编译原理与技术

LL(1)语法分析程序设计报告

姓　　名：刘立敏

学　　号：2018211398

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：2018211308

指导老师：张玉洁

目录

1、实验内容

编写语法分析程序，实现对算术表达式的语法分析。要求所分析的算术表达式由如下的文法产生（也可以实现通用文法）：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | num

编写 LL(1)语法分析程序，为给定文法自动构造预测分析表，并构造 LL(1)预测分析程序

2、实验环境

编程语言：C++

集成开发环境：Visual Studio 2019，Qt Creator 4.11

3、总体设计

该语法分析程序实现了对通用文法的分析，因此它的输入有两个，一是用户给定的文法，二是一个字符串。语法分析程序调用了上次实验实现的词法分析程序，将字符串转化为记号流，作为输入串，然后利用文法生成的预测分析表，对输入串进行分析。输出包含中间每一步处理得到的结果，包括等价转换后得到的 LL(1)文法，FIRST 集，FOLLOW 集，预测分析表，和对输入串进行的分析过程。如果用户给出的文法不能通过等价变换变成 LL(1)文法，或者是空文法，程序也会给出提示信息。

该语法分析程序分为四个阶段。

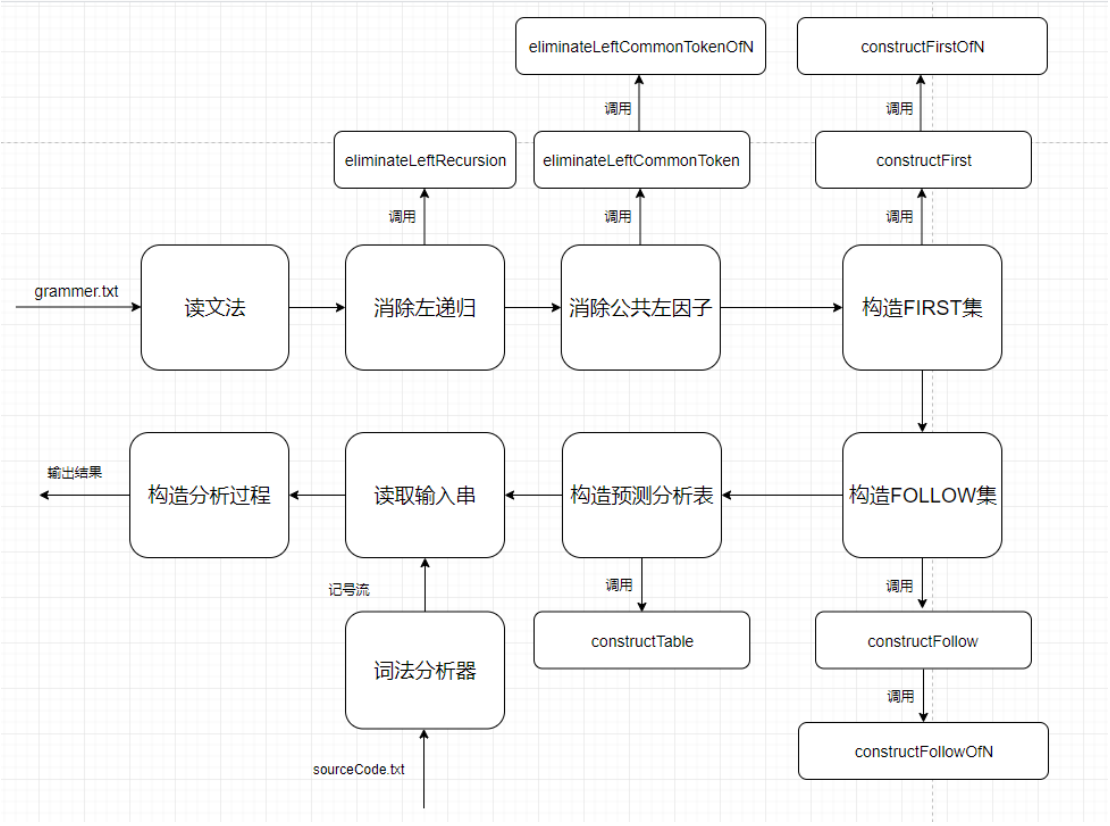
第一阶段：读取文法。

第二阶段：对文法进行处理，包括消左递归，消公共左因子，构造 FIRST 集，构造 FOLLOW 集，构造预测分析表。

第三阶段：读取输入串，对其进行词法分析，将分析所得的记号流作为语法分析程序的输入。

第四阶段：利用预测分析表对输入串进行分析。

流程图设计如下：



4、详细设计

4.1、输入与输出

4.1.1、文法输入

该程序实现了对通用文法的分析。文法输入采取文件读取的方式。程序默认使用课本上给出的文法进行分析，如果需要修改文法，请修改配置文件 grammer.txt。

默认的配置文件 grammer.txt 如下：

```
1   #nonterminals
2   E T F
3
4   #terminals
5   + - * / ( ) num
6
7   #startSymbol
8   E
9
10  #productions
11  E -> E + T | E - T | T
12  T -> T * F | T / F | F
13  F -> ( E ) | num
```

其中 1、4、7、10 行为注释，分别意为非终结符，终结符，起始符，产生式。

第 2 行为非终结符，（要求 1）多个非终结符之间以空格分隔开。

第 5 行为终结符，（要求 2）多个终结符之间以空格分隔开。

第 8 行为起始符。

第 11 行以后为多个产生式。每行的产生式左边为同一个非终结符，（要求 3）右边如果有多个候选式，则以字符‘|’分隔开。（要求 4）候选式中的每个终结符或非终结符之间必须以空格分隔，否则程序会将多个符号视为一个符号。（要求 5）左边的非终结符与中间的箭头“->”之间以空格分隔，否则程序会将箭头视作非终结符的一部分。

如果需要修改文法，请直接在默认文法的基础上修改，并且（要求 6）确保非终结符、终结符、起始符位于第 2、5、8 行，产生式位于第 11 行以后。

注：上面所有以空格分隔的要求均允许冗余的空格。例如下面的文法，添加了冗余的空格使得结构看起来非常混乱，但是仍然满足上面的六个要求，所以程序能正常运行。

```
1   #nonterminals
2        E    T    F
3
4   #terminals
5      +    - * / (     )    num
6
7   #startSymbol
8           E
9
10  #productions
11         E         ->      E +        T |E      - T|          T
12  T      ->T * F| T / F |              F
13  F -> ( E ) | num
```

4.1.2、字符串输入

输入的字符串在文件 sourceCode.txt 中。程序读取字符串后，将其交给词法分析器处理，得到 token 流。然后语法分析器再利用预测分析表分析该 token 流。

4.1.3、输出

输出以控制台方式输出。

如果给定的文法可以转化为等价的 LL(1)文法，那么该语法分析程序会依次输出转化后的 LL(1)文法、FIRST 集、FOLLOW 集、预测分析表、预测分析过程。如果不能转换，或者是空文法，程序输出该文法不是 LL(1)文法或该文法是空文法。

在输出格式上，为了美观，我都用 setw()函数指定了输出字段的宽度。默认的字段宽度参见文件”arg.h”

```
#pragma once
/*
    该文件定义了字符串显示宽度
*/
//产生式的宽度
#define PRODUCTION_WIDTH 20

//非终结符宽度
#define NONTERMINAL_WIDTH 20

//栈宽度
#define STACK_WIDTH 20

//输入串宽度
#define INPUT_STRING_WIDTH 20

//序号宽度
#define SEQUENCE_WIDTH 3
```

考虑到不同文法非终结符的长度不同，如果长度超过了上面指定的宽度，就会导致输出显示结构混乱，因此可以修改上面的常量值。

4.2、数据结构

4.2.1、文法

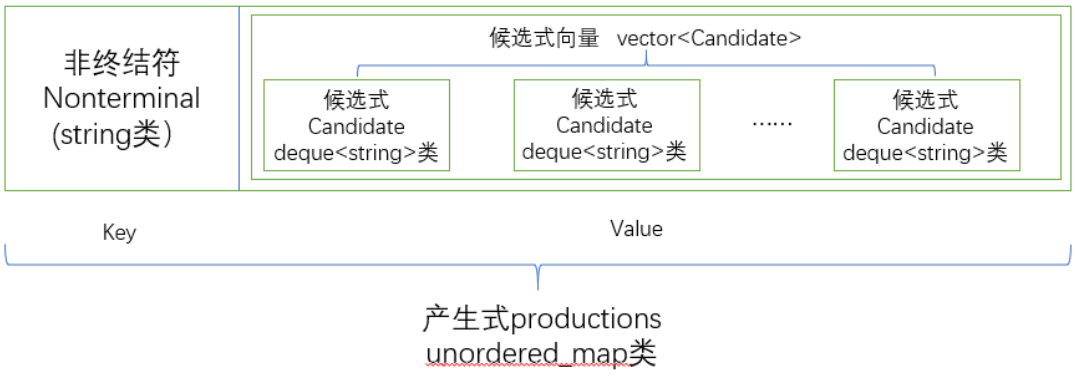
对于一个文法，我们需要存储它的非终结符、终结符、起始符和产生式信息。非终结符一般为单个的大写字母，但考虑到可以形如”E’”(修改文法后导致带有一撇)，因此每个非终结符用 C++的 string 类存储，所有的非终结符组成一个 string 类的容器 vector。终结符和起始符同理。

