**江苏科技大学**

**实 验 报 告**

课 程： 编译原理

课 题： 实验2·LL(1)预测分析

学 院： 计算机学院

学 号： 182210710119

姓 名： 陈四贵

班 级： 1822107101

指导老师： 肖 寒

# 一、实验目的

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；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，训练掌握开发应用程序的基本方法。提高专业素质，为培养适应社会多方面需要的能力。

# 二、设备与环境

windows 10环境、visual studio 2017

# 三、实验要求

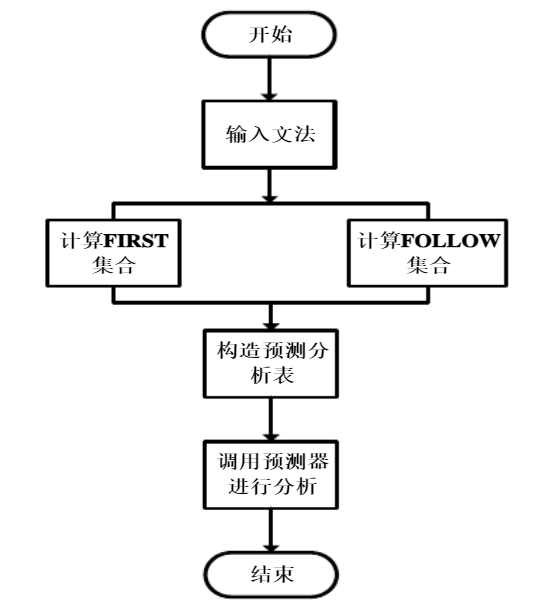
1）根据某一文法编制调试 LL（1）分析程序，以便对任意输入的符号串进行分析。

2）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。

3）分析法的功能是利用LL（1）控制程序根据显示栈顶内容、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。

# 四、实验原理

1、程序功能



2、算法简介

LL（1）文法是一种自上而下的分析，使用最左推导，从左至右扫描输入串，且对每次最左推导只需向前看一个输入符号，便可确定当前所应当选择的算法规则，分析器结构：

1）程序中写入某个文法的所有产生式；

2）分别提取终结符VT与非终结符VN（即扫描产生式）；

3）构造所有非终结符的First集，由程序自动构造LL（1）分析表 ；

4）预测分析表：

遍历每一个产生式；

如果右部的第一个字符tmp是终结符且不是空串，更新预测分析表；

如果右部的第一个字符是空串，遍历左部的Follow集更新预测分析表；

如果右部的第一个字符是非终结符，遍历它的First集更新预测分析表；

5）从键盘读入要识别的符号串；

6）由程序根据分析表进行预测分析符号串：

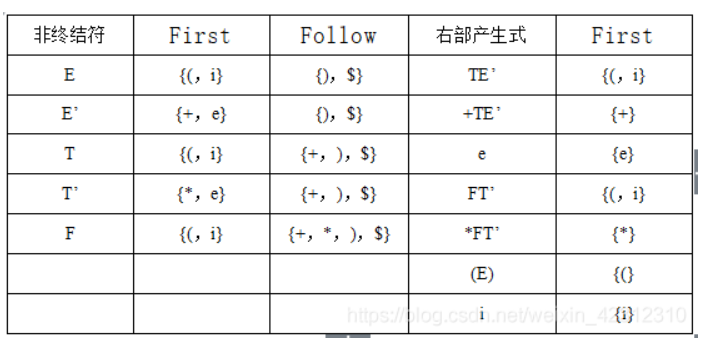
规定f1为符号栈栈顶字符，f2为剩余输入串的第一个字符;

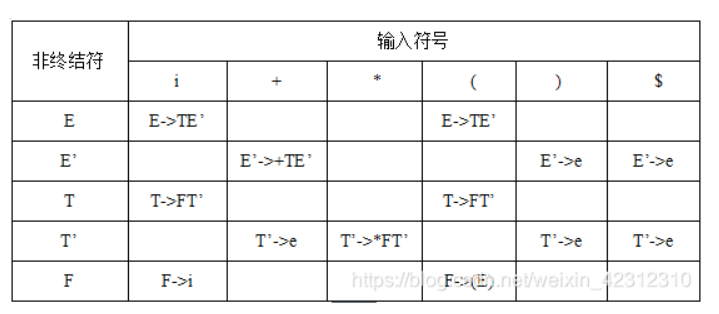
若f1=f2=“#”,则分析成功,停止分析Ø 若f1=f2≠”#”,则把f1从栈顶弹出,让f2指向下一个输入符号;

若f1是一个非终结符,则查看分析表,若table[f1][f2]中有值，则f1出栈，并且产生式的右部反序进栈;

再把产生式的右部符号推进栈的同时应做这个产生式相应得语义动作,若M[A,a]中存放着”出错标志”,则调用出错诊察程序error;

