

# Project 2 A better calculator

---

Name: 郭一潼

Student id:11911702

CS205 Project2 Report The code is at <https://github.com/Godblessmycode1/CS205>.

- 1.介绍
  - Project要求介绍
  - Project完成情况介绍
  - 开发环境
- 2.基本数据结构及代码思路
  - 数据结构介绍
  - Num内部结构
  - OperandStack内部结构
  - OperatorStack内部结构
  - Set内部结构
  - Map内部结构
- 3.文件结构及核心代码
  - Num类型加法
  - Num类型减法
  - Num类型乘法
  - Num类型除法
  - 操作符计算
  - 遇到右括号的操作
- 4.代码支持拓展部分
- 5.结果

## 1. 介绍

### 1.1 Project要求介绍

1. 计算器支持正确的表达式计算以及输出正确结果
2. 计算器支持括号使用来加强优先级
3. 计算器支持变量赋值及使用
4. 计算器支持函数使用
5. 计算器支持任意精度

### 1.2 Project完成情况介绍

本次project,我实现了一下几种功能。


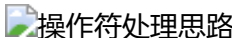
1. 此计算器支持输入保护以及检验函数和变量是否存在
2. 此计算器支持表达式计算
3. 此计算器支持变量使用,但需要对变量进行赋值(变量只支持'a'到'z'开头,后续只要不是操作符便不限制)
4. 此计算器支持部分函数使用,同时支持函数的二次开发(代码逻辑是不变的,需要使用者自己把函数名加到函数list里面,同时将函数名以及函数逻辑加到solve.cpp solve()函数里),**同时函数括号内支持表达式输入。**

5. 此计算器支持任意精度
6. 此计算器支持括号使用来提高优先级

## 1.3 开发环境

- x86\_64
  - vscode (version 1.71)
  - WSL (version 2)
  - Ubuntu(22.04)
  - g++(11.2.0)

## 2. 基本数据结构及代码思路

本次Project使用将中缀表达式变为后缀表达式的计算方式。其中使用了两个栈，一个为操作数栈，用于储存操作数。另一个为操作符栈，用于储存操作符函数名等。代码思路下图： 

### 2.1 数据结构介绍

本Project使用了五种数据结构:

- Num 用于计算的基本单元
- OperandStack 用于储存操作的栈
- OperatorStack 用于储存操作符的栈
- set 用于储存和符合定义的操作符以及函数名
- map 用于储存变量名以及其对应的Num 方便后续遇到时 push到OperandStack里

### 2.2. Num内部结构

本次计算的基本结构为Num

```
struct Num
{
    char input_num [200]={0};
    int scale=0;
};
```

其中char数组储存 输入的数的值，scale储存小数点位数。

Eg:  $2.0=20 \times 10^{-1}$  在 Num 的储存形式中 input\_num []储存 20, scale 为-1。

通过这种方式，可以将数组右移方式不断调整精度，从而达到实现任意精度的目的。

### 2.3 OperandStack内部结构

操作数栈用于储存操作数也就是Num

```
struct OperandStack
{
    Num operand_stack[200];
};
```

```
int top=-1;
};
```

## 2.4 OperatorStack内部结构

```
struct OperatorStack
{
    char operator_stack[200];
    int top=-1;
};
```

## 2.5 Set结构

```
const set<string>functions{"sqr"}; //函数表, 储存函数名字
set<char>operators {'+', '-', '*', '/', '(', ')'}; //operator集合
```

## 2.6 Map结构

```
map<string,Num> variable_list;//used to stored the variable already input
map<string,char> function_in_stack={{ "sqr", 's' }}; //因为operator_stack 里只能存char,
那么应该有个对应表
```

# 3 文件结构以及核心代码

main函数负责判断是赋值等式还是计算表达式, 若为赋值等式则创建变量并储存在variable\_list中, 若为等式则分解, 并将operator存到operator\_stack中, 将operand存到operand\_stack中。

solve.cpp主要储存的是根据operand\_stack以及operator\_stack的内容进行抽象操作的方法。

structure.cpp主要储存的是本次project中使用的structure以及Num 类型的具体加减乘除操作。

其中调用关系为, solve.cpp需要用到structure.cpp

main.cpp需要调用solve.cpp和structure.cpp

## 3.1 Num类型加法

为了支持无限精度,需要比较num1.scale以及num2.scale然后将较大scale的Num的input\_num右移他们scale的差值位, 再相加。Eg:  $3.1415 + 2 = 5.1415$  Num1指前者, 它的input\_num为51413初始化已反转,它的scale为-4,num2的input\_num位2, 它的scale为0, 则此时为了保证Num2的scale为-4维持精度不变, 需要将Num2的input\_num右移四位补零, 即00002, 再相加, 结果为51415, scale为-4。

