编译原理实验二

——语法分析器——

19335253 葉珺明

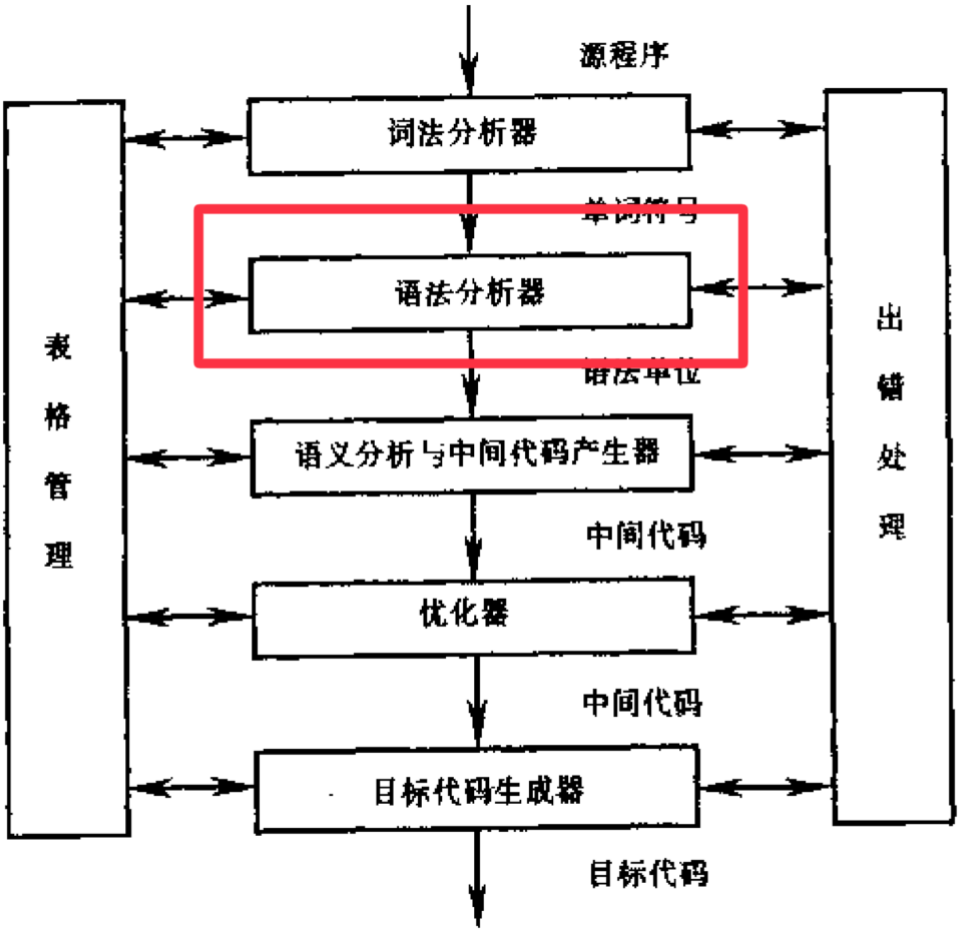
目录

所有文件源码地址： **click here**

一 实验目的

- 设计至少支持加减乘除以及括号操作的算术表达式语法分析器
- 实现词法分析器功能：输入源程序，输出单词符号
- 掌握LL(1)分析法和LR(0)分析法

编译程序的总体处理过程：红色框选部分为本次实验要求实现的部分。



二 实验要求

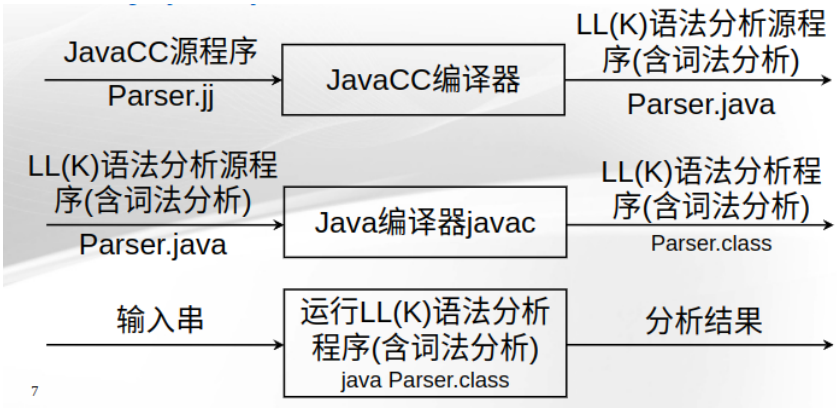
- 用**LL(1)分析法**和**LR(0)分析法**两种方法设计实现算术表达式的语法分析器

