**编译原理与技术程序设计**

**——语义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**

目录

[一、实验目的 3](#_Toc534026590)

[二、实验内容 3](#_Toc534026591)

[三、实验分析 3](#_Toc534026592)

[四、实验步骤 5](#_Toc534026593)

[1. 翻译方案 5](#_Toc534026594)

[2. 构造识别所有活前缀的DFA 5](#_Toc534026595)

[3. 构造LR分析表 6](#_Toc534026596)

[4. 扩充分析栈 7](#_Toc534026597)

[5. 改造分析程序 8](#_Toc534026598)

[五、实验结果 8](#_Toc534026599)

[六、总结与体会 13](#_Toc534026600)

[七、实验源代码 13](#_Toc534026601)

# 一、实验目的

让同学们更加深刻理解语义分析在自底向上分析程序中的具体应用。

# 二、实验内容

**题目**：语义分析程序的设计与实现。

**实验内容**：编写语义分析程序，实现对算术表达式的类型检查和求值。要求所分析算术表达式由如下的文法产生。



**实验要求**：

用自底向上的语法制导翻译技术实现对表达式的分析和翻译。

1. 写出满足要求的语法制导定义或翻译方案。
2. 编写分析程序，实现对表达式的类型进行检查和求值，并输出：
3. 分析过程中所有产生式。
4. 识别出的表达式的类型。
5. 识别出的表达式的值。
6. 实验方法：可以选用以下两种方法之一。
7. 自己编写分析程序。
8. 利用YACC自动生成工具。

