语法分析程序的设计与实现

一、题目

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->E+T|E-T|T

T->T*F|T/F|F

F->(E)|num

二、实验要求及实现方法要求：

1.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。 （必做）

(1) 编程实现算法 4.2，为给定文法自动构造预测分析表。

(2) 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。（必做）

(1) 构造识别该文法所有活前缀的 DFA。

(2) 构造该文法的 LR 分析表。

(3) 编程实现算法 4.3，构造 LR 分析程序。

方法一：递归调用

一、程序设计说明

消除左递归后，按照新的文法依次调用相应状态对应的函数，完成语法分析。

消除左递归后的新文法为：

E->TW

W->+TW|-TW|ε

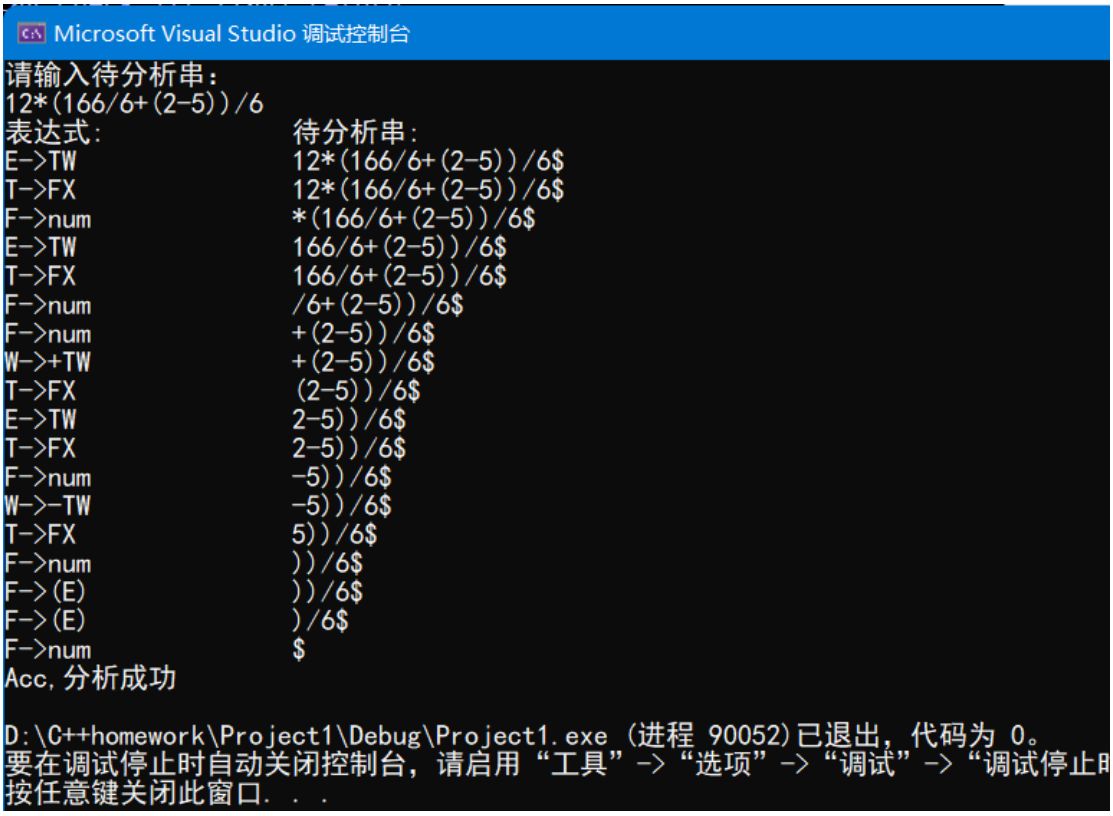
T->FX

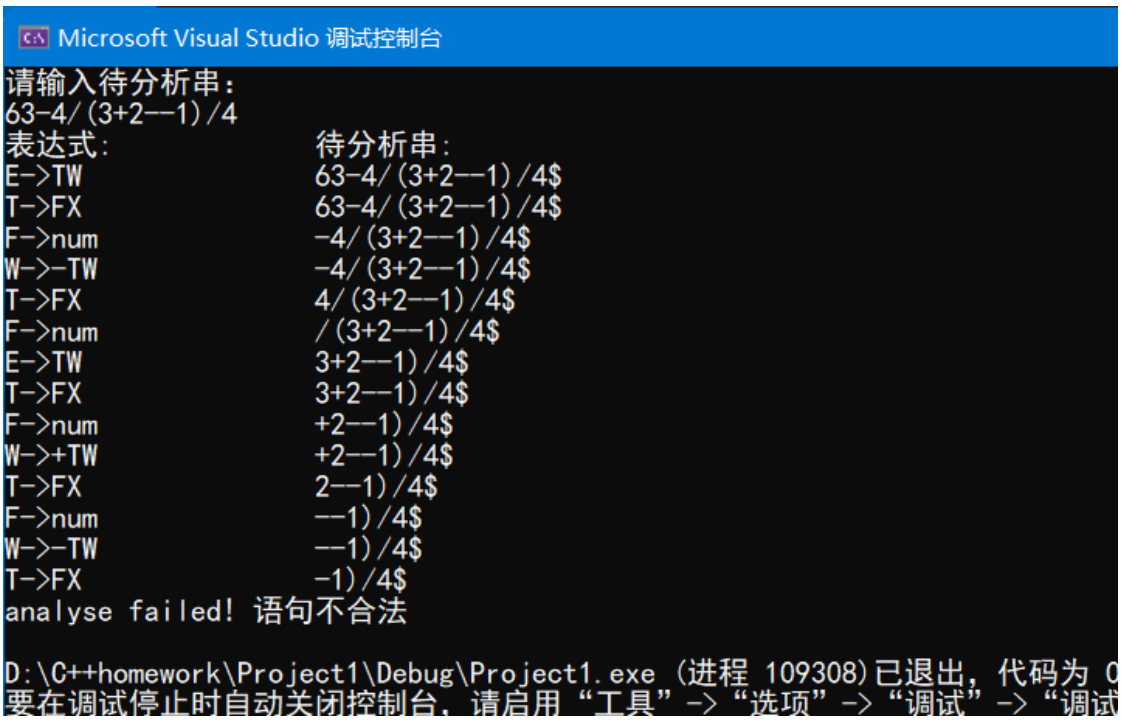
X->*FX|/FX|ε

F->(E)|num

因此有五个对应的函数，当分析过程中遇到错误情况时，如连续的运算符号，则会返回并报错。

二、运行结果





通过五个状态对应函数的递归调用，可以依次输出所采用的产生式，当遇见错误情况时，会直接跳出并报错。经过验证，程序结果正确。

三、源程序(附于文件尾)

方法二：LL(1)语法分析程序

一、程序设计说明：

通过编程实现算法 4.2 和算法 4.1,为给定文法自动构造预测分析表，并且构造 LL(1)预测分析程序

1.首先构造无左递归文法的 FIRST 集和 FOLLOW 集

E->TW

W->+TW|-TW| ε

T->FX

X->*FX|/FX| ε

F->(E)|num

	E	W	T	X	F
FIRST	(,num	+,-, ε	(,num	*,/, ε	(,num
FOLLOW	$,)	$,)	$,),+,-	$,),+,-	$,+,-,*,/,)

