

实验要求

实验描述	表达式语义分析器的设计与实现
实验要求	使用递归下降翻译法或LL(1)翻译法实现高级编程语言的语义分析，将其翻译为四元式格式的中间语言，至少支持算术表达式的语义分析。算术表达式至少支持加减乘除以及括号操作，即 (+, -, *, /, ())。
提交内容	1.实验报告，报告内容必须包括：•翻译文法；•若采用递归下降翻译法，须给出文法（至少实现算术表达式的文法）的子程序流程图，并在其上标注返回地址；•给出一个算术表达式实例的分析表（表项内容参考实验三PPT P17）；•运行结果展示；•以及其他必要内容2.语义分析源程序：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3”；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27日23:59
附件	编译原理实验三

语义分析器

语法制导翻译技术是指：语法分析技术+属性翻译文法构造技术。

LL(1) 文法

将文法变化为 LL(1) 文法。

$$E \rightarrow TE_1 \text{ ①}$$

$$E_1 \rightarrow \omega_0 T \{GEQ(\omega_0)\} E_1 \text{ ②} \mid \varepsilon \text{ ③}$$

$$T \rightarrow FT_1 \text{ ④}$$

$$T_1 \rightarrow \omega_1 F \{GEQ(\omega_1)\} T_1 \text{ ⑤} \mid \varepsilon \text{ ⑥}$$

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

其中 $\omega_0 : + -$

$w_1 : */$

I : 数字或常数

LL(1) 分析表

设计 LL(1) 分析表。如下：

	w_0	w_1	()	I	#
E			①		①	
E1	②			③		③
T			④		④	
T1	⑥	⑤		⑥		⑥
F			⑧		⑦	

程序实现

这里只讲解LL1函数的实现。输入是由词法分析得到的文件，运行过程中输出每一步的压栈（语法栈）操作，和产生的四元式，以及对四元式进行操作后的结果。

首先定义一个结构体 struct mydata 用于存储读取词法分析文件的数据：

```
1 struct mydata{
2     vector<int> nums;
3     vector<double> cons;
4 };
```

nums存储编码，cons存储常数。即cons只有在编码是0（常数）时才会对源数据进行存储。

声明必要的变量：

```
1     vector<string> stack;
2     vector<double> SEM;
3     struct mydata data = readLexicalAnalyzerOut(fileName);
4     int num;
5     double constant;
6     int index = 0;
7     int cons_index = 0;
```

其中stack是语法栈，SEM是语义栈，num用于存储当前处理的编码，constant用于存储当前读取的常数，index为data.nums的下标，cons_index为data.cons的下标。

然后从data.nums中读取第一个编码进行处理，并对语法栈进行初始化。

```
1     Getnum(num, data.nums, index);
2     stack.push_back("#");
3     stack.push_back("E");
```

然后进入主要的处理程序，按照上面LL(1)分析表，进行实现。

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 $w_0 T \{GEQ(w_0)\} E_1$ 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 $w_1 F \{GEQ(w_1)\} T_1$ 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，并把常数 i 压进语义栈，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。
- 当栈顶元素是 $GEQ(w_0)$ ，如果 w_0 是加号，则进行加法操作，如果 w_0 是减号，则进行减法操作。弹出语义栈顶两个元素，并生成相应的四元素，若语义栈此时的元素个数小于 2，则判断发生语义错误。设语义栈第一个元素是 cons1，第二个元素是 cons2，则生成四元素为 (w_0 cons2, cons1, res)。生成四元素后，将运算结果 $res = cons_2 \ w_0 \ cons_1$ 压入语义栈。同时将产生的四元素进行输出。
- 当栈顶元素是 $GEQ(w_1)$ ，如果 w_1 是乘号，则进行乘法操作，若 w_1 是除号，则进行除法操作。弹出语义栈顶两个元素，并生成相应的四元素，若语义栈此时的元素个数小于 2，则判断发生语义错误。设语义栈第一个元素是 cons1，第二个元素是 cons2，则生成四元素为 (w_1 cons2, cons1, res)。生成四元素后，将运算结果 $res = cons_2 \ w_1 \ cons_1$ 压入语义栈。同时将产生的四元素进行输出。

```
1 while( stack.size() > 0 ){
2     string s = stack.back();
3     cout << stack.size() << endl;
4     for( int i = 0; i < stack.size(); i++ ){
5         cout << stack[i] << endl;
6     }
7     getchar();
8     if( s == "E" ){
9         cout << "num is " << num << endl;
10        if( num == 0 || num == 71 ){
11            //if( isI(ch) || ch == '(' ){
12                stack.pop_back();
13                stack.push_back("e");
14                stack.push_back("T");
15            }
16        }
17        else{
18            cout << "Error in E" << endl;
19            getchar();
20            exit(EXIT_FAILURE);
21        }
22    }
23    else if( s == "e" ){
```

```

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

