

**编译原理与技术**



**题目: 语法分析程序设计与实现**

**姓 名 鄭毓恒**

**学 院 计算机学院**

**专 业 计算机科学与技术**

**班 级 2020211302**

**学 号 2020211262**

**任课教师 李文生**

**2022年 11 月**

**实验内容及要求**

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

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。

**实现方法**

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。

使用C++语言编写程序完成以上方法。

**开发环境**

Windows 11操作系统，Visual Studio 2022

**方法二**

**高层数据信息定义**

|  |  |
| --- | --- |
| 自定义数据类型 |  |
| typedef vector<string> rightSide; | 用于存储同一左部非终结符的多个产生式的右部 |
| typedef map<string, rightSide> generations; | 用于存储文法中所有产生式 |
| typedef map<string, vector<char>> FFTableItem; | 用于存储First表和Follow表 |
| typedef map<string, string> analyTableContent;  typedef map<char, analyTableContent> analyTableItem; | 用于存储预测分析表 |

**类定义**

LL1Grammar类接受无左递归、符合LL(1)语法要求的文法输入，然后给非终结符构造First和Follow集，再构造预测分析表。构造完成后，可进行输入文法的语法分析。

|  |  |
| --- | --- |
| 成员函数 | 函数功能 |
| public | |
| void inputGrammar(void) | 接受用户输入文法，要求文法没有左递归且能构成LL(1)语法，程序不提供消除左递归功能和合法性检测功能 |
| void toLL1(void) | 利用输入的文法，构造First集、Follow集和预测分析表 |
| void analyInput(string input) | 预测分析用户输入input |
| private | |
| void formFirstTable(void) | 初始化First集 |
| void getFirst(string n) | 获取非终结符n的First集 |
| void formFollowTable(void) | 初始化Follow集 |
| void getFollow(void) | 构造所有非终结符的Follow集 |
| void formAnalyTable(void) | 构造预测分析表 |
| int isExistN(string s) | 判断s是否文法的非终结符 |
| int isExistT(char c) | 判断c是否文法的终结符 |
| int isExistTinFirstTable(string left, char t) | 判断非终结符left的First集中是否有t |
| int isExistTinFollowTable(string left, char t) | 判断非终结符left的Follow集中是否有t |

|  |  |
| --- | --- |
| 成员变量 | 功能 |
| private | |
| generations gens | 存储文法的产生式 |
| vector<string> N | 存储非终结符 |
| vector<char> T | 存储终结符 |
| FFTableItem firstTable | 存储First集 |
| FFTableItem followTable | 存储Follow集 |
| analyTableItem analyTable | 存储预测分析表 |

**总体设计**

由于程序不提供消除左递归功能，所以先自己手动消除左递归，得到以下文法：

E -> TE’

E’ -> +TE’

E’ -> -TE’

E’ -> e

T -> FT’

T’ -> \*FT’

T’ -> /FT’

T’ -> e

F -> (E)

F -> n

为了方便处理，将空符号ɛ和终结符num分别以e和n代替。调用成员函数inputGrammar()将转换后的文法输入后，再调用toLL1()函数依次构造First集、Follow集和预测分析表。

在formFirstTable()函数为每个非终结符调用getFirst()函数，在getFirst()函数中获取该非终结符的First集。在该函数中，遍历所有该非终结符为左部的生成式。假如右部第一个符号为终结符且不重复，则可以直接加入。假如第一个符号为非终结符，设为b，则调用getFirst()函数求b的First集，再加入。

在formFollowTable()函数初始化followTable，先向起始符E的Follow集添加$符号，然后调用一次getFollow()函数构造所有Follow集。在getFollow()函数，若A→aBb是产生式，则把FIRST(b)中的所有非空元素加入到FOLLOW(B)中。若A→aB是产生式，或 A→aBb 是产生式并且b可以推导至空，则把FOLLOW(A)中的所有元素加入到FOLLOW(B)中。在一个循环里重复以上两步，直到所有Follow集合不再变化。

调用formAnalyTable()函数构造预测分析表。

按照以下方法编写代码：

for (文法G的每个产生式 A→a ) {

for (First(a)中的每个终结符号 i)

把 A→a 放入 M[A, i] 中;

if (First(a)中有空符号)

for (Follow(A)中的每个终结符号j)

把 A→a 放入 M[A, j] 中;