# 三、实验分析

由于要求进行类型检查和求值，所以可以**定义两个综合属性**，一个记录值一个记录类型，存放在结构中，一并传入传出。

**输出的产生式可以作为虚拟综合属性**，在产生式的最后打印出来。

**id认为是定义的变量名**，假设是26个小写字母，它们的值存于一个数组里。

将类型检查和求值归于**一次扫描**，当检查类型出错时则停止，否则继续。

**哈希**实现输入的映射，模拟词法分析的记号流。

输入格式为每个num和id对应两个输入字符，其他运算符仍对应一个字符。比如第4个num,输入为num4。

由于只具有综合属性，故可以用**S属性的自底向上翻译实现**，**利用LR分析程序来实现**，只需扩充分析站和改造分析程序。

# 四、实验步骤

## 1. 翻译方案



## 2. 构造识别所有活前缀的DFA

**构造扩展文法**

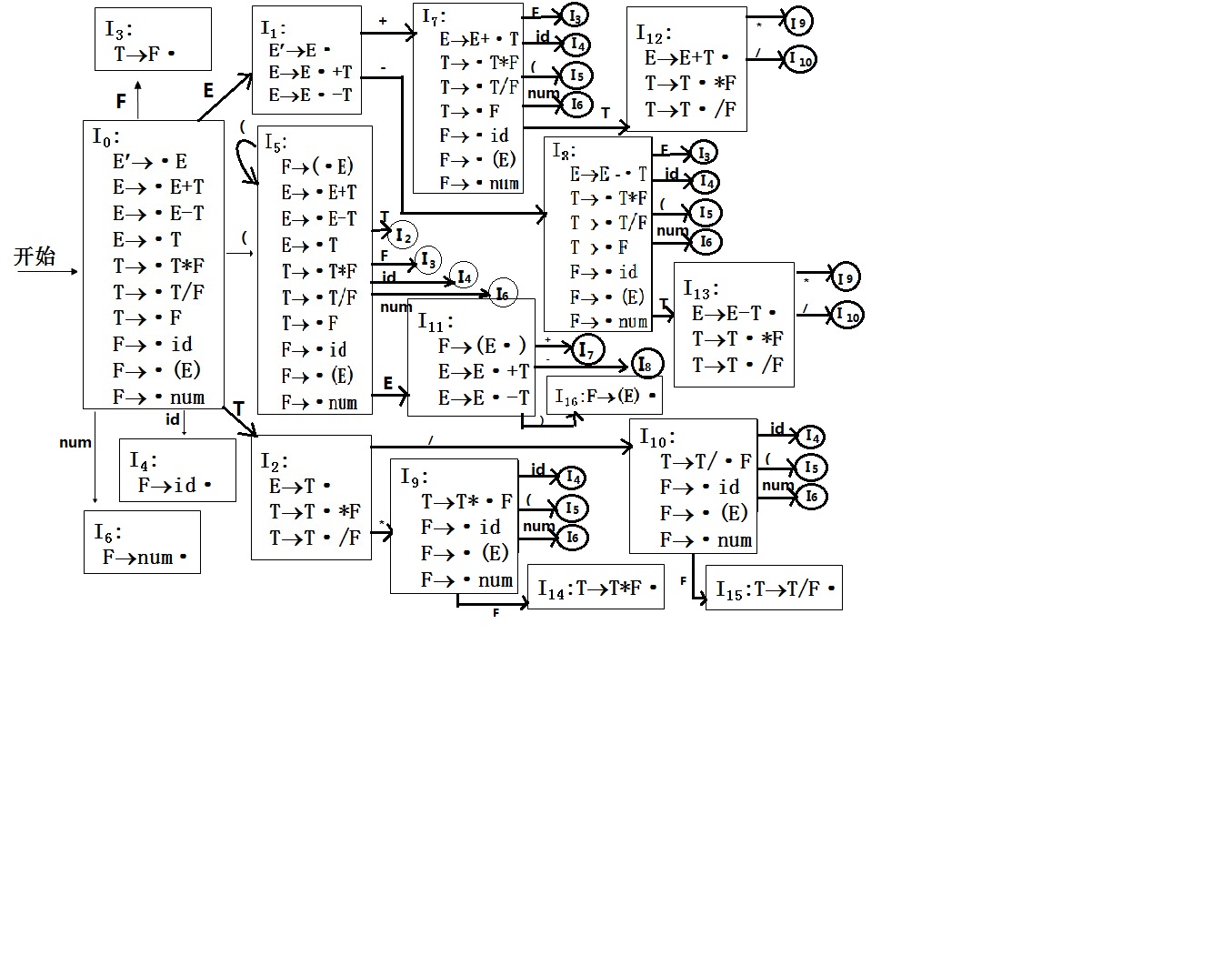




**FIRST和FOLLOW集**如下

|  |  |  |  |
| --- | --- | --- | --- |
|  | **E** | **T** | **F** |
| **FIRST** | id, (, num | id, (, num | id, (, num |
| **FOLLOW** | $, ), +, - | $, ), +, -, \*, / | $, ), +, -, \*, / |

**构造识别所有活前缀的DFA**如下：



## 3. 构造LR分析表

** (1)  (4)  (7)**

** (2)  (5)  (8)**

** (3)  (6)  (9)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **状态** | **action** | | | | | | | | | | **goto** | | | |
| **+** | **-** | **\*** | **/** | **id** | **num** | **(** | **)** | **$** | **E** | | **T** | **F** |
| **0** |  |  |  |  | **s4** | **S6** | **S5** |  |  | **1** | | **2** | **3** |
| **1** | **s7** | **s8** |  |  |  |  |  |  | **acc** |  | |  |  |
| **2** | **r3** | **r3** | **s9** | **s10** |  |  |  | **r3** | **r3** |  | |  |  |
