**编译原理与技术程序设计三**

**——LR分析程序的设计与实现**

**实**

**验**

**报**

**告**

**班 级 2016211310**

**姓 名 张绍磊**

**学 号 2016211392**

# 一、实验目的

让同学们更加深刻理解LR文法在自底向上分析程序中的具体应用。

# 二、实验内容

编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生。

E->E+T|E-T|T

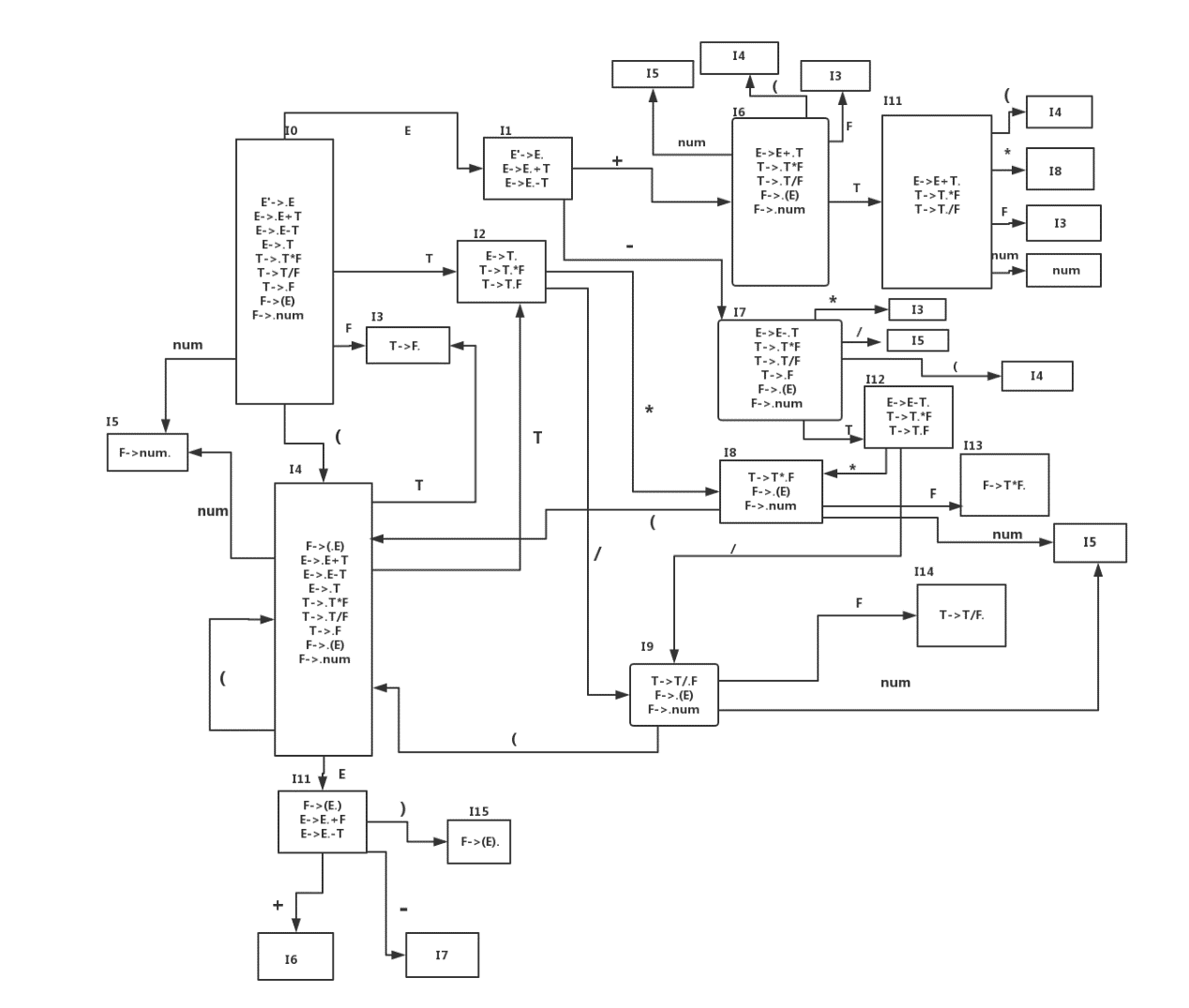
T->T\*F|T/F|F

F->id|(E)|num

1. 构造识别该文法所有活前缀的DFA
2. 构造该文法的SLR分析表
3. 要求编程实现算法4.3，构造SLR分析程序。

# 三、实验步骤

## 1. 该文法所有活前缀的DFA



## 2. 构造该文法的SLR分析表

所有的产生式如下所示：

1：E->E+T

2：E->E-T

3：E->T

4：T->T\*F

5：T->T/F

6：T->F

7：F->(E)

8：F->num

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | action | | | | | | | | goto | | |
| 状态 | （ | ） | + | - | \* | / | num | $ | E | T | F |
| 0 | S4 |  |  |  |  |  | S5 |  | 1 | 2 | 3 |
| 1 |  |  | S6 | S7 |  |  |  | ACC |  |  |  |
| 2 |  | R3 | R3 | R3 | S8 | S9 |  | R3 |  |  |  |
| 3 |  | R6 | R6 | R6 | R6 | R6 |  | R6 |  |  |  |
| 4 | S4 |  |  |  |  |  | S5 |  | 10 | 2 | 3 |
| 5 |  | R8 | R8 | R8 | R8 | R8 |  | R8 |  |  |  |
| 6 | S4 |  |  |  |  |  | S5 |  |  | 11 | 3 |
| 7 | S4 |  |  |  |  |  | S5 |  |  | 12 | 3 |
| 8 | S4 |  |  |  |  |  | S5 |  |  |  | 13 |
| 9 | S4 |  |  |  |  |  | S5 |  |  |  | 14 |
| 10 |  | S15 | S6 | S7 |  |  |  |  |  |  |  |
| 11 |  | R1 | R1 | R1 | S8 | S9 |  | R1 |  |  |  |
| 12 |  | R2 | R2 | R2 | S8 | S9 |  | R2 |  |  |  |
| 13 |  | R4 | R4 | R4 | R4 | R4 |  | R4 |  |  |  |
| 14 |  | R5 | R5 | R5 | R5 | R5 |  | R5 |  |  |  |
| 15 |  | R7 | R7 | R7 | R7 | R7 |  | R7 |  |  |  |

## 3. SLR分析程序

（1）生成式的数据结构

struct createRule

{

