编译原理语法分析实验报告

2018211302班 2018210074 熊宇

1. 实验题目

语法分析程序的设计与实现

1. 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。要求所分析算术表达式由如下的文法产生。

E→E+T|E-T|T

T→T\*F|T/F|F

F→(E)|num

1. 实验要求

在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

方法1：编写递归调用程序实现自顶向下的分析。

方法2：编写LL(1)语法分析程序，要求如下。

1. 编程实现算法4.2，为给定文法自动构造预测分析表。
2. 编程实现算法4.1，构造LL(1)预测分析程序。

方法3：编写语法分析程序实现自底向上的分析，要求如下。

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

方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

这里我用四种方法分别实现。

1. 方法1
2. 实验分析

观察给定语法，如果遇到左递归，则需要改写语法来消除左递归。根据给定的语法，生成相应符号的First集和Fllow集。依照First集和Fllow集实现语法分析器的代码。

1. 消除左递归

G→E

E→TP

P→+TP|-TP|e

T→FQ

Q→\*FQ|/FQ|e

F→(E)|num

1. 非终结符的FIRST集合和FOLLOW集合

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

1. 实验源代码
2. #include <iostream>
3. #include<stdio.h>
4. #include<stdlib.h>
5. #include<string.h>
7. **using** **namespace** std;
9. #define LEN 255
10. **bool** flaP=**true**;
11. **char** src[LEN];
12. **int** i;
13. **void** E();
14. **void** T();
15. **void** P();
16. **void** F();
17. **void** Q();
19. **int** main()
20. {
21. printf("请输入待检测语句串，以$结束）:\n");
23. scanf("%s",&src);
25. i=0;
26. E();
27. **if**(src[i]=='$'&&flaP==**true**)
28. {
29. printf("语句合法\n");
30. }
31. **else**
32. {
33. printf("不合法\n");
34. }
35. system("pause");
36. **return** 0;
37. }
39. **void** E()
40. {
41. printf("E->TP\n");
42. T();
43. P();
45. }
47. **void** T()
48. {
49. printf("T->FQ\n");
50. F();
51. Q();
52. }
53. **void** P()
54. {
55. **if**(src[i]=='+')
56. {
57. i++;
58. printf("P->+TP\n");
59. T();
60. P();
61. }
62. **else** **if**(src[i]=='-')
63. {
64. printf("P->-TP\n");
65. i++;
66. T();
67. P();
68. }
70. }
72. **void** F()
73. {
74. **if**(src[i]=='(')
75. {
76. i++;
77. E();
78. **if**(src[i]==')')
79. {
80. i++;
81. printf("F->(E)\n");
82. }
83. **else**
84. flaP=**false**;
85. }
86. **else** **if**(src[i]>='0'&&src[i]<='9')
87. {
88. printf("F->num\n");
89. i++;
90. }
91. **else**
92. flaP=**false**;
93. }
95. **void** Q(){
97. **if**(src[i]=='\*')
98. {
99. i++;
100. F();
101. Q();
102. }
103. **else** **if**(src[i]=='/')
104. {
105. i++;
106. F();
107. Q();
108. }
110. }
111. 输入及输出
112. 请输入待检测语句串，以$结束）:
113. 8\*(8-4)/2$
114. E->TP
115. T->FQ
116. F->num
117. E->TP
118. T->FQ
119. F->num
120. P->-TP
121. T->FQ
122. F->num
123. F->(E)
124. F->num
125. 语句合法
126. Press any key to **continue** . . .
127. 方法2
128. 实验分析

首先构造并存储预测分析表，该预测分析表可以分析语句串、进行一定程度的错误处理，最终实现LL(1)分析程序。

1. 设计文法

以下e都表示ε，为空。

E→TP

P→+TP|-TP|e

T→FQ

Q→\*FQ|/FQ|e

F→(E)|num

1. 非终结符的FIRST集合和FOLLOW集合

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

1. LL(1)预测分析表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | - | \* | / | ( | ) | num | $ |
| E | error | error | error | error | E→TP | synch | E→TP | synch |
| P | P→+TP | P→-TP | error | error | error | P→e | error | P→e |
| T | synch | synch | error | error | T→FQ | synch | T→FQ | synch |
| Q | Q→e | Q→e | Q→\*FQ | Q→/FQ | error | Q→e | error | Q→e |
| F | synch | synch | synch | synch | F→(E) | synch | F→num | synch |

