

实验要求：

实验描述	算术表达式语法分析器的设计与实现
实验要求	使用 LL(1)分析法 和 LR分析法 设计实现算术表达式的语法分析器算数表达式至少支持加减乘除以及括号操作，即 (+, -, *, /, ()) 。
提交内容	实验报告，报告内容必须包含：算术表达式所依据的文法；LL(1)和LR分析法所使用的分析表，以及简要分析；程序执行流程；程序运行结果展示。2.语法分析源程序：source.c（源程序包）3.可执行文件4.程序测试文件：test.txt（实验输入，将测试案例写入程序的可没有此项）
提交方式	1.提交文件压缩包，压缩包中包含以上内容，压缩包的命名为“1933XXXX-张三-实验2”；2.发送邮箱： wang0108153077@163.com ；3.邮件主题：同压缩包命名。
截止时间	2022年5月7日23:59
附件	编译原理实验二

LL(1) 文法

将文法变化为 LL(1) 文法。

$$E \rightarrow TE_1 \text{ ①}$$

$$E_1 \rightarrow \omega_0 TE_1 \text{ ②} \mid \varepsilon \text{ ③}$$

$$T \rightarrow FT_1 \text{ ④}$$

$$T_1 \rightarrow \omega_1 FT_1 \text{ ⑤} \mid \varepsilon \text{ ⑥}$$

$$F \rightarrow I \text{ ⑦} \mid (E) \text{ ⑧}$$

其中 $w_0 : + -$

$$w_1 : * /$$

I : 数字或常数

求上列文法的选择符集合，确认文法受否是 LL(1) 文法。

$\text{select}(\textcircled{1}) = \text{first}(T) = \text{first}(F) = \{I, (\}$

$\text{select}(\textcircled{2}) = \{w_0\}$

$\text{select}(\textcircled{3}) = \text{follow}(E_1) = \text{follow}(E) = \{), \#\}$

$\text{select}(\textcircled{4}) = \text{first}(F) = \{I, (\}$

$\text{select}(\textcircled{5}) = \{w_1\}$

$\text{select}(\textcircled{6}) = \text{follow}(T_1) = \text{follow}(T) = \text{first}(E_1) = \{w_0\} + \text{follow}(E_1) = \{w_0\} + \text{follow}(E) = \{w_0, \#,)\}$

$\text{select}(\textcircled{7}) = \{I\}$

$\text{select}(\textcircled{8}) = \{(\}$

其中, $\text{select}(\textcircled{2})$ 交 $\text{select}(\textcircled{3})$ 为空集, $\text{select}(\textcircled{5})$ 交 $\text{select}(\textcircled{6})$ 为空集, $\text{select}(\textcircled{7})$ 交 $\text{select}(\textcircled{8})$ 为空集, 因此该文法为 LL(1) 文法。

LL(1) 分析法

LL(1) 分析器包括 LL(1) 控制程序 + LL(1) 分析表。

LL(1) 分析表

设计 LL(1) 分析表。如下:

	w_0	w_1	()	I	#
E			①		①	
E_1	②			③		③
T			④		④	
T_1	⑥	⑤		⑥		⑥
F			⑧		⑦	

LL(1) 程序

然后设计 LL(1) 分析法控制程序。虽然上表中接收的信号是字符, 但实际上语法分析器接收的信号是词法分析器得到的每个字符串的编码, 因此将算术表达式经过词法分析器后, 得到每个字符的编码, 然后对编码序列进行语法分析, 对算术表达式的合法性进行判断。

回顾实验一, 将算术表达式用到的终结符对应的编码总结如下:

单词符号	种别编码	终结符号表达
常数	0	I
+	35	w_0
-	36	w_0
*	37	w_1
/	38	w_1
;	68	#
(71	(
)	72)