| **3** | **r6** | **r6** | **r6** | **r6** |  |  |  | **r6** | **r6** |  | |  |  |
| **4** | **r7** | **r7** | **r7** | **r7** |  |  |  | **r7** | **r7** |  | |  |  |
| **5** |  |  |  |  | **s4** | **s6** | **s5** |  |  | **11** | | **2** | **3** |
| **6** | **r9** | **r9** | **r9** | **r9** |  |  |  | **r9** | **r9** |  | |  |  |
| **7** |  |  |  |  | **s4** | **s6** | **s5** |  |  |  | | **12** | **3** |
| **8** |  |  |  |  | **s4** | **s6** | **s5** |  |  |  | | **13** | **3** |
| **9** |  |  |  |  | **s4** | **s6** | **s5** |  |  |  | |  | **14** |
| **10** |  |  |  |  | **s4** | **s6** | **s5** |  |  |  | |  | **15** |
| **11** | **s7** | **s8** |  |  |  |  |  | **s16** |  |  | |  | **16** |
| **12** | **r1** | **r1** | **s9** | **s10** |  |  |  | **r1** | **r1** |  | |  |  |
| **13** | **r2** | **r2** | **s9** | **s10** |  |  |  | **r2** | **r2** |  | |  |  |
| **14** | **r4** | **r4** | **r4** | **r4** |  |  |  | **r4** | **r4** |  | |  |  |
| **15** | **r5** | **r5** | **r5** | **r5** |  |  |  | **r5** | **r5** |  | |  |  |
| **16** | **r8** | **r8** | **r8** | **r8** |  |  |  | **r8** | **r8** |  | |  |  |

