

编译原理程序设计2

实验报告

——语法分析程序的设计与实现



姓 名：崔 思 颖

学 号：2018211290

学 院：计算机学院

班 级：2018211305

**目录**

[实验题目 3](#_Toc56864337)

[实验要求 3](#_Toc56864338)

[程序设计说明 3](#_Toc56864339)

[算法流程图 4](#_Toc56864340)

[词法分析 4](#_Toc56864341)

[文法读取 6](#_Toc56864342)

[First、Follow集生成 7](#_Toc56864343)

[从顶向下的分析表 8](#_Toc56864344)

[LL(1)分析句子 9](#_Toc56864345)

[生成项目集 10](#_Toc56864346)

[自底向上的分析表 12](#_Toc56864347)

[LR分析句子 13](#_Toc56864348)

[测试报告 14](#_Toc56864349)

[输入及运行结果 14](#_Toc56864350)

[结果分析 24](#_Toc56864351)

# 实验题目

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 算数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

# 实验要求

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4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

# 程序设计说明

语法分析是编译过程的核心部分，它的主要任务是按照程序的语法规则，从由词法分析输出的源程序符号串中识别出各类语法成分，同时进行词法检查，为语义分析和代码生成做准备。本次实验采用了LL(1)及LR(0)分析方法，对输入的字符串进行分析，并输出分析过程中构造的单词识别结果、符号表、FIRST、FOLLOW集、预测分析表、分析步骤、项目集等信息。

该程序的输入为:

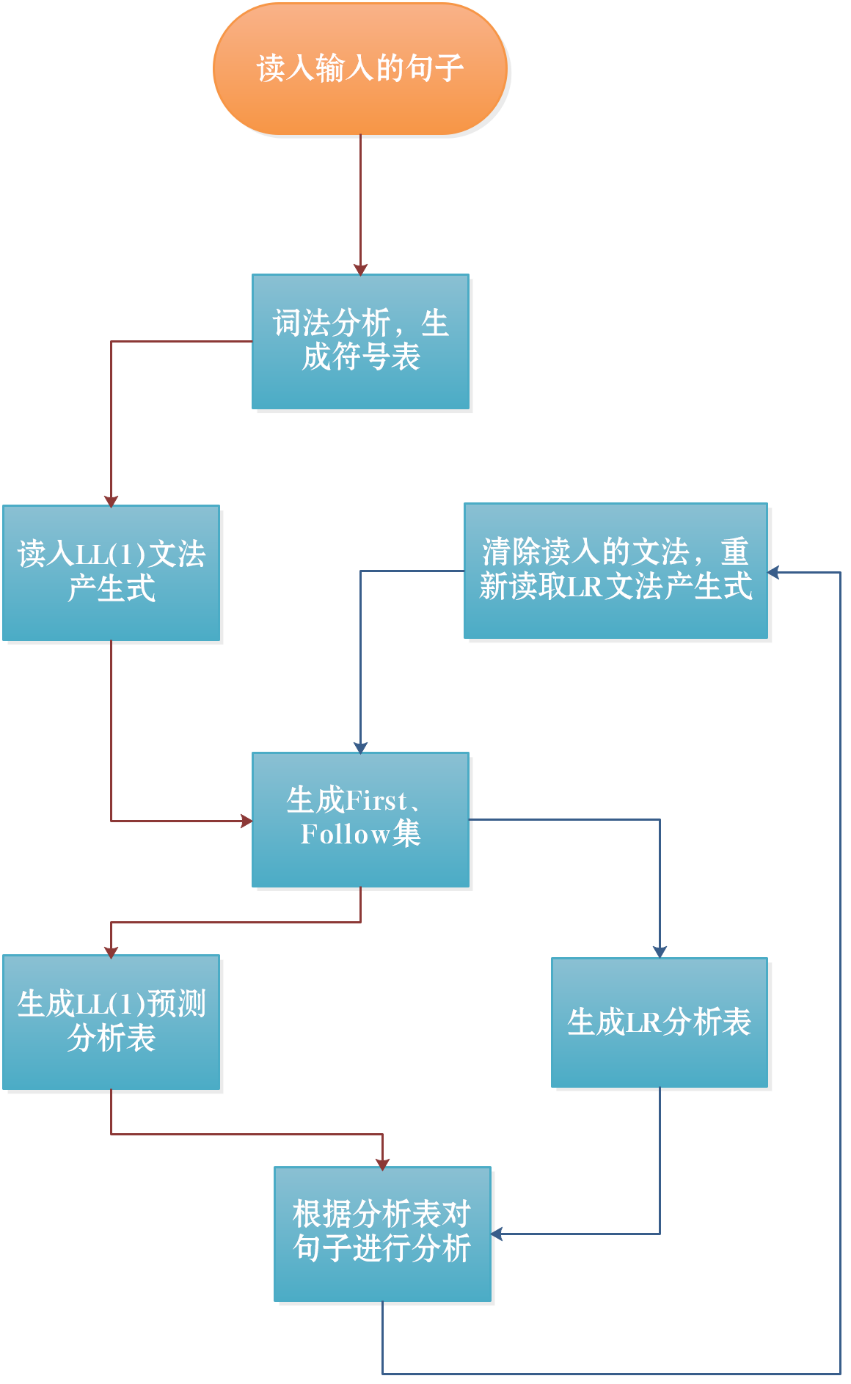
LL.txt：用LL(1)方法分析的文法产生式（已消除左递归）

LR.txt：用LR(0)方法分析的文法产生式（文法已转换为拓广文法）

input.txt: 待分析的句子

其中，文法产生式的非终结符使用“<>”括起，文法的各个成分用空格隔开，每一行为一个形式为“文法左部 -> 文法右部”的文法产生式，文法右部可以包含由“|”连接的多个产生式。空串由符号“@”表示。

