

专业课程设计I报告

（ 2017 / 2018 学年 第 2学期）

|  |  |
| --- | --- |
| 题 目： | **基于LL(1)分析法的表达式计算器** |

|  |  |
| --- | --- |
| 专 业 | 计算机科学与技术 |
| 学 生 姓 名 | 潘一帆 |
| 班 级 学 号 | B15021030 |
| 指 导 教 师 | 朱枫 |
| 指 导 单 位 | 计算机学院、软件学院、网络空间安全学院 |
| 日 期 | 2018.6.18——2018.6.29 |

**指导教师成绩评定表**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 |  | | 班级学号 |  | | 专业 | |  | | | | |
| 评分内容 | 评分标准 | | | | | | 优秀 | | 良好 | | 中等 | 差 |
| 平时成绩 | 认真对待课程设计，遵守实验室规定，上机不迟到早退，不做和设计无关的事 | | | | | |  | |  | |  |  |
| 设计成果 | 符合题目要求、功能丰富 | | | | | |  | |  | |  |  |
| 程序功能执行的正确性 | | | | | |  | |  | |  |  |
| 界面友好 | | | | | |  | |  | |  |  |
| 程序算法执行效能 | | | | | |  | |  | |  |  |
| 设计考虑优化，体现创新意识 | | | | | |  | |  | |  |  |
| 设计报告 | 文档格式规范、内容详实、反映系统设计流程 | | | | | |  | |  | |  |  |
| 综合分析实验结果，获得合理有效的结论 | | | | | |  | |  | |  |  |
| 能够正确描述复杂工程问题 | | | | | |  | |  | |  |  |
| 验收答辩 | 简练、准确阐述设计内容，能准确有条理回答各种问题，系统演示顺利。 | | | | | |  | |  | |  |  |
| 评分等级 |  | | | | | | | | | | | |
| 指导教师  简短评语 | 该同学出勤率(满勤、较高、一般，较低)，学习态度(端正、较端正、一般、较差)，程序设计基础（好、较好、一般、较差），演示程序(已经、没有)达到了基本要求，算法设计（好、较好、一般），界面友好程度（好、较好、一般），答辩过程中回答问题（准确、较准确、错误率较高），撰写报告格式(规范、一般)、内容（丰满、简单）、表述(清晰、一般、不清楚)，（圆满、较好、基本)完成了课题任务。 | | | | | | | | | | | |
| 指导教师签名 | |  | | | 日期 | | | | |  | | |
| 备注 | 评分等级有五种：优秀、良好、中等、及格、不及格 | | | | | | | | | | | |

**基于LL(1)分析法的表达式计算器**

1. **课题内容和要求**

课题内容：结合编译原理的词法分析、语法分析及语义分析方法，给出表达式计算器的系统设计过程，并用LL(1)分析法实现表达式计算器。

课题要求：

（1）计算器中能实现基本功能如下：支持加、减、乘、除等运算；支持括号运算；对用户输入的表达式是否正确进行判断并给出错误提示；可以对上述功能进行扩充，例如增加支持三角函数和对数函数以及幂运算等；

（2）词法分析、语法分析、语义处理模块结构清晰，可以借助Lex、Yacc或JavaCC、Antlr等工具自动生成词法分析和语法分析程序；

（3）界面美观。

**二、需求分析**

1、用户按键输入表达式进行计算，输出计算结果。

2、具有加减乘除、括号运算、幂运算功能。

3、具有sin、cos、tan、log函数运算。

4、输入表达式不合法会给出错误提示信息。

5、具有清空、退格按键。

**三、概要设计**

1、文法设计：

|  |
| --- |
| E ::= E+T | E-T | T |
| T ::= T\*F | T/F | F |
| F ::= F^M | M |
| M ::= -M | i | (E) | C |
| C ::= n(E) |
| 其中i代表数字，n代表函数名 |

消除左递归后文法：

|  |
| --- |
| E ::= TE’ |
| E’::= +TE’ | -TE’ | ε |
| T ::= FT’ |
| T’ ::= \*FT’ | /FT’ | ε |
| F ::= MF’ |
| F’ ::= ^MF’ | ε |
| M ::= -M | i | (E) | C |
| C ::= n(E) |

|  |  |
| --- | --- |
| 非终结符: | E E’ T T’ F F’ M C |
| 终结符: | + - \* / ^ ( ) i(代表数字) n（代表函数名） |

