# CS205 Project 2 - An Improved Calculator

**Name: Wang Haoyu**

**SID: 11911134**

# Part 1 - Analysis

## 1.1 Requirements

In this project, we aim to  implement a better calculator than that in Project 1. There are some basic requirements for this calculator.

> 1. When we run the program and input an express in a line, it can output the correct results. The operator precedence (order of operations) should be correct.
> 2. We can use parentheses to enforce the priorities.
> 3. We can define variables by ourselves and do operation through variable expression.
> 4. We can use some math functions in this calculator.
> 5. The calculator can support arbitrary precision.
> 6. If there are several files, we need to use CMake to manage the source files.

## 1.2 Solving Idea & Basic Steps

### 1.2.1 Generalization of the impletation procedure:

There are four main parts of the implementation shown below. The details are described in the explanations of each steps and functions.

**First Part:**

I implemented the basic requirement 1 and 2. The input is composed of number `0-9`, point `.` and operator signs `+-*/` and parentheses `()`. The input type is `string`. The input expression is usually in infix expression. To do calculation more conveniently for the computer, I changed the infix expression into postfix expression. And then do the calculation of postfix expression.

**Second Part:**

I implemented the basic requirement 3. The input will be tested before the program do the change that from infix expression to postfix expression. If the input contains variable definition, the terminal will start a new line until the user input a formula expression. After input the formula expression, the variables will be replaced by their corresponding values and we get an infix expression only composed of operators and numbers. Then do the operations I described in the First Part.

**Third Part:**

I implemented the basic requirement 4. If the input expression is satisfied the function format, the calculator will calculate and return the function value.

**Forth Part:**

I implemented the basic requirement 5. I wrote functions for calculating addition, subtraction and multiplication in arbitrary precision.

### 1.2.2 Details of Logic Steps

*The self-defined functions mentioned will be explained in 1.2.3, you can jump to the explanation by click the function name.*

**Step 1 - Test whether the expression is a number expression or a variable definition.**

I used a self-defined function isVariable to judge if the input is variable definition. If the expression is a variable definition, the function will return $true$. And the next step is to replace the variables with their corresponding values (step 2). If the expression is a number expression, the function will return $false$. And the next step is to change the infix expression into postfix expression (step 3).

**Step 2 - Replace variables with their corresponding values.**

If the expression is a variable definition, the program will enter a while loop, and the user can input next variable definition. The loop will end when the input is not a variable definition. After definitions, the next input is usually a calculating expression containing the variables.

In the former while loop, the variables will be stored in a vector with `char` type and the value of each variable will be stored in another vector with `string` type. The positions of each variable and its value in the vectors are one-one correspondence. Also, we get the number of variables through the loop.

After the loop, the program will deal with the last input, a calculation expression composed of the defined variables and operators and numbers. Go through each char of the input string, if find the char is a letter(variable), go through the vector storing variable to find the corresponding variable. Then replace the variable with its corresponding value in the expression.

After above operations, we get an infix expression without variables but only numbers and operators(including . And the program will do the step 3.

**Step 3 - Change infix expression into postfix expression.**

I used a self-defined function allnumber to judge if the input is a number calculating expression. If so, do this step.

I use a stack $s\_op$ to store the operators in the expression and a stack $s\_conv$ to store numbers and intermediate results.

Go through each char of the input string (number calculating expression):

- If the char is a number or point (they can be seen equivalent in this step), push into stack $s\_conv$.
- If the char is one of the four operators. Do a while loop to push the char into $s\_op$. In the while loop, the loop condition is `true`.
    - If there is no element in $s\_op$ or the priority of the char is higher than the top element of $s\_op$, push the char into $s\_op$ and push a space `' '` into $s\_conv$. Then break the while loop.
    - Else push a space into $s\_conv$. Then put the top element of $s\_op$ into $s\_conv$ and pop the top element of $s\_op$.
- If the char is a parenthese.

- If the char is `'('`, push the char into $s\_op$.
- If the char is `')'`, do a while loop. The loop condition is the top element of $s\_op$ is not a `'('`. In the loop, firstly push a space into $s\_conv$. Then put the top element of $s\_op$ into $s\_conv$ and pop the top element of $s\_op$. After the loop, pop the top element of $s\_op$ (delete `'('`).

After that, do a while loop with the condition is $s\_op$ is not empty. In the loop, firstly push a space into $s\_conv$. Then put the top element of $s\_op$ into $s\_conv$ and pop the top element of $s\_op$.

Now the composition of the elements in stack $s\_conv$ from bottom to up is the order of the postfix expression. We can begin to get the postfix expression.

Because stack is last in, first out, we can use $s\_op$ to reverse the order of the data in $s\_conv$. And then we can take out the data in $s\_op$ one by one and get the postfix expression.

P.S: The reason I push space into $s\_conv$ —— Because the stacks I defined store `char`. If the input number exceeds one digit, it cannot be distinguished later in generating postfix expression, so I add a space to distinguish it. There are four situations in total I push into space to confirm it can get correct postfix expression and make it more convenient to do the calculation of postfix expression.

**Step 4 - Do the calculation of postfix expression.**

The postfix expression we got is `string` type. I defined a function named ap_calculate to calculate and get the final result.

**Step 5 - Implementation of calculation for math functions.**

I defined a function mathfunction. If the input string contains the name of math function, the calculator will do the calculation for math functions. Firstly, get the string of name. Do the calculation according to the kind of math functions. The final result is `double` type and will be output.