## 算法流程图



## 词法分析

* **主要数据结构**

/\*单词属性\*/

struct property

{

string type;//单词属性类别

int line;//所在行数

property(string \_type, int \_line)

{

type = \_type;

line = \_line;

}

property(){}

};

/\*词法错误\*/

typedef struct error

{

errorType type;//错误类型

int line;//行号

string s;//错误单词

}Error;

enum errorType { DIGIT, COMMENT, STRING, CHAR };//拼写错误类型：数字、注释、字符串、特殊字符

ifstream in("input.txt");//输入文件

int Line = 1;//语句行数

int Sum;//字符总数

int Opt;//运算符总数

int Id;//标识符总数

int Con;//常数总数

int Split;//分界符总数

int Key;//关键字总数

vector<Error> Errors;//存储错误位置

map<string, struct property> Tokens;//符号表，由符号查找到其属性

vector<string> ana\_s ;//待分析的字符串

map<string, struct property> Tokens;//符号表，由符号查找到其属性

* **主要函数**

/\*词法分析程序\*/

void wordAna()

* **模块说明**

该程序的词法分析模块，我直接使用了上一次词法分析程序设计的代码，并做了一定简化，可以识别出单词的界符、实数、标识符、保留字、运算符、注释、转义符、字符串的属性，以及单词所在的行号、单词总数、各类单词的数量、词法错误，并输出到屏幕上。

在词法分析的过程中，句子的各个单词将存入符号表Tokens。在后续的句子分析程序模块中，会使用Token来把句子符号与非终结符进行匹配。

## 文法读取

* **主要数据结构**

/\*产生式\*/

typedef struct Node

{

string left;//产生式左部

vector<string>right;//产生式右部

int index = 0;//语法分析中的点

}node;

int grammar\_num;//文法数量

node grammar[400]; //文法

set<string>term;//终结符号

set<string>non\_term;//非终结符号

* **主要函数**

/\*判断为终结符还是非终结符\*/

bool is\_term(string s)

/\*读入文法\*/

void scan\_grammar(string filePath)

* **模块说明**

该模块的输入为包含文法产生式的文件名，程序进行分析后，将所有产生式储存到Node类型的结构体数组grammar中，并构造完成非终结符non\_term集合与终结符term集合，把最终读入的文法产生式拆成单个，按下标顺序打印在屏幕上。

Node结构体包含三个成员变量：left、right、index。left为“->”左边的非终结符，right为“->”右边的产生式，该产生式的各个成分被拆分为一个个string变量，按序存入right容器中。若右部包含“|”，那么各个产生式将被单独作为一个元素存入grammar数组里。index在LR项目集构造中起作用。

## First、Follow集生成

* **主要数据结构**

map<string, set<string>>first\_set;//first集

map<string, set<string>>follow\_set;//follow集

* **主要函数**

/\*生成first集\*/

void set\_first(string target)

/\*生成follow集\*/

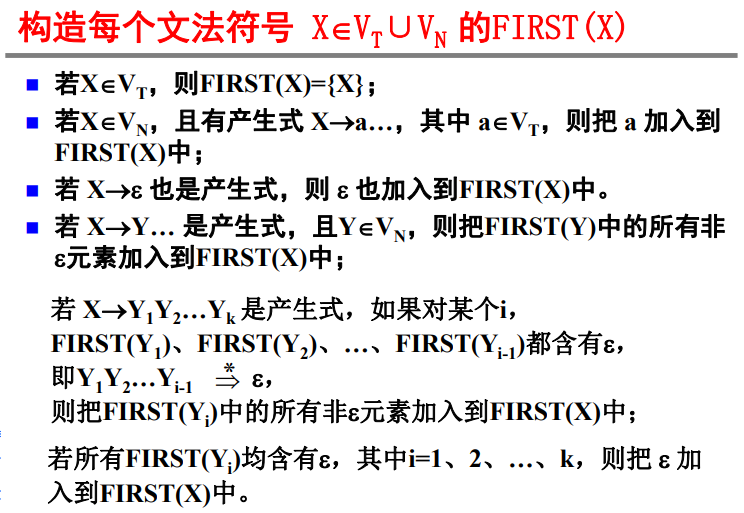
void set\_follow(string target,string start\_term)

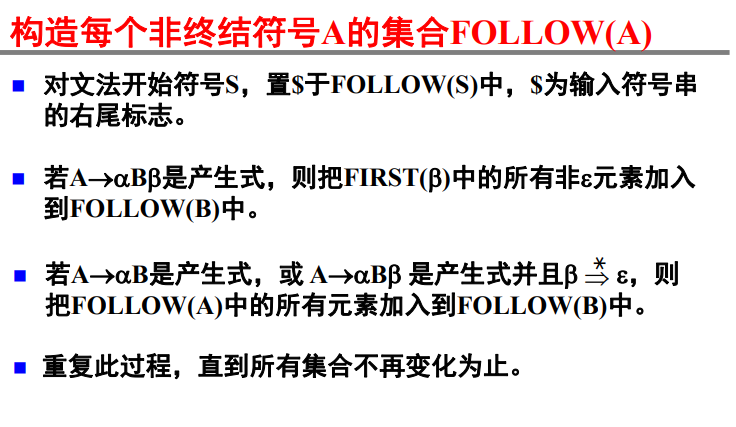
/\*产生first集和follow集，并输出到文件中\*/

void generate\_FirstAndFollow(string start\_term)

* **模块说明**

该模块按照如下算法，自动生成所读取文法的FIRST、FOLLOW集：





最后，FIRST、FOLLOW集将以哈希表的形式存储在程序中，通过非终结符字符串能够访问到其相应的FIRST、FOLLOW集。

## 从顶向下的分析表

* **主要数据结构**

map<pair<string, string>, int>analysisTable\_1;//自上而下分析表 (<文法左部, 终结符号+@>, 产生式下标)