产生式的数据结构相对比较复杂。一个产生式的左边是非终结符，右边是（一个或多个）候选式。我们先定义候选式的结构如下：

```
typedef deque<string> Candidate;
```

一个候选式可能由多个终结符或非终结符组成，每个符号是一个 string 类，因此候选式的结构被定义为一个由 string 类元素构成的双端队列 deque，之所以选择双端队列，是因为便于我们加入或删除队列头部或尾部的元素。

一个非终结符对应多个候选式,这些候选式组成一个容器 vector<Candidate>,作为非终结符(key)对应的值（value)。因此产生式的整体结构是一个 map，为保证顺序，我们选用 unordered_map 来替代 map。



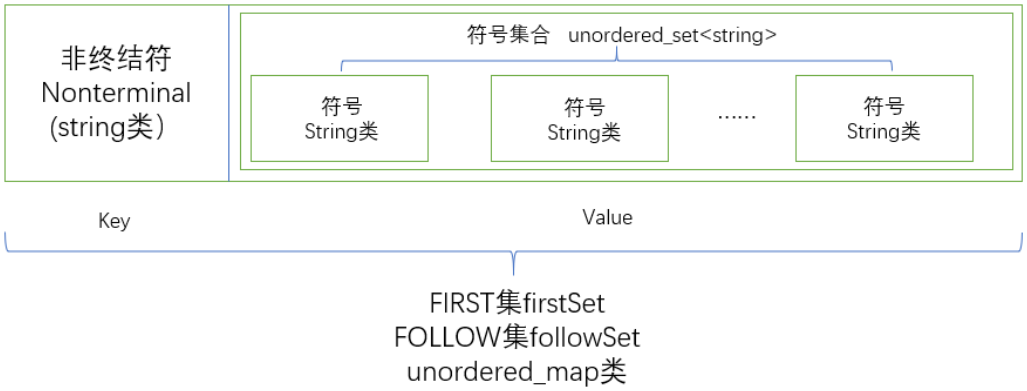
综上，文法存储结构为：

```
typedef string Nonterminal,Terminal;
typedef deque<string> Candidate;
class Parser
{
private:
    vector<Nonterminal>nonterminals;                    //非终结符
    vector<Terminal>terminals;                          //终结符
    Nonterminal startSymbol;                            //起始符
    unordered_map<Nonterminal,vector<Candidate>>productions;//产生式
    ……
};
```

4.2.2、FIRST 集和 FOLLOW 集

FIRST 集和 FOLLOW 集的数据结构与上面的产生式相似，不过现在一个非终结符对应的是由多个符号（string 类）构成的集合（unordered_set）。具体如下：

```
unordered_map<Nonterminal, unordered_set<string>>firstSet;  //first集
unordered_map<Nonterminal, unordered_set<string>>followSet; //follow 集
```



4.2.3、预测分析表

预测分析表的实际结构

	终结符 1	终结符 2	……	$
非终结符 1				
非终结符 2				
……				

存储时，依然采用 map，其中 key 是一个 pair 类型，它的 first 和 second 均为 string 类，分别表示非终结符和终结符（或$）;value 是 Candidate 类（即 deque<string>）,表示用到的候选式。

```
map<pair<string, string>, Candidate> table;   //预测分析表
```

4.2.4、符号栈和输入串

```
stack<string>tokenStack;          //符号栈
deque<Token>inputString;          //输入串
```

其中 Token 结构定义如下，它是词法分析器处理的结构。

```
typedef struct
{
    TokenType tokenType;
    string str;
}Token;
```

4.2.5、Parser 类

Parser 类是语法分析器的主体。

```
class Parser
{
private:
    bool isLL1;        //文法是否为LL1文法
    bool isEmpty;      //文法是否为空文法

    vector<Nonterminal>nonterminals;                        //非终结符
    vector<Terminal>terminals;                              //终结符
    Nonterminal startSymbol;                                //起始符
    unordered_map<Nonterminal, vector<Candidate>>productions;//产生式

    unordered_map<Nonterminal, unordered_set<string>>firstSet;
    //first集
```

```
    unordered_map<Nonterminal, unordered_set<string>>followSet; //follow集

    map<pair<string, string>, Candidate> table;   //预测分析表

    stack<string>tokenStack;        //符号栈
    deque<Token>inputString;        //输入串
};
```

4.3、方法

此处仅列举并简单描述 Parser 类的方法，其中一些重要的步骤和算法将在下一小节详细说明。

```
class Parser
{
public:
    //构造函数
    Parser() :isLL1(true),isEmpty(false) {};

    //判断是否是LL1文法
    bool getIsLL1() { return isLL1; }

    //判断是否是空文法
    bool getIsEmpty() { return isEmpty; }

    //判断字符串s是否为终结符
    inline bool isTerminal(string s);

    //判断字符串s是否为非终结符
    inline bool isNonterminal(string s);

    //读取文法
    void getGrammer();

    //打印文法
    void printGrammer();

    //消左递归
    void eliminateLeftRecursion();

    //消除公共左因子
    void eliminateLeftCommonToken();

    //消除某个非终结符产生式的公共左因子
    void eliminateLeftCommonTokenOfN(pair<Nonterminal, vector<Candidate>>);

    //构造所有非终结符的FIRST集
    void constructFirst();
```

```
    //构造非终结符N的FIRST集
    void constructFirstOfN(Nonterminal,map<Nonterminal, bool>*);

    //构造所有非终结符的FOLLOW集
    void constructFollow();

    //构造非终结符N的FOLLOW集
    void constructFollowOfN(Nonterminal,map<Nonterminal, bool>*,Nonterminal);

    //获取一个候选式的FIRST集
    unordered_set<string> getFirstOfCandidate(Candidate);

    //打印first集
    void printFirstSet();

    //打印follow集
    void printFollowSet();

    //构造预测分析表
    void constructTable();

    //打印预测分析表
    void printTable();

    //读取词法分析器处理过后的输入串
    void getInputString(deque<Token>);

    //分析过程
    void analysis();

    //输出当前栈内容
    inline void printStack();

    //输出当前输入串内容
    inline void printInputString();

    //将候选式（是一个string的deque）连接起来转化为string
    inline string candidate2String(Candidate);
};
```

4.4、算法

受篇幅所限，本部分列举的算法大多以伪码或算法描述来展示，具体内容请参见源代码。

4.4.1、读取文法

在文法的读取上首先要注意的是将字符串以某个指定的字符进行分割，我采用的方法是使用字符串流 stringstream。

```
inline Candidate split(string s, char c)        //将字符串s以字符c分割，得到的子串构成一个string类的双端队列
{
    Candidate candidate;
    string tmp;                                 //用来暂存分割后的字符串
    stringstream input(s);                      //用字符串s初始化字符串流
    while (getline(input, tmp, c))
        if (tmp.find_first_not_of(' ') != -1)   //tmp只由空格构成时，不加入candidate
            candidate.push_back(tmp);
    return candidate;
}
```

另一方面，还需要处理用户糟糕的输入情况（文法中添加了冗余空格），只需要一个函数去除掉字符串首尾的空格即可。

```
inline void trim(string& s)             //去掉一个字符串首尾的空格
{
    if (!s.empty())
    {
        s.erase(0, s.find_first_not_of(" "));
        s.erase(s.find_last_not_of(" ") + 1);
    }
}
```

最后使用了 isEmpty 这个 bool 类型变量来标记是否为空文法，一旦文法的四要素（非终结符、终结符、起始符、产生式）中有一个为空，那么将 isEmpty 置为 true，表示这是一个空文法。

对默认文法进行检测

```
#nonterminals
E T F

#terminals
+ - * / ( ) num

#startSymbol
E

#productions
E -> E+T | E-T | T
T -> T*F | T/F | F
F -> (E) | num
```

```
非终结符:
E T F

终结符:
+       -       *       /       (       )       num

起始符:
E

产生式:
E->E+T
E->E-T
E->T
T->T*F
T->T/F
T->F
F->(E)
F->num
```

删去起始符 E，变为空文法

```
#nonterminals
E A B L

#terminals
num id ( ) ,

#startSymbol


#productions
E -> A | B
A -> num |id
B -> ( L )
L -> E , L | E
```

```
该文法为空文法
```

修改文法后的检测效果（下图中故意添加冗余的空格以验证用户在糟糕的输入下也能正确运行）

```
#nonterminals
E       E'      T T' F

#terminals
+       *   ( ) id        epsilon

#startSymbol
E

#productions
E ->    TE'
   E'       -> +TE'| epsilon
T    ->     FT'
    T' -> *FT' |      epsilon
F       ->       (E)|id
```

```
非终结符:
E E' T T' F

终结符:
+       *       (       )       id      epsilon

起始符:
E

产生式:
E->TE'
E'->+TE'
E'->epsilon
T->FT'
T'->*FT'
T'->epsilon
F->(E)
F->id
```

4.4.2、消左递归

若一个文法中存在形如A→Aα的产生式,那么存在左递归。使用LL(1)分析方法需要消除左递归。消除左递归的函数如下：

```
void Parser::eliminateLeftRecursion()
```