### 1.2.3 Explanation of Functions Defined

**isVariable**

The input type is `string` and the return type is `bool`. If the input string contains equal sign `=` and the position of `=` is 1, then test the char at position 0. If the char is a letter (upper case and lower case are both accepted), then return $true$, else return $false$. For example, if the input is `x=2`, it will return $true$. The judgement of whether the char is a letter is also through a self-defined function isLetter.

**isLetter**

Input type is `char` and the return type is `bool`. If the char is a letter in the scope of A~Z or a~z, it will return $true$, else it will return $false$.

#### allnumber

Input type is `string` and return type is `bool`. Go through the input string, if each char is a number or an operator or a parenthese or a point, it will return $true$, else return $false$.

#### isOperator

Input type is `char` and return type is `bool`. If the input char is an operator (`+` or `-` or `*` or `/`), it will return $true$, else return $false$.

#### getPriority

Input type is `char` and return type is `int`. It is used to define the priority of the operator and patenthese. The priority level of `*` and `/` are the same and is higher than `+` and `-`. The priority level of `+` and `-` are the same and is higher than `(`. It defined the level of `*` and `/` as 3, the level of `+` and `-` as 2 and the level of `(` as 1. The default value of level is 0. It returns the value of the priority level.

#### mathfunction

Input type is `string` and return type is `bool`. In this function, I list nine math function name: $abs$ (get absolute value), $floor$ (round down), $ceil$ (round up), $sqrt$ (square root), $log$ (take logarithm of $e$), $exp$ (power of $e$), $sin$ , $cos$ , $tan$ . If find the name of one of the math functions in the input string, it will return $true$, else return $false$.

#### ap_calculate

The function is defined in $main.cpp$. The input type is string and the ouput type is string.

According to the former steps, in the postfix expression we got, every number and operator are seperated by space.

In this function, I used a stack $s\_cal$ to store strings to be calculated or intermediate results. The calculation will be done in a for loop. Go through the string from position 0:

Extract the substring (number or operator) and make judgements:

- If the substring is number, push the substring into $s\_cal$.
- If the substring is an operator, get the first top elements `num1` and the second top element `num2` in $s\_cal$, which will be calculated.
    - *If the operator is +*
        - If `num2` is a negative number (the first char is `'-'`), remove the negative sign and do subtraction of ($num1 - num2$) using self-defined function ap_sub. And push the calculation result into $s\_cal$.
        - Else if `num1` is a negative number (the first char is `'-'`), remove the negative sign and do subtraction of ($num2 - num1$) using self-defined function ap_sub. And push the calculation result into $s\_cal$.
        - Else do addition using self-defined function ap_add and push the calculation result into $s\_cal$.
    - *If the operator is -*

Do subtraction using self-defined function ap_sub and push the calculation result into $s\_cal$.

- *If the operator is **

  Do multiplication using self-defined function ap_mul and push the calculation result into $s\_cal$.

- *If the operator is /*

  Do division using self-defined function div and push the calculation result into $s\_cal$.

After the for loop, the top element of $s\_cal$ is the final calculation result. Return it.

#### ap_add

Inputs are two strings $a$ and $b$, and the output type is a string.

- Firstly, test whether they contain point. If at least one of them contains point, modify the format of the two strings before doing addition:

  If one of them contains point and the other does not, add string `".0"` after the string that does not contains point. Then compare the length of substring after the point position (equal to compare the length of fraction part), add zeros `'0'` to the string that having shorter fraction part until the length of fraction is equal to the length of the longer fraction part.

  After that, get the length of the fraction part and remove the point in the two strings.

- Secondly, do the addition in arbitrary precision,

  First of all, reverse the strings. Because we need to consider the carry bit and the length of strings is not always the same, so it is more convenient for us to deal with it from unit digit.

  Use an int array $res$ to store the intermediate reault.

  Go through each bit of the two strings, add the numbers in the same digit or get the number of one string if another does not have this digit. If the sum is not less than 10, the next digit of $res$ will be add 1 and this digit of $res$ is the remainder of the sum divide10.

```
for(int i=0; i<len; i++)
{
    if ( i < lena)
    {
       res[i] = res[i] + (str_a[i] - '0');
    }
    if (i < lenb)
    {
        res[i] = res[i] + (str_b[i] - '0');
    }

    if(res[i] >= 10)
    {
        res[i+1] = res[i+1] + res[i]/10;
        res[i] = res[i] % 10;
    }
}
```

- Thirdly, put every digit of $res$ into result in a reverse order.
- Finaly, if at least one of the original strings have point, extract the integer part ($result1$) and fraction part ($result2$) seperately according to the final length of fraction part we got. Then

add a point between them to get the final result. ($result = result1 + \text{"."} + result2$)

**ap_sub**

Inputs are two strings $a$ and $b$ ($a$ is the minuend), and the output type is a string $result$.

The method of dealing with strings if at least one of them contains point is almost the same as ap_add. However, there are some other operations to be done before removing the point:

To start with, seperate the modified strings into integer parts ($int\_a$ and $int\_b$) and fraction parts ($frac\_a$ and $frac\_b$).

