实验编号：2**四川师大 编译原理 实验报告 2020**年**4**月**12**日

**计算机科学学院** 2017级 4 班 实验名称： LL(1)分析法

姓名：\_韩勇\_\_\_ 学号：\_2017110409\_ 指导老师：\_\_吴贞东\_\_ 实验成绩:\_\_\_\_\_

**实验 二 LL(1)分析法\_**

1. 实验目的及要求

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区 别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方法， 训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质，为培养适应 社会多方面需要的能力。

1、编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

2、如果遇到错误的表达式，应输出错误提示信息。

3、对下列文法，用LL（1）分析法对任意输入的符号串进行分析： （1）E->TG （2）G->+TG|—TG （3）G->ε （4）T->FS （5）S->\*FS|/FS （6）S->ε （7）F->(E) （8）F->i 输出的格式如下：

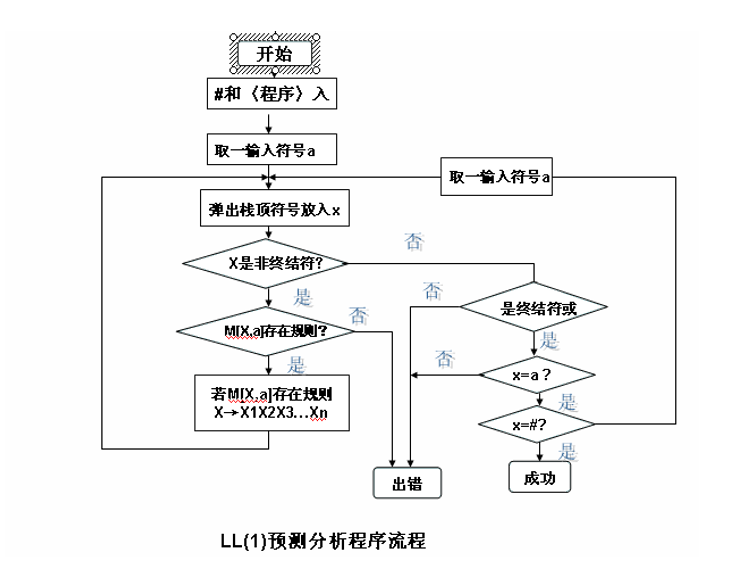


1. 实验内容

根据某一文法编制调试LL（1 ）分析程序，以便对任意输入的符号串进 行分析。 构造预测分析表，并利用分析表和一个栈来实现对上述程序设计语言的分析 程序。 分析法的功能是利用LL（1）控制程序根据显示栈栈顶内容、向前看符号 以及LL（1）分析表，对输入符号串自上而下的分析过程。