对于非终结符A，有产生式$A\rightarrow A\alpha_1|\ A\alpha_2|\cdots|\ A\alpha_m|\ \beta_1|\ \beta_2|\cdots|\ \beta_n$，该函数将其转化为

$$A\rightarrow \beta_1 A'\ |\beta_2\ A'\ |\cdots|\beta_n\ A'$$

$$A'\ \rightarrow \alpha_1 A'\ |\alpha_2\ A'\ |\cdots|\alpha_n\ A'\ |\ \varepsilon$$

```
#nonterminals
E T F

#terminals
+ - * / ( ) num

#startSymbol
E

#productions
E -> E + T | E - T | T
T -> T * F | T / F | F
F -> ( E ) | num
```

```
非终结符:
E T F E' T'

终结符:
+       -       *       /       (       )       num

起始符:
E

产生式:
E          -> T E'
E'         -> + T E'
E'         -> - T E'
E'         -> ε
T          -> F T'
T'         -> * F T'
T'         -> / F T'
T'         -> ε
F          -> ( E )
F          -> num
```

4.4.3、消公共左因子

如果产生式形如：A→$\alpha\beta_1 | \alpha\beta_2$，则可提取左公因子将其改造为：

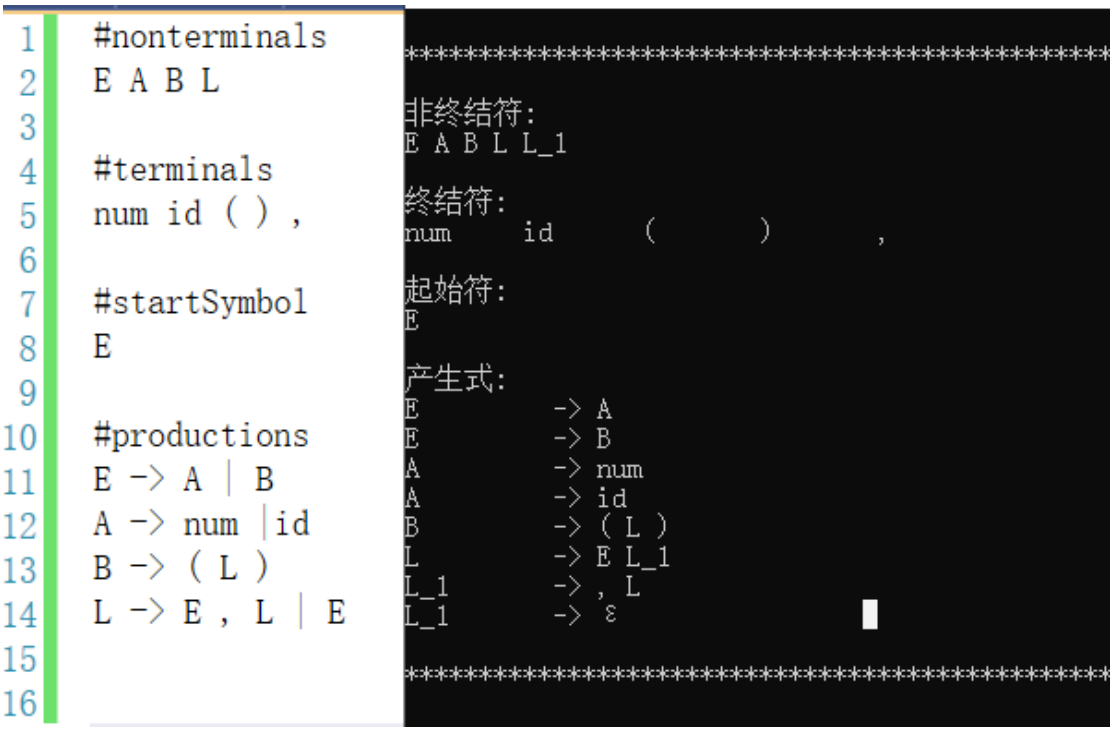
① $A \longrightarrow \alpha A_1$

② $A_1 \longrightarrow \beta_1 | \beta_2$

算法描述：

```
void Parser::eliminateLeftCommonToken()
{
    对每个非终结符，依次调用 eliminateLeftCommonTokenOfN。
    将新产生的非终结符加入到非终结符集合中；
}
void Parser::eliminateLeftCommonTokenOfN(pair<Nonterminal,vector<Candidate>> production)
{
    判断当前非终结符的所有候选式中是否有公共左因子；
    如果有，则提取左公因子，生成新的非终结符和新的产生式，并修改原来的产生式（按上述①和②式）；
    判断新生成的非终结符的候选式中是否有公共左因子，如果有，对该非终结符调用 eliminateLeftCommonTokenOfN；
}
```

注：对于原非终结符 S，新生成的非终结符的形式为 S_1,S_2…以此类推。



4.4.4、构造 FIRST 集

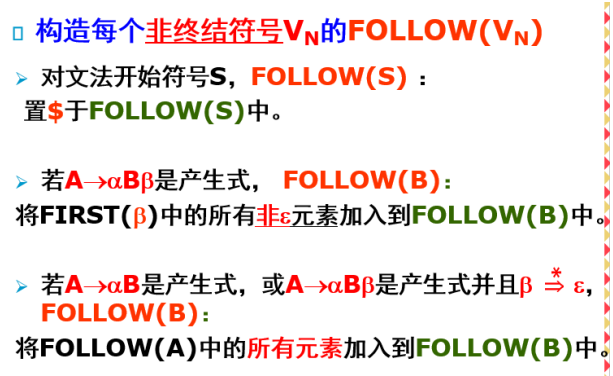
构造 FIRST 集的过程参考了深度优先搜索（DFS）的思想。先用 bool 类型变量标记当前非终结符的 FIRST 集是否构造完毕，如果没有，则进行构造，如果需要用到另一个非终结符的 FIRST 集，则先构造那个非终结符的 FIRST 集，构造完后再对当前非终结符进行构造。

```
void Parser::constructFirst()
{
    map<Nonterminal, bool>finished;          //记录某个非终结符的First集是否构造完毕

    for (auto nonteminal : nonterminals)     //初始化
        finished[nonteminal] = false;
    for (auto prodution : productions)
        firstSet[prodution.first].clear();

    for (auto prodution : productions)       //依次构造每个非终结符的FIRST集
        if(!finished[prodution.first])
            constructFirstOfN(prodution.first,&finished);
}
void Parser::constructFirstOfN(Nonterminal N,map<Nonterminal,bool> *finished)
{
    如果非终结符 N 已经完成构造，退出该函数;
    遍历非终结符 N 的每个候选式
    {
        遍历候选式的每个符号
        {
```

```
            if(当前符号是终结符或 epsilon)
                将当前符号加入到 N 的 FIRST 集中;
            else if(当前符号是非终结符)
            {
                先构造当前符号的 FIRST 集;
                将当前符号 FIRST 集除去 epsilon（如果有的话）加入到 N 的 FIRST 集中;
            }
        }
        if(候选式中的每个符号都可以推空)
            将 epsilon 加入到 N 的 FIRST 集中;
    }
    标记非终结符 N 完成构造;
}
```

4.4.5、构造 FOLLOW 集

FOLLOW 集的构造与 FIRST 集类似，也是利用 DFS 思想进行递归构造。



算法描述如上所示，但是要考虑一种特殊情况：

$$A \rightarrow aB$$

$$B \rightarrow aA$$

构造 B 的 FOLLOW 集时要用到 A 的 FOLLOW 集，构造 A 的 FOLLOW 时要用到 B 的 FOLLOW 集，两者互相递归调用，导致程序无穷无尽地进行下去。因此，在函数 constructFollowOfN 中，需要一个额外的参数来记录上次进行 FOLLOW 集构造的是哪一个非终结符。如果当前构造 B 的 FOLLOW 集要用到 A 的 FOLLOW 集，而上次构造的恰好是 A 的 FOLLOW 集，那么就不对 A 调用 constructFollowOfN。

```
void Parser::constructFollow()
void Parser::constructFollowOfN(Nonterminal N,map<Nonterminal, bool>*
finished,Nonterminal last)
```

```
#nonterminals
E T F

#terminals
+ - * / ( ) num

#startSymbol
E

#productions
E -> E + T | E - T | T
T -> T * F | T / F | F
F -> ( E ) | num
```

```
**********************************

FIRST集:
E          : ( num
T          : ( num
F          : ( num
E'         : + - ε
T'         : ε * /

**********************************


**********************************

FOLLOW集:
E          : $ )
T          : + - $ )
F          : * / + - $ )
E'         : $ )
T'         : + - $ )

**********************************
```

4.4.6、构造预测分析表

预测分析表的构造算法已经在课件里给出，此处不再赘述。

在构造预测分析表的同时，我也对文法是不是 LL(1)文法进行了判断，当发现一个单元内需要填入多重表项时，说明该文法不是 LL(1)文法，也就不用再进行 LL(1)语法分析了。

算法4.2 预测分析表的构造方法 书92页

输入：文法G

输出：文法G的预测分析表M

方法：

```
for (文法G的每个产生式A→α) {
    for (每个终结符号a∈FIRST(α))
        把A→α放入M[A,a]中;
    if (ε∈FIRST(α))
        for (任何b∈FOLLOW(A))
            把A→α放入M[A,b]中;
};
for (所有无定义的M[A,a]) 标上错误标志.
```

最后，我添加了同步信号到预测分析表中，以便处理输入串不是该文法的句子的情况。

对课本上指定的文法构造预测分析表（带 SYNCH 同步信号）如下：

	+	-	*	/	(	)	num	$
E					E->TE'	SYNCH	E->TE'	SYNCH
T	SYNCH	SYNCH			T->FT'	SYNCH	T->FT'	SYNCH
F	SYNCH	SYNCH	SYNCH	SYNCH	F->(E)	SYNCH	F->num	SYNCH
E'	E'->+TE'	E'->-TE'				E'->ε		E'->ε
T'	T'->ε	T'->ε	T'->*FT'	T'->/FT'		T'->ε		T'->ε

