目录

[程序说明 2](#_Toc57568354)

[DFA及LR分析表 2](#_Toc57568355)

[不带错误处理的分析程序 2](#_Toc57568356)

[带有错误处理的LR分析程序： 4](#_Toc57568357)

[运行结果 5](#_Toc57568358)

[不带错误处理的LR分析程序 5](#_Toc57568359)

[带错误处理的LR分析程序 6](#_Toc57568360)

[思路 11](#_Toc57568361)

[伪代码 11](#_Toc57568362)

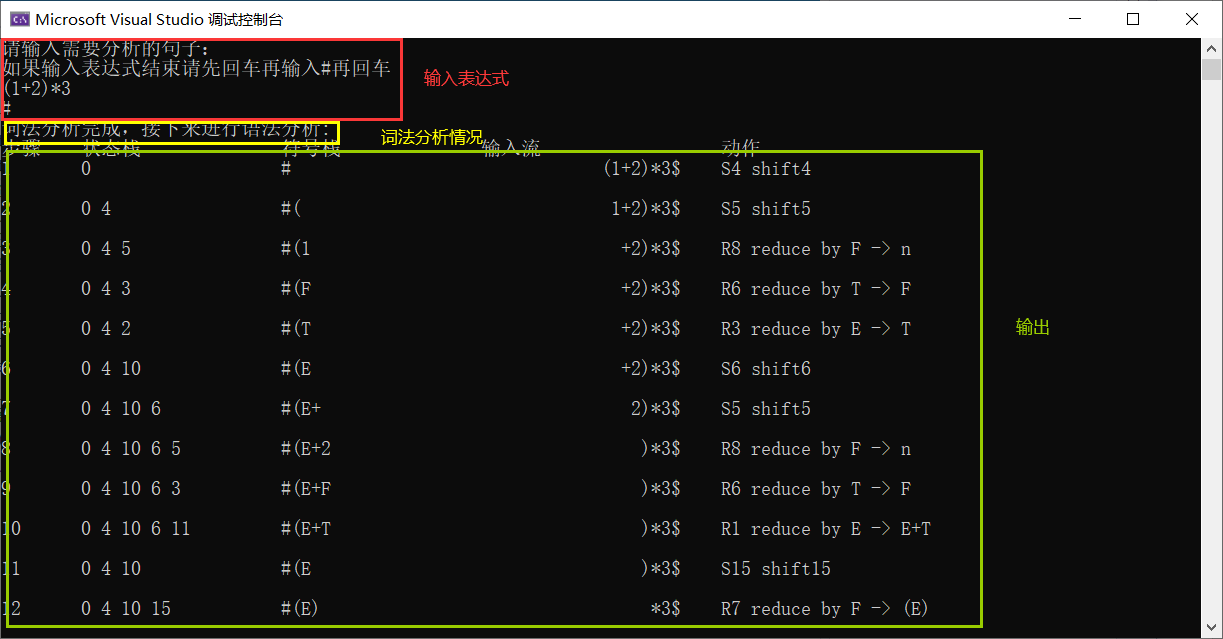
[重要函数解释 12](#_Toc57568363)

[实验心得 16](#_Toc57568364)

# 程序说明

输入格式：如果输入表达式结束请先回车再输入#再回车

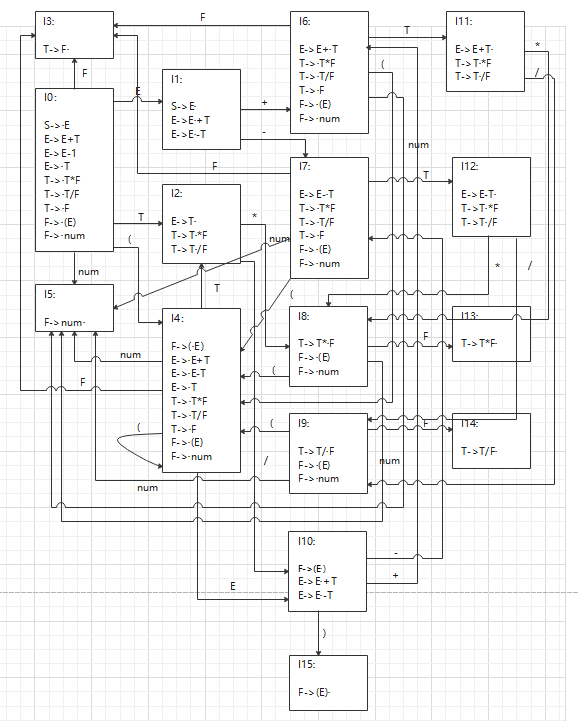
输出格式：



# DFA及LR分析表

不带错误处理的分析程序

识别该文法所有活前缀的DFA



SLR分析表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | + | - | \* | / | ( | ) | num | $ |  | E | T | F |
| I0 |  |  |  |  | S4 |  | S5 |  |  | 1 | 2 | 3 |
| I1 | S6 | S7 |  |  |  |  |  | ACC |  |  |  |  |
| I2 | R3 | R3 | S8 | S9 |  | R3 |  | R3 |  |  |  |  |
| I3 | R6 | R6 | R6 | R6 |  | R6 |  | R6 |  |  |  |  |
| I4 |  |  |  |  | S4 |  | S5 |  |  | 10 | 2 | 3 |
| I5 | R8 | R8 | R8 | R8 |  | R8 |  | R8 |  |  |  |  |
| I6 |  |  |  |  | S4 |  | S5 |  |  |  | 11 | 3 |
| I7 |  |  |  |  | S4 |  | S5 |  |  |  | 12 | 3 |
| I8 |  |  |  |  | S4 |  | S5 |  |  |  |  | 13 |
| I9 |  |  |  |  | S4 |  | S5 |  |  |  |  | 14 |
| I10 | S6 | S7 |  |  |  | S15 |  |  |  |  |  |  |
| I11 | R1 | R1 | S8 | S9 |  | R1 |  | R1 |  |  |  |  |
| I12 | R2 | R2 | S8 | S9 |  | R2 |  | R2 |  |  |  |  |
| I13 | R4 | R4 | R4 | R4 |  | R4 |  | R4 |  |  |  |  |
| I14 | R5 | R5 | R5 | R5 |  | R5 |  | R5 |  |  |  |  |
| I15 | R7 | R7 | R7 | R7 |  | R7 |  | R7 |  |  |  |  |

