**《编译原理》实验报告**

|  |  |
| --- | --- |
| **实验名称** | LL(1) 语法分析实验 |
| 1. **实验目的及内容：**   1. 了解 LL(1)语法分析是如何根据语法规则逐一分析词法分析所得到的单词，检查语法错误，即掌握语法分析过程。  2. 掌握LL(1)文法判别调剂和 LL(1)语法分析器的设计与调试。  针对任意的文法，编写相应的左递归消除、左公共因子提取程序，求解相应的FIRST、FOLLOW集，构造预测分析表，并编写LL(1)语法分析程序，并给出测试句子的分析过程。（注：左递归消除和左公共因子如果在实验三里做了，可以直接拿过来用）  判断LL(1)文法部分：  1. 输入：文法  2. 处理：左递归消除、左公共因子提取，FIRST、FOLLOW等集合构造，判断LL(1)  3. 输出：是LL(1)的情况输出预测分析表，否则判断不是LL(1)  LL(1)分析程序部分：  1. 输入：诸如对应文法的符号串，以$结束。  2. 处理：基于分析表进行 LL(1)语法分析，判断其是否符合文法。  3. 输出：串是否合法。 | |
| 1. **实验过程及步骤：** 2. 将文法存储到.txt文件中，格式如下：   5  S  S:a  S:^  S:(T)  T:T,S  T:S  5  a  ^  (  )  文本的第一行表示文法产生式的条数，第二行表示该文法的开始符号，接下来对应几行表示文法的每一条产生式，接下来给出了终结符号的数目，最后每一行表示一个终结符号。   1. 套用实验3的消除直接左递归、消除间接左递归、提取左公共因子等函数，对初始的输入文法进行改造。 2. 编写firstset函数，传入的参数为：非终结符号r，文法mat，终结符号列表finalpart   如果该非终结符号的产生式右部第一个字符为终结符号，则该符号的first集加入该终结符号；如果该非终结符号的产生式右部推出空，则该符号的first集加入#；如果该非终结符号的产生式右部第一个字符为非终结符号，则该符号的first集更与右部第一个符号的first集   1. 编写followset函数，传入的参数为：非终结符号r，文法mat，非终结符号列表mynotfinalpart，终结符号列表thisfinalpart，开始符号mystartsign   如果传入的非终结符号是开始符号，则r的follow集加入$，否则，遍历文法每一条产生式的右部，找到传入的非终结符号r，如果r后面还跟着字符，如果该字符是终结符号，则将该终结符号加入r的follow集，如果该符号是非终结符号，则将该非终结符号的first集-# 加入r的follow集，符过该符号能推出#且该符号后面没有字符，则将产生式左边符号的follow集加入r的follow集，如果高符号右面还有字符，则将该字符的first集-# 加入r的follow集。如果传入的非终结符号r后面没有字符，如果该产生式的左部不等于r，则将左部的follow集加入r的follow集。   1. 编写select函数，传入的参数为：文法mat，产生式 onemat，非终结符号列表mynotfinalpart，终结符号列表thisfinalpart，开始符号mystartsign。   如果输入的产生式的右部第一个字符为终结符号，则该产生式的select集等于该终结符号；如果输入的产生式的右部第一个字符为非终结符号，则该产生式的select集等于该非终结符号的first集；如果输入的产生式推出#，则该产生式的select集等于该产生式左部的follow集。   1. 编写预测分析函数predictionAnalysis，传入的参数为：栈 thisinput，用map存储的产生式和其对应的select集，开始符号mystartsign。该函数开始先新建一个匹配栈mystack，栈顶加入$，栈顶加入开始符号mystartsign。   如果匹配栈mystack的栈顶不等于$则一直循环，如果匹配栈的栈顶等于输入栈的栈顶，则两个栈都抛出栈顶。否则，如果输入栈的栈顶等于某一select集中的某一元素，且匹配栈的栈顶等于该select集对应产生式的左部，则匹配栈抛出栈顶，把该产生式的右部从右到左压入匹配栈，如果该产生式推出空，则抛出匹配栈的栈顶。   1. 结合以上函数，综合编写main函数。   **程序源代码如下：**  package souce\_code;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.HashSet;  import java.util.List;  import java.util.Map;  import java.util.Stack;  public class ll1 {  public List<String> RemoveLeftRecursion(List<String> mat) {// 消除直接在左递归函数  List<String> after = new ArrayList<String>();  List<String> myreg = new ArrayList<String>();  char splitcharacter = ':';  for (int i = 0; i < mat.size(); i++) {  if (mat.get(i).charAt(2) == mat.get(i).charAt(0)) {  // 形如 E：E+T 。 转换为 e:+Te  String a = "";  a += dic(mat.get(i).charAt(0)); // 调用转换函数  a += splitcharacter; // ：  a += mat.get(i).substring(3); // 街区左递归之后的string  a += dic(mat.get(i).charAt(0)); // 调用转换函数  after.add(a);  String b = "";  b += dic(mat.get(i).charAt(0));  b += splitcharacter;  b += '#';  after.add(b);  myreg.add(mat.get(i));// 寄存一条数据,后面用来判断是否要添加集合  for (int j = 0; j < mat.size(); j++) {  // 形如E：T 当T不等于E且E == E 时  if (mat.get(i).charAt(0) != mat.get(j).charAt(2) && mat.get(j).charAt(0) == mat.get(i).charAt(0)) {  // 形如 E:T 转换为 E：Te  