4.4.7、预测分析程序

预测分析程序使用一个符号栈和一个输入串。分析时，我们根据当前栈顶符号和输入串首字符，查询预测分析表，以确定下一步进行的动作。

- **预测分析控制程序：**

- **根据栈顶符号X和当前输入符号a，决定分析动作。**
有4种可能：

  - **X=a=$，宣告分析成功，停止分析**
  - **X=a≠$，从栈顶弹出X，输入指针前移一个位置**
  - **$X \in V_T$，但X≠a，报告发现错误，调用错误处理程序，以报告错误及进行错误恢复。**
  - **若$X \in V_N$，访问分析表M[X,a]，按照下列判断进行相应操作：**

- **若M[X,a]是产生式$X \to Y_1Y_2...Y_n$**
  先将X从栈顶弹出，然后把产生式的右部符号串按反序（即按Y_n、…、Y_{n-1}、Y_2、Y_1的顺序）——推入栈中；

- **若M[X,a]是产生式$X \to \varepsilon$**
  从栈顶弹出X；

- **若M[X,a]是error**
  调用出错处理程序。

错误处理分为下面几种情况：

第一种是栈顶符号为非终结符，但是分析表里对应的单元为空，这时我们采取跳过剩余输入串中的首个符号的方法来进行处理。

第二种是栈顶符号为非终结符，但是分析表里对应的单元为同步信号 synch，这时我们弹出栈顶符号。

第三种是栈顶符号为终结符，但是与剩余输入串中的首个字符不匹配，这时我们采取弹出栈顶的终结符号来处理。

最终，分析程序可能有三种结果。

第一，分析栈和输入串都只剩一个$相匹配，并且分析过程中没有用到同步信号，表示输入串对应的句子是该LL(1)文法的句子。

第二，分析栈和输入串都只剩一个$相匹配，但是分析过程中用到了同步信号，此时虽然成功完成分析，但是输入串不是该文法的句子。

第三，分析结束时，不满足“分析栈和输入串都只剩一个$相匹配”，此时分析失败。

5、测试

测试共分为3个部分。

第一部分是各个子模块（包括消除左递归，消除公共左因子，构造FIRST集，构造FOLLOW集，构造预测分析表，构造分析过程）的测试，具体内容参见“4.4、算法”部分。

第二部分是对各种文法的测试（由于我实现了对通用文法的处理）。

第三部分是对课本上指定的文法进行各种输入串测试。

5.1、文法测试

文法一：含左递归 （来源于课本习题4.5）

```
#nonterminals
E A B L

#terminals
num id ( )

#startSymbol
E

#productions
E -> A | B
A -> num |id
B -> ( L )
L -> L E | E
```

输入串：(a(b(2))(c))

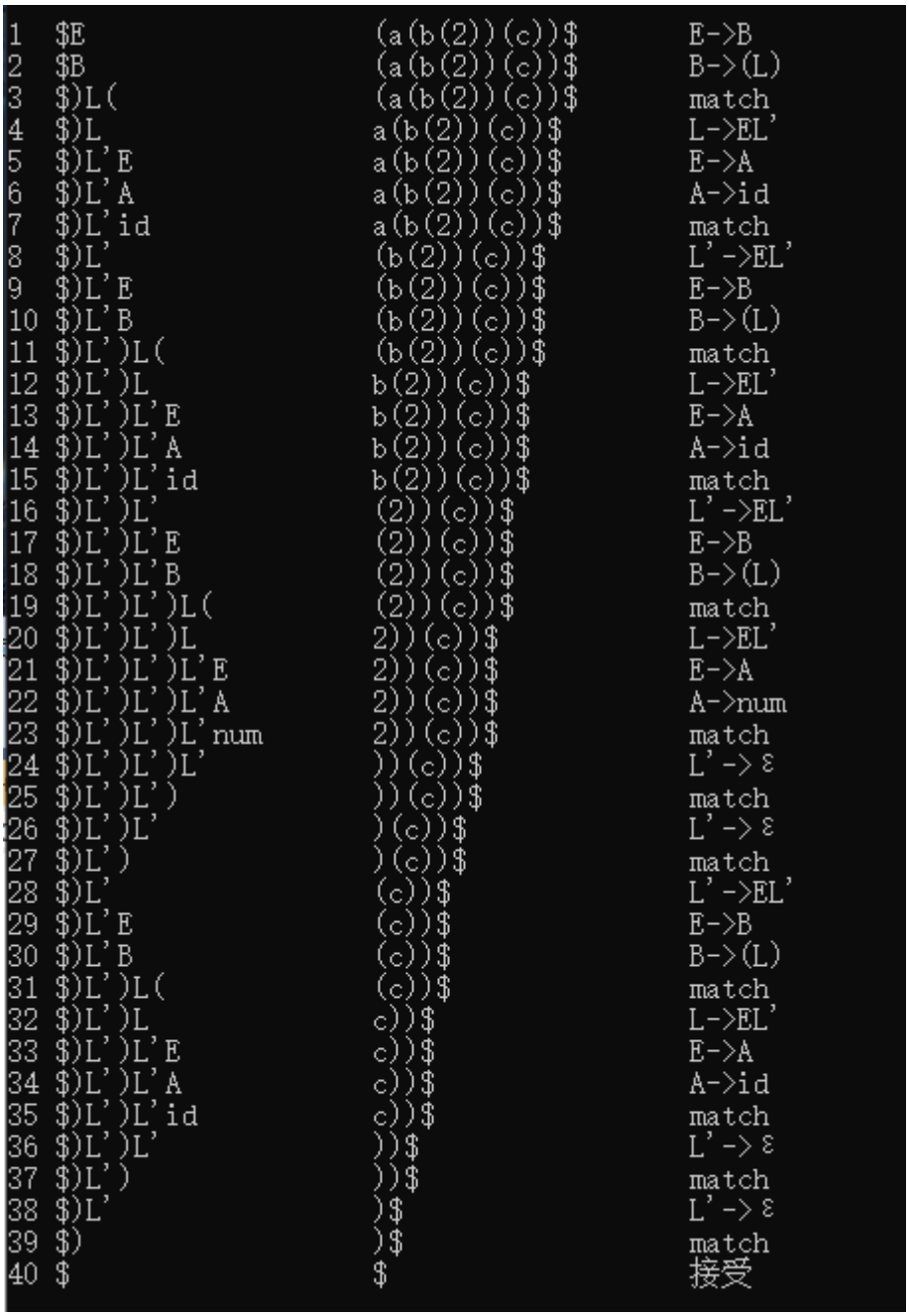
测试结果：（依次为修改后的文法，FIRST 集，FOLLOW 集，预测分析表，分析过程）

```
**********************************

非终结符:
E A B L L'

终结符:
num     id      (       )

起始符:
E

产生式:
E    -> A
E    -> B
A    -> num
A    -> id
L'   -> E L'
L'   -> ε
B    -> ( L )
L    -> E L'

**********************************
```

```
*****************************

FIRST集:
E    : num id (
A    : num id
B    : (
L    : num id (
L'   : num id ( ε

*****************************
```

```
*****************************

FOLLOW集:
E    : num $ id ( )
A    : $ num id ( )
B    : $ num id ( )
L    : )
L'   : )

*****************************
```

```
*************************************************************************************

     num                id                 (                  )                  $
E    E->A               E->A               E->B               SYNCH              SYNCH
A    A->num             A->id              SYNCH              SYNCH              SYNCH
B    SYNCH              SYNCH              B->(L)             SYNCH              SYNCH
L    L->EL'             L->EL'             L->EL'             SYNCH
L'   L'->EL'            L'->EL'            L'->EL'            L'->ε

*************************************************************************************
```



1	$E	(a(b(2))(c))$	E->B
2	$B	(a(b(2))(c))$	B->(L)
3	$)L(	(a(b(2))(c))$	match
4	$)L	a(b(2))(c))$	L->EL'
5	$)L'E	a(b(2))(c))$	E->A
6	$)L'A	a(b(2))(c))$	A->id
7	$)L'id	a(b(2))(c))$	match
8	$)L'	(b(2))(c))$	L'->EL'
9	$)L'E	(b(2))(c))$	E->B
10	$)L'B	(b(2))(c))$	B->(L)
11	$)L')L(	(b(2))(c))$	match
12	$)L')L	b(2))(c))$	L->EL'
13	$)L')L'E	b(2))(c))$	E->A
14	$)L')L'A	b(2))(c))$	A->id
15	$)L')L'id	b(2))(c))$	match
16	$)L')L'	(2))(c))$	L'->EL'
17	$)L')L'E	(2))(c))$	E->B
18	$)L')L'B	(2))(c))$	B->(L)
19	$)L')L')L(	(2))(c))$	match
20	$)L')L')L	2))(c))$	L->EL'
21	$)L')L')L'E	2))(c))$	E->A
22	$)L')L')L'A	2))(c))$	A->num
23	$)L')L')L'num	2))(c))$	match
24	$)L')L')L'	))(c))$	L'->ε
25	$)L')L')	))(c))$	match
26	$)L')L'	)(c))$	L'->ε
27	$)L')	)(c))$	match
28	$)L'	(c))$	L'->EL'
29	$)L'E	(c))$	E->B
30	$)L'B	(c))$	B->(L)
31	$)L')L(	(c))$	match
32	$)L')L	c))$	L->EL'
33	$)L')L'E	c))$	E->A
34	$)L')L'A	c))$	A->id
35	$)L')L'id	c))$	match
36	$)L')L'	))$	L'->ε
37	$)L')	))$	match
38	$)L'	)$	L'->ε
39	$)	)$	match
40	$	$	接受

