Lab Document for Syntax Analyzer

a)Motivation/Aim

利用自底向上方式对输入的字符流进行简单的语法分析，并返回归约序列。

b)Content description

程序以 java 编写，包含 Parser、LexicalAnalyzer、Formatter、FileHelper 四个类，其中 Parser 为启动类，其首先利用 LexicalAnalyzer.tokenizer()分析 input.txt 中的输入生成词法单元序列 tokens 输出到 tokens.txt，再借由 ArrayList<String> tokenList=Formatter.transform()将 tokens.txt 中的词法单元简化并输出到 simp_tokens.txt，最后根据事先构建的 LR(1)翻译表进行语法分析，而 FileHelper 为封装文件操作的工具类，包含文件创建及读写的方法。

c)Ideas/Methods

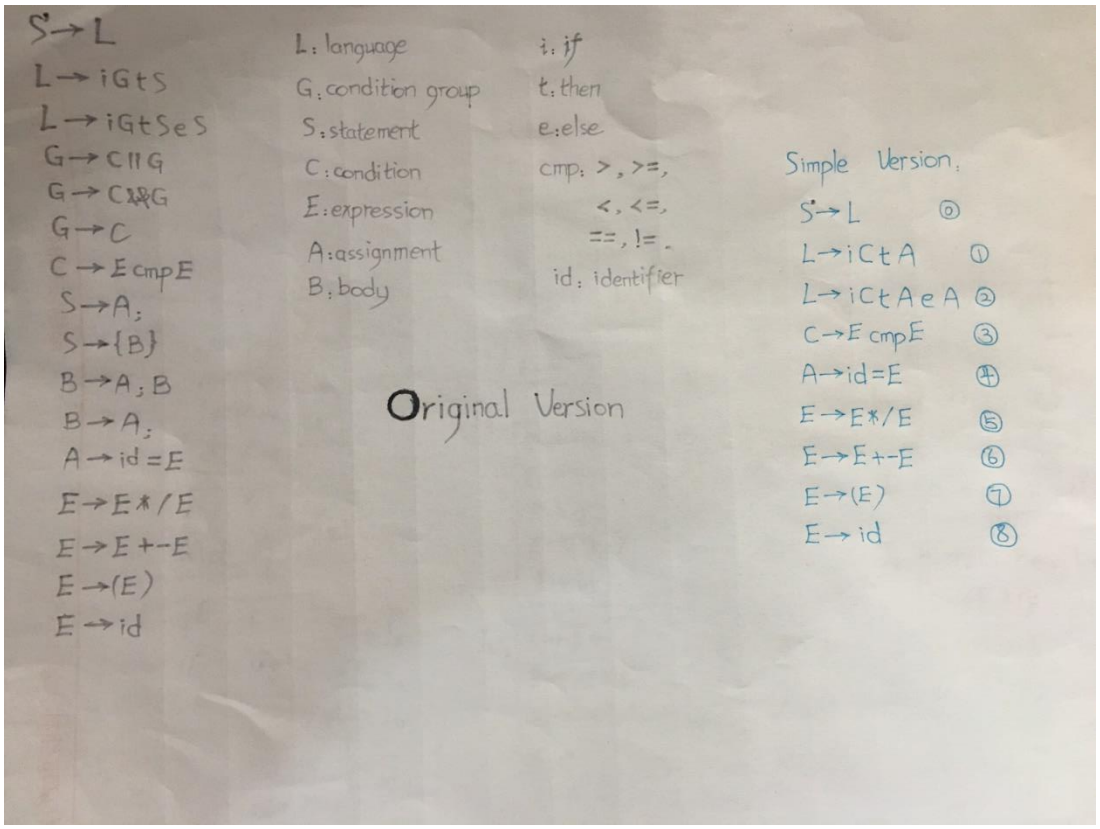
1)Construct LR(1) parsing table based on the CFG

