**北京科技大学实验报告**

学院： 专业： 班级：

姓名：刘丰 学号：42007005 实验日期： 2023 年 5月 18 日

**实验名称：**一个含有词法分析和语法分析的编译器，以pl0文法为例

**实验环境：java**

**实验原理：SLR文法**

**实验内容与步骤：**

PL/0的文法：

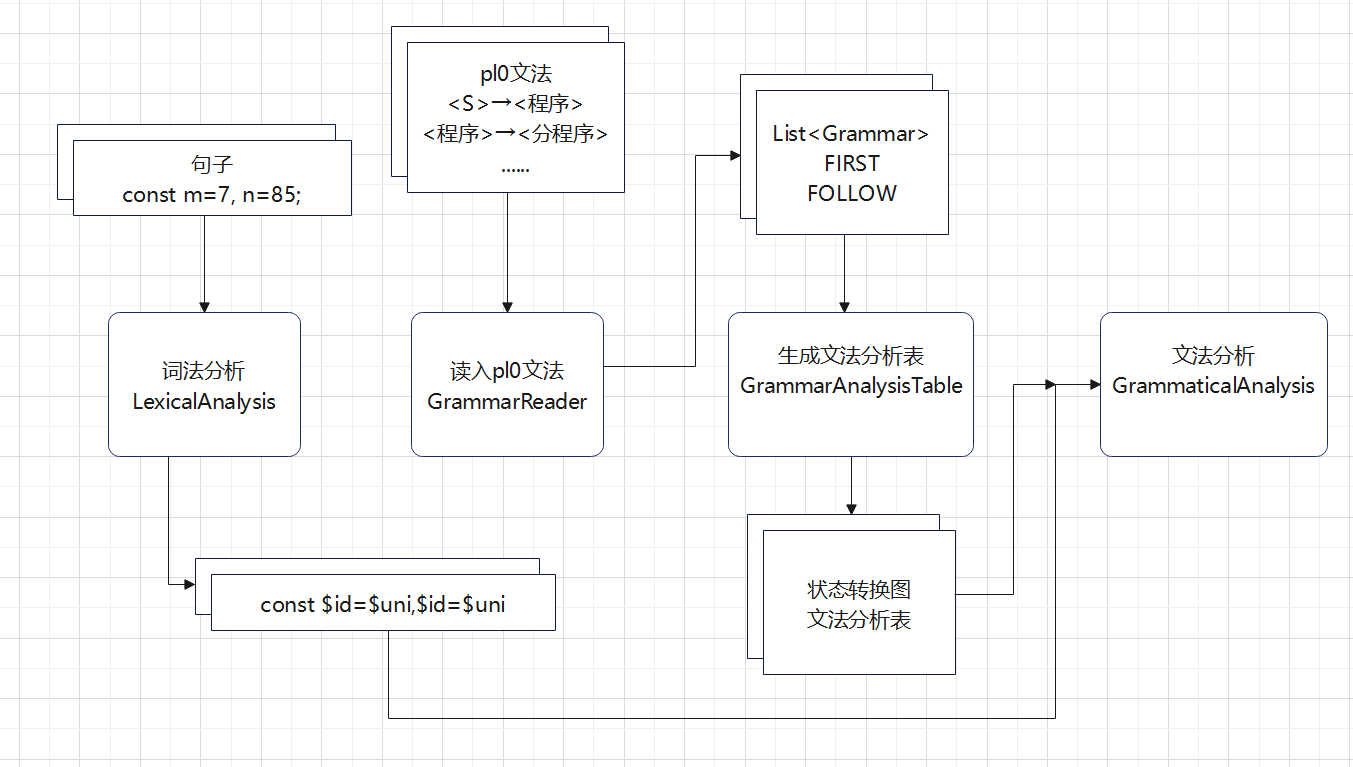


**改写后：**

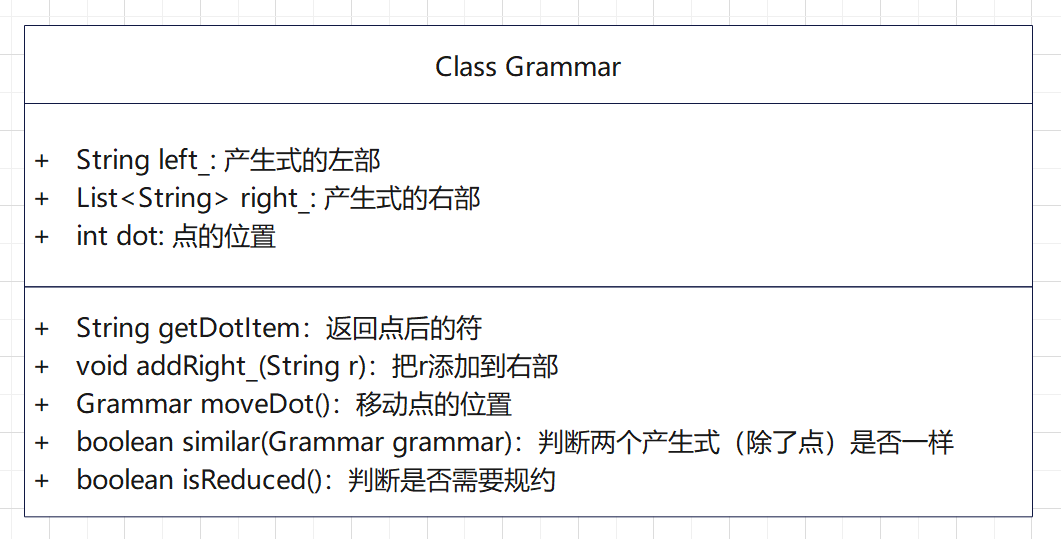


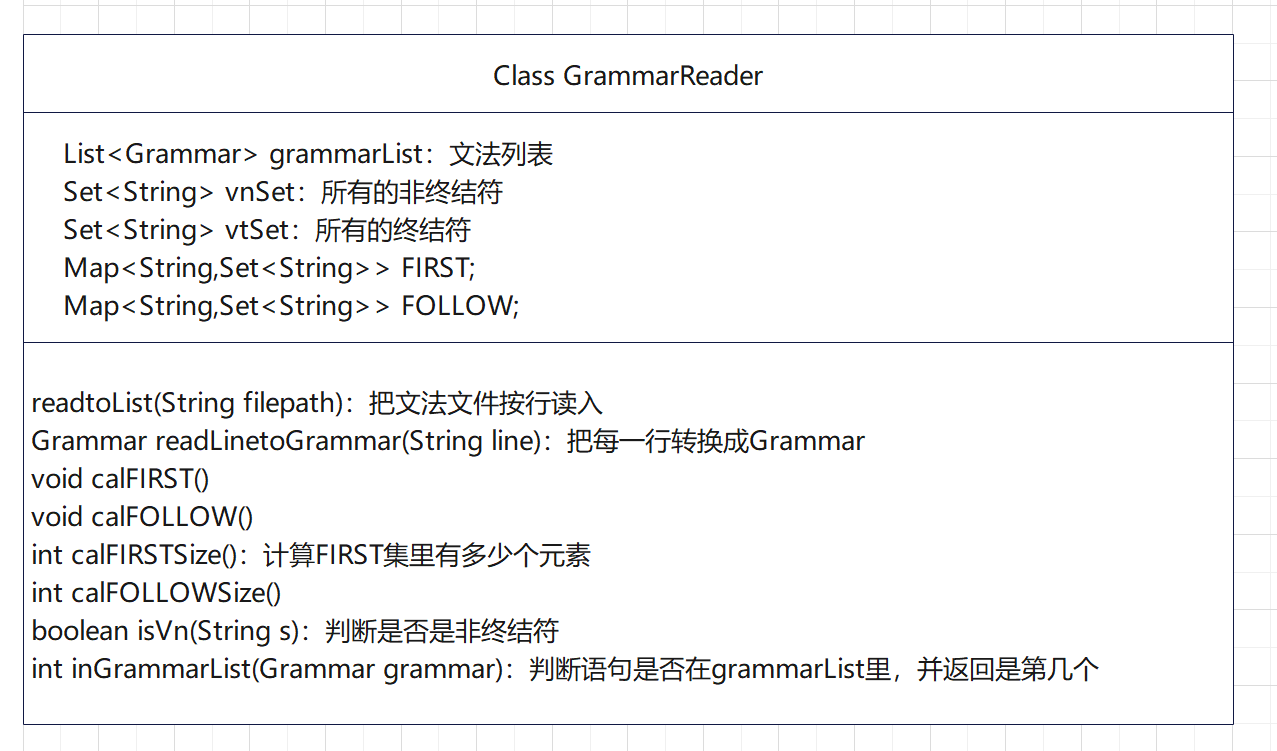
**因为pl0文法语句过多，手工计算语法分析表太繁琐，采用程序计算。