* **主要函数**

/\*生成由上至下的分析表\*/

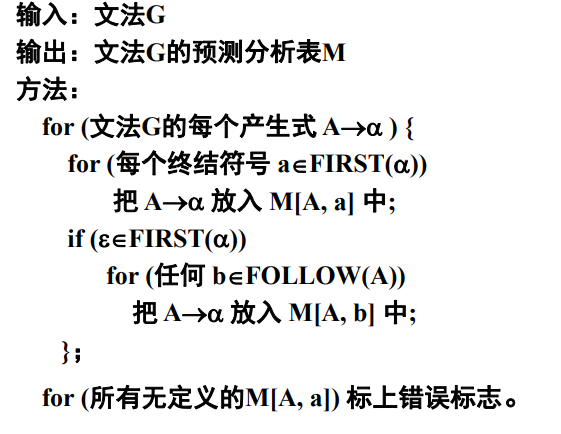
map<pair<string, string>, int> getTopToBottomTable()

/\*打印预测分析表\*/

void display\_Table()

* **模块说明**

该模块按照如下方法构造预测分析表：



表格每个单元的结构为(<文法左部，终结符号+@>,产生式下标)。

## LL(1)分析句子

* **主要数据结构**

string Stack[100];//分析栈

int ip = 1;//栈顶元素下标

vector<string> ana\_s\_LL = ana\_s;//复制待分析串

int gra\_index=-2;//使用到的文法规则下标

vector<string>::iterator it = ana\_s\_LL.begin();//遍历待分析句子各个成分的迭代器

* **主要函数**

/\*打印分析LL分析句子时的栈、输入、输出\*/

void printAnaStepLL(string S[], int ip,string input,int gra\_index)

/\*将单词转换为文法中的非终结符\*/

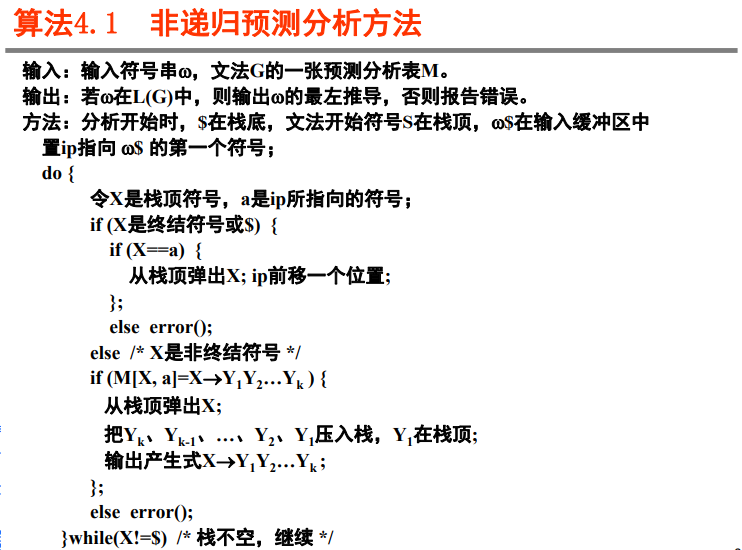
string tokenToTerm(string token)

/\*LL自顶向下分析句子\*/

void analyzeLL()

* **模块说明**

该模块基于构造好的预测分析表analysisTable\_1来对输入进行非递归预测分析，并将每一步循环里的分析栈、输入、输出情况打印在屏幕上。分析算法如下：



## 生成项目集

* **主要数据结构**

/\*项目集\*/

struct I

{

int id;//项目集编号

vector<node> vec;//项目集

map<string, int> m; //转化关系

};

vector<I> projectSet; //项目集

* **主要函数**

/\*清空文法信息\*/

void grammar\_clear()

/\*判断右部是否相同\*/

bool judgeIs\_SameVecOfString(vector<string> tmps, vector<string> vs)

/\*遍历判断是否为重复项目集\*/

bool judgeIs\_SameVecOfNode(vector<node> tmp, vector<node> v)

/\*打印项目集\*/

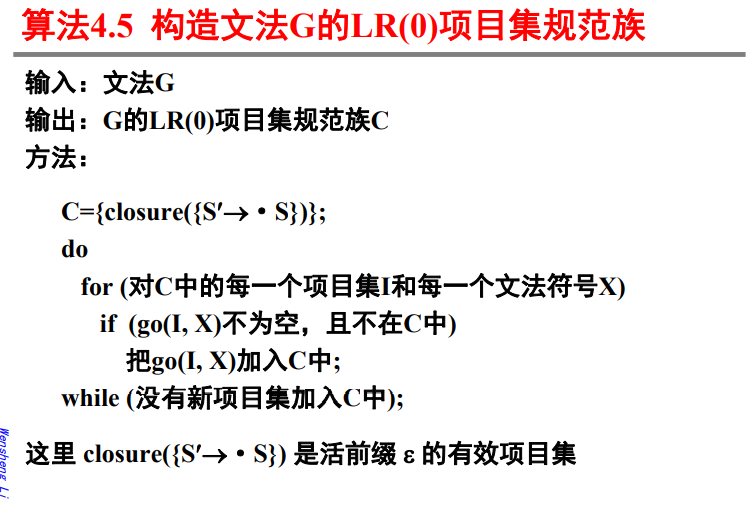
void printI(I pi)

/\*构造项目集\*/

void generateProjectSet()

* **模块说明**

该模块将清空之前读入的LL(1)分析的文法，重新读入LR.txt中的文法，并构造FIRST、FOLLOW集以及项目集规范族。每个项目集中的信息为之前定义的Node结构，即与文法的类型相同，项目的点位置为Node中的index值。转化关系m可以索引到读入终结符后转入的下一个项目集的id。项目集规范族构造的算法如下：