2.主要数据结构说明：

map<string, vector<string>> expression

存储非终结符对应德文法表达式

map<string, vector<string>> first_set

存储非终结符的 First 集合

map<string, vector<string>> follow_set

存储非终结符的 Follow 集合

3.主要函数说明：

void LL::create_anal_table()//分析表的构造及输出

说明:从非终结符开始，遍历终结符及$符号。

如若该终结符在该非终结符的 FIRST 集合中，则将对应的表达式填入分析表。

若该非终结符含空产生式以及终结符在 FOLLOW 集合中，将空产生式填入表。

若不含空产生式但终结符位于 FOLLOW 集合中，则将同步信息 synch 填入表

其余情况，则将 error 填入表

void LL::LL_analysis()//对输入符号串进行分析

说明：分析开始时，$在栈底，文法开始符号在栈顶，待分析串 input_s 在输入缓冲区中

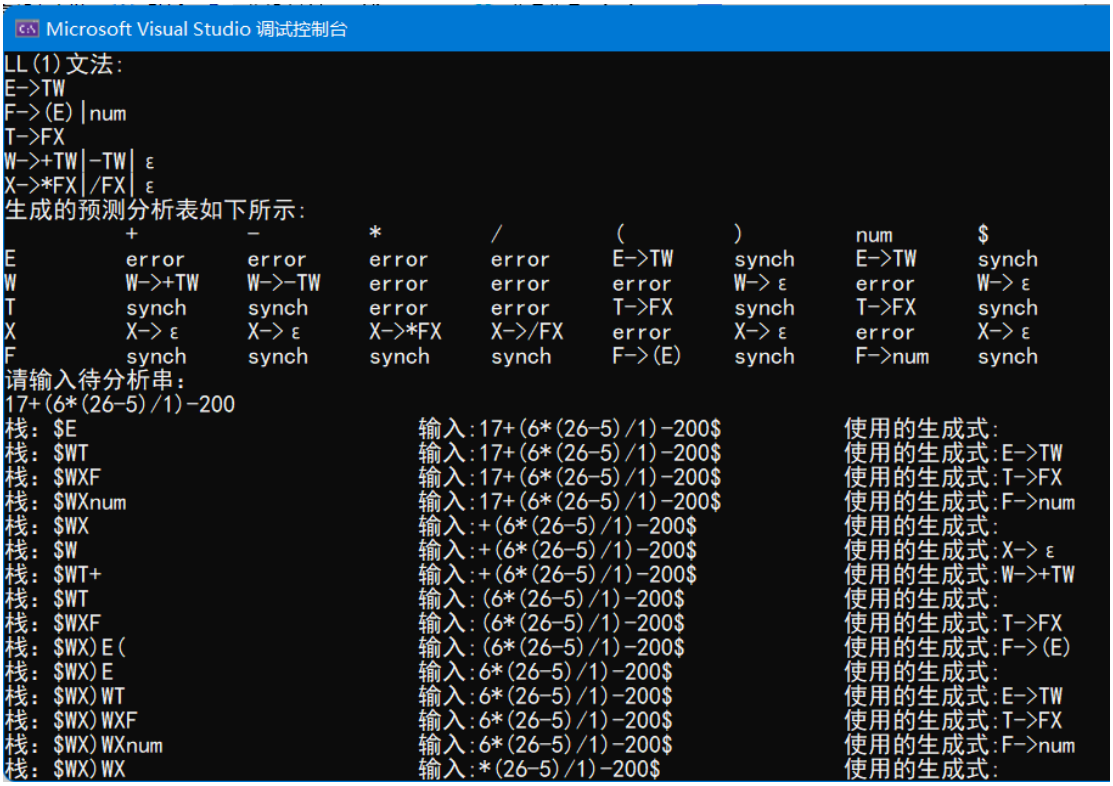
置 c_ptr 于 input_s 的第一个符号

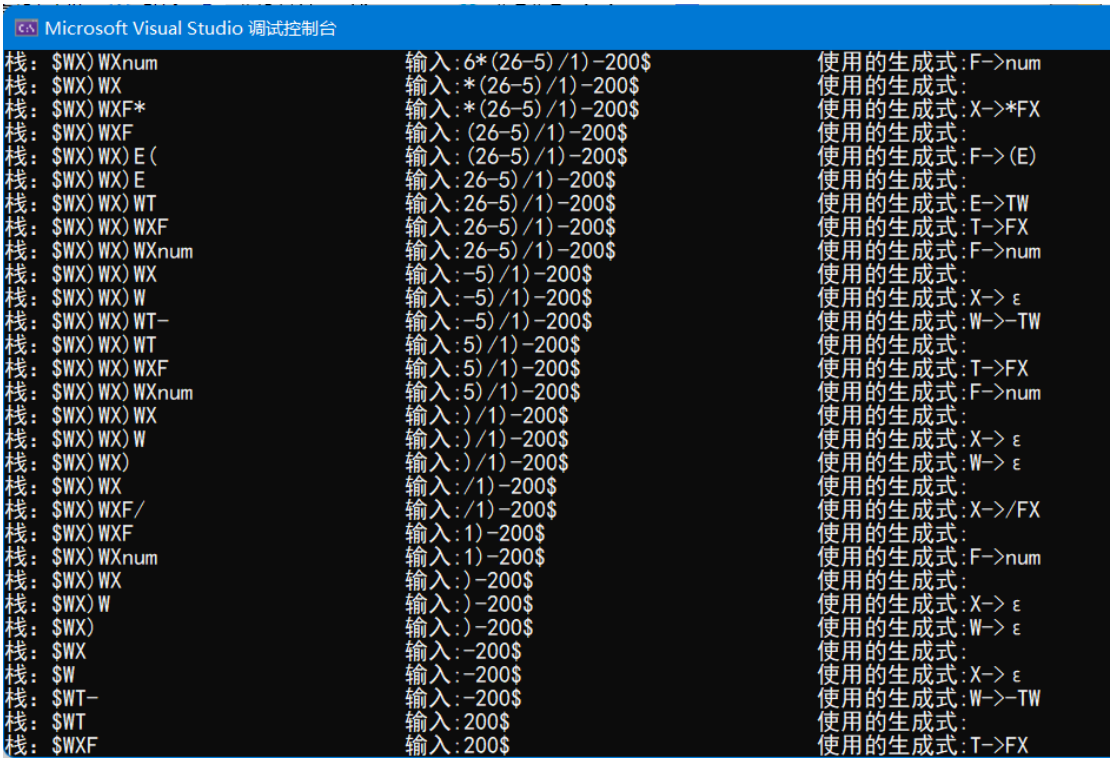
```
do{
    令 X 为栈顶符号,a 为 c_ptr 指向的符号;
    if(X 为终结符 or $)
        if(X==a)
            pop(X);c_ptr 前移
        else error();
    else//X 为非终结符
        if(M[X,a]有对应产生式)
            pop(X);
            产生式逆序入栈
        else error();
}while(X!=$)
```