```

```

81         stack.pop_back();
82         stack.push_back(")");
83         stack.push_back("E");
84         stack.push_back("(");
85     }
86     else if( isI(num) ){
87         //else if( isI(ch) ){
88         stack.pop_back();
89         //read next word
90         //Getchar();
91         Getcons(constant, data.cons, cons_index);
92         SEM.push_back(constant);
93         cout << "PUSH(" << constant << ")" << endl;
94         Getnum(num,data.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) || (
s == "w" && isw0(num)) || ( s == "w" && isw1(num)) ){
114     //else if(s == ch || ( s == 'w' && isw0(ch)) || ( s == 'w' &&
isw1(ch) ) ){
115         stack.pop_back();
116         //Getchar();
117         Getnum(num, data.nums, index);
118     }
119     else if( s == "GEQ(+)" ){
120         stack.pop_back();
121         double a, b;
122         double c;
123         if( SEM.size() >= 2 ){
124             a = SEM.back();
125             SEM.pop_back();
126             b = SEM.back();
127             SEM.pop_back();
128             c = a + b;
129             SEM.push_back(c);
130             cout << endl;
131             cout << "Produce: (+ " << b << ", " << a << ", c)" << endl;
132             cout << "c = " << c << endl;
133             cout << endl;
134         }
135         else{
136             cout << "Error in GEQ(+)" << endl;

```

```

137         cout << "SEM size is " << SEM.size() << endl;
138         return false;
139     }
140 }
141 else if( s == "GEQ(-)" ){
142     stack.pop_back();
143     double a, b;
144     double c;
145     if( SEM.size() >= 2 ){
146         a = SEM.back();
147         SEM.pop_back();
148         b = SEM.back();
149         SEM.pop_back();
150         c = b - a;
151         SEM.push_back(c);
152         cout << endl;
153         cout << "Produce: (- " << b << ", " << a << ", c)" << endl;
154         cout << "c = " << c << endl;
155         cout << endl;
156     }
157     else{
158         cout << "Error in GEQ(-)" << endl;
159         cout << "SEM size is " << SEM.size() << endl;
160         return false;
161     }
162 }
163 else if( s == "GEQ(*)" ){
164     stack.pop_back();
165     double a, b;
166     double c;
167     if( SEM.size() >= 2 ){
168         a = SEM.back();
169         SEM.pop_back();
170         b = SEM.back();
171         SEM.pop_back();
172         c = a * b;
173         SEM.push_back(c);
174         cout << endl;
175         cout << "Produce: (* " << b << ", " << a << ", c)" << endl;
176         cout << "c = " << c << endl;
177         cout << endl;
178     }
179     else{
180         cout << "Error in GEQ(*)" << endl;
181         cout << "SEM size is " << SEM.size() << endl;
182         return false;
183     }
184 }
185 else if( s == "GEQ(/)" ){
186     stack.pop_back();
187     double a, b;
188     double c;
189     if( SEM.size() >= 2 ){
190         a = SEM.back();
191         SEM.pop_back();
192         b = SEM.back();
193         SEM.pop_back();
194         c = b / a;

```

```

195         SEM.push_back(c);
196         cout << endl;
197         cout << "Produce: (/ " << b << ", " << a << ", c)" << endl;
198         cout << "c = " << c << endl;
199         cout << endl;
200     }
201     else{
202         cout << "Error in GEQ(/)" << endl;
203         cout << "SEM size is " << SEM.size() << endl;
204         return false;
205     }
206 }
207 else{
208     cout << "Error in " << s << endl;
209     getchar();
210     return false;
211 }
212 }