三 实验设计

> 使用JavaCC工具产生语法分析器

3.1 JavaCC工具

- LEX工具下词法分析器自动产生流程：



- JavaCC输入源程序构成格式如下：

```
options {
    JavaCC的选项
}
PARSER_BEGIN(解析器类名)
package 包名;
import 库名;
public class 解析器类名 {
    任意的Java代码
}
PARSER_END(解析器类名)
扫描器的描述
解析器的描述
```

3.2 算术表达式LL(1)文法

3.2.1 LL(1)文法设计

- 算术表达式的文法：

$$E \to T|E+T|E-T$$
$$T \to F|T*F|T/F$$
$$F \to id|-F|(E)$$

其中id代表数字或字母

- 消除上述文法的左递归

$$
\begin{aligned}
&E \to TE' \\
&E' \to +TE'|-TE'|\epsilon \\
&T \to FT' \\
&T' \to *FT'|/FT'|\epsilon \\
&F \to id|-F|(E)
\end{aligned}
$$

其中id代表数字或字母

- 构造LL(1)的预测分析表
  - FIRST集和FOLLOW集：

$$
\begin{aligned}
&FISRT(E) = \{id, (\} && FOLLOW(E) = \{\#,)\} \\
&FISRT(E') = \{+, -, \epsilon\} && FOLLOW(E') = \{\#,)\} \\
&FISRT(T) = \{id, (\} && FOLLOW(T) = \{\#, +, -,)\} \\
&FISRT(T') = \{*, /, \epsilon\} && FOLLOW(T') = \{\#, +, -,)\} \\
&FISRT(F) = \{id, -, \ (\} && FOLLOW(F) = \{\#, +, -, *, /,)\}
\end{aligned}
$$

  - 判断是否为LL(1)文法：

> G的任意两个具有相同左部的产生式A—>α|β 满足下列条件：
>
> （1）如果α、β均不能推导出ε，则 FIRST(α) ∩ FIRST(β) = ∅。
>
> （2）α 和 β 至多有一个能推导出 ε。
>
> （3）如果 β *=> ε，则 FIRST(α) ∩ FOLLOW(A) = ∅。

$$
\begin{aligned}
&E' \to +TE'|-TE'|\epsilon: \\
&FIRST(+TE') \cap FIRST(-TE') = \emptyset \\
&FIRST(+TE') \cap FOLLOW(E') = \emptyset \\
&FIRST(-TE') \cap FOLLOW(E') = \emptyset \\
&\_\_\_\_\_ \\
&T' \to *FT'|/FT'|\epsilon: \\
&FIRST(*FT') \cap FIRST(/FT') = \emptyset \\
&FIRST(*FT') \cap FOLLOW(T') = \emptyset \\
&FIRST(/FT') \cap FOLLOW(T') = \emptyset
\end{aligned}
$$

    故文法是LL(1)文法，有：

	id	$+$	$-$	$*$	$/$	$($	$)$	$\#$
E	$E \to TE'$					$E \to TE'$		
E'		$E' \to +TE'$	$E' \to -TE'$				$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$					$T \to FT'$		
T'				$T' \to *FT'$	$T' \to *FT'$		$T' \to \epsilon'$	$T' \to \epsilon'$
F	$F \to id$	$F \to -F$				$F \to (E)$		

3.2.2 LL(1)递归子程序

- 子程序E:

```
PROCEDURE E:
BEGIN
    T;  E';
END
```

- 子程序E':

```
PROCEDURE E':
BEGIN
    IF SYM = '+' OR SYM = '-' THEN
    BEGIN
        ADVANCE;
        T;  E';
    END
END
```

- 子程序T:

```
PROCEDURE T:
BEGIN
    F;  T';
END
```

- 子程序T':

```
PROCEDURE T'
BEGIN
    IF SYM = '*' OR SYM = '/' THEN
    BEGIN
        ADVANCE;
        F;  T';
    END
END
```

- 子程序F:

```
 PROCEDURE F:
 BEGIN
     IF SYM = id THEN
     BEGIN
         ADVANCE;
     END
     ELSE ID SYM = '-' THEN
     BEGIN
         ADVANCE;
         F;
     END
     ELSE IF SYM = '(' THEN
     BEGIN
         ADVANCE;
         E;
         IF SYM = ')' THEN
         BEGIN
             ADVANCE;
         END
```

```
            ELSE ERROR
        END
    END
```