}；

以上步骤完成后，为分析表加入同步化信息。对所有非终结符A，假如b属于A的Follow集，假如M[A, b]为空，则加入”synch”。使用带有同步化信息的分析表时，若M[A, b]为空，则跳过a；若M[A, b]为synch，则弹出A。

预测分析表构造完成后，接受用户输入，调用analyInput()分析输入。调用该函数前，由于用户可以输入数字而数字在预测分析表中以n表示，所以先将所有数字转换为符号n。

分析开始时，$在栈底，文法开始符号E在栈顶。置ip指向输入的第一个符号；

do { // X是栈顶符号，a是ip所指向的符号

if (X是终结符号或$) {

if (X==a) { 从栈顶弹出X; ip前移一个位置; }

else error();

else if(X是非终结符号)

if (M[X, a]==X→Y1Y2…Yk) {

从栈顶弹出X; 把Yk、Yk-1、…、Y2、Y1压入栈，Y1在栈顶;

输出产生式X→Y1Y2…Yk;

}

else if(M[X, a] == “synch”)

弹出X;

else

跳过a; //ip前移一个位置

} while(X!=$);

按以下步骤进行分析，每一步输出栈和输入缓冲区的状态。

**方法三**

**高层数据信息定义**

|  |  |
| --- | --- |
| 全局常量定义 | 功能 |
| #define SHIFT 0 | 表示action表的类型 |
| #define REDUCE 1 |
| #define ACCEPT 2 |

|  |  |
| --- | --- |
| 自定义数据结构 | 功能 |
| typedef struct  {  char left;  string right;  }generation; | 存储单个产生式 |
| typedef struct  {  int type;  int no;  }actionType; | 用于存储action表 |
| typedef map<char, actionType> actionContent; |
| typedef vector<actionContent> actionTable; |
| typedef map<char, int> gotoContent; | 用于存储goto表 |
| typedef vector<gotoContent> gotoTable; |

类定义

LRanalyTable类接受用户输入非终结符集、终结符集、产生式集合、action表和goto表然后存储，便可接受用户输入符号串进行语法分析。

|  |  |
| --- | --- |
| 成员函数 | 功能 |
| public | |
| void inputTable(); | 接受用户输入非终结符集、终结符集、产生式集合、action表和goto表 |
| void analysis(string input); | 对输入input进行LR语法分析 |
| private | |
| void output(int step, vector<int> stateStack, vector<char> symbolStack, string& input, string::iterator& ip); | 输出每一步时的状态栈、符号栈和输入的状态 |
| int isN(char n); | 判断n是否非终结符 |
| int isT(char t); | 判断t是否终结符 |

|  |  |
| --- | --- |
| 成员变量 | 功能 |
| private | |
| vector<char> N; | 非终结符 |
| vector<char> T; | 终结符 |
| vector<generation> genForms; | 产生式集合 |
| actionTable actionT; | action表 |
| gotoTable gotoT; | goto表 |

总体设计

首先，为给定文法构造识别所有活前缀的DFA。该DFA的有效项目集如下。

|  |  |
| --- | --- |
| 序号 |  |
| 0 | E’ -> •E  E -> •E+T  E -> •E-T  E -> •T  T -> •T\*F  T -> •T/F  T -> •F  F -> •(E)  F -> •n |
| 1 | F -> (•E)  E -> •E+T  E -> •E-T  E -> •T  T -> •T\*F  T -> •T/F  T -> •F  F -> •(E)  F -> •n |
| 2 | F -> n• |
| 3 | E’ -> E•  E -> E•+T  E -> E•-T |
| 4 | E -> T•  T -> T•\*F  T -> T•/F |
| 5 | T -> F• |
| 6 | F -> (E•)  E -> E•+T  E -> E•-T |
| 7 | F -> (E)• |
| 8 | E -> E+•T  T -> •T\*F  T -> •T/F  T -> •F  F -> •(E)  F -> •n |
| 9 | E -> E-•T  T -> •T\*F  T -> •T/F  T -> •F  F -> •(E)  F -> •n |
| 10 | T -> T\*•F  F -> •(E)  F -> •n |
| 11 | T -> T/•F  F -> •(E)  F -> •n |
| 12 | E -> E+T•  T -> T•\*F  T -> T•/F |
| 13 | E -> E-T•  T -> T•\*F  T -> T•/F |
| 14 | T -> T\*F• |
| 15 | T -> T/F• |