```

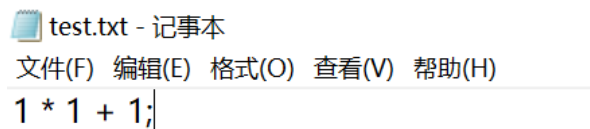
实验结果

说明：

支持的功能有包含加减乘除的算术表达式的语义分析，其中算术表达式只能支持常数的算术表达式，不支持变量算术表达式。

测试样例1：

原算术表达式如下：



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：

0 37 0 35 0 68

begin

2

SYN: # E

SEM:

Input: 37 0 35 0 68

num is 0

3

SYN: # e T

SEM:

Input: 37 0 35 0 68

4

SYN: # e t F

SEM:

Input: 37 0 35 0 68

PUSH(1)

3

SYN: # e t

SEM: 1

Input: 0 35 0 68

6

SYN: # e t GEQ(*) F w

SEM: 1

Input: 0 35 0 68

C:\Users\azhi\Desktop\hw\编译原理\hw\实验\hw3\LL(1).exe

```
PUSH(1)
4
SYN: # e GEQ(+) t
SEM: 1 1
Input:

3
SYN: # e GEQ(+)
SEM: 1 1
Input:

Produce: (+ 1, 1, c)
c = 2

2
SYN: # e
SEM: 2
Input:

1
SYN: #
SEM: 2
Input:

Arithmetic expression is legal.
```

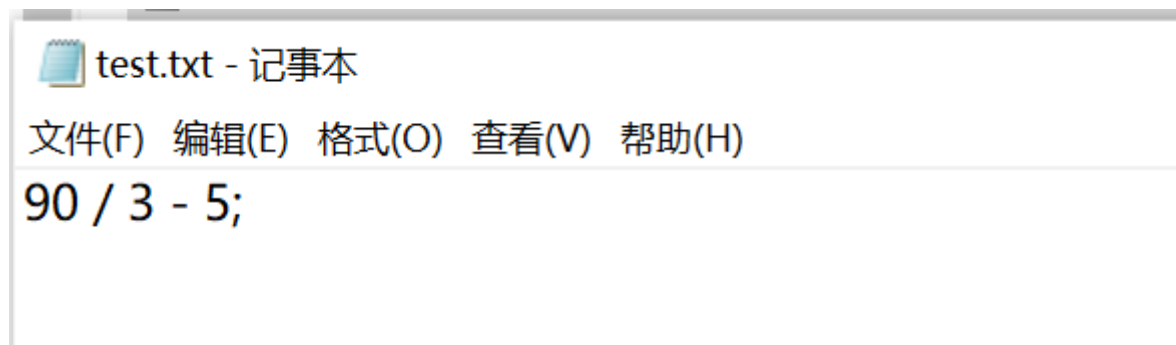
可以看到式子是合法的，最后的结果为2.

步骤分析：

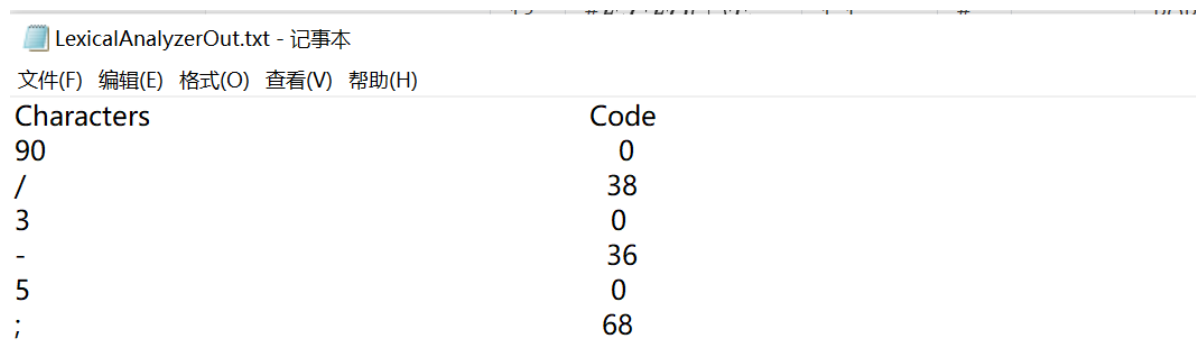
步骤	语法栈SYN	语义栈SEM	符号	输入	动作
1	#E		I	$w_1 I w_0 I$ #	$E \rightarrow T E_1$
2	#E ₁ T		I	$w_1 I w_0 I$ #	$T \rightarrow F T_1$
3	#E ₁ T ₁ F		I	$w_1 I w_0 I$ #	PUSH(1), POP(F)
4	#E ₁ T ₁	1	w ₁	$I w_0 I$ #	$T_1 \rightarrow \omega_1 F \{GEQ(\omega_1)\} T_1$
5	#E ₁ T ₁ GEQ(*)Fw ₁	1	w ₁	$I w_0 I$ #	POP(ω_1)
6	#E ₁ T ₁ GEQ(*)F	1	I	$w_0 I$ #	PUSH(1), POP(F)
7	#E ₁ T ₁ GEQ(*)	1, 1	w ₀	I #	产生四元式(* 1, 1, c), 语法栈 POP(GEQ(*)), 语义栈 POP(1), POP(1), PUSH(1)
8	#E ₁ T ₁	1	w ₀	I #	POP(T ₁)
9	#E ₁	1	w ₀	I #	$E_1 \rightarrow w_0 T \{GEQ(w_0)\} E_1$
10	#E ₁ GEQ(+)Tw ₀	1	w ₀	I #	POP(w ₀)
11	#E ₁ GEQ(+)T	1	I	#	$T \rightarrow F T_1$
12	#E ₁ GEQ(+)T ₁ F	1, 1	I	#	PUSH(1), POP(F)
13	#E ₁ GEQ(+)T ₁	1, 1	#		POP(T ₁)
14	#E ₁ GEQ(+)	1, 1	#		产生四元式(+ 1, 1, c), 语法栈 POP(GEQ(+)), 语义栈POP(1), POP(1), PUSH(2)
15	#E ₁	2	#		POP(E ₁)
16	#		#		OK

