

# 编译原理 语法分析 实验报告

姓名：陈俊卉    班级：2020219111    学号：2020212256

## 编译原理 语法分析 实验报告

- 一、实验内容
- 二、实验环境
- 三、方法一：递归
  - 1、原理分析
  - 2、前置运算
    - ① 消除左递归，提取左公因子
    - ② 生成first集合与follow集合
    - ③ 实现简介
  - 3、代码展示
  - 4、测试结果
- 四、方法二：LL(1)分析
  - 1、原理分析
  - 2、前置运算
    - ① 改写文法（与方法一是一致的）
    - ② 生成first集合与follow集合
    - ③ LL(1)预测分析表（目的是为了验证生成的分析表是否正确）
    - ④ 实现简介
  - 3、代码展示
  - 4、测试结果
- 五、方法三：LR分析
  - 1、原理分析
  - 2、前置运算
    - ① 增广文法
    - ② 生成first集合与follow集合
    - ③ 识别所有活前缀的DFA，构造项目及规范族（略，可以在LR分析表中体现）
    - ④ LR分析表
    - ⑤ 实现简介
  - 3、代码展示
  - 4、测试结果
- 六、总结与展望

## 一、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。
  - $E \rightarrow E+T \mid E-T \mid T$
  - $T \rightarrow T * F \mid T / F \mid F$
  - $F \rightarrow (E) \mid \text{num}$
- 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。
    - 编程实现算法4.2，为给定文法自动构造预测分析表。
    - 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。

- 构造识别 该文法所有活前缀的DFA。
- 构造该文法的LR分析表。
- 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。

- 本次实验笔者实现了前三个方法。
- 注意一：事实上，在调用语法分析时，语句/代码会事先预处理好，整理好词法传入语法分析代码内。所以所有的数字（包括整数、小数等）都会转化为一个文法符号表示。本次实验我们在输入时使用单个的数字（0~9）代表这个文法符号，以笼统地代表所有处理完毕后的所有数字类型。
  - 也就是说，我们输入的语句中，只需要输入单位数字，其代表的是在词法分析中解析出来的代表数字的文法符号。
- 注意二：当前版本的C++中，当想要将string直接当作char \*使用时会发生强制类型转换，这会引发编译警告

```
1 warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
```

可以用c\_str()或data()解决该问题。但为了代码的简洁起见，我们仍保留这样的方式。

## 二、实验环境

- windows10系统
- Visual Studio Code
- C++ 17

## 三、方法一：递归

### 1、原理分析

为了通过递归调用程序实现自顶向下的分析，我们需要将产生式进行变形，使之没有左递归，提取左公因子，并且生成非终结符的first集合和follow集合。随后根据最左推导的方式完成语句的分析。

### 2、前置运算

#### ① 消除左递归，提取左公因子

对于形如

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$$

使用如下公式：

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_m A'$$

$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \varepsilon$$

可以得到消除左递归后的文法如下（e为空符号，n为num）：

$$\begin{aligned} E &\rightarrow TP \\ P &\rightarrow +TP \\ P &\rightarrow -TP \\ P &\rightarrow e \\ T &\rightarrow FQ \\ Q &\rightarrow *FQ \\ Q &\rightarrow /FQ \\ Q &\rightarrow e \\ F &\rightarrow (E) \\ F &\rightarrow n \end{aligned}$$

## ② 生成first集合与follow集合

生成的first集合与follow集合如下：

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

## ③ 实现简介

- 构造每一个产生式左方非终结符的函数，按照产生式右方的符号依次执行代码。利用判断区分同一个左方非终结符的不同产生式。

## 3、代码展示

```

1  #include<iostream>
2  #include<string.h>
3  #include<string>
4  #include<stdlib.h>
5  #include<fstream>
6  #include <iomanip>
7  #include <unordered_map>
8  using namespace std;
9
10 // 方法一 本质：从左到右递归遍历求解
11
12 // 设置一个标记，如果有任何一个地方与产生式不符则置0
13 int flag = 1;
14
15 // 接收字符串
16 char str[100000];
17 // 指针
18 int p = 0;
19
20 void handle_F();

```

```

21 void handle_Q();
22 void handle_T();
23 void handle_P();
24 void handle_E();
25
26
27
28
29 void input_str(){
30     cout << "plz input string for analysis:" << endl;
31     int q = 0;
32     char c;
33     while (1) {
34         if ((c = getchar()) == '\n'){
35             str[q++] = '$';
36             str[q++] = '\0';
37             break;
38         }
39         str[q++] = c;
40     }
41 }
42
43
44 // 产生式
45 char *production[] = {
46     "E->TP",
47     "P->+TP",
48     "P->-TP",
49     "P->e",
50     "T->FQ",
51     "Q->*FQ",
52     "Q->/FQ",
53     "Q->e",
54     "F->(E)",
55     "F->n",
56 };
57
58 void print_str(){
59     cout << "left char: ";
60     for(int i = p; ;){
61         if(str[i] != '$'){
62             cout << str[i];
63             i++;
64         }
65         else
66             break;
67     }
68     cout << endl;
69     cout << endl;
70 }
71
72
73 void handle_F(){
74     if (str[p] == '(') {
75         cout << "F->(E)" << endl;
76         print_str();
77         p++;
78         handle_E();

```

```

79         if (str[p] == ')') {
80             p++;
81         }
82
83         else {
84             flag = 0;
85         }
86     }
87
88     else if (str[p] >= '0' && str[p] <= '9') {
89         cout << "F->num" << endl;
90         print_str();
91         p++;
92     }
93
94     else {
95         flag = 0;
96     }
97 }
98
99 void handle_Q(){
100     if (str[p] == '*') {
101         cout << "Q->*FQ" << endl;
102         print_str();
103         p++;
104         handle_F();
105         handle_Q();
106     }
107
108     else if (str[p] == '/') {
109         cout << "Q->/FQ" << endl;
110         print_str();
111         p++;
112         handle_F();
113         handle_Q();
114     }
115     else {
116         cout << "Q->e" << endl;
117         print_str();
118     }
119 }
120
121
122 void handle_T(){
123     cout << "T->FQ" << endl;
124     print_str();
125     handle_F();
126     handle_Q();
127 }
128
129 void handle_P(){
130     if (str[p] == '+') {
131         cout << "P->+TP" << endl;
132         print_str();
133         p++;
134         handle_T();
135         handle_P();
136     }

```

```

137
138     else if (str[p] == '-') {
139         cout << "P->-TP" << endl;
140         print_str();
141         p++;
142         handle_T();
143         handle_P();
144     }
145     else {
146         cout << "P->e" << endl;
147         print_str();
148     }
149 }
150
151
152 void handle_E(){
153     cout << "E->TP" << endl;
154     print_str();
155     handle_T();
156     handle_P();
157 }
158
159 void is_correct(){
160     if (flag == 1 && str[p] == '$')
161         cout << "the sentence is correct for the grammar." << endl;
162     else
163         cout << "the sentence is wrong for the grammar." << endl;
164 }
165
166 int main(){
167     input_str();
168     cout << endl;
169     cout << "start analysing:" << endl;
170     handle_E();
171     is_correct();
172 }