2)Design the program using LR(1) paring table

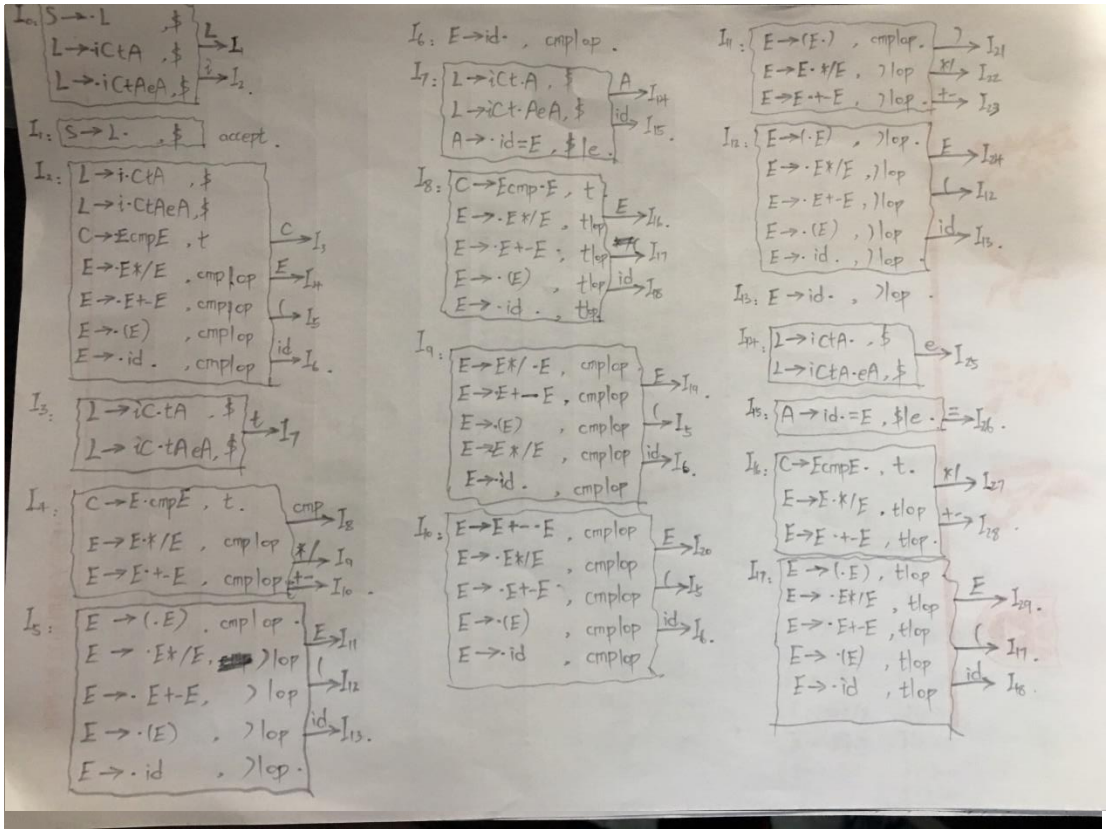
具体细节在文档后续内容提及，此处先略过。

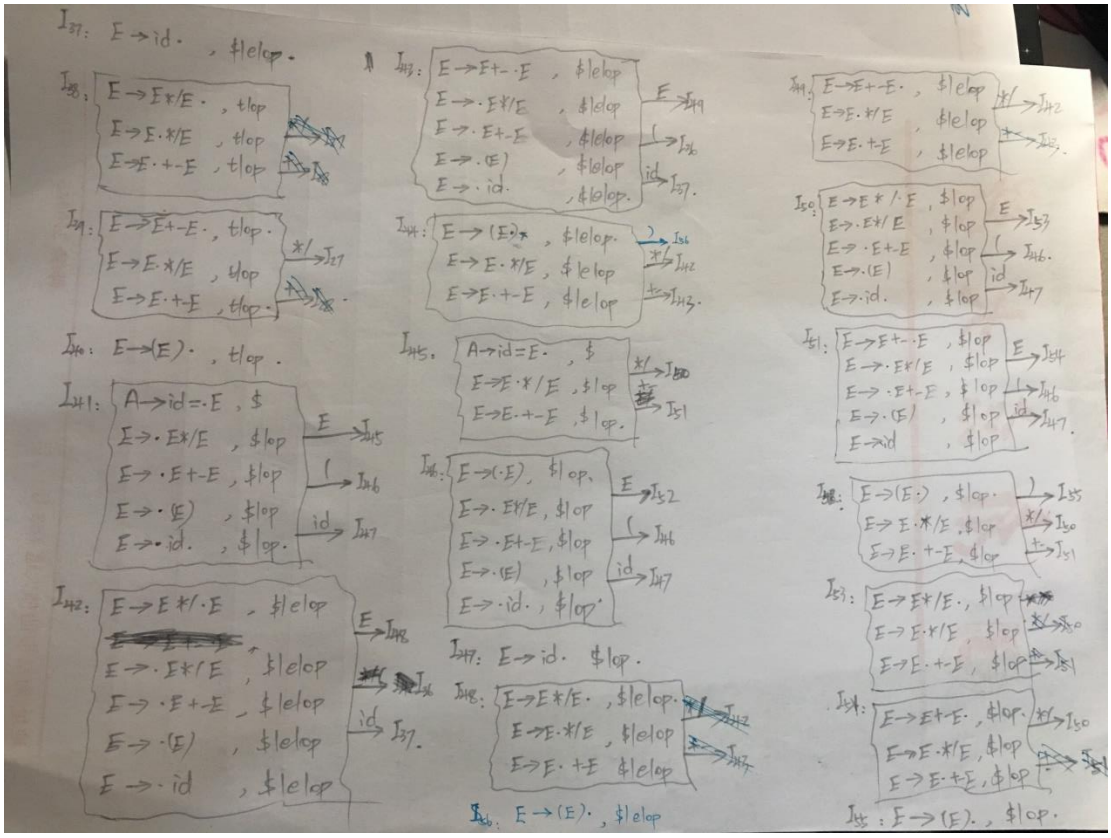
d)Assumptions

预先定义的 CFG 如下（原始定义文法状态过多，故采用简化后的文法）：


S→L
L→iGtS
L→iGtSeS
G→C||G
G→C&&G
G→C
C→EcmpE
S→A;
S→{B}
B→A;B
B→A;
A→id=E
E→E*/E
E→E+-E
E→(E)
E→id
L: language
G: condition group
S: statement
C: condition
E: expression
A: assignment
B: body
i: if
t: then
e: else
cmp: >, >=,
<, <=,
==, !=.
id: identifier
Original Version
Simple Version:
S→L ⓪
L→iCtA ①
L→iCtAeA ②
C→EcmpE ③
A→id=E ④
E→E*/E ⑤
E→E+-E ⑥
E→(E) ⑦
E→id ⑧