文法二：含公共左因子（来源于课本习题 4.6）

#nonterminals

E A B L

#terminals

num id ( ) ,

#startSymbol

E

#productions

E -> A | B

A -> num |id

B -> (L)

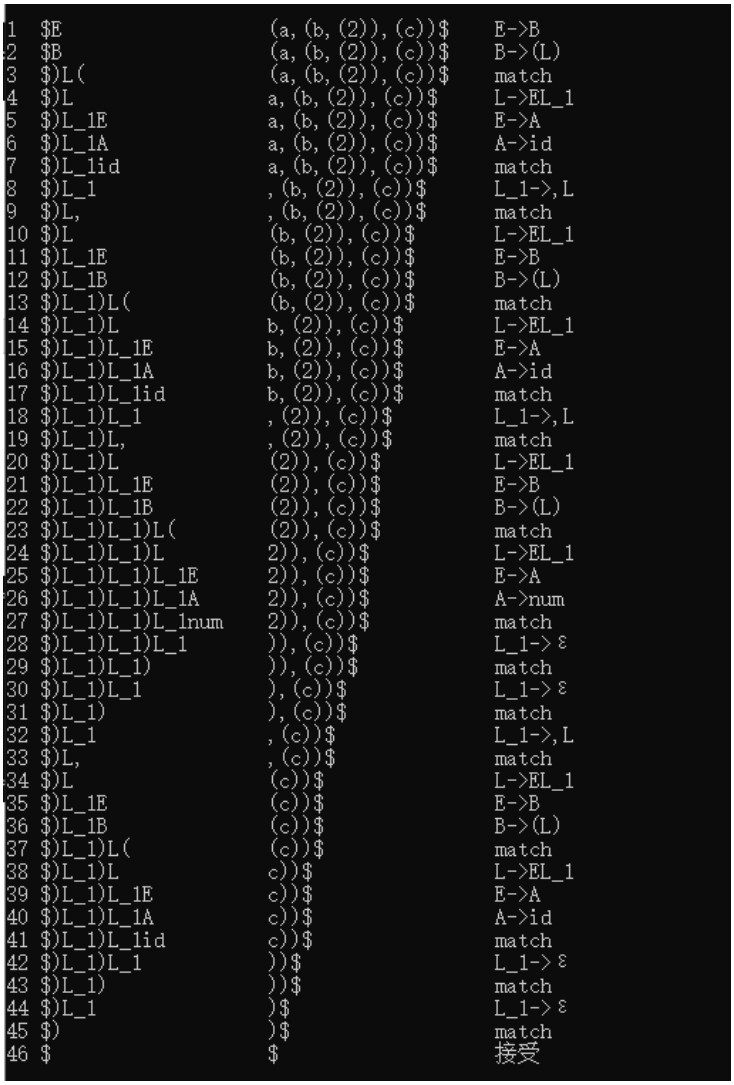
L -> E , L | E

输入串：(a,(b,(2)),(c))

```
**************************************************

非终结符:
E A B L L_1

终结符:
num     id      (       )       ,

起始符:
E

产生式:
E    -> A
E    -> B
A    -> num
A    -> id
B    -> ( L )
L    -> E L_1
L_1  -> , L
L_1  -> ε

**************************************************
```

```
****************************************

FIRST集:
E    : num id (
A    : num id
B    : (
L    : num id (
L_1  : , ε

****************************************


****************************************

FOLLOW集:
E    : , $ )
A    : $ , )
B    : $ , )
L    : )
L_1  : )

****************************************
```

```
********************************************************************************

     num                id                 (                  )                  ,                  $
E    E->A               E->A               E->B               SYNCH              SYNCH              SYNCH
A    A->num             A->id                                 SYNCH              SYNCH              SYNCH
B                                          B->(L)             SYNCH              SYNCH              SYNCH
L    L->EL_1            L->EL_1            L->EL_1            SYNCH
L_1                                                           L_1->ε             L_1->,L

********************************************************************************
```



1	$E	(a,(b,(2)),(c))$	E->B
2	$B	(a,(b,(2)),(c))$	B->(L)
3	$)L(	(a,(b,(2)),(c))$	match
4	$)L	a,(b,(2)),(c))$	L->EL_1
5	$)L_1E	a,(b,(2)),(c))$	E->A
6	$)L_1A	a,(b,(2)),(c))$	A->id
7	$)L_1id	a,(b,(2)),(c))$	match
8	$)L_1	,(b,(2)),(c))$	L_1->,L
9	$)L,	,(b,(2)),(c))$	match
10	$)L	(b,(2)),(c))$	L->EL_1
11	$)L_1E	(b,(2)),(c))$	E->B
12	$)L_1B	(b,(2)),(c))$	B->(L)
13	$)L_1)L(	(b,(2)),(c))$	match
14	$)L_1)L	b,(2)),(c))$	L->EL_1
15	$)L_1)L_1E	b,(2)),(c))$	E->A
16	$)L_1)L_1A	b,(2)),(c))$	A->id
17	$)L_1)L_1id	b,(2)),(c))$	match
18	$)L_1)L_1	,(2)),(c))$	L_1->,L
19	$)L_1)L,	,(2)),(c))$	match
20	$)L_1)L	(2)),(c))$	L->EL_1
21	$)L_1)L_1E	(2)),(c))$	E->B
22	$)L_1)L_1B	(2)),(c))$	B->(L)
23	$)L_1)L_1)L(	(2)),(c))$	match
24	$)L_1)L_1)L	2)),(c))$	L->EL_1
25	$)L_1)L_1)L_1E	2)),(c))$	E->A
26	$)L_1)L_1)L_1A	2)),(c))$	A->num
27	$)L_1)L_1)L_1num	2)),(c))$	match
28	$)L_1)L_1)L_1	)),(c))$	L_1->ε
29	$)L_1)L_1)	)),(c))$	match
30	$)L_1)L_1	),(c))$	L_1->ε
31	$)L_1)	),(c))$	match
32	$)L_1	,(c))$	L_1->,L
33	$)L,	,(c))$	match
34	$)L	(c))$	L->EL_1
35	$)L_1E	(c))$	E->B
36	$)L_1B	(c))$	B->(L)
37	$)L_1)L(	(c))$	match
38	$)L_1)L	c))$	L->EL_1
39	$)L_1)L_1E	c))$	E->A
40	$)L_1)L_1A	c))$	A->id
41	$)L_1)L_1id	c))$	match
42	$)L_1)L_1	))$	L_1->ε
43	$)L_1)	))$	match
44	$)L_1	)$	L_1->ε
45	$)	)$	match
46	$	$	接受

文法 3：含左递归（来源于课本习题 4.4）

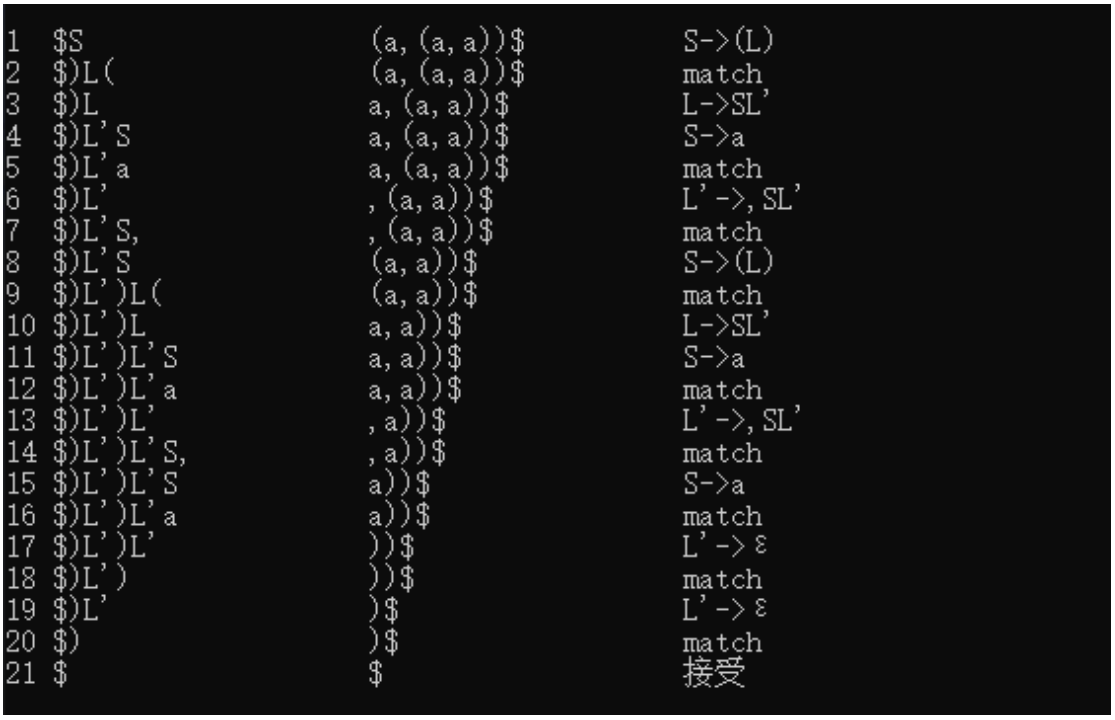
```
#nonterminals
S L

#terminals
( ) a ,

#startSymbol
S

#productions
S -> ( L ) | a
L -> L , S | S
```