**

**主要流程和相应的关键函数**



**GrammarReader**





private Grammar readLinetoGrammar(String line){

按"→"split分成左部left，右部right

按字符遍历右部：

' '就continue

'<'就find（'>'）substring这一段，添加进非终结符vnSet里

是字母(ch>='a' && ch<='z') || (ch>='A' && ch<='Z')就继续向下读，直至读到一个不是字母的，添加进终结符vtSet里

'$'就继续向下读，直至读到一个不是字母的，添加进终结符vtSet里

其他情况

如果是':'，把:=添加进终结符vtSet里

其他直接添加进终结符vtSet里

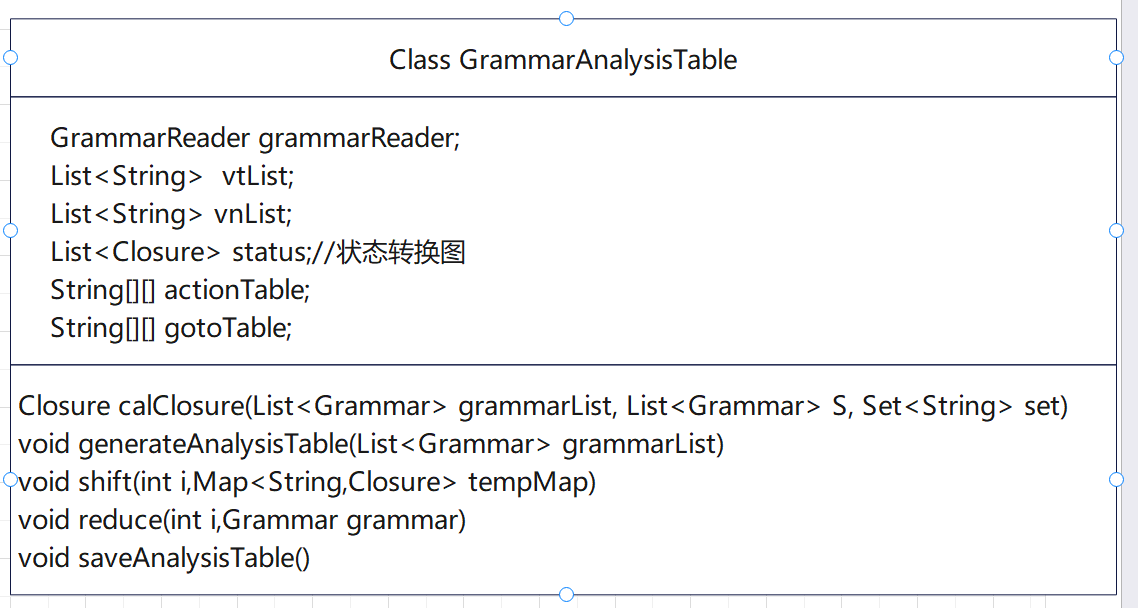
return grammar;