构建语法分析表的过程：


I0: S→·L , $; L→·iCtA , $; L→·iCtAeA , $; L→I1 ; i→I2
I1: S→L· , $ accept
I2: L→i·CtA , $; L→i·CtAeA , $; C→E cmp E , t ; E→·E*/E , cmplop ; E→·E+-E , cmplop ; E→·(E) , cmplop ; E→·id , cmplop ; C→I3 ; E→I4 ; (→I5 ; id→I6
I3: L→iC·tA , $; L→iC·tAeA , $; t→I7
I4: C→E·cmpE , t ; E→E·*/E , cmplop ; E→E·+-E , cmplop ; cmp→I8 ; */→I9 ; +-→I10
I5: E→(·E) , cmplop ; E→·E*/E ,)lop ; E→·E+-E ,)lop ; E→·(E) ,)lop ; E→·id ,)lop ; E→I11 ; (→I12 ; id→I13
I6: E→id· , cmplop
I7: L→iCt·A , $; L→iCt·AeA , $; A→·id=E , $le ; A→I14 ; id→I15
I8: C→E cmp·E , t ; E→·E*/E , tlop ; E→·E+-E , tlop ; E→·(E) , tlop ; E→·id , tlop ; E→I16 ; */→I17 ; (→I5 ; id→I6
I9: E→E*/·E , cmplop ; E→E+-·E , cmplop ; E→·(E) , cmplop ; E→E*/·E , cmplop ; E→·id , cmplop ; E→I19 ; (→I5 ; id→I6
I10: E→E+-·E , cmplop ; E→·E*/E , cmplop ; E→·E+-E , cmplop ; E→·(E) , cmplop ; E→·id , cmplop ; E→I20 ; (→I5 ; id→I6
I11: E→(E·) , cmplop ; E→E·*/E ,)lop ; E→E·+-E ,)lop ;)→I21 ; */→I22 ; +-→I23
I12: E→(·E) ,)lop ; E→·E*/E ,)lop ; E→·E+-E ,)lop ; E→·(E) ,)lop ; E→·id ,)lop ; E→I24 ; (→I12 ; id→I13
I13: E→id· ,)lop
I14: L→iCtA· , $; L→iCtA·eA , $; e→I25
I15: A→id·=E , $le ; =→I26
I16: C→E cmpE· , t ; E→E·*/E , tlop ; E→E·+-E , tlop ; */→I27 ; +-→I28
I17: E→(·E) , tlop ; E→·E*/E , tlop ; E→·E+-E , tlop ; E→·(E) , tlop ; E→·id , tlop ; E→I29 ; (→I17 ; id→I18
I18: E→id· , tlop
I19: E→E*/E· , cmplop ; E→E*/E , cmplop ; E→E+-E , cmplop ; */→I9 ; +-→I10
I20: E→E+-E· , cmplop ; E→E·*/E , cmplop ; E→E·+-E , cmplop ; */→I9 ; +-→I10
I21: E→(E)· , cmplop
I22: E→E*/·E ,)lop ; E→·E*/E ,)lop ; E→·E+-E ,)lop ; E→·(E) ,)lop ; E→·id ,)lop ; E→I30 ; (→I12 ; id→I13
I23: E→E+-·E ,)lop ; E→·E*/E ,)lop ; E→·E+-E ,)lop ; E→·(E) ,)lop ; E→·id ,)lop ; E→I31 ; (→I12 ; id→I13
I24: E→(E·) ,)lop ; E→E·*/E ,)lop ; E→E·+-E ,)lop ;)→I32 ; */→I22 ; +-→I23
I25: L→iCtAe·A , $; A→·id=E , $; A→I33 ; id→I34
I26: A→id=·E , $le ; E→·E*/E , $lelop ; E→·E+-E , $lelop ; E→·(E) , $lelop ; E→·id , $lelop ; E→I35 ; (→I36 ; id→I37
I27: E→E*/·E , tlop ; E→·E*/E , tlop ; E→·E+-E , tlop ; E→·(E) , tlop ; E→·id , tlop ; E→I38 ; (→I17 ; id→I18
I28: E→E+-·E , tlop ; E→·E*/E , tlop ; E→·E+-E , tlop ; E→·(E) , tlop ; E→·id , tlop ; E→I39 ; (→I17 ; id→I18
I29: E→(E·) , tlop ; E→E·*/E , tlop ; E→E·+-E , tlop ;)→I40 ; */→I27 ; +-→I28
I30: E→E*/E· ,)lop ; E→E·*/E ,)lop ; E→E·+-E ,)lop ; */→I22 ; +-→I23
I31: E→E+-E· ,)lop ; E→E·*/E ,)lop ; E→E·+-E ,)lop ; */→I22 ; +-→I23
I32: E→(E)· ,)lop
I33: L→iCtAeA· , $; A→·id=E , $; id→I34
I34: A→id·=E , $; =→I41
I35: A→id=E· , $le ; E→E·*/E , $lelop ; E→E·+-E , $lelop ; */→I42 ; +-→I43
I36: E→(·E) , $lelop ; E→·E*/E , $lelop ; E→·E+-E , $lelop ; E→·(E) , $lelop ; E→·id , $lelop ; E→I44 ; (→I36 ; id→I37