after.add(mat.get(j) + dic(mat.get(i).charAt(0)));  myreg.add(mat.get(j));  }  }  }  }  for (int i = 0; i < mat.size(); i++) {  for (int j = 0; j < myreg.size(); j++) {  if (mat.get(i).equals(myreg.get(j))) {  mat.remove(i);  }  }  }  for (int i = 0; i < mat.size(); i++) {  after.add(mat.get(i));  }  after = new ArrayList<String>(new HashSet<String>(after));  return after;  }  public List<String> RemoveAllRecursion(List<String> mat) {  List<String> afterall = new ArrayList<String>();  List<String> mynob = new ArrayList<String>();  for (int i = 0; i < mat.size(); i++) {  for (int j = 0; j < mat.size(); j++) {  if (mat.get(i).charAt(2) == mat.get(j).charAt(0) && mat.get(j).charAt(2) == mat.get(i).charAt(0)) {// 右部第一个字符等于左部的字符  String a = "";  a += mat.get(i).charAt(2);  afterall.add(mat.get(i).replaceAll(a, mat.get(j).substring(2)));// 进行替换  mynob.add(mat.get(i));  }  }  }  mynob = new ArrayList<String>(new HashSet<String>(mynob));  for (int i = 0; i < mat.size(); i++) {  for (int j = 0; j < mynob.size(); j++) {  if (mat.get(i).equals(mynob.get(j))) {  mat.remove(i);  }  }  }  // 把没有参与构造的加入  for (String t : mat) {  afterall.add(t);  }  // 把形如E:#H转换为E:H  for (int i = 0; i < afterall.size(); i++) {  for (int j = 0; j < afterall.get(i).length(); j++) {  if (afterall.get(i).charAt(j) == '#') {  if (j + 1 < afterall.get(i).length()) {  String a = "";  a += afterall.get(i).substring(0, j);  a += afterall.get(i).substring(j + 1);  afterall.remove(i);  afterall.add(a);  }  }  }  }  return afterall;  }  // 得到已被使用的字符之外的字符，用来改造  public List<Character> usetochange(List<Character> afterRemoveAllLeftRecursionRightpart) {  List<Character> list = new ArrayList<>();  for (int i = 0; i < 26; i++) {  list.add((char) (65 + i));  }  for (int i = 0; i < list.size(); i++) {  for (char t : afterRemoveAllLeftRecursionRightpart) {  if (list.get(i).equals(t)) {  list.remove(i);  }  }  }  return list;  }  // 提取左公共因子  public List<String> ExtractionLeftFactor(List<String> mat, List<Character> Leftpart) {  List<String> myregList = new ArrayList<String>();  String myfirstString = new String();  int regk = 0; // 用来指向第几个字符，用来分割文法  List<Integer> myintlist = new ArrayList<Integer>();// 记录那几条文法参与了改造  // 遍历list中每一个项  for (int i = 0; i < mat.size(); i++) {  for (int j = i; j < mat.size(); j++) {  // 如果右部第一个字符相等且左部相等  if (mat.get(i).charAt(0) == mat.get(j).charAt(0) && mat.get(i).charAt(2) == mat.get(j).charAt(2)  && i != j) {  myintlist.add(i);  myintlist.add(j);  // 遍历每一个字符，如果字符相等，且字符序号相等  for (int k = 0; k < mat.get(i).length(); k++) {  for (int l = k; l < mat.get(j).length(); l++) {  if (mat.get(i).charAt(k) == mat.get(j).charAt(l) && k == l) {  myfirstString += mat.get(i).charAt(k); // 把相等的字符加入string  regk = k + 1;  }  }  }  if (mat.get(i).equals(myfirstString)) {// 如果一个文法包含另一个文法，加入推出#和推出剩余不相等的符号  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + '#');  