1. 实验主要流程、基本操作或核心代码、算法片段（该部分如不够填写，请另加附页）

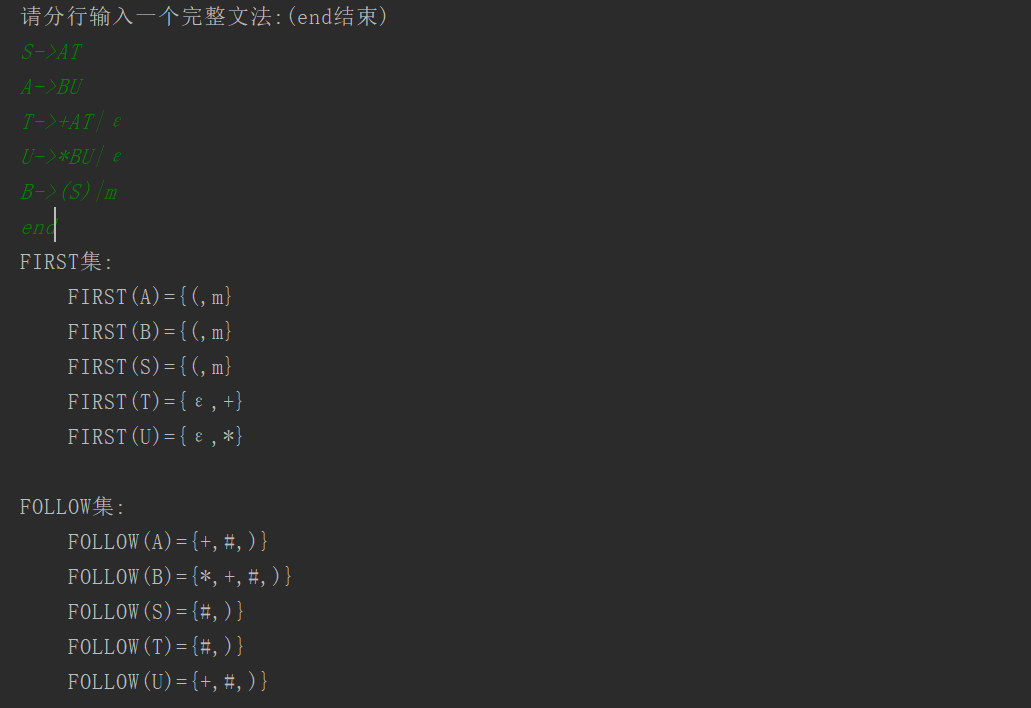
流程图：

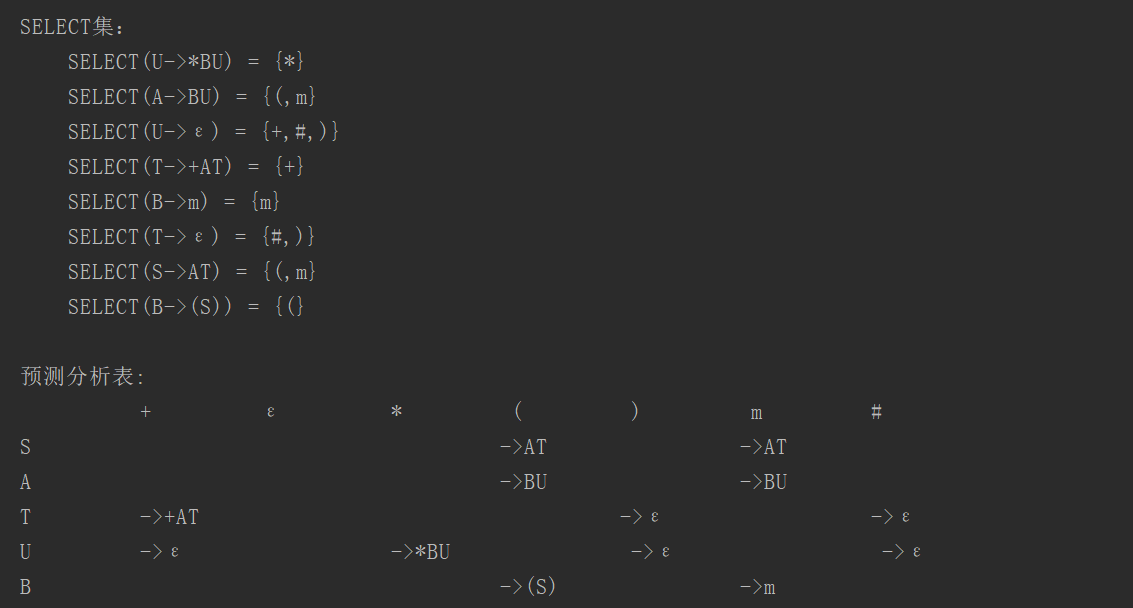


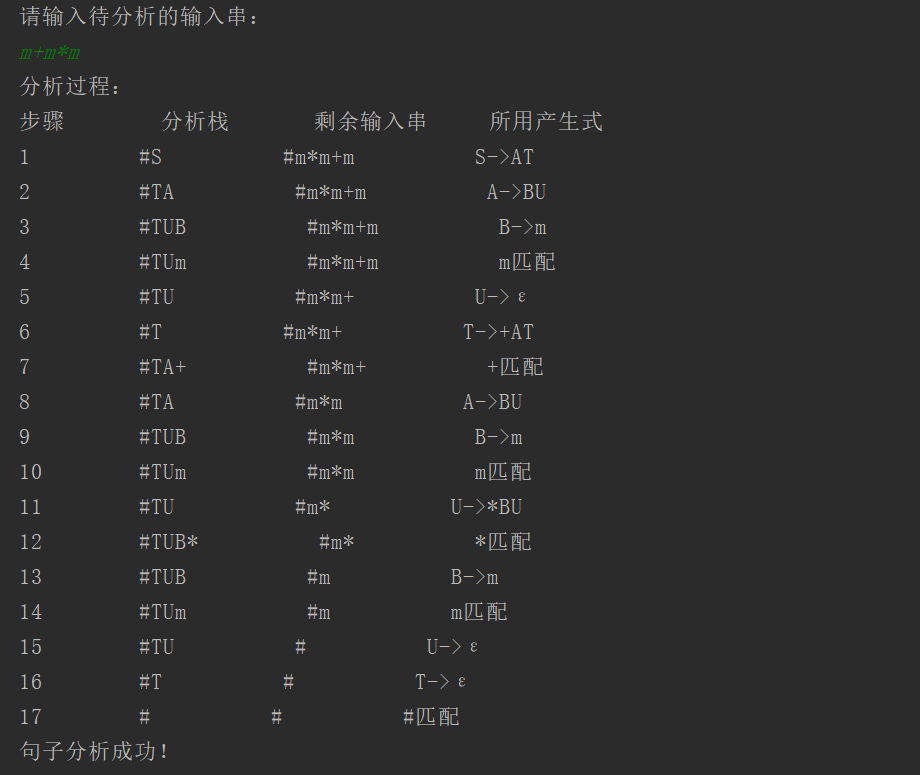
源程序：

package compiler\_test;  
import java.io.\*;  
import java.util.\*;  
  