最后输出每个项目集的名称、所含项目，以及接收某个终结符后转入的下一个项目集的id。

## 自底向上的分析表

* **主要数据结构**

map<pair<int, string>, pair<string, int>> ACTION; //ACTION表

map<pair<int, string>, int> GOTO; //GOTO表

* **主要函数**

/\*生成LR分析表\*/

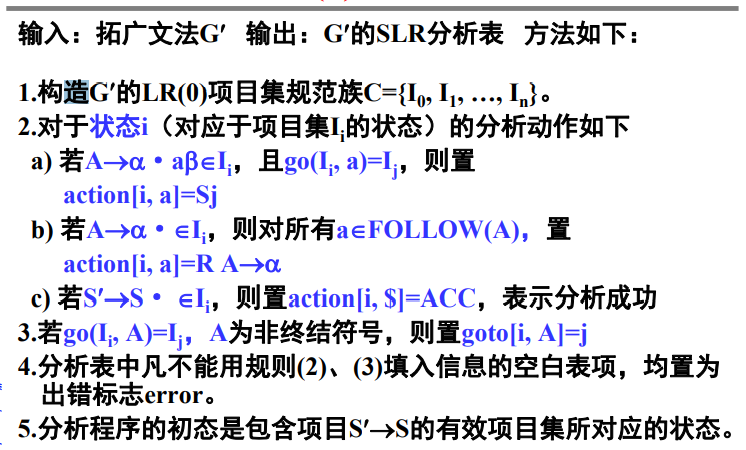
void generateLR0Table()

/\*打印LR分析表\*/

void printLR0Table()

* **模块说明**

该模块由已构造完成的项目集生成分析表，ACTION和GOTO表分成两个哈希表存储。具体构造方法如下：



## LR分析句子

* **主要数据结构**

pair<int,string> Stack[100];//分析栈，包含符号栈和状态栈

int ip = 0;//栈顶元素下标

vector<string> ana\_s\_LR = ana\_s;//复制待分析的句子

int gra\_index = -2;//使用到的文法规则下标，为-2表示空

vector<string>::iterator it = ana\_s\_LR.begin();//遍历待分析句子成分的迭代器

string token\_switch = tokenToTerm(\*it);//将句子成分转化为文法中的非终结符

string action = ACTION[make\_pair(Stack[ip].first, token\_switch)].first;//当前采取的分析动作

int next\_state = ACTION[make\_pair(Stack[ip].first, token\_switch)].second;//移进的项目集ID或规约使用产生式的下标

* **主要函数**

/\*打印分析LR分析句子时的栈、输入、分析动作\*/

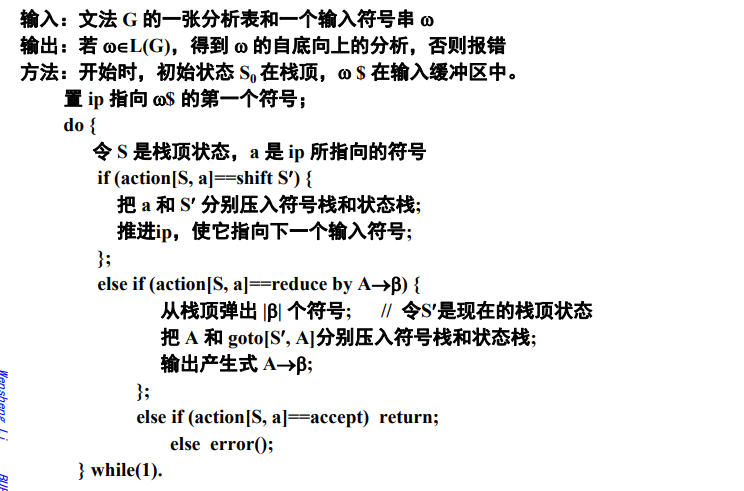
void printAnaStepLR(pair<int, string> S[], int ip, string input, int gra\_index, string action,int next\_state)

/\*LR自底向上分析句子\*/

void analyzeLR()

