# calc文档

# 原理

一个四则运算表达式，在去掉括号的情况下，每个数字只和其前后运算符有关，例如：

a+b\*c=

此时，可以将最后的’=’（或者’\0’）看做运算符，则每个数字均有其后紧跟的一个运算符对应，且只需判断相邻两个运算符，即可进行运算。

即上式拆分为a+，b\*，c=三项，顺序比较运算符，我们设定+的优先级低于\*，则将a+入栈，继续比较b\*和c=，我们将=的优先级设为最低，则此时进行b\*c运算，结果将合并至c=，然后a+出栈，与c=比较，+的优先级高于=，即进行a+c运算，并可得出结果。

与其他实现方法不同的是，这个实现只需要遍历1次表达式，在遍历过程中实时解析，实时检查，经过简单修改即可实现边输入边计算。

# 数据结构

栈：

typedef struct stack\_st {

size\_t top;

size\_t size;

void \*base[0];

}STACK;

数据：

typedef struct data\_st {

double num; //数字

int prio; //当前运算优先级

char sign; //运算符

}DATA;

# 主要函数接口

数据获取函数：

static int get\_data(DATA \*data)

从表达式（字符串）获取1个DATA结构的数据，即1个数字和1个运算符。

运算函数：

static int calculation(double \*result)

运算过程，包括入栈出栈等。

栈操作全部封装在stack.h中：

static inline STACK\* NEW\_STACK(size\_t size);

static inline void DESTROY\_STACK(STACK \*s);

STACK\_FULL(stack)

STACK\_EMPTY(stack)

STACK\_PUSH(stack, data)

STACK\_POP(stack)

# 表达式检查表

/\* 0 1 2 3 4 5 6 7

初始化 +,- \* / 数字 ( ) = \*/

int check[8][8] = {

{ERR, OK , ERR, ERR, OK , OK , ERR, ERR},

{ERR, ERR, ERR, ERR, OK , OK , ERR, ERR},

{ERR, ERR, ERR, ERR, OK , OK , ERR, ERR},

{ERR, ERR, ERR, ERR, OK , OK , ERR, ERR},

{ERR, OK , OK , OK , ERR, ERR, OK , OK },

{ERR, OK , ERR, ERR, OK , OK , ERR, ERR},

{ERR, OK , OK , OK , ERR, ERR, OK , OK },

{ERR, ERR, ERR, ERR, ERR, ERR, ERR, ERR}};

本质上是一个状态表，行表示上一次的状态，列表示本次状态，例如1(2，这是一个错误的表达式，首次取1(数字)，对应值为4，即check[0][4]=OK，第二次取(，对应值为5，即check[4][5]=ERR，即可检测出错误表达式。

# 用法

直接将表达式作为参数输入即可：

root@ubuntu:~/code/t# ./calc "(1.11-2.11)\*1.0009+(2+3.2332)\*2.111"

10.046385