string rule;//产生式本身

string leftsymbol;//生成式的左侧符号

int length;//产生式右部的字符串的长度

};

typedef createRule mycreateRule;

(2)符号表

string endsymbol[8] = { "(",")","+","-","\*","/","num","$" };//终结符符号表

(3)预测分析表

string actionAnalyseMap[16][11] =

{

{ "S4","error","error","error","error","error","S5","error","1","2","3" },

{ "error","error","S6","S7","error","error","error","ACC","error","error","error" },

{ "error","R3","R3","R3","S8","S9","error","R3","error","error","error" },

{ "error","R6","R6","R6","R6","R6","error","R6","error","error","error" },

{ "S4","error","error","error","error","error","S5","error","10","2","3" },

{ "error","R8","R8","R8","R8","R8","error","R8","error","error","error" },

{ "S4","error","error","error","error","error","S5","error","error","11","3" },

{ "S4","error","error","error","error","error","S5","error","error","12","3" },

{ "S4","error","error","error","error","error","S5","error","error","error","13" },

{ "S4","error","error","error","error","error","S5","error","error","error","14" },

{ "error","S15","S6","S7","error","error","error","error","error","error","error" },

{ "error","R1","R1","R1","S8","S9","error","R1","error","error","error", },

{ "error","R2","R2","R2","S8","S9","error","R2","error","error","error" },

{ "error","R4","R4","R4","R4","R4","error","R4","error","error","error" },

{ "error","R5","R5","R5","R5","R5","error","R5","error","error","error" },

{ "error","R7","R7","R7","R7","R7","error","R7","error","error","error" },

};