2、词法分析

词法分析的工作是扫描输入字符串，识别出不同类型的单词，以供语法分析使用。构造的状态转换图如下图所示：

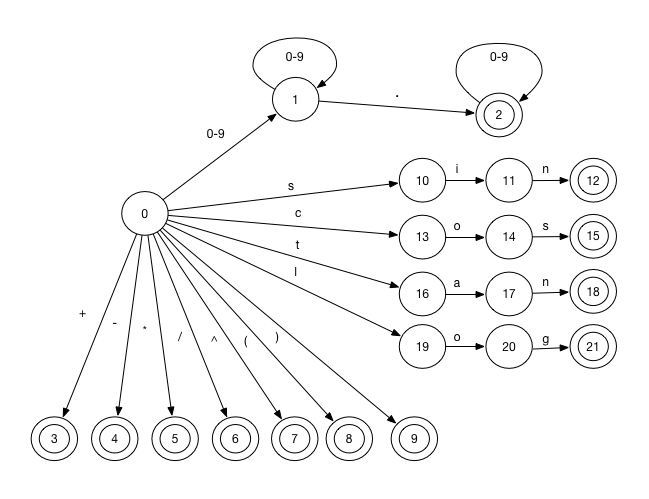


图1 状态转换图

3、语法分析

采用LL(1)分析法，首先构造First集和Follow集。

First集合构造方法如下所示：

对于文法中的每一个文法符X∈(VN∪VT)，构造FIRST(X)时，只要连续使用下列规则直至每个FIRST集不再扩大为止。

1. 若X∈VT，则FIRST(X)={X}
2. X∈VN，且有形如X::=aα规则(a∈VT)，或X::=ε的规则，把a或ε加入FIRST(X)中
3. 设文法G中有形如X::=Y1Y2…Yk的规则，若Y1∈VN，则将FIRST(Y1)中一切非ε符号加入FIRST(X)中，对于一切2≤i≤k，若Y1→\*ε，则把Y2中首符号集（除ε外）也加入FIRST(X)中，如此继续下去，直到Yi-1→\*ε，则把Yk中首符号集（除ε外）也加入FIRST(X)中
4. 若Y1Y2…Yk每个非终结符都可能推导出空符号串，即Y1Y2…Yk→\*ε，则把ε也加入FIRST(X)中

FOLLOW集合构造方法如下所示：

对文法中每个非终结符A，为了构造FOLLOW(A)，可反复使用以下规则，直到每个FOLLOW集都不再扩大为止。

1. 对于文法的开始符号S，令#∈FOLLOW(S)
2. 若文法中有形如A::=αBβ的规则，且β≠ε，则将FIRST(β)中一切非ε符号加入FOLLOW(B)
3. 若文法中有形如A::=αB或A::=αBβ的规则，且β→\*ε，则FOLLOW(A)中全部终结符均属于FOLLOW(B)

根据First集和Follow集构造文法分析表，构造方法如下：

对于G中每一个规则A::=α，可按如下算法确定表中各元素：

1. 对FIRST(α)中每一终结符a，置M[A,a]=” A→α”
2. 若ε∈FIRST(α)，则对属于FOLLOW(A)中的每一符号b(b为终结符或#)，置M[A,b]= ” A→α”
3. 把M中所有不能按上述两条规则定义的元素均值为出错

构造出分析表后，对分析表中每个产生式设计对应的语义规则：

表1 语义规则

|  |  |  |
| --- | --- | --- |
| 产生式 | 语义规则 | 对应子程序 |
| E ::= TE’ | E.syn = E'.syn , E'.inh = T.syn | expr() |
| E’::= +TE’ | E'.syn = E1'.syn, E1'.inh = E'.inh + T.syn | expr\_() |
| E’::= -TE’ | E'.syn = E1'.syn, E1'.inh = E'.inh - T.syn |
| E’::= ε | E'.syn = E'.inh |
| T ::= FT’ | T.syn = T'.syn, T'.inh = F.syn | term() |
| T’ ::= \*FT’ | T'.syn = T1'.syn, T1'.inh = T'.inh \* F.syn | term\_() |
| T’ ::= /FT’ | T'.syn = T1'.syn, T1'.inh = T'.inh / F.syn |
| T’ ::= ε | T'.syn = T'.inh |
| F ::= MF’ | F.syn = F'.syn, F'.inh = M.syn | factor() |
| F’ ::= ^MF’ | F'.syn = F1'.syn, F1'.inh = F'.inh ^ M.syn | factor\_() |
| F’ ::= ε | F'.syn = F'.inh |
| M ::= -M | M.syn = -M1.syn | power() |
| M ::= i | M.syn = i.val |
| M ::= (E) | M.syn = E.syn |
| M ::= C | M.syn = C.syn |
| C ::= n(E) | C.syn = n(E.val) | funcall() |