带有错误处理的LR分析程序：

带有错误处理过程表示的分析表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | + | - | \* | / | ( | ) | num | $ |  | E | T | F |
| I0 | E1 | E1 | E1 | E1 | S4 | E3 | S5 | E1 |  | 1 | 2 | 3 |
| I1 | S6 | S7 | E1 | E1 | E2 | E3 | E2 | ACC |  |  |  |  |
| I2 | R3 | R3 | S8 | S9 | R3 | R3 | R3 | R3 |  |  |  |  |
| I3 | R6 | R6 | R6 | R6 | R6 | R6 | R6 | R6 |  |  |  |  |
| I4 | E1 | E1 | E1 | E1 | S4 | E1 | S5 | E1 |  | 10 | 2 | 3 |
| I5 | R8 | R8 | R8 | R8 | R8 | R8 | R8 | R8 |  |  |  |  |
| I6 | E1 | E1 | E1 | E1 | S4 | E1 | S5 | E1 |  |  | 11 | 3 |
| I7 | E1 | E1 | E1 | E1 | S4 | E1 | S5 | E1 |  |  | 12 | 3 |
| I8 | E1 | E1 | E1 | E1 | S4 | E1 | S5 | E1 |  |  |  | 13 |
| I9 | E1 | E1 | E1 | E1 | S4 | E1 | S5 | E1 |  |  |  | 14 |
| I10 | S6 | S7 | E4 | E4 | E2 | S15 | E2 | E4 |  |  |  |  |
| I11 | R1 | R1 | S8 | S9 | R1 | R1 | R1 | R1 |  |  |  |  |
| I12 | R2 | R2 | S8 | S9 | R2 | R2 | R2 | R2 |  |  |  |  |
| I13 | R4 | R4 | R4 | R4 | R4 | R4 | R4 | R4 |  |  |  |  |
| I14 | R5 | R5 | R5 | R5 | R5 | R5 | R5 | R5 |  |  |  |  |
| I15 | R7 | R7 | R7 | R7 | R7 | R7 | R7 | R7 |  |  |  |  |