For the integer parts:

- If the length of $int\_a$ is smaller than the length of $int\_b$, `result[0]='-'` and exchange the value of the two strings.
- If the length of $int\_a$ is equal to the length of $int\_b$:
  - If `int_a==int_b`, do for loop to compare the corresponding digit of the fraction parts one by one:
    - If `frac_b[i] < frac_a[i]`: break the loop.
    - If `frac_b[i] == frac_a[i]`: continue the loop(i++).
    - If `frac_b[i] > frac_a[i]`: `result[0]='-'` and exchange the value of the two strings.
  - Else, do for loop to compare the corresponding digit of the integer parts one by one:
    - If `int_b[i] < int_a[i]`: break the loop.
    - If `int_b[i] == int_a[i]`: continue the loop(i++).
    - If `int_b[i] > int_a[i]`: `result[0]='-'` and exchange the value of the two strings.

After that, remove the point in the strings and reverse the strings.

Second, If the initial strings do not have point, compare the corresponding digit of the strings one by one. The method is the same as the comparation of integer parts above.

Third, go through each bit of the two strings. Use an int array $res$ to store the intermediate reault. The subtraction process is similar to the addition process. The difference is that it does subtraction of the same digit of the strings. If $res[i] < 0$, the next element $res[i+1]$ will be subtracted with borrow. And `res[i] = res[i] + 10`.

```
for (int i = 0; i < len; i++)
{
    if (str_a[i]>='0' && str_a[i]<='9')
    {
        res[i] = res[i] + (str_a[i] - '0');
    }

    if(str_b[i]>='0' && str_b[i]<='9')
    {
        res[i] = res[i] - (str_b[i] - '0');
    }

    if(res[i] < 0)
    {
        res[i+1] = res[i+1] - 1;
```

```
        res[i] = res[i] + 10;
    }
}
```

Forth, if at least one of the original strings have point, get the result and add point in the same way as the last two steps in ap_add. In addition, for subtraction, we need to test if there are zeros in the front. I initialize an `int zero = 0`. If `result[0]!='-'`, go through result from `result[0]` and if `result[0]=='-'`, go through result from `result[1]`. Get the number of zeros in the front and erase them.

```
if (result[0]!='-')
{
    for (int i = 0; i < p; i++)
    {
        if (result[i]=='0' && result[i+1]!='.')
        {
            zero = zero + 1;
            continue;
        }
        else break;
    }
}
else
{
    for (int i = 1; i < p; i++)
    {
        if (result[i]=='0' && result[i+1]!='.')
        {
            zero = zero + 1;
            continue;
        }
        else break;
    }
}

if (zero>0)
{
    if (result[0]=='-')
        result = result.erase(1,zero);
    else
        result = result.erase(0,zero);
}
```

Fifth, if the original strings do not have point, test the $res$ from the last position directly, if it is zero, subtract the length of $res$ and then get the result.

```
if ((str_a0.find('.')==string::npos) && (str_b0.find('.')==string::npos))
{
    while(res[len-1]==0 && len>1)
    {
        len = len - 1;
    }
    if (len==1)
    {
        result = result + to_string(res[0]);
        return result;
```

```
    }
    for (int i = len-1; i >= 0; i--)
    {
        result = result + to_string(res[i]);
    }
}
return result;
```

### ap_mul

Inputs are two strings $a$ and $b$, and the output type is a string.

Initialize `string str_a=a; string str_b=b`. If the string contains negative sign, remove the negative sign and get new $str\_a$ and $str\_b$. Note that if only one of the string has negtive sign, add a negative sign before the final result in the last. If both of them have negative sign, the multiplication result is supposed to be positive, so we do not need to add a negative sign.

If the length of string $str\_a$ is shorter than the length of string $str\_b$, do `ap_mul(b,a)`. Thus we can confirm the first string is longer.

Firstly, test whether they contain point. If at least one of them contains point, modify the format of the two strings before doing addition:

If one of them contains point and the other does not, add string `".0"` after the string that does not contains point. After that, add the length of the fraction parts of the two strings, which will be the fraction length of the fianl result. Then remove the point in each string.

Secondly, reverse the strings. Use a vector $re$ to store `int` type elements, the size of $re$ is the sum of lengths of the two strings.

The multiplication method is similar to the addition. The difference is that the product may be larger than 20, the mod ( `re[i]/10` may be larger than 1). We need to make every digit be seperate elements.

```
for(int i=0; i<lena; i++)
{
    for(int j=0; j<lenb; j++)
    {
        re[i+j] = re[i+j] + (str_a[i]-'0')*(str_b[j]-'0');
    }
}
for(int i=0; i<re.size()-1; i++)
{
    re[i+1] = re[i+1] + re[i]/10;
    re[i] = re[i] % 10;
}
int t = re.back();
re.back() = t%10;
t = t / 10;
while(t)
{
    re.push_back(t%10);
    t = t / 10;
}
while(re.size() > 1 && re.back() == 0)
    re.pop_back();
```

Finally, the method to get the result and the insert of point is the same as they in ap_add. But pay attention to the negative sign.

```
if (a[0]=='-' && b[0]!='-')
{
    result = "-" + result;
}

if (a[0]!='-' && b[0]=='-')
{
    result = "-" + result;
}
```

**div**

Inputs are two strings $a$ and $b$, and the output type is a string.

Transform the two input strings into `double` type $da$ and $db$. Do division, result in `double` type is equal to $db/da$.

Then transform the result from `double` type to `string` type and return it.

