# 一，词法分析：

letter[A-Za-z] ：字母

digit[0-9] ：数字

id({letter}|\_)({letter}|{digit}|\_)\* ：标识符

error\_id({digit})+({letter})+ ：错误的单词

num{digit}+(\.{digit}+)?(E[+\-]?{digit}+)? ：数字

whitespace[ ] ：空格

space [\t] ：制表符

enter[\n] :回车

**功能：**

1. **可以识别出用C语言编写的源程序中的每个单词符号，**
2. **可以识别并读取源程序中的注释。**
3. **可以统计源程序总的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，并输出统计结果。**
4. **检查源程序中存在的错误，并可以报告错误所在的位置。**
5. **发现源程序中存在的错误后，进行适当的恢复，使词法分析可以继续进行，通过一次词法分析处理，可以检查并报告源程序中那个存在的所有错误。**

# 二，语法分析：

拓广文法：

**L->E\n**

**E->E+T**

**E->E-T**

**E->T**

**T->T\*F**

**T->T/F**

**T->F**

**F->id**

**F->(E)**

**F->num**

**正则表达式：**

id [A-Za-z]

digit[0-9]

num{digit}+(\.{digit}+)?(E[+\-]?{digit}+)?

whitespace[ \t]+

enter[\n]

变量名称及结构

%token NUM :词法分析返回记号，整数 终结符

%token ID:词法分析返回记号，标识符 终结符

lines：文法起始符，L—>E\n 终结符

expr: E 非终结符

term: T 非终结符

factor: F 非终结符

# 三，语义分析：

**E—>E+T|E-T|T**

**T—>T\*F|T/F|F**

**F—>num.num|(E)|num**

在对输入表达式进行分析的过程中，输出所采用的产生式

对表达式进行类型检查和类型转换

**拓广文法：**

**L->E\n**

**E->E+T**

**E->E-T**

**E->T**

**T->T\*F**

**T->T/F**

**T->F**

**F->real(即num.num)**

**F->(E)**

**F->num**

**正则表达式：**

digit[0-9]

real{digit}+[.]{digit}+

num{digit}+

whitespace[ ]+

enter[\n]

**数据结构：**

typedef struct x

{ int i;

double f;

int c;

}type;//$$的类型，i为整型值，f为实型值，c为类型（0为整型，2为实型）

%union{

int num;

double realnum;

type c;

}//yylval的类型，联合类型

%token <num> NUM

%token <realnum> REAL

%type<c>expr term factor

Line:L 非终结符 起始符

expr: E 非终结符

term: T 非终结符

factor: F 非终结符