（4）数据成员

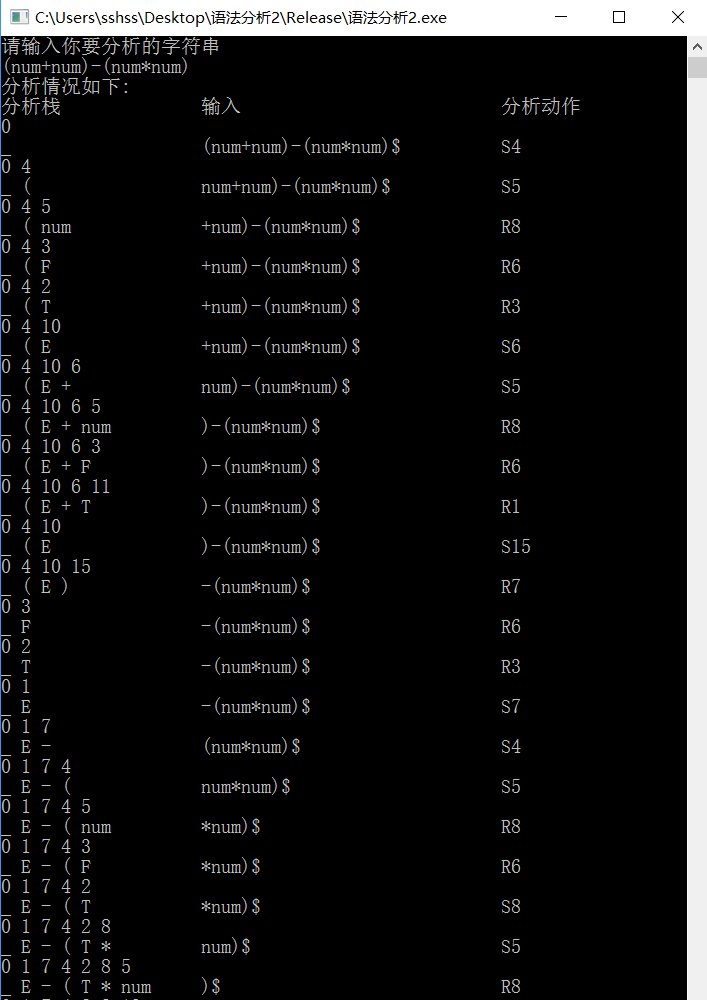
|  |  |  |
| --- | --- | --- |
| 序号 | 数据成员 | 说明 |
| 01 | vector<string> stateStack | 状态栈 |
| 02 | vector<string> symbolStack | 符号栈 |
| 03 | string input | 输入分析串 |

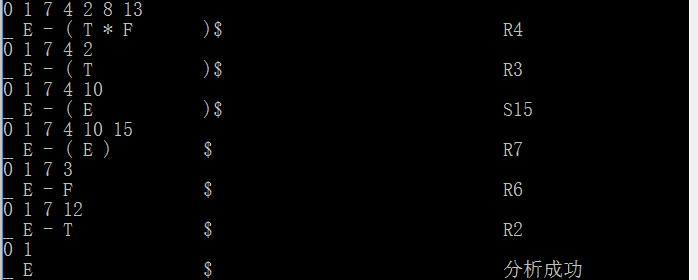
（5）子函数模块设计

|  |  |  |
| --- | --- | --- |
| 序号 | 子函数名称 | 说明 |
| 01 | int findendsymbol(string a) | 找出终结符在符号表的位置 |
| 02 | void error() | 错误分析 |