public class Test\_02 {  
 public static void main(String[] args) {  
 //输入一个文法  
 ArrayList<String[]> input = *getInput*();  
 //求出所有的非终结符  
 ArrayList<String> allNonTerminals = *getAllNonTerminals*(input);  
 //求出能直接推出ε的非终结符  
 ArrayList<String> nonTerminalsNullable = *getAllNonTerminalsNullable*(input);  
 //求解FIRST集  
 HashMap<String, ArrayList<String>> first = *getFirst*(allNonTerminals, nonTerminalsNullable, input);  
 //输出FIRST集  
 *print*(first, "FIRST");  
 System.*out*.println();  
 //求解Follow集  
 HashMap<String, ArrayList<String>> follow = *getFollow*(allNonTerminals, first, input, nonTerminalsNullable);  
 //输出FOLLOW集  
 *print*(follow, "FOLLOW");  
 System.*out*.println();  
 //求解SELECT集  
 HashMap<String[], ArrayList<String>> select = *getSelect*(input, allNonTerminals, nonTerminalsNullable, first, follow);  
 //输出SELECT集  
 *printSelect*(select);  
 //输出预测分析表  
 System.*out*.println();  
 System.*out*.println("预测分析表:");  
 ArrayList<String> allTerminators = *getAllTerminator*(input);  
 allTerminators.add("#");  
 System.*out*.printf("%-10s", "");  
 for (String s : allTerminators) {  
 System.*out*.printf("%-10s", s);  
 }  
 System.*out*.println();  
 for (String nonTerminator : allNonTerminals) {  
 System.*out*.printf("%-10s", nonTerminator);  
 for (String terminators : allTerminators) {  
 String result = *getResult*(select, nonTerminator, terminators);  
 System.*out*.printf("%-10s", result);  
 }  
 System.*out*.println();  
 }  
  
 //输出预测分析过程  
 System.*out*.println("请输入待分析的输入串：");  
 Scanner scan = new Scanner(System.*in*);  
 String formula = scan.next();  
 char[] chars = formula.toCharArray();  
 Stack<String> analysisStack = new Stack<>();  
 Stack<String> rest = new Stack<>();  
 //# 和 剩余串进剩余串栈  
 rest.push("#");  
 for (int i = chars.length - 1; i >= 0; i--) {  
 rest.push(String.*valueOf*(chars[i]));  
 }  
 //# 和 文法开始符号进分析栈  
 analysisStack.push("#");  
 analysisStack.push(input.get(0)[0]);  
 //分析栈栈顶符号  
 String topOfAnalysisStack;  
 //剩余串栈顶符号  
 String topOfRest;  
 System.*out*.println("分析过程：");  
 System.*out*.printf("%-10s", "步骤");  
 System.*out*.printf("%-10s", "分析栈");  
 System.*out*.printf("%-10s", "剩余输入串");  
 System.*out*.printf("%-10s", "所用产生式");  
 System.*out*.println();  
 //步骤数  
 int cnt = 1;  
 while (true) {  
 //输出步骤数，分析栈和剩余符号串  
 System.*out*.printf("%-10s", cnt++);  
 *printStack*(analysisStack);  
 *printStack*(rest);  
 topOfAnalysisStack = analysisStack.lastElement();  
 topOfRest = rest.lastElement();  
 //如果两栈顶符号相同，则出栈  
 if (topOfAnalysisStack.equals(topOfRest)) {  
 System.*out*.println(analysisStack.lastElement() + "匹配");  
 if (topOfAnalysisStack.equals("#")) {  
 System.*out*.println("句子分析成功！");  
 break;  
 }  
 analysisStack.pop();  
 rest.pop();  
 //输出步骤数，分析栈和剩余符号串  
 System.*out*.printf("%-10s", cnt++);  
 *printStack*(analysisStack);  
 *printStack*(rest);  
 }  
 topOfAnalysisStack = analysisStack.lastElement();  
 topOfRest = rest.lastElement();  
 //求解应该用哪个产生式  
 String s = *getResult*(select, topOfAnalysisStack, topOfRest);  
 //如果产生替换  
 if (!s.equals("")) {  
 System.*out*.println(topOfAnalysisStack + s);  
 //分析栈顶元素出栈  
 analysisStack.pop();  
 char[] charArray = s.substring(2).toCharArray();  
 //产生式右部进栈  
 for (int i = charArray.length - 1; i >= 0; i--) {  
 if (!String.*valueOf*(charArray[i]).equals("ε")) {  
 analysisStack.push(String.*valueOf*(charArray[i]));  
 }  
 }  
 } else {  
 System.*out*.println("该式子不可识别！");  
 return;  
 }  
 }  
 }  
  
 */\*\*  
 \* 从栈顶符号开始输出栈  
 \*  
 \** ***@param*** *stack 等待输出的栈  
 \*/* public static void printStack(Stack<String> stack) {  
 for (String s : stack) {  
 System.*out*.print(s);  
 }  
 System.*out*.printf("%-10s", "");  
 }  
  