其中，错误标识分为四类：

1. 缺少运算对象

恢复：把一个假想的num补入栈

1. 缺少运算符号

恢复：把一个假想的运算符号“+” 补入栈

1. 括号不匹配，有多余右括号

恢复：移出输入串中的多余右括号

1. 缺少右括号

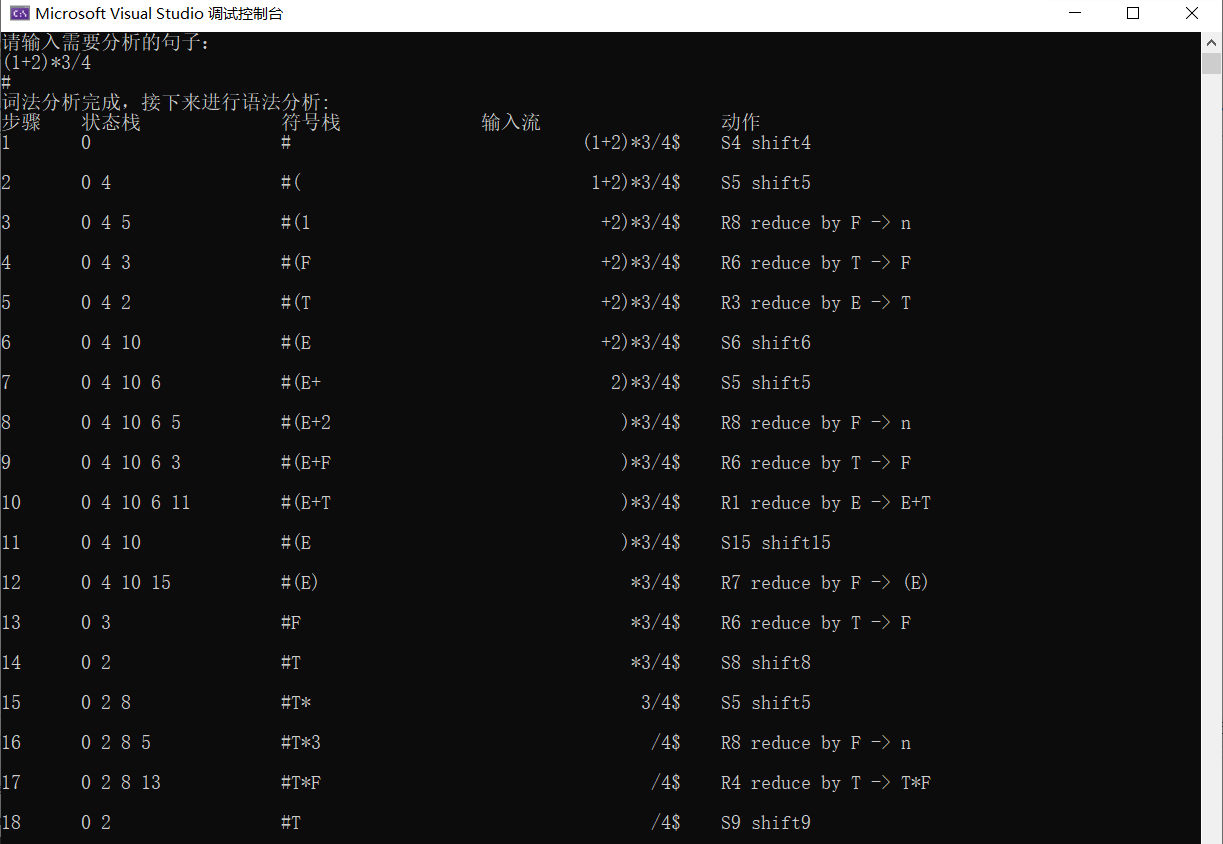