## 4. 扩充分析栈

多定义一个结构栈数组，结构里有两个变量，一个为val, 一个为type。实现时，val其实是定义了两个变量，一个表示int时的值，一个表示real时的值，因为无法公用一个类型的变量。

定义type只有三种，一种为int, 一种为real, 一种为type\_error。

val由外部提供。可通过数组搜索。

## 5. 改造分析程序

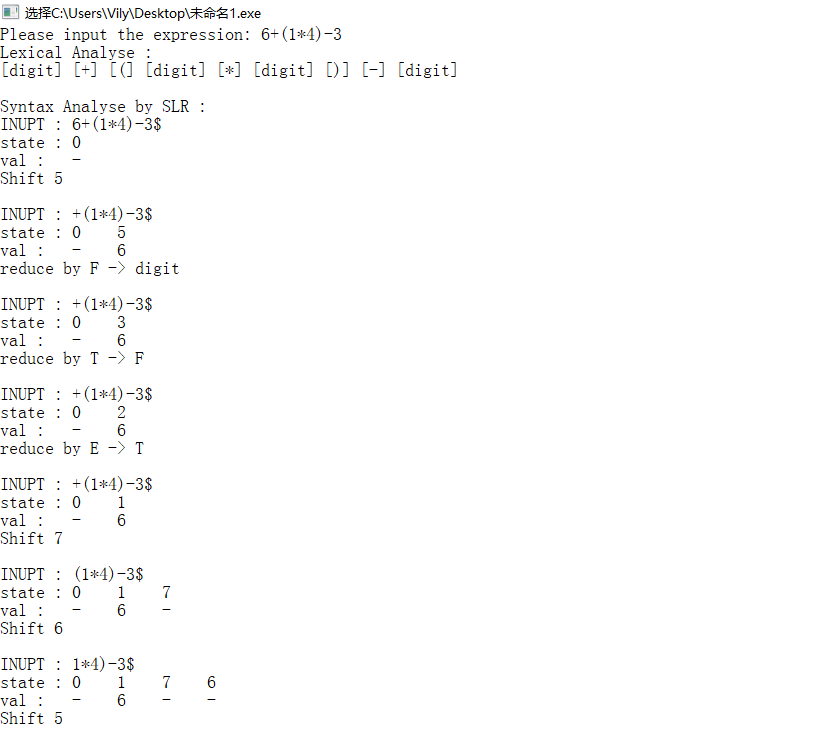
在归约时完成类型检查和求值。

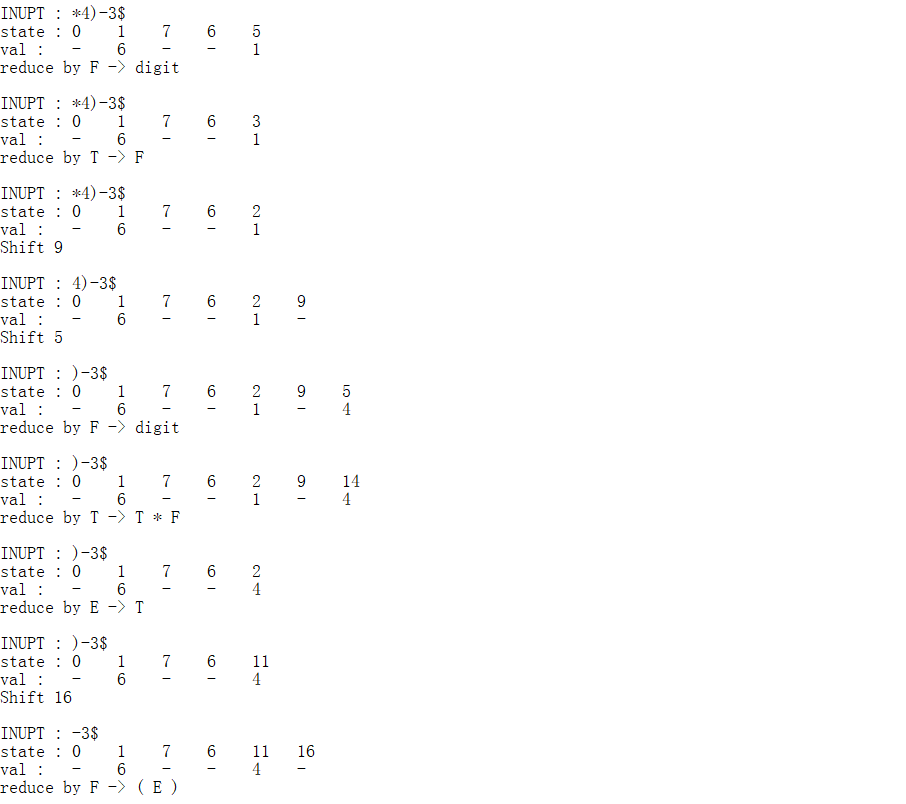
之所以与归约联系，是因为每一次归约决定着所用的是哪一个产生式。