3.3 算术表达式LR(0)文法

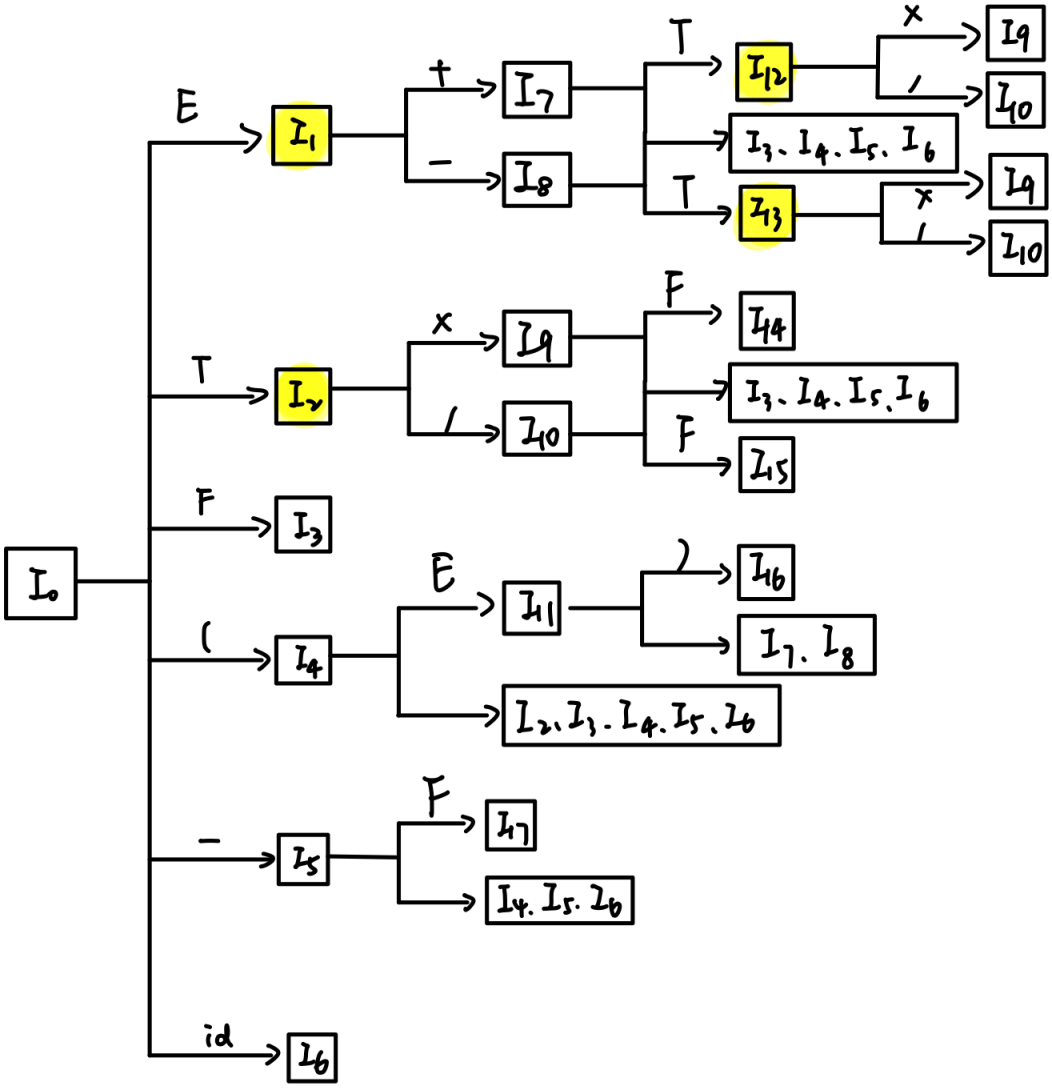
3.3.1 LR(0)文法

- 算术表达式经过文法增广：

$$
\begin{aligned}
&(1)\ E' \to E\\
&(2)\ E \to E+T\\
&(3)\ E \to E-T\\
&(4)\ E \to T\\
&(5)\ T \to T*F\\
&(6)\ T \to T/F\\
&(7)\ T \to F\\
&(8)\ F \to id\\
&(9)\ F \to (E)\\
&(10)\ F \to -F
\end{aligned}
$$

其中id代表数字或字母

- 活前缀DFA：



$I_0:$
$E' \to \cdot E$
$E \to \cdot E + T$
$E \to \cdot E - T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot T / F$
$T \to \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$T \to \cdot - F$

$I_1:$
$E' \to E \cdot$
$E \to E \cdot + T$
$E \to E \cdot - T$
$I_2:$
$E \to T \cdot$
$T \to T \cdot * F$
$T \to T \cdot / F$
$I_3:$
$T \to F \cdot$

$I_4:$
$F \to (\cdot E)$
$E \to \cdot E + T$
$E \to \cdot E - T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot T / F$
$T \to \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$T \to \cdot - F$