4、数据结构

设计三个类：铺垫类Prework、词法分析类WordParser、语法分析类Analyzer

铺垫类Prework的任务是计算文法的First集、Follow集，并生成文法分析表。

词法分析类WordParser的任务是进行单词的识别，语法分析类Analyzer负责完成语法分析和语义分析。Prework和WordParser均服务于Analyzer。

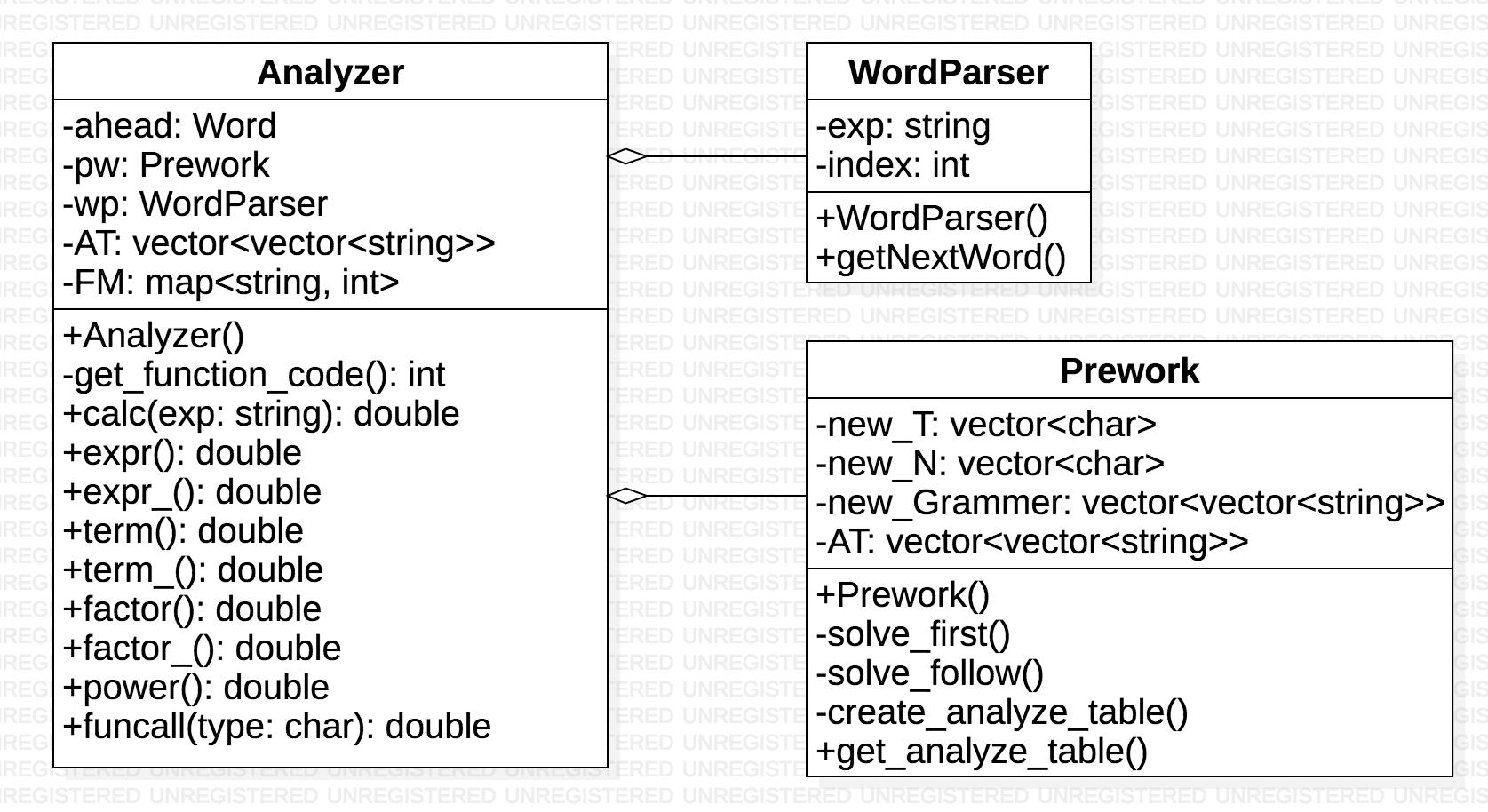


图2 类图

**四、详细设计**

1、铺垫类Prework

void Prework::**solve\_first**() //求解First集

{

int n\_num = new\_N.size();

