**实 验 报 告**

课程名称： 编译原理

实验名称： 语法分析程序设计

院 （系）： 信控学院

实验二 语法分析程序设计

一. 实验目的

理解词法分析器的任务和工作原理；掌握计算机语言语法分析程序的设计方法，并能够针对给定语言的语法规则，使用某种高级编程语言实现其语法分析器。

二. 实验内容

对于只含有+、\*运算的算术表达式，编写相应的语法分析程序，要求：

1. 用表驱动的预测分析法进行语法分析。

2. 采用某种高级程序设计语言，设计并实现语法分析程序。

3. 设计恰当的测试用例对语法分析程序进行测试。

三．实验设备

计算机、Windows 操作系统、编程语言集成开发环境。

四．实验原理（或程序框图）及步骤

1. 设计思路

首先，使用文法表示只含有+、\*运算的算术表达式语言如下：

由于该文法存在左递归，所以不符合上下文无关文法的前提，需要先对其进行消除左递归的运算，消除左递归以后得到的上下文无关文法如下：

求解文法的FIRST集以及FOLLOW集：

根据构造预测分析表的算法，构造预测分析表如下所示，在算法实现过程中，因为使用Character存储每个终结符以及非终结符，所以使用G代表E'、H代表T’。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 终结符 | 输入符号 | | | | | |
| id | + | \* | ( | ) | $ |
| E | E->TG |  |  | E->TG |  |  |
| G |  | G->+TG |  |  | G->ε | G->ε |
| T | T->FH |  |  | T->FH |  |  |
| H |  | H->ε | H->\*FH |  | H->ε | H->ε |
| F | F->i |  |  | F->(E) |  |  |

表 1 只含有+、\*运算的文法的预测分析表

1. 流程图

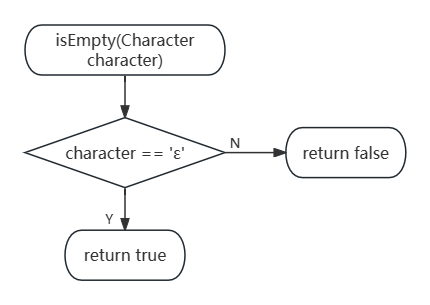
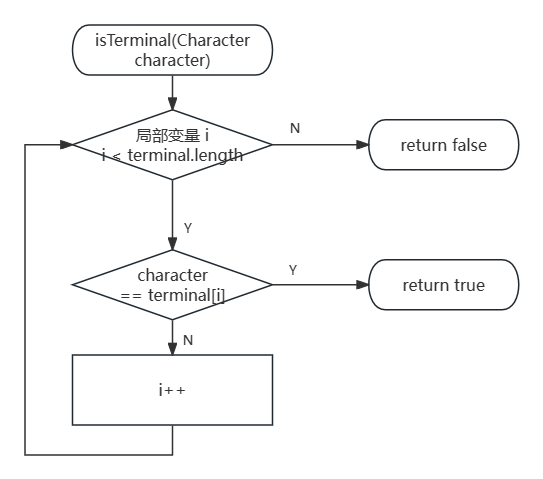