恢复：把一个假想的右括号补入栈

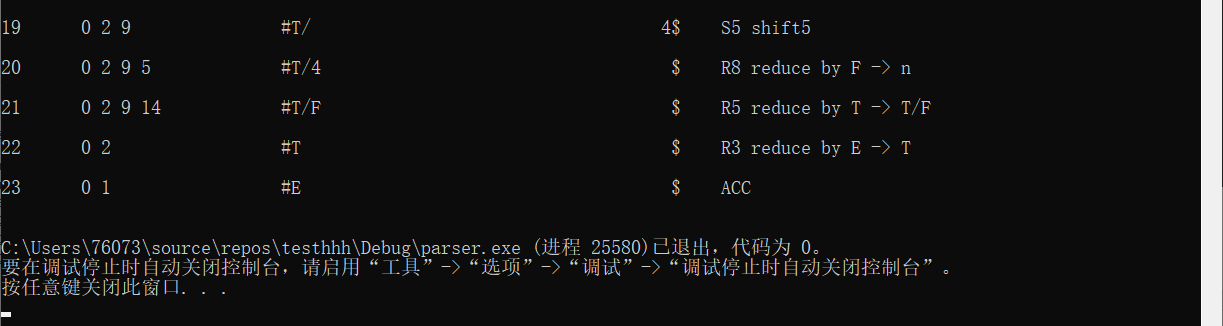
# 运行结果

不带错误处理的LR分析程序

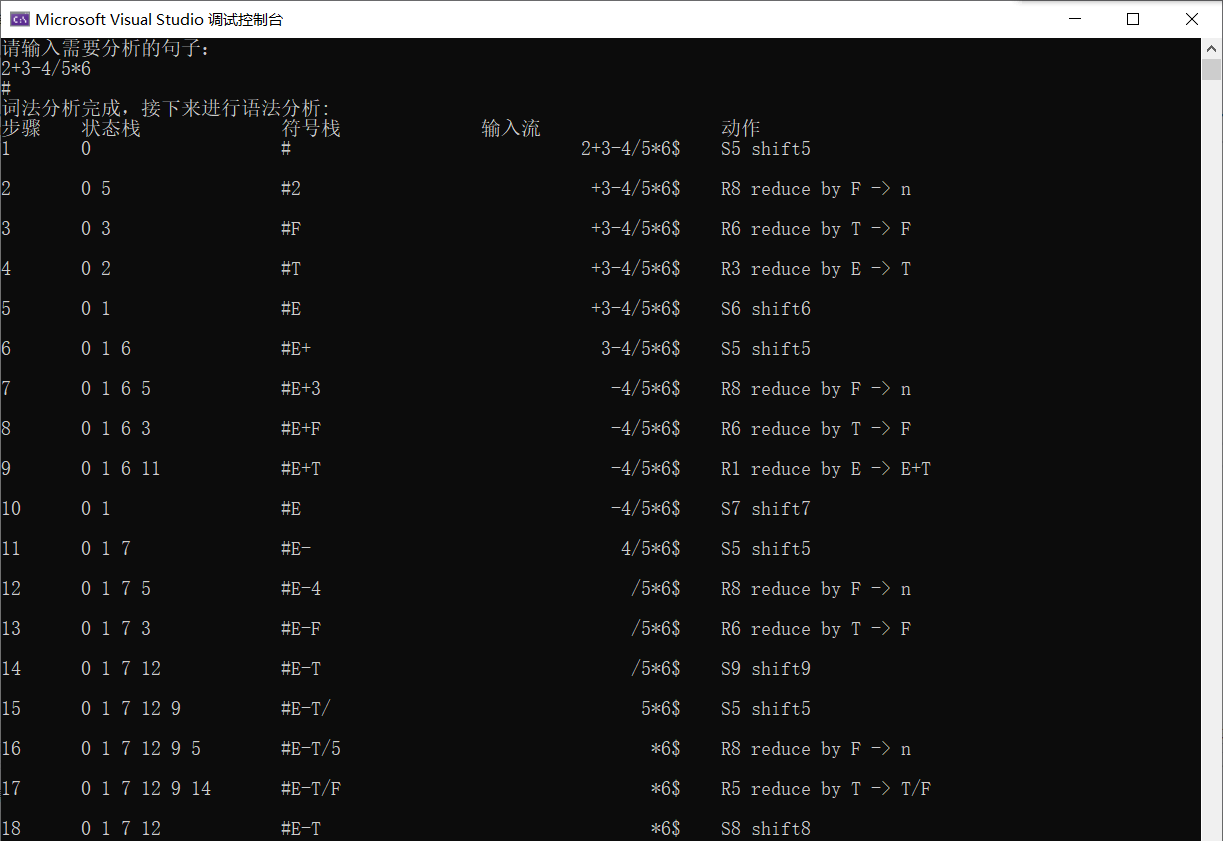
正确识别

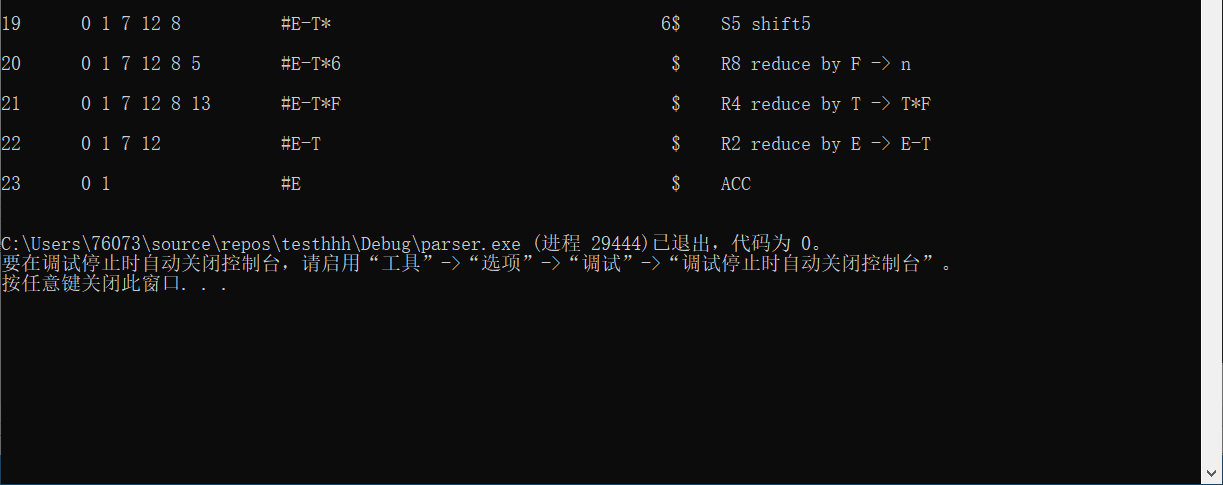