Code part

```
int compareArrayByteByByte(char input1[],char input2[]){//默认都是非负数。因为负数
之间比较都可以转为整数取负。
    int len1=strlen(input1);
    int len2=strlen(input2);
```

```

int len=max(len1,len2);
int first=len-1;
while(first>=0){
    if(input1[first]<input2[first]){
        return -1;
    }
    else if(input1[first]==input2[first]){
        first--;
    }
    else{
        break;
    }
}
if(first<0){
    return 0;
}
return 1;
}

int compareArray(char input1[],char input2[]){ //用于比较两数大小。1代表
input1>=input2,-1则相反。此时已经根据scale差值,完成移位。
    int len1=strlen(input1);
    int len2=strlen(input2);
    //只考虑input1>input2的情况。
    if(input1[len1-1]!='-'&&input2[len2-1]=='-'){//一正一负
        return 1;
    }
    else if(input1[len1-1]!='-'&&input2[len2-1]!='-'){ //同为正。
        return compareArrayByteByByte(input1,input2);
    }
    else if(input1[len1-1]=='-'&&input2[len2-1]=='-'){
        char temp1[200];
        char temp2[200];
        strncpy(temp1,input1+0,len1);
        strncpy(temp2,input2+0,len2);
        temp1[len1-1]=0;
        temp2[len2-1]=0;
        return -compareArrayByteByByte(temp1,temp2);
    }
    return -1;
}

void addArray(char res[],char input1[],char input2[]){//数组已经反转并且移动完。默认
都是正数。
    int len1=strlen(input1);
    int len2=strlen(input2);
    int len=max(len1,len2);
    for(int i=0;i<len;i++){
        if(input1[i]!=0&&input2[i]!=0){
            res[i]=input1[i]-'0'+input2[i]-'0';
        }
        else if(input1[i]!=0){
            res[i]=input1[i]-'0';
        }
        else{
            res[i]=input2[i]-'0';
        }
    }
}

```

```

    }
}
for(int i=0;i<len;i++){
    if(res[i]>=10){
        res[i+1]+=res[i]/10;
        res[i]=res[i]%10;
    }
}
for(int i=0;i<=len;i++){
    res[i]=res[i]+'0';
}
int first=len;
while(first>0){
    if(res[first]=='0'){
        res[first]=0;
        first--;
    }
    else{
        break;
    }
}
}

void addNum(Num* res,Num* num1,Num* num2){ //用于两数字相加
    moveArrayRightBeforeCal(res,num1,num2);
    int len1=strlen((*num1).input_num);
    int len2=strlen((*num2).input_num);
    if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]=='-'){
        (*num1).input_num[len1-1]=0;
        (*num2).input_num[len2-1]=0;
        addArray((*res).input_num,(*num1).input_num,(*num2).input_num);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
    else if((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]!='-'){
        addArray((*res).input_num,(*num1).input_num,(*num2).input_num);
    }
    else if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]!='-'){
        (*num1).input_num[len1-1]=0;
        if(compareArray((*num1).input_num,(*num2).input_num)>=0){
            minusArray((*res).input_num,(*num1).input_num,(*num2).input_num);
            int len=strlen((*res).input_num);
            (*res).input_num[len]='-';
        }
        else{
            minusArray((*res).input_num,(*num2).input_num,(*num1).input_num);
        }
    }
    else{
        (*num2).input_num[len2-1]=0;
        if(compareArray((*num1).input_num,(*num2).input_num)>=0){
            minusArray((*res).input_num,(*num1).input_num,(*num2).input_num);
        }
        else{
            minusArray((*res).input_num,(*num2).input_num,(*num1).input_num);
        }
    }
}