$I_5:$
$F \to - \cdot F$
$I_6:$
$F \to id \cdot$
$I_7:$
$E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot T / F$
$T \to \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$F \to \cdot - F$

$I_8:$
$E \to E - \cdot T$
$T \to \cdot T * F$
$T \to \cdot T / F$
$T \to \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$F \to \cdot - F$

$I_9:$
$T \to T * \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$F \to \cdot - F$
$I_{10}:$
$T \to T / \cdot F$
$F \to \cdot id$
$F \to \cdot (E)$
$F \to \cdot - F$

$I_{11}:$
$F \to (E \cdot)$
$E \to E \cdot + T$
$E \to E \cdot - T$
$I_{12}:$
$E \to E + T \cdot$
$T \to T \cdot * F$
$T \to T \cdot / F$
$I_{17}:$
$F \to -F \cdot$

$I_{13}:$
$E \to E - T \cdot$
$T \to T \cdot * F$
$T \to T \cdot / F$
$I_{14}:$
$T \to T * F \cdot$
$I_{15}:$
$T \to T / F \cdot$
$I_{16}:$
$F \to (E) \cdot$

其中I_1、I_2、I_{12}、I_{13}存在移进——归约冲突，采用$SLR(1)$方法解决冲突。$SLR(1)$分析表如下:

状态	ACTION								GOTO		
	+	-	*	/	(	)	id	#	E	T	F
0		S5			S4		S6		1	2	3
1	S7	S8						acc			
2	r4	r4	S9	S10		r4		r4			
3	r7	r7	r7	r7		r7		r7			3
4		S5			S4		S6		11	2	3
5		S5			S4		S6				17
6	r8	r8	r8	r8		r8		r8			
7		S5			S4		S6			12	3
8		S5			S4		S6			13	3
9		S5			S4		S6				14
10		S5			S4		S6				15
11	S7	S8				S16					
12	r2	r2	S9	S10		r2		r2			
13	r3	r3	S9	S10		r3		r3			
14	r5	r5	r5	r5		r5		r5			
15	r6	r6	r6	r6		r6		r6			
16	r9	r9	r9	r9		r9		r9			
17	r10	r10	r10	r10		r10		r10			

四 实验代码

4.1 利用JavaCC实现LL(1)文法

options块和Class块：

```
// options块
options {
    STATIC = false;
}
// Class声明块
PARSER_BEGIN(IsAriExp)
    import java.io.PrintStream ;
    class IsAriExp {
        public static void main( String[] args )
            throws ParseException, TokenMgrError, NumberFormatException {
            IsAriExp parser = new IsAriExp( System.in ) ;
            parser.Start( System.out ) ;
        }
    }
PARSER_END(IsAriExp)
```

4.1.1 词法描述文件

扫描器描述部分，声明token，匹配源文件中的字符

```
SKIP: { " " }   // 跳过空格
TOKEN: { < EOL : "\n" | "\r" | "\r\n" > }   // 处理换行
TOKEN: { < PLUS : "+" > }
TOKEN: { < MINUS : "-" > }
TOKEN: { < TIMES : "*" > }
TOKEN: { < DIVIDE : "/" > }
TOKEN: { < OPEN_PAR : "(" > }
TOKEN: { < CLOSE_PAR : ")" > }
TOKEN: { < ID : <DIGITS>
                |<DIGITS>"."<DIGITS>
                |<DIGITS>"."
                |"."<DIGITS>
                |<LETTER>(<LETTER>|<DIGITS>)* >}    // 变量可由数字或字母表示
TOKEN: { < #DIGITS : (["0"-"9"])+ > }
TOKEN: { < #LETTER : (["A"-"Z","a"-"z","_"])+ > }
```

4.1.2 语法描述文件

对解析器的描述，按照文法的表达式改写成函数：

```
//PROCEDURE MAIN
void Start(PrintStream printStream) throws NumberFormatException :
{}
{
    (
        E()
        <EOL> { printStream.println( "TRUE" ) ; }
```

```
    )*
    <EOF>
}

// PROCEDURE E
void E() throws NumberFormatException :
{}
{
    T()
    E1()
}

// PROCEDURE E'
void E1() throws NumberFormatException :
{}
{
    (
        <PLUS> (
            T()
            E1()
        )
        | <MINUS> (
            T()
            E1()
        )
    )?
}

//PROCEDURE T
void T() throws NumberFormatException :
{}
{
    F()
    T1()
}

// PROCEDURE T1
void T1() throws NumberFormatException :
{}
{
    (
        <TIMES> (
            F()
            T1()
        )
        | <DIVIDE> (
            F()
            T1()
        )
    )?
}

// PROCEDURE F
void F() throws NumberFormatException :
{}
{
    <ID>
    | <OPEN_PAR> E() <CLOSE_PAR>
```

```
66        | <MINUS> F()
67    }
```

