编译原理实验二实验报告

学号：171250592

姓名：王有鑫

1. 实验目的

自己定义方法，运用所学的语法分析方法对输入的语句进行语法分析并输出结果，加深对所学内容的理解和运用。

2. 内容描述

程序通过从控制台读取字符序列，以实验一的词法分析器为基础，再对其进行语法分析（基于LL（1）进行自顶向下分析，输出产生式序列。

3. 方法

- 自定义文法
- 消除左递归
- 提取最大左公因子
- 构造预测分析表
- 基于实验一的词法分析程序，以PPT为基础编写语法分析程序

4. 假设

1. 文法中的变量名为$(a|b)^*$, 数字为(0|1)^*（实验一时为了方便画图的处理）
2. 为避免二义性，if后面都有else
3. if、else、while后的语句均有{}，以区分程序块
4. 比较运算符仅以`<`作为代表

5. 相关过程

1. 自定义文法如下：
    0. S->id=E
    1. S->if(C){S}else{S}
    2. S->while(C){S}
    3. E->E+T|T
    4. T->T*F|F
    5. C->F<F
    6. F->num|id
2. 消除左递归及提取最大公共左因子之后的文法：
    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->num
10. F->id
11. C->F<F

3. 构造的预测分析表如下：

	id	=	if	(	)	{	}	else	while	+	*	num	<	$
S	0		1						2					
E	3											3		
E'							5			4				5
T	6											6		
T'	8						8			8	7	8		8
C	11											11		
F	10											9		

6. 重要数据结构

1. `Predictive Parsing Table`

```
private static int[][] p_parsingTable={
    //   id  =   if  (   )   {   } else while +   * num   <   $
    { 0, -1,  1, -1, -1, -1, -1, -1,  2, -1, -1, -1, -1, -1},   //S
    { 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  3, -1, -1},   //E
    {-1, -1, -1, -1, -1, -1,  5, -1, -1,  4, -1, -1, -1,  5},   //E1
    { 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  6, -1, -1},   //T
    { 8, -1, -1, -1, -1, -1,  8, -1, -1,  8,  7,  8, -1,  8},   //T1
    {11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 11, -1, -1},   //C
    {10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  9, -1, -1}    //F
};
```

2. `Stack`

```
import java.util.ArrayList;

public class Stack {

    private ArrayList<Integer> stack;

    public Stack(){
        stack = new ArrayList<Integer>();
        stack.add(Token.tokensMap.get("DOLLARS"));
    }

    public void push(int t){
        stack.add(t);
    }

    public void pop(){
        stack.remove(stack.size() - 1);
    }
```

```
19
20      public int get(){
21          return stack.get(stack.size() - 1);
22      }
23
24  }
```

3. `Token`

```
1   import java.util.HashMap;
2   import java.util.Map;
3
4   public class Token {
5       public static Map<String, Integer> tokensMap=new HashMap<>();
6       public Token(){
7           tokensMap.put("VOID", 0);
8           tokensMap.put("MAIN", 1);
9           tokensMap.put("ID", 2);
10          tokensMap.put("INT", 3);
11          tokensMap.put("ASSIGN", 4);
12          tokensMap.put("NUM", 5);
13          tokensMap.put("L_BRACKET", 6);
14          tokensMap.put("R_BRACKET", 7);
15          tokensMap.put("L_BRACE", 8);
16          tokensMap.put("R_BRACE", 9);
17          tokensMap.put("IF", 10);
18          tokensMap.put("ELSE", 11);
19          tokensMap.put("WHILE", 12);
20          tokensMap.put("ADD", 13);
21          tokensMap.put("MUL", 14);
22          tokensMap.put("LESS_THAN", 15);
23          tokensMap.put("DOLLARS", 16);
24
25          tokensMap.put("S", 100);
26          tokensMap.put("E", 101);
27          tokensMap.put("E'", 102);
28          tokensMap.put("T", 103);
29          tokensMap.put("T'", 104);
30          tokensMap.put("C", 105);
31          tokensMap.put("F", 106);
32      }
33  }
34
```

7. 核心算法

1. `parse()方法`

   作用：将栈中的元素与token序列中的元素进行匹配分析

```
1   static void parse(){
2       int s1, s2; //s1栈中元素，s2序列中的元素
3
4       while(tokens.get(0)!=Token.tokensMap.get("DOLLARS")){
5           s1=stack.get();
6           s2=tokens.get(0);
```

```
        if(s1>99){
            if(!generate(s1, s2)){
                System.out.println("Error");
                return;
            }
        }else{
            if(s1==s2){
                stack.pop();
                tokens.remove(0);
            }else{
                System.out.println("Error");
                return;
            }
        }
    }
}
```