输入1：(1+2)\*3/4





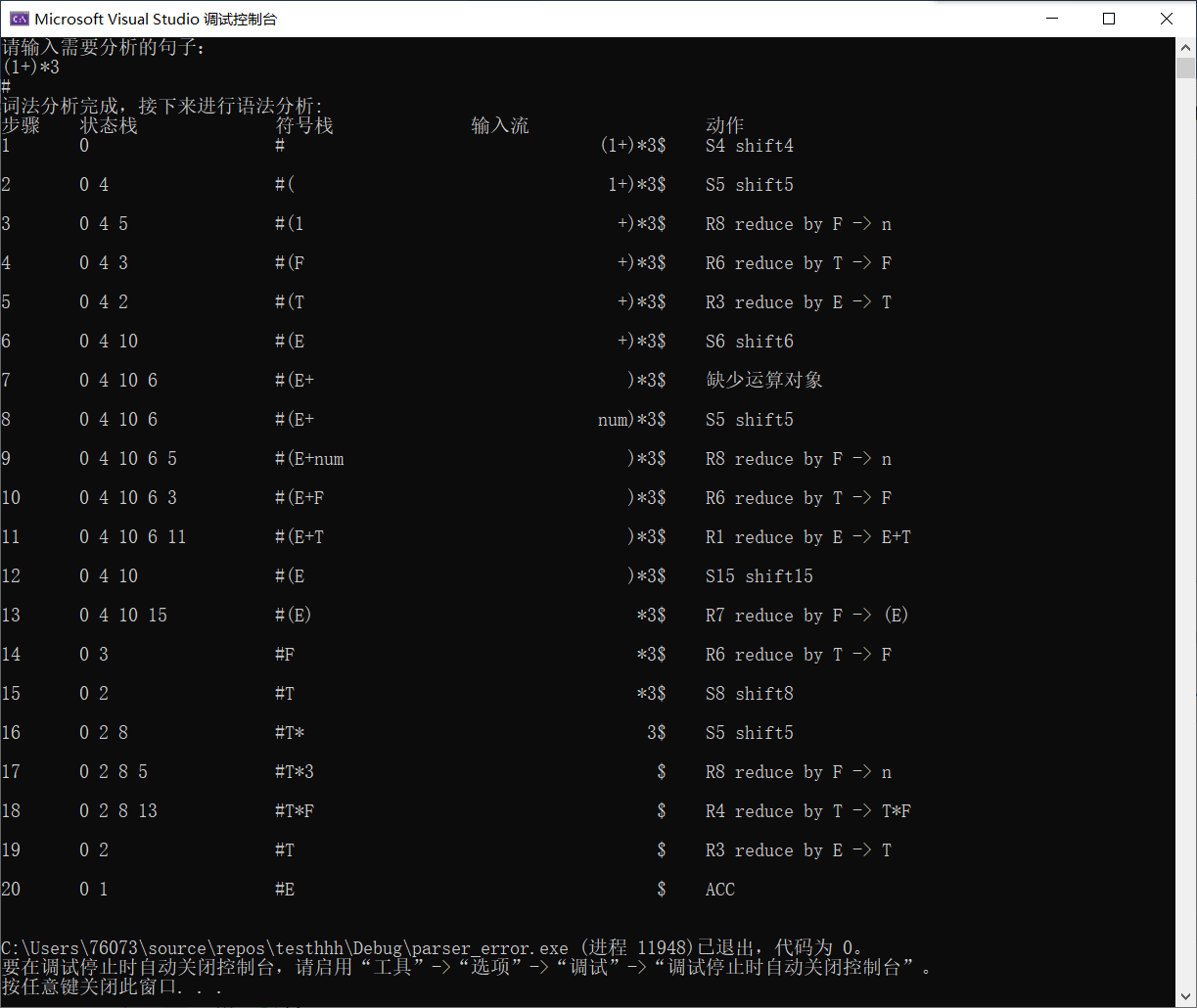
输入2：2+3-4/5\*6



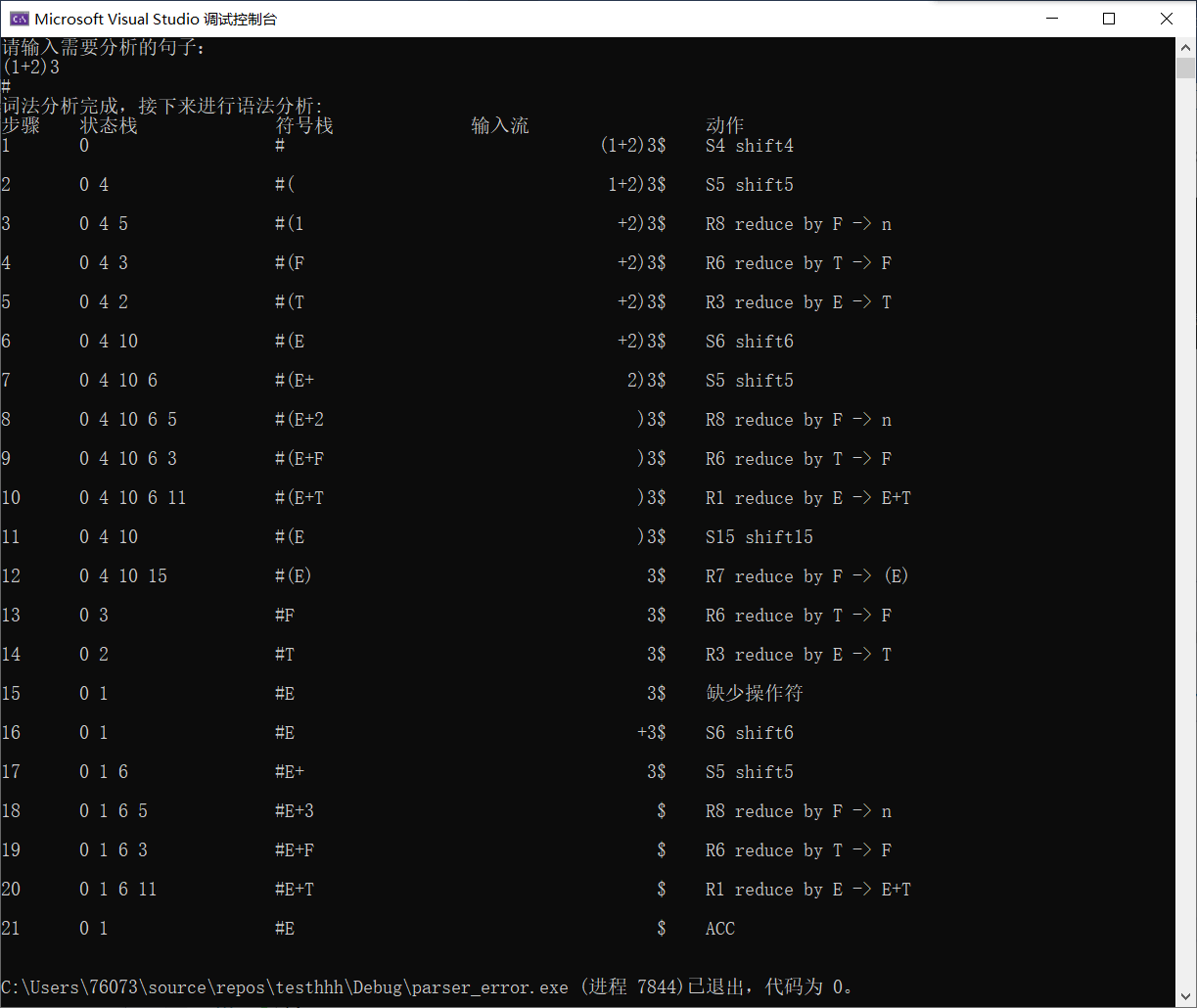


带错误处理的LR分析程序