测试样例2:

原算术表达式如下:

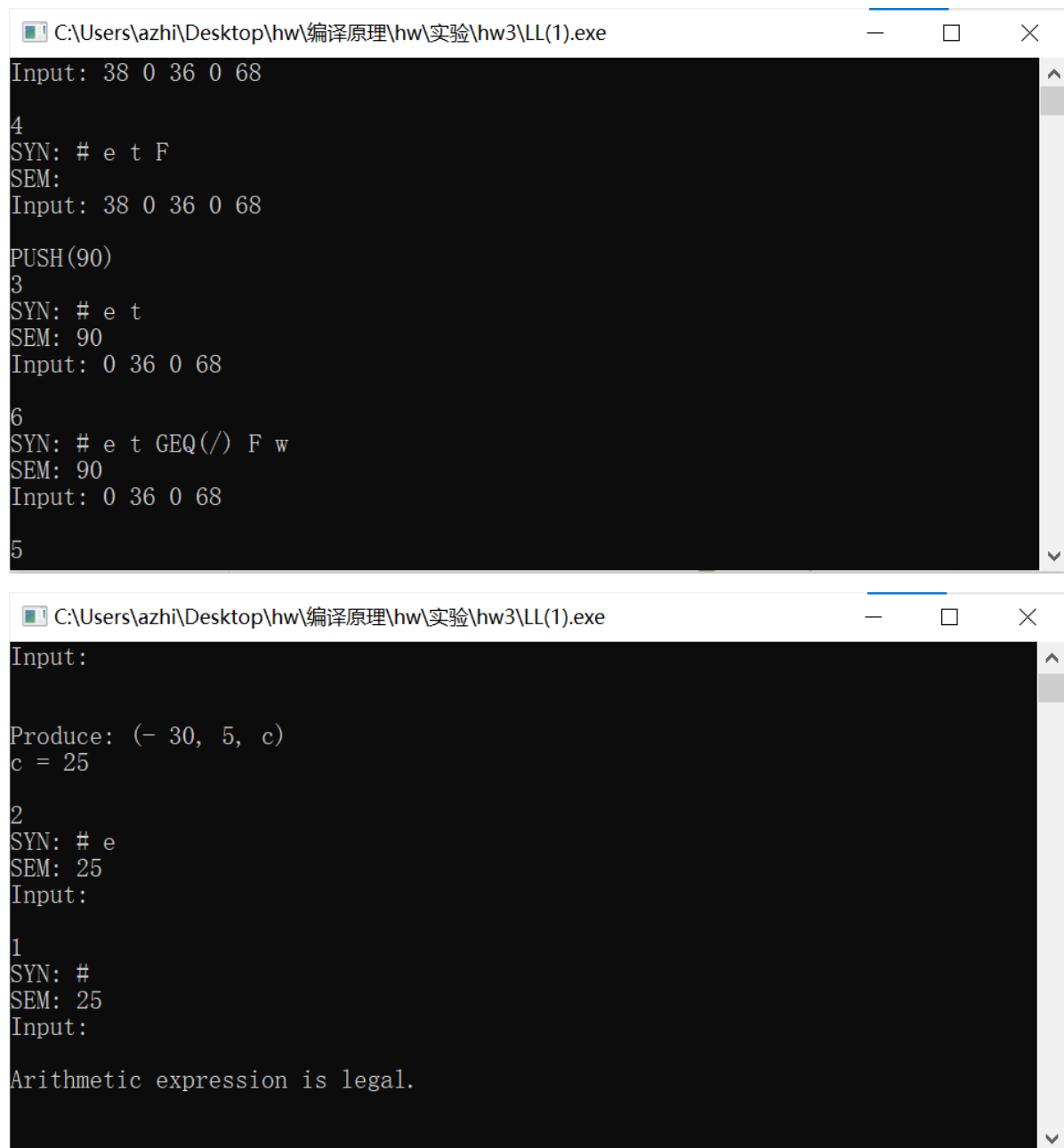


通过词法分析器得到编码序列：



Characters	Code
90	0
/	38
3	0
-	36
5	0
;	68

运行程序，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\hw\编译原理\hw\实验\hw3\LL(1).exe
Input: 38 0 36 0 68
4
SYN: # e t F
SEM:
Input: 38 0 36 0 68
PUSH(90)
3
SYN: # e t
SEM: 90
Input: 0 36 0 68
6
SYN: # e t GEQ(/) F w
SEM: 90
Input: 0 36 0 68
5

C:\Users\azhi\Desktop\hw\编译原理\hw\实验\hw3\LL(1).exe
Input:
Produce: (- 30, 5, c)
c = 25
2
SYN: # e
SEM: 25
Input:
1
SYN: #
SEM: 25
Input:
Arithmetic expression is legal.
```

可以看到式子是合法的，最后的结果为25.

步骤分析：

步骤	语法栈SYN	语义栈SEM	符号	输入	动作
1	#E		I	$w_1 I w_0 I$ #	$E \rightarrow T E_1$
2	#E ₁ T		I	$w_1 I w_0 I$ #	$T \rightarrow F T_1$
3	#E ₁ T ₁ F		I	$w_1 I w_0 I$ #	PUSH(90), POP(F)