```

```

        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
}
}

```

### 3.2 Num类型减法

减法本质也是加法。思路是类似的。

Code part

```

void minusNum(Num* res,Num* num1,Num* num2){//两数相减
moveArrayRightBeforeCal(res,num1,num2);
int len1=strlen((*num1).input_num);
int len2=strlen((*num2).input_num);
if((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]!='-'){
    if(compareArray((*num1).input_num,(*num2).input_num)>=0){
        minusArray((*res).input_num,(*num1).input_num,(*num2).input_num);
    }
    else{
        minusArray((*res).input_num,(*num2).input_num,(*num1).input_num);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
}
else if((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]=='-'){
    (*num2).input_num[len2-1]=0;
    addArray((*res).input_num,(*num1).input_num,(*num2).input_num);
}
else if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]!='-'){
    (*num1).input_num[len1-1]=0;
    addArray((*res).input_num,(*num1).input_num,(*num2).input_num);
    int len=strlen((*res).input_num);
    (*res).input_num[len]='-';
}
else if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]=='-'){
    (*num2).input_num[len2-1]=0;
    (*num1).input_num[len1-1]=0;
    if(compareArray((*num1).input_num,(*num2).input_num)>0){
        minusArray((*res).input_num,(*num1).input_num,(*num2).input_num);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
    else{
        minusArray((*res).input_num,(*num2).input_num,(*num1).input_num);
    }
}
}
}

```

### 3.3 Num类型乘法

Num1和Num2的乘法储存到Num res中那么Num res.scale=num1.scale+num2.scale,Num res.input\_num可以将上次乘法器的核心代码复用。

Code part

```
void mulArrays(char res[],char input1[],char input2[]){//默认都为正数。
    int len1=strlen(input1);
    int len2=strlen(input2);
    int len=len1+len2;
    for(int i=0;i<len1;i++){
        for(int j=0;j<len2;j++){
            res[i+j]+=(input1[i]-'0')*(input2[j]-'0');
            if(res[i+j]>9){
                res[i+j+1]+=res[i+j]/10;
                res[i+j]=res[i+j]%10;
            }
        }
    }
    for(int i=0;i<len;i++){
        res[i]='0';
    }
    int first=strlen(res)-1;
    while(first>0){
        if (res[first]=='0')
        {
            res[first]=0;
            first--;
        }
        else{
            break;
        }
    }
}

void mulNum(Num* res,Num* num1,Num* num2){
    int len1=strlen((*num1).input_num);
    int len2=strlen((*num2).input_num);
    (*res).scale=(*num1).scale+(*num2).scale;
    if((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]!='-'){
        mulArrays((*res).input_num,(*num1).input_num,(*num2).input_num);
    }
    else if ((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]=='-'){
        (*num2).input_num[len2-1]=0;
        mulArrays((*res).input_num,(*num1).input_num,(*num2).input_num);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
    else if ((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]!='-'){
        (*num1).input_num[len1-1]=0;
        mulArrays((*res).input_num,(*num1).input_num,(*num2).input_num);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
}
```

```

    else{
        (*num1).input_num[len1-1]=0;
        (*num2).input_num[len2-1]=0;
        mulArrays((*res).input_num,(*num1).input_num,(*num2).input_num);
    }
}

```

### 3.4 Num类型除法

除法类型我的想法是为了确保精度，首先应当找到num1.scale和num2.scale中较小的部分。将num res的scale设为其较小，并移动位置。除法本质根据被除数的长度来截取除数的一部分（可能需要借位）然后不断进行相减。**本计算器计算小数除法时结果为四舍五入，整数为向下取整。**

Eg:  $0.2/0.5 = 20/5 \times 10^{(-1)} = 0.4$  首先num1.input\_num为2,num2.input\_num为5, num1.scale=-1,num2.scale=-1, 所以需要将num1.input\_num右移一位, 浮点数除法, 为了支持四舍五入, 需要再移动一位, 将num1.scale右移一位. num1.input\_num为002,再与num2.input\_num不断相减. num2.input\_num长度为1,首先取num1.input\_num最高位, 为2小于五, 无法相见, 所以res.input\_num[2]为0, 然后看num1.input\_num的左数第一位, 为0, 但前一位不为0, 因此需要借位, 结果是20, 循环相减知道不能减了为止. res.input\_num[1]为4, 同理res.input\_num[0]为0, 四舍五入后结果为 res.input\_num={'4','0'} res.scale=-1, 因此输出0.4