# Part 2 - Code

I also uploaded the code onto Github. However, it is my first time to do this, I am afraid of happening something wrong, so I pasted my code in the report, too. You can find in the appendix of the report. (Click here → Entire Code)

# Part 3 - Result & Verification

## 3.1 Functions of the Improved Calculator

My final program can do these functions below:

1. Run the program and input an expression in a line, it can output the correct results. The operator precedence (order of operations) can be correctly used. → Test case1
2. The user can use parentheses to enforce the priorities. → Test case2
3. The user can define variables and it can do operation through variable expression. → Test case3
4. The user can use some math functions. → Test case4
5. It supports arbitrary precision for addition, subtraction and multiplication. → Test case5
6. The calculator can do addition, subtraction and multiplication in arbitrary precision not only for two numbers, but also for the formula expression or variable definition. → Test case6
7. It supports both integer and floating-point number calculations. → Test case7

## 3.2 The Limits of the Improved Calculator

It should be noted that there are some restrictions for the input format. If the input expression is not satisfied with the format, the calculator may not do right executions. However, it is not often to meet the problem. Here are the restrictions:

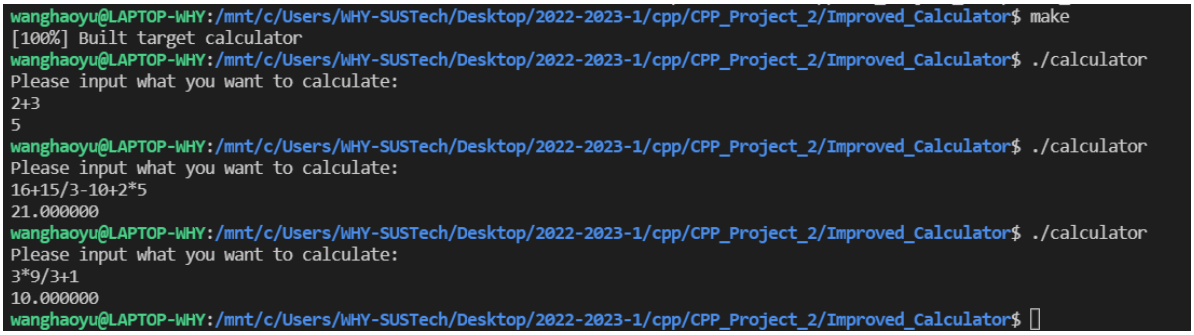
- The variable defined by ourselves can only be $a$ to $z$ or $A$ to $Z$.
- You are not supposed to input single negative number like `-2+3`, you are supposed to input as `3-2` or `0-2+3`.
- The input expression to calculate must be in infix expression.
- The math function can only be used to calculate the function value of one number. It cannot be used in a formula expression. And there are only nine kinds of math functions can be used: abs, floor, ceil, sqrt, log, exp, sin, cos, tan.
- The calculator cannot recognize strange input. If the input is not satisfied the normal format, the calculator cannot work or return `Please input your expression in an appropriate format.`

## 3.3 Verification of the Functions

I wrote some test cases to verify the function of the improved calculator.

**Test case #1**

*Run the program and input an expression in a line, it can output the correct results. The operator precedence (order of operations) can be correctly used.*




**Test case #2**

*The user can use parentheses to enforce the priorities.*


**Test case #3**

*The user can define variables and it can do operation through variable expression.*

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
x=2
y=3
x+y
5
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
x=1
y=2
z=3
x+y*2-z+5*(y+z)
27
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
a=1
b=2
c=3
d=4
e=5
a*b*c*d*e-a-b-c-d-e
105
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

### Test case #4

*The user can use some math functions.*

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
abs(-999)
999
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
sqrt(144)
12
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
floor(1.9)
1
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
ceil(2.3)
3
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
log(1)
0
```

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
exp(2)
7.38906
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
sin(1)
0.841471
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
cos(0)
1
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
tan(2)
-2.18504
```

### Test case #5

*It supports arbitrary precision for addition, subtraction and multiplication.*

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
19999999999999999999999999999999999999999999999999999999999+2*10000000000000000000000000000000000000000000000000000000000
20000019999999999999999999999999999999999999999999999999999
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
200000000000000000000000000000000000000000000000000000000000000000000000*200000000000000000000000000000000000000000000000000000000001
400000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
99999999999999999999999999999999999999999999999999-888888888888888888888888888888888888888888888888888888888888
-888788888888888888888888888888888888888888888888889
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
99999999999999999999999999999999999999999999999999999999999999999999999999999-77777777777777777777777777777777777777777777
99999999999999999222222222222222222222222222222222222222222222222
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
99999999999999999999999999999999999999999999999999999999999999999999999*0.00000000000000000000000000000000001+2
10000000000000000000000000001.99999999999999999999999999999999999990
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
9.99999999999999999999999999999999*0.1+12345678901234567890
12345678901234567890.99999999999999999999999999999999
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

**Test case #6**

*The calculator can do addition, subtraction and multiplication in arbitrary precision not only for two numbers, but also for the formula expression or variable definition.*

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
x=1.11111111111111111111111111111111111111111111111111111111111
y=99
z=1
x+y*(z+3*3)-0.11111111111111111111111111111111111111111
991.00000000000000000000000000000000000000000000011111111111111111
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
99999999999999999999999999999999999999999999999999999999999999*99999999999999999999999999999999999999999999999999*0.00000000000000000000000001-99999999999999999999999999999999
9999999999999999999999999999999999999999999999999999999898999999999000000000000000000000000001.0000000000000000000000000010
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

**Test case #7**

*It supports both integer and floating-point number calculations.*

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
8.8/2+(9-2*5)*0.4
4.000000
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
abs(-0.999)
0.999
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ ./calculator
Please input what you want to calculate:
x=2
y=3
(x+0.1)*2+y*0.2
4.80
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

# Part 4 - Difficulties & Solutions

**Difficulty 1**

If I follow my first project to determine what each character is one by one, and then calculate the result step by step from the result of the determination is very troublesome.