输入串：(a,(a,a))

```
**************************************

非终结符:
S L L'

终结符:
(       )       a       ,

起始符:
S

产生式:
S         -> ( L )
S         -> a
L         -> S L'
L'        -> , S L'
L'        -> ε

**************************************
```

```
**************************************

FIRST集:
S         : ( a
L         : ( a
L'        : ,  ε

**************************************



**************************************

FOLLOW集:
S         : , $ )
L         : )
L'        : )

**************************************
```

```
*********************************************************************************

          (                   )                   a                   ,                   $
S         S->(L)              SYNCH               S->a                SYNCH               SYNCH
L         L->SL'              SYNCH               L->SL'
L'                            L'->ε                                   L'->,SL'

*********************************************************************************
```



1	$S	(a,(a,a))$	S->(L)
2	$)L(	(a,(a,a))$	match
3	$)L	a,(a,a))$	L->SL'
4	$)L'S	a,(a,a))$	S->a
5	$)L'a	a,(a,a))$	match
6	$)L'	,(a,a))$	L'->,SL'
7	$)L',S,	,(a,a))$	match
8	$)L'S	(a,a))$	S->(L)
9	$)L')L(	(a,a))$	match
10	$)L')L	a,a))$	L->SL'
11	$)L')L'S	a,a))$	S->a
12	$)L')L'a	a,a))$	match
13	$)L')L'	,a))$	L'->,SL'
14	$)L')L'S,	,a))$	match
15	$)L')L'S	a))$	S->a
16	$)L')L'a	a))$	match
17	$)L')L'	))$	L'->ε
18	$)L')	))$	match
19	$)L'	)$	L'->ε
20	$)	)$	match
21	$	$	接受

文法 4：来源于课件

#nonterminals

E E' T T' F

#terminals

+ * ( ) id

#startSymbol

E

#productions

E -> T E'

E' -> + T E' | epsilon

T -> F T'

T' -> * F T' | epsilon

F -> (E) | id

输入串：*id*+id （该串不是文法的句子，利用了 SYNCH 同步信号进行处理）

测试结果：

```
**************************************************

非终结符:
E E' T T' F

终结符:
+       *       (       )       id

起始符:
E

产生式:
E          -> T E'
E'         -> + T E'
E'         -> ε
T          -> F T'
T'         -> * F T'
T'         -> ε
F          -> ( E )
F          -> id

**************************************************
```

```
**************************************************************************************

FIRST集:
E          : ( id
E'         : + ε
T          : ( id
T'         : ε *
F          : ( id

**************************************************************************************


**************************************************************************************

FOLLOW集:
E          : $ )
E'         : $ )
T          : + $ )
T'         : + $ )
F          : * + $ )

**************************************************************************************
```

```
**************************************************************************

          +               *               (               )               id              $
E                                         E->TE'          SYNCH           E->TE'          SYNCH
E'        E'->+TE'                                        E'->ε                           E'->ε
T         SYNCH                           T->FT'          SYNCH           T->FT'          SYNCH
T'        T'->ε           T'->*FT'                        T'->ε                           T'->ε
F         SYNCH           SYNCH           F->(E)          SYNCH           F->id           SYNCH

**************************************************************************
```

```
1  $E                *id*+id$          error: skip *
2  $E                id*+id$           E->TE'
3  $E'T              id*+id$           T->FT'
4  $E'T'F            id*+id$           F->id
5  $E'T'id           id*+id$           match
6  $E'T'             *+id$             T'->*FT'
7  $E'T'F*           *+id$             match
8  $E'T'F            +id$              synch: pop F
9  $E'T'             +id$              T'->ε
10 $E'               +id$              E'->+TE'
11 $E'T+             +id$              match
12 $E'T              id$               T->FT'
13 $E'T'F            id$               F->id
14 $E'T'id           id$               match
15 $E'T'             $                 T'->ε
16 $E'               $                 E'->ε
17 $                 $                 分析完毕，输入串不是该文法的句子
```