7）得出接受符号串的动作结果并打印。

3、文法中非终结符的First与Follow集及右部产生式的First集

4、LL(1)文法分析表

# 五、程序测试与调试

1、测试用例

输入：i+i\*i

2、运行结果参考

# 六、程序源码

1. #include<iostream>
2. #include<string>
3. #include<map>
4. #include<vector>
5. #include<stack>
6. #include<set>
7. #include<cstring>
8. #include <cstdlib>
9. #include <algorithm>
10. **using** **namespace** std;
11. **void** input\_G();  //LL(1)文法输入函数以及VN和VT之间的分离
12. **void** Select();
13. **bool** Judge\_LL1();
14. **void** Predict();
15. **void** Analyze(string str);
16. string char2str(**char** ch);
17. string p[100];       //表示产生式
18. **int** n\_chan;             //产生式的个数
19. **int** n\_VN;
20. vector<string> VN;      //非终结符集,和set\_VN存储相同，但向量存储便于随机访问
21. vector<string> VT;      //终结符集
22. set<string> set\_VN;      //非终结符集合,和VN存储相同，但集合存储便于查找元素和去重
23. set<string> set\_VT;     //终结符集合
24. string all\_left[100];      //产生式所有左部VN
25. string all\_right[100];      //产生式所有右部VN+VN
26. string S;  //开始符号
27. //map 一对一的key，value对,通过迭代器来实现
28. map<string, **int**> First;    //VN映射到FIRST集合下标
29. map<string, **int**> Follow;    //VN映射到FOLLOW集合下标
30. set<string> SELECT[100];
31. vector<string> predict[100];  //预测分析表
32. string kong = "error";  //预测分析表空白处填写字符串
33. **int** step = 0;  //预测分析过程步骤序数
34. **int** main() {
35. input\_G();
36. Select();
37. Predict();
38. cout << "请输入字符串(输入'-'结束)(如i+i):";
39. string str;
40. cin >> str;
41. **while** (str[str.length() - 1] != '-') {
42. Analyze(str);
43. cout << endl << "请继续输入字符串(输入'-'结束)(如i+i):";
44. cin >> str;
45. }
46. **return** 0;
47. }
48. ///输入合法文法&&分离VT和VN
49. **void** input\_G() {
50. string tmp;   //用于拼接字符串
51. string tmp1;
52. n\_chan = 8;
53. p[0] = "E->TE'";
54. p[1] = "E'->+TE'";
55. p[2] = "E'->e";
56. p[3] = "T->FT'";
57. p[4] = "T'->\*FT'";
58. p[5] = "T'->e";
59. p[6] = "F->(E)";
60. p[7] = "F->i";
61. ///提取非终结符VN的循环
62. **for** (**int** i = 0; i < n\_chan; i++) {
63. tmp = "";
64. tmp1 = "";
65. **int** j = 0;
66. **while** (p[i][j] != '-') {
67. tmp += p[i][j];  //因为非终结符可能是形如E'之类的两个字符，所以需要拼接，遇到箭头就结束
68. j++;
69. }
70. j += 2;
71. **for** (; j < (**int**)p[i].length(); j++)
72. tmp1 += p[i][j];
73. cout << p[i] << endl;
74. all\_left[i] = tmp;      //存储每个产生式的左部
75. all\_right[i] = tmp1;    //存储每个产生式的右部
76. set\_VN.insert(tmp);  //将每行产生式拼接好的非终结符插入到集合中
77. }
78. n\_VN = set\_VN.size();   //读取VN个数
79. ///提取终结符VT的循环
80. **for** (**int** i = 0; i < n\_chan; i++) {      //n\_chan是产生式的个数
81. **int** chan\_len = p[i].length(); //每个产生式的长度
82. **for** (**int** j = 0; j < chan\_len; j++)
83. **if** (p[i][j] == '>') {
84. tmp = "";
85. **for** (**int** k = j + 1; k < chan\_len; k++) { //k记录"->"之后的下标
86. tmp1 = p[i][k];
87. **if** (set\_VN.count(tmp1) == 0) { //count==0表示该字符不在集合中，即∈VT
88. tmp += p[i][k];           //拼接字符串
89. tmp1 = p[i][k + 1];
90. **if** ((k == chan\_len - 1) || ((k < chan\_len - 1) && set\_VN.count(tmp1) == 1)) {
91. //判断(已是最后一个字符||后一个字符是VN则之前的字符串是一个VT)
92. set\_VT.insert(tmp);
93. tmp = "";
94. }
95. }
96. }
97. **break**;  //本行产生式已扫描结束
98. }
99. }
100. set\_VT.erase("'");  //VT集合中删除"'"和"e"符号
101. //通过集合迭代器，将其中元素转移到向量VN中(因为集合不易随机存取，但可以筛除重复元素)
102. **for** (set<string>::iterator it = set\_VN.begin(); it != set\_VN.end(); it++)
103. VN.push\_back(\*it);
104. **for** (**int** i = 0; i < (**int**)VN.size(); i++) {
105. First[VN[i]] = i;  ///每个VN映射到FIRST集合下标
106. }
107. **for** (set<string>::iterator it = set\_VT.begin(); it != set\_VT.end(); it++)
108. VT.push\_back(\*it);