 */\*\*  
 \* 通过非终结符和终结符求出推导时应该选择的的产生式（构造预测分析表的时候用）  
 \*  
 \** ***@param*** *select select集  
 \** ***@param*** *nonTerminator 非终结符  
 \** ***@param*** *terminator 终结符  
 \** ***@return*** *""或者产生式的右部  
 \*/* public static String getResult(HashMap<String[], ArrayList<String>> select, String nonTerminator, String terminator) {  
 Set<String[]> keySet = select.keySet();  
 for (String[] key : keySet) {  
 if (key[0].equals(nonTerminator)) {  
 if (select.get(key).contains(terminator)) {  
 return "->" + key[1];  
 }  
 }  
 }  
 return "";  
 }  
  
 */\*\*  
 \* 求解Select集  
 \*  
 \** ***@param*** *input 输入的文法  
 \** ***@param*** *allNonTerminals 所有的非终结符  
 \** ***@param*** *nonTerminalsNullable 可推导出空的非终结符  
 \** ***@param*** *first first集  
 \** ***@param*** *follow follow集  
 \** ***@return*** *select集  
 \*/* public static HashMap<String[], ArrayList<String>> getSelect(ArrayList<String[]> input, ArrayList<String> allNonTerminals, ArrayList<String> nonTerminalsNullable, HashMap<String, ArrayList<String>> first, HashMap<String, ArrayList<String>> follow) {  
 HashMap<String[], ArrayList<String>> select = new HashMap<>();  
 String left;//产生式的左部  
 String right;//产生式的右部  
 char[] chars;//将产生式的右部转为字符数组  
 for (String[] strings : input) {  
 left = strings[0];  
 right = strings[1];  
 chars = right.toCharArray();  
 //α不能 ->ε  
 if (!*isAbleToNull*(nonTerminalsNullable, chars)) {  
 ArrayList<String> stringFirst = *getStringFirst*(allNonTerminals, first, chars);  
 String[] formula = {left, right};  
 select.put(formula, stringFirst);  
 } else {//α能->ε  
 ArrayList<String> stringFirst = *getStringFirst*(allNonTerminals, first, chars);  
 ArrayList<String> leftFollow = follow.get(left);  
 stringFirst.remove("ε");  
 ArrayList<String> res = *addTwoArrayList*(stringFirst, leftFollow);  
 String[] formula = {left, right};  
 select.put(formula, res);  
 }  
 }  
 return select;  
 }  
  
  
 */\*\*  
 \* 输出select集合  
 \*  
 \** ***@param*** *select 被输出的select集合  
 \*/* public static void printSelect(HashMap<String[], ArrayList<String>> select) {  
 System.*out*.println("SELECT集：");  
 Set<String[]> keySet = select.keySet();  
 for (String[] key : keySet) {  
 String left = "SELECT(" + key[0] + "->" + key[1] + ") = ";  
 StringBuilder right = new StringBuilder();  
 ArrayList<String> chars = select.get(key);  
 for (int i = 0; i < chars.size(); i++) {  
 if (i == 0) {  
 right.append(chars.get(i));  
 } else {  
 right.append(",").append(chars.get(i));  
 }  
 }  
 System.*out*.println("\t" + left + "{" + right + "}");  
 }  
 }  
  
 */\*\*  
 \* 输出first或者follow集  
 \*  
 \** ***@param*** *map 待输出的first集huozhe follow集  
 \** ***@param*** *firstOrFollow 输出的是first集或者follow集  
 \*/* public static void print(HashMap<String, ArrayList<String>> map, String firstOrFollow) {  
 if (firstOrFollow.equals("FIRST")) {  
 System.*out*.println("FIRST集:");  
 } else if (firstOrFollow.equals("FOLLOW")) {  
 System.*out*.println("FOLLOW集:");  
 }  
 Set<String> keys = map.keySet();  
 for (String key : keys) {  
 StringBuilder res = new StringBuilder();  
 ArrayList<String> chars = map.get(key);  
 for (int i = 0; i < chars.size(); i++) {  
 if (i == 0) {  
 res.append(chars.get(i));  
 } else {  
 res.append(",").append(chars.get(i));  
 }  
 }  
 System.*out*.println("\t" + firstOrFollow + "(" + key + ")" + "=" + "{" + res + "}");  
 }  
 }  
  
 */\*\*  
 \* 判断某个产生式是否可推出ε  
 \*  
 \** ***@param*** *nonTerminalsNullable 所有能推出ε的非终结符  
 \** ***@param*** *chars 产生式的右部  
 \** ***@return*** *true or false  
 \*/* public static boolean isAbleToNull(ArrayList<String> nonTerminalsNullable, char[] chars) {  
 if (chars.length == 1 && chars[0] == 'ε') {  
 return true;  
 } else {  
 for (char aChar : chars)  
 if (!nonTerminalsNullable.contains(String.*valueOf*(aChar))) {  
 return false;  
 }  
 return true;  
 }  
 }  
  