```

## 4、测试结果

- 约定：e为空（下文也同样）

```

1  plz input string for analysis:
2  5-3*(1+2)
3
4  start analysing:
5  E->TP
6  left char: 5-3*(1+2)
7
8  T->FQ
9  left char: 5-3*(1+2)
10
11 F->num
12 left char: 5-3*(1+2)
13
14 Q->e

```

```
15 left char: -3*(1+2)
16
17 P->-TP
18 left char: -3*(1+2)
19
20 T->FQ
21 left char: 3*(1+2)
22
23 F->num
24 left char: 3*(1+2)
25
26 Q->*FQ
27 left char: *(1+2)
28
29 F->(E)
30 left char: (1+2)
31
32 E->TP
33 left char: 1+2)
34
35 T->FQ
36 left char: 1+2)
37
38 F->num
39 left char: 1+2)
40
41 Q->e
42 left char: +2)
43
44 P->+TP
45 left char: +2)
46
47 T->FQ
48 left char: 2)
49
50 F->num
51 left char: 2)
52
53 Q->e
54 left char: )
55
56 P->e
57 left char: )
58
59 Q->e
60 left char:
61
62 P->e
63 left char:
64
65 the sentence is correct for the grammar.
```

```
1 plz input string for analysis:
2 5*(7-1)/3
3
```

```
4 start analysing:
5 E->TP
6 left char: 5*(7-1)/3
7
8 T->FQ
9 left char: 5*(7-1)/3
10
11 F->num
12 left char: 5*(7-1)/3
13
14 Q->*FQ
15 left char: *(7-1)/3
16
17 F->(E)
18 left char: (7-1)/3
19
20 E->TP
21 left char: 7-1)/3
22
23 T->FQ
24 left char: 7-1)/3
25
26 F->num
27 left char: 7-1)/3
28
29 Q->e
30 left char: -1)/3
31
32 P->-TP
33 left char: -1)/3
34
35 T->FQ
36 left char: 1)/3
37
38 F->num
39 left char: 1)/3
40
41 Q->e
42 left char: )/3
43
44 P->e
45 left char: )/3
46
47 Q->/FQ
48 left char: /3
49
50 F->num
51 left char: 3
52
53 Q->e
54 left char:
55
56 P->e
57 left char:
58
59 the sentence is correct for the grammar.
```



经过检验，输出完全正确。

## 四、方法二：LL(1)分析

### 1、原理分析

由于我们需要编写LL(1)语法分析程序，所以我们首先需要将给出的文法变为LL(1)文法，再写出相应的first集合和follow集合，以使用算法4.2自动构造预测分析表；随后根据预测分析表构造LL(1)预测分析程序。**LL(1)同样是自上而下的分析方法，而且是有效的无回溯的。**无回溯的原因是构造并使用了分析表。

### 2、前置运算

#### ① 改写文法（与方法一是一致的）

$$\begin{aligned} E &\rightarrow TP \\ P &\rightarrow +TP \\ P &\rightarrow -TP \\ P &\rightarrow e \\ T &\rightarrow FQ \\ Q &\rightarrow *FQ \\ Q &\rightarrow /FQ \\ Q &\rightarrow e \\ F &\rightarrow (E) \\ F &\rightarrow n \end{aligned}$$

#### ② 生成first集合与follow集合

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

#### ③ LL(1)预测分析表（目的是为了验证生成的分析表是否正确）

	+	-	*	/	(	)	num	\$
E					$E \rightarrow TP$		$E \rightarrow TP$	
P	$P \rightarrow +TP$	$P \rightarrow -TP$				$P \rightarrow e$		$P \rightarrow e$
T					$T \rightarrow FQ$		$T \rightarrow FQ$	
Q	$Q \rightarrow e$	$Q \rightarrow e$	$Q \rightarrow *FQ$	$Q \rightarrow /FQ$		$Q \rightarrow e$		$Q \rightarrow e$
F					$F \rightarrow (E)$		$F \rightarrow \text{num}$	

#### ④ 实现简介

- 使用table数组记录预测分析表。

```

109 // table
110 string table[100][100][100];
111 /*
112     表格式:
113     +   -   *   /   (   )   n   $
114     E
115     T
116     F
117     P
118     Q
119 */

```

- 使用p数组记录表中某处是否存在产生式。

```

97 // 三维数组最后一维的指针
98 int p[6][9] = {
99     {0,0,0,0,0,0,0,0,0},
100     {0,-1,-1,-1,-1,-1,-1,-1,-1},
101     {0,-1,-1,-1,-1,-1,-1,-1,-1},
102     {0,-1,-1,-1,-1,-1,-1,-1,-1},
103     {0,-1,-1,-1,-1,-1,-1,-1,-1},
104     {0,-1,-1,-1,-1,-1,-1,-1,-1},
105 };
106

```

- 代码前半部分使用switch语句，提高可读性；后半部分输出使用unordered\_map，不再重复case过程，缩减代码冗余，增加效率。

#### 算法 4.2 预测分析表的构造方法

输入：文法  $G$ 。

输出：文法  $G$  的预测分析表  $M$ 。

方法：

```

for (文法  $G$  的每一个产生式  $A \rightarrow \alpha$ ) {
    for (每个终结符号  $a \in \text{FIRST}(\alpha)$ ) 把  $A \rightarrow \alpha$  放入表项  $M[A, a]$  中;
    if ( $\epsilon \in \text{FIRST}(\alpha)$ )
        for (每个  $b \in \text{FOLLOW}(A)$ ) 把  $A \rightarrow \alpha$  放入表项  $M[A, b]$  中;
};
for (所有无定义的表项  $M[A, a]$ ) 标上错误标志。

```

**算法 4.1** 非递归预测分析方法。

输入：输入符号串  $\omega$ , 文法  $G$  的一张预测分析表  $M$ 。

88

第 4 章

输出：若  $\omega \in L(G)$ , 则输出  $\omega$  的最左推导, 否则报告错误。

方法：

首先, 初始化, 即将  $\$$  压入栈底, 将文法开始符号  $S$  压入栈顶; 将  $\omega \$$  放入输入缓冲区中, 并置向前指针  $ip$  指向  $\omega \$$  的第一个符号。

然后, 预测分析控制程序根据分析表  $M$  对输入符号串  $\omega$  作出自顶向下的分析, 过程如下：

```
do{
    令  $x$  是栈顶文法符号,  $a$  是  $ip$  所指向的输入符号;
    if(  $x$  是终结符号或  $\$$  )
        if(  $x == a$  ) {从栈顶弹出  $x$ ;  $ip$  前移一个位置;};
        else error();
    else /*  $x$  是非终结符号 */
        if(  $M[x, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$  ) {
            从栈顶弹出  $x$ ;
            依次把  $Y_k, Y_{k-1}, \dots, Y_2, Y_1$  压入栈; /*  $Y_1$  在栈顶 */
            输出产生式  $X \rightarrow Y_1 Y_2 \cdots Y_k$ ;
        };
        else error();
} while (  $x != \$$  ) /* 栈非空, 分析继续 */
```