图 1 判断字符是否是终结符流程图 图 2 判断字符是否是ε

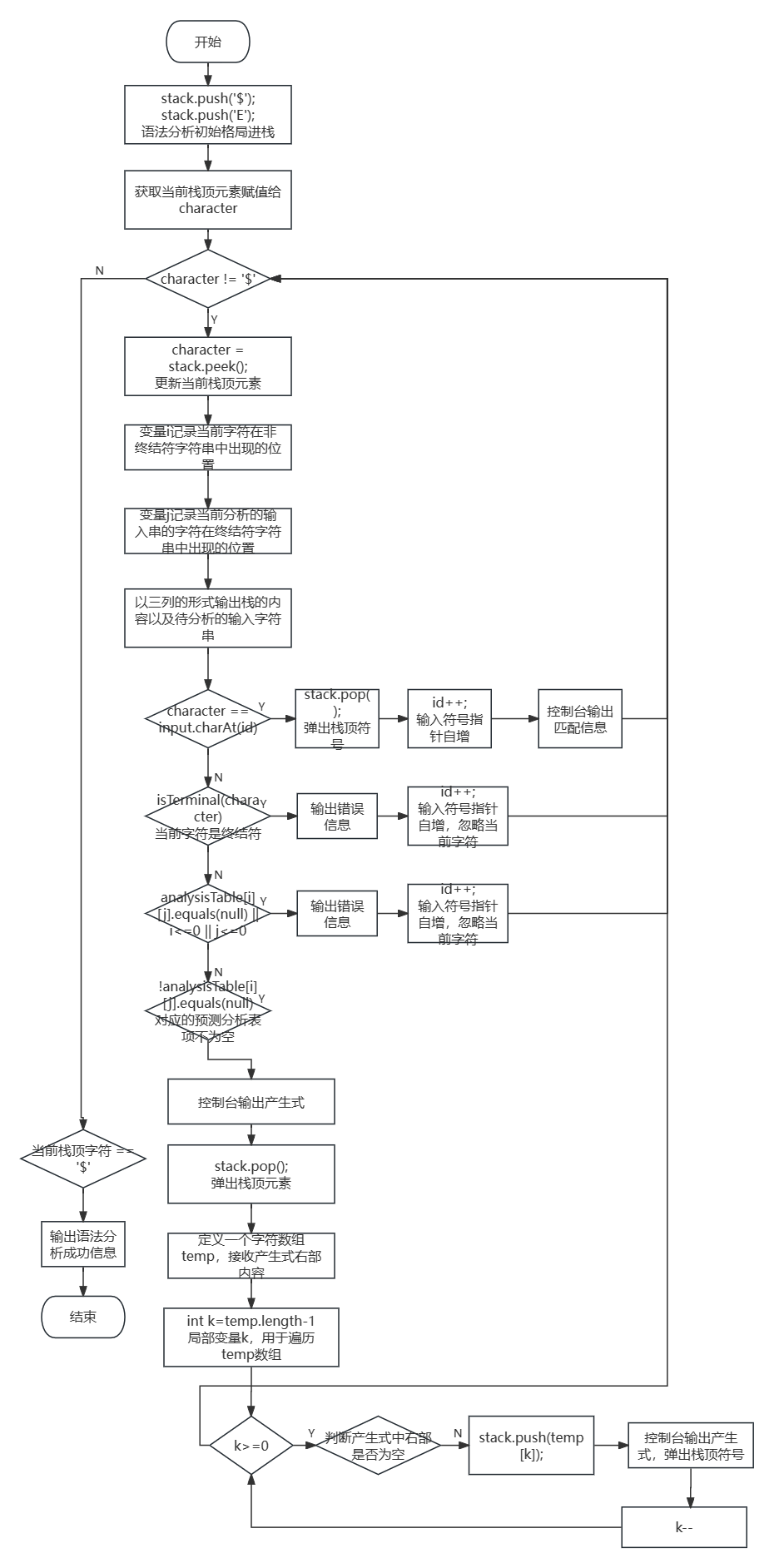


图 3 根据预测分析表进行语法分析流程图

1. 数据结构

使用字符数组terminal、nonterminal存储终结符以及非终结符号。使用字符串类型的二维数组analysisTable存储预测分析表，构造数组的时候使分析表的第一列的非终结符顺序对应于非终结符字符数组的顺序、第一行终结符顺序对应于终结符字符数组的顺序。