4.1.3 编译步骤

```
1   $ javacc parser.jj
2   $ javac *.java
3   $ java parser < test.txt
```

4.2 利用LEX和python实现LR(0)文法

- 通过LEX生成tokens.txt
- 将tokens.txt作为语法分析器的输入
- 建立文法索引用于归约和移位；建立SLR分析表用于状态转换
- 核心代码，以输入token为‘+’为例：

```
1     if token == '+' and AG[st][index['+']] != '=':
2         if next < 0:
3            # 归约
4            for j in range(len(GRAMMAR[next])):
5                sym.pop()
6                status.pop()
7         sym.push(index[next])
8         st = status.peek()
9         nextt = sym.peek()
10        status.push(AG[st][index[nextt]])
11        i = i - 1
12        else:
13           # 移位
14           sym.push(token)
15           status.push(next)
```

五 实验结果

5.1 LL(1)文法实验结果

测试文件为$source.c$，内容是算术表达式。

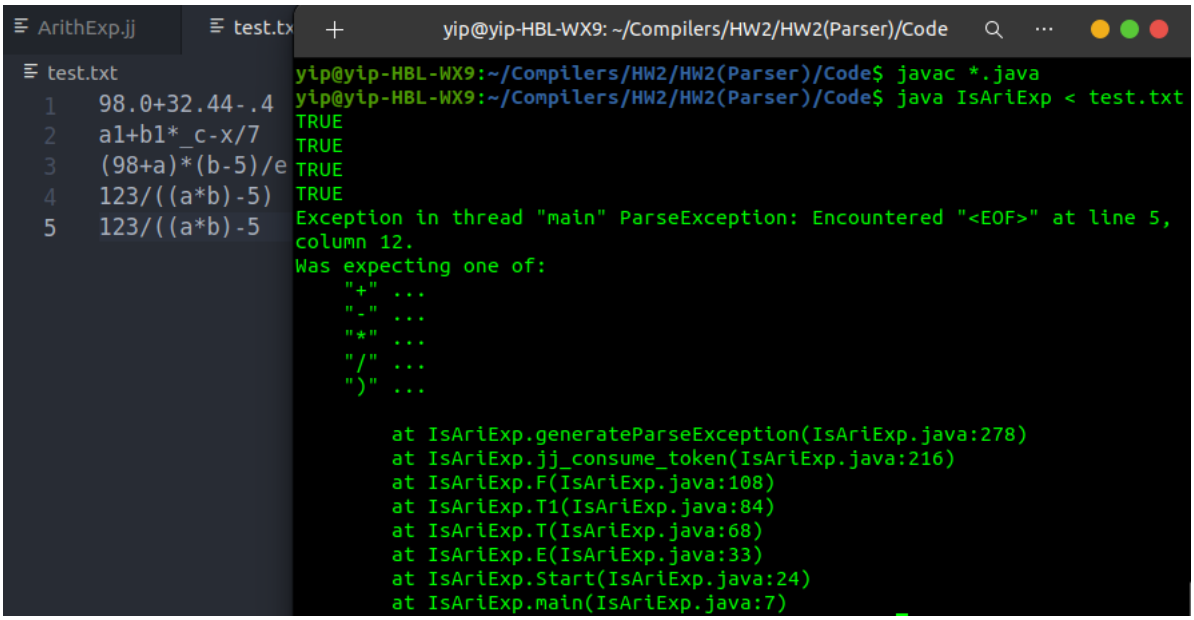
输入$source.c$，语法分析器直接输出分析结果

实验结果为：

- 测试代码：

```
1   98.0+32.44-.4
2   a1+b1*_c-x/7
3   (98+a)*(b-5)/e
4   123/((a*b)-5)
5   123/((a*b)-5
```