4	#E ₁ T ₁	90	w ₁	$I w_0 I$ #	$T_1 \rightarrow \omega_1 F \{GEQ(\omega_1)\} T_1$
5	#E ₁ T ₁ GEQ(/)Fw ₁	90	w ₁	$I w_0 I$ #	POP(ω_1)
6	#E ₁ T ₁ GEQ(/)F	90	I	$w_0 I$ #	PUSH(3), POP(F)
7	#E ₁ T ₁ GEQ(/)	90, 3	w ₀	I #	产生四元式(/ 90, 3, c), 语法栈 POP(GEQ(/)), 语义栈 POP(3), POP(90), PUSH(30)
8	#E ₁ T ₁	30	w ₀	I #	POP(T_1)
9	#E ₁	30	w ₀	I #	$E_1 \rightarrow w_0 T \{GEQ(w_0)\} E_1$
10	#E ₁ GEQ(-)Tw ₀	30	w ₀	I #	POP(w_0)
11	#E ₁ GEQ(-)T	30	I	#	$T \rightarrow F T_1$
12	#E ₁ GEQ(-)T ₁ F	30, 5	I	#	PUSH(5), POP(F)
13	#E ₁ GEQ(-)T ₁	30, 5	#		POP(T_1)
14	#E ₁ GEQ(-)	30, 5	#		产生四元式(- 30, 5, c), 语法栈 POP(GEQ(-)), 语义栈POP(5), POP(30), PUSH(25)
15	#E ₁	25	#		POP(E_1)
16	#		#		OK

实验感想

此次实验在第二次实验的语法分析器的基础上加了属性翻译文法构造技术，使之变成一个语义分析器，可以在语义层面上分析句子，对程序的语义做出解释。对于算术表达式，则是检查算术表达式是否合法，以及产生程序可以计算的中间表达式。

github链接: <https://github.com/aZhiChen/hw.git>