使用字符类型的栈stack来存放语法分析过程中的残缺的左句型，使用一个字符串input

代表输入的串。

1. 类图

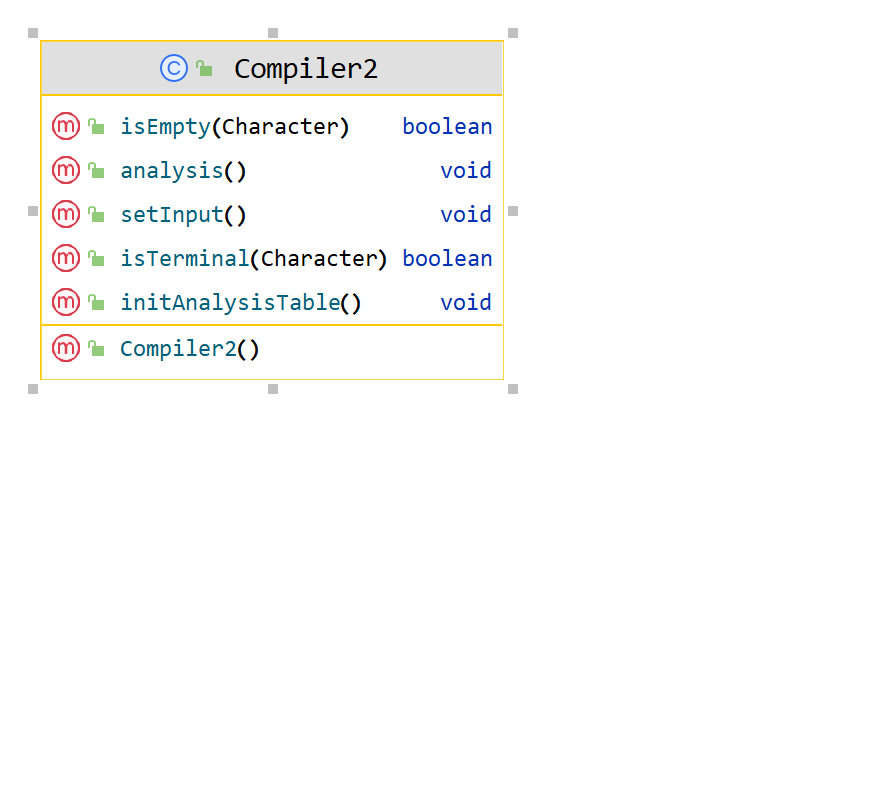


图 4 语法分析类类图

五．程序源代码

public class Compiler2 {  
 *//输入字符指针* private int id = 0;

*//判断语法分析中是否存在错误*  
 public boolean flag = true;  
 */\*\*  
 \* 终结符字符串  
 \*/* char[] terminal = new char[]{'i','+','\*','(',')','$'};  
  
 */\*\*  
 \* 非终结符字符串数组  
 \* E' => G T' => H  
 \*/* char[] nonterminal = new char[]{'E','G','T','H','F'};  
 private int row = nonterminal.length;  
 private int colomn = terminal.length;  
 String[][] analysisTable = new String[row + 1][colomn + 1];  
  
 private Stack<Character> stack = new Stack<Character>();  
  
 private String input = new String();  
  
 */\*\*  
 \* 给analysisTable初始化,防止在出现错误时因为引用了未初始化的变量而抛出异常  
 \* 根据分析表内容,给analysisTable赋初值  
 \* E' => G T' => H  
 \*/* public void initAnalysisTable(){  
 for (int i = 0; i < analysisTable.length; i++) {  
 for (int j = 0; j < analysisTable[i].length; j++) {  
 analysisTable[i][j] = "";  
 }  
 }  
 analysisTable[1][1] = "E->TG";  
 analysisTable[1][4] = "E->TG";  
 analysisTable[2][2] = "G->+TG";  
 analysisTable[2][5] = "G->ε";  
 analysisTable[2][6] = "G->ε";  
 analysisTable[3][1] = "T->FH";  
 analysisTable[3][4] = "T->FH";  
 analysisTable[4][2] = "H->ε";  
 analysisTable[4][3] = "H->\*FH";  
 analysisTable[4][5] = "H->ε";  
 analysisTable[4][6] = "H->ε";  
 analysisTable[5][1] = "F->i";  
 analysisTable[5][4] = "F->(E)";  
 }  
 public void analysis(){  
 System.*out*.println("stack\tinput\t");  
 stack.push('$');  
 stack.push('E');  
 *//查看当前栈顶元素,用变量character接收* Character character = stack.peek();  
 *//当前栈不为空,则循环*  
 while (character != '$'){  
 character = stack.peek();  
 String str1 = new String(nonterminal);  
 int i = str1.indexOf(character) + 1;  
 String str2 = new String(terminal);  
 int j = str2.indexOf(input.charAt(id)) + 1;  
  