*for*(int i=0; i<n\_num; i++)

add\_to\_first(i,i);

*for*(int cnt=0; cnt<N; cnt++)

{

*for*(int i=0; i<N; i++)

{

*for*(int j=0; j<new\_Grammer[i].size(); j++)

{

*if*(isN(new\_Grammer[i][j][0]))

add\_first\_to\_first(i, locate(new\_Grammer[i][j][0]));

}

}

}

}

void Prework::**solve\_follow**() //求解FOLLOW集

{

int n\_num = new\_N.size();

follow[0].push\_back('#');

*for*(int cnt=0; cnt<n\_num; cnt++)

{

*for*(int i=0; i<n\_num; i++)

{

*for*(int j=0; j<new\_Grammer[i].size(); j++)

{

add\_to\_follow(i,new\_Grammer[i][j]);

}

}

}

}

求解出的First集和Follow集如下表所示：

表2 First集和Follow集

|  |  |  |
| --- | --- | --- |
| 非终结符 | First集 | Follow集 |
| E | - i ( n | # ) |
| E’ | + - ε | # ) |
| T | - i ( n | # + - ) |
| T’ | \* / ε | # + - ) |
| F | - i ( n | # + - \* / ) |
| F’ | ^ ε | # + - \* / ) |
| M | - i ( n | # + - \* / ^ ) |
| C | n | # + - \* / ^ ) |

void Prework::**create\_analyze\_table**() //创建分析表

{

int n\_num = new\_N.size();

int t\_num = new\_T.size();

*for*(int i=0; i<n\_num; i++)

{

*for*(int j=0; j<new\_Grammer[i].size(); j++)

{

*if*(isN(new\_Grammer[i][j][0]))

{

*for*(int k=0; k<first[locate(new\_Grammer[i][j][0])].size(); k++)

*if*(first[locate(new\_Grammer[i][j][0])][k] != 'e')

AT[i][locate2(first[locate(new\_Grammer[i][j][0])][k])] = new\_Grammer[i][j];

}

*else* *if*(isT(new\_Grammer[i][j][0]))

AT[i][locate2(new\_Grammer[i][j][0])] = new\_Grammer[i][j];

*else*

{

*for*(int k=0; k<follow[i].size(); k++)

*if*(follow[i][k] == '#')

AT[i][t\_num] = new\_Grammer[i][j];

*else*

AT[i][locate2(follow[i][k])] = new\_Grammer[i][j];

}

}

}

}

求解出的文法分析表如表3、表4所示：

表3 文法分析表（一）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ( | ) | + | - | \* |
| E | E::=TE’ |  |  | E::=TE’ |  |
| E’ |  | E’::=ε | E’::=+TE’ | E’::=TE’ |  |
| T | T::=FT’ |  |  | T ::= FT’ |  |
| T’ |  | T’::=ε | T’ ::= ε | T’ ::= ε | T’ ::= \*FT’ |
| F | F::=MF’ |  |  | F::=MF’ |  |
| F’ |  | F’ ::= ε | F’ ::= ε | F’ ::= ε | F’ ::= ε |
| M | M::=(E) |  |  | M ::= -M |  |
| C |  |  |  |  |  |

表4 文法分析表（二）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | / | ^ | i | n | # |
| E |  |  | E::=TE’ | E::=TE’ |  |
| E’ |  |  |  |  | E’::=ε |
| T |  |  | T::=FT’ | T::=FT’ |  |
| T’ | T’ ::= /FT’ |  |  |  | T’ ::= ε |
| F |  |  | F::=MF’ | F::=MF’ |  |
| F’ | F’ ::= ε | F’ ::= ^MF’ |  |  | F’ ::= ε |
| M |  |  | M ::= i | M ::= C |  |
| C |  |  |  | C ::= n(E) |  |

vector<vector<string>> Prework::**get\_analyze\_table**() //转换分析表格式Analyzer

{ 调用

solve\_first();

solve\_follow();

create\_analyze\_table();

vector<vector<string> > AT2(30,vector<string>(256,string()));