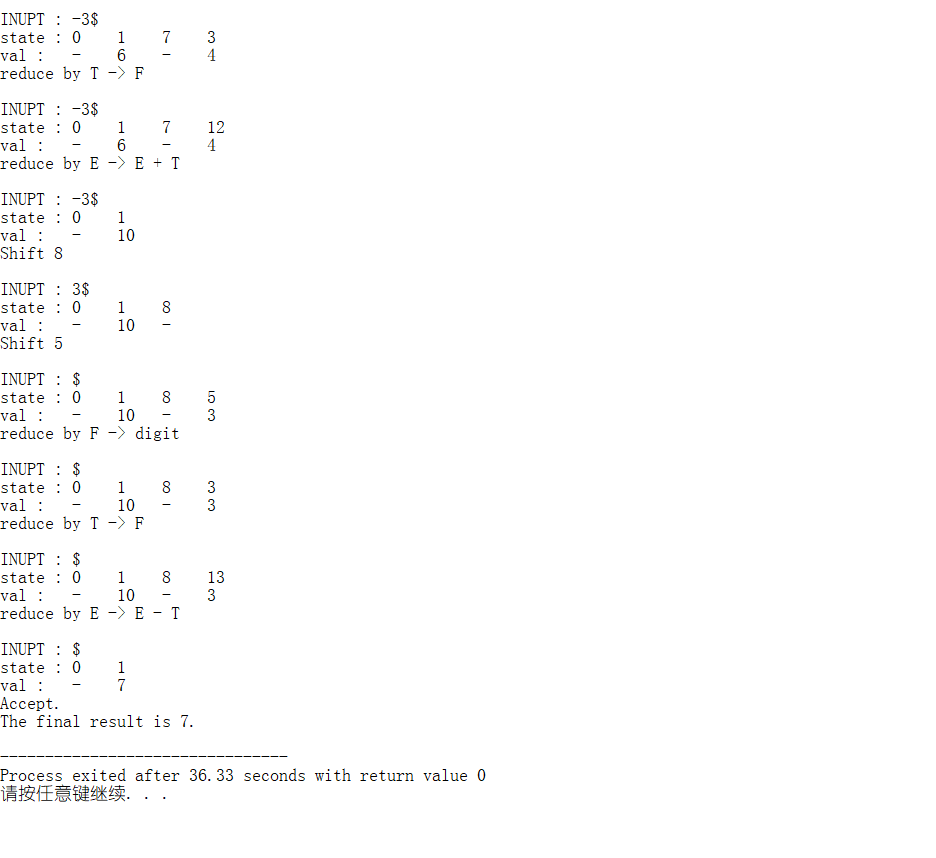
acc时打印最终求值结果和表达式类型。

# 五、实验结果

1. 测试样例：6+(1\*4)-3

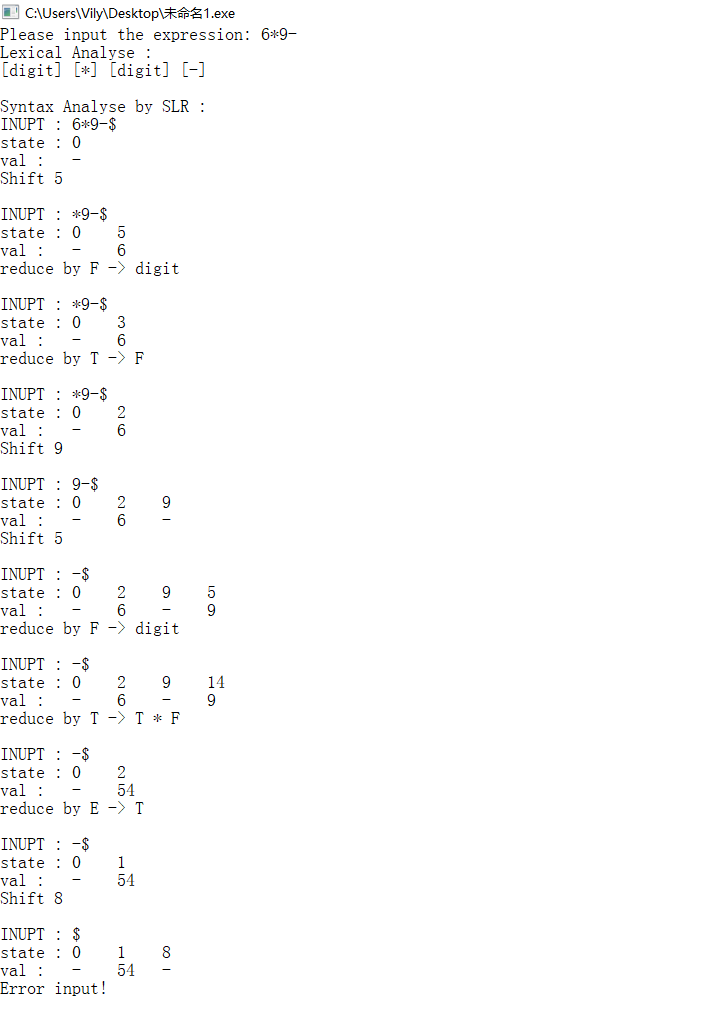






接受，输出正确结果。

2. 测试样例：6\*9-



输入错误，输出error。

# 六、总结与体会

通过本次对语义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，也更加理解了语义分析方法的精髓，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。