在进行栈的相关操作时，需主要将 num 三个字符视作一个整体，在识别多位数字时，应将 c_ptr 移动到第一位不是数字的字符。

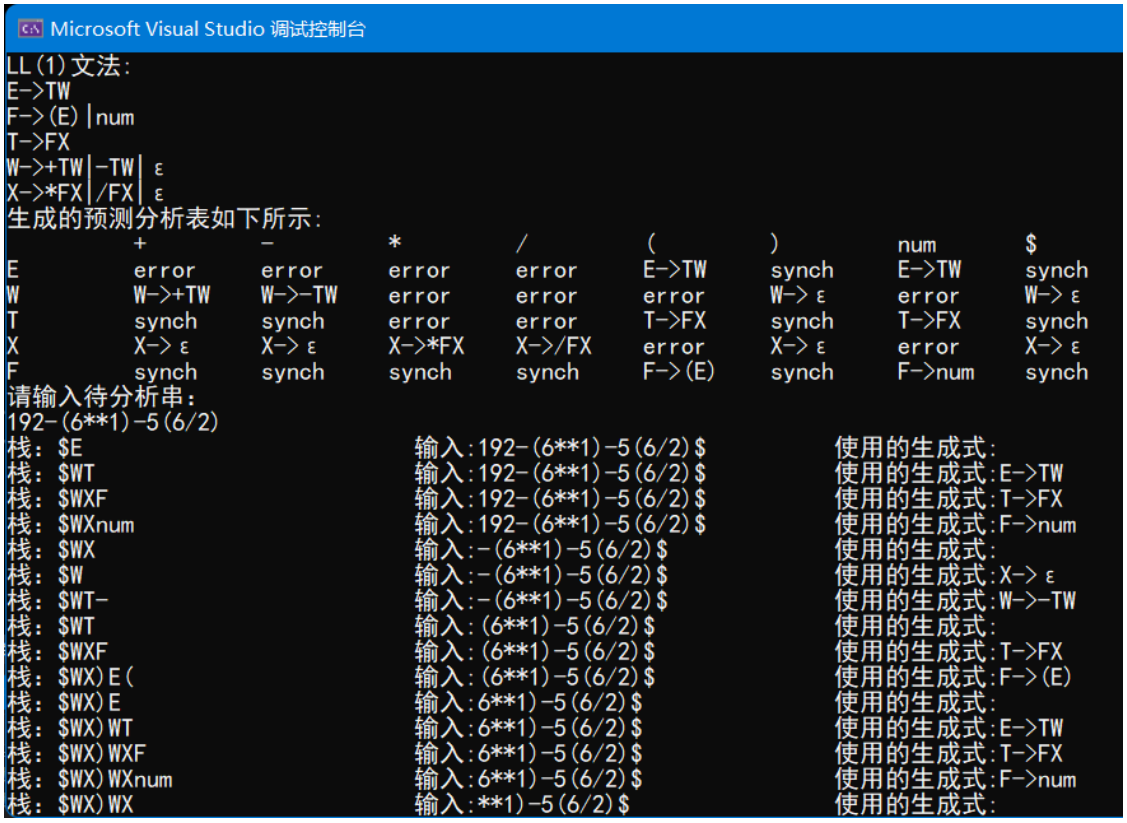
在错误处理 error()中，若 M[X,a]为 error,则 c_ptr 前移一位；若 M[X,a]为 synch，则将 X 弹出栈。

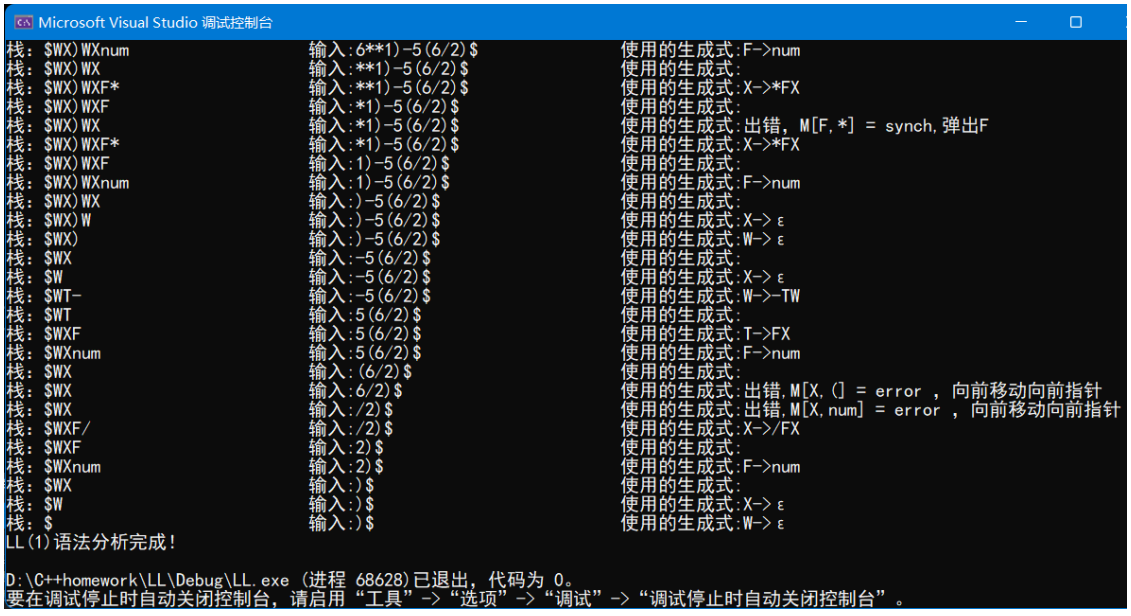
4.程序结果：


Microsoft Visual Studio 调试控制台
LL(1)文法:
E->TW
F->(E)|num
T->FX
W->+TW|-TW|ε
X->*FX|/FX|ε
生成的预测分析表如下所示:
+ - * / ( ) num $
E error error error error E->TW synch E->TW synch
W W->+TW W->-TW error error error W-> ε error W-> ε
T synch synch error error T->FX synch T->FX synch
X X-> ε X-> ε X->*FX X->/FX error X-> ε error X-> ε
F synch synch synch synch F->(E) synch F->num synch
请输入待分析串:
17+(6*(26-5)/1)-200
栈: $E 输入:17+(6*(26-5)/1)-200$ 使用的生成式:
栈: $WT 输入:17+(6*(26-5)/1)-200$ 使用的生成式:E->TW
栈: $WXF 输入:17+(6*(26-5)/1)-200$ 使用的生成式:T->FX
栈: $WXnum 输入:17+(6*(26-5)/1)-200$ 使用的生成式:F->num
栈: $WX 输入:+(6*(26-5)/1)-200$ 使用的生成式:
栈: $W 输入:+(6*(26-5)/1)-200$ 使用的生成式:X-> ε
栈: $WT+ 输入:+(6*(26-5)/1)-200$ 使用的生成式:W->+TW
栈: $WT 输入:(6*(26-5)/1)-200$ 使用的生成式:
栈: $WXF 输入:(6*(26-5)/1)-200$ 使用的生成式:T->FX
栈: $WX)E(输入:(6*(26-5)/1)-200$ 使用的生成式:F->(E)
栈: $WX)E 输入:6*(26-5)/1)-200$ 使用的生成式:
栈: $WX)WT 输入:6*(26-5)/1)-200$ 使用的生成式:E->TW
栈: $WX)WXF 输入:6*(26-5)/1)-200$ 使用的生成式:T->FX
栈: $WX)WXnum 输入:6*(26-5)/1)-200$ 使用的生成式:F->num
栈: $WX)WX 输入:*(26-5)/1)-200$ 使用的生成式:


Microsoft Visual Studio 调试控制台
栈: $WX)WXnum 输入:6*(26-5)/1)-200$ 使用的生成式:F->num
栈: $WX)WX 输入:*(26-5)/1)-200$ 使用的生成式:
栈: $WX)WXF* 输入:*(26-5)/1)-200$ 使用的生成式:X->*FX
栈: $WX)WXF 输入:(26-5)/1)-200$ 使用的生成式:
栈: $WX)WX)E(输入:(26-5)/1)-200$ 使用的生成式:F->(E)
栈: $WX)WX)E 输入:26-5)/1)-200$ 使用的生成式:
栈: $WX)WX)WT 输入:26-5)/1)-200$ 使用的生成式:E->TW
栈: $WX)WX)WXF 输入:26-5)/1)-200$ 使用的生成式:T->FX
栈: $WX)WX)WXnum 输入:26-5)/1)-200$ 使用的生成式:F->num
栈: $WX)WX)WX 输入:-5)/1)-200$ 使用的生成式:
栈: $WX)WX)W 输入:-5)/1)-200$ 使用的生成式:X-> ε
栈: $WX)WX)WT- 输入:-5)/1)-200$ 使用的生成式:W->-TW
栈: $WX)WX)WT 输入:5)/1)-200$ 使用的生成式:
栈: $WX)WX)WXF 输入:5)/1)-200$ 使用的生成式:T->FX
栈: $WX)WX)WXnum 输入:5)/1)-200$ 使用的生成式:F->num
栈: $WX)WX)WX 输入:)/1)-200$ 使用的生成式:
栈: $WX)WX)W 输入:)/1)-200$ 使用的生成式:X-> ε
栈: $WX)WX) 输入:)/1)-200$ 使用的生成式:W-> ε
栈: $WX)WX 输入:/1)-200$ 使用的生成式:
栈: $WX)WXF/ 输入:/1)-200$ 使用的生成式:X->/FX
栈: $WX)WXF 输入:1)-200$ 使用的生成式:
栈: $WX)WXnum 输入:1)-200$ 使用的生成式:F->num
栈: $WX)WX 输入:)-200$ 使用的生成式:
栈: $WX)W 输入:)-200$ 使用的生成式:X-> ε
栈: $WX) 输入:)-200$ 使用的生成式:W-> ε
栈: $WX 输入:-200$ 使用的生成式:
栈: $W 输入:-200$ 使用的生成式:X-> ε
栈: $WT- 输入:-200$ 使用的生成式:W->-TW
栈: $WT 输入:200$ 使用的生成式:
栈: $WXF 输入:200$ 使用的生成式:T->FX