### 3、代码展示

- 注：其实代码本身若都使用 `unordered_map`, 不需要这么多行。前面使用 `switch` 是因为这样可读性更高。可以使用映射表映射来减轻代码冗余, 但可读性也相应地降低。

```
1  #include<iostream>
2  #include<string.h>
3  #include<string>
4  #include<stdlib.h>
5  #include<fstream>
6  #include <iomanip>
7  #include <unordered_map>
8  using namespace std;
9
10
11 // 产生式 LL(1)
12 char *production[] = {
13     "E->TP",
14     "P->+TP",
15     "P->-TP",
```

```

16     "P->e",
17     "T->FQ",
18     "Q->*FQ",
19     "Q->/FQ",
20     "Q->e",
21     "F->(E)",
22     "F->n",
23 };
24
25 // 终结符
26 char *terminator[] = {
27     "+",
28     "-",
29     "*",
30     "/",
31     "(",
32     ")",
33     "n",
34 };
35
36 // F的first集合和follow集合
37 char *first_F[] = {
38     "(",
39     "n",
40 };
41 char *follow_F[] = {
42     "*",
43     "/",
44     "+",
45     "-",
46     "$",
47     ")",
48 };
49
50 // T的first集合和follow集合
51 char *first_T[] = {
52     "(",
53     "n",
54 };
55 char *follow_T[] = {
56     "+",
57     "-",
58     "$",
59     ")",
60 };
61
62 // E的first集合和follow集合
63 char *first_E[] = {
64     "(",
65     "n",
66 };
67 char *follow_E[] = {
68     ")",
69     "$",
70 };
71
72 // P的first集合和follow集合
73 char *first_P[] = {

```

```

74     "+",
75     "-",
76     "e",
77 };
78 char *follow_P[] = {
79     ")",
80     "$",
81 };
82
83 // Q的first集合和follow集合
84 char *first_Q[] = {
85     "*",
86     "/",
87     "e",
88 };
89 char *follow_Q[] = {
90     "+",
91     "-",
92     ")",
93     "$",
94 };
95
96
97 // 三维数组最后一维的指针
98 int p[6][9] = {
99     {0,0,0,0,0,0,0,0,0},
100    {0,-1,-1,-1,-1,-1,-1,-1,-1},
101    {0,-1,-1,-1,-1,-1,-1,-1,-1},
102    {0,-1,-1,-1,-1,-1,-1,-1,-1},
103    {0,-1,-1,-1,-1,-1,-1,-1,-1},
104    {0,-1,-1,-1,-1,-1,-1,-1,-1},
105 };
106
107
108
109 // table
110 string table[100][100][100];
111 /*
112     表格式:
113         +   -   *   /   (   )   n   $
114     E
115     T
116     F
117     P
118     Q
119 */
120
121 // 实现char *的substr
122 char *sub_str(char *s,int n,int len)
123 {
124     static char p[20]; //或者设置为静态变量
125     int i, j = 0;
126     while (n--) {
127         s++; //确定字符串的首位置
128     }
129     for(i = n; i < n + len; i++){
130         p[j++] = *s;
131         s++;

```

```

132     }
133     return p;
134 }
135
136 void set_table(){
137     table[0][1][0] = "+";
138     table[0][2][0] = "-";
139     table[0][3][0] = "*";
140     table[0][4][0] = "/";
141     table[0][5][0] = "(";
142     table[0][6][0] = ")";
143     table[0][7][0] = "n";
144     table[0][8][0] = "$";
145
146     table[1][0][0] = "E";
147     table[2][0][0] = "T";
148     table[3][0][0] = "F";
149     table[4][0][0] = "P";
150     table[5][0][0] = "Q";
151
152 }
153
154 void print_table(){
155     for (int i = 1; i <= 5; i++){
156         for (int j = 1; j <= 8; j++){
157
158             char left;
159             char right;
160             switch (i)
161             {
162                 case 1:
163                 {
164                     left = 'E';
165                     break;
166                 }
167                 case 2:
168                 {
169                     left = 'T';
170                     break;
171                 }
172                 case 3:
173                 {
174                     left = 'F';
175                     break;
176                 }
177                 case 4:
178                 {
179                     left = 'P';
180                     break;
181                 }
182                 case 5:
183                 {
184                     left = 'Q';
185                     break;
186                 }
187             }
188             switch (j)
189             {

```

```

190         case 1:
191         {
192             right = '+';
193             break;
194         }
195         case 2:
196         {
197             right = '-';
198             break;
199         }
200         case 3:
201         {
202             right = '*';
203             break;
204         }
205         case 4:
206         {
207             right = '/';
208             break;
209         }
210
211         case 5:
212         {
213             right = '(';
214             break;
215         }
216         case 6:
217         {
218             right = ')';
219             break;
220         }
221         case 7:
222         {
223             right = 'n';
224             break;
225         }
226         case 8:
227         {
228             right = '$';
229             break;
230         }
231     }
232     cout << "(" << left << "," << right << "): ";
233     // if (p[i][j] == -1) cout << "error";
234     for (int k = 0; k <= p[i][j]; k++) {
235         cout << table[i][j][k] << " ";
236     }
237     cout << endl;
238 }
239 }
240 }
241
242
243 void predict_analysis_table(){
244     for (int i = 0; i <= 9; i++){
245         char *pro = production[i];
246         // first集合
247         char *first_set[100];

```

```
248     int top_first_set = 0;
249
250     // 由于产生式格式一样，直接将指针往右移动三位得到产生式右边的字符串
251     pro += 3;
252
253     // 如果第一个为终结符，那么first( $\alpha$ )就是该终结符
254
255     // 构造 $\alpha$ 的first集合
256     switch (pro[0])
257     {
258         // 如果是终结符
259         case '+':
260         case '-':
261         case '*':
262         case '/':
263         case '(':
264         case ')':
265         case '\n':
266         case 'e':
267         {
268             first_set[top_first_set++] = sub_str(pro, 0, 1);
269             break;
270         }
271
272         // 如果是非终结符
273         case 'F':
274         {
275             for (int i = 0; i < 2; i++){
276                 first_set[top_first_set++] = first_F[i];
277             }
278             break;
279         }
280
281         case 'T':
282         {
283             for (int i = 0; i < 2; i++){
284                 first_set[top_first_set++] = first_T[i];
285             }
286             break;
287         }
288
289         case 'E':
290         {
291             for (int i = 0; i < 2; i++){
292                 first_set[top_first_set++] = first_E[i];
293             }
294             break;
295         }
296
297         case 'P':
298         {
299             for (int i = 0; i < 3; i++){
300                 first_set[top_first_set++] = first_P[i];
301             }
302             break;
303         }
304     }
305
```



```

306         case 'Q':
307         {
308             for (int i = 0; i < 3; i++){
309                 first_set[top_first_set++] = first_Q[i];
310             }
311             break;
312         }
313     }
314
315
316
317
318
319     // 指针倒回去 构造第一种情况
320     pro -= 3;
321     for (int i = 0; i < top_first_set; i++)
322     {
323         // cout << pro << endl;
324         switch (pro[0])
325         {
326             case 'E':
327             {
328                 switch (first_set[i][0])
329                 {
330                     case '+':
331                     {
332                         table[1][1][++p[1][1]] = pro;
333                         break;
334                     }
335
336                     case '-':
337                     {
338                         table[1][2][++p[1][2]] = pro;
339                         break;
340                     }
341
342                     case '*':
343                     {
344                         table[1][3][++p[1][3]] = pro;
345                         break;
346                     }
347
348                     case '/':
349                     {
350                         table[1][4][++p[1][4]] = pro;
351                         break;
352                     }
353
354                     case '(':
355                     {
356                         table[1][5][++p[1][5]] = pro;
357                         break;
358                     }
359
360                     case ')':
361                     {
362                         table[1][6][++p[1][6]] = pro;
363                         break;