DFA的转换函数表为：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | - | \* | / | （ | ） | n | E | T | F |
| 0 |  |  |  |  | 1 |  | 2 | 3 | 4 | 5 |
| 1 |  |  |  |  | 1 |  | 2 | 6 | 4 | 5 |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 | 8 | 9 |  |  |  |  |  |  |  |  |
| 4 |  |  | 10 | 11 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 6 | 8 | 9 |  |  |  | 7 |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  | 1 |  | 2 |  | 12 | 5 |
| 9 |  |  |  |  | 1 |  | 2 |  | 13 | 5 |
| 10 |  |  |  |  | 1 |  | 2 |  |  | 14 |
| 11 |  |  |  |  | 1 |  | 2 |  |  | 15 |
| 12 |  |  | 10 | 11 |  |  |  |  |  |  |
| 13 |  |  | 10 | 11 |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |

为产生式分配编号。

|  |  |
| --- | --- |
| 序号 | 产生式 |
| 0 | E’ -> 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 -> n |

构造该文法的LR分析表。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | + | - | \* | / | （ | ） | n | $ | E | T | F |
| 0 |  |  |  |  | S1 |  | S2 |  | 3 | 4 | 5 |
| 1 |  |  |  |  | S1 |  | S2 |  | 6 | 4 | 5 |
| 2 | R8 | R8 | R8 | R8 |  | R8 |  | R8 |  |  |  |
| 3 | S8 | S9 |  |  |  |  |  | ACC |  |  |  |
| 4 | R3 | R3 | S10 | S11 |  | R3 |  | R3 |  |  |  |
| 5 | R6 | R6 | R6 | R6 |  | R6 |  | R6 |  |  |  |
| 6 | S8 | S9 |  |  |  | S7 |  |  |  |  |  |
| 7 | R7 | R7 | R7 | R7 |  | R7 |  | R7 |  |  |  |
| 8 |  |  |  |  | S1 |  | S2 |  |  | 12 | 5 |
| 9 |  |  |  |  | S1 |  | S2 |  |  | 13 | 5 |
| 10 |  |  |  |  | S1 |  | S2 |  |  |  | 14 |
| 11 |  |  |  |  | S1 |  | S2 |  |  |  | 15 |
| 12 | R1 | R1 | S10 | S11 |  | R1 |  | R1 |  |  |  |
| 13 | R2 | R2 | S10 | S11 |  | R2 |  | R2 |  |  |  |
| 14 | R4 | R4 | R4 | R4 |  | R4 |  | R4 |  |  |  |
| 15 | R5 | R5 | R5 | R5 |  | R5 |  | R5 |  |  |  |

在输入以上内容后，可以输入符号串进行LR语法分析。按以下步骤进行分析：

开始时，初始状态 S0 在栈顶，置 ip 指向输入串的第一个符号；

do { // 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; 推进ip，使它指向下一个符号;

};

else if (action[S, a]==reduce by A→b) {

从栈顶弹出 |b| 个符号; // 令S’是现在的栈顶状态

把 A 和 goto[S’, A] 分别压入符号栈和状态栈;

};

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

else error();

} while(1)

并且，在每一步都会输出状态栈、符号栈、输入和输出的状态

**测试结果和分析**

**方法二**

|  |
| --- |
| 文法输入 |
| 5  E  E'  T  T'  F  8  +  -  \*  /  e  (  )  n  10  E  TE'  E'  +TE'  E'  -TE'  E'  e  T  FT'  T'  \*FT'  T'  /FT'  T'  e  F  (E)  F  n |

运行程序后，输入以上内容将文法输入，然后进行测试

|  |  |
| --- | --- |
|  | 测试样例 |
| 1 | ((99\*88-77)/(66+55-44)) |
| 2 | -1 |
| 3 | abc100 |