}

*//计算FIRST*public void calFIRST(){  
 int preSize=-1;  
 int currentSize=0;for (String s : vnSet) {*// 初始化*  
 this.FIRST.put(s,new HashSet<>());  
 }while(preSize!=currentSize){ *// 当集合有变化时*  
 preSize=currentSize;for (Grammar grammar : grammarList) {*// 遍历grammarList*  
 String left\_=grammar.left\_; *//左部* String first=grammar.right\_.get(0); *//右部第一个字符* if(isVn(first)){ *// 如果右部第一个字符是非终结符*  
 int size=grammar.right\_.size();  
 *// 遍历grammar的右部* for(int i=0;i<size;i++){  
 String righti=grammar.right\_.get(i);  
 *// 如果是终结符，就退出* if(!isVn(righti)){  
 break;  
 }  
 *// 如果是终结符，就把righti的FIRST（除了ε）添加进left\_* for (String s : this.FIRST.get(righti)) {  
 if(!"ε".equals(s)){  
 this.FIRST.get(left\_).add(s);  
 }else if (i==size){*// 如果所有右部的都能推出ε，就把ε添加进left\_* this.FIRST.get(first).add("ε");  
 }  
 }  
 *// 如果righti的FIRST没有ε，就退出* if(!this.FIRST.get(righti).contains("ε")){  
 break;  
 }  
 }  
   
 }  
 else {*//如果右部第一个字符是终结符,直接加入* this.FIRST.get(left\_).add(first);  
 }  
 }  
 *// 计算现在FIRST集大小* currentSize=this.calFIRSTSize();  
 }  
}

private void calFOLLOW(){  
 int preSize=-1;  
 int currentSize=0;  
 *// 初始化* for (String s : vnSet) {  
 this.FOLLOW.put(s,new HashSet<>());  
 }  
 *// 对于文法的开始符号，把ε加进去* this.FOLLOW.get(this.grammarList.get(0).left\_).add("ε");  
 while(preSize!=currentSize){*// 当集合有变化时* preSize=currentSize;  
 for (Grammar grammar : grammarList) {  
 String left\_=grammar.left\_;  
 List<String> rights=grammar.right\_;  
 int rightsize=rights.size();  
 for (int i=0;i<rightsize;i++){  
 if(isVn(rights.get(i))){*// 如果当前这个是非终结符* if(i+1<rightsize && isVn(rights.get(i+1))){*// 下一个也是非终结符  
 // 把除了ε 下一个所有FIRST元素都加进去* for (String s : this.FIRST.get(rights.get(i+1))) {  
 if(!"ε".equals(s)){  
 this.FOLLOW.get(rights.get(i)).add(s);  
 }  
 }  
 }  
 else if(i+1>=rightsize){  
 this.FOLLOW.get(rights.get(i)).add("ε");  
 }  
 else{  
 this.FOLLOW.get(rights.get(i)).add(rights.get(i+1));  
 }  
 }  
 }  
 for(int i=rightsize-1;i>=0;i--){  
 *// 如果是非终结符，就把左部的follow集加入* if(isVn(rights.get(i))){  
 for (String s : this.FOLLOW.get(left\_)) {  
 this.FOLLOW.get(rights.get(i)).add(s);  
 }  
 }  
 else {  
 break;  
 }  
  
 if(!this.FIRST.get(rights.get(i)).contains("ε")){  
 break;  
 }  
 }  
 }  
 currentSize=this.calFOLLOWSize();  
 }  
}

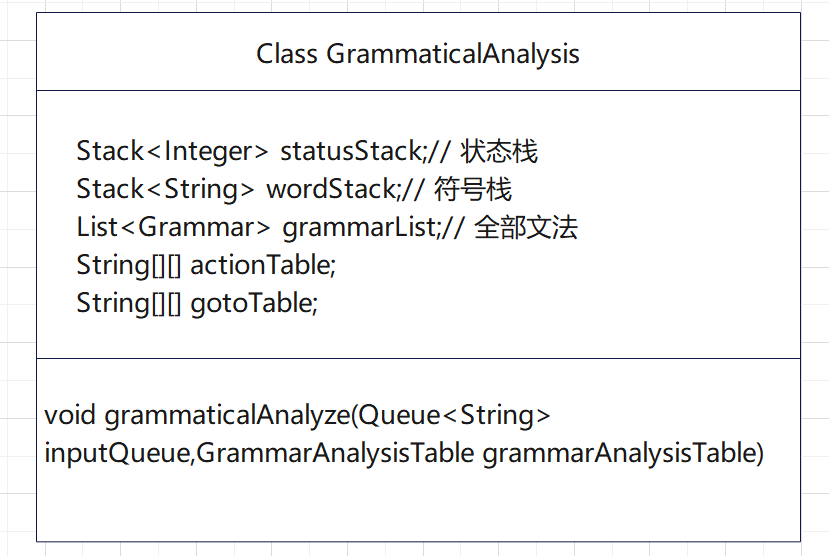
**GrammarAnalysisTable**



*// 计算闭包（文法集，前一个状态，新状态的·后的字符）*Closure calClosure(List<Grammar> grammarList, List<Grammar> S, Set<String> set){  
 List<Grammar> closure =new ArrayList<>();  
 Queue<Grammar> queue=new LinkedList<>();  
 for (Grammar grammar : S) {  
 queue.add(grammar);  
 closure.add(grammar);  
 }  
 while(!queue.isEmpty()){  
 Grammar temp=queue.remove();*//获取队首* String item=temp.getDotItem();*//获得·后面的字符* if(item==null || item.equals("ε")){  
 continue;  
 }  
 set.add(item);  
  