 System.*out*.print(stack.toString()+"\t\t");  
 *//只输出待分析的字符串* for (int index = id; index < input.length(); index++) {  
 System.*out*.print(input.charAt(index));  
 }  
 System.*out*.print("\t\t");  
  
 if(character == input.charAt(id)) {  
 stack.pop();  
 id++;  
 System.*out*.println("匹配");  
 } else if (i <= 0 || j <= 0) {  
 *//如果当前M[X,a]空白,说明M[X,a]是一个报错条目* System.*out*.println("错误,忽略当前字符");  
 *//抛弃一个输入记号,直到能够匹配为止* id++;  
 flag = false;  
 } else if (isTerminal(character)) {  
 *//如果当前是终结符,但是没有匹配,表示语法分析错误* System.*out*.println("错误,忽略当前字符");  
 id++;  
 flag = false;  
 } else if (analysisTable[i][j].length() == 0) {  
 *//如果当前M[X,a]空白,说明M[X,a]是一个报错条目* System.*out*.println("错误,忽略当前字符");  
 *//抛弃一个输入记号,直到能够匹配为止* id++;  
 flag = false;  
 } else if(analysisTable[i][j].length() != 0){  
 System.*out*.println(analysisTable[i][j]);  
 stack.pop();  
 char[] temp = analysisTable[i][j].substring(3).toCharArray();  
 *//把字符数组内容逆序入栈* for (int k = temp.length - 1; k >= 0; k--) {  
 *//如果不是 -> ε 的产生式,那么将产生式的字符加入到分析栈中* if(!isEmpty(temp[k])){  
 stack.push(temp[k]);  
 }  
 }  
 }  
 }  
 if(character == '$' && flag){  
 System.*out*.println("接受,语法分析结束");  
 }  
 }  
 */\*\*  
 \* 判断某个字符是不是终结符  
 \*/* public boolean isTerminal(Character character){  
 for (int i = 0; i < terminal.length; i++) {  
 if(character == terminal[i]) {  
 return true;  
 }  
 }  
 return false;  
 }  
 public void setInput(){  
 System.*out*.println("请输入待分析的字符串:");  
 Scanner sc = new Scanner(System.*in*);  
 input = sc.next();  
 }  
 public boolean isEmpty(Character character){  
 return character == 'ε';  
 }  
}