Solution:

Create a stack, convert the input midfix expression to a suffix expression, and then perform the suffix expression calculation

**Difficulty 2**

The infix-to-postfix function is initially declared in calculator.hpp and defined in calculator.cpp, but for functions that return a string type, there is an error when called in main.cpp

Solution:

Write the infix-to-postfix process in main.cpp to apply it directly, not as a function.

**Difficulty 3**

When using characters into the stack, numbers with more than one digit are broken into the stack one by one.

Solution:

Add spaces to separate them

**Difficulty 4**

In the variable arithmetic part, at the beginning, I used the following string to store the variables and variable values, and then substituted the variables in order to find that if the variable entered later appeared first, the corresponding value would be disordered.

Solution:

Use two vectors to store variables and variable values separately, the position are one-to-one corresponce. And replace variables with their value correspondingly in the expression later.

```
    string str_ncal = str_vcal; //代入数值后的计算式（不含变量）
    int istart = 0;

    for (int i = 0; i < str_vcal.length(); i++)
    {
        if (isLetter(str_vcal[i]))
        {
            for (int j = 0; j < varN; j++)
            {
                if (var_store[j]==str_vcal[i])
                {
                    str_ncal.replace(str_ncal.find(str_vcal[i]), 1, value_store[j]);
                }
                else continue;
            }
        }
        else continue;
    }

    str = str_ncal;
}
```

**Some References**

(46条消息) 使用栈计算后缀表达式的计算-----C++实现（详解）好好学习。天天编程的博客-*CSDN*博客利用栈计算后缀表达式

(46条消息) 中缀表达式转后缀表达式（c++）_乌药ice的博客-CSDN博客

# Appendix - Entire Code

**calculator.hpp**

```hpp
#pragma once
#include <string>
#include <string.h>

//函数声明1：判断是否是操作符
bool isOperator(char ch);

//函数声明2：获取优先级
int getPriority(char ch);

//函数声明3：后缀表达式计算
//double calculate(const std::string str);

//函数声明4：判断是否为纯数字计算
bool allnumber(const std::string str);

//函数声明5：判断是否用到数学函数
bool mathfunction(const std::string str);

//函数声明6：判断是否是字母
bool isLetter(char ch);

//函数声明7：判断是否是变量运算
bool isVariable(const std::string str);
```

**calculator.cpp**

```cpp
#include "calculator.hpp"
#include <iostream>
#include <string.h>
#include <stack>
#include <math.h>
using namespace std;

//函数定义1：判断是否是操作符
bool isOperator(char ch)
{
    if(ch=='+' || ch=='-' || ch=='*' || ch=='/')
        return true;
    else
        return false;
}

//函数定义2：获取优先级
int getPriority(char ch)
{
    int level = 0;
    switch (ch)
    {
    case '(':
        level = 1;
        break;
```

```
    case '+':
        level = 2;
        break;
    case '-':
        level = 2;
        break;
    case '*':
        level = 3;
        break;
    case '/':
        level = 3;
        break;
    default:
        level = 0;
        break;
    }
    return level;
}

//函数定义3：后缀表达式的计算

//函数定义4：判断是否为纯数字计算
bool allnumber(const std::string str)
{
    int a = 0;
    for (int i = 0; i < str.length(); i++)
    {
        char ch = str[i];
        if ((ch>=48 && ch<=57) || isOperator(ch) || ch=='.' || ch=='(' ||
ch==')')
            a = a + 1;
        else break;
    }
    if (a==str.length())
        return true;
    else
        return false;
}

//函数定义5：判断是否用到数学函数
bool mathfunction(const std::string str)
{
    string s1 = "abs";  //绝对值
    string s2 = "floor";//向下取整
    string s3 = "ceil"; //向上取整
    string s4 = "sqrt"; //平方根
    string s5 = "log";  //自然对数，e为底
    string s6 = "exp";  //e的多少次方
    string s7 = "sin";  //sin
    string s8 = "cos";  //cos
    string s9 = "tan";  //tan

    if (str.find(s1)==string::npos && str.find(s2)==string::npos &&
str.find(s3)==string::npos
    && str.find(s4)==string::npos && str.find(s5)==string::npos &&
str.find(s6)==string::npos
    && str.find(s7)==string::npos && str.find(s8)==string::npos &&
str.find(s9)==string::npos)
```

```
    {
        return false;
    }
    else
        return true;
}

//函数定义6：判断是否是字母
bool isLetter(char ch)
{
    if ((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
    {
        return true;
    }
    else
        return false;
}

//函数定义7：判断是否是变量运算
bool isVariable(const std::string str)
{
    if (str.find_first_of('=')!=string::npos)
    {
        int equal = str.find_first_of('=');
        if (equal==1)
        {
            char v = str[0];
            if (isLetter(v))
            {
                return true;
            }
            else
                return false;
        }
        else
            return false;
    }
    else
        return false;
}
```