 if(isVt(item)>=0){*//是终结符* continue;  
 }  
 *//是非终结符Vn,遍历grammarList,寻找左部为Vt的表达式,加入closure* for (Grammar grammar : grammarList) {  
 String left\_ = grammar.left\_;  
 if(item.equals(left\_)){  
 *// 如果语句在closure里就跳过* if(this.inStatus1(grammar,closure)){  
 continue;  
 }  
 *// 不在closure里* else{  
 closure.add(grammar);  
 *// 不是左递归* if(!temp.left\_.equals(grammar.getDotItem())){  
 queue.add(grammar);  
 }  
 }  
 }  
 }  
  
 }  
 Closure status = new Closure(closure);  
 return status;  
}

*//生成分析表*void generateAnalysisTable(List<Grammar> grammarList){  
 List<Closure> analysisTable=new ArrayList<>();  
 *//将开始元素添加进来* List<Grammar> start=new ArrayList<>();  
 Grammar S = grammarList.get(0);  
 start.add(S);  
 *//创建记录·后字符的集合* Set<String> set=new HashSet<>();  
 Queue<List<String>> itemQ=new LinkedList<>();  
 Queue<Closure> closureQ=new LinkedList<>(); *//计算第0个状态* int i=0;  
 Closure closure = this.calClosure(grammarList, start,set);  
 closure.index=i++;  
 List<String> itemL=new ArrayList<>(set);  
 set.clear();  
 *//放到他们该在的地方* itemQ.add(itemL);  
 closureQ.add(closure);  
 analysisTable.add(closure);  
 *// 深度优先* while (!closureQ.isEmpty()){  
 *//弹出第一个状态* closure=closureQ.remove();  
 *//弹出第一组该移动的* itemL=itemQ.remove();  
  
 for (String s : itemL) {  
 *//转移状态，对·移动* Closure temp = closure.moveDot(s);  
 *//计算闭包* Closure newClosure = this.calClosure(grammarList, temp.status1, set);  
 int flag=this.inClosureList(newClosure,analysisTable);  
  
 List<String> itemList=new ArrayList<>(set);  
 set.clear();  
  
 *// 如果已经出现过* if(flag>=0){  
 newClosure=analysisTable.get(flag);  
 }  
 *// 如果没有出现过* else{  
 newClosure.index=i++;  
 *//放到他们该在的地方* itemQ.add(itemList);  
 closureQ.add(newClosure);  
 analysisTable.add(newClosure);  
 }  
 *// closure指向newClosure* closure.addNewClosure(newClosure,s);  
 }  
 }  
 this.status =analysisTable;  
}  
  
*// 移进*void shift(int i,Map<String,Closure> tempMap){  
 for(Map.Entry<String,Closure> entry:tempMap.entrySet()){  
 String move=entry.getKey();  
 int index=entry.getValue().index;  
 *// 如果是终结符,ACTION* if(this.isVt(move)>=0){  
 if(this.actionTable[i][this.isVt(move)].equals("")|| this.actionTable[i][this.isVt(move)].equals("s"+index)){  
 this.actionTable[i][this.isVt(move)]="s"+index;  
 }else{  
 System.*out*.println("不是SLR");  
 }  
  
 }  
 *// 如果是非终结符,GOTO* else{  
 if(this.gotoTable[i][this.isVn(move)].equals("") || this.gotoTable[i][this.isVn(move)].equals(index+"")){  
 this.gotoTable[i][this.isVn(move)]=index+"";  
 }else{  
 System.*out*.println("不是SLR");  
 }  
 }  
 }  
}  
*// 归约*void reduce(int i,Grammar grammar) {  
 *// 查找语句所在编号* int index = this.grammarReader.inGrammarList(grammar);  
 *// 查找左部的FOLLOW* Set<String> tempFOLLOW = this.grammarReader.FOLLOW.get(grammar.left\_);  
 *// 遍历FOLLOW* for (String s : tempFOLLOW) {  
 if (this.actionTable[i][this.isVt(s)].equals("") || this.actionTable[i][this.isVt(s)].equals("r"+index)) {  
 this.actionTable[i][this.isVt(s)] = "r" + index;  
 } else {  
 System.*out*.println("不是SLR");  
 }  
 }  
}