栈: $WT- 输入:-200$ 使用的生成式:W->-TW
栈: $WT 输入:200$ 使用的生成式:
栈: $WXF 输入:200$ 使用的生成式:T->FX
栈: $WXnum 输入:200$ 使用的生成式:F->num
栈: $WX 输入:$ 使用的生成式:
栈: $W 输入:$ 使用的生成式:X-> ε
栈: $ 输入:$ 使用的生成式:W-> ε
LL(1)语法分析完成!
D:\C++homework\LL\Debug\LL.exe (进程 129380)已退出, 代码为 0。





当程序正常运行时，会根据 LL 分析表中的产生式进行规约等动作，当遇见 error 情况时，如 9(3+2)，c_ptr 会向前移动继续分析；当遇见同步信息 synch 时，会将当前的非终结符弹出栈，继续分析。经过验证，程序结果正确。

4. 源程序(附于文件尾)

方法三：LR(0)语法分析程序

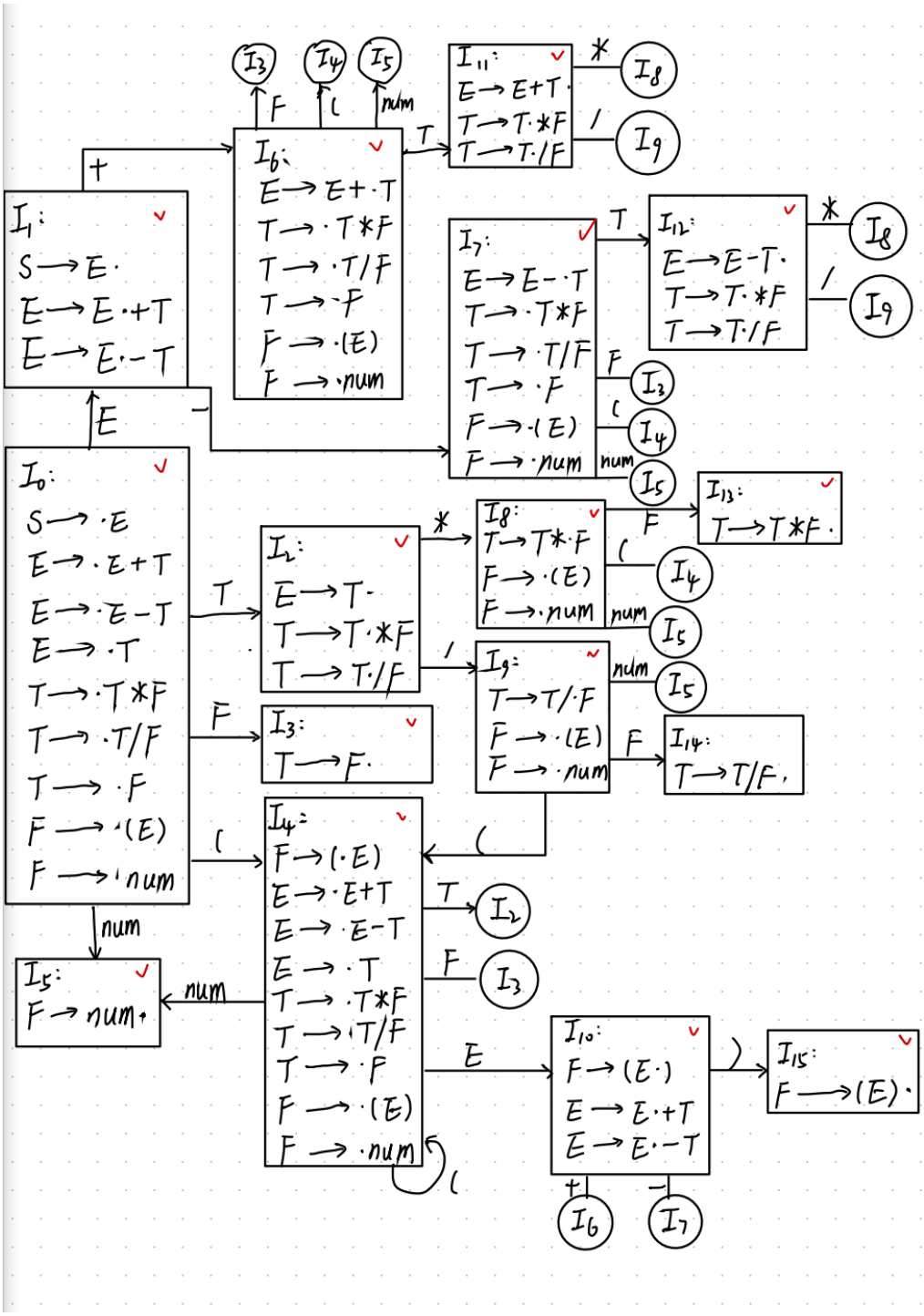
一、程序设计说明：

首先拓广文法：

(0)S->E　(1)E->E+T　(2)E->E-T　(3)E->T　(4)T->T*F

(5)T->T/F　(6)T->F　(7)F->(E)　(8)F->num

然后构造识别该文法所有活前缀的 DFA



然后构造该文法的 LR 分析表：

状态	Action								Goto		
	(	)	+	-	*	/	num	$	E	T	F
0	S4						S5		1	2	3
1			S6	S7				acc			
2		R3	R3	R3	S8	S9		R3			
3		R6	R6	R6	R6	R6		R6			
4	S4						S5		10	2	3
5		R8	R8	R8	R8	R8		R8			
6	S4						S5			11	3
7	S4						S5			12	3
8	S4						S5				13
9	S4						S5				14
10		S15	S6	S7							
11		R1	R1	R1	S8	S9		R1			
12		R2	R2	R2	S8	S9		R2			
13		R4	R4	R4	R4	R4		R4			
14		R5	R5	R5	R5	R5		R5			
15		R7	R7	R7	R7	R7		R7			