测试以上样例。

|  |  |
| --- | --- |
|  | 运行结果 |
| 1 | Step: 1  Stack: $E  Input: ((n\*n-n)/(n+n-n))$  Output: E -> TE'  Step: 2  Stack: $E'T  Input: ((n\*n-n)/(n+n-n))$  Output: T -> FT'  Step: 3  Stack: $E'T'F  Input: ((n\*n-n)/(n+n-n))$  Output: F -> (E)  Step: 4  Stack: $E'T')E(  Input: ((n\*n-n)/(n+n-n))$  Step: 5  Stack: $E'T')E  Input: (n\*n-n)/(n+n-n))$  Output: E -> TE'  Step: 6  Stack: $E'T')E'T  Input: (n\*n-n)/(n+n-n))$  Output: T -> FT'  Step: 7  Stack: $E'T')E'T'F  Input: (n\*n-n)/(n+n-n))$  Output: F -> (E)  Step: 8  Stack: $E'T')E'T')E(  Input: (n\*n-n)/(n+n-n))$  Step: 9  Stack: $E'T')E'T')E  Input: n\*n-n)/(n+n-n))$  Output: E -> TE'  Step: 10  Stack: $E'T')E'T')E'T  Input: n\*n-n)/(n+n-n))$  Output: T -> FT'  Step: 11  Stack: $E'T')E'T')E'T'F  Input: n\*n-n)/(n+n-n))$  Output: F -> n  Step: 12  Stack: $E'T')E'T')E'T'n  Input: n\*n-n)/(n+n-n))$  Step: 13  Stack: $E'T')E'T')E'T'  Input: \*n-n)/(n+n-n))$  Output: T' -> \*FT'  Step: 14  Stack: $E'T')E'T')E'T'F\*  Input: \*n-n)/(n+n-n))$  Step: 15  Stack: $E'T')E'T')E'T'F  Input: n-n)/(n+n-n))$  Output: F -> n  Step: 16  Stack: $E'T')E'T')E'T'n  Input: n-n)/(n+n-n))$  Step: 17  Stack: $E'T')E'T')E'T'  Input: -n)/(n+n-n))$  Output: T' -> e  Step: 18  Stack: $E'T')E'T')E'  Input: -n)/(n+n-n))$  Output: E' -> -TE'  Step: 19  Stack: $E'T')E'T')E'T-  Input: -n)/(n+n-n))$  Step: 20  Stack: $E'T')E'T')E'T  Input: n)/(n+n-n))$  Output: T -> FT'  Step: 21  Stack: $E'T')E'T')E'T'F  Input: n)/(n+n-n))$  Output: F -> n  Step: 22  Stack: $E'T')E'T')E'T'n  Input: n)/(n+n-n))$  Step: 23  Stack: $E'T')E'T')E'T'  Input: )/(n+n-n))$  Output: T' -> e  Step: 24  Stack: $E'T')E'T')E'  Input: )/(n+n-n))$  Output: E' -> e  Step: 25  Stack: $E'T')E'T')  Input: )/(n+n-n))$  Step: 26  Stack: $E'T')E'T'  Input: /(n+n-n))$  Output: T' -> /FT'  Step: 27  Stack: $E'T')E'T'F/  Input: /(n+n-n))$  Step: 28  Stack: $E'T')E'T'F  Input: (n+n-n))$  Output: F -> (E)  Step: 29  Stack: $E'T')E'T')E(  Input: (n+n-n))$  Step: 30  Stack: $E'T')E'T')E  Input: n+n-n))$  Output: E -> TE'  Step: 31  Stack: $E'T')E'T')E'T  Input: n+n-n))$  Output: T -> FT'  Step: 32  Stack: $E'T')E'T')E'T'F  Input: n+n-n))$  Output: F -> n  Step: 33  Stack: $E'T')E'T')E'T'n  Input: n+n-n))$  Step: 34  Stack: $E'T')E'T')E'T'  Input: +n-n))$  Output: T' -> e  Step: 35  Stack: $E'T')E'T')E'  Input: +n-n))$  Output: E' -> +TE'  Step: 36  Stack: $E'T')E'T')E'T+  Input: +n-n))$  Step: 37  Stack: $E'T')E'T')E'T  Input: n-n))$  Output: T -> FT'  Step: 38  Stack: $E'T')E'T')E'T'F  Input: n-n))$  Output: F -> n  Step: 39  Stack: $E'T')E'T')E'T'n  Input: n-n))$  Step: 40  Stack: $E'T')E'T')E'T'  Input: -n))$  Output: T' -> e  Step: 41  Stack: $E'T')E'T')E'  Input: -n))$  Output: E' -> -TE'  Step: 42  Stack: $E'T')E'T')E'T-  Input: -n))$  Step: 43  Stack: $E'T')E'T')E'T  Input: n))$  Output: T -> FT'  Step: 44  Stack: $E'T')E'T')E'T'F  Input: n))$  Output: F -> n  Step: 45  Stack: $E'T')E'T')E'T'n  Input: n))$  Step: 46  Stack: $E'T')E'T')E'T'  Input: ))$  Output: T' -> e  Step: 47  Stack: $E'T')E'T')E'  Input: ))$  Output: E' -> e  Step: 48  Stack: $E'T')E'T')  Input: ))$  Step: 49  Stack: $E'T')E'T'  Input: )$  Output: T' -> e  Step: 50  Stack: $E'T')E'  Input: )$  Output: E' -> e  Step: 51  Stack: $E'T')  Input: )$  Step: 52  Stack: $E'T'  Input: $  Output: T' -> e  Step: 53  Stack: $E'  Input: $  Output: E' -> e  Step: 54  Stack: $  Input: $  Output: Accept |
| 2 |  |
| 3 |  |