e)Related FA descriptions

LR(1) parsing table 如下：

State	Action											goto			
	i	t	e	cmp	id	(	)	*/	+-	=	$	L	C	E	A
0	S2											S1			
1											ACC				
2					S6	S5							S3	S4	
3		S7													
4				S8				S9	S10						
5					S13	S12								S11	
6				R8	R8										
7					S15										S14
8					S18	S17								S16	
9					S6	S5								S19	
10					S6	S5								S20	
11							S21	S22	S23						
12					S13	S12								S24	
13							R8	R8	R8						

14			S25								R1				
15										S26					
16		R3						S27	S28						
17					S18	S17								S29	
18		R8						R8	R8						
19				R5				R5	R5						
20				R6				S9	R6						
21				R7				R7	R7						
22					S13	S12								S30	
23					S13	S12								S31	
24							S32	S22	S23						
25					S34										S33
26					S37	S36								S35	
27					S18	S17								S38	
28					S18	S17								S39	
29							S40	S27	S28						
30							R5	R5	R5						
31							R6	S22	R6						
32							R7	R7	R7						
33					S34						R2				
34										S41					
35			R4					S42	S43		R4				
36					S37	S36								S44	
37			R8					R8	R8		R8				
38		R5						R5	R5						
39		R6						S27	R6						
40		R7						R7	R7						
41					S47	S46								S45	
42					S37	S36								S48	
43					S37	S36								S49	
44							S56	S42	S43						
45								S50	S51		R4				
46					S47	S46								S52	
47								R8	R8		R8				
48			R5					R5	R5		R5				
49			R6					S42	R6		R6				
50					S47	S46								S53	
51					S47	S46								S54	
52							S55	S50	S51						
53								R5	R5		R5				
54								S50	R6		R6				
55								R7	R7		R7				
56			R7					R7	R7		R7				