Code part

```

void divideArrays(char res[],char input1[], char input2[],int scale){//scale用于储存目标小数精度,input1是被除数, input2是除数,temp储存目前的小数位数。
    if(scale!=0){
        moveArrayRight(input1,1);
    }
    int len1=strlen(input1);
    int len2=strlen(input2);
    char temp[len2]={0};//用于截取这段区间来和被除数比较大小。
    for(int i=len1-len2;i>=0;i--){
        memset(temp,0,sizeof(temp)); //清零
        if(i!=len1-len2&&input1[i+len2]>'0'){
            input1[i+len2-1]+=(input1[i+len2]-'0')*10;
            input1[i+len2]='0';
        }
        strncpy(temp,input1+i,len2);
        temp[len2]=0;
        while(compareArray(temp,input2)>=0){
            minusArray(temp,temp,input2);
            for(int j=0;j<len2;j++){
                input1[j+i]=temp[j];
            }
            res[i]++;
            memset(temp,0,sizeof(temp));
            strncpy(temp,input1+i,len2);
            temp[len2]=0;
        }
    }
    if(scale!=0){
        int temp1=res[0];

```



```

        if(res[0]>=5){
            res[1]++;
        }
        for(int i=1;i<max(len1-len2,0);i++){
            if(res[i]>=10){
                res[i+1]=res[i+1]+res[i]/10;
                res[i]=res[i]%10;
            }
        }
        for(int i=0;i<max(len1-len2+1,0);i++){
            res[i]=res[i]+'0';
        }
        moveArrayLeftOneBit(res);
    }
    else{
        for(int i=max(len1-len2,0);i>=0;i--){
            res[i]=res[i]+'0';
        }
    }
    int first=strlen(res)-1;
    while(first>0){
        if(res[first]=='0'){
            res[first]=0;
            first--;
        }
        else{
            break;
        }
    }
}

void divNum(Num* res,Num* num1,Num* num2){
    int scale=min((*num1).scale,(*num2).scale);
    (*res).scale=scale;
    moveArrayRight((*num1).input_num,(*num1).scale-(*num2).scale-scale);
    int len1=strlen((*num1).input_num);
    int len2=strlen((*num2).input_num);
    if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]=='-'){
        (*num1).input_num[len1-1]=0;
        (*num2).input_num[len2-1]=0;
        divideArrays((*res).input_num,(*num1).input_num,(*num2).input_num,scale);
    }
    else if((*num1).input_num[len1-1]!='-'&&(*num2).input_num[len2-1]!='-'){
        (*num2).input_num[len2-1]=0;
        divideArrays((*res).input_num,(*num1).input_num,(*num2).input_num,scale);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
    else if((*num1).input_num[len1-1]=='-'&&(*num2).input_num[len2-1]!='-'){
        (*num1).input_num[len1-1]=0;
        divideArrays((*res).input_num,(*num1).input_num,(*num2).input_num,scale);
        int len=strlen((*res).input_num);
        (*res).input_num[len]='-';
    }
    else{

```

```

        divideArrays((*res).input_num,(*num1).input_num,(*num2).input_num,scale);
    }

}

```

### 3.5 操作符计算

若operator是函数名则只需要操作栈顶一个元素，若为操作符+ - \* /等则需要两个栈顶元素进行操作。

Code part

```

void solve(char operator_input,OperandStack* operand_stack,OperatorStack*
operator_stack){
    if(!isEmptyOperatorStack(operator_stack)&&function_char_in_stack-
>count(operator_input)!=0){ //调用函数的话只需要一个操作数。
        Num* operand;
        Num res;
        if(operator_input=='s'){ //这里代表平方函数 just a example
            operand=topOperandStack(operand_stack);
            popOperandStack(operand_stack);
            mulNum(&res,operand,operand);
            pushOperandStack(operand_stack,&res);
            popOperatorStack(operator_stack);
        }
        return;
    }
    if ((*operand_stack).top<1) //加减乘除必须至少两位操作数。
    {
        return;
    }
    Num* operand1;
    Num* operand2;
    Num res;
    operand2=topOperandStack(operand_stack); //后一操作数
    popOperandStack(operand_stack);
    operand1=topOperandStack(operand_stack); //前一操作数
    popOperandStack(operand_stack);
    if(operator_input=='+'){ //开始运算。
        addNum(&res,operand1,operand2);
    }
    else if(operator_input=='-'){
        minusNum(&res,operand1,operand2);
    }
    else if(operator_input=='*'){
        mulNum(&res,operand1,operand2);
    }
    else if(operator_input=='/'){
        divNum(&res,operand1,operand2);
    }
    pushOperandStack(operand_stack,&res); //push结果
    popOperatorStack(operator_stack); //把操作符pop掉，已经计算完了。
}