主要数据结构说明：

vector<string> s_state:状态栈中字符

vector<string> s_symbol:symbol 栈中字符

stack<string> state_stack:状态栈

stack<string> symbol_stack:symbol 栈

map<string, int> states_row:状态对应行号

map<string, int> ter_line:终结符，非终结符对应列号

主要函数说明：

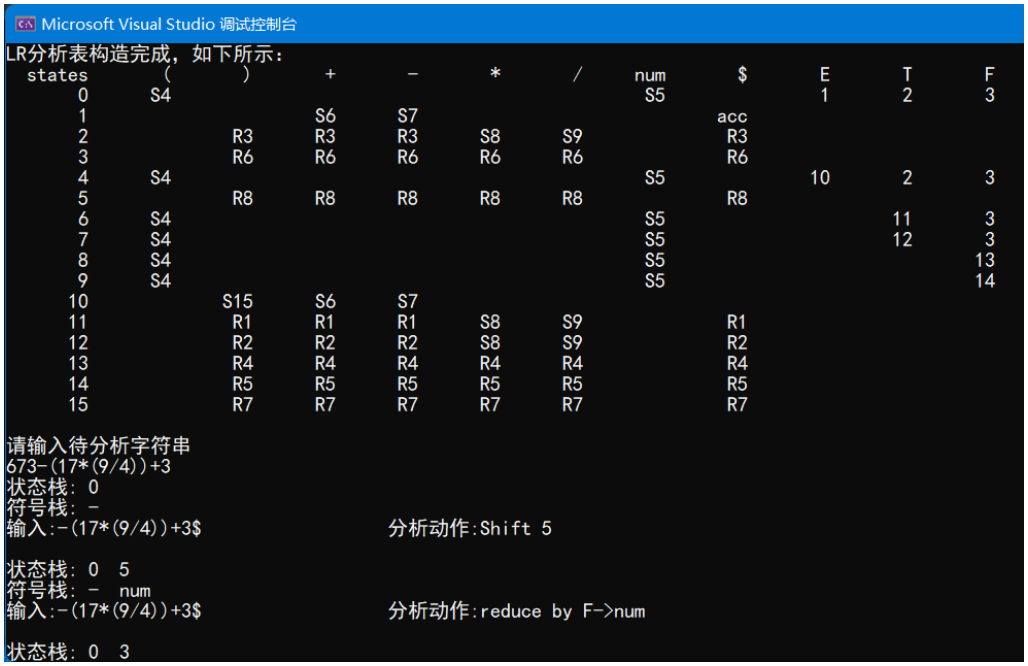
void LR::LR_analy(string& str)：

伪码描述：

```
while(true)
{
    S<-栈顶符号，a<-str[c_ptr]
    if(action[S,a]==Shift S’)
    {
        a 入符号栈，S’入状态栈
        c_ptr++
    }
    else if(action[S,a]==reduce by A->β)
    {
        从栈顶弹出|β|个符号；//令 S’为当前栈顶状态
        A 入符号栈,goto[S’,A]入状态栈
        printf A->β
    }
    else if(action[A,a]==acc) return
    else   error();
}
```

二、运行结果

样例一(Acc)：673-(17*(9/4))+3



```
状态栈: 0  3
符号栈: -  F
输入:-(17*(9/4))+3$                    分析动作:reduce by T->F

状态栈: 0  2
符号栈: -  T
输入:-(17*(9/4))+3$                    分析动作:reduce by E->T

状态栈: 0  1
符号栈: -  E
输入:(17*(9/4))+3$                     分析动作:Shift 7

状态栈: 0  1  7
符号栈: -  E  -
输入:17*(9/4))+3$                      分析动作:Shift 4

状态栈: 0  1  7  4
符号栈: -  E  -  (
输入:*(9/4))+3$                        分析动作:Shift 5

状态栈: 0  1  7  4  5
符号栈: -  E  -  (  num
输入:*(9/4))+3$                        分析动作:reduce by F->num

状态栈: 0  1  7  4  3
符号栈: -  E  -  (  F
输入:*(9/4))+3$                        分析动作:reduce by T->F

状态栈: 0  1  7  4  2
```

```
状态栈: 0  1  7  4  2
符号栈: -  E  -  (  T
输入:(9/4))+3$                         分析动作:Shift 8

状态栈: 0  1  7  4  2  8
符号栈: -  E  -  (  T  *
输入:9/4))+3$                          分析动作:Shift 4

状态栈: 0  1  7  4  2  8  4
符号栈: -  E  -  (  T  *  (
输入:/4))+3$                           分析动作:Shift 5

状态栈: 0  1  7  4  2  8  4  5
符号栈: -  E  -  (  T  *  (  num
输入:/4))+3$                           分析动作:reduce by F->num

状态栈: 0  1  7  4  2  8  4  3
符号栈: -  E  -  (  T  *  (  F
输入:/4))+3$                           分析动作:reduce by T->F

状态栈: 0  1  7  4  2  8  4  2
符号栈: -  E  -  (  T  *  (  T
输入:4))+3$                            分析动作:Shift 9

状态栈: 0  1  7  4  2  8  4  2  9
符号栈: -  E  -  (  T  *  (  T  /
输入:))+3$                             分析动作:Shift 5

状态栈: 0  1  7  4  2  8  4  2  9  5
```

```
状态栈: 0  1  7  4  2  8  4  2  9  5
符号栈: -  E  -  (  T  *  (  T  /  num
输入:))+3$                              分析动作:reduce by F->num

状态栈: 0  1  7  4  2  8  4  2  9  14
符号栈: -  E  -  (  T  *  (  T  /  F
输入:))+3$                              分析动作:reduce by T->T/F

状态栈: 0  1  7  4  2  8  4  2
符号栈: -  E  -  (  T  *  (  T
输入:))+3$                              分析动作:reduce by E->T

状态栈: 0  1  7  4  2  8  4  10
符号栈: -  E  -  (  T  *  (  E
输入:)+3$                               分析动作:Shift 15

状态栈: 0  1  7  4  2  8  4  10 15
符号栈: -  E  -  (  T  *  (  E  )
输入:)+3$                               分析动作:reduce by F->(E)

状态栈: 0  1  7  4  2  8  13
符号栈: -  E  -  (  T  *  F
输入:)+3$                               分析动作:reduce by T->T*F

状态栈: 0  1  7  4  2
符号栈: -  E  -  (  T
输入:)+3$                               分析动作:reduce by E->T

状态栈: 0  1  7  4  10
符号栈: -  E  -  (  E
输入:+3$                                分析动作:Shift 15

状态栈: 0  1  7  4  10 15
符号栈: -  E  -  (  E  )
输入:+3$                                分析动作:reduce by F->(E)

状态栈: 0  1  7  3
符号栈: -  E  -  F
输入:+3$                                分析动作:reduce by T->F

状态栈: 0  1  7  12
符号栈: -  E  -  T
输入:+3$                                分析动作:reduce by E->E-T

状态栈: 0  1
符号栈: -  E
输入:3$                                 分析动作:Shift 6

状态栈: 0  1  6
符号栈: -  E  +
输入:$                                  分析动作:Shift 5

状态栈: 0  1  6  5
符号栈: -  E  +  num
输入:$                                  分析动作:reduce by F->num

状态栈: 0  1  6  3
```

```
状态栈: 0  1  6  3
符号栈: -  E  +  F
输入:$                                  分析动作:reduce by T->F

状态栈: 0  1  6  11
符号栈: -  E  +  T
输入:$                                  分析动作:reduce by E->E+T

状态栈: 0  1
符号栈: -  E
输入:$                                  分析动作:acc

Accepted!分析完成
```