  
 */\*\*  
 \* 求解串的FIRST集  
 \*  
 \** ***@param*** *allNonTerminals 所有的非终结符  
 \** ***@param*** *first first集  
 \** ***@param*** *chars 待求串  
 \** ***@return*** *串的first集合  
 \*/* public static ArrayList<String> getStringFirst  
 (ArrayList<String> allNonTerminals, HashMap<String, ArrayList<String>> first,  
 char[] chars) {  
 ArrayList<String> stringFirst = new ArrayList<>();  
 for (int i = 0; i < chars.length; i++) {  
 //1.向stringFirst中加入First(char[0])的非ε元素  
 if (i == 0) {  
 ArrayList<String> currentFirst = *getCharFirst*(allNonTerminals, first, chars[i]);  
 for (String s : currentFirst) {  
 if (!s.equals("ε") && !stringFirst.contains(s)) {  
 stringFirst.add(s);  
 }  
 }  
 }  
 //2如果current为最后一个字符，且first(current)包含ε，则将ε加入stringFirst  
 if (i == chars.length - 1) {  
 ArrayList<String> currentFirst = *getCharFirst*(allNonTerminals, first, chars[i]);  
 if (currentFirst.contains("ε")) {  
 if (!stringFirst.contains("ε")) {  
 stringFirst.add("ε");  
 }  
 }  
 } else {//current不是最后一个字符  
 ArrayList<String> currentFirst = *getCharFirst*(allNonTerminals, first, chars[i]);  
 ArrayList<String> nextFirst = *getCharFirst*(allNonTerminals, first, chars[i + 1]);  
  
 if (currentFirst.contains("ε")) {//3.1如果FIRST(current)包含ε，向stringFirst中加入First(char[i+1])的非ε元素  
 for (String s : nextFirst) {  
 if (!s.equals("ε") && !stringFirst.contains(s)) {  
 stringFirst.add(s);  
 }  
 }  
 } else { //3.2如果FIRST(current)不包含ε,向stringFirst中加入First(current)的非ε元素并break  
 for (String s : currentFirst) {  
 if (!s.equals("ε") && !stringFirst.contains(s)) {  
 stringFirst.add(s);  
 }  
 }  
 break;  
 }  
 }  
 }  
 return stringFirst;  
 }  
  
 */\*\*  
 \* 求解一个符号的first集  
 \*  
 \** ***@param*** *allNonTerminals 所有的非终结符  
 \** ***@param*** *first first集  
 \** ***@param*** *c 等待求解的字符  
 \** ***@return*** *ArrayList  
 \*/* public static ArrayList<String> getCharFirst  
 (ArrayList<String> allNonTerminals, HashMap<String, ArrayList<String>> first, char c) {  
 ArrayList<String> charFirst = new ArrayList<>();  
 String theChar = String.*valueOf*(c);  
 if (theChar.equals("ε")) {  
 charFirst.add("ε");  
 }  
 //如果字符为终结符  
 if (!allNonTerminals.contains(theChar)) {  
 charFirst.add(theChar);  
 } else {//如果为非终结符  
 ArrayList<String> theFirst = first.get(theChar);  
 for (String s : theFirst) {  
 if (!charFirst.contains(s)) {  
 charFirst.add(s);  
 }  
 }  
 }  
 return charFirst;  
 }  
  
 */\*\*  
 \* 两个arrayList相加  
 \*  
 \** ***@param*** *firstArrayList 第一个list  
 \** ***@param*** *secondArrayList 第二个list  
 \** ***@return*** *两个list相加的结果  
 \*/* public static ArrayList<String> addTwoArrayList  
 (ArrayList<String> firstArrayList, ArrayList<String> secondArrayList) {  
 ArrayList<String> res = new ArrayList<>();  
 for (String s : firstArrayList) {  
 if (!res.contains(s)) {  
 res.add(s);  
 }  
 }  
 for (String s : secondArrayList) {  
 if (!res.contains(s)) {  
 res.add(s);  
 }  
 }  
 return res;  
 }  
  
  
 /\*  
 求解FOLLOW集的过程  
 1.设S为文法的开始符号，把{#}加入FOLLOW(S)中  
 2.若A->aBC 则把First(C)除了ε的元素加入到Follow(B)中  
 3.若C->ε 则把Follow(A)也加入Follow(B)中  
\*/  
 */\*\*  
 \* 求解follow集合  
 \*  
 \** ***@param*** *allNonTerminals 文法中所有的非终结符  
 \** ***@param*** *first 文法所有非终结符的first集  
 \** ***@param*** *input 输入的文法  
 \** ***@param*** *nonTerminalsNullable 可以直接推导出ε的非终结符  
 \** ***@return*** *follow集  
 \*/* public static HashMap<String, ArrayList<String>> getFollow(ArrayList<String> allNonTerminals, HashMap<String, ArrayList<String>> first, ArrayList<String[]>  
 input, ArrayList<String> nonTerminalsNullable) {  
 HashMap<String, ArrayList<String>> follow = new HashMap<>();  
 //先将非终结符加入到follow中  
 for (String s : allNonTerminals) {  
 follow.put(s, new ArrayList<>());  
 }  
 //将#加入文法开始符号的Follow集中  
 ArrayList<String> list = new ArrayList<>();  
 list.add("#");  
 follow.put(input.get(0)[0], list);  
  