**GrammaticalAnalysis**



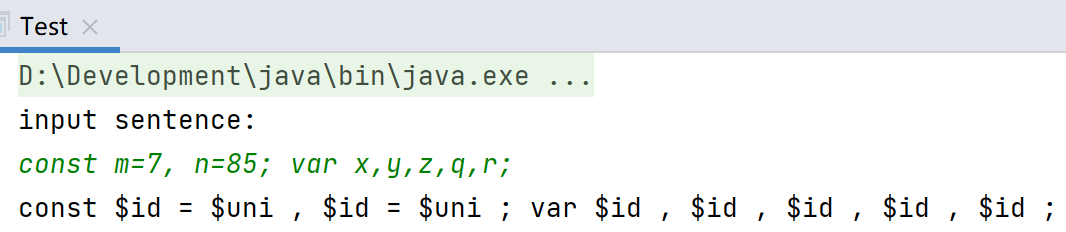
public void grammaticalAnalyze(Queue<String> inputQueue,GrammarAnalysisTable grammarAnalysisTable) throws IOException {  
 File file=new File("grammarAnalysisRecord.csv");  
 if(file.exists()) {file.delete();}  
 file.createNewFile();  
  
 FileWriter fileWriter=new FileWriter(file.getAbsoluteFile());  
 BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);  
  
 bufferedWriter.write("步骤\t状态栈\t符号栈\t输入串\t动作\n");  
 String input="";  
 for (String s : inputQueue) {  
 input+=s;  
 }  
 bufferedWriter.write("1\t0\t#\t"+input+"\t\n");  
  
 int flag=1;  
 int index=1;  
 *// 把状态0 push进状态栈* this.statusStack.push(0);  
 while(!inputQueue.isEmpty()){  
 *// 当前输入进的字符* String tempWord=inputQueue.peek();  
 *// 当前状态栈的栈顶* int currentStatus=this.statusStack.peek();  
 *// 查找tempWord对应的编号* int vtIndex = grammarAnalysisTable.isVt(tempWord);  
 *// 获取对应状态* String newStatus = this.actionTable[currentStatus][vtIndex];  
 *// 出错* if(newStatus.equals("")){  
 System.*out*.println("1出错");  
 this.error(inputQueue,tempWord);  
 flag=0;  
 break;  
 }  
  
 *// 移进* if(newStatus.charAt(0)=='s'){  
 shift(inputQueue,newStatus,tempWord);  
 }  
 *// 规约* else{  
 Grammar tempGrammar=this.reduce(newStatus);  
 *// 获取新状态  
 // 查找文法左部对应的编号* int vnIndex = grammarAnalysisTable.isVn(tempGrammar.left\_);  
 currentStatus=this.statusStack.peek();  
 String nextStatus=this.gotoTable[currentStatus][vnIndex];  
  
 *// 出错* if(nextStatus.equals("")){  
 System.*out*.println("2出错");  
 this.error(inputQueue,tempWord);  
 flag=0;  
 break;  
 }  
  
 *// 把新状态push进状态栈* this.statusStack.push(Integer.*parseInt*(nextStatus));  
  
 }  
 index++;  
 input="";  
 for (String s : inputQueue) {  
 input+=s+"";  
 }  
 this.showRecord(bufferedWriter,input,index,newStatus);  
 }  
  
 *// 如果符号栈里还有* String S=grammarList.get(0).left\_;  
 int vtIndex = grammarAnalysisTable.isVt("ε");  
 while(wordStack.size()!=1 || !S.equals(wordStack.peek())){  
 *// 当前状态栈的栈顶* int currentStatus=this.statusStack.peek();  
 *// 获取对应状态* String newStatus = this.actionTable[currentStatus][vtIndex];  
 *// 报错* if(newStatus.equals("")){  
 *//顺序输出符号栈* error(inputQueue," ");  
 flag=0;  
 break;  
 }  
  
 *// 规约* Grammar tempGrammar=this.reduce(newStatus);  
  
 if(S.equals(wordStack.peek())){  
 index++;  
 this.showRecord(bufferedWriter,"\t",index,newStatus);  
 break;  
 }  
  
 *// 获取新状态  
 // 查找文法左部对应的编号* int vnIndex = grammarAnalysisTable.isVn(tempGrammar.left\_);  
 currentStatus=this.statusStack.peek();  
 String nextStatus=this.gotoTable[currentStatus][vnIndex];  
  
 *// 出错* if(nextStatus.equals("")){  
 System.*out*.println("4出错");  
 this.error(inputQueue," ");  
 }  
 *// 把新状态push进状态栈* this.statusStack.push(Integer.*parseInt*(nextStatus));  
  
 index++;  
 this.showRecord(bufferedWriter,",",index,newStatus);  
 }  
   
 this.wordStack.clear();  
 this.statusStack.clear();  
 if(flag==1){  
 System.*out*.println("accept");  
 }  
 bufferedWriter.close();  
}

*// 出错*private void error(Queue<String> inputQueue,String tempWord){  
 *//顺序输出符号栈* for (String s : wordStack) {  
 System.*out*.print(s+" ");  
 }  
 System.*out*.print("※");  
 System.*out*.print(tempWord+" ");  
 for (String s : inputQueue) {  
 System.*out*.print(s);  
 }  
}  
private void shift(Queue<String> inputQueue,String newStatus,String tempWord){  
 *// 队首出队* inputQueue.remove();  
 *// 把新状态push进状态栈* this.statusStack.push(Integer.*parseInt*(newStatus.substring(1)));  
 *// 把tempWord push进符号栈* this.wordStack.push(tempWord);  
}  
private Grammar reduce(String newStatus){  
 *// 状态栈、符号栈弹出对应数量的word  
 // 获取规约的文法* Grammar tempGrammar=this.grammarList.get(Integer.*parseInt*(newStatus.substring(1)));  
 *// 获取对应数量：grammarList的第newStatus个文法右部的size* int num=0;  
 if("ε".equals(tempGrammar.right\_.get(0))){  
 num=0;  
 }else{  
 num=tempGrammar.right\_.size();  
 }  
 *//弹出* for(int i=0;i<num;i++){  
 this.statusStack.pop();  
 this.wordStack.pop();  
 }  
 *// 把文法左部push进符号栈* this.wordStack.push(tempGrammar.left\_);  
 return tempGrammar;  
}