```

### 3.6 遇到右括号的操作

遇到)的情况下, 需要不断执行operator直到(为止, 如果operator\_stack非空情况下还需要top是否为函数, 若是函数的话需要执行函数。

Eg:sqr(2+3) (sqr是平方函数, 本计算器支持) 当读到)时, 执行2+3, 此时operand\_stack的栈顶为5,同时发现operator\_stack栈顶是sqr,因此执行函数, 将5 pop出, 再将25push进operand\_stack。**本计算器通过这种方式来运行函数, 因此函数支持表达式。**

Code part

```
void solveRightBra(OperatorStack* operator_stack,OperandStack* operand_stack){ //
遇到右括号是需要计算。到左括号之后还要检验是否为函数。

while(!isEmptyOperatorStack(operator_stack)&&topOperatorStack(operator_stack)!='(')
{
    solve(topOperatorStack(operator_stack),operand_stack,operator_stack);
}
popOperatorStack(operator_stack); //把左括号pop掉。
if(!isEmptyOperatorStack(operator_stack)&&function_char_in_stack-
>count(topOperatorStack(operator_stack))!=0){ //判断是否为函数调用。
    solve(topOperatorStack(operator_stack),operand_stack,operator_stack);
}
}
```

## 4.如何自定义拓展函数

本项目支持二次开发首先在main.cpp中找到 **const setfunctions{"sqr"}**, 将函数名加入其中, 然后在main.cpp中找到**const map<string,char> function\_in\_stack={{"sqr",'s'}}**, (operator\_stack里是char数组, 因此在map中要加入function名字对应的char),最后在solve.cpp中找到solve函数, 并拓展代码。 Code part

```
void solve(char operator_input,OperandStack* operand_stack,OperatorStack*
operator_stack){
    if(!isEmptyOperatorStack(operator_stack)&&function_char_in_stack-
>count(operator_input)!=0){ //证明是函数同时, 调用函数的话只需要一个操作数。
        Num* operand;
        Num res;
        if(operator_input=='s'){
            operand=topOperandStack(operand_stack);
            popOperandStack(operand_stack);
            mulNum(&res,operand,operand);
            pushOperandStack(operand_stack,&res);
            popOperatorStack(operator_stack);
        }
        ////在这里进行函数自定义!!!!
        //begin
        //here is your function code
        //end
        return;
    }
}
```

```

if ((*operand_stack).top<1) //加减乘除必须至少两位操作数。
{
    return;
}
Num* operand1;
Num* operand2;
Num res;
operand2=topOperandStack(operand_stack); //后一操作数
popOperandStack(operand_stack);
operand1=topOperandStack(operand_stack); //前一操作数
popOperandStack(operand_stack);
if(operator_input=='+'){ //开始运算。
    addNum(&res,operand1,operand2);
}
else if(operator_input=='-'){
    minusNum(&res,operand1,operand2);
}
else if(operator_input=='*'){
    mulNum(&res,operand1,operand2);
}
else if(operator_input=='/'){
    divNum(&res,operand1,operand2);
}
pushOperandStack(operand_stack,&res); //push结果
popOperatorStack(operator_stack); //把操作符pop掉，已经计算完了。
}

```

## 5.结果

```

Solve complete!
2+3
5
Solve complete!
5+2*3
11
Solve complete!
x=3
y=6
x+2*y
15
Solve complete!
sqr(sqr(1.0)+sqr(2.0))
25.0000
Solve complete!
99999999999.22222+1.0
100000000000.22222
Solve complete!

```

### 对于非法输入

```

lucky@LAPTOP-QJU5Q8M6: /mnt/c/Users/84781/Desktop/课程/大四上/cs205/projects/project2$ ./main
Please enter the expressions or variables.
Press q to quit
1+(2
There is an invalid input.
The input has been already cleared
a=2
1+b
There is an invalid input.
The input has been already cleared
sqrt(2)
There is an invalid input.
The input has been already cleared

```