```

```

364         }
365
366         case 'n':
367         {
368             table[1][7][++p[1][7]] = pro;
369             break;
370         }
371
372         case '$':
373         {
374             table[1][8][++p[1][8]] = pro;
375             break;
376         }
377     }
378     break;
379 }
380
381 case 'T':
382 {
383     switch (first_set[i][0])
384     {
385         case '+':
386         {
387             table[2][1][++p[2][1]] = pro;
388             break;
389         }
390
391         case '-':
392         {
393             table[2][2][++p[2][2]] = pro;
394             break;
395         }
396
397         case '*':
398         {
399             table[2][3][++p[2][3]] = pro;
400             break;
401         }
402
403         case '/':
404         {
405             table[2][4][++p[2][4]] = pro;
406             break;
407         }
408
409         case '(':
410         {
411             table[2][5][++p[2][5]] = pro;
412             break;
413         }
414
415         case ')':
416         {
417             table[2][6][++p[2][6]] = pro;
418             break;
419         }
420
421         case 'n':

```

```
422         {
423             table[2][7][++p[2][7]] = pro;
424             break;
425         }
426
427         case '$':
428         {
429             table[2][8][++p[2][8]] = pro;
430             break;
431         }
432     }
433     break;
434 }
435
436 case 'F':
437 {
438     switch (first_set[i][0])
439     {
440         case '+':
441         {
442             table[3][1][++p[3][1]] = pro;
443             break;
444         }
445
446         case '-':
447         {
448             table[3][2][++p[3][2]] = pro;
449             break;
450         }
451
452         case '*':
453         {
454             table[3][3][++p[3][3]] = pro;
455             break;
456         }
457
458         case '/':
459         {
460             table[3][4][++p[3][4]] = pro;
461             break;
462         }
463
464         case '(':
465         {
466             table[3][5][++p[3][5]] = pro;
467             break;
468         }
469
470         case ')':
471         {
472             table[3][6][++p[3][6]] = pro;
473             break;
474         }
475
476         case 'n':
477         {
478             table[3][7][++p[3][7]] = pro;
479             break;
```

```

480     }
481
482     case '$':
483     {
484         table[3][8][++p[3][8]] = pro;
485         break;
486     }
487 }
488 break;
489 }
490
491 case 'P':
492 {
493     switch (first_set[i][0])
494     {
495         case '+':
496         {
497             table[4][1][++p[4][1]] = pro;
498             break;
499         }
500
501         case '-':
502         {
503             table[4][2][++p[4][2]] = pro;
504             break;
505         }
506
507         case '*':
508         {
509             table[4][3][++p[4][3]] = pro;
510             break;
511         }
512
513         case '/':
514         {
515             table[4][4][++p[4][4]] = pro;
516             break;
517         }
518
519         case '(':
520         {
521             table[4][5][++p[4][5]] = pro;
522             break;
523         }
524
525         case ')':
526         {
527             table[4][6][++p[4][6]] = pro;
528             break;
529         }
530
531         case 'n':
532         {
533             table[4][7][++p[4][7]] = pro;
534             break;
535         }
536
537         case '$':

```

```
538         {
539             table[4][8][++p[4][8]] = pro;
540             break;
541         }
542     }
543     break;
544 }
545
546 case 'Q':
547 {
548     switch (first_set[i][0])
549     {
550         case '+':
551         {
552             table[5][1][++p[5][1]] = pro;
553             break;
554         }
555
556         case '-':
557         {
558             table[5][2][++p[5][2]] = pro;
559             break;
560         }
561
562         case '*':
563         {
564             table[5][3][++p[5][3]] = pro;
565             break;
566         }
567
568         case '/':
569         {
570             table[5][4][++p[5][4]] = pro;
571             break;
572         }
573
574         case '(':
575         {
576             table[5][5][++p[5][5]] = pro;
577             break;
578         }
579
580         case ')':
581         {
582             table[5][6][++p[5][6]] = pro;
583             break;
584         }
585
586         case 'n':
587         {
588             table[5][7][++p[5][7]] = pro;
589             break;
590         }
591
592         case '$':
593         {
594             table[5][8][++p[5][8]] = pro;
595             break;
```

```

596         }
597     }
598     break;
599 }
600
601 }
602 }
603
604
605 // 构造第二种情况
606 // 判断是否存在  $e \in \text{FIRST}(\alpha)$ 
607 bool empty_flag = 0;
608 for (int i = 0; i < top_first_set; i++){
609     // cout << first_set[i] << ' ';
610     if ('e' == first_set[i][0])
611     {
612         empty_flag = 1;
613         break;
614     }
615 }
616 // cout << endl;
617
618 // cout << "empty_flag:" << empty_flag << endl;
619
620 if (empty_flag == 1){
621     // 选择pro[0]
622     // cout << pro << endl;;
623     switch (pro[0])
624     {
625         case 'E':
626         {
627             for (int i = 0; i <= 1; i++){
628                 switch (follow_E[i][0])
629                 {
630                     case '+':
631                     {
632                         table[1][1][++p[1][1]] = pro;
633                         break;
634                     }
635                     case '-':
636                     {
637                         table[1][2][++p[1][2]] = pro;
638                         break;
639                     }
640                     case '*':
641                     {
642                         table[1][3][++p[1][3]] = pro;
643                         break;
644                     }
645                     case '/':
646                     {
647                         table[1][4][++p[1][4]] = pro;
648                         break;
649                     }
650                     case '(':
651                     {
652                         table[1][5][++p[1][5]] = pro;
653                         break;

```

```

654         }
655         case ')':
656         {
657             table[1][6][++p[1][6]] = pro;
658             break;
659         }
660         case 'n':
661         {
662             table[1][7][++p[1][7]] = pro;
663             break;
664         }
665         case '$':
666         {
667             table[1][8][++p[1][8]] = pro;
668             break;
669         }
670         break;
671     }
672 }
673 break;
674 }
675
676 case 'T':
677 {
678     for (int i = 0; i <= 3; i++){
679         switch (follow_T[i][0])
680         {
681             case '+':
682             {
683                 table[2][1][++p[2][1]] = pro;
684                 break;
685             }
686             case '-':
687             {
688                 table[2][2][++p[2][2]] = pro;
689                 break;
690             }
691             case '*':
692             {
693                 table[2][3][++p[2][3]] = pro;
694                 break;
695             }
696             case '/':
697             {
698                 table[2][4][++p[2][4]] = pro;
699                 break;
700             }
701             case '(':
702             {
703                 table[2][5][++p[2][5]] = pro;
704                 break;
705             }
706             case ')':
707             {
708                 table[2][6][++p[2][6]] = pro;
709                 break;
710             }
711             case 'n':