* **模块说明**

该模块由已生成的分析表对输入串进行自底向上分析，并将每一步循环中的分析栈、输入、分析动作打印在屏幕上，直到acc接受符号串为止。具体分析方法如下：



**源程序及可执行程序见附件**

# 测试报告

## 输入及运行结果

本次测试共测试了三组句子，使用的文法规则为实验题目给出的文法生成式如下：

E→ E+T | E–T | T

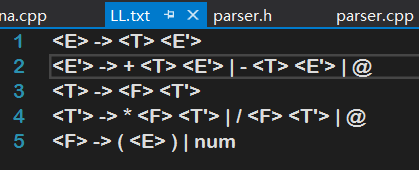
T→ T\*F | T/F | F

F→ (E) | num

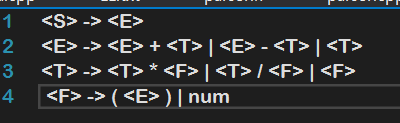
并转化为程序能够正确读取的格式如下：

（LL.txt为消除左递归后的等价文法，LR.txt为拓广后的等价文法）

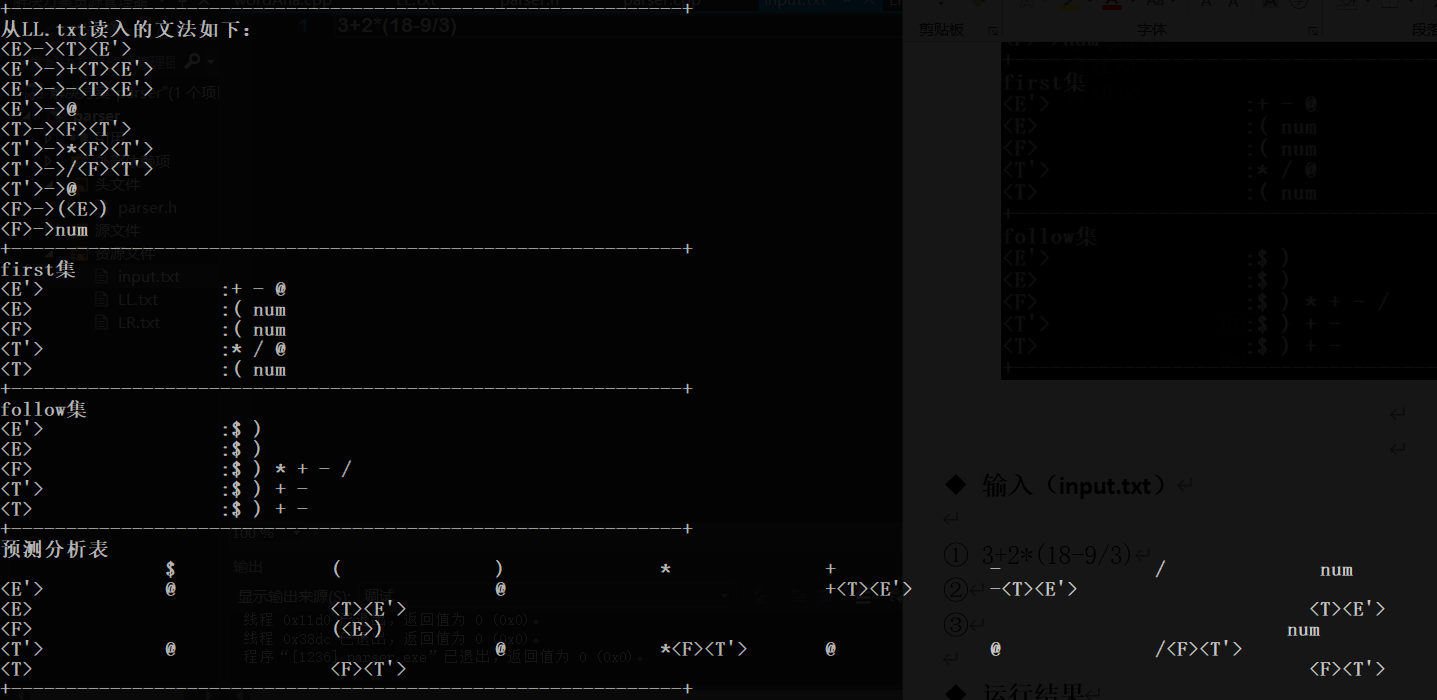
**LL.txt**

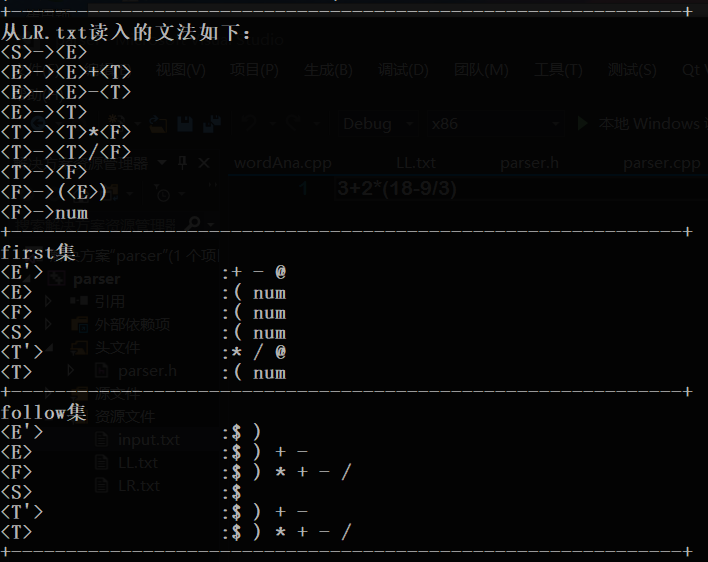


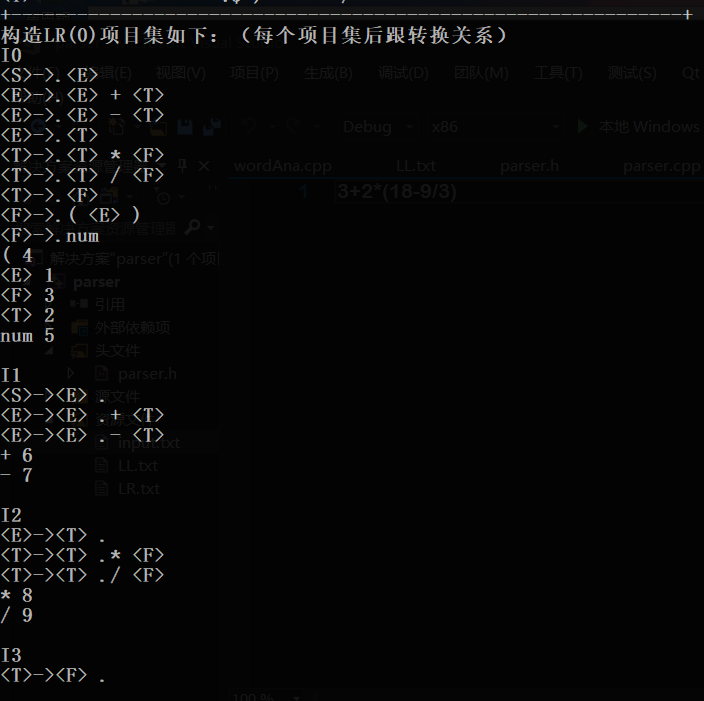
**LR.txt**

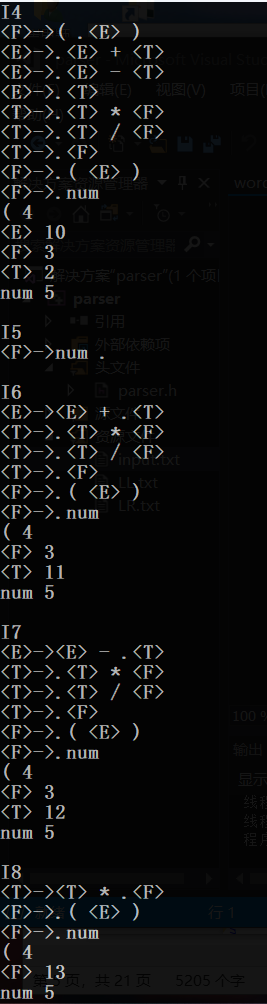
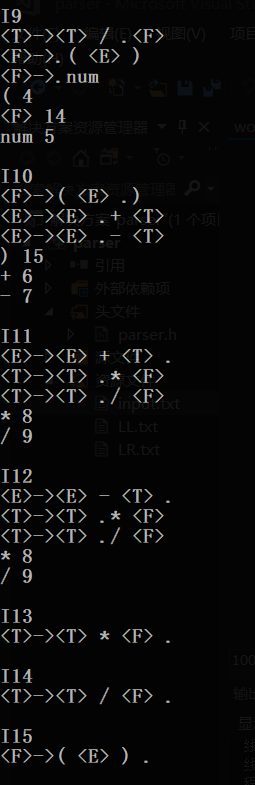


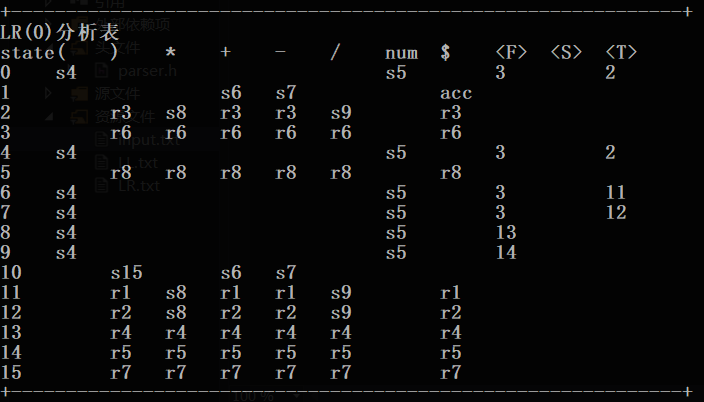
* **文法分析结果**









* **输入（input.txt）**

① 3+2\*(18-9/3)