样例 2(No acc)(17-(3--2))/7

```
LR分析表构造完成，如下所示:
  states       (       )       +       -       *       /     num       $       E       T       F
       0      S4                                              S5                      1       2       3
       1                      S6      S7                                acc
       2              R3      R3      R3      S8      S9              R3
       3              R6      R6      R6      R6      R6              R6
       4      S4                                              S5                      10      2       3
       5              R8      R8      R8      R8      R8              R8
       6      S4                                              S5                      11      3
       7      S4                                              S5                      12      3
       8      S4                                              S5                              13
       9      S4                                              S5                              14
      10             S15      S6      S7
      11              R1      R1      R1      S8      S9              R1
      12              R2      R2      R2      S8      S9              R2
      13              R4      R4      R4      R4      R4              R4
      14              R5      R5      R5      R5      R5              R5
      15              R7      R7      R7      R7      R7              R7

请输入待分析字符串
(17-(3--2))/7
状态栈: 0
符号栈: -
输入:17-(3--2))/7$              分析动作:Shift 4

状态栈: 0  4
符号栈: -  (
输入:-(3--2))/7$                分析动作:Shift 5

状态栈: 0  4  5
```

```
状态栈: 0  4  5
符号栈: -  (  num
输入:-(3--2))/7$                        分析动作:reduce by F->num

状态栈: 0  4  3
符号栈: -  (  F
输入:-(3--2))/7$                        分析动作:reduce by T->F

状态栈: 0  4  2
符号栈: -  (  T
输入:-(3--2))/7$                        分析动作:reduce by E->T

状态栈: 0  4  10
符号栈: -  (  E
输入:(3--2))/7$                         分析动作:Shift 7
```

```
状态栈: 0  4  10 7
符号栈: -  (  E  -
输入:3--2))/7$                  分析动作:Shift 4

状态栈: 0  4  10 7  4
符号栈: -  (  E  -  (
输入:--2))/7$                   分析动作:Shift 5

状态栈: 0  4  10 7  4  5
符号栈: -  (  E  -  (  num
输入:--2))/7$                   分析动作:reduce by F->num

状态栈: 0  4  10 7  4  3
符号栈: -  (  E  -  (  F
输入:--2))/7$                   分析动作:reduce by T->F

状态栈: 0  4  10 7  4  2
符号栈: -  (  E  -  (  T
输入:--2))/7$                   分析动作:reduce by E->T

状态栈: 0  4  10 7  4  10
符号栈: -  (  E  -  (  E
输入:-2))/7$                    分析动作:Shift 7

状态栈: 0  4  10 7  4  10 7
符号栈: -  (  E  -  (  E  -
输入:-2))/7$                    分析动作:No acc

Failed!
```

由运行结果可知，运行结果与预期结果相符。当遇到错误情况时，如运算符号相邻,或括号不匹配时，会跳出程序并报错。

源程序：

(1)递归

```cpp
#include<iostream>
#include<string>
#include<iomanip>
#define sub_str str.substr(c_ptr)
#define flag_check if(!flag) return;
using namespace std;
void E();
void W();
void T();
void X();
void F();
bool flag = true;
int c_ptr = 0;//指向被分析的字符
string str = "";
```

```
int main()
{
    cout << "请输入待分析串: " << endl;
    cin >> str;
    str += "$";
    cout << left << setw(20) << "表达式:" << setw(20) << "待分析串:"
<< endl;
    E();
    if (str[c_ptr] == '$' && flag)
        cout << "Acc,分析成功" << endl;
    else
        cout << "analyse failed! 语句不合法" << endl;
    system("pause");
    return 0;
}
void E()
{
    flag_check;
    cout << left << setw(20) << "E->TW" << setw(20) << sub_str <<
endl;
    T();
    W();
}
void W()
{
    flag_check;
    if (str[c_ptr] == '+')
    {
        cout << left << setw(20) << "W->+TW" << setw(20) <<
sub_str << endl;
        c_ptr++;
        T();
        W();
    }
    else if (str[c_ptr] == '-')
    {
        cout << left << setw(20) << "W->-TW" << setw(20) <<
sub_str << endl;
        c_ptr++;
        T();
        W();
    }
}
void T()
```

```
{
    flag_check;
    cout << left << setw(20) << "T->FX" << setw(20) << sub_str <<
endl;
    F();
    X();
}
void X()
{
    flag_check;
    if (str[c_ptr] == '*')
    {
        c_ptr++;
        F();
        X();
    }
    else if (str[c_ptr] == '/')
    {
        c_ptr++;
        F();
        X();
    }
}
void F()
{
    flag_check;
    if (str[c_ptr] == '(')
    {
        c_ptr++;
        E();
        if (str[c_ptr] == ')')
        {
            cout << left << setw(20) << "F->(E)" << setw(20) <<
sub_str << endl;
            c_ptr++;
        }
        else
            flag = false;
    }
    else if (isdigit(str[c_ptr]))
    {
        while(isdigit(str[c_ptr]))
            c_ptr++;
```

```
        cout << left << setw(20) << "F->num" << setw(20) <<
sub_str << endl;
    }
    else
        flag = false;
}
```