 HashMap<String, ArrayList<String>> beforeLoop = *clone*(follow);  
 HashMap<String, ArrayList<String>> afterLoop = new HashMap<>();  
 while (!beforeLoop.equals(afterLoop)) {  
 beforeLoop = *clone*(follow);  
 for (String[] strings : input) {  
 //先得到产生式的右部  
 String productionFormula = strings[1];  
 //将其转化为字符数组  
 char[] chars = productionFormula.toCharArray();  
 //遍历当前右部  
 for (int j = 0; j < chars.length; j++) {  
 //如果第j个字符是终结符  
 if (!Character.*isUpperCase*(chars[j])) {  
 if (j >= 1) {  
 //如果第j-1字符为非终结符  
 if (Character.*isUpperCase*(chars[j - 1])) {  
 ArrayList<String> jPreFollow = follow.get(String.*valueOf*(chars[j - 1]));  
 if (!jPreFollow.contains(String.*valueOf*(chars[j - 1]))) {  
 jPreFollow.add(String.*valueOf*(chars[j - 1]));  
 }  
 follow.put(String.*valueOf*(chars[j - 1]), jPreFollow);  
 }  
 }  
 } else { //第j个字符是非终结符  
 if (j + 1 < chars.length) {  
 //第j个字符后还有字符  
 if (!Character.*isUpperCase*(chars[j + 1])) {//第j+1个字符为终结符  
 //则将该终结符加入Follow(chars[j])中  
 ArrayList<String> jCharFollow = follow.get(String.*valueOf*(chars[j]));  
 if (!jCharFollow.contains(String.*valueOf*(chars[j + 1]))) {  
 jCharFollow.add(String.*valueOf*(chars[j + 1]));  
 follow.put(String.*valueOf*(chars[j]), jCharFollow);  
 }  
 } else {//第j+1个字符为非终结符  
 //则将First(j+1)-ε加入Follow(chars(j))中  
 ArrayList<String> jNextCharFirst = first.get(String.*valueOf*(chars[j + 1]));  
 ArrayList<String> jCharFollow = follow.get(String.*valueOf*(chars[j]));  
 for (String s : jNextCharFirst) {  
 if (!s.equals("ε") && !jCharFollow.contains(s)) {  
 jCharFollow.add(s);  
 }  
 }  
 follow.put(String.*valueOf*(chars[j]), jCharFollow);  
 }  
 } else {//第j个字符为最后一个字符且是非终结符  
 //首先将Follow(左部)加到Follow(chars[j])中  
 ArrayList<String> leftFollow = follow.get(strings[0]);  
 ArrayList<String> jCharFollow = follow.get(String.*valueOf*(chars[j]));  
 for (String s : leftFollow) {  
 if (!jCharFollow.contains(s)) {  
 jCharFollow.add(s);  
 }  
 }  
 follow.put(String.*valueOf*(chars[j]), jCharFollow);  
 //如果chars[j]能->ε  
 if (nonTerminalsNullable.contains(String.*valueOf*(chars[j]))) {  
 //如果chars[j-1]为非终结符  
 if (Character.*isUpperCase*(chars[j - 1])) {  
 //将Follow(左部)加到Follow(chars[j-1])中  
 ArrayList<String> \_leftFollow = follow.get(strings[0]);  
 ArrayList<String> jPreCharFollow = follow.get(String.*valueOf*(chars[j - 1]));  
 for (String s : \_leftFollow) {  
 if (!jPreCharFollow.contains(s)) {  
 jPreCharFollow.add(s);  
 }  
 }  
 follow.put(String.*valueOf*(chars[j - 1]), jPreCharFollow);  
 }  
 }  
 }  
 }  
 }  
 }  
 afterLoop = *clone*(follow);  
 }  
 Set<String> keySet = follow.keySet();  
 for (String key : keySet) {  
 ArrayList<String> strings = follow.get(key);  
 strings.removeIf(allNonTerminals::contains);  
 }  
 return follow;  
 }  
  
  
 /\*  
 FIRST集求解过程  
 1.若右边第一个符号是终结符或 ε ，则直接将其加入 First（X）  
 2.若右边第一个符号是非终结符，则将这个非终结符的 First 集中的非 ε 元素加入 First（X）  
 3.若右边第一个符号是非终结符而且紧随其后的是很多个非终结符，这个时候就要注意是否有 ε 。  
 3.1.若第 i 个非终结符的 First 集有 ε ，则可将第 i+1 个非终结符去除 ε 的 First 集加入 First（X）。  
 3.2.若所有的非终结符都能够推导出 ε ，则将 ε 也加入到 First（X）  
 \*/  
  