通过实验一的程序，对算术表达式进行词法分析，得到算术表达式中每个字符串的编码序列，然后将编码序列作为 LL(1) 分析器的输入。

首先，通过函数 `vector<int> readLexicalAnalyzerOut(char* fileName)` 从词法分析器的输出文件中读取字符串编码，并将编码以 `int` 型保存到一个 `vector` 中。处理时需注意，编码序列压入 `vector` 后，顺序与源顺序相反。

```

1 //1. Read message from LexicalAnalyzerOut.txt
2 //2. Produce key message for Parsing
3 vector<int> readLexicalAnalyzerOut( char* fileName ){
4     ifstream infile(fileName);
5     if( !infile ){
6         cout << "Unable to open LexicalAnalyzerOut.txt" << endl;
7         exit(EXIT_FAILURE);
8     }
9     ofstream outfile("./ParsingIn.txt");
10    if( !outfile ){
11        cout << "Unable to open outfile" << endl;
12        exit(EXIT_FAILURE);
13    }
14    //infile.getline(); // 第一行
15    string line;
16    getline(infile,line);
17    vector<int> nums;
18    string tmp;
19    while( getline(infile, line) ){
20        //getline(infile, line);
21        //line = infile.getline();
22        stringstream input(line);
23        input >> tmp;
24        input >> tmp;
25        int i = stoi(tmp);
26        nums.push_back(i);
27    }
28    //for(int i = 0; i < nums.size(); i++){
29        // cout << nums[i] << endl;
30    //}
31    infile.close();
32    outfile.close();
33    return nums;

```

在 LL(1) 分析器中：

- 调用函数 `readLexicalAnalyzerOut` 得到编码序列的 vector `nums`;
- 用一个 vector `stack` 模拟一个栈，并将 `#` 和 `E` 压栈；
- 当栈不空时，读取 `nums` 元素保存至 `num`，根据栈顶符号对 `num` 进行处理（注意读取 `nums` 元素时从最后往前读取，因为 `nums` 中元素顺序是反的）：
 - 当栈顶符号是 `E`，如果当前 `num` 是 0 或 71，则将 `E` 弹出栈，将 `e` 和 `T` 压栈；如果 `num` 是其他数字，则可以判断算术表达式发生语法错误。（为了方便处理，将 E_1 表示为 `e`，将 T_1 表示为 `t`，将 w_0 表示为 `W`，将 w_1 表示为 `w`）。
 - 当栈顶符号是 `e`，如果 `num` 是 w_0 ，则将 `e` 弹出栈，将 w_0TE_1 逆序压栈；如果 `num` 是 68 或 72，直接将 `e` 弹出栈；如果 `num` 是其他数字，则判断发生语法错误。
 - 当栈顶符号是 `T`，如果 `num` 是常数或 `num = 71`，则将 `T` 弹出栈，将 FT_1 逆序压栈；如果 `num` 是其他情况，则判断发生语法错误。
 - 当栈顶元素是 `t` (T_1)，如果 `num` 是 w_0 或 72 或 68，则将 `t` 弹出栈；如果 `num` 是 w_1 ，则将 `t` 弹出栈，并将 w_1FT_1 逆序压栈；如果 `num` 是其他情况，则判断发生语法错误。
 - 当栈顶元素是 `F`，如果 `num` 是 71，则将 `F` 弹出栈，并将 `(E)` 逆序压栈，如果 `num` 是常数 `I`，则将 `F` 弹出栈，并读取 `nums` 中的下个编码到 `num` 中；如果 `num` 是其他情况，则判断发生语法错误。
 - 当栈顶元素是 `#`，如果 `num = 68`，则判断此算术表达式合法，返回 `true`；否则，判断发生语法错误。
 - 当栈顶元素是终结符（或）或 `W` 或 $w(w_1)$ ，只需判断 `num` 是否是这些终结符对应的编码，如果是则将栈顶元素弹出栈，然后读取 `nums` 中的下一个编码到 `num` 中；如果 `num` 是其他情况，则判断发生语法错误。

```

1  bool LL1(char* fileName){
2      vector<char> stack;
3      //e = E1, t = T1, w = w0, w = w1
4      vector<int> nums = readLexicalAnalyzerOut(fileName);
5      int num;
6      int index = 0;
7      Getnum(num, nums, index);
8      stack.push_back('#');
9      stack.push_back('E');
10     cout << "begin" << endl;
11     while( stack.size() > 0 ){
12         char s = stack.back();
13         cout << stack.size() << endl;
14         for( int i = 0; i < stack.size(); i++ ){
15             cout << stack[i] << endl;
16         }
17         getchar();
18         if( s == 'E' ){
19             cout << "num is " << num << endl;
20             if( num == 0 || num == 71 ){
21                 //if( isI(ch) || ch == '(' ){
22                 stack.pop_back();
23                 stack.push_back('e');
24                 stack.push_back('T');
25             }
26             else{
27                 cout << "Error in E" << endl;
28                 getchar();