*for*(int j=0; j<new\_N.size(); j++){

*for*(int i=0; i<=new\_T.size(); i++){

*if*(i < new\_T.size()){

int pos = locate2(new\_T[i]);

AT2[j][new\_T[i]] = AT[j][pos];

}

*else*{

AT2[j][127] = AT[j][new\_T.size()];

}

}

}

}

2、词法分析类WordParser

Word WordParser::getNextWord(){。//分析下一个单词

Word res;

res.type = END;

res.value = INF;

if(index >= exp.size())

return res;

char now = exp[index];

if(isdigit(now) || now == '.'){ //识别数字

for(res.value = 0; isdigit(now); now = exp[++index])

res.value = 10 \* res.value + (now - '0');

if(now == '.'){

double decplace = 1;

now = exp[++index];

while(isdigit(now)){

res.value = res.value + (now - '0') \* (decplace /= 10);

now = exp[++index];

}

}

res.type = 'd';

return res;

}

if(isalpha(now)){ //识别函数名

char name = now;

string funcname;

while(isalpha(now)){

funcname.push\_back(now);

now = exp[++index];

}

if(isfunc(funcname))

res.type = name;

else

res.type = 'x';

return res;

}

res.type = now; //如果不是数字和函数名，则保存ASCII码为类型值

index++;

return res;

}

3、语法分析类Analyzer

double Analyzer::**calc**(string exp){ //计算表达式

wp.resetIndex();

wp.setExp(exp);

ahead = wp.getNextWord();

*return* expr();

}

int Analyzer::**get\_function\_code**(int n, char c) //选择语义对应的函数

{

*if*(c == 's' || c == 'c' || c == 't' || c == 'l')

c = 'n';

*if*(c == 'd')

c = 'i';

*if*( !(pw.isT(c)||c==END) || AT[n][c] == " ")

*return* -1;

*return* FM[AT[n][c]];

}

double Analyzer::**expr**() {

int code = get\_function\_code(0,ahead.type);

*if*(code == 1)

*return* term() + expr\_();

*else*{

errorHandler();

*return* 0;

}

}

double Analyzer::**expr\_**()

{

char type = ahead.type;

int code = get\_function\_code(1,type);

*if*(code == 4)

*return* 0;

*else* *if*(code == 2){

ahead = wp.getNextWord();

*return* term() + expr\_();

}

*else* *if*(code == 3){

ahead = wp.getNextWord();

*return* -term() + expr\_();

}

*else*{

errorHandler();

*return* 0;

}

}

double Analyzer::**term**() {

int code = get\_function\_code(2,ahead.type);

*if*(code == 5)

*return* factor() \* term\_();

*else*{

errorHandler();

*return* 0;

}

}

double Analyzer::**term\_**() {

char type = ahead.type;

double temp;

int code = get\_function\_code(3,ahead.type);

*if*(code == 4)

*return* 1;

*else* *if*(code == 6){

ahead = wp.getNextWord();

*return* factor() \* term\_();

}

*else* *if*(code == 7){

ahead = wp.getNextWord();

temp = factor();

*if*(temp == 0){

errorHandler();

*return* 1;

}

*else*

*return* 1 / temp \* term\_();

}

*else*{

errorHandler();

*return* 1;

}

}

double Analyzer::**factor**(){

double a,b;

int code = get\_function\_code(4,ahead.type);

*if*(code == 8){

a = power();

b = factor\_();

*if*(a == 0 && b == 0){

errorHandler();

*return* 0;

}

*else*

*return* pow(a,b);

}

*else*{

errorHandler();

*return* 0;

}

}

double Analyzer::**factor\_**(){

char type = ahead.type;

int code = get\_function\_code(5,type);

double a,b;

*if*(code == 4){

*return* 1;

}

*else* *if*(code == 9){

ahead = wp.getNextWord();

a = power();

b = factor\_();

*if*(a == 0 && b == 0){

errorHandler();

*return* 0;

}

*else*

*return* pow(a,b);

}

*else*{

errorHandler();

*return* 1;

}

}

double Analyzer::**power**(){

*//* *cout* *<<* *"power()"* *<<* *endl;*

char type = ahead.type;

int code = get\_function\_code(6,type);

double value = ahead.value;