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + mat.get(j).substring(regk));  } else if (mat.get(j).equals(myfirstString)) {  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + '#');  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + mat.get(i).substring(regk));  } else {// 如果两个文法没有包含关系  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + mat.get(i).substring(regk));  myregList.add(usetochange(Leftpart).get(i).toString() + ':' + mat.get(j).substring(regk));  }  myfirstString += usetochange(Leftpart).get(i);  myregList.add(myfirstString);  myfirstString = "";  } else {  myregList.add(mat.get(j));  }  }  }  for (int i = 0; i < myregList.size(); i++) {  for (int j = 0; j < myintlist.size(); j++) {  if (myregList.get(i).equals(mat.get(myintlist.get(j)))) {  myregList.remove(i);  }  }  }  return myregList;  }  public char dic(char A) {// 转换函数,形如E转换成e  return (char) (A ^ ' ');  }  public String myFirstSet(char requiredValue, List<String> mat, List<Character> thisfinalpart) {// 传入最后的文法和最后的终结符号  String thisfirstset = "";  for (int i = 0; i < mat.size(); i++) {  for (char t : thisfinalpart) {  if (mat.get(i).charAt(0) == requiredValue && mat.get(i).charAt(2) == t) {// 非终结符号右部第一个符号为终结符号  thisfirstset += mat.get(i).charAt(2);  } else if (mat.get(i).charAt(0) == requiredValue && mat.get(i).charAt(2) == '#') {// 如果推出空  thisfirstset += '#';  } else {  for (int j = +1; j < mat.size(); j++) {  if (mat.get(i).charAt(0) == requiredValue && mat.get(i).charAt(2) == mat.get(j).charAt(0)) {// 递归查询  thisfirstset = myFirstSet(mat.get(j).charAt(0), mat, thisfinalpart);  }  }  }  }  }  for (char t : thisfinalpart) {  if (requiredValue == t) {// 终结符号的first等于本身  thisfirstset += t;  return thisfirstset;  }  }  thisfirstset = removeMethod(thisfirstset);  return thisfirstset;  }  public String myFollowSet(char requiredValue, List<String> mat, List<Character> mynotfinalpart,  List<Character> thisfinalpart, char mystartsign) {  List<Character> myfollowset = new ArrayList<Character>();  String thisfollowset = "";  List<String> thisrightpart = new ArrayList<String>();  for (int i = 0; i < mat.size(); i++) {// 提取右部  thisrightpart.add(mat.get(i).substring(2));  }  if (requiredValue == mystartsign) {// 如果是开始符号则加入$  thisfollowset += '$';  }  for (int i = 0; i < mat.size(); i++) {  for (int j = 0; j < thisrightpart.get(i).length(); j++) {  if (thisrightpart.get(i).charAt(j) == requiredValue) {// 找到字符  if (j + 1 < thisrightpart.get(i).length()) {// 如果字符后面还有字符  for (char t : thisfinalpart) {  if (thisrightpart.get(i).charAt(j + 1) == t) { // 如果该字符是终结符号则加入follow集  myfollowset.add(t);  } else {  for (char k : mynotfinalpart) {  if (thisrightpart.get(i).charAt(j + 1) == k) {// 如果该字符是非终结符号  String myfirstto = "";  myfirstto = myFirstSet(k, mat, thisfinalpart);// 把该符号的first集加入  for (int q = 0; q < myfirstto.length(); q++) {  myfollowset.add(myfirstto.charAt(q));  }  for (String p : mat) {  if (p.charAt(0) == k && p.charAt(2) == '#') {// 如果该非终结符号能推出#,把产生式左部符号的follow集加入  String myfollowto = "";  myfollowto = myFollowSet(p.charAt(0), mat, mynotfinalpart,  thisfinalpart, mystartsign);  for (int q = 0; q < myfollowto.length(); q++) {  myfollowset.add(myfollowto.charAt(q));  }  }  }  }  }  }  }  } else if (j + 1 == thisrightpart.get(i).length()) { // 后面没有字符  if (mat.get(i).charAt(0) != requiredValue) { // 如果请求的字符不等于产生式的左部，将左部的follow加入  String myfollowtoto = "";  myfollowtoto = myFollowSet(mat.get(i).charAt(0), mat, mynotfinalpart, thisfinalpart,  mystartsign);  for (int q = 0; q < myfollowtoto.length(); q++) {  myfollowset.add(myfollowtoto.charAt(q));  }  }  }  }  }  }  myfollowset = new ArrayList<Character>(new HashSet<Character>(myfollowset));  for (int i = 0; i < myfollowset.size(); i++) {  if (myfollowset.get(i) == '#') {  myfollowset.remove(i);  }  }  for (char t : myfollowset) {  thisfollowset += t;  }  return thisfollowset;  }  public String mySelect(List<String> mat, String onemat, List<Character> mynotfinalpart, List<Character> myfinalpart,  char mystartsign) {  String thisselectset = "";  for (char a : mynotfinalpart) {  if (onemat.charAt(2) == a) {// 如果右部第一个是非终结符号,求该符号的first  thisselectset = myFirstSet(a, mat, myfinalpart);  } else if (onemat.charAt(2) == '#') {// 如果推出空，求左部的follow  thisselectset = myFollowSet(onemat.charAt(0), mat, mynotfinalpart, myfinalpart, mystartsign);  } else {  for (char p : myfinalpart) {// 如果右部第一个是终结符号，等于中介符号  if (onemat.charAt(2) == p) {  thisselectset += p;  }  }  }  }  thisselectset = removeMethod(thisselectset);  return thisselectset;  }  public void predictionAnalysis(Stack<Character> thisinput, Map<String, String> thisselectmap, char mystartsign) {  Stack<Character> mystack = new Stack<Character>();  mystack.push('$');  mystack.push(mystartsign);  while (mystack.peek() != '$') {// 如果栈顶元素不是$dc  if (mystack.peek() == thisinput.peek()) { // 如果输入栈和匹配栈的栈顶相等则两个都抛出  mystack.pop();  thisinput.pop();  } else {  for (String t : thisselectmap.keySet()) {  for (int i = 0; i < thisselectmap.get(t).length(); i++) {  if (thisinput.peek() == thisselectmap.get(t).charAt(i)) {// 如果输入的栈顶等于某一select的元素  if (t.charAt(0) == mystack.peek()) {// 匹配栈的栈顶等于文法的左部  mystack.pop();  for (int j = t.length() - 1; j >= 2; j--) {//把产生式的右部从右到左压入  mystack.push(t.charAt(j));  if(t.charAt(j) == '#') {//如果产生式推出空，抛出匹配栈的栈顶  mystack.pop();  }  }  System.out.println(t);  }  }  }  }  }  }  }  public String removeMethod(String s) { // 去重  StringBuffer sb = new StringBuffer();  int len = s.length();  int i = 0;  boolean flag = false;  for (i = 0; i < len; i++) {  char c = s.charAt(i);  if (s.indexOf(c) != s.lastIndexOf(c)) {  flag = false;  } else {  flag = true;  }  if (i == s.indexOf(c))  flag = true;  if (flag) {  sb.append(c);  }  }  return sb.toString();  }  public static void main(String[] args) throws