PS: Using Precedence and Associativity to Resolve Parsing Action Conflicts

f)Description of important Data Structures

```
//程序内语法分析表以二维整型数组表示
//其中正数表示移入，负数表示归约，0 表示 Error，Integer.MAX_VALUE 表示 Accept
public static int[][] parsingTb = {
        {2,0,0,0,0,0,0,0,0,0,0,1,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,Integer.MAX_VALUE,0,0,0,0},
        {0,0,0,0,6,5,0,0,0,0,0,0,3,4,0},
        {0,7,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,8,0,0,0,9,10,0,0,0,0,0,0},
        {0,0,0,0,13,12,0,0,0,0,0,0,0,11,0},
        {0,0,0,-8,-8,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,15,0,0,0,0,0,0,0,0,0,14},
        {0,0,0,0,18,17,0,0,0,0,0,0,0,16,0},
        {0,0,0,0,6,5,0,0,0,0,0,0,0,19,0},
        {0,0,0,0,6,5,0,0,0,0,0,0,0,20,0},
        {0,0,0,0,0,0,21,22,23,0,0,0,0,0,0},
        {0,0,0,0,13,12,0,0,0,0,0,0,0,24,0},
        {0,0,0,0,0,0,-8,-8,-8,0,0,0,0,0,0},
        {0,0,25,0,0,0,0,0,0,0,-1,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,26,0,0,0,0,0},
        {0,-3,0,0,0,0,0,27,28,0,0,0,0,0,0},
        {0,0,0,0,18,17,0,0,0,0,0,0,0,29,0},
        {0,-8,0,0,0,0,0,-8,-8,0,0,0,0,0,0},
        {0,0,0,-5,0,0,0,-5,-5,0,0,0,0,0,0},
        {0,0,0,-6,0,0,0,9,-6,0,0,0,0,0,0},
        {0,0,0,-7,0,0,0,-7,-7,0,0,0,0,0,0},
        {0,0,0,0,13,12,0,0,0,0,0,0,0,30,0},
        {0,0,0,0,13,12,0,0,0,0,0,0,0,31,0},
        {0,0,0,0,0,0,32,22,23,0,0,0,0,0,0},
        {0,0,0,0,34,0,0,0,0,0,0,0,0,0,33},
        {0,0,0,0,37,36,0,0,0,0,0,0,0,35,0},
        {0,0,0,0,18,17,0,0,0,0,0,0,0,38,0},
        {0,0,0,0,18,17,0,0,0,0,0,0,0,39,0},
        {0,0,0,0,0,0,40,27,28,0,0,0,0,0,0},
        {0,0,0,0,0,0,-5,-5,-5,0,0,0,0,0,0},
        {0,0,0,0,0,0,-6,22,-6,0,0,0,0,0,0},
        {0,0,0,0,0,0,-7,-7,-7,0,0,0,0,0,0},
        {0,0,0,0,34,0,0,0,0,0,-2,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,41,0,0,0,0,0},
        {0,0,-4,0,0,0,0,42,43,0,-4,0,0,0,0},
```

```
        {0,0,0,0,37,36,0,0,0,0,0,0,0,44,0},
        {0,0,-8,0,0,0,0,-8,-8,0,-8,0,0,0,0},
        {0,-5,0,0,0,0,0,-5,-5,0,0,0,0,0,0},
        {0,-6,0,0,0,0,0,27,-6,0,0,0,0,0,0},
        {0,-7,0,0,0,0,0,-7,-7,0,0,0,0,0,0},
        {0,0,0,0,47,46,0,0,0,0,0,0,0,45,0},
        {0,0,0,0,37,36,0,0,0,0,0,0,0,48,0},
        {0,0,0,0,37,36,0,0,0,0,0,0,0,49,0},
        {0,0,0,0,0,0,56,42,43,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,50,51,0,-4,0,0,0,0},
        {0,0,0,0,47,46,0,0,0,0,0,0,0,52,0},
        {0,0,0,0,0,0,0,-8,-8,0,-8,0,0,0,0},
        {0,0,-5,0,0,0,0,-5,-5,0,-5,0,0,0,0},
        {0,0,-6,0,0,0,0,42,-6,0,-6,0,0,0,0},
        {0,0,0,0,47,46,0,0,0,0,0,0,0,53,0},
        {0,0,0,0,47,46,0,0,0,0,0,0,0,54,0},
        {0,0,0,0,0,0,55,50,51,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,-5,-5,0,-5,0,0,0,0},
        {0,0,0,0,0,0,0,50,-6,0,-6,0,0,0,0},
        {0,0,0,0,0,0,0,-7,-7,0,-7,0,0,0,0},
        {0,0,-7,0,0,0,0,-7,-7,0,-7,0,0,0,0}
};
```

```
//输入符号到其索引的映射
Map<String, Integer> map = new HashMap<String, Integer>();

//产生式编号到相应产生式的映射
Map<Integer,String> prodMap=new HashMap<>();

//产生式编号到产生式体符号个数的映射
Map<Integer,Integer> bodyMap=new HashMap<>();

//产生式编号到产生式头非终结符的映射
Map<Integer,String> headMap=new HashMap<>();

//用来保存语法分析器状态的栈
Stack stateStack = new Stack<Integer>();
stateStack.push(0);//初始时仅含状态 0

int cur_state;//栈顶状态
int symbol;//当前输入符号在语法分析表对应索引
```

```
int action;//翻译表所采取动作
ArrayList<String> reductions=new ArrayList<>();//保存归约式子的链表
String s="";//当前输入符号
int index=0;//当前输入符号在简化的 tokens 序列中对应索引
```