1. 实验源代码
2. #include <iostream>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
7. #define NumOfPro 10 //产生式个数
8. #define MaxLenOfPro 8 //产生式最大长度
9. #define NumOfEnd 8 //终结符个数
10. #define NumOfNonEnd 5 //非终结符个数
11. #define MaxOfF 8 //FIRST\FOLLOW集大小
12. #define MaxLenOfStr 100 //待输入的语句的最大长度
13. #define MaxLenOfNum 10 //待输入表达式中num的最大长度
14. #define ERROR -1 //分析表中error
15. #define SYNCH -2 //分析表中synch
16. #define StackInitialLen 10 //栈初始大小
17. #define StackInpleLen 5 //栈空间增量
18. #define NUM 1 //数字标记
19. #define OTHER 0 //其他字符标记
20. #define TRUE 1
21. #define FALSE 0
22. #define NOTFOUND -1
23. #define OVERFLOW -2
25. **using** **namespace** std;
27. **typedef** **struct** stack
28. {
29. **char**\* top; //栈顶指针
30. **char**\* bottom; //栈底指针
31. **int** sizeofstack; //栈大小
32. }stack;
34. //一些变量
35. **char** grammar[NumOfPro][MaxLenOfPro]; //文法集
36. **char** End[NumOfEnd]; //终结符集
37. **char** nonend[NumOfNonEnd]; //非终结符表
38. **char** FIRST[NumOfNonEnd][MaxOfF]; //FIRST集
39. **char** FOLLOW[NumOfNonEnd][MaxOfF]; //FOLLOW集
40. **int** L[NumOfNonEnd][NumOfEnd]; //语法预测分析表
41. **char** str[MaxLenOfStr+1]; //语句输入缓冲区
42. stack s; //栈
44. //LL(1)文法分析器函数
45. **void** InitialLL(); //初始化
46. **void** ProduceAnalysis(); //生成预测分析表
47. **void** GetString(); //获取输入的语句串
48. **void** Analysis(); //使用预测分析表进行分析
50. //分析函数
51. **int** GetStrLen(); //获取输入语句的长度
52. **int** JudgeEnd(**char** ch); //判断字符是否为终结符
53. **int** JudgeNonEnd(**char** ch); //判断字符是否为非终结符
54. **int** GetEndNum(**char** ch); //获取终结符ch在终结符集合里的下标
55. **int** GetNonEndNum(**char** ch); //获取非终结符ch在非终结符集合里的下标
56. **int** JudgeFirst(**char** A, **char** ch); //判断ch是否在A的FIRST集合里
57. **int** JudgeFollow(**char** A, **char** ch); //判断ch是否在A的FOLLOW集合里面
58. **void** PrintPro(**int** i); //打印产生式
59. **void** PrintStack(); //打印当前栈中符号
60. **void** PrintBuffer(**int** num); //打印当前输入缓冲区中的内容
61. **int** JudgeNum(**char** ch); //判断是否为num
62. **int** JuegeOverflow(); //判断是否溢出
64. //栈操作
65. **void** InitStack(); //构造一个空栈
66. **void** Push(**char** ch); //将字符ch压入栈顶
67. **void** Pop(); //弹出栈顶字符
68. **char** GetTop(); //获取栈顶字符
70. **int** main()
71. {
72. InitialLL(); //初始化LL
73. ProduceAnalysis(); //构造预测分析表
74. **for**(**int** i=0;i<NumOfNonEnd;i++)
75. {
76. **for**(**int** j=0;j<NumOfEnd;j++)
77. {
78. cout<<nonend[i]<<" "<<End[j]<<" ";
79. PrintPro(L[i][j]);
80. cout<<endl;
81. }
82. }
83. GetString(); //获取输入
84. Analysis(); //分析
85. system("pause");
86. **return** 0;
87. }
89. //栈操作
90. **void** InitStack() //构造一个空栈
91. {
92. s.bottom=(**char** \*)malloc(StackInitialLen\***sizeof**(**char**));
93. **if**(!s.bottom)
94. exit(OVERFLOW);
95. s.top=s.bottom;
96. s.sizeofstack=StackInitialLen;
97. }
98. **void** Push(**char** ch) //将字符ch压入栈顶
99. {
100. **if**(s.top-s.bottom >= s.sizeofstack)
101. {
102. //栈不够了，需要增加空间
103. s.bottom=(**char** \*)realloc(s.bottom,(s.sizeofstack+StackInpleLen)\***sizeof**(**char**));
104. **if**(!s.bottom)
105. exit(OVERFLOW);
106. s.top=s.bottom+s.sizeofstack;
107. s.sizeofstack+=StackInpleLen;
108. }
109. \*s.top++=ch; //压栈
110. }
111. **void** Pop() //弹出栈顶字符
112. {
113. **if**(s.bottom!=s.top)
114. s.top--;
115. }
116. **char** GetTop() //获取栈顶字符
117. {
118. **char** ch;
119. **if**(s.bottom!=s.top)
120. ch=\*(s.top-1);
121. **return** ch;
122. }
124. //LL(1)文法分析器函数
125. **void** InitialLL() //初始化
126. {
127. **int** i, j;
128. InitStack();
129. Push('$');
130. Push('E');  //初始化栈：构造空栈，并压入'$'与'E'
132. //设置文法表，存储P代表E'，Q代表T'，e代表ε,n代表num
133. strcpy(grammar[0], "E#TP#");  //E →TE'
134. strcpy(grammar[1], "P#+TP#");  //E' →+TE'
135. strcpy(grammar[2], "P#-TP#");  //E' →-TE'
136. strcpy(grammar[3], "P#e#");  //E' →ε
137. strcpy(grammar[4], "T#FQ#");  //T →FT'
138. strcpy(grammar[5], "Q#\*FQ#");  //T' →\*FT'
139. strcpy(grammar[6], "Q#/FQ#");  //T' →/FT'
140. strcpy(grammar[7], "Q#e#");  //T' →ε
141. strcpy(grammar[8], "F#(E)#");  //F →(E)
142. strcpy(grammar[9], "F#n#");  //F →num
144. **for** (i = 0; i < NumOfNonEnd; i++)  //LL(1)语法预测分析表初始化：所有表项置为错误ERROR
145. **for** (j = 0; j < NumOfEnd; j++)
146. L[i][j] = ERROR;
147. strcpy(End, "+-\*/()n$");  //初始化终结符表
148. strcpy(nonend, "EPTQF");  //初始化非终结符表
150. //初始化FIRST集
151. strcpy(FIRST[0], "(n#");
152. strcpy(FIRST[1], "+-e#");
153. strcpy(FIRST[2], "(n#");
154. strcpy(FIRST[3], "\*/e#");
155. strcpy(FIRST[4], "(n#");
157. //初始化FOLLOW集
158. strcpy(FOLLOW[0], ")$#");
159. strcpy(FOLLOW[1], ")$#");
160. strcpy(FOLLOW[2], "+-)$#");
161. strcpy(FOLLOW[3], "+-)$#");
162. strcpy(FOLLOW[4], "+-\*/)$#");
163. }
164. **int** GetStrLen() //获取输入语句的长度
165. {
166. **int** count=0;
167. **for**(count=0;str[count]!='\0';count++);
168. **return** count;
169. }
170. **int** JudgeEnd(**char** ch) //判断字符是否为终结符
171. {
172. **if** (ch=='+' || ch=='-' || ch=='\*' || ch=='/' || ch=='(' || ch==')' ||  ch=='n' || ch=='$')
173. **return** TRUE;
174. **else**
175. **return** FALSE;
176. }
177. **int** JudgeNonEnd(**char** ch) //判断字符是否为非终结符
178. {
179. **if** (ch=='E' || ch=='P' || ch=='T' || ch=='Q' || ch=='F')
180. **return** TRUE;
181. **else**
182. **return** FALSE;
183. }
184. **int** GetEndNum(**char** ch) //获取终结符ch在终结符集合里的下标
185. {
186. **int** i=0;
187. **for**(i=0;i<NumOfEnd;i++)
188. {
189. **if**(ch==End[i])