1. 实验数据、结果分析
2. 测试用例及运行结果截图

正确输入串的测试：在命令行窗口中输入一个正确的算术表达式 (i+i\*i)$

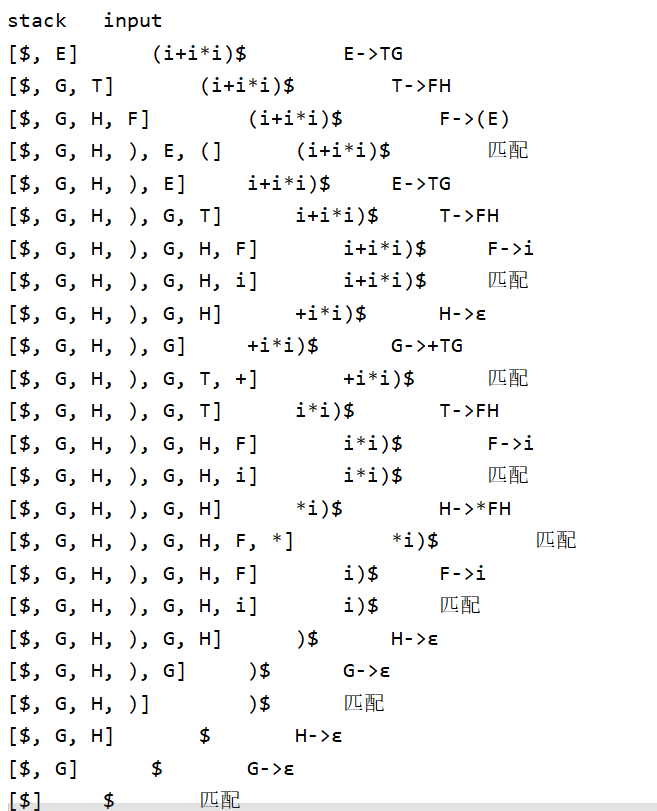


图 5 对正确输入串的测试结果

错误输入串的测试:在命令行窗口中输入一个不符合语法规则的算术表达式 i(i$，产生了一个异常语法分析程序不能对剩余的字符串进行语法分析。



图 6 对错误输入串的测试结果

1. 错误原因分析以及改进

通过断点调试可以发现，出现异常错误的原因是尝试调用未初始化的对象的长度值。检查代码发现，存储分析表的字符串数组只给具有产生式的表项进行了初始化，对于没有产生式的表项即错误情况没有初始化，当使用这些对象的时候，就会抛出空引用异常。

解决方法是在初始化分析表的字符串数组的方法中加入对于没有产生式的表项的初始化，给其赋值为””，然后在代码中通过对字符串数组元素的长度判断来判断其中有无内容。改进后输入不符合语法规则的算术表达式的运行结果如图：

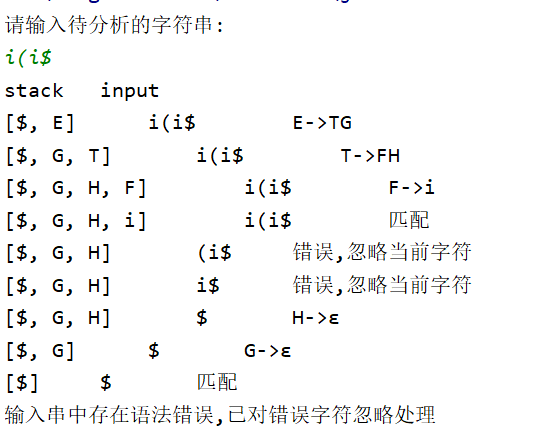


图 7 改进程序以后对错误输入串的测试结果

七. 实验小结

通过本次实验复习了如何使用上下文无关文法将语法规则描述出来。因为要采用自顶向下语法分析方法，所以需要考虑文法中是否存在左递归、是否需要提取左因子等问题。并对文法进行了相应的改写，设计好文法以后，求出产生式的First集合以及非终结符的Follow集合，然后再根据算法构建预测分析表。

在程序设计过程中，主要实现了根据语法分析表以及栈来分析输入串是否符合只含有+、\*运算的算术表达式的语法规则，并将语法分析的每个步骤以及栈和输入串格局的变化输出到控制台上。程序能够对符合语法规则的串进行语法分析，同时也能对不符合语法规则的串进行简单的错误处理。但是，该程序还存在一定的局限性，因为预测分析表是提前计算并写入到二维数组中的，所以本程序只能对一种语法规则进行语法分析，在后续的迭代中，应该考虑根据将消除左递归、消除左公因子；求First集合、求Follow集合；构造预测分析表等算法进行实现，最后能够实现对于任意文法都能自动构建预测分析表并对输入串进行语法分析。