测试样例1为文法合法的符号串，可见结果输出了语法分析每一步的详细内容，结果正确。测试样例2和3均不合法，可见结果尝试利用同步信息处理错误以及最后输出文法不接受符号串，结果正确。

**方法三**

|  |
| --- |
| 文法输入 |
| 4  S  E  T  F  7  +  -  \*  /  (  )  n  9  S  E  E  E-T  E  E+T  E  T  T  T\*F  T  T/F  T  F  F  (E)  F  n  16  3  3  3  3  0  1  3  0  2  3  3  3  3  3  0  1  3  0  2  3  1  8  1  8  1  8  1  8  3  1  8  3  1  8  0  8  0  9  3  3  3  3  3  2  1  3  1  3  0  10  0  11  3  1  3  3  1  3  1  6  1  6  1  6  1  6  3  1  6  3  1  6  0  8  0  9  3  3  3  0  7  3  3  1  7  1  7  1  7  1  7  3  1  7  3  1  7  3  3  3  3  0  1  3  0  2  3  3  3  3  3  0  1  3  0  2  3  3  3  3  3  0  1  3  0  2  3  3  3  3  3  0  1  3  0  2  3  1  1  1  1  0  10  0  11  3  1  1  3  1  1  1  2  1  2  0  10  0  11  3  1  2  3  1  2  1  4  1  4  1  4  1  4  3  1  4  3  1  4  1  5  1  5  1  5  1  5  3  1  5  3  1  5  3  4  5  6  4  5  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  12  5  -1  13  5  -1  -1  14  -1  -1  15  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1 |

运行程序后，输入以上内容将文法和分析表输入。

|  |  |
| --- | --- |
|  | 测试样例 |
| 1 | ((99\*88-77)/(66+55-44)) |
| 2 | -1 |
| 3 | abc100 |