 */\*\*  
 \* 求解First集合  
 \*  
 \** ***@param*** *allNonTerminals 文法中所有的非终结符  
 \** ***@param*** *nonTerminalsNullable 文法中可直接推出ε的非终结符  
 \** ***@param*** *input 输入的文法  
 \** ***@return*** *first集  
 \*/* public static HashMap<String, ArrayList<String>> getFirst(ArrayList<String> allNonTerminals, ArrayList<String> nonTerminalsNullable, ArrayList<String[]> input) {  
 HashMap<String, ArrayList<String>> first = new HashMap<>();  
 //先将非终结符加入到first的左部中  
 for (String s : allNonTerminals) {  
 first.put(s, new ArrayList<>());  
 }  
 //计算first集的过程  
 //1.1若右边第一个符号 ε ，则直接将其加入 First（X）  
 for (String s : nonTerminalsNullable) {  
 ArrayList<String> strings = first.get(s);  
 if (strings == null) {  
 strings = new ArrayList<>();  
 }  
 strings.add("ε");  
 first.put(s, strings);  
 }  
 //1.2若右边第一个符号是终结符 ，则直接将其加入 First（X）  
 for (String[] strings1 : input) {  
 String productionFormula = strings1[1];  
 if (!Character.*isUpperCase*(productionFormula.charAt(0)) && !productionFormula.equals("ε")) {  
 ArrayList<String> strings = first.get(strings1[0]);  
 if (strings == null) {  
 strings = new ArrayList<>();  
 }  
 strings.add(String.*valueOf*(productionFormula.charAt(0)));  
 first.put(strings1[0], strings);  
 }  
 }  
 HashMap<String, ArrayList<String>> beforLoop = *clone*(first);  
 HashMap<String, ArrayList<String>> afterLoop = new HashMap<>();  
 while (!beforLoop.equals(afterLoop)) {  
 beforLoop = *clone*(first);  
 //2.若右边第一个符号是非终结符，则将其 First 集的的非 ε 元素加入 First（X）  
 for (String[] strings1 : input) {  
 String productionFormula = strings1[1];  
 if (Character.*isUpperCase*(productionFormula.charAt(0))) {  
 //firstNonTerminal 产生式右部的第一个非终结符  
 String firstNonTerminal = String.*valueOf*(productionFormula.charAt(0));  
 //currentNonTerminal 产生式左部的非终结符  
 String currentNonTerminal = strings1[0];  
 //currentFirst 当前产生式左部的first集  
 ArrayList<String> currentFirst = first.get(currentNonTerminal);  
 //firstNonTerminalFirst 当前产生式右部的第一个非终结符的first集合  
 ArrayList<String> firstNonTerminalFirst = first.get(firstNonTerminal);  
 for (String s : firstNonTerminalFirst) {  
 if (!s.equals("ε") && !currentFirst.contains(s)) {  
 currentFirst.add(s);  
 }  
 }  
 first.put(currentNonTerminal, currentFirst);  
 }  
 }  
 //若右边第一个符号是非终结符而且紧随其后的是很多个非终结符，这个时候就要注意是否有 ε  
 //3.1 第 i 个非终结符的 First 集有 ε ，则可将第 i+1 个非终结符去除 ε 的 First 集加入 First（X）  
 //3.2 若所有的非终结符都能够推导出 ε ，则将 ε 也加入到 First（X）  
 for (String[] strings : input) {  
 String productionFormula = strings[1];  
 ArrayList<String> currentFirst = first.get(strings[0]);  
 //产生式右部第一个符号为非终结符  
 if (Character.*isUpperCase*(productionFormula.charAt(0))) {  
 //将产生式转为字符数组  
 char[] chars = productionFormula.toCharArray();  
 //遍历字符数组  
 for (int j = 0; j < chars.length; j++) {  
 //如果碰到终结符  
 if (!Character.*isUpperCase*(chars[j])) {  
 break;  
 }  
 //第j个字符的first集 jCharFirst  
 ArrayList<String> jCharFirst = first.get(String.*valueOf*(chars[j]));  
 if (jCharFirst != null && jCharFirst.contains("ε")) {  
 if (j + 1 < chars.length) {  
 if (Character.*isUpperCase*(chars[j + 1])) {  
 ArrayList<String> jNextFirst = first.get(String.*valueOf*(chars[j + 1]));  
 for (String s : jNextFirst) {  
 if (!s.equals("ε") && !currentFirst.contains(s)) {  
 currentFirst.add(s);  
 }  
 }  
 } else {  
 if (!String.*valueOf*(chars[j + 1]).equals("ε") && !currentFirst.contains(String.*valueOf*(chars[j + 1]))) {  
 currentFirst.add(String.*valueOf*(chars[j + 1]));  
 }  
 }  
 first.put(strings[0], currentFirst);  
 } else {  
 if (!currentFirst.contains("ε")) {  
 currentFirst.add("ε");  
 first.put(strings[0], currentFirst);  
 }  
 }  
 } else {  
 break;  
 }  
 }  
 }  
 }  
 afterLoop = *clone*(first);  
 }  
 return first;  
 }  
  