- 终端输出与$source.c$的表达式对应：



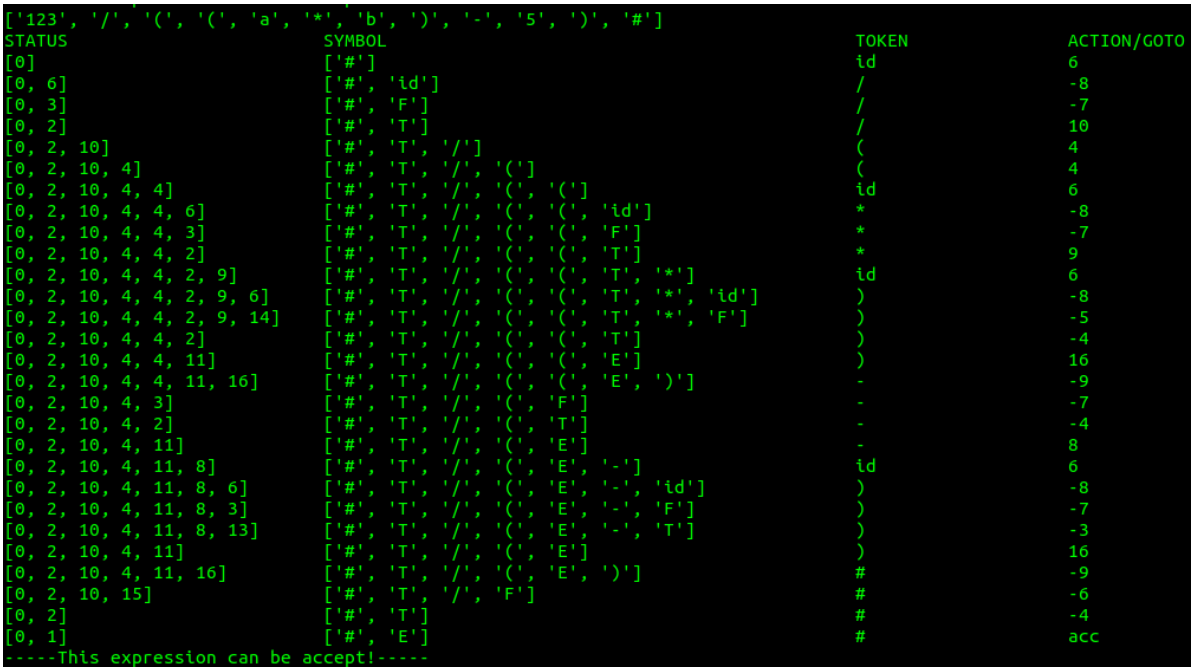
- 分析：比较可以发现，前四项表达式能够被判断为$TRUE$，在第5行的算术表达式第12列即末尾检测到$< EOF >$的token，但此处所希望的输入应为$+, -, *, /,)$中的一个而不是换行符，因此该表达式不是我们所定义的算术表达式，没有输出$TRUE$。

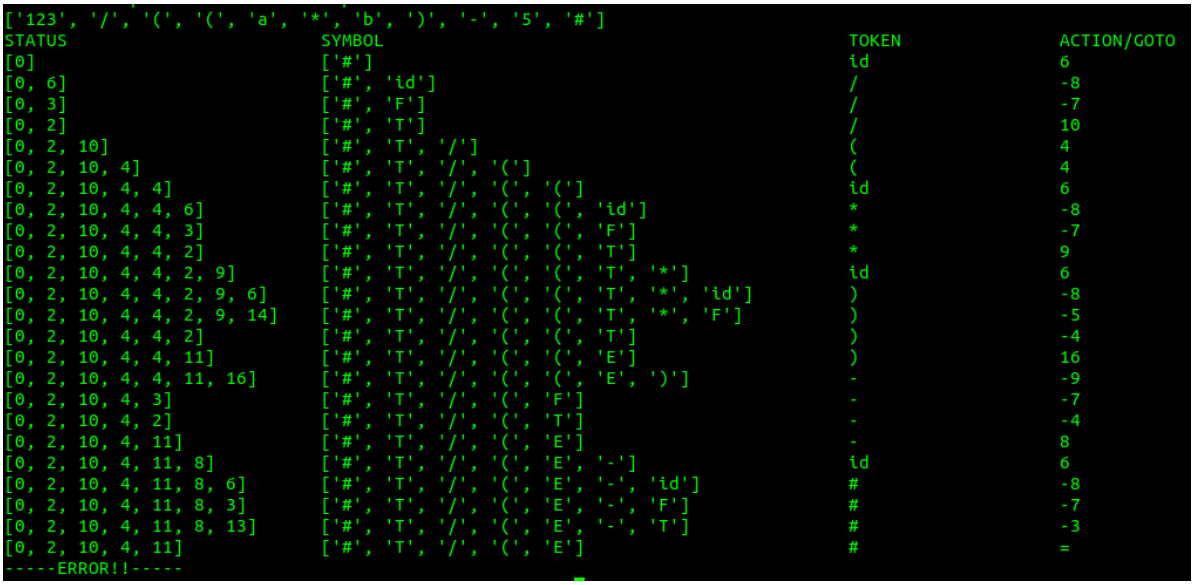
5.2 LR(0)文法实验结果

测试文件为$tokens.txt$，内容是算术表达式的$tokens$。

读取文件$tokens.txt$，语法分析器直接输出分析结果：

- 判断算术表达式：





- 分析：第一行代表需要分析的tokens，需要判断的的表达式相似，第二个表达式缺少右括号，ACTION/GOTO中正数代表移位操作，负数代表归约操作，通过字典索引。如实验结果的图2，在进行括号的归约操作时，ACTION/GOTO跳转的到分析表的‘=’，代表没有找到状态跳转，故输出ERROR。