```

```

29         exit(EXIT_FAILURE);
30     }
31 }
32 else if( s == 'e' ){
33     if( isw0(num) ){
34         //if( isw0(ch) ){
35             stack.pop_back();
36             stack.push_back('e');
37             stack.push_back('T');
38             stack.push_back('w');
39         }
40     else if( num == 68 || num == 72 ){
41         stack.pop_back();
42     }
43     else{
44         cout << "Error in E1" << endl;
45         getchar();
46         exit(EXIT_FAILURE);
47     }
48 }
49 else if( s == 'T' ){
50     //cout << "ch is " << ch << endl;
51     if( isI(num) || num == 71 ){
52         //if( isI(ch) || ch == '(' ){
53             stack.pop_back();
54             stack.push_back('t');
55             stack.push_back('F');
56         }
57     else{
58         cout << "Error in T" << endl;
59         getchar();
60         exit(EXIT_FAILURE);
61     }
62 }
63 else if( s == 't' ){
64     if( isw0(num) || num == 72 || num == 68 ){
65         //if( isw0(ch) || ch == ')' || ch == ';' ){
66             stack.pop_back();
67         }
68     else if( isw1(num) ){
69         //else if( isw1(ch) ){
70             stack.pop_back();
71             stack.push_back('t');
72             stack.push_back('F');
73             stack.push_back('w');
74         }
75     else{
76         cout << "Error in t" << endl;
77         getchar();
78         exit(EXIT_FAILURE);
79     }
80 }
81 else if( s == 'F' ){
82     if( num == 71 ){
83         //if( ch == '(' ){
84             stack.pop_back();
85             stack.push_back(')');
86             stack.push_back('E');

```

```

87         stack.push_back('(');
88     }
89     else if( isI(num) ){
90         //else if( isI(ch) ){
91         stack.pop_back();
92         //read next word
93         //Getchar();
94         Getnum(num,nums, index);
95         //inFile >> ch;
96     }
97     else{
98         cout << "Error in F" << endl;
99         getchar();
100        exit(EXIT_FAILURE);
101    }
102 }
103 else if( s == '#' ){
104     if( num == 68 )
105         //if( ch == ';' )
106         return true;
107     else{
108         cout << "Error in #" << endl;
109         getchar();
110         return false;
111     }
112 }
113 else if( ( s == '(' && num == 71) || ( s == ')' && num == 72) || (
114 s == 'w' && isw0(num)) || ( s == 'w' && isw1(num)) ){
115     //else if(s == ch || ( s == 'w' && isw0(ch)) || ( s == 'w' &&
116 isw1(ch) ) ){
117     stack.pop_back();
118     //Getchar();
119     Getnum(num, nums, index);
120 }
121 else{
122     cout << "Error in " << s << endl;
123     getchar();
124     return false;
125 }
126 }
127 }