190. **return** i;
191. }
192. **return** NOTFOUND;
193. }
194. **int** GetNonEndNum(**char** ch) //获取非终结符ch在非终结符集合里的下标
195. {
196. **int** i=0;
197. **for**(i=0;i<NumOfNonEnd;i++)
198. {
199. **if**(ch==nonend[i])
200. **return** i;
201. }
202. **return** NOTFOUND;
203. }
204. **int** JudgeFirst(**char** A, **char** ch) //判断ch是否在A的FIRST集合里
205. {
206. **int** i,j;
207. i=GetNonEndNum(A);
208. **for**(j=0;FIRST[i][j]!='#';j++)
209. {
210. **if**(ch==FIRST[i][j])
211. **return** TRUE;
212. }
213. **return** FALSE;
214. }
215. **int** JudgeFollow(**char** A, **char** ch) //判断ch是否在A的FOLLOW集合里面
216. {
217. **int** i,j;
218. i=GetNonEndNum(A);
219. **for**(j=0;FOLLOW[i][j]!='#';j++)
220. {
221. **if**(ch==FOLLOW[i][j])
222. **return** TRUE;
223. }
224. **return** FALSE;
225. }
226. **void** PrintPro(**int** i) //打印产生式
227. {
228. printf ("  当前产生式为 :   ");
229. **switch** (i)  //根据产生式的标号打印出对应是输出产生式
230. {
231. **case** 0:
232. printf ("E → TE'\n");
233. **break**;
234. **case** 1:
235. printf ("E' → +TE'\n");
236. **break**;
237. **case** 2:
238. printf ("E' → -TE'\n");
239. **break**;
240. **case** 3:
241. printf ("E' → ε\n");
242. **break**;
243. **case** 4:
244. printf ("T → FT'\n");
245. **break**;
246. **case** 5:
247. printf ("T' → \*FT'\n");
248. **break**;
249. **case** 6:
250. printf ("T' → /FT'\n");
251. **break**;
252. **case** 7:
253. printf ("T' → ε\n");
254. **break**;
255. **case** 8:
256. printf ("F → (E)\n");
257. **break**;
258. **case** 9:
259. printf ("F → num\n");
260. **break**;
261. **default**:
262. **break**;
263. }
264. }
265. **void** PrintStack() //打印当前栈中符号
266. {
267. **char** \*sp = NULL;
268. printf ("  栈： ");
269. **for** (sp = s.bottom; sp != s.top; sp++)
270. {
271. **switch** (\*sp)
272. {
273. **case** 'P':  //将P转换为E'输出
274. printf ("E'");
275. **break**;
276. **case** 'Q':  //将Q转换为T'输出
277. printf ("T'");
278. **break**;
279. **case** 'n':  //将n转换为num输出
280. printf ("num");
281. **break**;
282. **default**:  //其他情况直接输出
283. printf ("%c", \*sp);
284. **break**;
285. }
286. }
287. printf ("\n");
288. }
289. **void** PrintBuffer(**int** num) //打印当前输入缓冲区中的内容
290. {
291. **int** i;
292. printf ("  输入： ");
293. **for** (i = num; str[i] != '$'; i++)
294. printf ("%c", str[i]);
295. printf ("$\n");
296. }
297. **int** JudgeNum(**char** ch) //判断是否为num
298. {
299. **if** (ch >= '0' && ch <= '9')
300. **return** TRUE;
301. **else**
302. **return** FALSE;
303. }
304. **int** JuegeOverflow() //判断是否溢出
305. {
306. **int** ip, bp, width;
307. **char** a;
308. **int** flag1 = OTHER, flag2 = OTHER;  //字符类型标记
309. **int** toolong = FALSE;  //过长标记
310. **for** (ip = 0; (str[ip] != '\0') && (! toolong); ip += width)
311. {
312. a = str[ip];
313. **if** (a=='+' || a=='-' || a=='\*' || a=='/' || a=='(' || a==')' || a=='$')
314. width = 1;
315. **else**  //对输入表达式中代表num的子串的长度进行判断
316. {
317. bp = ip;
318. width = 0;
319. **if** (JudgeNum(a))
320. flag1 = flag2 = NUM;
321. bp ++;
322. **do**  //获取输入表达式中代表num的子串的长度
323. {
324. a = str[bp];
325. width ++;
326. **if** (width > MaxLenOfNum)  //若大于子串的最大长度，标记过长并跳出循环
327. {
328. toolong = TRUE;
329. **break**;
330. }
331. **if** (JudgeNum (a))
332. flag2 = NUM;
333. **else**
334. flag2 = OTHER;
335. bp ++;
336. } **while** (flag1 == flag2);  //两个标记相等，即子串没有结束，继续循环
337. }
338. }
339. **return** toolong;  //返回过长标记
340. }
341. **void** ProduceAnalysis() //生成预测分析表
342. {
343. **int** i, j;
344. **int** n1, n2;
345. **char** ch, A;
346. **for** (i = 0; i < NumOfPro; i++)   //对于每个产生式A →α
347. {
348. A = grammar[i][0];
349. ch = grammar[i][2];
350. **if** (JudgeNonEnd (ch))  //若a∈ FIRST(α)，M[A,a]中应放入产生式A →α
351. {
352. **for** (j = 0; j < NumOfEnd; j++)
353. {
354. **if** (JudgeFirst (ch, End[j]))
355. {
356. n1 = GetNonEndNum(A);
357. L[n1][j] = i;
358. }
359. }
360. }
361. **else** **if** (JudgeEnd(ch))
362. {
363. n1 = GetNonEndNum(A);
364. n2 = GetEndNum(ch);
365. L[n1][n2] = i;
366. }
367. **else** **if** (ch == 'e')  //若ε∈FIRST(α)，且b∈FOLLOW(A)，M[A,b]中应放入产生式A →α
368. {
369. n1 = GetNonEndNum(A);
370. **for** (j = 0; FOLLOW[n1][j] != '#'; j++)
371. {
372. n2 = GetEndNum(FOLLOW[n1][j]);
373. L[n1][n2] = i;
374. }
375. }
376. }
377. **for** (i = 0; i < NumOfNonEnd; i++)  //置同步出错信息
378. {
379. **for** (j = 0; FOLLOW[i][j] != '#'; j++)  //若b∈FOLLOW(A)，且M[A,b]为ERROR，则把M[A,b]赋值为为同步信息SYNCH
380. {
381. n1 = GetEndNum(FOLLOW[i][j]);
382. **if** (L[i][n1] == ERROR)
383. L[i][n1] = SYNCH;
384. }
385. }
386. }
387. **void** GetString() //获取输入的语句串
388. {
389. **int** len = 0;  //待分析的输入表达式的长度
390. **int** flag = FALSE;
391. **do**
392. {
393. printf ("请输入待分析语句串，用'$'结束:\n");
394. scanf ("%s", &str);
395. len = GetStrLen ();
396. **if** (str[len-1] != '$')  //若每输入结尾符'$'，将其补在待分析输入表达式的最后
397. {
398. str[len] = '$';
399. str[len+1] = '\0';
400. len ++;
401. }
402. **if** (len > MaxLenOfStr)  //判断输入表达式是否过长
403. printf ("语句过长，请重新输入！\n");
404. flag = JuegeOverflow();  //判断输入表达式中代表num的子串是否过长
405. **if** (flag)
406. printf ("数字过长，请重新输入！\n");
407. } **while** (len > MaxLenOfStr || flag);
408. }
409. **void** Analysis() //使用预测分析表进行分析
410. {
411. **int** ip = 0;  //输入缓冲区指针
412. **int** step = 0;  //分析步数
413. **char** X, a, c;
414. **int** i, j, n1, n2, bp;