**main.cpp**

```
#include "calculator.hpp"
#include <iostream>
#include <string>
#include <string.h>
#include <stack>
#include <math.h>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;

const int maxn = 100005;

string ap_add(const std::string a, const std::string b)
```

```
{
    string str_a = a;
    string str_b = b;
    string str_a0 = a;
    string str_b0 = b;
    string frac_a;
    string frac_b;
    int pointa = 0;
    int pointb = 0;
    int fraclen = 0;

    if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
    {
        if (str_a.find('.')!=string::npos)
        {
            pointa = str_a.find_first_of('.');
            frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
        }
        else
        {
            str_a = str_a + ".0";
            pointa = str_a.find_first_of('.');
            frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
        }

        if (str_b.find('.')!=string::npos)
        {
            pointb = str_b.find_first_of('.');
            frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
        }
        else
        {
            str_b = str_b + ".0";
            pointb = str_b.find_first_of('.');
            frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
        }

        if (str_a.find('.')!=string::npos && str_b.find('.')!=string::npos)
        {
            if (frac_a.length() > frac_b.length())
            {
                fraclen = frac_a.length();
                for (int i = 1; i <= (frac_a.length()-frac_b.length());
i++)
                {
                    str_b = str_b + '0';
                }
                pointb = str_b.find_first_of('.');
                frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
            }
            else if (frac_b.length() > frac_a.length())
            {
                fraclen = frac_b.length();
                for (int i = 1; i <= (frac_b.length()-frac_a.length());
i++)
                {
                    str_a = str_a + '0';
```

```
                }
                pointa = str_a.find_first_of('.');
                frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
            }
            else fraclen = frac_a.length();

            str_a = str_a.erase(str_a.find('.'),1);
            str_b = str_b.erase(str_b.find('.'),1);
        }
    }

    int res[maxn] = {0};
    int lena = str_a.length();
    int lenb = str_b.length();
    reverse(str_a.begin(), str_a.end()); //翻转数组
    reverse(str_b.begin(), str_b.end());
    int len = max(lena, lenb);
    for(int i=0; i<len; i++)
    {
        if ( i < lena)
        {
           res[i] = res[i] + (str_a[i] - '0');
        }
        if (i < lenb)
        {
            res[i] = res[i] + (str_b[i] - '0');
        }

        if(res[i] >= 10)
        {
            res[i+1] = res[i+1] + res[i]/10;
            res[i] = res[i] % 10;
        }
    }

    string result = {};

    if(res[len] > 0)
    {
        result = result + to_string(res[len]);
    }
    for(int i=len-1; i>=0; i--)
    {
        result = result + to_string(res[i]);
    }

    if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
    {
        string result1 = result.substr(0, result.length()-fraclen);
        string result2 = result.substr(result.length()-fraclen, fraclen);
        result = result1 + "." + result2;
    }

    return result;
}

string ap_sub(const std::string a, const std::string b)
```

```
{
    string str_a = a;
    string str_b = b;
    string str_a0 = a;
    string str_b0 = b;
    string frac_a;
    string frac_b;
    string int_a;
    string int_b;
    int pointa = 0;
    int pointb = 0;
    int fraclen = 0;
    string result = {};

    if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))//a-b
    {
        if (str_a.find('.')!=string::npos)
        {
            pointa = str_a.find_first_of('.');
            frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
        }
        else
        {
            str_a = str_a + ".0";
            pointa = str_a.find_first_of('.');
            frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
        }

        if (str_b.find('.')!=string::npos)
        {
            pointb = str_b.find_first_of('.');
            frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
        }
        else
        {
            str_b = str_b + ".0";
            pointb = str_b.find_first_of('.');
            frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
        }

        if (str_a.find('.')!=string::npos && str_b.find('.')!=string::npos)
        {
            if (frac_a.length() > frac_b.length())
            {
                fraclen = frac_a.length();
                for (int i = 1; i <= (frac_a.length()-frac_b.length());
i++)
                {
                    str_b = str_b + '0';
                }
                pointb = str_b.find_first_of('.');
                frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
            }
            else if (frac_b.length() > frac_a.length())
            {
                fraclen = frac_b.length();
```

```
                for (int i = 1; i <= (frac_b.length()-frac_a.length());
i++)
                {
                    str_a = str_a + '0';
                }
                pointa = str_a.find_first_of('.');
                frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
            }
            else fraclen = frac_a.length();

            int_a = str_a.substr(0, pointa);
            int_b = str_b.substr(0, pointb);

            if(int_a.length() < int_b.length())
            {
                result = result + "-";
                swap(str_a, str_b);
            }

            if(int_a.length() == int_b.length())
            {
                if (int_a==int_b)
                {
                    for(int i = 0; i < fraclen; i++)
                    {
                        if(frac_b[i] < frac_a[i]) break;
                        else if(frac_b[i] == frac_a[i]) continue;
                        else if(frac_b[i] > frac_a[i])
                        {
                            result = result + "-";
                            swap(str_a, str_b);
                            break;
                        }
                    }