IOException {  ll1 myll1 = new ll1();  List<String> allline = new ArrayList<String>();// 文法  List<Character> leftpart = new ArrayList<Character>();// 文法非终结符号  List<Character> finalpart = new ArrayList<Character>();// 文法终结符号  List<String> afterRemoveDirectLeftRecursion = new ArrayList<String>();// 文法消除直接左递归  List<String> afterRemoveAllLeftRecursion = new ArrayList<String>();// 文法消除所有左递归  List<Character> afterRemoveAllLeftRecursionLeftpart = new ArrayList<Character>(); // 消除所有左递归后的文法非终结符号  List<String> afterAll = new ArrayList<String>(); // 提取左公共因子后的文法  List<Character> afterAllLeftpart = new ArrayList<Character>(); // 提取左公共因子后的文法非终结符号  Map<String, String> selectmap = new HashMap<String, String>();  char startsign;// 开始符号  // List<String> firstset = new ArrayList<String>();  String pathname = "E://t3.txt";  File filename = new File(pathname);  InputStreamReader read;  read = new InputStreamReader(new FileInputStream(filename));  BufferedReader br = new BufferedReader(read);  String oneline = "";  oneline = br.readLine();  int firstoneline = Integer.parseInt(oneline);  startsign = br.readLine().charAt(0);  // allline 读取文法  for (int i = 0; i < firstoneline; i++) {  oneline = br.readLine();  allline.add(oneline);  }  int NumberOfFinal;  NumberOfFinal = Integer.parseInt(br.readLine());  for (int i = 0; i < NumberOfFinal; i++) {  finalpart.add(br.readLine().charAt(0));  }  // 读取文法的左部  for (int i = 0; i < allline.size(); i++) {  leftpart.add(allline.get(i).charAt(0));  }  // 保存文法的非终结符号  leftpart = new ArrayList<Character>(new HashSet<Character>(leftpart));  System.out.println("文法");  for (String t : allline) {  System.out.println(t);  }  System.out.println("开始符号");  System.out.println(startsign);  System.out.println("非终结符号");  for (char t : leftpart) {  System.out.println(t);  }  System.out.println("终结符号");  for (char t : finalpart) {  System.out.println(t);  }  // 调用函数消除直接左递归  System.out.println("消除直接左递归");  afterRemoveDirectLeftRecursion = myll1.RemoveLeftRecursion(allline);  for (String t : afterRemoveDirectLeftRecursion) {  System.out.println(t);  }  System.out.println("消除所有左递归");  afterRemoveAllLeftRecursion = myll1.RemoveAllRecursion(afterRemoveDirectLeftRecursion);  for (int i = 0; i < afterRemoveAllLeftRecursion.size(); i++) {  for (int j = 0; j < afterRemoveAllLeftRecursion.size(); j++) {  if (afterRemoveAllLeftRecursion.get(i).charAt(2) == afterRemoveAllLeftRecursion.get(j).charAt(0)) {  afterRemoveAllLeftRecursion = myll1.RemoveAllRecursion(afterRemoveAllLeftRecursion);  }  }  }  for (String t : afterRemoveAllLeftRecursion) {  System.out.println(t);  }  // 存入非终结符号  for (String t : afterRemoveAllLeftRecursion) {  afterRemoveAllLeftRecursionLeftpart.add(t.charAt(0));  }  afterRemoveAllLeftRecursionLeftpart = new ArrayList<Character>(  new HashSet<Character>(afterRemoveAllLeftRecursionLeftpart));  System.out.println("提取左公共因子");  