(2)LL(1)

```
#include<iostream>
#include<fstream>
#include<vector>
#include<iomanip>
#include<string>
#include<stack>
#include<map>
#define rows 6
#define cols 9
using namespace std;
class LL
{
private:
    string input_s;//输入，即待分析字符串
    string stack_s;//栈中字符串
    string ana_table[rows][cols];//预测分析表
    stack<string> stack;//栈
    vector<string> Terminator;//终结符
    vector<string> nTerminator;//非终结符
    map<string, vector<string>> expression;//文法表达式
    map<string, vector<string>> first_set;//First 集合
    map<string, vector<string>> follow_set;//Follow 集合
    map<string, int> col_nter;//非终结符对应的行号
    map<string, int> row_ter;//终结符对应的列号
public:
    string init_s;//最原始的分析字符串
    void init_set();//各种符号，集合的初始化
    void init_stack();//栈的初始化
    void create_anal_table();//构造、输出分析表
    void LL_analysis();//对输入符号串进行分析
};

int main()
{
    LL L_table;
    L_table.init_set();
    L_table.create_anal_table();
```

```
    cout << "请输入待分析串: \n";
    cin >> L_table.init_s;
    L_table.init_s += "$";
    L_table.init_stack();
    L_table.LL_analysis();
    system("pause");
    return 0;
}

void LL::init_set()//各种符号，集合的初始化
{
    /*-----------------存入无左递归文法---------------*/
    expression["E"] = { "TW" };
    expression["W"] = { "+TW","-TW","ε" };
    expression["T"] = { "FX" };
    expression["X"] = { "*FX","/FX","ε" };
    expression["F"] = { "(E)","num" };
    cout << "LL(1)文法:" << endl;
    for (auto ptr = expression.begin(); ptr != expression.end();
ptr++)
    {
        cout << ptr->first << "->";
        for (auto eptr = ptr->second.begin(); eptr <
ptr->second.end(); eptr++)
            if (eptr < ptr->second.end() - 1)
                cout << *eptr << "|";
            else
                cout << *eptr << endl;
    }
    /*-------------------终结符--------------------*/
    ana_table[0][1] = "+", ana_table[0][2] = "-", ana_table[0][3]
= "*", ana_table[0][4] = "/";
    ana_table[0][5] = "(", ana_table[0][6] = ")", ana_table[0][7]
= "num", ana_table[0][8] = "$";
    /*-------------------非终结符-----------------*/
    ana_table[1][0] = "E", ana_table[2][0] = "W", ana_table[3][0]
= "T",
        ana_table[4][0] = "X", ana_table[5][0] = "F";
    /*-------------------FIRST 集合----------------*/
    first_set["E"] = {"(", "num"};
    first_set["W"] = { "+","-","ε" };
    first_set["T"] = { "(","num" };
    first_set["X"] = { "*","/","ε" };
```

```
    first_set["F"] = { "(","num" };
    /*------------------FOLLOW集合-----------------*/
    follow_set["E"] = { "$",")" };
    follow_set["W"] = { "$",")" };
    follow_set["T"] = { "$",")","+","-" };
    follow_set["X"] = { "$",")","+","-" };
    follow_set["F"] = { "$","+","-","*","/",")" };
    /*---------终结符和非终结符对应的行号和列号-------*/
    for (int i = 1; i < rows; i++)
        row_ter[ana_table[i][0]] = i;
    for (int j = 1; j < cols; j++)
        col_nter[ana_table[0][j]] = j;
}
void LL::init_stack()//栈的初始化
{
    //将起始符号 E 和$开始
    stack.push("$");
    stack.push(ana_table[1][0]);
    stack_s = "$" + ana_table[1][0];
}
void LL::create_anal_table()//构造、输出分析表
{
    ana_table[0][0] = "";
    for (int i = 1; i < rows; i++)
    {
        string n_ter = ana_table[i][0];//取出非终结符的 first 集和
follow 集
        vector<string>first_tmp = first_set[n_ter];
        vector<string>follow_tmp = follow_set[n_ter];
        bool e_empty = false;//exit empty，判断是否含空
        if (*(first_tmp.end() - 1) == "ε")
            e_empty = true;
        for (int j = 1; j < cols; j++)
        {
            string ter = ana_table[0][j];//终结符
            if (find(first_tmp.begin(), first_tmp.end(), ter) !=
first_tmp.end())//找到对应的终结符
            {
                vector<string> sentence = expression[n_ter];
                if (sentence.size() == 1)//单个生成式
                    ana_table[i][j] = n_ter + "->" + sentence[0];
                else//多个生成式
                {
                    for (int k = 0; k < sentence.size(); k++)
```

```
                    {
                        string tmp = sentence[k];
                        if (ter.size() == 1)
                            tmp = tmp.at(0);
                        if (tmp == ter)
                        {
                            ana_table[i][j] = n_ter + "->" +
sentence[k];
                            break;
                        }
                    }
                }
            }
            //含空产生式且终结符在 FOLLOW 集中
            else if (e_empty && find(follow_tmp.begin(),
follow_tmp.end(), ter) != follow_tmp.end())
                ana_table[i][j] = n_ter + "->ε";
            else if (find(follow_tmp.begin(), follow_tmp.end(),
ter) != follow_tmp.end())
                ana_table[i][j] = "synch";
            else
                ana_table[i][j] = "error";
        }
    }
    cout << "生成的预测分析表如下所示:" << endl;
    for (int i = 0; i <rows; i++)
    {
        for (int j = 0; j < cols; j++)
            cout << setw(10) << left << ana_table[i][j];
        cout << endl;
    }
}
void LL::LL_analysis()//对输入符号串进行分析
{
    input_s = init_s;
    int c_ptr = 0;//指向正在分析的字符
    string genrate_s = "";//所用生成式
    while (!stack.empty())
    {
        cout << "栈: " << setw(input_s.size() + 10) << stack_s << "
输入:" << setw(input_s.size() + 10) << init_s.substr(c_ptr) << "使
用的生成式:" << genrate_s << endl;
        string s_flag = "";//字符标记
        genrate_s = "";//产生式
```

```
        char ch = input_s.at(c_ptr);//当前分析的字符
        string s_top = stack.top();//栈顶符号
        if (s_top == "$")
        {
            cout << "LL(1)语法分析完成! " << endl;
            return;
        }
        if (ch <= '9' && ch >= '0')
            s_flag = "num";
        else
            s_flag = ch;
        //当前字符符号和栈顶符号相同时
        if (s_flag == s_top)
        {
            if (stack.top() == "num")
                stack_s.erase(stack_s.end() - 3 - stack_s.begin(),
3);//删除 n u m
            else
                stack_s.erase(stack_s.end() - 1);
            stack.pop();//栈顶出栈
            if (s_flag == "num")
            {
                //为了读取多位数字，将指针移动到非数字的第一位
                for (; c_ptr < input_s.size(); c_ptr++)
                {
                    char c_ch = input_s.at(c_ptr);//当前字符
                    if (c_ch == '+' || c_ch == '-' || c_ch == '*'
||c_ch == '/' || c_ch == '(' || c_ch == ')' || c_ch == '$')
                        break;
                }
            }
            else
                c_ptr++;
        }
        else
        {
            //栈顶符号对应的行号
            int row = row_ter[s_top];
            int col = col_nter[s_flag];
            //在分析表中找到对应信息
            genrate_s = ana_table[row][col];
            if (genrate_s == "synch")
            {//从栈顶弹出
```

```
                genrate_s = "出错, M[" + ana_table[row][0] + "," +
s_flag + "]" + " = synch,弹出" + stack.top();
                if (stack.top() == "num")
                    stack_s.erase(stack_s.end() - 3 -
stack_s.begin(), 3);
                else
                    stack_s.erase(stack_s.end() - 1);
                stack.pop();
            }
            else if (genrate_s == "error")
            {
                genrate_s = "出错,M[" + ana_table[row][0] + "," +
s_flag + "]" + " = error , 向前移动向前指针";
                c_ptr++;
            }
            else
            {
                if (stack.top() == "num")
                    stack_s.erase(stack_s.end() - 3 -
stack_s.begin(), 3);
                else
                    stack_s.erase(stack_s.end() - 1);
                stack.pop();
                int pos = genrate_s.find_first_of(">") + 1;
                string tmpstr = genrate_s.substr(pos);
                if (tmpstr == "num")
                {
                    stack.push(tmpstr);
                    stack_s += tmpstr;
                }
                else if (tmpstr == "ε");
                else
                {
                    for (int i = tmpstr.size() - 1; i >= 0; i--)
                    {//逆序入栈
                        char tmp1 = tmpstr[i];//获取生成式右部
                        string tmp = "";
                        tmp += tmp1;
                        stack.push(tmp);
                        stack_s += tmp;
                    }
                }
            }
        }
```

```
    }
}
```