415. **int** width = 0;  //输入表达式中代表num的子串的长度
416. **char** b[MaxLenOfNum+1];  //存储输入表达式中代表num的子串
417. **int** flag1 = OTHER, flag2 = OTHER;  //字符类型标记
418. **do**
419. {
420. step++;
421. printf ("Step %d :\n", step);
422. X = GetTop ();  //获取栈顶符号
423. a = str[ip];  //获取输入串中将要进行分析的符号
424. **for** (j = 0; j <= MaxLenOfNum; j++)  //对存储输入串中代表num的子串的数组进行初始化
425. b[j] = '\0';
426. **if** (a=='+' || a=='-' || a=='\*' || a=='/' || a=='(' || a==')' || a=='$')  //对输入串中将要进行分析的符号进行处理
427. {
428. width = 1;
429. b[0] = a;
430. }
431. **else**  //处理输入表达式中代表num的子串，将它们转化为'n'进行分析
432. {
433. bp = ip;
434. width = 0;
435. **if** (JudgeNum (a))
436. flag1 = flag2 = NUM;
437. bp ++;
438. c = a;
439. **do**  //获取输入表达式中代表num的子串，将其存入数组b
440. {
441. width ++;
442. b[width-1] = c;
443. c = str[bp];
444. **if** (JudgeNum (c))
445. flag2 = NUM;
446. **else**
447. flag2 = OTHER;
448. bp ++;
449. } **while** (flag1 == flag2);  //两个标记相等，即子串没有结束，继续循环
451. **if** (flag1 == NUM)
452. a = 'n';
453. }
454. PrintStack();  //打印当前栈中的符号
455. PrintBuffer(ip);  //打印当前输入缓冲区中的符号串
456. **if** (JudgeEnd(X))  //栈顶符号是终结符：不论正确与否，都弹出栈顶符，ip前移
457. {
458. **if** (X == a)
459. {
460. Pop();
461. ip += width;
462. }
463. **else**  //若栈顶终结符与ip指向的字符不匹配，提示错误
464. {
465. Pop();
466. ip += width;
467. printf ("  输出：错误 !\n");
468. }
469. }
470. **else**  //栈顶符号是非终结符
471. {
472. n1 = GetNonEndNum(X);
473. n2 = GetEndNum(a);
474. **if** (L[n1][n2] != ERROR && L[n1][n2] != SYNCH)  //正确情形：分析表项M[X][a]是非终结符X的一个产生式
475. {
476. Pop();  //弹出栈顶符
477. i = L[n1][n2];
478. **if** (grammar[i][2] != 'e')  //如果产生式右边不是ε，将对应产生式的右边逆序压入栈中
479. {
480. **for** (j = 2; grammar[i][j] != '#'; j++);  //将对应产生式的右边逆序压入栈中
481. **for** (j--; j >= 2; j--)
482. Push(grammar[i][j]);
483. }
484. PrintPro(i);  //输出产生式
485. }
486. **else** **if** (L[n1][n2] == ERROR)  //分析表项M[X][a]为空，ip前移，跳过当前输入字符（串）
487. {
488. ip += width;
489. printf ("  输出：错误 ! 跳过 %s\n", b);
490. }
491. **else** **if** (L[n1][n2] == SYNCH)  //分析表项M[X][a]为同步信息，则弹出栈顶符
492. {
493. Pop();
494. printf ("  输出：错误 ! 弹栈 ");
495. **switch** (X)
496. {
497. **case** 'P':  //将P转换为E'输出
498. printf ("E'");
499. **break**;
500. **case** 'Q':  //将Q转换为T'输出
501. printf ("T'");
502. **break**;
503. **default**:  //其他情况直接输出
504. printf ("%c", X);
505. **break**;
506. }
507. printf ("\n");
508. }
509. }
510. } **while** (X != '$');
511. }
512. 程序输入输出
513. E +   当前产生式为 :
514. E -   当前产生式为 :
515. E \*   当前产生式为 :
516. E /   当前产生式为 :
517. E (   当前产生式为 :   E → TE'
519. E )   当前产生式为 :
520. E n   当前产生式为 :   E → TE'
522. E $   当前产生式为 :
523. P +   当前产生式为 :   E' → +TE'
525. P -   当前产生式为 :   E' → -TE'
527. P \*   当前产生式为 :
528. P /   当前产生式为 :
529. P (   当前产生式为 :
530. P )   当前产生式为 :   E' → ε
532. P n   当前产生式为 :
533. P $   当前产生式为 :   E' → ε
535. T +   当前产生式为 :
536. T -   当前产生式为 :
537. T \*   当前产生式为 :
538. T /   当前产生式为 :
539. T (   当前产生式为 :   T → FT'
541. T )   当前产生式为 :
542. T n   当前产生式为 :   T → FT'
544. T $   当前产生式为 :
545. Q +   当前产生式为 :   T' → ε
547. Q -   当前产生式为 :   T' → ε
549. Q \*   当前产生式为 :   T' → \*FT'
551. Q /   当前产生式为 :   T' → /FT'
553. Q (   当前产生式为 :
554. Q )   当前产生式为 :   T' → ε
556. Q n   当前产生式为 :
557. Q $   当前产生式为 :   T' → ε
559. F +   当前产生式为 :
560. F -   当前产生式为 :
561. F \*   当前产生式为 :
562. F /   当前产生式为 :
563. F (   当前产生式为 :   F → (E)
565. F )   当前产生式为 :
566. F n   当前产生式为 :   F → num
568. F $   当前产生式为 :
569. 请输入待分析语句串，用'$'结束:
570. 8\*(8-6)/4$
571. Step 1 :
572. 栈： $E
573. 输入： 8\*(8-6)/4$
574. 当前产生式为 :   E → TE'
575. Step 2 :
576. 栈： $E'T
577. 输入： 8\*(8-6)/4$
578. 当前产生式为 :   T → FT'
579. Step 3 :
580. 栈： $E'T'F
581. 输入： 8\*(8-6)/4$
582. 当前产生式为 :   F → num
583. Step 4 :
584. 栈： $E'T'num
585. 输入： 8\*(8-6)/4$
586. Step 5 :
587. 栈： $E'T'
588. 输入： \*(8-6)/4$
589. 当前产生式为 :   T' → \*FT'
590. Step 6 :
591. 栈： $E'T'F\*
592. 输入： \*(8-6)/4$
593. Step 7 :
594. 栈： $E'T'F
595. 输入： (8-6)/4$
596. 当前产生式为 :   F → (E)
597. Step 8 :
598. 栈： $E'T')E(
599. 输入： (8-6)/4$
600. Step 9 :
601. 栈： $E'T')E
602. 输入： 8-6)/4$
603. 当前产生式为 :   E → TE'
604. Step 10 :
605. 栈： $E'T')E'T
606. 输入： 8-6)/4$
607. 当前产生式为 :   T → FT'
608. Step 11 :
609. 栈： $E'T')E'T'F
610. 输入： 8-6)/4$
611. 当前产生式为 :   F → num
612. Step 12 :
613. 栈： $E'T')E'T'num
614. 输入： 8-6)/4$
615. Step 13 :
616. 栈： $E'T')E'T'
617. 输入： -6)/4$
618. 当前产生式为 :   T' → ε
619. Step 14 :
620. 栈： $E'T')E'
621. 输入： -6)/4$
622. 当前产生式为 :   E' → -TE'
623. Step 15 :
624. 栈： $E'T')E'T-
625. 输入： -6)/4$
626. Step 16 :
627. 栈： $E'T')E'T
628. 输入： 6)/4$
629. 当前产生式为 :   T → FT'
630. Step 17 :
631. 栈： $E'T')E'T'F
632. 输入： 6)/4$
633. 当前产生式为 :   F → num
634. Step 18 :
635. 栈： $E'T')E'T'num
636. 输入： 6)/4$
637. Step 19 :
638. 栈： $E'T')E'T'
639. 输入： )/4$
640. 当前产生式为 :   T' → ε
641. Step 20 :
642. 栈： $E'T')E'
643. 输入： )/4$
644. 当前产生式为 :   E' → ε
645. Step 21 :
646. 栈： $E'T')
647. 输入： )/4$
648. Step 22 :
649. 栈： $E'T'
650. 输入： /4$
651. 当前产生式为 :   T' → /FT'
652. Step 23 :
653. 栈： $E'T'F/
654. 输入： /4$
655. Step 24 :
656. 栈： $E'T'F
657. 输入： 4$
658. 当前产生式为 :   F → num
659. Step 25 :
660. 栈： $E'T'num
661. 输入： 4$
662. Step 26 :
663. 栈： $E'T'
664. 输入： $
665. 当前产生式为 :   T' → ε
666. Step 27 :
667. 栈： $E'
668. 输入： $
669. 当前产生式为 :   E' → ε
670. Step 28 :
671. 栈： $
672. 输入： $
673. Press any key to **continue** . . .
674. 方法3
     1. 实验分析