六 实验心得

本次实验中，主要是实现语法分析器的功能，在此之前需要熟悉语法分析的LL(1)和LR(0)文法。在LL(1)文法中，重点为消除左递归的消除，以及寻找FIRST集和FOLLOW集，建立LL(1)分析表；在LR(0)文法中，需要对文法进行增广，然后寻找状态的转换即活前缀DFA，再建立LR(0)分析表。

本次实现借助JavaCC工具实现LL(1)，难度不高。借助该工具能够正确输出表达式的判断，错误表达式的错误出现在哪。通过python实现LR(0)文法，主要难点在进行归约后状态的跳转，此步骤需要花费较长时间整理清晰。总体通过本次实验，加深了对这两种文法的理解。

附录A

A.1 LL1文法实现代码：

```
// Option块

options {
    STATIC = false;
}

// Class声明块
PARSER_BEGIN(IsAriExp)
    import java.io.PrintStream ;
    class IsAriExp {
        public static void main( String[] args )
            throws ParseException, TokenMgrError, NumberFormatException {
            IsAriExp parser = new IsAriExp( System.in ) ;
            parser.Start( System.out ) ;
        }
    }
PARSER_END(IsAriExp)

SKIP: { " " }
TOKEN: { < EOL : "\n" | "\r" | "\r\n" > }
TOKEN: { < PLUS : "+" > }
TOKEN: { < MINUS : "-" > }
TOKEN: { < TIMES : "*" > }
TOKEN: { < DIVIDE : "/" > }
TOKEN: { < OPEN_PAR : "(" > }
TOKEN: { < CLOSE_PAR : ")" > }
TOKEN: { < ID : <DIGITS>
                |<DIGITS>"."<DIGITS>
                |<DIGITS>"."
                |"."<DIGITS>
                |<LETTER>(<LETTER>|<DIGITS>)* >}
TOKEN: { < #DIGITS : (["0"-"9"])+ > }
TOKEN: { < #LETTER : (["A"-"Z","a"-"z","_"])+ > }


//PROCEDURE MAIN
void Start(PrintStream printStream) throws NumberFormatException :
{}
{
    (
        E()
        <EOL> { printStream.println( "TRUE" ) ; }
    )*
    <EOF>
}


// PROCEDURE E
void E() throws NumberFormatException :
{}
{
```

```
    T()
    E1()
}

// PROCEDURE E'
void E1() throws NumberFormatException :
{}
{
    (
        <PLUS> (
            T()
            E1()
        )
        | <MINUS> (
            T()
            E1()
        )
    )?
}

//PROCEDURE T
void T() throws NumberFormatException :
{}
{
    F()
    T1()
}

// PROCEDURE T1
void T1() throws NumberFormatException :
{}
{
    (
        <TIMES> (
            F()
            T1()
        )
        | <DIVIDE> (
            F()
            T1()
        )
    )?
}

// PROCEDURE F
void F() throws NumberFormatException :
{}
{
    <ID>
    | <OPEN_PAR> E() <CLOSE_PAR>
    | <MINUS> F()
}
```