(3)LR(0)

```
#include<iostream>
#include<string>
#include<vector>
#include<iomanip>
#include<stack>
#include<map>
#define rows 17
#define cols 12
using namespace std;
class LR
{
private:
    string input_s;//输入，即待分析字符串
    string LR_Table[rows][cols];//分析表
    vector<string> expression_vec;//文法
    vector<string> s_state;//状态栈中字符
    vector<string> s_symbol;//symbol 栈中字符
    stack<string> state_stack;//状态栈
    stack<string> symbol_stack;//symbol 栈
    map<string, int> states_row;//状态对应行号
    map<string, int> ter_line;//终结符，非终结符对应列号
public:
    void LR_analy(string& str);
    void show_info(string &c_ch,string &express,int c_ptr);//分析过
程
    void init_stack();//栈初始化
    void init_set();//规范族，文法等的初始化
};

int main()
{
    LR lr;
    lr.init_set();
    lr.init_stack();
    cout << "请输入待分析字符串" << endl;
    string str;
    cin >> str;
    str += "$";
    lr.LR_analy(str);
    system("pause");
    return 0;
```

```
}

void LR::init_stack()//栈初始化
{
    state_stack.push("0");
    symbol_stack.push("-");
    s_state.push_back("0");
    s_symbol.push_back("-");
}

void LR::init_set()//规范族，文法等的初始化
{
    string states = "states";//存储符号
    LR_Table[0][0] = states, LR_Table[0][1] = "(", LR_Table[0][2]
= ")", LR_Table[0][3] = "+",
    LR_Table[0][4] = "-", LR_Table[0][5] = "*", LR_Table[0][6] =
"/", LR_Table[0][7] = "num",
    LR_Table[0][8] = "$", LR_Table[0][9] = "E", LR_Table[0][10] =
"T", LR_Table[0][11] = "F";
    //存储表达式
    expression_vec.push_back("S->E"),
expression_vec.push_back("E->E+T"),
expression_vec.push_back("E->E-T");
    expression_vec.push_back("E->T"),
expression_vec.push_back("T->T*F"),
expression_vec.push_back("T->T/F");
    expression_vec.push_back("T->F"),
expression_vec.push_back("F->(E)"),
expression_vec.push_back("F->num");
    //存储状态
    for (int i = 1; i < rows; i++)
        LR_Table[i][0] = to_string(i-1);
    states_row["0"] = 1, states_row["1"] = 2, states_row["2"] = 3,
states_row["3"] = 4;
    states_row["4"] = 5, states_row["5"] = 6, states_row["6"] = 7,
states_row["7"] = 8;
    states_row["8"] = 9, states_row["9"] = 10, states_row["10"] =
11, states_row["11"] = 12;
    states_row["12"] = 13, states_row["13"] = 14, states_row["14"]
= 15, states_row["15"] = 16;

    ter_line["("] = 1, ter_line[")"] = 2, ter_line["+"] = 3,
ter_line["-"] = 4;
```

```
    ter_line["*"] = 5, ter_line["/"] = 6, ter_line["num"] = 7,
ter_line["$"] = 8;
    ter_line["E"] = 9, ter_line["T"] = 10, ter_line["F"] = 11;

    /*----------存储分析表--------*/
    LR_Table[1][1] = "S4", LR_Table[1][7] = "S5", LR_Table[1][9] =
"1", LR_Table[1][10] = "2", LR_Table[1][11] = "3";
    LR_Table[2][3] = "S6", LR_Table[2][4] = "S7", LR_Table[2][8] =
"acc";
    LR_Table[3][2] = "R3", LR_Table[3][3] = "R3", LR_Table[3][4] =
"R3", LR_Table[3][5] = "S8", LR_Table[3][6] = "S9",
LR_Table[3][8] = "R3";
    LR_Table[4][2] = "R6", LR_Table[4][3] = "R6", LR_Table[4][4] =
"R6", LR_Table[4][5] = "R6", LR_Table[4][6] = "R6",
LR_Table[4][8] = "R6";
    LR_Table[5][1] = "S4", LR_Table[5][7] = "S5", LR_Table[5][9] =
"10", LR_Table[5][10] = "2", LR_Table[5][11] = "3";
    LR_Table[6][2] = "R8", LR_Table[6][3] = "R8", LR_Table[6][4] =
"R8", LR_Table[6][5] = "R8", LR_Table[6][6] = "R8",
LR_Table[6][8] = "R8";
    LR_Table[7][1] = "S4", LR_Table[7][7] = "S5", LR_Table[7][10]
= "11", LR_Table[7][11] = "3";
    LR_Table[8][1] = "S4", LR_Table[8][7] = "S5", LR_Table[8][10]
= "12", LR_Table[8][11] = "3";
    LR_Table[9][1] = "S4", LR_Table[9][7] = "S5", LR_Table[9][11]
= "13";
    LR_Table[10][1] = "S4", LR_Table[10][7] = "S5",
LR_Table[10][11] = "14";
    LR_Table[11][2] = "S15", LR_Table[11][3] = "S6",
LR_Table[11][4] = "S7";
    LR_Table[12][2] = "R1", LR_Table[12][3] = "R1",
LR_Table[12][4] = "R1", LR_Table[12][5] = "S8", LR_Table[12][6] =
"S9", LR_Table[12][8] = "R1";
    LR_Table[13][2] = "R2", LR_Table[13][3] = "R2",
LR_Table[13][4] = "R2", LR_Table[13][5] = "S8", LR_Table[13][6] =
"S9", LR_Table[13][8] = "R2";
    LR_Table[14][2] = "R4", LR_Table[14][3] = "R4",
LR_Table[14][4] = "R4", LR_Table[14][5] = "R4", LR_Table[14][6] =
"R4", LR_Table[14][8] = "R4";
    LR_Table[15][2] = "R5", LR_Table[15][3] = "R5",
LR_Table[15][4] = "R5", LR_Table[15][5] = "R5", LR_Table[15][6] =
"R5", LR_Table[15][8] = "R5";
```

```
    LR_Table[16][2] = "R7", LR_Table[16][3] = "R7",
LR_Table[16][4] = "R7", LR_Table[16][5] = "R7", LR_Table[16][6] =
"R7", LR_Table[16][8] = "R7";

    cout << "LR 分析表构造完成，如下所示：" << endl;
    for (int row = 0; row < rows; ++row) {
        for (int col = 0; col < cols; ++col)
            cout << setw(8) << LR_Table[row][col];
        cout << endl;
    }
    cout << endl;
}
void LR::show_info(string& c_ch, string& express, int c_ptr)//分析
过程
{
    cout << left << "状态栈: ";
    for (auto it : s_state)//遍历
        cout << setw(3) << it;
    cout << endl << left << "符号栈: ";
    for (auto it : s_symbol)//遍历
        cout << setw(3) << it;

    cout << endl << left << "输入:" << setw(c_ch.size() + 15) <<
        c_ch.substr(c_ptr) << "分析动作:" << setw(25) << express <<
endl << endl;
}

void LR::LR_analy(string& str)
{
    int c_ptr = 0;//指向当前分析字符
    int row = 0, col = 0;//行号，列号
    input_s = str;
    string express = "";
    while (true)
    {
        char current_ch = input_s[c_ptr];//当前字符
        string ch = "";
        ch += current_ch;//化为 string 型
        //cout << ch << endl;
        //为了读取多位数字，读取指针移动到第一位非数字符号之前
        if (isdigit(current_ch))
        {
            for (; c_ptr < input_s.size(); c_ptr++)
            {
```

```
                if (input_s[c_ptr] == '+' || input_s[c_ptr] == '-'
|| input_s[c_ptr] == '*' || input_s[c_ptr] == '/' ||
input_s[c_ptr] == '(' ||
                    input_s[c_ptr] == ')' || input_s[c_ptr] == '$'
|| input_s[c_ptr] == 'E' || input_s[c_ptr] == 'T' ||
input_s[c_ptr] == 'F')
                {
                    c_ptr--;
                    break;
                }
            }
            ch = "num";
        }
        express = "";
        string state_stack_top = state_stack.top();//状态栈顶
        //相应表达式
        string expression =
LR_Table[states_row[state_stack_top]][ter_line[ch]];
        /*cout << states_row[state_stack_top] << " " <<
ter_line[ch] << endl;
        cout << expression << endl;*/
        if (expression[0] == 'S')//移进
        {
            c_ptr++;
            expression = expression.substr(1);
            express = "Shift " + expression;
            show_info(input_s, express, c_ptr);
            symbol_stack.push(ch);//符号入栈
            state_stack.push(expression);//状态入栈
            s_state.push_back(expression);
            s_symbol.push_back(ch);
        }
        else if (expression[0] == 'R')//规约
        {
            expression = expression.substr(1);
            int index = stoi(expression);//规约后的状态
            express = expression_vec[index];
            string tmp = "reduce by ";
            tmp += express;
            show_info(input_s, tmp, c_ptr);
            //分离表达式的前后两部分
            string f_string = express.substr(0,
express.find_first_of("-"));
```

```
            string c_string =
express.substr(express.find_first_of(">") + 1);
            int top = c_string.size();//top 弹栈次数
            //cout << c_string << " " << top << endl;
            if (c_string == "num")
                top = 1;
            while (top--)
            {//弹栈
                s_state.erase(s_state.end() - 1);
                s_symbol.erase(s_symbol.end() - 1);
                state_stack.pop();
                symbol_stack.pop();
            }
            symbol_stack.push(f_string);
            s_symbol.push_back(f_string);
            //获取栈顶符号，为新入栈做准备
            state_stack_top = state_stack.top();
            row = states_row[state_stack_top];
            col = ter_line[f_string];
            state_stack.push(LR_Table[row][col]);
            s_state.push_back(LR_Table[row][col]);
        }
        else if
(LR_Table[states_row[state_stack_top]][ter_line[ch]] == "acc")
        {
            express = "acc";
            show_info(input_s, express, c_ptr);
            cout << "Accepted!分析完成" << endl;
            return;
        }
        else
        {
            express = "No acc";
            show_info(input_s, express, c_ptr);
            cout << "Failed! " << endl;
            return;
        }
    }
    return;
}
```