```

```

712         {
713             table[2][7][++p[2][7]] = pro;
714             break;
715         }
716         case '$':
717         {
718             table[2][8][++p[2][8]] = pro;
719             break;
720         }
721         break;
722     }
723 }
724 break;
725 }
726
727 case 'F':
728 {
729     for (int i = 0; i <= 5; i++){
730         switch (follow_F[i][0])
731         {
732             case '+':
733             {
734                 table[3][1][++p[3][1]] = pro;
735                 break;
736             }
737             case '-':
738             {
739                 table[3][2][++p[3][2]] = pro;
740                 break;
741             }
742             case '*':
743             {
744                 table[3][3][++p[3][3]] = pro;
745                 break;
746             }
747             case '/':
748             {
749                 table[3][4][++p[3][4]] = pro;
750                 break;
751             }
752             case '(':
753             {
754                 table[3][5][++p[3][5]] = pro;
755                 break;
756             }
757             case ')':
758             {
759                 table[3][6][++p[3][6]] = pro;
760                 break;
761             }
762             case 'n':
763             {
764                 table[3][7][++p[3][7]] = pro;
765                 break;
766             }
767             case '$':
768             {
769                 table[3][8][++p[3][8]] = pro;

```



```

770             break;
771         }
772         break;
773     }
774 }
775 break;
776 }
777
778 case 'P':
779 {
780     for (int i = 0; i <= 1; i++){
781         switch (follow_P[i][0])
782         {
783             case '+':
784             {
785                 table[4][1][++p[4][1]] = pro;
786                 break;
787             }
788             case '-':
789             {
790                 table[4][2][++p[4][2]] = pro;
791                 break;
792             }
793             case '*':
794             {
795                 table[4][3][++p[4][3]] = pro;
796                 break;
797             }
798             case '/':
799             {
800                 table[4][4][++p[4][4]] = pro;
801                 break;
802             }
803             case '(':
804             {
805                 table[4][5][++p[4][5]] = pro;
806                 break;
807             }
808             case ')':
809             {
810                 table[4][6][++p[4][6]] = pro;
811                 break;
812             }
813             case 'n':
814             {
815                 table[4][7][++p[4][7]] = pro;
816                 break;
817             }
818             case '$':
819             {
820                 table[4][8][++p[4][8]] = pro;
821                 break;
822             }
823             break;
824         }
825     }
826     break;
827 }

```

```

828
829     case 'Q':
830     {
831         for (int i = 0; i <= 3; i++){
832             // cout << follow_Q[i][0];
833             switch (follow_Q[i][0])
834             {
835                 case '+':
836                 {
837                     table[5][1][++p[5][1]] = pro;
838                     break;
839                 }
840                 case '-':
841                 {
842                     table[5][2][++p[5][2]] = pro;
843                     break;
844                 }
845                 case '*':
846                 {
847                     table[5][3][++p[5][3]] = pro;
848                     break;
849                 }
850                 case '/':
851                 {
852                     table[5][4][++p[5][4]] = pro;
853                     break;
854                 }
855                 case '(':
856                 {
857                     table[5][5][++p[5][5]] = pro;
858                     break;
859                 }
860                 case ')':
861                 {
862                     table[5][6][++p[5][6]] = pro;
863                     break;
864                 }
865                 case 'n':
866                 {
867                     table[5][7][++p[5][7]] = pro;
868                     break;
869                 }
870                 case '$':
871                 {
872                     table[5][8][++p[5][8]] = pro;
873                     break;
874                 }
875                 break;
876             }
877             // cout << i;
878         }
879         break;
880     }
881
882
883
884     default:
885         break;

```

```

886     }
887 }
888
889
890
891 }
892 }
893
894 // 输入数组
895 char w[10000];
896 char* input_str(){
897     cout << "plz input string for analysis:" << endl;
898     int p = 0;
899     char c;
900     while (1) {
901         if ((c = getchar()) == '\n'){
902             w[p++] = '\0';
903             break;
904         }
905         w[p++] = c;
906     }
907     return w;
908 }
909 void LLI_prediction(){
910     cout << endl;
911     cout << "start analysis:" << endl;
912
913     // 建立一个与下标对应的map
914     unordered_map<char, int> map{
915         {'E', 1},
916         {'T', 2},
917         {'F', 3},
918         {'P', 4},
919         {'Q', 5},
920
921         {'+', 1},
922         {'-', 2},
923         {'*', 3},
924         {'/', 4},
925         {'(', 5},
926         {')', 6},
927         {'n', 7},
928         {'$', 8},
929     };
930
931     // 初始化
932
933     char stack[10000];
934     // stack_p指向stack中的栈顶元素
935     int stack_p = -1;
936     // 将终止符压入栈
937     stack[++stack_p] = '$';
938     // 将文法开始符号E压入栈
939     stack[++stack_p] = 'E';
940
941     char buffer[10000];
942     int buffer_p = 0;
943     // 将w$放入输入缓冲区

```

```

944 while (1) {
945     if(w[buffer_p] != '\0'){
946         buffer[buffer_p] = w[buffer_p];
947         buffer_p++;
948     }
949     else break;
950 }
951 buffer[buffer_p++] = '$';
952
953 // 设置向前指针指向w$的第一个符号
954 int forward = 0;
955
956 // 计数
957 int count = 1;
958
959 while (stack[stack_p] != '$') {
960     cout << "step" << count << ": " << endl;
961     count++;
962     // 判断是否为终止符的记号
963     int ter_flag = 0;
964     for (int i = 0; i <= 6; i++) {
965         if (stack[stack_p] == terminator[i][0] ||
966 (stack[stack_p]>='0' && stack[stack_p]<='9')){
967             ter_flag = 1;
968             break;
969         }
970     }
971
972     // log
973     cout << "stack: ";
974     for (int i = 0; i <= stack_p; i++) cout << stack[i];
975     cout << endl;
976
977     cout << "buffer: ";
978     for (int i = forward; i < buffer_p; i++) cout << buffer[i];
979     cout << endl;
980
981     if (stack[stack_p] == '$' || ter_flag == 1) {
982         if (stack[stack_p] == buffer[forward] || (stack[stack_p] ==
983 'n' && (buffer[forward] >= '0' && buffer[forward] <= '9')))) {
984             // 弹出x，其实就是stack_p--即可
985             stack_p--;
986             // forward
987             forward++;
988         }
989         else {
990             cout << "error!" << endl;
991             return ;
992         }
993     }
994     else {
995         // cout << "else" << endl;
996         char buffer_forward = buffer[forward];
997         // 如果是数字需要将其变为n
998         if (buffer[forward] >= '0' && buffer[forward] <= '9') {
999             buffer_forward = 'n';

