njdx

151250002@smail.nju.edu.cn

Abstract

本次词法分析实验实现了对LL（1）文法分析

词法分析实验报告

卞纯源 151250002

目录

[变更历史 3](#_Toc498445412)

[1、 实验目的 3](#_Toc498445413)

[2、 实验内容 3](#_Toc498445414)

[3、 实验方法 3](#_Toc498445415)

[4、 假设 3](#_Toc498445416)

[5、 相关分析过程 4](#_Toc498445417)

[5.1自定义的文法 4](#_Toc498445418)

[5.2消除左递归与左公共因子后的文法 4](#_Toc498445419)

[5.3构造First和Follow集 4](#_Toc498445420)

[5.4构造LL(1)分析表 5](#_Toc498445421)

[6、 重要数据结构 5](#_Toc498445422)

[7、核心算法 7](#_Toc498445423)

[8、 输入与输出 8](#_Toc498445424)

[9、 问题与解决 8](#_Toc498445425)

[10、感受 8](#_Toc498445426)

# 变更历史

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **修改人员** | **变更原因** | **版本号** |
| 2017/11/14 | 卞纯源 | 文档初稿 | V0.0 |

# 实验目的

本实验的目的是在词法分析程序的基础上，运用语法分析方法对输入的语句进行分析，并输出推导过程。

# 实验内容

本实验主要采取自顶向下的语法分析方法，根据文法构造预测分析表，再根据预测分析表进行语法分析。程序由Java编写。程序读取一个文本文件，运用词法分析程序得到TOKEN序列，对其进行语法分析，最后输出产生式序列。词法分析器基于lab1实现的lex修改re而得

定义的具体文法有：

1. 一般赋值表达式
2. if-else语句

该文法可识别以上两种文法的组合

# 实验方法

1. 自定义要分析的文法
2. 对文法进行预处理（消除左递归与二义性）
3. 构造预测分析表
4. 基于分析表进行语法分析
5. 语法分析模块：根据输入队列当前读头下的字符与状态栈的第一个元素进行分析，若是终结符则进行匹配，否则产生子项，循环至处理至输入队列最后一个字符

# 假设

1. 程序中所有变量都为id，所有常数都为number
2. 为避免二义性，if后都有else，并且if>;>else

# 相关分析过程

## 5.1自定义的文法

(0)S→id=E;S| if(C){S}else{S}S|ε

(1)E→E+T | T

(2)T→T\*F | F

(3)F→(E) | id | number

(4)C→C||D | D

(5)D→(C) | id==number

## 5.2消除左递归与左公共因子后的文法

(0) S→id=E;S

(1) S→if(C){S}else{S}S

(2) S→ε

(3) E→TE’

(4) E’→+TE’

(5) E’→ε

(6) T→FT’

(7) T’→\*FT’

(8) T’→ε

(9) F→(E)

(10) F→id

(11) F→number

(12) C→DC’

(13) C’→||DC’

(14) C’→ε

(15) D→(C)

(16) D→id==number

## 5.3构造First和Follow集

|  |  |  |
| --- | --- | --- |
| **Symbol** | **First** | **Follow** |
| S | { id, if, ε} | { $ , } } |
| E | { ( , id , number } | { ; , } } |
| E’ | { + , ε } | { ; , } } |
| T | { ( , id , number } | { + , ; , ) } |
| T’ | { \* , ε } | { +, ) , ; } |
| F | { (, id, number) | { \* , + , ; , ) } |
| C | { (, id } | { ) } |
| C’ | { || , ε } | { ) } |
| D | { (, id } | { || , ) } |