                }
                else
                {
                    for(int i = 0; i<int_a.length(); i++)
                    {
                        if(int_b[i] < int_a[i]) break;
                        else if(int_b[i] == int_a[i]) continue;
                        else if(int_b[i] > int_a[i])
                        {
                            result = result + "-";
                            swap(str_a, str_b);
                            break;
                        }
                    }
                }
            }

            str_a = str_a.erase(str_a.find('.'),1);
            str_b = str_b.erase(str_b.find('.'),1);
        }
    }

    int res[maxn] = {0};
    int lena = str_a.length();
```

```
    int lenb = str_b.length();
    reverse(str_a.begin(), str_a.end()); //翻转数组
    reverse(str_b.begin(), str_b.end());
    int len = max(lena, lenb);

    if((str_a0.find('.')==string::npos) &&
(str_b0.find('.')==string::npos))
    {
        if(lena < lenb)
        {
            result = result + "-";
            swap(str_a, str_b);
        }

        if(lena == lenb)
        {
            for(int i = lena-1; i>=0; i--)
            {
                if(str_b[i] < str_a[i]) break;
                else if(str_b[i] == str_a[i]) continue;
                else if(str_b[i] > str_a[i])
                {
                    result = result + "-";
                    swap(str_a, str_b);
                    break;
                }
            }
        }
    }

    for (int i = 0; i < len; i++)
    {
        if (str_a[i]>='0' && str_a[i]<='9')
        {
            res[i] = res[i] + (str_a[i] - '0');
        }

        if(str_b[i]>='0' && str_b[i]<='9')
        {
            res[i] = res[i] - (str_b[i] - '0');
        }

        if(res[i] < 0)
        {
            res[i+1] = res[i+1] - 1;
            res[i] = res[i] + 10;
        }
    }


    if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
    {
        for (int i = len-1; i >= 0; i--)
        {
            result = result + to_string(res[i]);
        }
        string result1 = result.substr(0, result.length()-fraclen);
```

```
295         string result2 = result.substr(result.length()-fraclen, fraclen);
296         result = result1 + "." + result2;
297         int p = result.find_first_of('.');
298         int zero = 0;
299 
300         if (result[0]!='-')
301         {
302             for (int i = 0; i < p; i++)
303             {
304                 if (result[i]=='0' && result[i+1]!='.')
305                 {
306                     zero = zero + 1;
307                     continue;
308                 }
309                 else break;
310             }
311         }
312         else
313         {
314             for (int i = 1; i < p; i++)
315             {
316                 if (result[i]=='0' && result[i+1]!='.')
317                 {
318                     zero = zero + 1;
319                     continue;
320                 }
321                 else break;
322             }
323         }
324 
325         if (zero>0)
326         {
327             if (result[0]=='-')
328                 result = result.erase(1,zero);
329             else
330                 result = result.erase(0,zero);
331         }
332     }
333 
334     if ((str_a0.find('.')==string::npos) &&
    (str_b0.find('.')==string::npos))
335     {
336         while(res[len-1]==0 && len>1)
337         {
338             len = len - 1;
339         }
340         if (len==1)
341         {
342             result = result + to_string(res[0]);
343             return result;
344         }
345         for (int i = len-1; i >= 0; i--)
346         {
347             result = result + to_string(res[i]);
348         }
349     }
350     return result;
351 }
```

```
352
353 string ap_mul(const std::string a, const std::string b)
354 {
355     string str_a = a;
356     string str_b = b;
357
358     if (a[0]=='-')
359     {
360         str_a.erase(0, 1);
361     }
362     if (b[0]=='-')
363     {
364         str_b.erase(0, 1);
365     }
366
367     if(str_a.length()<str_b.length())
368         return ap_mul(b, a);
369
370     int fraclen = 0;
371     if (a.find('.')!=string::npos || b.find('.')!=string::npos)
372     {
373         if (a.find('.')==string::npos)
374         {
375             str_a = str_a + ".0";
376         }
377
378         if (b.find('.')==string::npos)
379         {
380             str_b = str_b + ".0";
381         }
382
383         if (str_a.find('.')!=string::npos || str_b.find('.')!=string::npos)
384         {
385             int pointa = str_a.find_first_of('.');
386             int pointb = str_b.find_first_of('.');
387             string frac_a = str_a.substr(pointa+1, str_a.length()-pointa-
    1);
388             string frac_b = str_b.substr(pointb+1, str_b.length()-pointb-
    1);
389
390             fraclen = frac_a.length() + frac_b.length();
391
392             str_a = str_a.erase(pointa, 1);
393             str_b = str_b.erase(pointb, 1);
394         }
395     }
396
397     reverse(str_a.begin(), str_a.end());
398     reverse(str_b.begin(), str_b.end());
399     int lena = str_a.length();
400     int lenb = str_b.length();
401     vector<int> re(lena+lenb, 0);
402     string result;
403
404     for(int i=0; i<lena; i++)
405     {
406         for(int j=0; j<lenb; j++)
407         {
```

```
            re[i+j] = re[i+j] + (str_a[i]-'0')*(str_b[j]-'0');
        }
    }
    for(int i=0; i<re.size()-1; i++)
    {
        re[i+1] = re[i+1] + re[i]/10;
        re[i] = re[i] % 10;
    }
    int t = re.back();
    re.back() = t%10;
    t = t / 10;
    while(t)
    {
        re.push_back(t%10);
        t = t / 10;
    }
    while(re.size() > 1 && re.back() == 0)
        re.pop_back();

    for (int i = re.size()-1; i>=0; i--)
    {
        result = result + to_string(re[i]);
    }

    if (a.find('.')!=string::npos || b.find('.')!=string::npos)
    {
        string re_int = result.substr(0, result.length()-fraclen);
        string re_frac = result.substr(result.length()-fraclen, fraclen);
        result = re_int + "." + re_frac;
    }


    if (a[0]=='-' && b[0]!='-')
    {
        result = "-" + result;
    }

    if (a[0]!='-' && b[0]=='-')
    {
        result = "-" + result;