② 5\*(10/2)-88

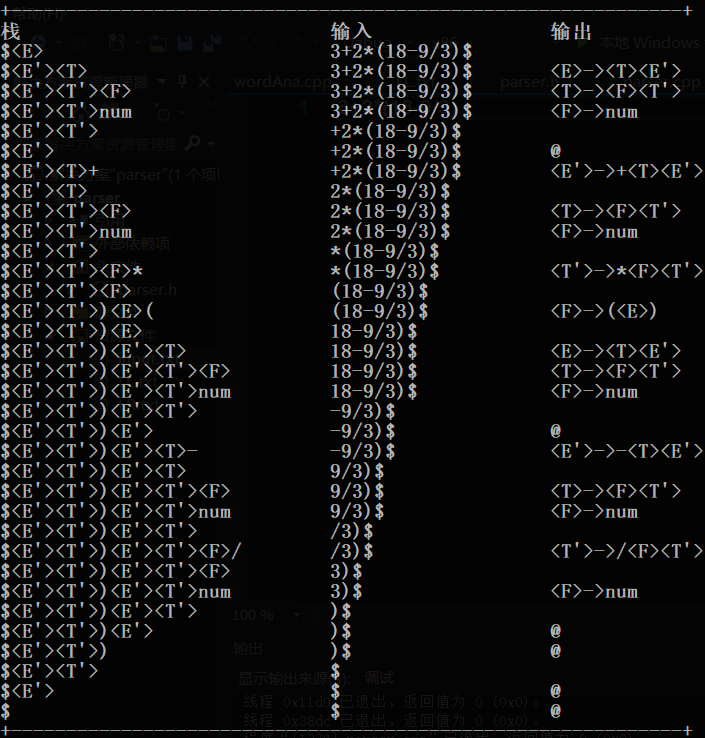
③ 2048+1024-9\*10086

* **运行结果**

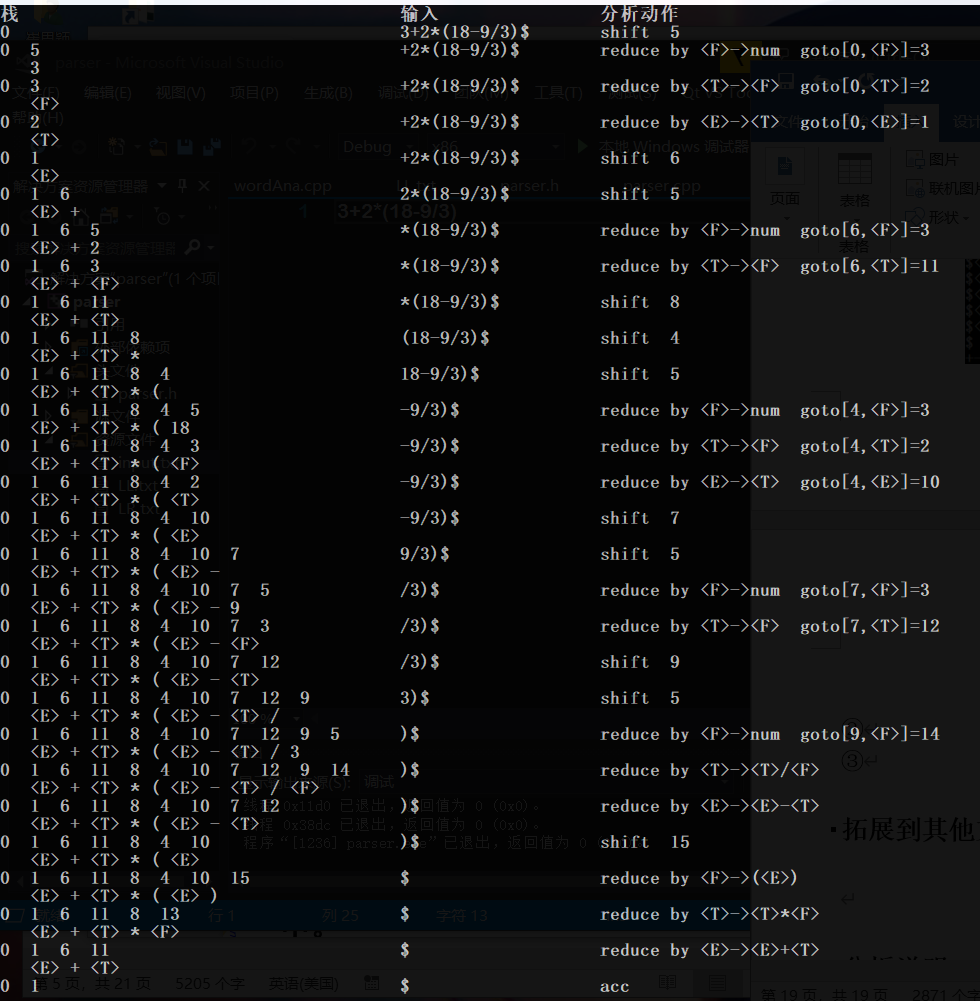
①3+2\*(18-9/3)