```

```

1000     }
1001
1002     // 如果有产生式
1003     if (p[ map[stack[stack_p]] ][ map[buffer_forward] ] != -1) {
1004         // 坐标
1005         int x = map[stack[stack_p]];
1006         int y = map[buffer_forward];
1007         int z = p[ map[stack[stack_p]] ][ map[buffer_forward] ];
1008         // 栈顶弹出
1009         stack_p--;
1010         // 依次将Y_k Y_{k-1} ... Y_1压入栈
1011         string pro = table[x][y][z];
1012         for (int i = pro.length() - 1; i >= 0; i--){
1013
1014             if (pro[i] == '>') break;
1015             else if (pro[i] == 'e') continue;
1016             else stack[++stack_p] = pro[i];
1017         }
1018
1019         // 输出产生式
1020         cout << "using production: " << pro << endl;
1021
1022     }
1023
1024     else {
1025         cout << "error!" << endl;
1026         return ;
1027     }
1028 }
1029 cout << endl;
1030 }
1031 if (buffer[forward] == '$')
1032     cout << "successful!" << endl;
1033 else
1034     cout << "error!" << endl;
1035 }
1036
1037
1038
1039 int main() {
1040     set_table();
1041     cout << "analysis_table: " << endl;
1042     predict_analysis_table();
1043     print_table();
1044     cout << endl;
1045     input_str();
1046     LL1_prediction();
1047 }

```

## 4、测试结果

```

1 analysis_table:
2 (E,+):
3 (E,-):
4 (E,*):

```

```
5 (E,/):
6 (E,(): E->TP
7 (E,)):
8 (E,n): E->TP
9 (E,$):
10 (T,+):
11 (T,-):
12 (T,*):
13 (T,/):
14 (T,(): T->FQ
15 (T,)):
16 (T,n): T->FQ
17 (T,$):
18 (F,+):
19 (F,-):
20 (F,*):
21 (F,/):
22 (F,(): F->(E)
23 (F,)):
24 (F,n): F->n
25 (F,$):
26 (P,+): P->+TP
27 (P,-): P->-TP
28 (P,*):
29 (P,/):
30 (P,():
31 (P,)): P->e
32 (P,n):
33 (P,$): P->e
34 (Q,+): Q->e
35 (Q,-): Q->e
36 (Q,*): Q->*FQ
37 (Q,/): Q->/FQ
38 (Q,():
39 (Q,)): Q->e
40 (Q,n):
41 (Q,$): Q->e
42
43 plz input string for analysis:
44 5*(1+7)/2
45
46 start analysis:
47 step1:
48 stack: $E
49 buffer: 5*(1+7)/2$
50 using production: E->TP
51
52 step2:
53 stack: $PT
54 buffer: 5*(1+7)/2$
55 using production: T->FQ
56
57 step3:
58 stack: $PQF
59 buffer: 5*(1+7)/2$
60 using production: F->n
61
62 step4:
```

```
63 stack: $PQn
64 buffer: 5*(1+7)/2$
65
66 step5:
67 stack: $PQ
68 buffer: *(1+7)/2$
69 using production: Q->*FQ
70
71 step6:
72 stack: $PQF*
73 buffer: *(1+7)/2$
74
75 step7:
76 stack: $PQF
77 buffer: (1+7)/2$
78 using production: F->(E)
79
80 step8:
81 stack: $PQ)E(
82 buffer: (1+7)/2$
83
84 step9:
85 stack: $PQ)E
86 buffer: 1+7)/2$
87 using production: E->TP
88
89 step10:
90 stack: $PQ)PT
91 buffer: 1+7)/2$
92 using production: T->FQ
93
94 step11:
95 stack: $PQ)PQF
96 buffer: 1+7)/2$
97 using production: F->n
98
99 step12:
100 stack: $PQ)PQn
101 buffer: 1+7)/2$
102
103 step13:
104 stack: $PQ)PQ
105 buffer: +7)/2$
106 using production: Q->e
107
108 step14:
109 stack: $PQ)P
110 buffer: +7)/2$
111 using production: P->+TP
112
113 step15:
114 stack: $PQ)PT+
115 buffer: +7)/2$
116
117 step16:
118 stack: $PQ)PT
119 buffer: 7)/2$
120 using production: T->FQ
```

```
121
122 step17:
123 stack: $PQ)PQF
124 buffer: 7)/2$
125 using production: F->n
126
127 step18:
128 stack: $PQ)PQn
129 buffer: 7)/2$
130
131 step19:
132 stack: $PQ)PQ
133 buffer: )/2$
134 using production: Q->e
135
136 step20:
137 stack: $PQ)P
138 buffer: )/2$
139 using production: P->e
140
141 step21:
142 stack: $PQ)
143 buffer: )/2$
144
145 step22:
146 stack: $PQ
147 buffer: /2$
148 using production: Q->/FQ
149
150 step23:
151 stack: $PQF/
152 buffer: /2$
153
154 step24:
155 stack: $PQF
156 buffer: 2$
157 using production: F->n
158
159 step25:
160 stack: $PQn
161 buffer: 2$
162
163 step26:
164 stack: $PQ
165 buffer: $
166 using production: Q->e
167
168 step27:
169 stack: $P
170 buffer: $
171 using production: P->e
172
173 successful!
```

经过对照，分析表生成正确，且分析过程也正确。



# 五、方法三：LR分析

## 1、原理分析

LR分析实际上是从下而上的分析，是最右推导的逆过程，也就是最左规约。为了分析器只有一个接受状态（LR分析以归约到初始符号为接受），我们需要对文法进行增广。然后求first集与follow集，构造识别所有活前缀的DFA，构造项目集规范族，并以此构造LR分析表，最后使用LR分析表完成算法的实现。

## 2、前置运算

### ① 增广文法

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + T \\ E &\rightarrow E - T \\ E &\rightarrow T \\ T &\rightarrow T * F \\ T &\rightarrow T / F \\ T &\rightarrow F \\ F &\rightarrow (E) \\ F &\rightarrow n \end{aligned}$$

### ② 生成first集合与follow集合

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

### ③ 识别所有活前缀的DFA，构造项目及规范族（略，可以在LR分析表中体现）

### ④ LR分析表

状		A	C	T	I	O	N		G	O	T	O
态	(	)	+	-	*	/	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				

### ⑤ 实现简介

- 无需画出表格，所以全程使用unordered\_map，目的是使用符号映射到下标。

```

26 // 文法分析表
27 // 建立map
28 unordered_map<char, int> map{
29     {'(', 0},
30     {')', 1},
31     {'+', 2},
32     {'-', 3},
33     {'*', 4},
34     {'/', 5},
35     {'n', 6},
36     {'$', 7},
37
38     {'S', 0},
39     {'E', 1},
40     {'T', 2},
41     {'F', 3},
42 };