文法 5：该文法不是 LL(1)文法

#nonterminals

S A B C D

#terminals

a b c

#startSymbol

S

#productions

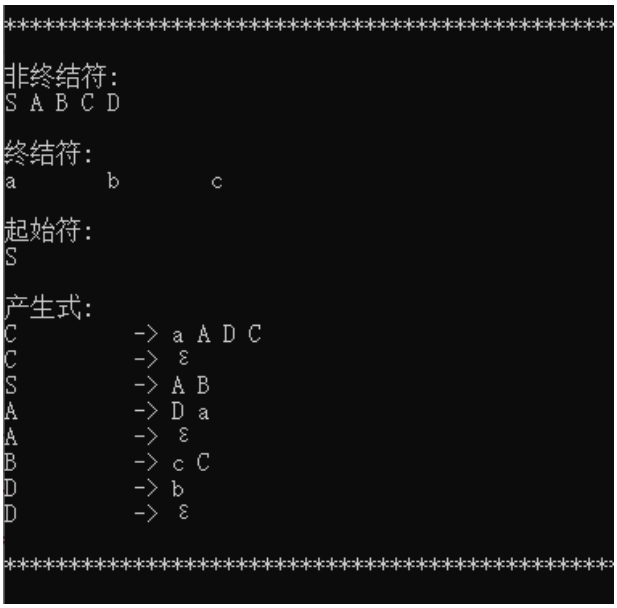
S -> A B

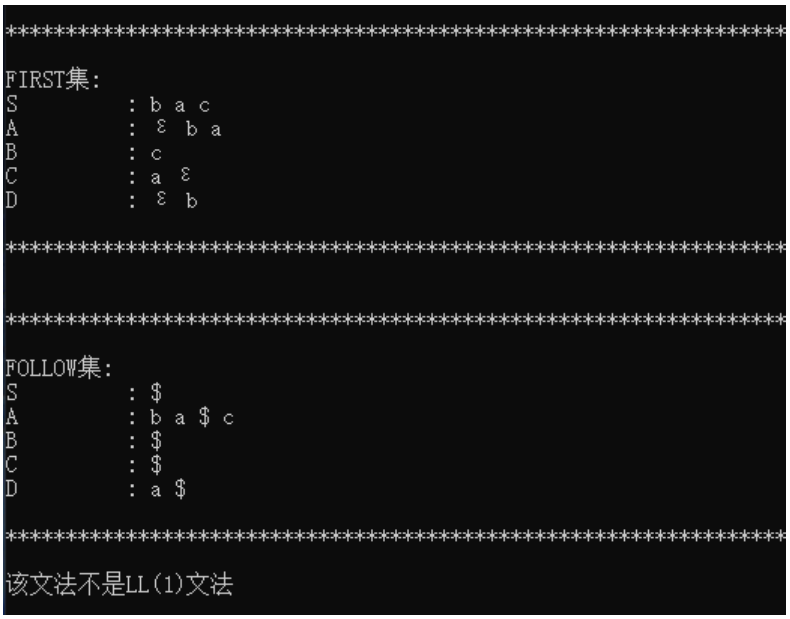
A -> D a | epsilon

B -> c C

C -> a A D C | epsilon

D -> b | epsilon





输出结果显示，该文法不是LL(1)文法

5.2、对课本上的文法进行输入串测试

课本上的文法：

#nonterminals

E T F

#terminals

+ - * / ( ) num

#startSymbol

E

#productions

E -> E + T | E - T | T

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

F -> (E) | num

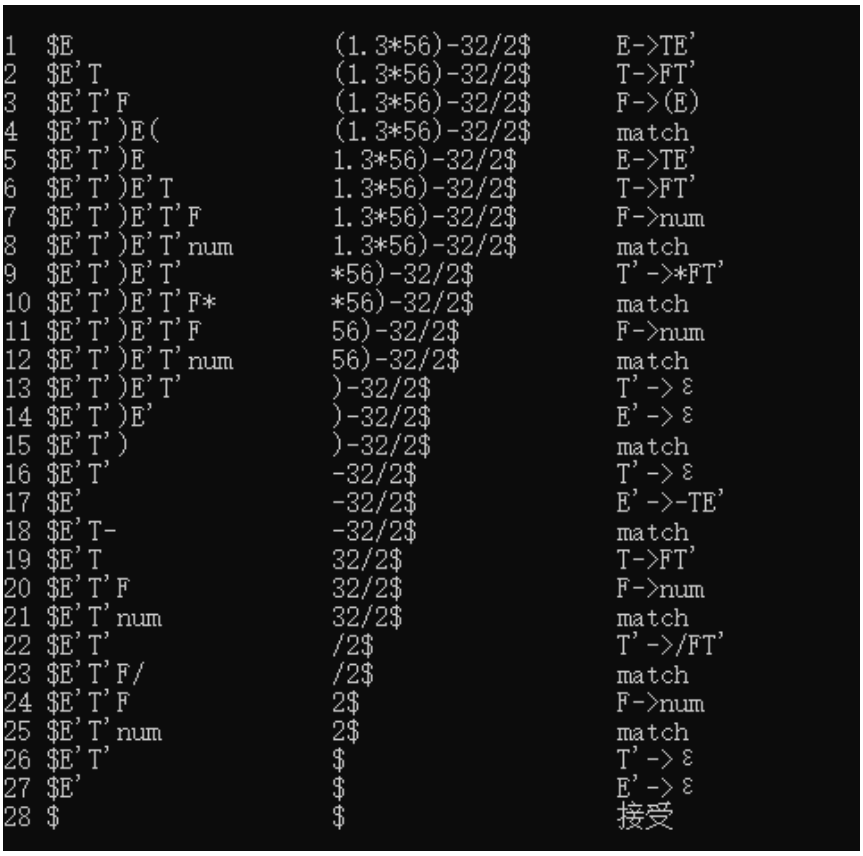
```
**************************************************************

FIRST集:
E          : ( num
T          : ( num
F          : ( num
E'         : + -  ε
T'         :  ε  * /

**************************************************************



**************************************************************

FOLLOW集:
E          : $ )
T          : + - $ )
F          : * / + - $ )
E'         : $ )
T'         : + - $ )

**************************************************************
```

```
*****************************************************************************************

          +         -         *         /         (         )         num       $
E                                                 E->TE'    SYNCH     E->TE'    SYNCH
T         SYNCH     SYNCH                         T->FT'    SYNCH     T->FT'    SYNCH
F         SYNCH     SYNCH     SYNCH     SYNCH     F->(E)    SYNCH     F->num    SYNCH
E'        E'->+TE'  E'->-TE'                                E'->ε               E'->ε
T'        T'->ε     T'->ε     T'->*FT'  T'->/FT'            T'->ε               T'->ε

*****************************************************************************************
```

输入串 1：1+2

```
1  $E                  1+2$                E->TE'
2  $E'T                1+2$                T->FT'
3  $E'T'F              1+2$                F->num
4  $E'T'num            1+2$                match
5  $E'T'               +2$                 T'->ε
6  $E'                 +2$                 E'->+TE'
7  $E'T+               +2$                 match
8  $E'T                2$                  T->FT'
9  $E'T'F              2$                  F->num
10 $E'T'num            2$                  match
11 $E'T'               $                   T'->ε
12 $E'                 $                   E'->ε
13 $                   $                   接受
```

输入串 2：(1.3*56)-32/2



1	$E	(1.3*56)-32/2$	E->TE'
2	$E'T	(1.3*56)-32/2$	T->FT'
3	$E'T'F	(1.3*56)-32/2$	F->(E)
4	$E'T')E(	(1.3*56)-32/2$	match
5	$E'T')E	1.3*56)-32/2$	E->TE'
6	$E'T')E'T	1.3*56)-32/2$	T->FT'
7	$E'T')E'T'F	1.3*56)-32/2$	F->num
8	$E'T')E'T'num	1.3*56)-32/2$	match
9	$E'T')E'T'	*56)-32/2$	T'->*FT'
10	$E'T')E'T'F*	*56)-32/2$	match
11	$E'T')E'T'F	56)-32/2$	F->num
12	$E'T')E'T'num	56)-32/2$	match
13	$E'T')E'T'	)-32/2$	T'->ε
14	$E'T')E'	)-32/2$	E'->ε
15	$E'T')	)-32/2$	match
16	$E'T'	-32/2$	T'->ε
17	$E'	-32/2$	E'->-TE'
18	$E'T-	-32/2$	match
19	$E'T	32/2$	T->FT'
20	$E'T'F	32/2$	F->num
21	$E'T'num	32/2$	match
22	$E'T'	/2$	T'->/FT'
23	$E'T'F/	/2$	match
24	$E'T'F	2$	F->num
25	$E'T'num	2$	match
26	$E'T'	$	T'->ε
27	$E'	$	E'->ε
28	$	$	接受

输入串 3：((3+3)*3+3*(3+3))

1	$E	((3+3)*3+3*(3+3))$	E->TE'
2	$E'T	((3+3)*3+3*(3+3))$	T->FT'
3	$E'T'F	((3+3)*3+3*(3+3))$	F->(E)
4	$E'T')E(	((3+3)*3+3*(3+3))$	match
5	$E'T')E	(3+3)*3+3*(3+3))$	E->TE'
6	$E'T')E'T	(3+3)*3+3*(3+3))$	T->FT'
7	$E'T')E'T'F	(3+3)*3+3*(3+3))$	F->(E)
8	$E'T')E'T')E(	(3+3)*3+3*(3+3))$	match
9	$E'T')E'T')E	3+3)*3+3*(3+3))$	E->TE'
10	$E'T')E'T')E'T	3+3)*3+3*(3+3))$	T->FT'
11	$E'T')E'T')E'T'F	3+3)*3+3*(3+3))$	F->num
12	$E'T')E'T')E'T'num	3+3)*3+3*(3+3))$	match
13	$E'T')E'T')E'T'	+3)*3+3*(3+3))$	T'->ε
14	$E'T')E'T')E'	+3)*3+3*(3+3))$	E'->+TE'
15	$E'T')E'T')E'T+	+3)*3+3*(3+3))$	match
16	$E'T')E'T')E'T	3)*3+3*(3+3))$	T->FT'
17	$E'T')E'T')E'T'F	3)*3+3*(3+3))$	F->num
18	$E'T')E'T')E'T'num	3)*3+3*(3+3))$	match
19	$E'T')E'T')E'T'	)*3+3*(3+3))$	T'->ε
20	$E'T')E'T')E'	)*3+3*(3+3))$	E'->ε
21	$E'T')E'T')	)*3+3*(3+3))$	match
22	$E'T')E'T'	*3+3*(3+3))$	T'->*FT'
23	$E'T')E'T'F*	*3+3*(3+3))$	match
24	$E'T')E'T'F	3+3*(3+3))$	F->num
25	$E'T')E'T'num	3+3*(3+3))$	match
26	$E'T')E'T'	+3*(3+3))$	T'->ε
27	$E'T')E'	+3*(3+3))$	E'->+TE'
28	$E'T')E'T+	+3*(3+3))$	match
29	$E'T')E'T	3*(3+3))$	T->FT'
30	$E'T')E'T'F	3*(3+3))$	F->num
31	$E'T')E'T'num	3*(3+3))$	match
32	$E'T')E'T'	*(3+3))$	T'->*FT'
33	$E'T')E'T'F*	*(3+3))$	match
34	$E'T')E'T'F	(3+3))$	F->(E)
35	$E'T')E'T')E(	(3+3))$	match
36	$E'T')E'T')E	3+3))$	E->TE'
37	$E'T')E'T')E'T	3+3))$	T->FT'
38	$E'T')E'T')E'T'F	3+3))$	F->num
39	$E'T')E'T')E'T'num	3+3))$	match
40	$E'T')E'T')E'T'	+3))$	T'->ε
41	$E'T')E'T')E'	+3))$	E'->+TE'
42	$E'T')E'T')E'T+	+3))$	match
43	$E'T')E'T')E'T	3))$	T->FT'
44	$E'T')E'T')E'T'F	3))$	F->num
45	$E'T')E'T')E'T'num	3))$	match
46	$E'T')E'T')E'T'	))$	T'->ε
47	$E'T')E'T')E'	))$	E'->ε
48	$E'T')E'T')	))$	match
49	$E'T')E'T'	)$	T'->ε
50	$E'T')E'	)$	E'->ε
51	$E'T')	)$	match
52	$E'T'	$	T'->ε
53	$E'	$	E'->ε
54	$	$	接受

输入串 4：3+

```
1  $E                3+$               E->TE'
2  $E'T              3+$               T->FT'
3  $E'T'F            3+$               F->num
4  $E'T'num          3+$               match
5  $E'T'             +$                T'->ε
6  $E'               +$                E'->+TE'
7  $E'T+             +$                match
8  $E'T              $                 synch: pop T
9  $E'               $                 E'->ε
10 $                 $                 分析完毕，输入串不是该文法的句子
```

输入串 5：(3*3(3+3)