# 四、实验结果

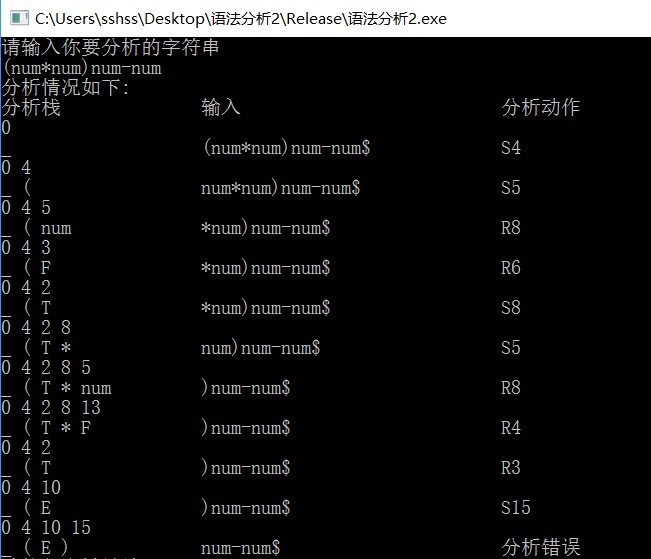
1. 样例1：（num+num）-(num\*num)





分析成功，该句子属于该LR文法。

2. 样例2：(num\*num)num-num



出错，无状态15遇到num时的分析动作。

# 五、总结与体会

通过本次对语法分析程序的设计和编写，自己获得了很大的收获，语法分析程序的功能有了更进一步认识，也更加理解了LR分析方法的精髓，加深了对课本知识的了解与应用。

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，顺利地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写，获益匪浅。

# 六、实验源代码

1. #include<iostream>
2. #include<string>
3. #include<vector>
4. #include <iomanip>
5. **using** **namespace** std;
7. //产生式结构
8. **struct** createRule
9. {
10. string rule;//产生式本身
11. string leftsymbol;//生成式的左侧符号
12. **int** length;//产生式右部的字符串的长度
13. };
14. **typedef** createRule mycreateRule;
16. //生成式的数组
17. mycreateRule nowCreateRule[8] =
18. {
19. { { "E->E+T" },{ "E" },{ 3 } },
20. { { "E->E-T" },{ "E" },{ 3 } },
21. { { "E->T"   },{ "E" },{ 1 } },
22. { { "T->T\*F" },{ "T" },{ 3 } },
23. { { "T->T/F" },{ "T" },{ 3 } },
24. { { "T->F"   },{ "T" },{ 1 } },
25. { { "F->(E)" },{ "F" },{ 3 } },
26. { { "F->num" },{ "F" },{ 1 } },
27. };
28. string endsymbol[8] = { "(",")","+","-","\*","/","num","$" };//终结符符号表
30. //分析表
31. string actionAnalyseMap[16][11] =
32. {
33. { "S4","error","error","error","error","error","S5","error","1","2","3" },
34. { "error","error","S6","S7","error","error","error","ACC","error","error","error" },
35. { "error","R3","R3","R3","S8","S9","error","R3","error","error","error" },
36. { "error","R6","R6","R6","R6","R6","error","R6","error","error","error" },
37. { "S4","error","error","error","error","error","S5","error","10","2","3" },
38. { "error","R8","R8","R8","R8","R8","error","R8","error","error","error" },
39. { "S4","error","error","error","error","error","S5","error","error","11","3" },
40. { "S4","error","error","error","error","error","S5","error","error","12","3" },
41. { "S4","error","error","error","error","error","S5","error","error","error","13" },
42. { "S4","error","error","error","error","error","S5","error","error","error","14" },
43. { "error","S15","S6","S7","error","error","error","error","error","error","error" },
44. { "error","R1","R1","R1","S8","S9","error","R1","error","error","error", },
45. { "error","R2","R2","R2","S8","S9","error","R2","error","error","error" },
46. { "error","R4","R4","R4","R4","R4","error","R4","error","error","error" },
47. { "error","R5","R5","R5","R5","R5","error","R5","error","error","error" },
48. { "error","R7","R7","R7","R7","R7","error","R7","error","error","error" },
49. };
51. //错误处理程序
52. **void** error()
53. {
54. cout << "分析错误" << endl;
55. }
57. //找出终结符号在哪个位置
58. **int** findendsymbol(string a)
59. {
60. **int** i;
61. **for** (i = 0; i < 8; i++)
62. {
63. **if** (a == endsymbol[i])
64. {
65. **return** i;
66. }
67. }
68. **return** -1;
69. }