# 七、实验源代码

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <string.h>
5. #include <cstdio>
6. #define NN 20
7. #define MAXN 1000
8. #define INF 9999999
9. #define ERROR -1
10. **using** **namespace** std;
11. **char** \*token\_str[]={"","ID","digit","+","-","\*","/","(",")","$"};
12. **const** **char** \*ori\_grammer[]={"E' -> E","E -> E + T","E -> E - T","E -> T","T -> T \* F","T -> T / F","T -> F","F -> id","F -> ( E )","F -> digit"};
13. **enum**{E\_=MAXN+1,E,T,F};
14. **enum**{NUL=0,ID,NUM,ADD,SUB,MUL,DIV,OPL,OPR,END};
15. **const** **int** GRAMMER[NN][NN]={{E\_,E},{E,E,ADD,T},{E,E,SUB,T},{E,T},{T,T,MUL,F},{T,T,DIV,F},{T,F},{F,ID},{F,OPL,E,OPR},{F,NUM}};
16. **const** **int** GrammerL[]={1,3,3,1,3,3,1,1,3,1};
17. **const** **int** ACTION[NN][NN]={{INF,4,5,INF,INF,INF,INF,6,INF,INF},
18. {INF,INF,INF,7,8,INF,INF,INF,INF,0},
19. {INF,INF,INF,-3,-3,9,10,INF,-3,-3},
20. {INF,INF,INF,-6,-6,-6,-6,INF,-6,-6},
21. {INF,INF,INF,-7,-7,-7,-7,INF,-7,-7},
22. {INF,INF,INF,-9,-9,-9,-9,INF,-9,-9},
23. {INF,4,5,INF,INF,INF,INF,6,INF,INF},
24. {INF,4,5,INF,INF,INF,INF,6,INF,INF},
25. {INF,4,5,INF,INF,INF,INF,6,INF,INF},
26. {INF,4,5,INF,INF,INF,INF,6,INF,INF},
27. {INF,4,5,INF,INF,INF,INF,6,INF,INF},
28. {INF,INF,INF,7,8,INF,INF,INF,16,INF},
29. {INF,INF,INF,-1,-1,9,10,INF,-1,-1},
30. {INF,INF,INF,-2,-2,9,10,INF,-2,-2},
31. {INF,INF,INF,-4,-4,-4,-4,INF,-4,-4},
32. {INF,INF,INF,-5,-5,-5,-5,INF,-5,-5},
33. {INF,INF,INF,-8,-8,-8,-8,INF,-8,-8}};
35. **const** **int** GOTO[NN][NN]={{INF,INF,1,2,3},
36. {INF,INF,INF,INF,INF},
37. {INF,INF,INF,INF,INF},
38. {INF,INF,INF,INF,INF},
39. {INF,INF,INF,INF,INF},
40. {INF,INF,INF,INF,INF},
41. {INF,INF,11,2,3},
42. {INF,INF,INF,12,3},
43. {INF,INF,INF,13,3},
44. {INF,INF,INF,INF,14},
45. {INF,INF,INF,INF,15},{INF,INF,INF,INF,INF},
46. {INF,INF,INF,INF,INF},
47. {INF,INF,INF,INF,INF},
48. {INF,INF,INF,INF,INF},
49. {INF,INF,INF,INF,INF},
50. {INF,INF,INF,INF,INF}};
52. **int** isIDsym(**char** ch){
53. **return** (ch=='\_'||ch>='a'&&ch<='z'||ch>='A'&&ch<='Z');
54. }
56. **int** isDigit(**char** ch){
57. **return** (ch>='0'&&ch<='9');
58. }
60. **int** isBlank(**char** ch){
61. **return** (ch==' '||ch=='\t'||ch=='\n');
62. }
64. **int** SLRcalculate(vector<**int**> &token,vector<**int**> &token\_val,**int** &answer){
66. **char** tmp\_str[MAXN];
68. vector<**int**> S,val;
70. **int** I,a,pos=0,top;
71. S.push\_back(0);
72. val.push\_back(INF);
73. token.push\_back(END);
74. **while**(1){
75. I=S.back();
76. a=token[pos];
77. top=val.size()-1;
78. printf("INUPT : ");
79. **for**(**int** i=pos;i<token.size();i++){
80. **if**(token[i]==NUM)
81. printf("%d",token\_val[i]);
82. **else**
83. printf("%s",token\_str[token[i]]);
84. }
85. printf("\n");
86. printf("state :\t");
87. **for**(**int** i=0;i<S.size();i++){
88. printf("%-5d",S[i]);
89. }
90. printf("\n");printf("val :\t");
91. **for**(**int** i=0;i<val.size();i++){
92. **if**(val[i]!=INF)
93. printf("%-5d",val[i]);
94. **else**
95. printf("%-5s","- ");
96. }
97. printf("\n");
98. strcpy(tmp\_str,"");
99. **if**(ACTION[I][a]==INF)
100. **return** ERROR;
101. **if**(ACTION[I][a]>0){
102. sprintf(tmp\_str,"Shift %d",ACTION[I][a]);
103. S.push\_back(ACTION[I][a]);
104. val.push\_back(token\_val[pos]);
105. pos++;
106. }
107. **else** **if**(ACTION[I][a]<0){
108. **int** n=-ACTION[I][a];
109. **int** ntop=top-GrammerL[n]+1;
110. sprintf(tmp\_str,"reduce by %s",ori\_grammer[n]);
111. **switch**(n){
112. **case** 1: val[ntop]=val[top-2]+val[top];**break**;
113. **case** 2: val[ntop]=val[top-2]-val[top];**break**;
114. **case** 4: val[ntop]=val[top-2]\*val[top];**break**;
115. **case** 5: val[ntop]=val[top-2]/val[top];**break**;
116. **case** 8: val[ntop]=val[top-1];**break**;
117. }
118. **for**(**int** i=0;i<top-ntop;i++)
119. val.pop\_back();
120. **for**(**int** i=0;i<GrammerL[n];i++){
121. S.pop\_back();
122. }
123. **int** tmpgo=GOTO[S.back()][GRAMMER[n][0]-MAXN];
124. S.push\_back(tmpgo);
125. }
126. **else**{
127. answer=val[top];
128. **return** 0;
129. }
130. **if**(pos>=token.size())
131. **return** ERROR;
132. printf("%s\n\n",tmp\_str);}
133. }

