[1：目标 1](#_Toc30729)

[2：难点 1](#_Toc31111)

[2-1：解决加、减、乘、除、atan等运算符的优先级 1](#_Toc28183)

[2-2：当语法解析程序出现问题时，如何恢复到最新一次解析成功的节点，以及如何跳过不成功的部分。 1](#_Toc19772)

[2-3：如何定位出错的语句的位置 1](#_Toc16768)

[3：具体实现 2](#_Toc5900)

[3-1：定义需要使用的符号 2](#_Toc2323)

[3-2：定义语法 2](#_Toc32234)

[3-3：定义lex函数 3](#_Toc5584)

[4：定义main函数，调用yyparse函数开始解析 3](#_Toc9548)

# 1：目标

实现一个支持加、减、乘、除、atan、cos、exp、log、sin、sqrt、ceil、floor功能的计算器。

# 2：难点

## 2-1：解决加、减、乘、除、atan等运算符的优先级

比如说1 + 3 \* 5，可以解析成（1 + 3） \* 5，同样也可以解析为1 + （3 \* 5），如何正确地进行解析。

## 2-2：当语法解析程序出现问题时，如何恢复到最新一次解析成功的节点，以及如何跳过不成功的部分。

比如说

a = 4

1 + + 7

解析上面语句时，a = 4可以被成功解析，但是1 + + 7由于两个+之间没有具体的数字，因此无法进行成功的解析。当遇到这种情况时，比较科学的做法是将1 ++ 7这条语句作废，然后继续进行接下来的解析。

## 2-3：如何定位出错的语句的位置

作为一个合格的编译器，应当在语句出现问题时，提示使用人员，出现问题的语句的具体位置。

还是拿 1 + + 7举例子，其出错的位置应当是第3列，有一个多余的+。

# 3：具体实现

由于前期语法结构可能会发生改变我们这边使用gnu的[bison](https://www.gnu.org/software/bison/manual/bison.pdf)作为我们的解析器生成器，用于解析对应的使用BNF编写的计算器描述语言。

## 3-1：定义需要使用的符号

1：NUM 表示数字

2：VAR 表示变量

3：FNCT 表示函数

4：exp表示表达式（比如1 + 2 + 3 \* sin(1)）

5：单字符符号 = - + \* / ^

6：优先级符号 NEG

token表示terminal symbol

type表示unterminal symbol

precedence 表示优先级

left表示优先级以及左结合

right表示优先级以及右结合

## 3-2：定义语法

input符号，定义语法的起点，可以是空，也可以是line符号的左嵌套结构（为什么要进行左嵌套？），表示可以由多个line符号。

line符号，可以是换行符、或者exp符号后面跟换行符、或者error后面跟换行，error是bison中的一个保留符号，当exp符号在解析过程中出现错误时，bison可以将状态回退到最近一次解析成功的位置，然后将error符号放入解析栈中，然后舍弃掉回车符前面所有的符号。

exp符号，可以是表示数字的NUM符号、可以是给一个变量赋值VAR = exp、可以是调用一个函数FNCT(exp)、可以是普通的四则运算。当进行除法运算时，需要注意除数不能为0，如果为0时需要显示出错的位置。

## 3-3：定义lex函数

lex用来给bison生成的解析程序提供对应的symbol，包括terminal symbol以及group symbol。

这里的lex函数对终端中的一行字符串进行分析，从左至右，如果是十进制数字，则返回NUM，如果是字母开头的字符串的变量，从变量表中查找对应的变量；如果是字母开头的函数，从函数表中查找对应的函数，并且返回。

lex函数中，还对字符串的位置进行了定义。

# 4：定义main函数，调用yyparse函数开始解析

解析函数会不停地调用lex函数，直至lex函数返回EOF或者解析出现问题。