109. **for** (**int** i = 0; i < (**int**)VT.size(); i++) {
110. **if** (VT[i] == "e")
111. Follow["#"] = i;
112. **else**
113. Follow[VT[i]] = i;
114. }
115. }
116. ///SELECT集
117. **void** Select() {
118. SELECT[0].insert("(");
119. SELECT[0].insert("i");
120. SELECT[1].insert("+");
121. SELECT[2].insert(")");
122. SELECT[2].insert("#");
123. SELECT[3].insert("(");
124. SELECT[3].insert("i");
125. SELECT[4].insert("\*");
126. SELECT[5].insert("+");
127. SELECT[5].insert(")");
128. SELECT[5].insert("#");
129. SELECT[6].insert("(");
130. SELECT[7].insert("i");
131. }
132. ///预测分析表
133. **void** Predict() {
134. **bool** flag;
135. **for** (**int** i = 0; i < (**int**)VT.size(); i++)
136. **if** (VT[i] == "e") {
137. VT[i] = "#";
138. **break**;
139. }
140. **for** (**int** i = 0; i < n\_VN; i++) {
141. **for** (**int** j = 0; j < (**int**)VT.size(); j++) {
142. flag = **true**;
143. **for** (**int** k = 0; k < n\_chan; k++)
144. **if** (all\_left[k] == VN[i] && SELECT[k].count(VT[j]) == 1) {
145. predict[i].push\_back(all\_right[k]);
146. flag = **false**;
147. **break**;
148. }
149. **if** (flag) predict[i].push\_back(kong);
150. }
151. }
152. }
153. ///分析输入字符串
154. **void** Analyze(string str) {
155. S = VN[0];  //将VN集中首元素定为开始符号元素
156. vector<string> analy;  //分析向量(用向量代替可以随机访问，便于显示)
157. vector<string> input;    //输入串向量(用向量代替可以随机访问，便于显示)
158. string fun;      //存储在预测分析表中找到的表达式
159. analy.push\_back("#");
160. analy.push\_back(S);
161. input.push\_back("#");
162. **for** (**int** i = (**int**)str.length() - 1; i >= 0; i--)
163. input.push\_back(char2str(str[i]));
164. ///\*\*\*\*\*\*\*\*\*\*\*\*\*\*初始化分割线\*\*\*\*\*\*\*\*\*\*\*\*\*\*
165. cout << endl << "步骤\t分析栈\t\t剩余输入串\t推导所用产生式" << endl;
166. **while** (**true**) {
167. cout << ++step << "\t";
168. **for** (**int** i = 0; i < (**int**)analy.size(); i++) {
169. cout << analy[i];
170. }
171. cout << "\t\t";
172. **for** (**int** i = (**int**)input.size() - 1; i >= 0; i--) {
173. cout << input[i];
174. }
175. **if** (input.back() == "#"&&analy.back() == "#") {  //向量尾元素,模拟栈顶元素
176. cout << "\t\t接受";
177. **break**;
178. }
179. **else** **if** (input.back() == analy.back()) {   //两个栈顶元素同是一个VT则均出栈一个元素
180. cout << "\t\t\"" << input.back() << "\"匹配";
181. analy.pop\_back();
182. input.pop\_back();
183. }//两个都是VT但不相等，则出错
184. **else** **if** (set\_VT.count(input.back()) == 1 && set\_VT.count(analy.back()) == 1 && input.back() != analy.back()) {
185. cout << "出错:两栈顶算符不匹配！" << endl;
186. **break**;
187. }//分析栈顶元素是VN
188. **else** **if** (set\_VN.count(analy.back()) == 1) {
189. fun = predict[First[analy.back()]][Follow[input.back()]];
190. **if** (fun != kong) { //能在预测分析表中找到表达式
191. cout << "\t\t" << analy.back() << "->" << fun;
192. analy.pop\_back(); //模拟出栈
193. **for** (**int** i = (**int**)fun.length() - 1; i >= 0; i--) {   //倒序入栈
194. **if** (fun[i] == '\'') {
195. analy.push\_back(char2str(fun[i - 1]) + char2str(fun[i]));
196. i--;
197. }
198. **else** **if** (fun[i] == 'e') **break**;
199. **else**
200. analy.push\_back(char2str(fun[i]));
201. }
202. }
203. **else** {
204. cout << endl << "出错:未在预测表中找到产生式！" << endl;
205. **break**;
206. }
207. }//两个都是VT但不在终结符集中
208. **else** **if** (set\_VT.count(input.back()) == 0) {
209. cout << endl << "出错:该字符不属于该文法！" << endl;
210. **break**;
211. }
212. cout << endl;
213. }
214. }
215. ///字符转换成字符串函数
216. string char2str(**char** ch) {
217. **char** t[] = { ch,'\0' };
218. **return** t;
219. }