```

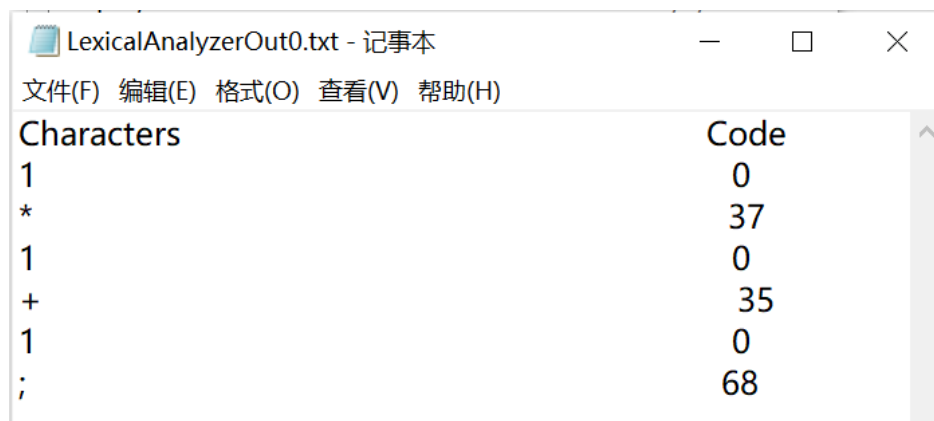
实验结果

测试样例1:

原始算术表达式如下:

test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1 * 1 + 1;

通过词法分析器得到编码序列:



Characters	Code
1	0
*	37
1	0
+	35
1	0
;	68

运行 LL(1) 程序，每进行一次压栈和弹栈操作，都需要从键盘输入一个字符（getchar() 函数进行控制），程序会输出栈的元素和当前 num 的值，这样可以看到程序运行过程，得到输出结果如下：

C:\Users\azhi\Desktop\作业\编译原理\hw\实验\hw2\LL(1).exe

begin

2

#

E

num is 0

3

#

e

T

4

#

e

t

F

3

#

e

t

5

#

e

t

F

w

4

#

e

t

F

3

#

e

t

2

e

4

e
T
W

3

e
T

4

e
t
F

3

e
t

2

e

1
#

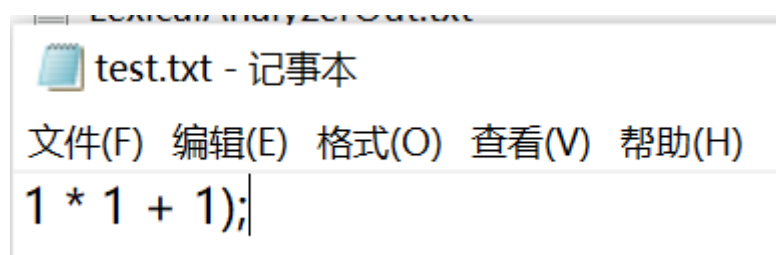
Arithmetic expression is legal.

步骤分析：

分析栈	X	W	剩余序列	操作
#E	E	1	1 * 1 + 1;	pop(E), push(E_1T)
# E_1T	T	1	* 1 + 1;	pop(T), push(T_1F)
# E_1T_1F	F	1	* 1 + 1;	pop(F), 匹配 1
# E_1T_1	T_1	*	1 + 1;	pop(T_1), push(T_1Fw_1)
# $E_1T_1Fw_1$	w_1	*	1 + 1;	pop(w_1), 匹配 *
# E_1T_1F	F	1	+ 1;	pop(F), 匹配 1
# E_1T_1	T_1	+	1;	pop(T_1)
# E_1	E_1	+	1;	pop(E_1), push(Tw_0)
# E_1Tw_0	w_0	+	1;	pop(w_0), 匹配 +
# E_1T	T	1	;	pop(T), push(T_1F)
# E_1T_1F	F	1	;	pop(F), 匹配 1
# E_1T_1	T_1	;		pop(T_1)
# E_1	E_1	;		pop(E_1)
#	#	;		匹配成功!!!

测试样例2:

原始算术表达式如下:



通过词法分析器得到编码序列:

LexicalAnalyzerOut.txt - 记事本		—	□	×
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)				
Characters	Code			
1	0			
*	37			
1	0			
+	35			
1	0			
)	72			
;	68			

运行 LL(1) 程序分析:

C:\Users\azhi\Desktop\作业\编译原理\hw\实验\hw2\LL(1).exe

begin

2

#

E

num is 0

3

#

e

T

4

#

e

t

F

3

#

e

t

5

#

e

t

F

w

4

#

e

t

F

3

#

e

t

```
2
#
e

4
#
e
T
W

3
#
e
T

4
#
e
t
F

3
#
e
t

2
#
e

1
#

Error in #
```

可以看到对合法的算术表达式和非法的算术表达式，LL(1) 程序都能正确判断。

步骤分析：

分析栈	X	W	剩余序列	操作
#E	E	1	1 * 1 + 1);	pop(E), push(E_1T)
# E_1T	T	1	* 1 + 1);	pop(T), push(T_1F)
# E_1T_1F	F	1	* 1 + 1);	pop(F), 匹配 1
# E_1T_1	T_1	*	1 + 1);	pop(T_1), push(T_1Fw_1)
# $E_1T_1Fw_1$	w_1	*	1 + 1);	pop(w_1), 匹配 *
# E_1T_1F	F	1	+ 1);	pop(F), 匹配 1
# E_1T_1	T_1	+	1);	pop(T_1)
# E_1	E_1	+	1);	pop(E_1), push(Tw_0)
# E_1Tw_0	w_0	+	1);	pop(w_0), 匹配 +
# E_1T	T	1);	pop(T), push(T_1F)
# E_1T_1F	F	1);	pop(F), 匹配 1
# E_1T_1	T_1)	;	pop(T_1)
# E_1	E_1)	;	pop(E_1)
#	#)	;	匹配失败!!!

