表、单词流、语法分析栈的元素。

类的方法即为实现读入BNF文法、求取First集、求取Follow集、填写预测分析表、读入单词流、进行语法分析的方法，还有一个可以输出上述求解结果的方法。

类中包含一个程序入口main方法，负责顺序调用上述方法。

## 2、类Yacc的属性

//起始符号

static String startSymbol;

//存放终结符

static List<String> Terminal;

//存放非终结符

static List<String> Nonterminal;

//存储每一条产生式的每个符号

static ArrayList<ArrayList<String>> Production;

//存放非终结符号的first集合

static HashMap<String,HashSet<String>> FirstSet;

//存放非终结符号的follow集合

static HashMap<String,HashSet<String>> FollowSet;

//存放预测分析表，填写产生式的序号，用-1表示error

static int [][]AnalyzeTable;

//要进行语法分析的单词流

static List<String> Sentence;

//语法分析栈

static Stack<String> SyntaxAnalyzeStack;

## 2、类Yacc的主要方法

//读入并存储BNF定义

static void readBNF()

计算First集时由getFirstSet()对每个非终结符调用FirstFunction()求解其First集。

//计算非终结符号的First集

static void getFirstSet()

//参数是要计算First集的非终结符号

static void FirstFunction(String nonterminal)

计算Follow集需要进行多轮求解，由getFollowSet()调用FollowFunction()进行求解，其中FollowFunction()进行一轮的求解。

//计算非终结符号的Follow集

static void getFollowSet()

//一轮Follow集的计算

static void FollowFunction()

//填写预测分析表

static void fillAnalyzeTable()

//读入要分析的单词流

static void readSentence()

//进行语法分析

static void syntaxAnalyze()

此外还有一些输出计算结果的函数

# 四、实验小结

本次实验主要是实现一个比较简单的Yacc。

现有的Yacc主要是使用LALR(1)分析，在这次实验中，使用的是语法分析能力低于LALR(1)分析的LL(1)分析。

通过前面课程对于LL(1)分析的学习，本来觉得对LL(1)分析已经比较熟悉了，但是在这次实验中还是遇到了不少的问题。

首先是对BNF文法的处理，需要对输入的产生式进行分割处理，从其中分离出非终结符和终结符。并且对每一条产生式，将符号一个个存储。在选择存储结构时一开始我是使用键值对储存符号的，并且没有将终结符号和非终结符号分开存储的，只是用true和false来区分他们。但是在后来进行处理时发现不太方便，才又更改了存储结构。

其次在求解时，发现书上的算法转化成程序设计语言还是需要考虑各种问题的。常常会不小心忽略了对空串的处理。但是通过一步步的处理，一步步调试，也最终可以解决问题。

目前我只是实现了比较简单的LL(1)分析的Yacc，可能其中也还存在其他问题，所以我也还在陆续的完善中。