根据题目所给出的文法构造相应的拓广文法，并求出该文法各非终结符的FIRST、FOLLOW集合；构造拓广文法的项目集规范族，并构造出识别所有前缀的DFA；构造文法的LR分析表；由此构造LR分析程序。

* 1. 设计文法

（1）S→E

（2）E→E+T

（3）E→E-T

（4）E→T

（5）T→T\*F

（6）T→T/F

（7）T→F

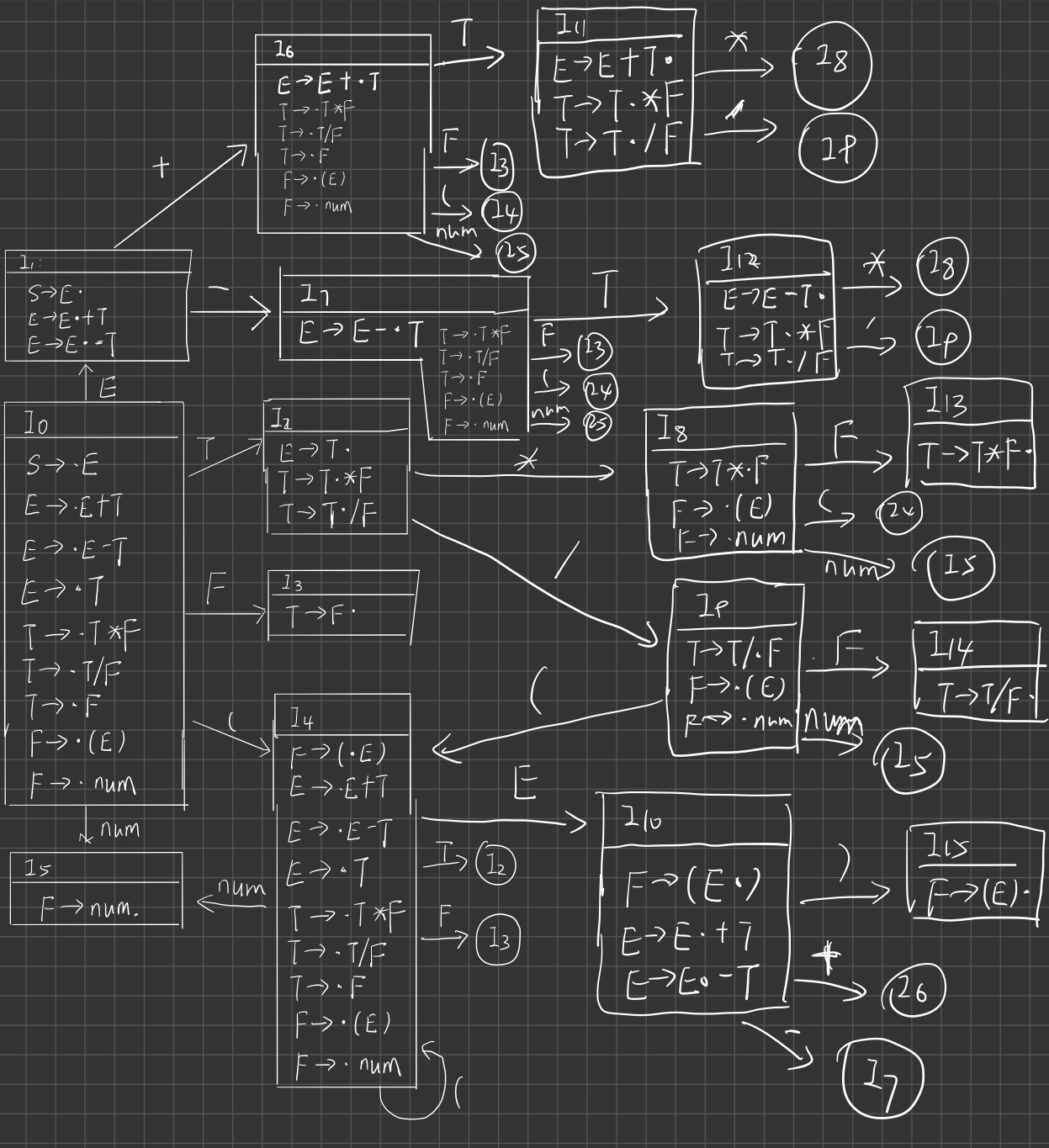
（8）F→(E)