LR() 分析法

表达式文法

$E \rightarrow E w_0 T (1) \mid T (2)$

$T \rightarrow T w_1 F (3) \mid F (4)$

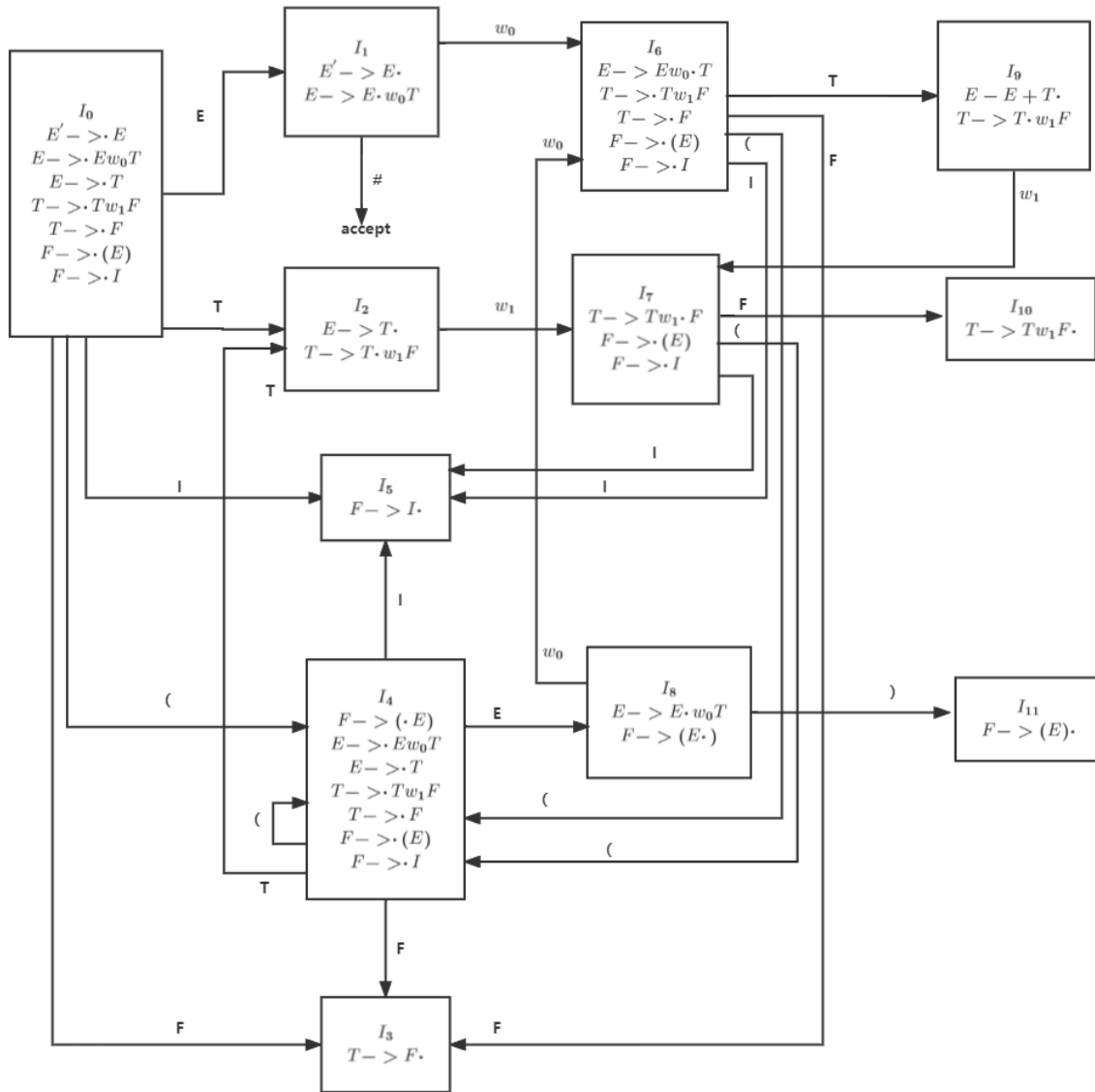
$F \rightarrow (E) (5) \mid I (6)$

其中 w_0 : + -

w_1 : * /

I : 数字或常数

表达式文法的 LR(0) 自动机



LR(0) 分析表构造

(1) 扩展文法，构造句柄识别器

$E' \rightarrow E$

$E \rightarrow E w_0 T \quad (1) \mid T \quad (2)$

$T \rightarrow T w_1 F \quad (3) \mid F \quad (4)$

$F \rightarrow (E) \quad (5) \mid I \quad (6)$

(2) 根据句柄识别器，填写 LR(0) 分析表：

	I	w_0	w_1	()	#	E	T	F
0	s5			s4			1	2	3
1		s6				accept			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

程序实现

伪代码：

```

1  令a为w$的第一个符号；
2  while(1){
3      令 s 是栈顶的状态；
4      if( ACTION[s,a] = 移入 t ){
5          将 t 压入栈中；
6          令 a 为下一个输入符号；
7      }
8      else if( ACTION[s,a] = 规约 A-> b ){
9          从栈中弹出|b|个符号；
10         令 t 为当前的栈顶状态；
11         将 GOTO[t,A]压入栈中；
12         输出产出式 A->b；
13     }
14     else if( ACTION[s,a] = 接受 ) break; /*语法分析完毕*/
15     else 调用错误恢复例程；
16 }


```

代码实现见文件 LR.cpp。


实验结果

测试样例1：

原始算术表达式如下：

 test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1 * 1 + 1;

通过词法分析器得到编码序列：

 LexicalAnalyzerOut0.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Characters	Code
1	0
*	37
1	0
+	35
1	0
;	68

运行 LR 程序，每进行一次操作，都需要从键盘输入一个字符（getchar() 函数进行控制），这样便于观察程序的运行情况。输出每次 state 栈的信息和符号表的信息，可以看到运行情况：


```
C:\Users\azhi\Desktop\作业\编译原理\hw\实验\hw2\LR.exe
states: 0
Characters:
states: 0 5
Characters: I
states: 0 3
Characters: F
states: 0 2
Characters: T
states: 0 2 7
Characters: T w
states: 0 2 7 5
Characters: T w I
states: 0 2 7 a
Characters: T w F
states: 0 2
Characters: T
states: 0 1
Characters: E
states: 0 1 6
Characters: E W
states: 0 1 6 5
Characters: E W I
states: 0 1 6 3
Characters: E W F
states: 0 1 6 9
Characters: E W T
states: 0 1
Characters: E
Arithmetic expression is legal.
```