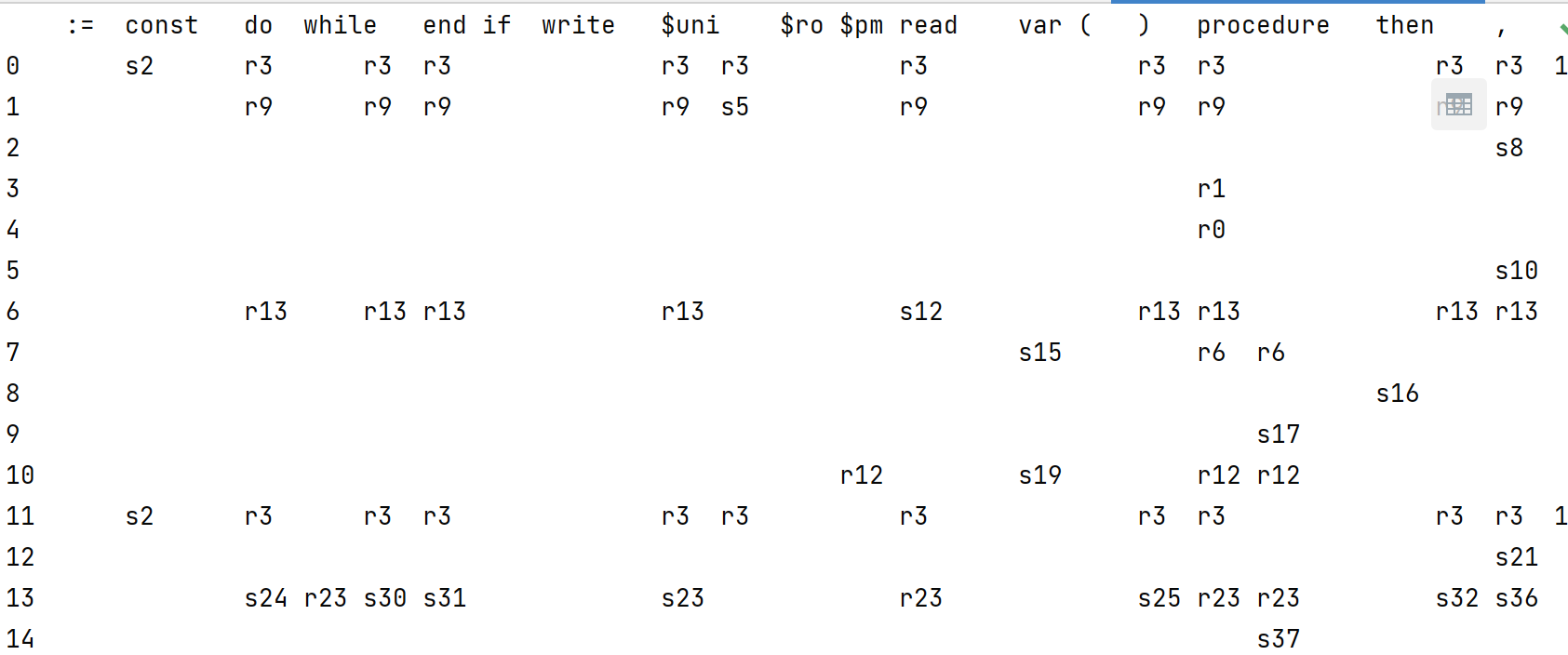
**实验结果与分析：**

输入串：const m=7, n=85; var x,y,z,q,r;