（9）F→num

* 1. 非终结符的FIRST和FOLLOW集合

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

* 1. 识别所有活前缀的DFA,构造项目集规范族



* 1. LR分析表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | Action | | | | | | | | Goto | | | |
| ( | ) | + | - | \* | / | num | $ | S | E | T | F |
| 0 | S4 |  |  |  |  |  | S5 |  |  | 1 | 2 | 3 |
| 1 |  |  | S6 | S7 |  |  |  | acc |  |  |  |  |
| 2 |  | R4 | R4 | R4 | S8 | S9 |  | R4 |  |  |  |  |
| 3 |  | R7 | R7 | R7 | R7 | R7 |  | R7 |  |  |  |  |
| 4 | S4 |  |  |  |  |  | S5 |  |  | 10 | 2 | 3 |
| 5 |  | R9 | R9 | R9 | R9 | R9 |  | R9 |  |  |  |  |
| 6 | S4 |  |  |  |  |  | S5 |  |  |  | 11 | 3 |
| 7 | S4 |  |  |  |  |  | S5 |  |  |  | 12 | 3 |
| 8 | S4 |  |  |  |  |  | S5 |  |  |  |  | 13 |
| 9 | S4 |  |  |  |  |  | S5 |  |  |  |  | 14 |
| 10 |  | S15 | S6 | S7 |  |  |  |  |  |  |  |  |
| 11 |  | R2 | R2 | R2 | S8 | S9 |  | R2 |  |  |  |  |
| 12 |  | R3 | R3 | R3 | S8 | S9 |  | R3 |  |  |  |  |
| 13 |  | R5 | R5 | R5 | R5 | R5 |  | R5 |  |  |  |  |
| 14 |  | R6 | R6 | R6 | R6 | R6 |  | R6 |  |  |  |  |
| 15 |  | R8 | R8 | R8 | R8 | R8 |  | R8 |  |  |  |  |