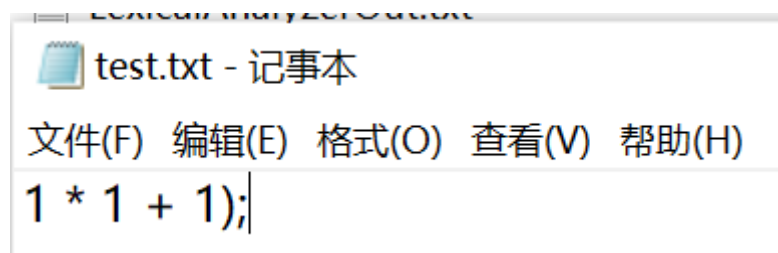
可以看到该式子是合法的。

步骤分析：

步骤	栈	符号	输入	动作
1	0		$Iw_1Iw_0I\#$	移入
2	0 5	I	$w_1Iw_0I\#$	根据 $F \rightarrow I$ 归约
3	0 3	F	$w_1Iw_0I\#$	根据 $T \rightarrow F$ 归约
4	0 2	T	$w_1Iw_0I\#$	移入
5	0 2 7	Tw_1	$Iw_0I\#$	移入
6	0 2 7 5	Tw_1	$w_0I\#$	根据 $F \rightarrow I$ 归约
7	0 2 7 10	Tw_1I	$w_0I\#$	根据 $T \rightarrow Tw_1F$ 归约
8	0 2	Tw_1F	$w_0I\#$	根据 $E \rightarrow T$ 归约
9	0 1	T	$w_0I\#$	移入
10	0 1 6	E	$I\#$	移入
11	0 1 6 5	EW_0	$\#$	根据 $F \rightarrow I$ 归约
12	0 1 6 3	EW_0I	$\#$	根据 $T \rightarrow F$ 归约
13	0 1 6 9	EW_0T	$\#$	根据 $E \rightarrow E + T$ 归约
14	0 1	E	$\#$	接受

测试样例2:

原始算术表达式如下:



通过词法分析器得到编码序列:

Characters	Code
1	0
*	37
1	0
+	35
1	0
)	72
;	68

运行 LR 程序, 输出每次 state 栈的信息和符号表的信息, 可以看到运行情况:

```
C:\Users\azhi\Desktop\作业\编译原理\hw\实验\hw2\LR.exe
states: 0
Characters:

states: 0 5
Characters: I

states: 0 3
Characters: F

states: 0 2
Characters: T

states: 0 2 7
Characters: T w

states: 0 2 7 5
Characters: T w I

states: 0 2 7 a
Characters: T w F

states: 0 2
Characters: T

states: 0 1
Characters: E

states: 0 1 6
Characters: E W

states: 0 1 6 5
Characters: E W I

states: 0 1 6 3
Characters: E W F

states: 0 1 6 9
Characters: E W T

states: 0 1
Characters: E

Error in state 1
Arithmetic expression is not legal
```

可以看到此算术表达式不合法。

步骤分析：

步骤	栈	符号	输入	动作
1	0		$Iw_1Iw_0I)\#$	移入
2	0 5	I	$w_1Iw_0I)\#$	根据 $F \rightarrow I$ 归约
3	0 3	F	$w_1Iw_0I)\#$	根据 $T \rightarrow F$ 归约
4	0 2	T	$w_1Iw_0I)\#$	移入
5	0 2 7	Tw_1	$Iw_0I)\#$	移入
6	0 2 7 5	Tw_1	$w_0I)\#$	根据 $F \rightarrow I$ 归约
7	0 2 7 10	Tw_1I	$w_0I)\#$	根据 $T \rightarrow Tw_1F$ 归约
8	0 2	Tw_1F	$w_0I)\#$	根据 $E \rightarrow T$ 归约
9	0 1	T	$w_0I)\#$	移入
10	0 1 6	E	$I)\#$	移入
11	0 1 6 5	EW_0	$)\#$	根据 $F \rightarrow I$ 归约
12	0 1 6 3	EW_0I	$)\#$	根据 $T \rightarrow F$ 归约
13	0 1 6 9	EW_0T	$)\#$	根据 $E \rightarrow E + T$ 归约
14	0 1	E	$)\#$	发生错误，调用错误程序