## 5.4构造LL(1)分析表

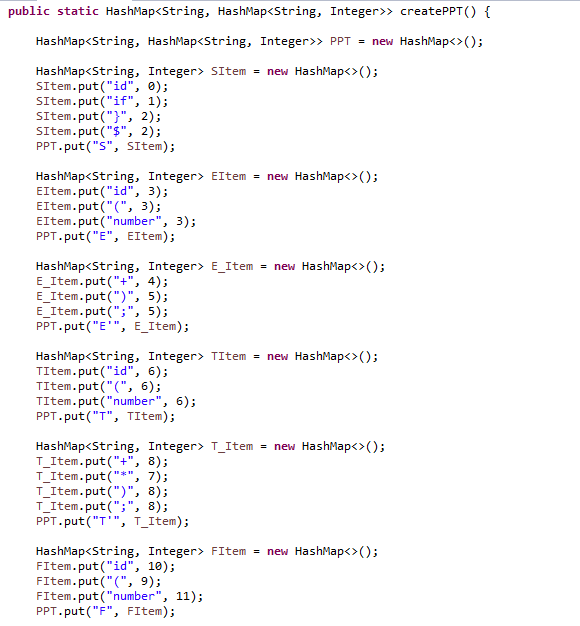
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Vn\Vt** | **id** | **if** | **+** | **\*** | **(** | **)** | **{** | **}** |
| **S** | **0** | **1** |  |  |  |  |  | **2** |
| **E** | **3** |  |  |  | **3** |  |  |  |
| **E’** |  |  | **4** |  |  | **5** |  |  |
| **T** | **6** |  |  |  | **6** |  |  |  |
| **T’** |  |  | **8** | **7** |  | **8** |  |  |
| **F** | **10** |  |  |  | **9** |  |  |  |
| **C** | **12** |  |  |  | **12** |  |  |  |
| **C’** |  |  |  |  |  | **14** |  |  |
| **D** | **16** |  |  |  | **15** |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vn\Vt** | **else** | **number** | **||** | **==** | **;** | **$** |
| **S** |  |  |  |  |  | **2** |
| **E** |  | **3** |  |  |  |  |
| **E’** |  |  |  |  | **5** |  |
| **T** |  | **6** |  |  |  |  |
| **T’** |  |  |  |  | **8** |  |
| **F** |  | **11** |  |  |  |  |
| **C** |  |  |  |  |  |  |
| **C’** |  |  | **13** |  |  |  |
| **D** |  |  |  |  |  |  |

# 重要数据结构

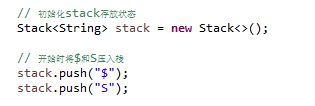
1. LL（1）分析表

以硬编码的方式将手动生成的LL（1）分析表存入HashMap中：



1. Stack

由一个Stack对象保存状态，并初始化栈，将$和S压入栈中



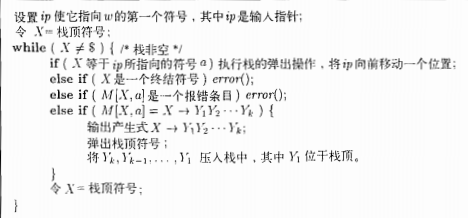
1. Token

由实验一写好的词法分析器进行词法分析并返回token

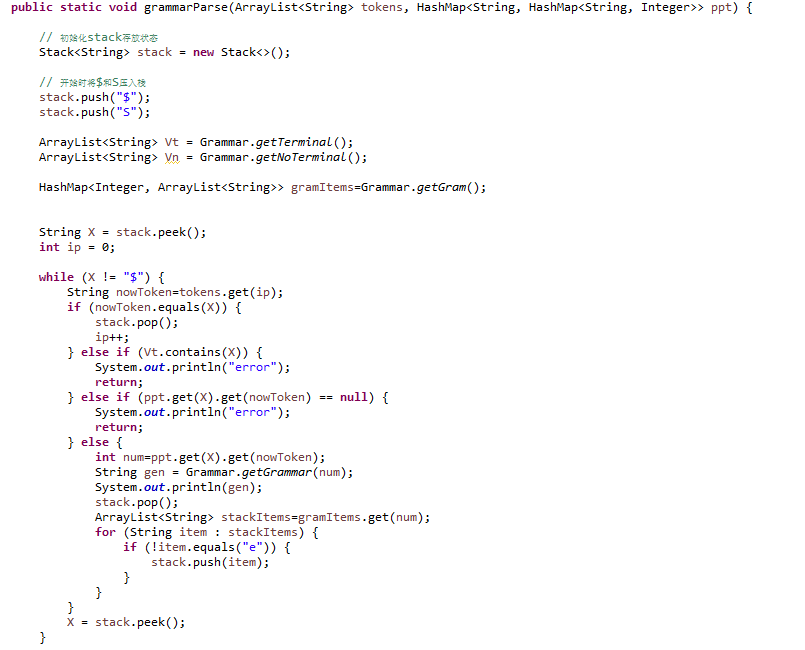


# 7、核心算法

本程序的核心算法为grammarParse方法。

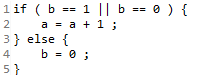


具体程序如下：

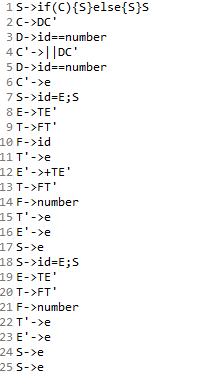


# 输入与输出

1. input.txt



1. output.txt



# 问题与解决

1. 在构造PPT时，数据结构的选择一直没有想好。最后考虑到查找的方便性，决定用HashMap
2. 在输入文法对应的入栈符号的时候输入错误两处，导致一开始的输出都会出现error

# 10、感受

总体来说，这次实验算法上并不困难，但非常考验一个人的细心与耐心。硬编码非常耗费时间，并且容易错误。由于时间原因，没有像lab1一样构造自己的yacc，没有能够全部完成从产生式到语法分析的部分，等考试周结束了，想再尝试一下YACC的编写。