* 1. 源代码

1. #include <iostream>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
6. **using** **namespace** std;
8. **int** siganl,state; //移进规约压入栈的符号，以及转移到的状态
9. **int** tstack[50]; //栈
10. **int** top=-1; //栈顶元素下标
11. **char** buffer[50]; //输入缓冲区
12. **int** number[50]; //将输入字符串转化为编码
13. **int** pos=0; //number数组下标
14. **int** X; //当前获取到的输入符号的编码
15. **int** flag=1; //循环标志
17. **typedef** **struct** Gra
18. {
19. **int** left; //产生式左侧非终结符的编码
20. **int** right[4]; //产生式右侧的编码串
21. **int** len; //产生式右侧的长度
22. }Gra;
24. Gra pro[10]={
25. //所有产生式
26. //下标从1开始
27. //SETF 101-104
28. //$()+-\*/num 01234567
29. {0},
30. {101,{0,102},1},/\*S -> E\*/
31. {102,{0,102,3,103},3},/\*E -> E+T\*/
32. {102,{0,102,4,103},3},/\*E -> E-T\*/
33. {102,{0,103},1},/\*E -> T\*/
34. {103,{0,103,5,104},3},/\*T -> T\*F\*/
35. {103,{0,103,6,104},3},/\*T -> T/F\*/
36. {103,{0,104},1},/\*T -> F\*/
37. {104,{0,1,101,2},3},/\*F -> (E)\*/
38. {104,{0,7},1},/\*F -> num\*/
39. };
41. **int** GOTO[16][5]={/\*LR分析表goto\*/
42. //多构造一列为了下标从1开始 SETF
43. {0,0,1,2,3},/\*0\*/
44. {0,0,0,0,0},/\*1\*/
45. {0,0,0,0,0},/\*2\*/
46. {0,0,0,0,0},/\*3\*/
47. {0,0,10,2,3},/\*4\*/
48. {0,0,0,0,0},/\*5\*/
49. {0,0,0,11,3},/\*6\*/
50. {0,0,0,12,3},/\*7\*/
51. {0,0,0,0,13},/\*8\*/
52. {0,0,0,0,14},/\*9\*/
53. {0,0,0,0,0},/\*10\*/
54. {0,0,0,0,0},/\*11\*/
55. {0,0,0,0,0},/\*12\*/
56. {0,0,0,0,0},/\*13\*/
57. {0,0,0,0,0},/\*14\*/
58. {0,0,0,0,0},/\*15\*/
59. };
61. **int** ACTION[16][8]={/\*LR分析表action\*/
62. //acc为999
63. //51期待(或运算对象首字符，但出现运算符或者$
64. //52括号不匹配，删掉右括号
65. //53期待运算符号，但出现(或运算对象
66. //S为正数，R为负数
67. //$()+-\*/num 01234567
68. {51,4,52,51,51,51,51,5},//0
69. {999,53,52,6,7,0,0,53},//1
70. {-4,53,-4,-4,-4,8,9,53},//2
71. {-7,0,-7,-7,-7,-7,-7,0},//3
72. {0,4,52,51,51,51,51,5},//4
73. {-9,0,-9,-9,-9,-9,-9,0},//5
74. {51,4,52,51,51,51,51,5},//6
75. {51,4,52,51,51,51,51,5},//7
76. {51,4,52,51,51,51,51,5},//8
77. {51,4,52,51,51,51,51,5},//9
78. {0,53,15,6,7,0,0,53},//10
79. {-2,53,-2,-2,-2,8,9,53},//11
80. {-3,53,-3,-3,-3,8,9,53},//12
81. {-5,0,-5,-5,-5,-5,-5,0},//13
82. {-6,0,-6,-6,-6,-6,-6,0},//14
83. {-8,0,-8,-8,-8,-8,-8,0}//15
84. };
86. **void** Transform(); //将读入缓冲区的字符数组转换为相应的编码
87. **void** GetNumber(); //获取当前输入符号串的元素
88. **void** Push(**int** A,**int** s); //符号、状态入栈
89. **void** Pop(); //出栈
90. **void** Shift(); //移进
91. **void** Reduce(); //规约
92. **void** Acc(); //accept
93. //错误处理
94. //51期待(或运算对象首字符，但出现运算符或者$
95. //52括号不匹配，删掉右括号
96. //53期待运算符号，但出现(或运算对象
97. **void** Err1();
98. **void** Err2();
99. **void** Err3();
101. **int** main()
102. {
103. cout<<"请输入一个待分析串，以$结束"<<endl;
104. scanf("%s",buffer);
105. Transform(); //将字符转化为编码
106. Push(0,0); //状态S0入栈
107. GetNumber(); //从已经变成编码的buffer读取一个字符
108. **while**(flag)
109. {
110. //cout<<tstack[top]<<" "<<X<<endl;
111. **if**(ACTION[tstack[top]][X]>0 && ACTION[tstack[top]][X]<50)
112. {
113. //进行移进操作
114. Shift();
115. GetNumber();
116. }
117. **else** **if**(ACTION[tstack[top]][X]<0)
118. {
119. //进行规约操作
120. Reduce();
121. }
122. **else** **if**(ACTION[tstack[top]][X]==999)
123. {
124. //accept
125. Acc();
126. }
127. **else**
128. {
129. **switch**(ACTION[tstack[top]][X])
130. {
131. **case** 51:
133. Err1();
134. **break**;
135. **case** 52:
136. Err2();
137. GetNumber();
138. **break**;
139. **case** 53:
140. Err3();
141. GetNumber();
142. **break**;
143. }
144. }
145. }
146. system("pause");
147. **return** 0;
148. }
150. **void** Transform() //将读入缓冲区的字符数组转换为相应的编码
151. {
152. //把读入的字符串转化为编码存在number这个编码数组
153. **int** i=0,j=0; //i为buffer下标，j为number下标
154. **while**(buffer[i]!='$')
155. {
156. **if**(buffer[i]>=48 && buffer[i]<=57)
157. {
158. //当前字符为数字，num
159. **while**(buffer[i]>=48 && buffer[i]<=57)
160. i++;
161. number[j++]=7;
162. }
163. **else**
164. {
165. **switch** (buffer[i++])
166. {
167. //$()+-\*/num 01234567
168. **case** '&':
169. number[j++]=0;
170. **break**;
171. **case** '(':
172. number[j++]=1;
173. **break**;
174. **case** ')':
175. number[j++]=2;
176. **break**;
177. **case** '+':
178. number[j++]=3;
179. **break**;
180. **case** '-':
181. number[j++]=4;
182. **break**;
183. **case** '\*':
184. number[j++]=5;
185. **break**;
186. **case** '/':
187. number[j++]=6;
188. **break**;
189. }
190. }
192. }
193. number[j]=0;
194. }
195. **void** GetNumber() //获取当前输入符号串的元素
196. {
197. X=number[pos++];
198. }
199. **void** Push(**int** A,**int** s) //符号、状态入栈
200. {
201. top++;
202. tstack[top]=A; //符号A入栈
203. top++;
204. tstack[top]=s; //状态S入栈
205. }
206. **void** Pop() //出栈
207. {
208. top--;
209. top--;
210. }
211. **void** Shift() //移进
212. {
213. **int** temp;
214. temp=ACTION[tstack[top]][X]; //查找表，确定需要移入的状态
215. Push(X,temp); //当前读入字符与状态入栈
216. printf("S%d\t移进.\n",temp);
217. }
218. **void** Reduce() //规约
219. {
220. **int** x,y,z;
221. x=-ACTION[tstack[top]][X];
222. printf("R%d\t规约\t",x);
223. **switch** (pro[x].left)
224. {
225. **case** 101:
226. cout<<"S -> ";
227. **break**;
228. **case** 102:
229. cout<<"E -> ";
230. **break**;
231. **case** 103:
232. cout<<"T -> ";
233. **break**;
234. **case** 104:
235. cout<<"F -> ";
236. **break**;
237. }
238. **for**(**int** i=1;i<=pro[x].len;i++)
239. {
240. Pop(); //将栈中元素，按照产生式右边的长度依次弹出
241. **switch** (pro[x].right[i])
242. {
243. **case** 0:
244. printf("$");
245. **break**;
246. **case** 1:
247. printf("(");
248. **break**;
249. **case** 2:
250. printf(")");
251. **break**;
252. **case** 3:
253. printf("+");
254. **break**;
255. **case** 4:
256. printf("-");
257. **break**;
258. **case** 5:
259. printf("\*");
260. **break**;
261. **case** 6:
262. printf("/");
263. **break**;
264. **case** 7:
265. printf("num");
266. **break**;
267. **case** 101:
268. printf("S");
269. **break**;
270. **case** 102:
271. printf("E");
272. **break**;
273. **case** 103:
274. printf("T");
275. **break**;
276. **case** 104:
277. printf("F");
278. **break**;
279. }
280. }
281. cout<<endl;
282. y=GOTO[tstack[top]][pro[x].left-100];
283. //cout<<pro[x].left<<y<<endl;
284. Push(pro[x].left,y);
285. //将规约产生式的左部压入栈，并将转移状态也压栈
286. }
287. **void** Acc() //accept
288. {
289. flag=0; //不再循环
290. cout<<"ACC!"<<endl;
291. }
292. **void** Err1()
293. {
294. //期待(或运算对象首字符，但出现运算符或者$
295. printf("error1\t\t\t缺少运算符，将num入栈.\n");
296. Push(7,5);
297. }
298. **void** Err2()
299. {
300. //括号不匹配，删掉右括号
301. printf("error2\t\t\t括号不匹配，请删除右括号.\n");
302. }
303. **void** Err3()
304. {
305. **int** pointer=pos;
306. **switch**(tstack[top])
307. {
308. **case** 1:
309. Push(3,6);
310. printf("error3\t\t\t缺少操作符，请添加操作符到栈里.");
311. **break**;
312. **case** 2:
313. **case** 12:
314. **case** 13:
315. Push(5,8);
316. printf("error3\t\t\t缺少操作符，请添加操作符到栈里.");
317. **break**;
318. **case** 11:
319. **if**(number[pointer]>=3 && number[pointer]<=6)
320. {
321. //期待运算符
322. Push(2,15);
323. printf("e3\t\t\t缺少右括号，请添加右括号到栈.");
324. }
325. **else** **if**(number[pointer]==7)
326. {
327. Push(3,6);
328. printf("error3\t\t\t缺少操作符，请添加操作符到栈里.");
329. }
330. **break**;
331. }
332. }
     1. 实验输入及输出
333. 请输入一个待分析串，以$结束
334. 8\*(8-6)/4$
335. S5      移进.
336. R9      规约    F -> num
337. R7      规约    T -> F
338. S8      移进.
339. S4      移进.
340. S5      移进.
341. R9      规约    F -> num
342. R7      规约    T -> F
343. R4      规约    E -> T
344. S7      移进.
345. S5      移进.
346. R9      规约    F -> num
347. R7      规约    T -> F
348. R3      规约    E -> E-T
349. S15     移进.
350. R8      规约    F -> (S)
351. R5      规约    T -> T\*F
352. S9      移进.
353. S5      移进.
354. R9      规约    F -> num
355. R6      规约    T -> T/F
356. R4      规约    E -> T
357. ACC!
358. Press any key to **continue** . . .
359. 方法4
     1. 实验分析