136. **int** LexicalAnalyse(**char** \*str,vector<**int**> &token,vector<**int**> &token\_val){
137. **int** state=0;
138. **char** \*ptr=str,ch;
139. **int** value=0;
140. **while**(ptr){
141. ch=\*ptr;
142. **switch**(state){
143. **case** 0:
144. **if**(isDigit(ch)){
145. state=1;
146. }
147. **else** **if**(isIDsym(ch)){
148. state=2;
149. }
150. **else**{
151. state=3;
152. }
153. **break**;
154. **case** 1:
155. **if**(isDigit(ch)){
156. value=value\*10+ch-'0';
157. ptr++;
158. }
159. **else**{
160. token.push\_back(NUM);
161. token\_val.push\_back(value);
162. value=0;
163. state=0;
164. }
165. **break**;
166. **case** 2:
167. **if**(isIDsym(ch)||isDigit(ch)){
168. ptr++;
169. }
170. **else**{
171. token.push\_back(ID);
172. token\_val.push\_back(INF);
173. state=0;
174. }
175. **break**;
176. **case** 3:
177. **switch**(ch){
178. **case** '+': token.push\_back(ADD);**break**;
179. **case** '-': token.push\_back(SUB);**break**;
180. **case** '\*': token.push\_back(MUL);**break**;**case** '/': token.push\_back(DIV);**break**;
181. **case** '(': token.push\_back(OPL);**break**;
182. **case** ')': token.push\_back(OPR);**break**;
183. **default**:**if**(!isBlank(ch))**return** ERROR;
184. }
185. ptr++;
186. state=0;
187. **if**(!isBlank(ch))
188. token\_val.push\_back(INF);
189. **break**;
190. }
191. }
192. }
193. **int** main(){
194. **char** str[MAXN];
195. **int** answer,flag;
196. vector<**int**> token,token\_val;
197. printf("Please input the expression: ");
198. gets(str);
199. LexicalAnalyse(str,token,token\_val);
200. printf("Lexical Analyse :\n");
201. **for**(**int** i=0;i<token.size();i++){
202. printf("[%s] ",token\_str[token[i]]);
203. }
204. printf("\n\n");
205. printf("Syntax Analyse by SLR :\n");
206. flag=SLRcalculate(token,token\_val,answer);
207. **if**(flag==0){
208. printf("Accept.\n");
209. printf("The final result is %d.\n",answer);
210. }
211. **else**{
212. printf("Error input!\n");
213. }
214. **return** 0;
215. }