1.缺少运算对象

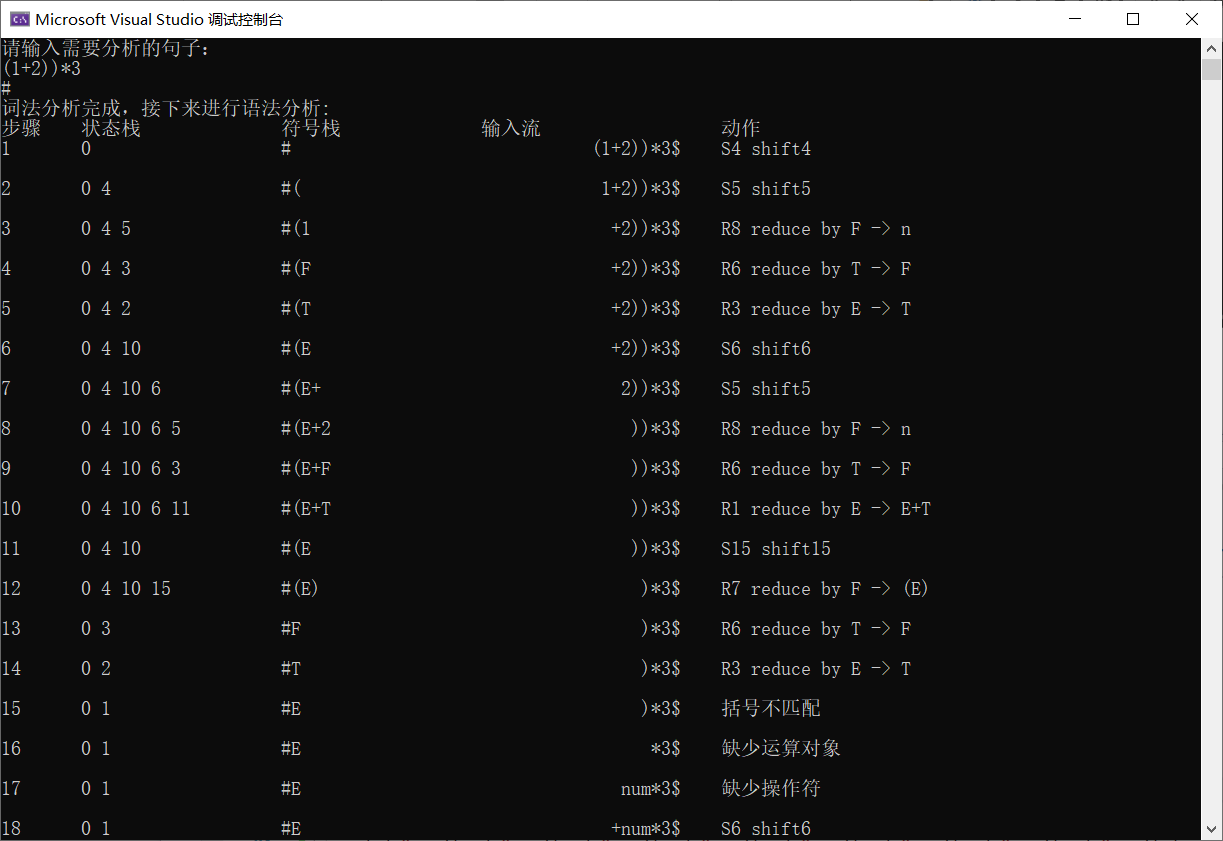


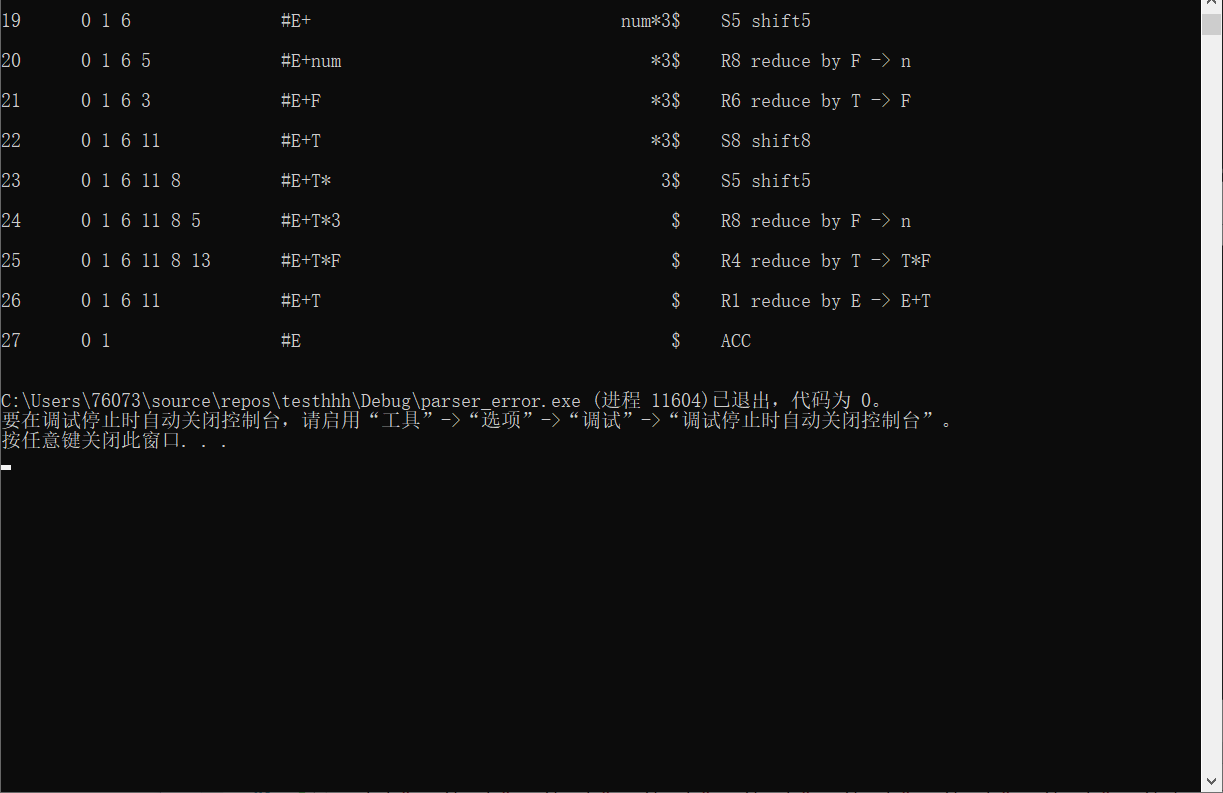
2.缺少操作符



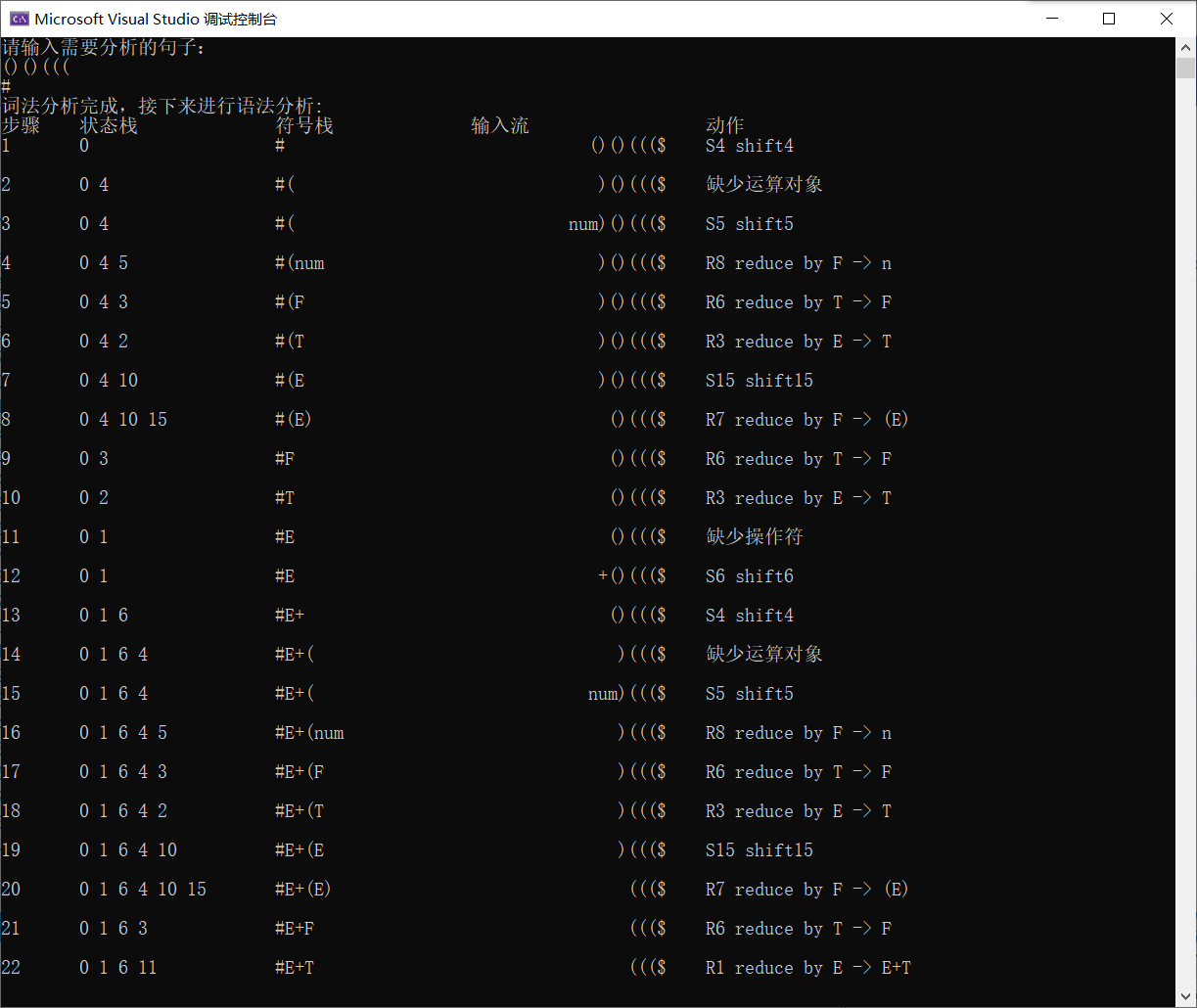
3.括号不匹配，有多余)