72. **void** main()
73. {
74. vector<string> stateStack;//状态栈
75. vector<string> symbolStack;//符号栈
76. stateStack.push\_back("0");//状态栈的初始状态
77. symbolStack.push\_back("\_");//符号栈的初始状态
78. string input;//分析串
79. cout << "请输入你要分析的字符串" << endl;
80. cin >> input;
81. input += "$";
82. **int** ip = 0;
83. string a;
84. string s;
85. cout << "分析情况如下:" << endl;
86. cout.setf(ios::left);//设置左对齐
87. cout << setw(20) << "分析栈" << setw(30) << "输入" << setw(20) << "分析动作" << endl;
88. **do**
89. {
90. string string\_content = "";
91. **for** (**int** k = 0; k < stateStack.size(); k++)//输出状态栈内容
92. string\_content += stateStack[k]+" ";
93. cout << setw(20) << string\_content << endl;
95. string\_content = "";
96. **for** (**int** k = 0; k < symbolStack.size(); k++)//输出符号栈内容
97. string\_content += symbolStack[k]+" ";
98. cout << setw(20) << string\_content;
99. cout << setw(30) << input.substr(ip);//取子串输出分析符号
100. s = stateStack.back();
101. a = input.at(ip);
102. **if** (a == "n")//处理“num”
103. {
104. a = "num";
105. ip = ip + 2;
106. }
107. **int** index=findendsymbol(a);
108. **if** (actionAnalyseMap[stoi(stateStack.back())][index].at(0) == 'S')//遇到分析动作是移进
109. {
111. string i =actionAnalyseMap[stoi(stateStack.back())][index].substr(1) ;
112. cout << setw(20) << actionAnalyseMap[stoi(stateStack.back())][index] << endl;
113. symbolStack.push\_back(a);
114. stateStack.push\_back(i);
115. ip = ip + 1;
116. }
117. **else** **if** (actionAnalyseMap[stoi(stateStack.back())][index].at(0) == 'R')//遇到分析动作是规约
118. {
119. **int** i = stoi(actionAnalyseMap[stoi(stateStack.back())][index].substr(1));
120. cout << setw(20) << actionAnalyseMap[stoi(stateStack.back())][index] << endl;
121. **for** (**int** j = 0; j < nowCreateRule[i - 1].length; j++)//出栈当前生成式左边符号长度个符号
122. {
123. symbolStack.pop\_back();
124. stateStack.pop\_back();
125. }
126. string s1;
127. s1 = stateStack.back();
129. symbolStack.push\_back(nowCreateRule[i - 1].leftsymbol);
131. **int** s2;
132. **if** (nowCreateRule[i - 1].leftsymbol == "E")//遇到“E”时转移的状态
133. {
134. s2 = 8;
135. }
136. **else** **if** (nowCreateRule[i - 1].leftsymbol == "T")//遇到“T”时转移的状态
137. {
138. s2 = 9;
139. }
140. **else** **if** (nowCreateRule[i - 1].leftsymbol == "F")//遇到“F”时转移的状态
141. {
142. s2 = 10;
143. }
145. **if** (actionAnalyseMap[stoi(s1)][s2] != "error")//若不是错误，则进栈
146. {
147. stateStack.push\_back(actionAnalyseMap[stoi(s1)][s2]);
149. }
150. **else**
151. {
152. error();
153. **break**;
154. }
155. }
156. **else** **if** (actionAnalyseMap[stoi(stateStack.back())][index] == "ACC")
157. {
158. cout << "分析成功" << endl;
159. **break**;
160. }
161. **else**
162. {
163. error();
164. **break**;
165. }
167. } **while** (1);
168. system("pause");
169. }