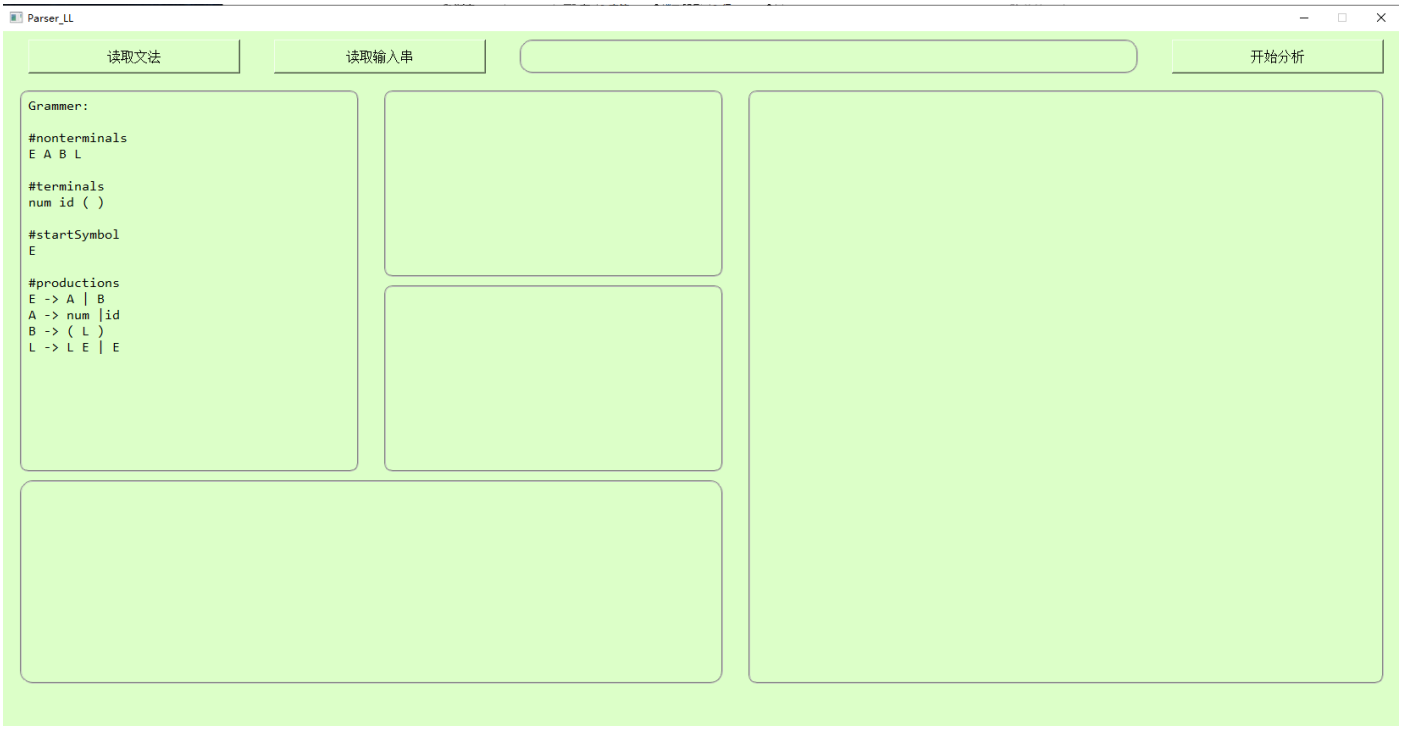
```
1  $E                (3*3(3+3)$        E->TE'
2  $E'T              (3*3(3+3)$        T->FT'
3  $E'T'F            (3*3(3+3)$        F->(E)
4  $E'T')E(          (3*3(3+3)$        match
5  $E'T')E           3*3(3+3)$         E->TE'
6  $E'T')E'T         3*3(3+3)$         T->FT'
7  $E'T')E'T'F       3*3(3+3)$         F->num
8  $E'T')E'T'num     3*3(3+3)$         match
9  $E'T')E'T'        *3(3+3)$          T'->*FT'
10 $E'T')E'T'F*      *3(3+3)$          match
11 $E'T')E'T'F       3(3+3)$           F->num
12 $E'T')E'T'num     3(3+3)$           match
13 $E'T')E'T'        (3+3)$            error: skip (
14 $E'T')E'T'        3+3)$             error: skip 3
15 $E'T')E'T'        +3)$              T'->ε
16 $E'T')E'          +3)$              E'->+TE'
17 $E'T')E'T+        +3)$              match
18 $E'T')E'T         3)$               T->FT'
19 $E'T')E'T'F       3)$               F->num
20 $E'T')E'T'num     3)$               match
21 $E'T')E'T'        )$                T'->ε
22 $E'T')E'          )$                E'->ε
23 $E'T')            )$                match
24 $E'T'             $                 T'->ε
25 $E'               $                 E'->ε
26 $                 $                 分析完毕，输入串不是该文法的句子
```

6、图形界面

基于和之前词法分析器同样的原因，这次语法分析我同样实现了图形界面。图形界面的好处在于，一个大界面可以分成好几个部分，同时展示文法、FIRST集、FOLLOW集、预测分析表、分析过程等信息，便于对比观察。此外，还可以直接选择不同的文件，而不需考虑路径问题。

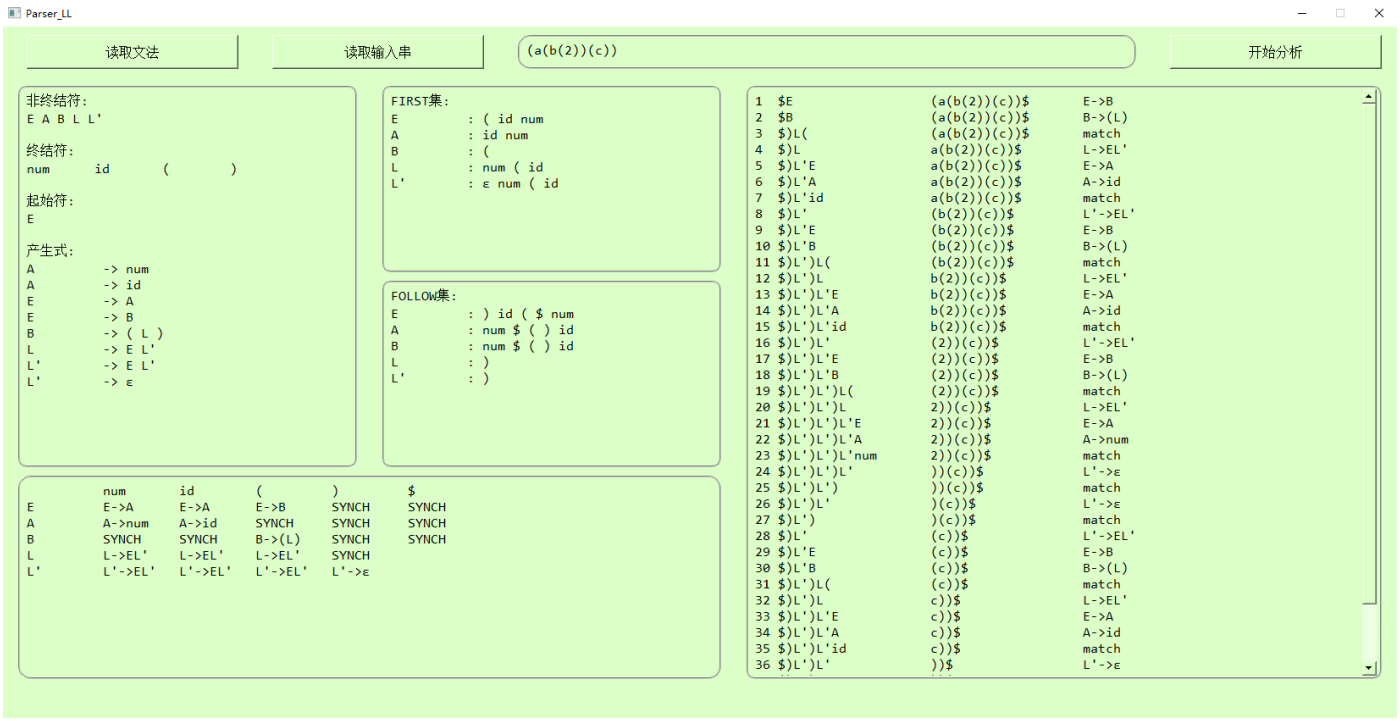
初始界面如下所示

第一步：点击左上角“读取文法”按钮，可以选择一个文法文件，确认后，文法将显示在文法区域。



第二步：点击左上角“读取输入串”按钮，可以选择一个输入串文件，确认后，输入串将显示在对应区域。

第三步：点击右上角“开始分析”按钮，即可进行 LL(1) 分析。



这是另外一个测试样例

7、总结

7.1、程序实现的效果

该 LL(1)语法分析程序实现了题目所要求的全部功能。从文法的读取，等价变换，FIRST 集 FOLLOW 集、预测分析表和预测分析过程全部由程序自动生成。程序不仅能够处理课本给出的文法，也可以处理任意的文法。

7.2、实验遇到的问题

实验中麻烦问题贯穿始终，各种算法基本都需要用到递归，需要对边界条件和各种情况进行充分而细致的考虑。程序卡死崩溃，往往都是源于陷入了无限递归，需要反复进行调试才能解决。好在各个模块划分的比较清晰，互不干扰，可以快速地定位错误。

7.3、设计亮点

亮点 1：给出文法和输入串，所有过程均是由程序自动完成，无需任何手工处理。

亮点 2：程序能够实现超过课本要求的功能，即对任意给定的文法，都能够进行处理。

亮点 3：数据结构、类的定义和接口设计得比较合理，各模块划分清晰。

7.4、设计缺点

程序的缺点主要在于算法的实现上。完成算法后我发现程序各块遍布了嵌套多重循环，这导致算法的时间复杂度达到一个很高的水平。虽然说由于输入的文法规模不会很大（文法本身的性质决定了一个有实际意义的文法必然不会有过于繁多的产生式），时间复杂度高其实也无伤大雅，但是程序的很多地方其实还可以进一步优化。受限于时间和精力，目前只能做到这种程度，将来有机会我会继续学习和改进。

7.5、设计创新点

该程序的创新之处主要在于图形界面的设计，通过图形界面可以将输入和输出进行对比展示，使得分析结果变得更加直观，对用户来说比较友好，也便于设计者发现bug进行修改。

7.6、实验心得

通过本次LL(1)语法分析程序的设计，我已经能熟练掌握LL(1)语法分析中的每一个步骤，大大加深了我对语法分析的理解。通过亲自动手实践，我明白了LL(1)语法分析所需要注意的各种细节。总之，这次实验给我带来了很大的收获。期待接下来的LR语法分析。