通过定义翻译规则，YACC 使用 LALR 文法进行规约。通过 LEX 将读取到的字符串分析为符号串，再对符号串按照定义的表达式进行规约，判断输入的字符串是否符合表达式定义。

* 1. 相关文法及分析表等

同方法3

* 1. 源代码

1. YACC 程序：
2. %{
3. #include <ctype.h> #include <stdio.h> #include <stdlib.h> #include <string.h>
4. #define YYSTYPE double /\*double type for YACC stack\*/
6. **int** yylex(**void**);
7. **void** yyerror(**char** \* s); **int** yywrap(**void**);
9. %}
11. %token NUM
12. %token ID
13. %left '+' '-'
14. %left '\*' '/'
15. %right UMINUS
16. %%

19. lines : lines expr '\n' {printf("Result: %f\n",$2);}
20. |lines '\n'
21. |   /\* empty \*/
22. |error '\n' {yyerror("Retry please."); yyerrok;}
23. |'-' expr %prec UMINUS {$$ = -$2;printf("E->-E\n");}
24. ;
25. expr : expr '+' term {$$ = $1 + $3;printf("E->E+T\n");}
26. |expr '-' term {$$ = $1 - $3;printf("E->E-T\n");}
27. |term {printf("E->T\n");}
28. ;
29. term : term '\*' form {$$ = $1 \* $3;printf("T->T\*F\n");}
30. |term '/' form {$$ = $1 / $3;printf("T->T/F\n");}
31. |form {printf("T->F\n");}
32. ;
34. form : ID {printf("F->id\n");}
35. |'(' expr ')' {$$ = $2;printf("F->(E)\n");}
36. |NUM {printf("F->num\n");}
37. ;
39. %%
40. #include "lex.yy.c" void yyerror(char \* s){
41. printf("%s\n",s);
42. }
44. **int** main (){
45. **return** yyparse();
46. }
48. LEX 程序：
49. %option noyywrap
50. %{
51. #include "first.tab.h" char\* id[100];
52. **double** num[100]; **int** count=0;
53. **extern** **double** yylval;
55. **double** install\_id(); **int** yywarp (**void**);
57. %}
58. space   [' ']\*
59. wrap    [\n]
60. letter  [A-Za-z]
61. digit   [0-9]
62. id  {letter}({letter}|{digit})\*
63. num {digit}+(\.{digit}+)?

66. %%
67. {id}    {yylval=install\_id();**return** (ID);}
68. {num}   {yylval=atof(yytext);**return** (NUM);}
69. {space} {printf(" ");}
70. {wrap}  {**return** '\n';}
72. .   {**return** yytext[0];}
73. %%
75. **double** install\_id(){ **int** i=0;
76. **for**(i=0;i<count;i++){ **if**(strcmp(yytext,id[i])==0){
77. **return** num[i];
78. }
79. }
80. printf("Found an unassigned id:\n%s = ",yytext); id[count]=(**char**\*)malloc(**sizeof**(**char**)\*yyleng); strcpy(id[count],yytext);
81. scanf("%lf", &num[count]); count++;
82. **return** num[count-1];
83. }
85. **int** yywarp (**void**){ **return** 1;
86. }
    1. 实验输入及输出
87. 输入：(5+6)/(9-2)
88. 输出：F->num T->F
89. E->T
90. F->num T->F
91. E->E+T F->(E) T->F
92. F->num T->F
93. E->T
94. F->num T->F
95. E->E-T F->(E) T->T/F E->T
96. Result: 1.571429
97. 心得体会

在本次语法分析的C++实现中，我利用编写程序更加理清了LL(1)文法和LR文法在分析过程中的差异，对于课堂上的理论知识有了更深刻的理解和认识，可谓是实践出真知。