*if* (type != END)

ahead = wp.getNextWord();

*if*(code == 11)

*return* value;

*else* *if*(code == 12){

value = expr();

ahead = wp.getNextWord();

*return* value;

}

*else* *if*(code == 10){

*return* -power();

}

*else* *if*(code == 13){

*return* funcall(type);

}

*else*{

errorHandler();

*return* 0;

}

}

double Analyzer::**funcall**(char functype){

cout << "funcall()" << endl;

char type = ahead.type;

cout << "type: " << type << endl;

*if*(type == '('){

ahead = wp.getNextWord();

double value;

*switch*(functype){

*case* 's':

value = sin(expr());

ahead = wp.getNextWord();

cout << "ahead: " << ahead.type << endl;

*return* value;

*case* 'c':

value = cos(expr());

ahead = wp.getNextWord();

*return* value;

*case* 't':

value = tan(expr());

ahead = wp.getNextWord();

*return* value;

*case* 'l':

value = log10(expr());

ahead = wp.getNextWord();

*return* value;

}

}

*else*{

errorHandler();

*return* 0;

}

}

**五、测试数据及其结果分析**

1、计算器界面



2、输入正确表达式，计算结果

计算2\*3+2-1/2

经检验，计算结果正确

计算1.2+(3-6)^2-5

经检验，计算结果正确

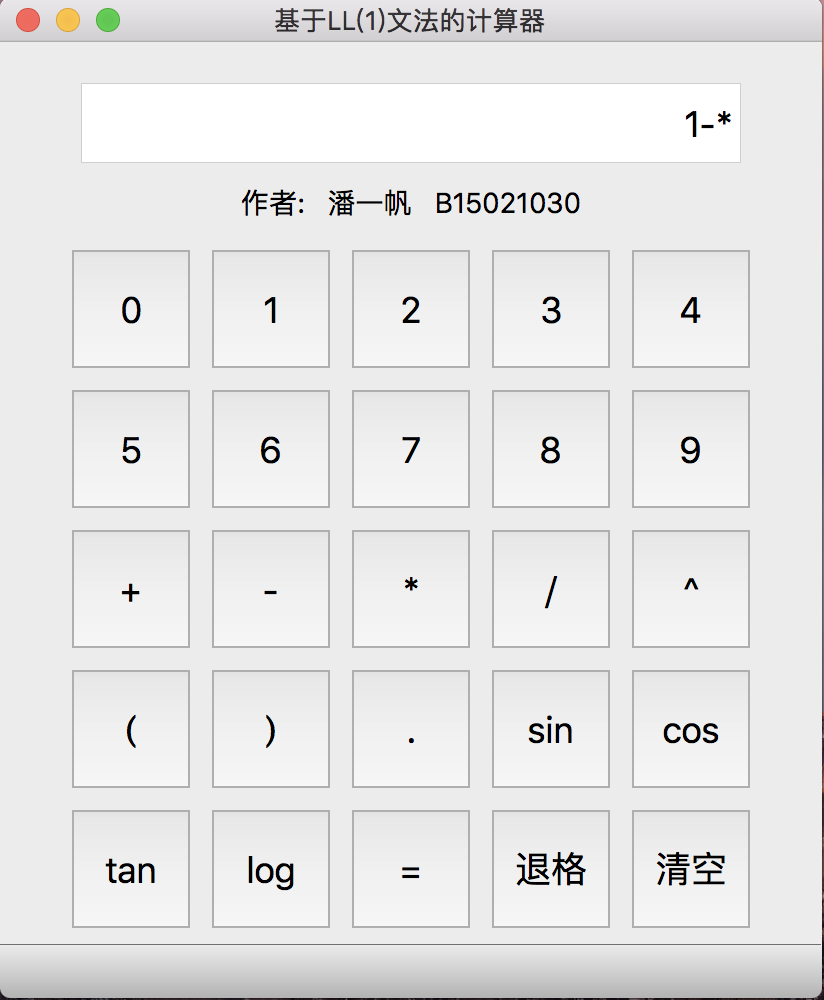
计算2.1+sin(cos(tan(log(3.2))))^2

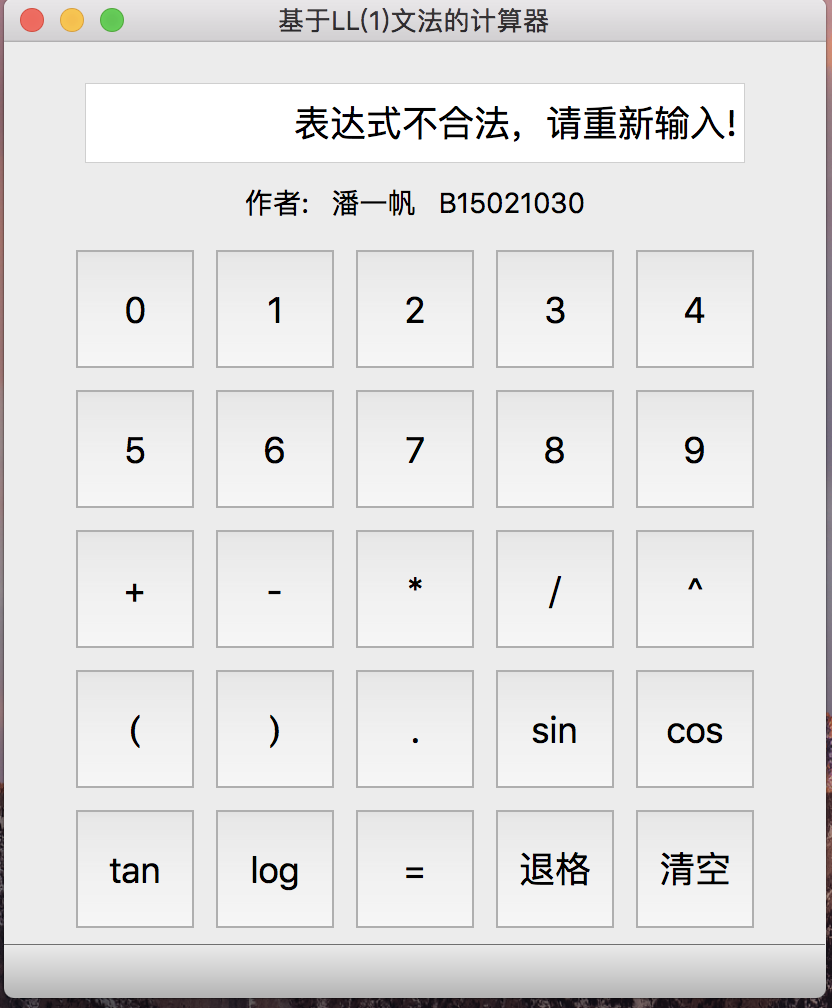
经检验，计算结果正确

3、输入不合法表达式进行测试

计算 1-\*

计算1/0

计算0^0

从运行结果可以看出程序能正确处理用户不合法的输入。

**六、调试过程中的问题**

一开始做出的计算器的计算顺序存在问题，幂运算的优先级和乘除一样，仔细检查后发现是文法设计存在问题，幂运算应该在乘除之前规约，将幂运算提取出来，即添加一层语法规则F ::= F^M | M，然后修改对应的语法分析程序，最后运算的顺序终于正确了。

**七、参考文献和查阅的资料**

|  |  |  |  |
| --- | --- | --- | --- |
| 文档名称 | 版本号 | 日期 | 出版单位/来源 |
| 编译原理及其实现方法 | 第一版 | 1998年11月 | 成都科技大学出版社 |
| 编译原理 | 第一版 | 2003年8月 | 机械工业出版社 |

**八、课程设计总结**

**（一）遇到的难点及解决办法**

遇到的难点在于得到文法分析表后如何编写语法制导翻译程序，如何给每个产生式制定相应的语义规则，这部分内容教材上没有详尽的解释，于是我查阅了编译原理领域中大名鼎鼎的“龙书”，自学了相关章节，了解了综合属性和继承属性及相应语法制导技术，结合书中相关的例子，给我的文法设计了正确的语义规则并编写出了完善的程序。

**（二）心得与体会**

这次课程设计还是比较复杂的，不过好在之前编译原理课做过词法分析和LL(1)语法分析的实验，这次直接把实验写的代码拿过来用了，生成了最关键的文法分析表，然后只要对着那张表编写分析程序就行了，大大减轻了工作量。这次课程设计让我把词法分析、语法分析结合了起来，完整的实现了一个简易的算术文法编译器，对编译原理的精髓有了更好的体会。