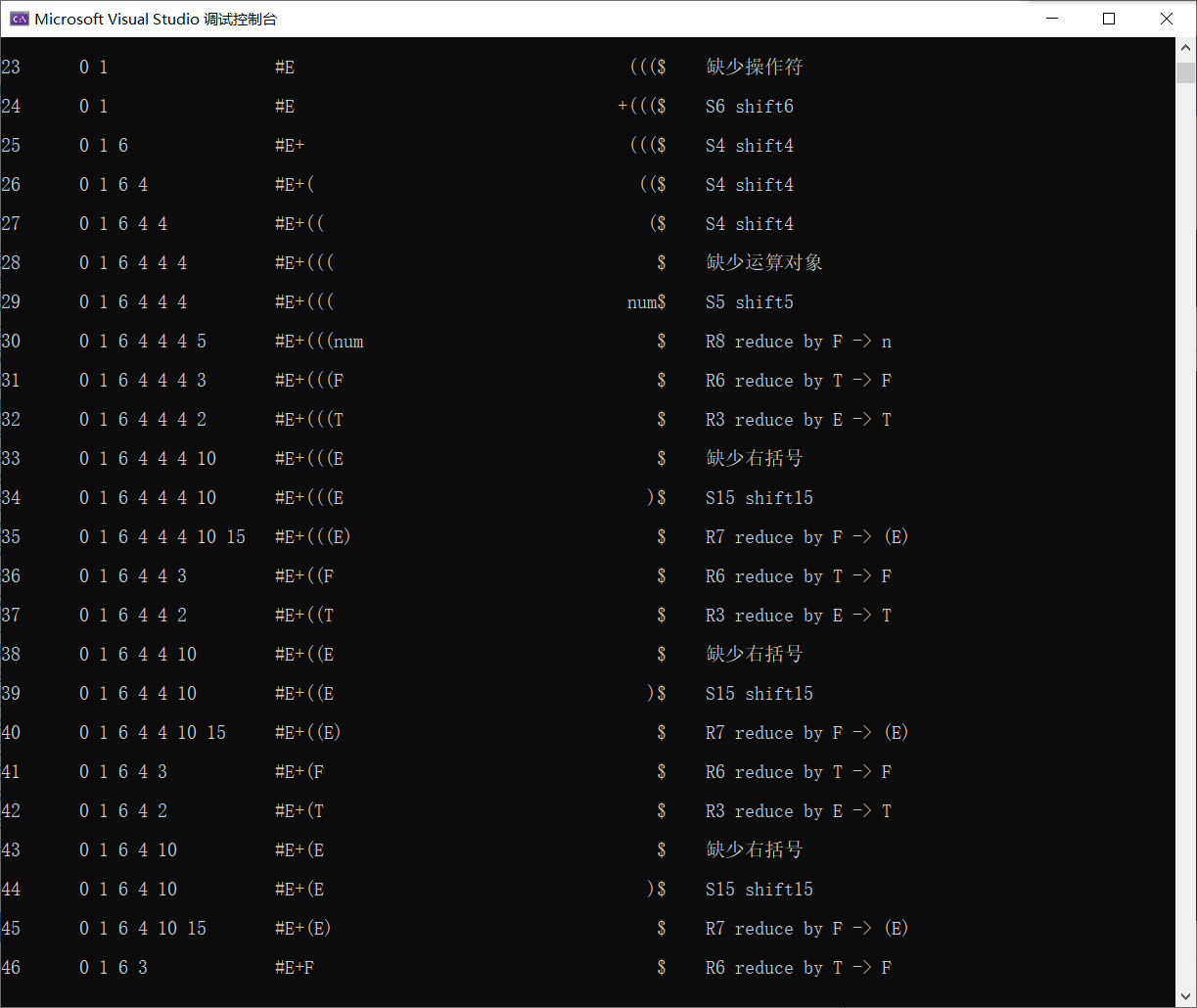
测试1：

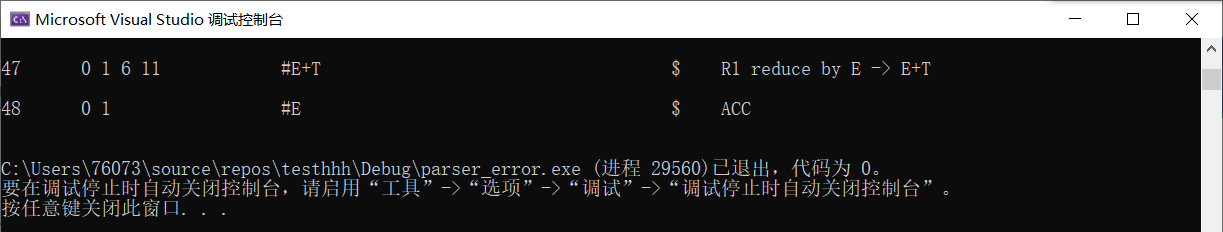




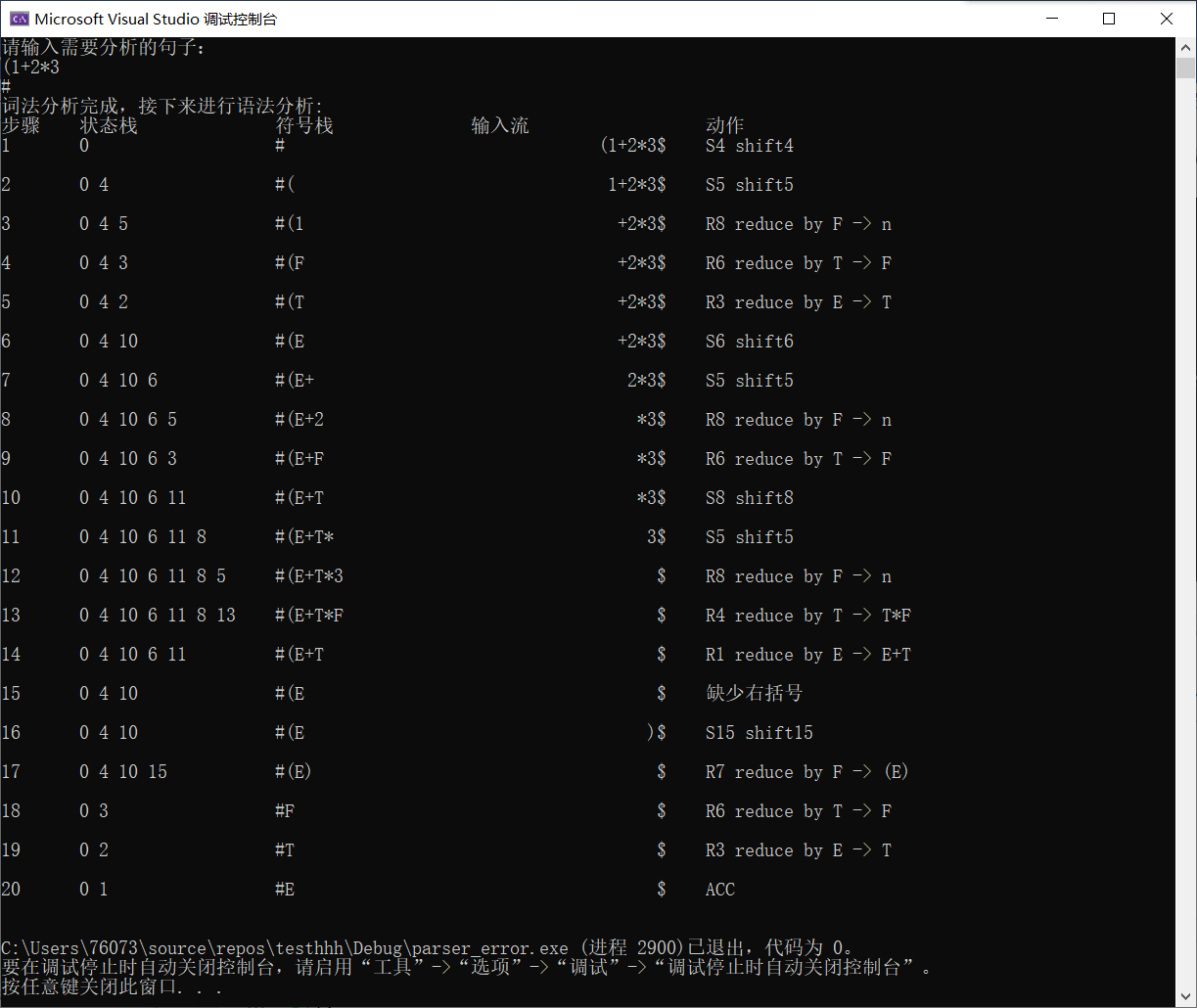
测试2：







4.缺少)



# 思路

伪代码

不带有错误处理的LR分析：

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->p) {

从栈顶弹出|p|个符号和状态；

令S'是当前栈顶状态，把A和goto[S',A]分别压人符号栈和状态栈的栈顶；

输出产生式A->P；

}；

else if (action[S,a]=ACC) return;

else if (action[S,a]=Nl){

出现了未定义的动作

};

else if (action[S,a]=NOTFIND){

出现了所给文法无法识别的符号

};

} while(1);

带有错误处理的LR分析：

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->p) {

从栈顶弹出|p|个符号和状态；

令S'是当前栈顶状态，把A和goto[S',A]分别压人符号栈和状态栈的栈顶；

输出产生式A->P；

}；

else if (action[S,a]=ACC) return;

else if (action[S,a]=Nl){

出现了未定义的动作

};

else if (action[S,a]=NOTFIND){

出现了所给文法无法识别的符号

};

else if (action[S,a]=E){

当前动作为错误标识，根据错误标识的类型，有以下处理：

若错误标识类型为unnecessary，即出现了多余的右括号，则将当前输入串中的多余右括号弹出。若为其余错误标识，则将错误恢复需要用到的symbol加入到当前所指输入串位置即可

};

} while(1);

重要函数解释

int getIndex(string\* array, int size, string symbol)

array为需要被用来匹配的终结符或者非终结符表，size为表的大小，symbol为当前读入的字符。在匹配表中找到当前读入的字符的位置返回下标，如果未找到，证明当前读入字符不是该文法可以识别的符号，返回error

Action getNextAction(int state, Symbol input)

得到动作

通过getIndex函数得到当前输入字符在匹配表中对应的下标index，结合当前栈顶状态state，在分析表中找到所对应的动作为anly\_action[state][index]，如果index==error，则当前输入符号不是该文法可以识别的符号，当前执行动作为手动设置的NOTFIND

int getGoto(int state, Symbol stack)