```

- 使用action数组、goto数组记录表格内容：

```

47 // S为正数 R为负数 acc为100 -100为err
48 int ACTION[16][8] = {
49     {4,-100,-100,-100,-100,-100,5,-100},
50     {-100,-100,6,7,-100,-100,-100,100},
51     {-100,-4,-4,-4,8,9,-100,-4},
52     {-100,-7,-7,-7,-7,-7,-100,-7},
53     {4,-100,-100,-100,-100,-100,5,-100},
54     {-100,-9,-9,-9,-9,-9,-100,-9},
55     {4,-100,-100,-100,-100,-100,5,-100},
56     {4,-100,-100,-100,-100,-100,5,-100},
57     {4,-100,-100,-100,-100,-100,5,-100},
58     {4,-100,-100,-100,-100,-100,5,-100},
59     {-100,15,6,7,-100,-100,-100,-100},
60     {-100,-2,-2,-2,8,9,-100,-2},
61     {-100,-3,-3,-3,8,9,-100,-3},
62     {-100,-5,-5,-5,-5,-5,-100,-5},
63     {-100,-6,-6,-6,-6,-6,-100,-6},
64     {-100,-8,-8,-8,-8,-8,-100,-8},
65 };
66
67 // -100为err
68 int GOTO[16][4] = {
69     {-100,1,2,3},
70     {-100,-100,-100,-100},
71     {-100,-100,-100,-100},
72     {-100,-100,-100,-100},
73     {-100,10,2,3},
74     {-100,-100,-100,-100},
75     {-100,-100,11,3},
76     {-100,-100,12,3},
77     {-100,-100,-100,13},
78     {-100,-100,-100,14},
79     {-100,-100,-100,-100},
80     {-100,-100,-100,-100},
81     {-100,-100,-100,-100},
82     {-100,-100,-100,-100},
83     {-100,-100,-100,-100},
84     {-100,-100,-100,-100},
85 };

```

#### 算法 4.3 LR 分析程序。

输入：文法  $G$  的一张分析表和一个输入符号串  $\omega$ 。

输出：若  $\omega \in L(G)$ ，得到  $\omega$  的自底向上的分析，否则报错。

方法：

首先初始化，将初始状态  $S_0$  入栈，将  $\omega \$$  存入输入缓冲区中；并置  $ip$  指向  $\omega \$$  的第一个符号

```
do {
    令  $S$  是栈顶状态,  $a$  是  $ip$  所指向的符号;
    if (action[ $S, a$ ] = shift  $S'$ ) {
        把  $a$  和  $S'$  分别压入符号栈和状态栈的栈顶;
        推进  $ip$ , 使它指向下一个输入符号;
    };
    else if (action[ $S, a$ ] = reduce by  $A \rightarrow \beta$ ) {
        从栈顶弹出  $|\beta|$  个符号;
        令  $S'$  是当前栈顶状态, 把  $A$  和 goto[ $S', A$ ] 分别压入符号栈和状态栈的栈顶;
        输出产生式  $A \rightarrow \beta$ ;
    };
    else if (action[ $S, a$ ] = accept) return;
    else error();
} while(1);
```

### 3、代码展示

- 注：分析表使用两个有mask的数组表示，正数表示S，负数表示R，-100为error，100为acc.

```
1  #include<iostream>
2  #include<string.h>
3  #include<string>
4  #include<stdlib.h>
5  #include<fstream>
6  #include <iomanip>
7  #include <unordered_map>
8  #include<typeinfo>
9  #include <stdio.h>
10 using namespace std;
11
12
13 // 接收字符串
14 char str[100000];
15 int ip = 0;
16
17 // 状态栈
18 int state_stack[10000];
19 int state_stack_p = 0;
20
21 // 符号栈
22 char sign_stack[10000];
23 int sign_stack_p = 0;
24
25
26 // 文法分析表
27 // 建立map
28 unordered_map<char, int> map{
29     {'(', 0},
```

```

30     {'}', 1},
31     {'+', 2},
32     {'-', 3},
33     {'*', 4},
34     {'/', 5},
35     {'n', 6},
36     {'$', 7},
37
38     {'S', 0},
39     {'E', 1},
40     {'T', 2},
41     {'F', 3},
42 };
43
44
45
46
47 // S为正数 R为负数 acc为100 -100为err
48 int ACTION[16][8] = {
49     {4, -100, -100, -100, -100, -100, 5, -100},
50     {-100, -100, 6, 7, -100, -100, -100, 100},
51     {-100, -4, -4, -4, 8, 9, -100, -4},
52     {-100, -7, -7, -7, -7, -7, -100, -7},
53     {4, -100, -100, -100, -100, -100, 5, -100},
54     {-100, -9, -9, -9, -9, -9, -100, -9},
55     {4, -100, -100, -100, -100, -100, 5, -100},
56     {4, -100, -100, -100, -100, -100, 5, -100},
57     {4, -100, -100, -100, -100, -100, 5, -100},
58     {4, -100, -100, -100, -100, -100, 5, -100},
59     {-100, 15, 6, 7, -100, -100, -100, -100},
60     {-100, -2, -2, -2, 8, 9, -100, -2},
61     {-100, -3, -3, -3, 8, 9, -100, -3},
62     {-100, -5, -5, -5, -5, -5, -100, -5},
63     {-100, -6, -6, -6, -6, -6, -100, -6},
64     {-100, -8, -8, -8, -8, -8, -100, -8},
65 };
66
67 // -100为err
68 int GOTO[16][4] = {
69     {-100, 1, 2, 3},
70     {-100, -100, -100, -100},
71     {-100, -100, -100, -100},
72     {-100, -100, -100, -100},
73     {-100, 10, 2, 3},
74     {-100, -100, -100, -100},
75     {-100, -100, 11, 3},
76     {-100, -100, 12, 3},
77     {-100, -100, -100, 13},
78     {-100, -100, -100, 14},
79     {-100, -100, -100, -100},
80     {-100, -100, -100, -100},
81     {-100, -100, -100, -100},
82     {-100, -100, -100, -100},
83     {-100, -100, -100, -100},
84     {-100, -100, -100, -100},
85 };
86
87

```

```

88
89
90
91
92 // 产生式 LR
93 // 因为E在多个产生式的左部出现，接受项目不唯一，所以需要增加S->E
94 char *production[] = {
95     "S->E",
96     "E->E+T",
97     "E->E-T",
98     "E->T",
99     "T->T*F",
100    "T->T/F",
101    "T->F",
102    "F->(E)",
103    "F->n",
104 };
105
106
107
108 void set_up_stack(){
109     state_stack[0] = 0;
110     sign_stack[0] = '$';
111 }
112
113
114
115
116 void input_str(){
117     cout << "plz input string for analysis:" << endl;
118     int q = 0;
119     char c;
120     while (1) {
121         if ((c = getchar()) == '\n'){
122             str[q++] = '$';
123             str[q++] = '\0';
124             break;
125         }
126         str[q++] = c;
127     }
128 }
129
130
131
132 void LR_analisis(){
133     while (1) {
134         char str_ip = str[ip];
135         if (str_ip >= '0' && str_ip <= '9') str_ip = 'n';
136
137         // log
138         // cout << "ACTION:";
139         // cout << ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] <<
endl;
140
141         // cout << "state_p:";
142         // cout << state_stack_p << endl;
143
144         // cout << "ip:";