afterAll = myll1.ExtractionLeftFactor(afterRemoveAllLeftRecursion, afterRemoveAllLeftRecursionLeftpart);  afterAll = new ArrayList<String>(new HashSet<String>(afterAll));  for (String t : afterAll) {  System.out.println(t);  }  System.out.println("提取左公共因子后的文法非终结符号");  for (String t : afterAll) {  afterAllLeftpart.add(t.charAt(0));  }  afterAllLeftpart = new ArrayList<Character>(new HashSet<Character>(afterAllLeftpart));  for (char t : afterAllLeftpart) {  System.out.println(t);  }  String firstset = "";  String followset = "";  System.out.print("非终结符号");  System.out.print(" ");  System.out.print("first集");  System.out.print(" ");  System.out.print("follow集");  System.out.println();  for (char t : afterAllLeftpart) {  System.out.print(t);  firstset = myll1.myFirstSet(t, afterAll, finalpart);  System.out.print(" ");  System.out.print(firstset);  System.out.print(" ");  followset = myll1.myFollowSet(t, afterAll, afterAllLeftpart, finalpart, startsign);  System.out.print(followset);  System.out.println();  }  String selectset = "";  System.out.println();  System.out.print("产生式");  System.out.print(" ");  System.out.print("select集");  System.out.println();  for (String t : afterAll) {  selectset = myll1.mySelect(afterAll, t, afterAllLeftpart, finalpart, startsign);  selectmap.put(t, selectset);  }  for (String t : selectmap.keySet()) {  System.out.print(t);  System.out.print(" ");  System.out.print(selectmap.get(t));  System.out.println();  }  // 判断是否LL1  List<String> selectreg = new ArrayList<String>();  int isll1 = 0;  for (int i = 0; i < afterAll.size(); i++) {  for (int j = i + 1; j < afterAll.size(); j++) {  if (afterAll.get(i).charAt(0) == afterAll.get(j).charAt(0)) {// 判断同一非终结符号推出的select是否相等  selectreg.add(0, myll1.mySelect(afterAll, afterAll.get(i), afterAllLeftpart, finalpart, startsign));  selectreg.add(1, myll1.mySelect(afterAll, afterAll.get(j), afterAllLeftpart, finalpart, startsign));  if (selectreg.get(0).equals(selectreg.get(1))) {  } else {  isll1++;  }  }  }  }  System.out.println();  if (isll1 == 0) {  System.out.println("不是LL1文法");  } else {  System.out.println("是LL1文法");  }  System.out.println();  Stack<Character> myinput = new Stack<Character>();  /\*myinput.push('$');  myinput.push(')');  myinput.push('a');  myinput.push(',');  myinput.push('a');  myinput.push('(');  \*/    myinput.push('$');  myinput.push('i');  myinput.push('\*');  myinput.push('i');  myinput.push('\*');  myinput.push('i');  myinput.push('\*');  myinput.push(')');  myinput.push('i');  myinput.push('(');      /\*myinput.push('$');  myinput.push(')');  myinput.push('i');  myinput.push('(');  \*/  System.out.println("预测分析如下：");  myll1.predictionAnalysis(myinput, selectmap, startsign);  }  } | |
| 1. **实验结果：** | |
| 1. **实验总结：**   通过对语法分析程序的设计和编写，使自己获得了很大的收获，并且使自己对语法分析程序的功能有了更进一步认识。虽然在程序的设计和编写过程中出现了一些错误，但是经过同学的帮助和指导，顺利的将程序中存在的错误顺利解决，从而顺利完成了本程序的设计和编程。 | |