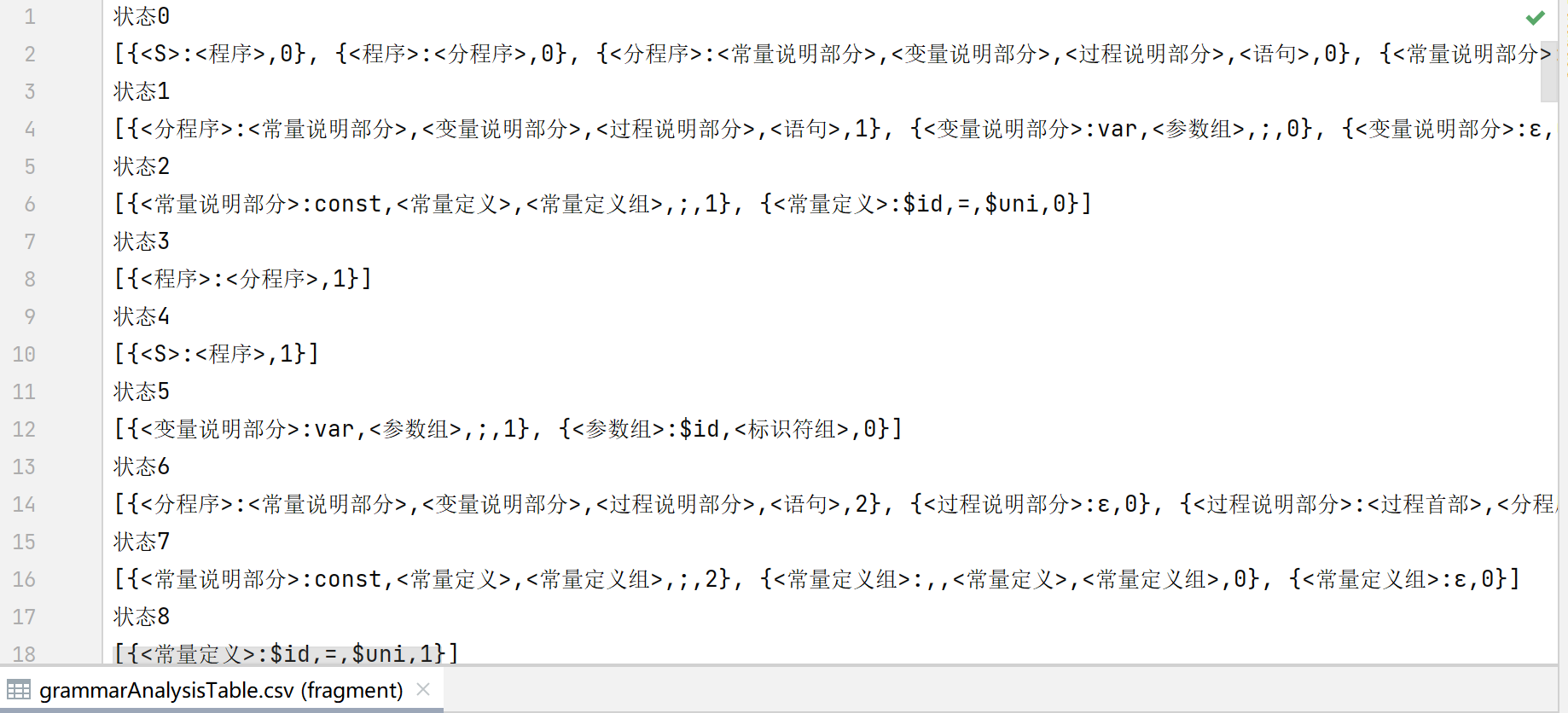
词法分析：



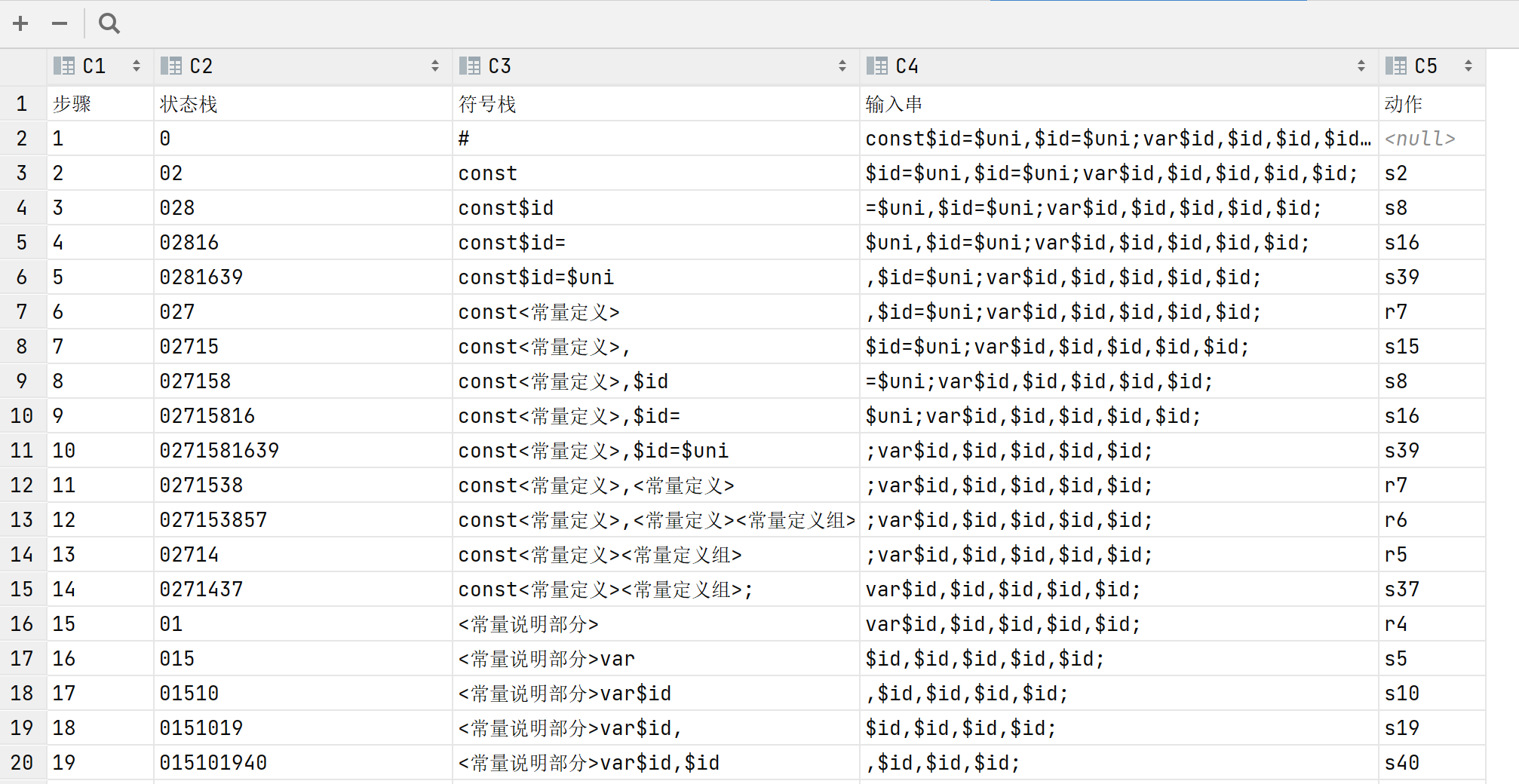
语法分析表：

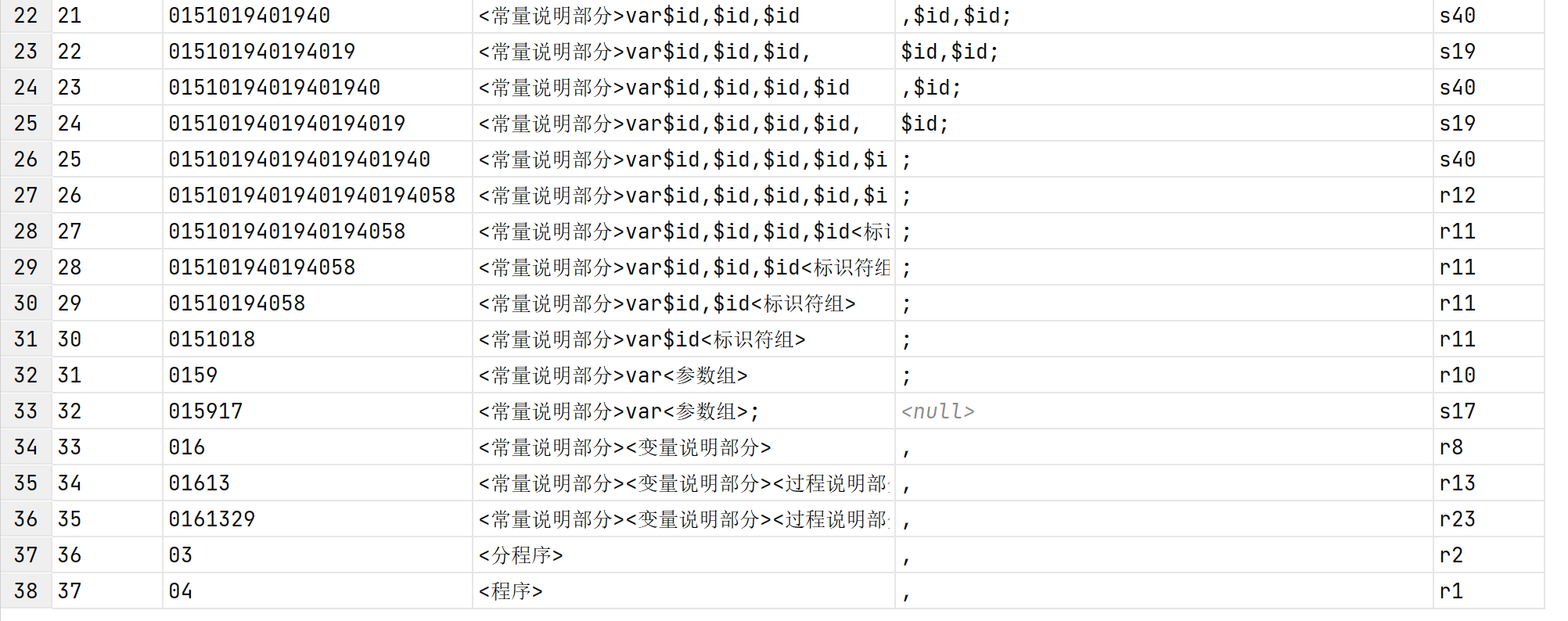


状态图：



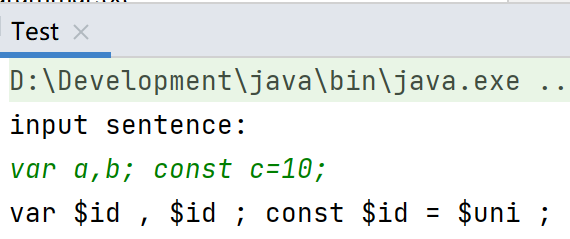
语法分析：