A.2 LL2文法实现代码：

```
import re

AG = [
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  ,  '=' ,  1  ,  2  ,  3 ],
    [  7  ,  8  , '=' , '=' , '=' , '=' , '=' , 'acc', '=' , '=' , '='],
    [ -4  , -4  ,  9  , 10  , '=' , -4  , '=' ,  -4  , '=' , '=' , '='],
    [ -7  , -7  , -7  , -7  , '=' , -7  , '=' ,  -7  , '=' , '=' ,  3 ],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  ,  11 ,  2  ,  3 ],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  , '=' , '=' , 17 ],
    [ -8  , -8  , -8  , -8  , '=' , -8  , '=' , -8   , '=' , '=' , '='],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  , '=' ,  12 ,  3 ],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  , '=' ,  13 ,  3 ],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  , '=' , '=' , 14 ],
    [ '=' ,  5  , '=' , '=' ,  4  , '=' ,  6  , '='  , '=' , '=' , 15 ],
    [  7  ,  8  , '=' , '=' , '=' ,  16 , '=' , '='  , '=' , '=' , '='],
    [ -2  , -2  ,  9  , 10  , '=' , -2  , '=' , -2   , '=' , '=' , '='],
    [ -3  , -3  ,  9  , 10  , '=' , -3  , '=' , -3   , '=' , '=' , '='],
    [ -5  , -5  , -5  , -5  , '=' , -5  , '=' , -5   , '=' , '=' , '='],
    [ -6  , -6  , -6  , -6  , '=' , -6  , '=' , -6   , '=' , '=' , '='],
    [ -9  , -9  , -9  , -9  , '=' , -9  , '=' , -9   , '=' , '=' , '='],
    [ -10 , -10 , -10 , -10 , '=' , -10 , '=' , -10  , '=' , '=' , '='],
]


GRAMMAR = { -1 :'E' , -2 : 'E+T' , -3 : 'E-T' , -4 : 'T' ,
            -5 : 'T*F' , -6 : 'T/F', -7 : 'F' , -8 : 'i' ,
            -9 : '(E)' , -10 : '-F' }

index = {'+': 0 , '-': 1 , '*': 2 ,  '/': 3 ,
         '(': 4 , ')': 5 , 'id': 6 , '#': 7 ,
         'E': 8 , 'T': 9 , 'F': 10 ,
         -1 : 'E1', -1 : 'E', -2 : 'E',
         -3 : 'E', -4 : 'E', -5 : 'T',
         -6 : 'T', -7 : 'T', -8 : 'F',
         -9 : 'F', -10 : 'F'}


exps = []

class Stack:          # 定义一个栈
    def __init__(self):    # 初始化栈为空列表
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self,item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]
```

```
56
57      def size(self):
58          return len(self.items)
59
60      def all(self):
61          return self.items
62
63
64  def get_tokens(filepath):
65      file = open(filepath, 'r')
66      line = file.readlines()
67      exp = []
68      for i in line:
69          i = i.strip()
70          exp.append(i)
71          if i == '#':
72              exps.append(exp)
73              exp = []
74
75
76  def LR():
77      for line in exps:
78          status = Stack()
79          sym = Stack()
80          status.push(0)
81          sym.push('#')
82          i = 0
83          print(line)
84          print('%-30s%-50s%-20s%-10s'%('STATUS', 'SYMBOL', 'TOKEN',
    'ACTION/GOTO'))
85          while i < len(line):
86              token = line[i]
87              regular = r'\d+\.\d+|\d+|\w+(\w|\d)*'
88              t = re.fullmatch(regular, token)
89              if t != None:
90                  token = 'id'
91
92              st = status.peek()
93              next = AG[st][index[token]]
94
95              print('%-30s%-50s%-20s%-10s'%(status.all(),sym.all(),token, next))
96              #print(status.all(),'\t', sym.all(),'\t', token)
97
98
99
100             if token == '+' and AG[st][index['+']] != '=':
101                 if next < 0:
102                     # 归约
103                     for j in range(len(GRAMMAR[next])):
104                         sym.pop()
105                         status.pop()
106                     sym.push(index[next])
107                     st = status.peek()
108                     nextt = sym.peek()
109                     status.push(AG[st][index[nextt]])
110                     i = i - 1
111                 else:
112                     # 移位
```

```
                    sym.push(token)
                    status.push(next)

            elif token == '-' and AG[st][index['-']] != '=':
                if next < 0:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()
                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1
                else:
                    sym.push(token)
                    status.push(next)

            elif token == '*' and AG[st][index['*']] != '=':
                if next < 0:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()
                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1
                else:
                    sym.push(token)
                    status.push(next)

            elif token == '/' and AG[st][index['/']] != '=':
                if next < 0:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()
                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1
                else:
                    sym.push(token)
                    status.push(next)

            elif token == '(' and AG[st][index['(']] != '=':
                if next < 0:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()

                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1
                else:
                    sym.push(token)
```

```
                    status.push(next)

            elif token == ')' and AG[st][index[')']] != '=':
                if next < 0:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()
                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1
                else:
                    sym.push(token)
                    status.push(next)

            elif token == 'id' and AG[st][index['id']] != '=':
                    sym.push(token)
                    status.push(next)

            elif token == '#' and AG[st][index['#']] != '=' :
                if next == 'acc':
                    print('-----This expression can be accept!-----')
                else:
                    for j in range(len(GRAMMAR[next])):
                        sym.pop()
                        status.pop()
                    sym.push(index[next])
                    st = status.peek()
                    nextt = sym.peek()
                    status.push(AG[st][index[nextt]])
                    i = i - 1

            else:
                print('-----ERROR!!-----')
            i += 1


get_tokens("tokens.txt")
# print(exps)
LR()
```