2. `generate()方法`

作用：获取非终结符对应的产生式并入栈

```
static boolean generate(int stack_element, int queue_element){
    try{
        int column=getIndex(queue_element);
        int gi=p_parsingTable[stack_element-100][column];
        //            System.out.println((stack_element - 100) + " "+ column);
        output.add(generations[gi]);
        stack.pop();
        switch (gi){
            case 0:
                stack.push(Token.tokensMap.get("E"));
                stack.push(Token.tokensMap.get("ASSIGN"));
                stack.push(Token.tokensMap.get("ID"));
                break;
            case 1:
                stack.push(Token.tokensMap.get("R_BRACE"));
                stack.push(Token.tokensMap.get("S"));
                stack.push(Token.tokensMap.get("L_BRACE"));
                stack.push(Token.tokensMap.get("ELSE"));
                stack.push(Token.tokensMap.get("R_BRACE"));
                stack.push(Token.tokensMap.get("S"));
                stack.push(Token.tokensMap.get("L_BRACE"));
                stack.push(Token.tokensMap.get("R_BRACKET"));
                stack.push(Token.tokensMap.get("C"));
                stack.push(Token.tokensMap.get("L_BRACKET"));
                stack.push(Token.tokensMap.get("IF"));
                break;
            case 2:
                stack.push(Token.tokensMap.get("R_BRACE"));
                stack.push(Token.tokensMap.get("S"));
                stack.push(Token.tokensMap.get("L_BRACE"));
                stack.push(Token.tokensMap.get("R_BRACKET"));
                stack.push(Token.tokensMap.get("C"));
                stack.push(Token.tokensMap.get("L_BRACKET"));
                stack.push(Token.tokensMap.get("WHILE"));
```

```
                break;
            case 3:
                stack.push(Token.tokensMap.get("E'"));
                stack.push(Token.tokensMap.get("T"));
                break;
            case 4:
                stack.push(Token.tokensMap.get("E'"));
                stack.push(Token.tokensMap.get("T"));
                stack.push(Token.tokensMap.get("ADD"));
                break;
            case 5:
                break;
            case 6:
                stack.push(Token.tokensMap.get("T'"));
                stack.push(Token.tokensMap.get("F"));
                break;
            case 7:
                stack.push(Token.tokensMap.get("T'"));
                stack.push(Token.tokensMap.get("F"));
                stack.push(Token.tokensMap.get("MUL"));
                break;
            case 8:
                break;
            case 9:
                stack.push(Token.tokensMap.get("NUM"));
                break;
            case 10:
                stack.push(Token.tokensMap.get("ID"));
                break;
            case 11:
                stack.push(Token.tokensMap.get("F"));
                stack.push(Token.tokensMap.get("LESS_THAN"));
                stack.push(Token.tokensMap.get("F"));
                break;
            default:
                System.out.println("Error");
                return false;
        }
        return true;
    }catch(Exception e){
        System.out.println("Error");
        return false;
    }
}
```

8. 用例及运行结果

- 用例

Main.java × input.txt × Lex

```
while( abba < 1 ){
    if( a < 10 ){
        a = a * 10 + 1
    }else{
        a = a + 1
    }
}

```

- 结果

output.txt × Main.java × in

```
S->while(C){S}
C->F<F
F->id
F->num
S->if(C){S}else{S}
C->F<F
F->id
F->num
S->id=E
E->TE'
T->FT'
F->id
T->*FT'
F->num
T'->ε
E'->+TE'
T->FT'
F->num
T'->ε
E'->ε
S->id=E
E->TE'
T->FT'
F->id
T'->ε
E'->+TE'
T->FT'
F->num
T'->ε
E'->ε

```

9. 问题及解决方案

1.
   - 问题：实验一因为作图原因，规定的正则表达式过于简单，不能满足实验二的基本需求
   - 解决方案：重新构建DFA^{O}并根据新的DFA^{O}对原来的词法分析程序做变更，同时使其能作为实验二的工具使用
2.
   - 问题：在进行语法分析的时候总是报错
   - 解决方案：通过调试，发现是预测分析表构建有误，因此，重新构建了预测分析表，方能使程序正确运行
3.

- 问题：受C++影响，想通过枚举类对token进行枚举，以使其对应相应的整数，便于查表，后发现Java枚举不能类比C++的枚举使用
- 解决方案：改为使用`HashMap`，将token和整数对应起来

10. 个人感想

通过实验，进一步加深了对课堂所学知识的理解和运用，同时，也通过自己编写编译前端，对编译之美有了更深的体会。不过，较为遗憾的是，有很多步骤是通过手动实现的，且使用的是LL(1)文法，具有一定的缺陷，希望有机会能对此做进一步的尝试！