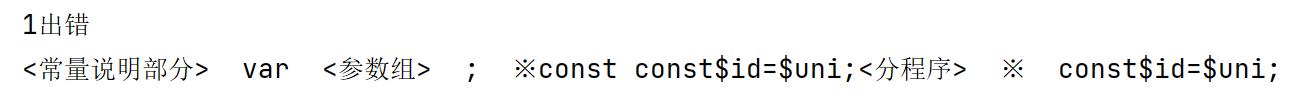


输入串：var a,b; const c=10;

词法分析：

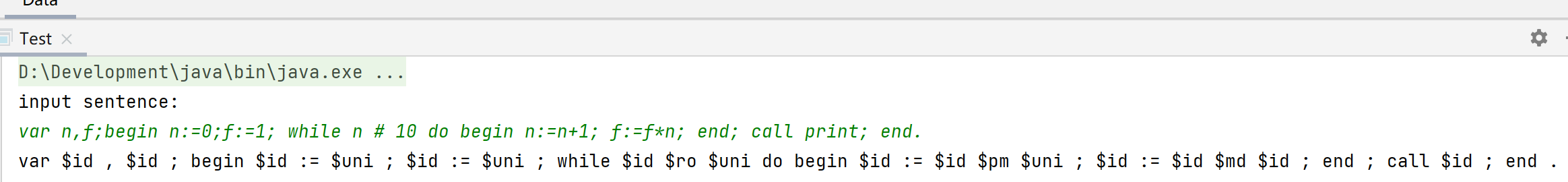


语法分析：



输入串：var n,f;begin n:=0;f:=1; while n # 10 do begin n:=n+1; f:=f\*n; end; call print; end.

词法分析：



语法分析：

（部分）