测试以上样例。

|  |  |
| --- | --- |
|  | 运行结果 |
| 1 | Step: 1  State Stack: 0  Symbol Stack:  Input: ((n\*n-n)/(n+n-n))$  Output: Shift 1  Step: 2  State Stack: 01  Symbol Stack: (  Input: (n\*n-n)/(n+n-n))$  Output: Shift 1  Step: 3  State Stack: 011  Symbol Stack: ((  Input: n\*n-n)/(n+n-n))$  Output: Shift 2  Step: 4  State Stack: 0112  Symbol Stack: ((n  Input: \*n-n)/(n+n-n))$  Output: Reduce F->n  Step: 5  State Stack: 0115  Symbol Stack: ((F  Input: \*n-n)/(n+n-n))$  Output: Reduce T->F  Step: 6  State Stack: 0114  Symbol Stack: ((T  Input: \*n-n)/(n+n-n))$  Output: Shift 10  Step: 7  State Stack: 011410  Symbol Stack: ((T\*  Input: n-n)/(n+n-n))$  Output: Shift 2  Step: 8  State Stack: 0114102  Symbol Stack: ((T\*n  Input: -n)/(n+n-n))$  Output: Reduce F->n  Step: 9  State Stack: 01141014  Symbol Stack: ((T\*F  Input: -n)/(n+n-n))$  Output: Reduce T->T\*F  Step: 10  State Stack: 0114  Symbol Stack: ((T  Input: -n)/(n+n-n))$  Output: Reduce E->T  Step: 11  State Stack: 0116  Symbol Stack: ((E  Input: -n)/(n+n-n))$  Output: Shift 9  Step: 12  State Stack: 01169  Symbol Stack: ((E-  Input: n)/(n+n-n))$  Output: Shift 2  Step: 13  State Stack: 011692  Symbol Stack: ((E-n  Input: )/(n+n-n))$  Output: Reduce F->n  Step: 14  State Stack: 011695  Symbol Stack: ((E-F  Input: )/(n+n-n))$  Output: Reduce T->F  Step: 15  State Stack: 0116913  Symbol Stack: ((E-T  Input: )/(n+n-n))$  Output: Reduce E->E+T  Step: 16  State Stack: 0116  Symbol Stack: ((E  Input: )/(n+n-n))$  Output: Shift 7  Step: 17  State Stack: 01167  Symbol Stack: ((E)  Input: /(n+n-n))$  Output: Reduce F->(E)  Step: 18  State Stack: 015  Symbol Stack: (F  Input: /(n+n-n))$  Output: Reduce T->F  Step: 19  State Stack: 014  Symbol Stack: (T  Input: /(n+n-n))$  Output: Shift 11  Step: 20  State Stack: 01411  Symbol Stack: (T/  Input: (n+n-n))$  Output: Shift 1  Step: 21  State Stack: 014111  Symbol Stack: (T/(  Input: n+n-n))$  Output: Shift 2  Step: 22  State Stack: 0141112  Symbol Stack: (T/(n  Input: +n-n))$  Output: Reduce F->n  Step: 23  State Stack: 0141115  Symbol Stack: (T/(F  Input: +n-n))$  Output: Reduce T->F  Step: 24  State Stack: 0141114  Symbol Stack: (T/(T  Input: +n-n))$  Output: Reduce E->T  Step: 25  State Stack: 0141116  Symbol Stack: (T/(E  Input: +n-n))$  Output: Shift 8  Step: 26  State Stack: 01411168  Symbol Stack: (T/(E+  Input: n-n))$  Output: Shift 2  Step: 27  State Stack: 014111682  Symbol Stack: (T/(E+n  Input: -n))$  Output: Reduce F->n  Step: 28  State Stack: 014111685  Symbol Stack: (T/(E+F  Input: -n))$  Output: Reduce T->F  Step: 29  State Stack: 0141116812  Symbol Stack: (T/(E+T  Input: -n))$  Output: Reduce E->E-T  Step: 30  State Stack: 0141116  Symbol Stack: (T/(E  Input: -n))$  Output: Shift 9  Step: 31  State Stack: 01411169  Symbol Stack: (T/(E-  Input: n))$  Output: Shift 2  Step: 32  State Stack: 014111692  Symbol Stack: (T/(E-n  Input: ))$  Output: Reduce F->n  Step: 33  State Stack: 014111695  Symbol Stack: (T/(E-F  Input: ))$  Output: Reduce T->F  Step: 34  State Stack: 0141116913  Symbol Stack: (T/(E-T  Input: ))$  Output: Reduce E->E+T  Step: 35  State Stack: 0141116  Symbol Stack: (T/(E  Input: ))$  Output: Shift 7  Step: 36  State Stack: 01411167  Symbol Stack: (T/(E)  Input: )$  Output: Reduce F->(E)  Step: 37  State Stack: 0141115  Symbol Stack: (T/F  Input: )$  Output: Reduce T->T/F  Step: 38  State Stack: 014  Symbol Stack: (T  Input: )$  Output: Reduce E->T  Step: 39  State Stack: 016  Symbol Stack: (E  Input: )$  Output: Shift 7  Step: 40  State Stack: 0167  Symbol Stack: (E)  Input: $  Output: Reduce F->(E)  Step: 41  State Stack: 05  Symbol Stack: F  Input: $  Output: Reduce T->F  Step: 42  State Stack: 04  Symbol Stack: T  Input: $  Output: Reduce E->T  Step: 43  State Stack: 03  Symbol Stack: E  Input: $  Output: Accept |
| 2 |  |
| 3 |  |

测试样例1对文法是合法的符号串，因此可见程序输入了LR语法分析详细的每一步过程。测试样例2和3均不合法，可见程序输出了不接受，可以识别错误的符号串。程序运行结果正确。

**心得体会**

本实验只需实现语法分析，而不是完整且复杂的编译程序，使我对设计编译程序有了基础的认识。虽然我在过程中简化了过程，比如，程序将所有非终结符默认为单个char类型数据，例如将num符号用n存储，但是在词法分析程序预先处理各个单词记号，识别并去除空格、注释，语法分析才会处理源代码，因此可以使用简化的记号。

实验源程序仍有许多不足，例如无法检查输入的合法性。例如，假如输入含有左递归，不是LL(1)文法，程序无法检测。同时，输入的方式不够便利，使用了抽象的方式进行输入，并且没有输入检测，容易发生错误。

源代码和可执行文件见附件。