g)Description of core Algorithms

先贴代码，再行分析:)

```
Stack stateStack = new Stack<Integer>();
stateStack.push(0);
int cur_state;
int symbol;
int action;
ArrayList<String> reductions=new ArrayList<>();
String s="";
int index=0;
while(index<tokenList.size()){
   s=tokenList.get(index);
   cur_state=(int)stateStack.peek();
   symbol=map.get(s);
   action=parsingTb[cur_state][symbol];
   if(action==0){
        reductions.add("Error!!!(Current State:"+cur_state+"&&Input Symbol:"+s+")");
        break;
   }else if(action>0){
        if(action==Integer.MAX_VALUE){
            if(stateStack.search(0)==2){
                reductions.add(prodMap.get(0));
                reductions.add("Accept!");
                break;
            }
            else{
                reductions.add(prodMap.get(0));
                reductions.add("Unexpected End!!!");
                break;
            }
        }else{
            stateStack.push(action);
            index++;
        }
   }else{
```

```
            reductions.add(prodMap.get(-action));
            for(int i=0;i<bodyMap.get(-action);i++){
                stateStack.pop();
            }
            index--;
            tokenList.set(index,headMap.get(-action));
        }
    }
```

分析：

根据当前状态以及输入符号借由语法分析表可以得到应该采取的动作：若为数值 0，则表明语法分析出错；若大于 0，考虑 Integer.MAX_VALUE，如果 action 等于最大值且栈中除当前状态外仅含状态 0，说明分析结束，输入串可以接受，不然同样是语法分析出错，而 action 不为 Integer.MAX_VALUE 时则需将对应状态移入栈中并将处理序列的索引值加一；若小于 0，则根据对应产生式执行归约，从栈中移除数目等同产生式体符号的状态，并把用于遍历的索引值减一，将其对应位置的简化词法单元设为产生式头。重复上述过程，直到分析过程发生错误或结束输入序列的分析。

h)Use cases on running

在 input.txt 中，有三个测试用例，执行时需保证有且只有一个用例未被注释。

```
//Just to have a Test

//Test 1
/*if x>y
then
diff=x-y
else
diff=y-x*/

//Test 2
/*if z>x+y
then
D_value=z-(x+y)*/

//Test 3
/*if z>x+y
D_value=z-(x+y)*/
```

其中归约序列会输出到 rSequence.txt 中，下面为各测试用例输出：

```
//Test 1
if x>y
then
diff=x-y
else
diff=y-x
//rSequence
E->id
E->id
C->EcmpE
E->id
E->id
E->E+-E
A->id=E
E->id
E->id
E->E+-E
A->id=E
L->iCtAeA
S->L
Accept!

//Test 2
if z>x+y
then
D_value=z-(x+y)
//rSequence
E->id
E->id
E->id
E->E+-E
C->EcmpE
E->id
E->id
E->E+-E
E->id
E->E+-E
A->id=E
L->iCtA
S->L
Accept!
```

```
//Test 3
if z>x+y
D_value=z-(x+y)
//rSequence
E->id
E->id
Error!!!(Current State:18&&Input Symbol:id)
```

i)Problems occurred and related solutions

错误处理的话，均会在 rSequence.txt 加入错误信息（代码见下）：

```
//出现语法分析表中未定义状态转换
if(action==0){
reductions.add("Error!!!(Current State:"+cur_state+"&&Input Symbol:"+s+")");
        break;
}


//未结束输入分析便提前读取到$并进入终止状态
reductions.add("Unexpected End!!!");


//输入全部读取完毕，语法分析机仍未进入接受状态
if(!reductions.get(reductions.size()-1).equals("Accept!")&&index==tokenList.size()-1){
reductions.add("The input ends in an unacceptable state...");
}
```

j) Your feelings and comments

语法分析器从词法分析器获取一个由词法单元组成的串，并验证这个串可以由源语言的文法生成，其常用方法可以分为自顶向下的(LL(1))和自底向上的(LR(1))。

本次实验我采用了 LR(1)语法分析，相较于 LL(1)而言，虽然可处理文法种类更多，但增加了复杂度（57 个状态 QWQ，画图画到怀疑人生）……果然，LR(1)文法还是适合 YACC 语法分析生成程序来做而不是人工，反正我不想再画了☹

PS(附上老师 ppt 上的说明…):

LL(1) grammars(often implemented by hand)

LR grammars(often constructed by automated tools)

By 161250098 彭俊杰