编译原理实验报告

语法分析程序

141210026

宋奎熹

软件学院

1. 实验目的

编写、调试一个语法分析程序，并对程序段进行语法分析，从而更好理解语法分析原理。此次生成 LL(1) 分析表。

2. 内容描述

此程序用 Java 编写。程序读取一个文本文件，根据所定义的CFG文法（见5），对其中内容进行语法分析。并得出相应的产生式列表。

3. 思路方法

1） 根据要识别的文法，写出 CFG。

2） 构造其对应的预测分析表。

3） 根据预测分析表，编写代码。

4） 代码实现：用 Lab1 的词法分析工具生成 Token 序列，并将其输入文本文件中，然后读取此文本文件，基于 CFG 的预测分析表，得出产生式列表。

4. 假设

假设输入的文件内容是正常的 Java 程序，即包含合法的保留字和运算符。if 语句后都有唯一的 else 语句。id 代表变量，num 代表数字。

5. CFG 定义

本次定义的语言可识别如下形式的语句：

1) while(**CONDITION**){**STATEMENT**};
2) if(**CONDITION**){**STATEMENT**}else{**STATEMENT**}
3) **CONDITION** && (**CONDITION** && **CONDITION**) (只有&&运算符)
4) id = num

最终 CFG 定义如下：

0. S → id=E;
1. S → if(C){S}else{S}
2. S → while(C){S}
3. E → TE′
4. E′ → +TE′
5. E′ → ε
6. T → FT′
7. T′ → *FT′
8. T′ → ε
9. F → (E)
10. F → num

11.F → id
12.C → DC′
13.C′ → &&DC′
14.C′ → ε
15.D → (C)
16.D → id==num

6. 预测分析表

	id	if	else	(	)	{	}	while	+	*	num	&&	==	;	=	$
S	0	1						2								
E	3			3							3					
E′					5				4					5		5
T	6			6							6					
T′					8				8	7				8		8
F	11			9							10					
C	12			12												
C′					14								13			14
D	16			15												

7. 关键数据结构

```
private ArrayList<Token> tokenArrayList = new ArrayList<>();
private static Stack<Token> tokenStack = new Stack<>();
private static Queue<Token> tokenQueue = new LinkedList<>();
private static String[] generations = {
            "S->id=E;",
            "S->if(C){S}else{S}",
            "S->while(C){S}",
            "E->TE'",
            "E'->+TE'",
            "E'->ε",
            "T->FT'",
            "T'->*FT'",
            "T'->ε",
            "F->(E)",
            "F->num",
            "F->id",
            "C->DC'",
            "C'->&&DC'",
            "C'->ε",
```

```
            "D->(C)",
            "D->id==num"
    };
private static int[][] ppt = new int[][]{
            { 0, 1,-1,-1,-1,-1,-1, 2,-1,-1,-1,-1,-1,-1,-1,-1},
            { 3,-1,-1, 3,-1,-1,-1,-1,-1,-1, 3,-1,-1,-1,-1,-1},
            {-1,-1,-1,-1, 5,-1,-1,-1, 4,-1,-1,-1,-1, 5,-1, 5},
            { 6,-1,-1, 6,-1,-1,-1,-1,-1,-1, 6,-1,-1,-1,-1,-1},
            {-1,-1,-1,-1, 8,-1,-1,-1, 8, 7,-1,-1,-1, 8,-1, 8},
            {11,-1,-1, 9,-1,-1,-1,-1,-1,-1,10,-1,-1,-1,-1,-1},
            {12,-1,-1,12,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
            {-1,-1,-1,-1,14,-1,-1,-1,-1,-1,-1,13,-1,-1,-1,14},
            {16,-1,-1,15,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1},
    };
private ArrayList<String> output = new ArrayList<>();
```

8. 样例输出

输入样例：

```
while(x==0&&y==1){
      if(y==3&&(y==3&&y+2==14)){
            x=x+2*y;
      }else{
            y=y+2*(4+x);
      }
}
```

输出样例：

```
-----------------------------------
Beginning of output.
-----------------------------------
Tokens :
-----------------------------------
Token : <8,while>
Token : <4,(>
Token : <1,x>
Token : <13,==>
Token : <11,0>
Token : <12,&&>
Token : <1,y>
Token : <13,==>
Token : <11,1>
```

```
Token : <5,)>
Token : <6,{>
Token : <2,if>
Token : <4,(>
Token : <1,y>
Token : <13,==>
Token : <11,3>
Token : <12,&&>
Token : <4,(>
Token : <1,y>
Token : <13,==>
Token : <11,3>
Token : <12,&&>
Token : <1,y>
Token : <9,+>
Token : <11,2>
Token : <13,==>
Token : <11,14>
Token : <5,)>
Token : <5,)>
Token : <6,{>
Token : <1,x>
Token : <15,=>
Token : <1,x>
Token : <9,+>
Token : <11,2>
Token : <-1,->
Token : <1,y>
Token : <14,;>
Token : <7,}>
Token : <3,else>
Token : <6,{>
Token : <1,y>
Token : <15,=>
Token : <1,y>
Token : <9,+>
Token : <11,2>
Token : <-1,->
Token : <4,(>
Token : <11,4>
```

```
Token : <9,+>
Token : <1,x>
Token : <5,)>
Token : <14,;>
Token : <7,}>
Token : <7,}>
Token : <16,$R>
-----------------------------------
Generators :
-----------------------------------
Generator : S->while(C){S}
Generator : C->DC'
Generator : D->id==num
Generator : C'->&&DC'
Generator : D->id==num
Generator : C'->ε
Generator : S->if(C){S}else{S}
Generator : C->DC'
Generator : D->id==num
Generator : C'->&&DC'
Generator : D->(C)
Generator : C->DC'
Generator : D->id==num
Generator : C'->&&DC'
Generator : D->id==num
-----------------------------------
End of output.
-----------------------------------
```

9. 问题及解决

在编程过程中出现过中途无法继续解析的情况，后来发现是将“=”与“==”的命名混淆了，以后要多加细心。通过此次编程，对“CFG 生成 LL(1) 分析表”的整个过程有了更深刻的了解，收获颇丰。