表达式的处理：

从键盘输入一个字符串，由数字、运算符和括号组成；

要求判断表达式数学意义上是否成立，若成立，则计算结果。

运算符有：

+

-

\*

/

^

数学意义上成立的意思，包括：

括号匹配正确；表达式正确；数字正确。

e.g.正确的表达式：

35.6 \* (4.6^(5 + 4.3) + 5.7)

特殊的正确情况：

3 + (4)

.5 \* 6

普通的错误情况：

34.5+\*6.7

)4\*5

特殊错误：

3 + ()

3. \*5

上述过程相当于我们自己定义了一套语法，制定了其语法，词法。

编译基础：

编译系统是将源代码程序文本文件中的字符串转换成最终能在裸机上执行的可执行语言的系统软件。

其工作过程基本上分为四大步：

1、词法分析；

2、语法分析；

3、语义分析；

4、优化。

词法分析：将源代码字符串，分割成一个一个的基本单元：变量名、常量、运算符、函数、关键字等等。

double a = 3.45 \* 5;

[double] [a] [=] [3.45] [\*] [5] [;] <==> [类型][变量名][=][表达式][;]

[表达式]部分 常量 \* 常量

符合变量名定义及赋初值语法。

语义分析主要完成：类型匹配。

有关词法分析：

‘\’’

为了能表达一个’字符，如果直接如下：

‘’’

C语言会认为这是语法错误，其实，这在词法分析时就能判定！

‘后面只能跟如下几种字符：

\

除了’以外的所有字符

上述内容其实就是字符常量词法分析的框架。

括号匹配的判断：假设字符串只有()组成；要求给一段时间复杂度不大于O（N），且程序长度不超过10行的程序。

意思是，只需要将字符串遍历一遍（甚至不到一遍），就必须得到判断结果。

解决方案：

设定两个栈顶指针：middleMatch和bigMatch，并设其初值为0，对字符串进行遍历，遇到左括

号相应的栈顶指针++，遇到与之匹配的右括号，相应的栈顶指针--，若其中任意一个栈顶指针在

遍历字符串的过程中为负数，则代表右括号多了，不匹配，若遍历完成字符串后相应的栈顶指针

不为零，则代表括号不匹配，并且可以得知不匹配的原因：

若栈顶指针大于零，则代表左括号多了，若小于零则代表右括号多了;

将字符串中，从当前位置开始，能够构成正确小数常量的字符串内容转换策划国内与其字面值相同的数值：

e.g.：

“34.769 + 56.78” => 34.769

正确的小数数值的常量表达形式：

+3.4 -5.67 5 .5 -.5

不正确的小数数值表达形式：

5. ++3 --5 4.5.6 +-5 . 4.-5 .5.6 4.12345678 45.67空格8

构成上述表达式的字符种类有4种：

1、数字字符（数）；

2、+/-号（号）；

3、.（点）；

4、结束标志（结）。

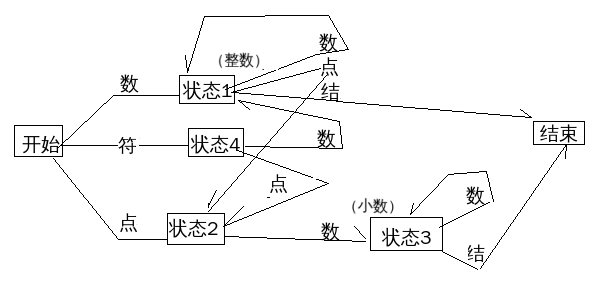
始：数|点|号

数：数|点|结

号：数|点

点：数

状态变迁：



开始状态

状态1：整数

状态2：点

状态3：小数

状态4：符号

结束状态

假设有字符串：

3.45 + 5.676

getNum()函数可以取得3.45，但是u，对于后面的运算符和第二运算数，无法继续处理。因为，我们不知道第一运算数到底由几个符号组成。

3.14+5.67

1、取出3.14；

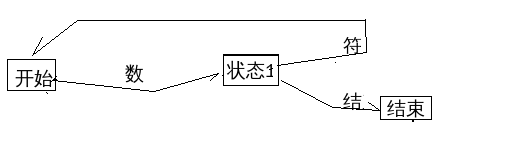
2、取出5.67

3、取出+

4、3.14+5.67

5、输出结果。

不带括号的表达式状态变迁图：



while(结束标志 != TRUE)

取数V；

V入运算数堆栈；

取运算符C；

while(栈非空 && !(C比运算符堆栈栈顶运算符优先级高))

计算；

if(C是0结束标志)

结束标志 = TRUE；

else

C入运算符堆栈；

计算：

将运算数堆栈栈顶运算数出栈作为右操作数；V1

运算数堆栈栈顶指针--；

将运算数堆栈栈顶运算数出栈作为左操作数；V2

将运算符堆栈栈顶运算符出栈；C

运算当前表达式；V1 C V2 --->V1;

V1 将运算结果入运算数堆栈；

while(没结束){

if(status == 开始){

if(str[i]是数值)；

getNum(str, &i);

V入运算数堆栈；

status = 状态1；

else

语法错误；

}

else if(status == 状态1){

if(str[i] == 运算符)

取运算符C；

while(栈非空 && !(C比运算符堆栈栈顶运算符优先级高))

计算；

if(C是0结束标志)

结束标志 = TRUE；

else

C入运算符堆栈；

else if(str[i] == 结束标志)

status == 结束状态；

else

语法错误；

}

}

isStackLikeEmpty(STACK stack, char emptyFlag){

ELEMENT v;

boolean ok = TRUE;

readTop(stack, &v);

if(v.oper == emptyFlag)

ok = FALSE;

return ok;

}

带括号的表达式状态变迁图：