 */\*\*  
 \* 对象深度复制(对象必须是实现了Serializable接口)  
 \*  
 \** ***@param*** *obj 被复制的对象  
 \** ***@return*** *T  
 \** ***@author*** *Muscleape  
 \*/* @SuppressWarnings("unchecked")  
 public static <T extends Serializable> T clone(T obj) {  
 T clonedObj = null;  
 try {  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 ObjectOutputStream oos = new ObjectOutputStream(baos);  
 oos.writeObject(obj);  
 oos.close();  
  
 ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());  
 ObjectInputStream ois = new ObjectInputStream(bais);  
 clonedObj = (T) ois.readObject();  
 ois.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return clonedObj;  
 }  
  
 */\*\*  
 \* 计算所有可以推出空的非终结符  
 \*  
 \** ***@param*** *input 输入的文法  
 \** ***@return*** *非空终结符集合  
 \*/* public static ArrayList<String> getAllNonTerminalsNullable(ArrayList<String[]> input) {  
 ArrayList<String> nonTerminalsNullable = new ArrayList<>();  
 for (String[] strings : input) {  
 if (strings[1].equals("ε")) {  
 nonTerminalsNullable.add(strings[0]);  
 }  
 }  
 return nonTerminalsNullable;  
 }  
  
 */\*\*  
 \* 求出所有的非终结符  
 \*  
 \** ***@param*** *input 输入的文法  
 \** ***@return*** *非终结符集合  
 \*/* public static ArrayList<String> getAllNonTerminals(ArrayList<String[]> input) {  
 ArrayList<String> nonTerminals = new ArrayList<>();  
 for (String[] strings : input) {  
 String nonTerminal = strings[0];  
 if (!nonTerminals.contains(nonTerminal)) {  
 nonTerminals.add(nonTerminal);  
 }  
 }  
 return nonTerminals;  
 }  
  
 */\*\*  
 \* 求出所有的终结符  
 \*  
 \** ***@param*** *input 输入的文法  
 \** ***@return*** *非终结符集合  
 \*/* public static ArrayList<String> getAllTerminator(ArrayList<String[]> input) {  
 ArrayList<String> terminals = new ArrayList<>();  
 for (String[] strings : input) {  
 String right = strings[1];  
 char[] chars = right.toCharArray();  
 for (char aChar : chars) {  
 if (!Character.*isUpperCase*(aChar) && !terminals.contains(String.*valueOf*(aChar))) {  
 terminals.add(String.*valueOf*(aChar));  
 }  
 }  
 }  
 return terminals;  
 }  
  
 */\*\*  
 \* 从控制台输入一个文法，将其保存到ArrayList<String[]>中，  
 \* 其中String[0]保存非终结符  
 \* String[1]保存产生式  
 \*  
 \** ***@return*** *输入的文法  
 \*/* public static ArrayList<String[]> getInput() {  
 ArrayList<String[]> input = new ArrayList<>();  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.println("请分行输入一个完整文法:(end结束)");  
 String sline;  
 sline = sc.nextLine();  
 while (!sline.startsWith("end")) {  
 StringBuilder buffer = new StringBuilder(sline);  
 int l = buffer.indexOf(" ");  
 //去除空格  
 while (l >= 0) {  
 buffer.delete(l, l + 1);  
 l = buffer.indexOf(" ");  
 }  
 sline = buffer.toString();  
 //s存储左推导符（既非终结符）  
 String[] s = sline.split("->");  
 if (s.length == 1) {  
 System.*out*.println("文法有误");  
 System.*exit*(0);  
 }  
 //使用StringTokenizer的原因是可以指定两种类型的分隔符|︱（中文和英文）  
 /\*  
 \* private StringTokenizer(String str,String delim,boolean returnDelims)  
 \* str - 要解析的字符串。  
 \* delim - 分隔符。  
 \* returnDelims - 指示是否将分隔符作为标记返回的标志。  
 \*/  
 StringTokenizer fx = new StringTokenizer(s[1], "|︱");  
 //如果产生式的右部出现了 | 则按多条产生式进行存储  
 while (fx.hasMoreTokens()) {  
 String[] productionFormula = new String[2];  
 productionFormula[0] = s[0].trim();//0的位置放非终结符  
 productionFormula[1] = fx.nextToken().trim();//1的位置放导出的产生式  
 input.add(productionFormula);  
 }  
 sline = sc.nextLine();  
 }  
 return input;  
 }  
  
  
}

运行结果：







1. 实验结果的分析与评价（该部分如不够填写，请另加附页）

通过实验更好的了解了语言运行的根本，有利于我们以后更好的进行相关方面的学习。

注：实验成绩等级分为（90－100分）优，（80－89分）良，(70-79分)中，（60－69分）及格，（59分）不及格。