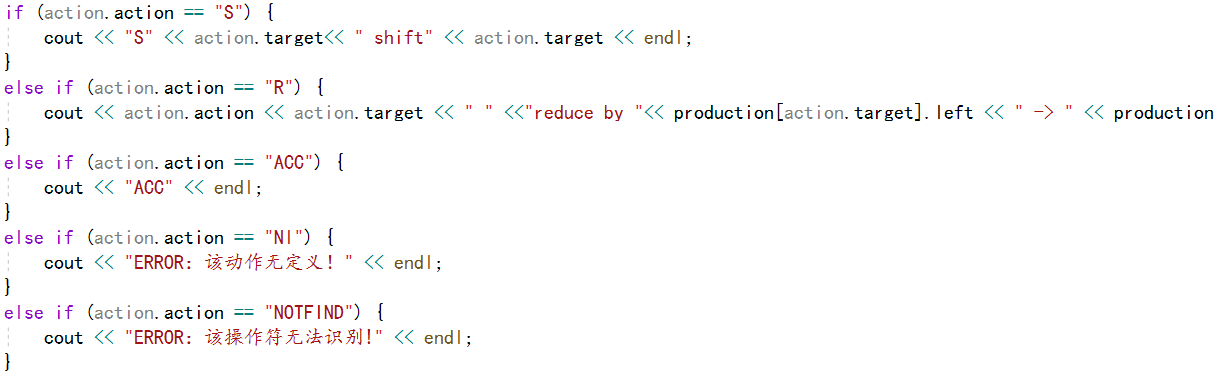
得到规约后的状态。

通过当前状态state和符号栈中栈顶符号在非终结表中的下标index，在goto表中找到需要压入的状态为anly\_goto[state][index]

void printStack(Action action)

打印。

分别需要打印状态栈、符号栈、输入表达式和动作。借助于to\_string函数和输出格式函数setw()进行打印。将栈中的符号输入到string，再将该string打印出来。在打印动作时根据动作的不同action属性进行不同的打印操作

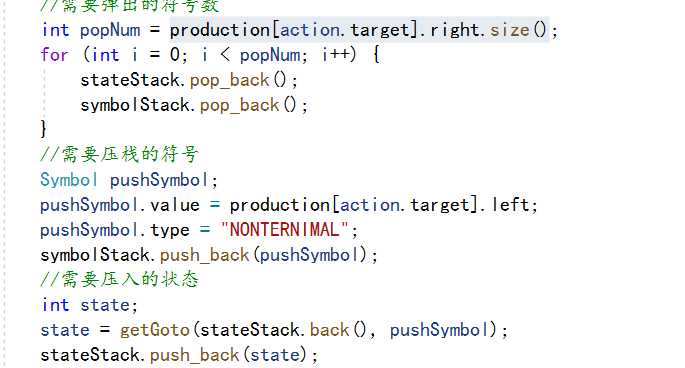


void LR()

分析程序。在每次循环中，首先调用getNextAction函数得到当前需要进行的动作。

如果该动作为S（移进），则将状态压入状态栈，并将输入符号串中的当前字符压入符号栈，在输入符号串中将该符号出栈。

如果该动作为R（规约），则通过production[action.target]得到规约所使用的产生式。production[action.target].right.size()为符号栈和状态栈中应该弹出的元素数目。



在弹出完毕后，将产生式左侧符号压入符号栈，调用getGoto函数得到此时应该压入状态栈的状态号。

如果动作为ACC，该表达式是该文法的句子，return。

如果动作为Nl，则出现了未定义的动作，该句子不属于此文法，return。

如果动作为NOTFIND，则出现了改文法无法识别的符号，该句子不属于此文法，return。

void LR\_error()

带有错误恢复的分析程序

在每次循环中，首先调用getNextAction函数得到当前需要进行的动作。

如果该动作为S（移进），则将状态压入状态栈，并将输入符号串中的当前字符压入符号栈，在输入符号串中将该符号出栈。

如果该动作为R（规约），则通过production[action.target]得到规约所使用的产生式。production[action.target].right.size()为符号栈和状态栈中应该弹出的元素数目。

在弹出完毕后，将产生式左侧符号压入符号栈，调用getGoto函数得到此时应该压入状态栈的状态号。

如果动作为ACC，该表达式是该文法的句子，return。

如果动作为Nl，则出现了未定义的动作，该句子不属于此文法，return。

如果动作为NOTFIND，则出现了改文法无法识别的符号，该句子不属于此文法，return。

如果动作为E，使用getError函数得到错误恢复需要使用到的字符结构。如果该错误恢复块的类型为UNNECESSARY（有多余右括号），则只需要将输入符号串中的该右括号删除。对于其余类型的错误，把错误恢复符号压入栈。对应的错误恢复符号如下：

1.缺少运算对象

恢复：把一个假想的num补入栈

2.缺少运算符号

恢复：把一个假想的运算符号“+” 补入栈

3.缺少右括号

恢复：把一个假想的右括号补入栈

void input\_expression()

输入表达式，当输入#时视为输入结束

void lex\_anly(FILE\* input, FILE\* output, FILE\* error, FILE\* note)

词法分析函数，该函数使用的是上次实验的内容，只是对词法分析输出结果进行了修改，不再赘述

Symbol getError(Action action, int state)

得到错误恢复需要使用的Symbol。

手动生成恢复Symbol的规则如下：

错误标识分为四类：

1.缺少运算对象

恢复：把一个假想的num补入栈，类型为NUMBER

2.缺少运算符号

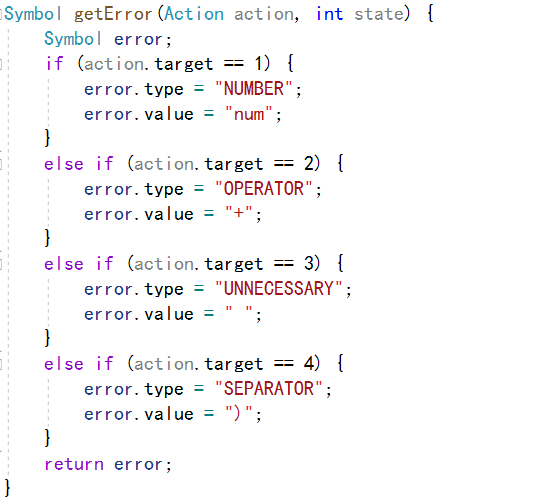
恢复：把一个假想的运算符号“+” 补入栈，类型为OPERATOR

3.括号不匹配，有多余右括号

恢复：移出输入串中的多余右括号，类型为UNNECESSARY

4.缺少右括号

恢复：把一个假想的右括号补入栈，类型为SEPARATOR



# 实验心得

在编程实现语法分析器的过程中，最重要的是要根据状态栈栈顶状态和当前输入字符在分析表中查找到相应的动作，按照相应的动作进行压栈、出栈处理。拿到相应的动作尤为重要。若出现了该文法无法识别的符号已经未定义的动作要注意进行错误提示。更进一步的，可以改写SLR分析表，将其改为带有错误标识的分析表，利用错误标识来对分析句子过程中出现的问题进行处理和恢复。

我在实验中遇到的一个比较有意思的bug是，再将状态栈中的状态读入string时，状态0会被读成\0出现异常，解决方法为借助to\_string函数，将数字加入到string中。