**LL分析**



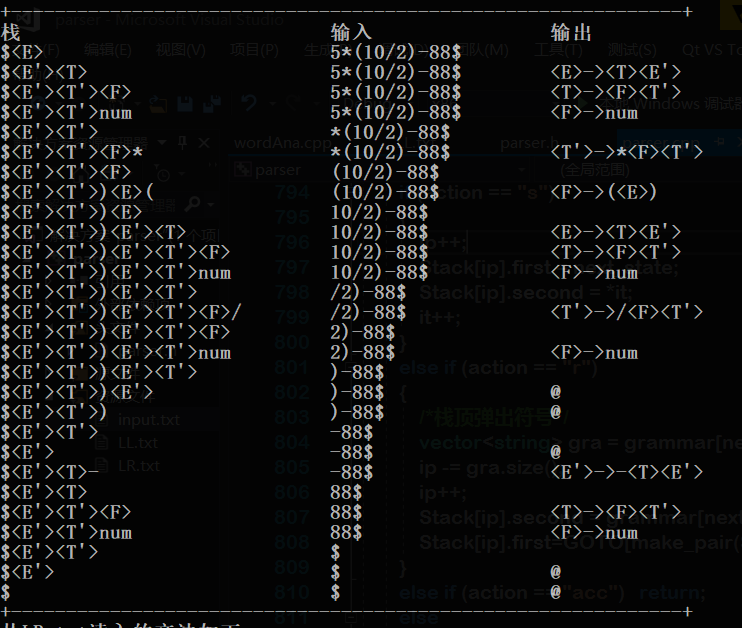
**LR分析**



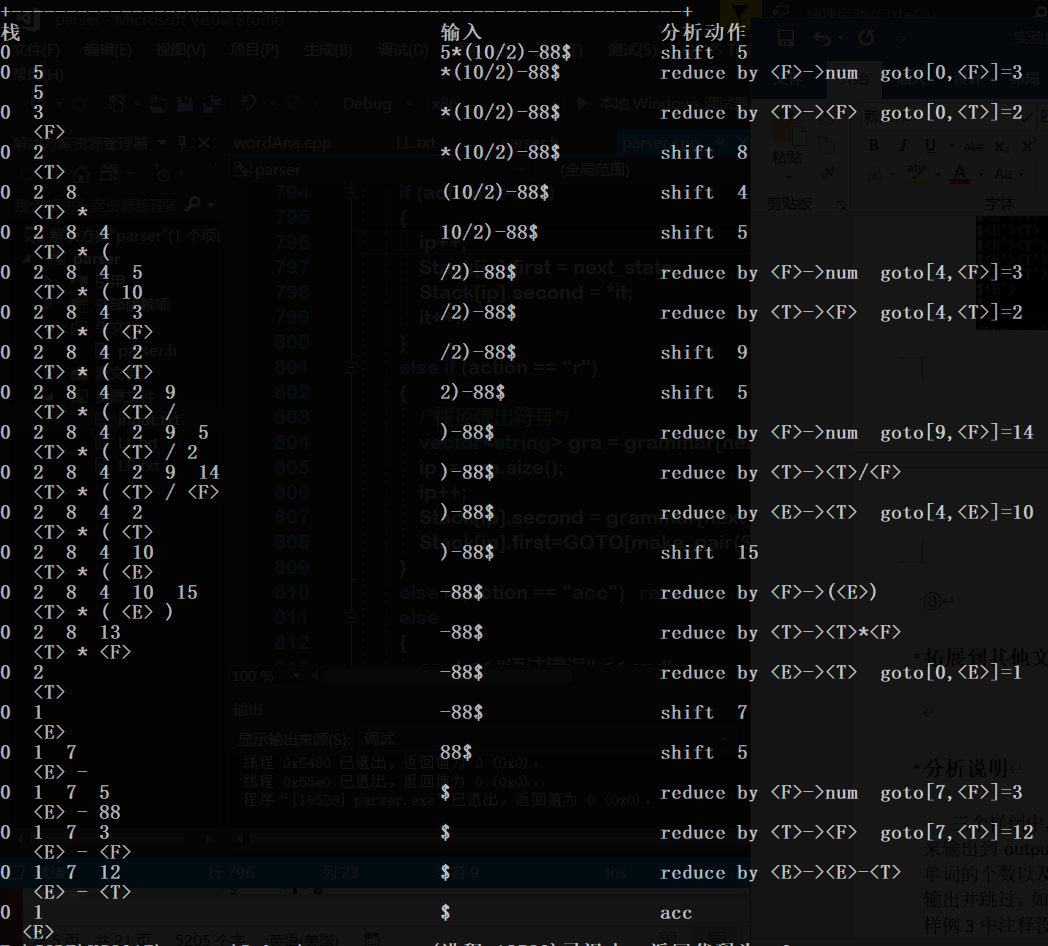
②5\*(10/2)-88



**LL分析**



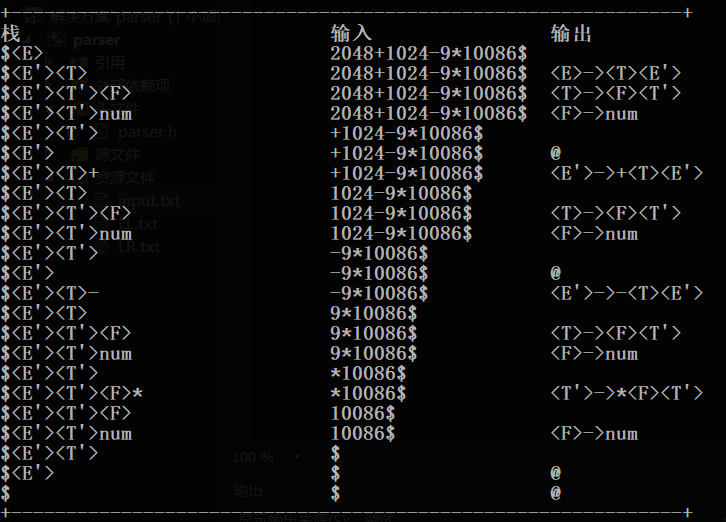
**LR分析**



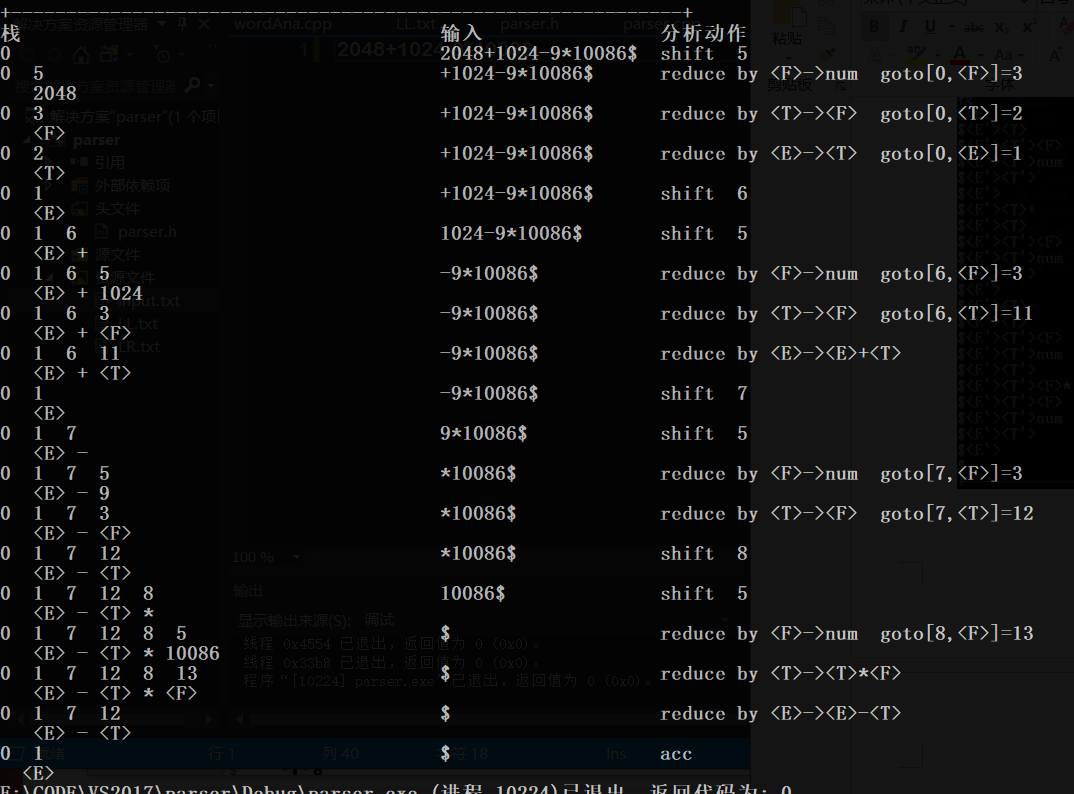
③2048+1024-9\*10086



**LL分析**



**LR分析**



## 结果分析

三个样例中，程序均能准确将input.txt中的算术表达式分析并接受。程序调用了上次程序设计作业的词法分析程序，首先输出对句子单词的识别结果，生成单词符号表。接着读入文法，分别用LL(1)与LR(0)的方法生成分析表，将对句子的分析过程清晰地输出。程序的各个模块相对独立，且灵活性和可拓展性较强，下次的程序设计可以继续在此基础上进行拓展。

通过本次实验，我加深了对语法分析原理的理解，也进一步了解了平时使用的编译器其背后的原理，接下来的实验我也要再接再厉、认真完成。