任务：

使用flex+bison构造简易四则运算器。

文法：

根据给出的文法构建四则运算器：

下面是本次实验的参考文法，根据此文法生成功能完整的四则运算器。

1. Tokens

ADD → +

SUB → -

MUL → \*

DIV → /

LP → (

RP → )

EOL → \n

2. Expressions

Calclist → Calclist Exp EOL

| ɛ

Exp → Exp ADD Exp

| Exp SUB Exp

| Exp MUL Exp

| Exp DIV Exp

| LP Exp RP

| interger-constant

| floating-constant

3. Integer constants

interger-constant → decimal-constant

| octal-constant

| hexadecimal-constant

decimal-constant→ nonzero-digit

| decimal-constant digit

octal-constant → 0

| octal-constant octal-digit

hexadecimal-constant → hexadecimal-prefix hexadecimal-digit

| hexadecimal-constant hexadecimal-digit

hexadecimal-prefix → one of

0x

0X

nonzero-digit → one of

1 2 3 4 5 6 7 8 9

octal-digit → one of

0 1 2 3 4 5 6 7

hexadecimal-digit → one of

0 1 2 3 4 5 6 7 8 9

a b c d e f

A B C D E F

digit → one of

0 1 2 3 4 5 6 7 8 9

4. Floating constants

floating-constant → decimal-floating-constant

| hexadecimal-floating-constant

decimal-floating-constant → fractional-constant exponent-partopt

| digit-sequence exponent-part

hexadecimal-floating-constant →

hexadecimal-prefix hexadecimal-fractional-constant binary-exponent-part

| hexadecimal-prefix hexadecimal-digit-sequence binary-exponent-part

fractional-constant → digit-sequenceopt . digit-sequence

| digit-sequence .

exponent-part → e signopt digit-sequence

| E signopt digit-sequence

sign → one of

+ -

digit-sequence → digit

| digit-sequence digit

hexadecimal-fractional-constant →

hexadecimal-digit-sequenceopt . hexadecimal-digit-sequence

| hexadecimal-digit-sequence .

binary-exponent-part → p signopt digit-sequence

| P signopt digit-sequence

hexadecimal-digit-sequence →

hexadecimal-digit

| hexadecimal-digit-sequence hexadecimal-digit

附加说明

(1)Tokens: 这一部分主要与词法分析相关，定义了不同类型的终结符。其中

EOL 表示换行符。当你的程序在进行词法分析时识别到了无法被分配到任何

token 的字符时，应当出现词法错误。

(2)Expressions: 这一部分定义了关于表达式的非终结符的产生式。Calclist 表

示由若干行表达式组成的序列，同时也可以表示整个程序。在没有出现语法错

误的情况下，Calclist 会不断读取语法正确的表达式。在本次实验中，在初次归

约出 Calclist 时，需要终止语法分析。ɛ 表示空串。Exp 表示四则运算表达式。

(3)优先级：为了消除潜在的二义性问题，括号在所有运算符中优先级最高，

乘除运算符的优先级比加减运算符更高。所有运算符的结合性均为左结合。

(4)Interger constants: 这一部分定义了整型常量。一个整型常量从一个数位开

始，不包含句点和指数部分。它可能会包含一个表示其进制的一个前缀。例如

一个八进制常量会包含一个前缀 0 和一组 0~7 的数字序列。

(5)Floating constants: 这一部分定义了浮点型常量。一个浮点型常量包含一个

有效数字部分和一个可选的指数部分。

【输入格式】：

每个测试样例包含一个表达式，表达式包含常量、加减乘除运算符以及括号，且由换行符结尾。其中常量包括整型常量和浮点型常量，关于常量的合法性以及对应的文法，请参考附件中的文法说明。

【输出格式】：

(1) 对于包含语法错误的表达式，只需要输出语法有误的信息即可（首字母大写，末尾有英文句号），错误信息如下：

Syntactical Error.

(2) 对于没有任何语法错误的表达式，输出等号('=')以及表达式的计算结果（结果无论是整数还是小数，均保留小数点后三位），详见下方的样例输出。

(3) 所有样例输出均以单个换行符结尾。

【样例1输入】1+1

【样例1输出】=2.000

【样例2输入】1++1

【样例2输出】Syntactical Error.

【样例3输入】(1)

【样例3输出】=1.000

【样例4输入】(1+2)\*(3+(4-(2\*5)/(6-1)+2))

【样例4输出】=21.000

【样例5输入】1e-2

【样例5输出】=0.010

在源码的基础上进行改进，源码如下：

parser.y：

%{

#include<stdio.h>

void yyerror(char \*msg);

int yylex();

%}

%union{

int INT\_;

int exp\_;

}

%token <INT\_> INT

%left ADD

%left MUL

%token EOL

%type <exp\_> exp;

%%

calclist:

calclist exp EOL { printf("=%d\n", $2); YYACCEPT; }

|

;

exp:

exp ADD exp { $$ = $1 + $3; }

| exp MUL exp { $$ = $1 \* $3; }

| INT { $$ = $1; }

;

%%

lexer.l：

%{

#include <stdio.h>

#include <stdlib.h>

#include "parser.tab.h"

%}

%%

[0-9]+ { yylval.INT\_ = atoi(yytext); return INT; }

"+" { return ADD; }

"\*" { return MUL; }

\n { return EOL; }

[ \t] { /\* 忽略空白字符 \*/ }

. { printf("Mystery character %c\n", \*yytext); }

%%

int main(int argc, char \*\*argv)

{

yyparse();

}

void yyerror(char \*msg){

printf("%s\n",msg);

}

int yywrap(){

return 1;

}

代码提交方式

提交itlab地址后，平台会拉取你的代码仓库并使用下述命令编译执行：gcc ./\*.c

(3) 尽管项目中的.l和.y文件不会影响评测结果，但你依然需要在你的代码仓库中存放这些文件。

(4) 在提交后，请保管好提交地址对应的代码仓库。

说明：

如果有用到fprintf，请改成普通的printf，可以解决cg上无输出的问题

flex根据规则按照顺序进行匹配，因此涉及规则优先级的设置，请妥善考虑.l文件中匹配规则的顺序。

当bison警告出现了二义性文法时，可以添加-Wcounterexamples参数来显示产生规约冲突的例子，以便修改文法。