```

```

145         // cout << ip << endl;
146
147         cout << "state stack:";
148         for (int i = 0; i <= state_stack_p; i++) cout << state_stack[i] << '
';
149         cout << endl;
150
151         cout << "sign stack:";
152         for (int i = 0; i <= sign_stack_p; i++) cout << sign_stack[i];
153         cout << endl;
154
155         cout << "the rest:";
156         int temp_ip = ip;
157         while (str[temp_ip] != '$') {
158             cout << str[temp_ip];
159             temp_ip++;
160         }
161         cout << endl;
162
163
164
165
166
167
168         // 如果是S
169         if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] > 0 &&
ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] != 100) {
170             cout << "S" << ACTION[ state_stack[state_stack_p] ][
map[str_ip] ] << endl;
171
172
173             // 将状态加入
174
175             state_stack[state_stack_p + 1] = ACTION[
state_stack[state_stack_p] ][ map[str_ip] ];
176
177             state_stack_p++;
178
179             // 将符号加入
180             sign_stack[++sign_stack_p] = str[ip];
181             // 推进ip
182             ip++;
183
184             cout << endl;
185             cout << endl;
186         }
187
188         // 如果是R
189         else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] < 0 &&
ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] != -100) {
190             cout << "R" << abs(ACTION[ state_stack[state_stack_p] ][
map[str_ip] ]) << '\t';
191             // 绝对值的下标从1开始，所以我们需要先取绝对值再减一得到对应production数
组的下标
192             int pro_p = abs(ACTION[ state_stack[state_stack_p] ][
map[str_ip] ]) - 1;
193
194             // 标记到箭头右边

```

```

195     int right_flag = 0;
196     // 计数
197     int count = 0;
198     for (int i = 0; ; i++) {
199         if (production[pro_p][i] == '>') {
200             right_flag = 1;
201             continue;
202         }
203         else if (production[pro_p][i] == '\\0') {
204             break;
205         }
206         else if (right_flag == 1) {
207             count++;
208         }
209     }
210     // 两个栈分别弹出count个符号
211     state_stack_p -= count;
212     sign_stack_p -= count;
213
214     // 将当前状态的goto压入状态栈栈顶， 将产生式左边的非终结符压入符号栈栈顶
215     // 将状态加入
216
217     state_stack[state_stack_p + 1] = GOTO[
state_stack[state_stack_p] ][ map[production[pro_p][0]] ];
218     state_stack_p++;
219     sign_stack[++sign_stack_p] = production[pro_p][0];
220
221     // 输出
222     cout << production[pro_p] << endl;
223
224     cout << endl;
225 }
226 // 如果是acc
227 else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] ==
100) {
228     cout << "accept!" << endl;
229     return ;
230 }
231
232 else if (ACTION[ state_stack[state_stack_p] ][ map[str_ip] ] ==
-100) {
233     cout << "error!" << endl;
234     return ;
235 }
236 }
237 }
238
239 int main(){
240     set_up_stack();
241     input_str();
242     LR_analisis();
243 }

```



## 4、测试结果

```
1  plz input string for analysis:
2  5*(1+7)/2
3  state stack:0
4  sign stack:$
5  the rest:5*(1+7)/2
6  S5
7
8
9  state stack:0 5
10 sign stack:$5
11 the rest:*(1+7)/2
12 R9      F->n
13
14 state stack:0 3
15 sign stack:$F
16 the rest:*(1+7)/2
17 R7      T->F
18
19 state stack:0 2
20 sign stack:$T
21 the rest:*(1+7)/2
22 S8
23
24
25 state stack:0 2 8
26 sign stack:$T*
27 the rest:(1+7)/2
28 S4
29
30
31 state stack:0 2 8 4
32 sign stack:$T*(
33 the rest:1+7)/2
34 S5
35
36
37 state stack:0 2 8 4 5
38 sign stack:$T*(1
39 the rest:+7)/2
40 R9      F->n
41
42 state stack:0 2 8 4 3
43 sign stack:$T*(F
44 the rest:+7)/2
45 R7      T->F
46
47 state stack:0 2 8 4 2
48 sign stack:$T*(T
49 the rest:+7)/2
50 R4      E->T
51
52 state stack:0 2 8 4 10
53 sign stack:$T*(E
54 the rest:+7)/2
55 S6
```

```
56
57
58 state stack:0 2 8 4 10 6
59 sign stack:$T*(E+
60 the rest:7)/2
61 S5
62
63
64 state stack:0 2 8 4 10 6 5
65 sign stack:$T*(E+7
66 the rest:)/2
67 R9      F->n
68
69 state stack:0 2 8 4 10 6 3
70 sign stack:$T*(E+F
71 the rest:)/2
72 R7      T->F
73
74 state stack:0 2 8 4 10 6 11
75 sign stack:$T*(E+T
76 the rest:)/2
77 R2      E->E+T
78
79 state stack:0 2 8 4 10
80 sign stack:$T*(E
81 the rest:)/2
82 S15
83
84
85 state stack:0 2 8 4 10 15
86 sign stack:$T*(E)
87 the rest:/2
88 R8      F->(E)
89
90 state stack:0 2 8 13
91 sign stack:$T*F
92 the rest:/2
93 R5      T->T*F
94
95 state stack:0 2
96 sign stack:$T
97 the rest:/2
98 S9
99
100
101 state stack:0 2 9
102 sign stack:$T/
103 the rest:2
104 S5
105
106
107 state stack:0 2 9 5
108 sign stack:$T/2
109 the rest:
110 R9      F->n
111
112 state stack:0 2 9 14
113 sign stack:$T/F
```

```
114 | the rest:
115 | R6      T->T/F
116 |
117 | state stack:0 2
118 | sign stack:$T
119 | the rest:
120 | R4      E->T
121 |
122 | state stack:0 1
123 | sign stack:$E
124 | the rest:
125 | accept!
```

**结果完全正确。**

## 六、总结与展望

本次实验，我分别使用递归分析法，LL(1)分析法与LR分析法对输入的算术表达式进行语法分析。其中，递归分析法是自上而下的分析，且存在回溯；LL(1)分析法是自上而下的分析，由于使用了分析表所以没有回溯发生；LR分析是下而上的分析。

在文法改写方面，自上而下的分析**需要去除文法左递归并提取左公因子**。而自下而上的分析**需要增广文法，确保接受状态只有一个**。

本次实验，我不仅对三种不同语法分析的方法有了更加深刻的理解，而且也很好地锻炼了我的C++代码编写能力与整体功能架构的设计能力。我还明白了事先构思好数据处理的方式可能能够大幅度减少代码的冗余（如方法二中的代码，经过unordered\_map的替换之后可以压缩到300行左右）。

美中不足的是，由于时间紧张，本次实验我采用的都是面向过程的编程方法。希望我在之后能够对语法分析有更加清晰和深刻的认识，也希望我的C++代码能力能够进一步的提高。