    }

    return result;
}

string div(const std::string a, const std::string b)
{
    double da = stod(a);
    double db = stod(b);
    double re = db/da;
    string res = to_string(re);
    return res;
}

string ap_calculate(const std::string str)
{
    stack<std::string> s_cal;
    string space = " ";
```

```
466     string s = "0123456789+-*/";
467 
468     int start = 0;
469     int end = 0;
470     string num;
471     string num1;
472     string num2;
473 
474     for (int i = 0; i < str.length(); )
475     {
476         start = str.find_first_of(s, i);
477         end = str.find_first_of(space, i);
478 
479         if (end == string::npos) //处理最后一个符号没有空格的情况
480         {
481             end = str.length();
482         }
483 
484         string tempstr = str.substr(start, (end-start));
485         if ((tempstr=="+") || (tempstr=="-") || (tempstr=="*") ||
(tempstr=="/"))
486         {
487             num1 = s_cal.top();
488             s_cal.pop();
489             num2 = s_cal.top();
490             s_cal.pop();
491 
492             if (tempstr == "+")
493             {
494                 if (num2[0]=='-')
495                 {
496                     num2.erase(0, 1);
497                     num = ap_sub(num1, num2);
498                     s_cal.push(num);
499                 }
500                 else if (num1[0]=='-')
501                 {
502                     num1.erase(0, 1);
503                     num = ap_sub(num2, num1);
504                     s_cal.push(num);
505                 }
506 
507                 else
508                 {
509                     num = ap_add(num1, num2);
510                     s_cal.push(num);
511                 }
512             }
513             else if (tempstr == "-")
514             {
515                 num = ap_sub(num2, num1);
516                 //num = to_string(stod(num2) - stod(num1));
517                 s_cal.push(num);
518             }
519             else if (tempstr == "*")
520             {
521                 num = ap_mul(num1, num2);
522                 s_cal.push(num);
```

```
            }
            else
            {
                num = div(num1, num2);
                s_cal.push(num);
            }
        }
        else
        {
            //double tempnum = stod(tempstr);
            s_cal.push(tempstr);
        }
        i = end + 1;
    }

    return s_cal.top();
}

int main()
{
    cout << "Please input what you want to calculate:" << endl;
    char ch[maxn] = {};
    cin.getline(ch, maxn);

    string str = ch;
    int varN = 0;

    if (isVariable(str)) //variable,转换成纯数字计算式之后，中缀->后缀->后缀表达式计算
    {
        varN = 1;
        string str_v;
        string str_vcal; //含变量的计算式
        string str_value = str.substr(2, str.length()-2); //储存变量值
        vector<char> var_store;
        vector<string> value_store;

        var_store.push_back(str[0]);
        value_store.push_back(str_value);

        while (true)
        {
            char ch_v[maxn] = {};
            cin.getline(ch_v, maxn);
            str_v = ch_v;

            if (isVariable(str_v))
            {
                varN = varN + 1;
                string value = str_v.substr(2, str_v.length()-2);
                var_store.push_back(str_v[0]);
                value_store.push_back(value);
                continue;
            }
            else
            {
                str_vcal = str_v;
                break;
```

```
            }
        }

        string str_ncal = str_vcal; //代入数值后的计算式（不含变量）
        int istart = 0;

        for (int i = 0; i < str_vcal.length(); i++)
        {
            if (isLetter(str_vcal[i]))
            {
                for (int j = 0; j < varN; j++)
                {
                    if (var_store[j]==str_vcal[i])
                    {
                        str_ncal.replace(str_ncal.find(str_vcal[i]), 1, value_store[j]);
                    }
                    else continue;
                }
            }
            else continue;
        }

        str = str_ncal;
    }


    if (allnumber(str)) //纯算数计算
    {
        //获取字符串长度
        int len = str.length();
        //初始化stack，s_op存运算符，s_conv存中间结果，最后s_conv逆序输出
        stack<char> s_op;
        stack<char> s_conv;
        //输出目标，后缀表达式
        char ch_conv[len] = {};
        //遍历字符串，按优先级放入中间结果栈
        for (int i = 0; i < len; i++)
        {
            if ((str[i]>=48 && str[i]<=57) || str[i]=='.') //数字-直接放入栈
            {
                s_conv.push(str[i]);
            }
            else if (isOperator(str[i])) //操作符-判断优先级
            {

                while (true)
                {
                    if (s_op.empty() || getPriority(str[i]) > getPriority(s_op.top())) //操作符栈为空或优先级更大时放入，包括栈顶是'('的情况
                    {
                        s_op.push(str[i]);
                        s_conv.push(' '); //如果输入的数超过个位无法区分，加空格来区分
                        break;
                    }
                    else //优先级小或相等的时候拿出栈顶，再进行循环
                    {
```

```
                        s_conv.push(' ');
                        char c1 = s_op.top();
                        s_op.pop();
                        s_conv.push(c1);
                    }
                }
            }
            else //括号
            {
                if (str[i]=='(')
                {
                    s_op.push(str[i]);
                }
                else //str[i]=')'的情况
                {
                    while (s_op.top()!='(')
                    {
                        s_conv.push(' ');
                        char c2 = s_op.top();
                        s_op.pop();
                        s_conv.push(c2);
                    }
                    s_op.pop(); //丢掉'('
                }
            }
        }

        while (s_op.empty()==0)
        {
            s_conv.push(' ');
            char c3 = s_op.top();
            s_op.pop();
            s_conv.push(c3);
        }

        //此时栈s_conv从底部往上为后缀表达式顺序，要按顺序赋给ch_conv
        //可以利用s_op将s_conv中数据的顺序颠倒，再赋给ch_conv

        while (s_conv.empty()==0)
        {
            char c4 = s_conv.top();
            s_conv.pop();
            s_op.push(c4);
        }

        int ci = 0;
        while (s_op.empty()==0)
        {
            ch_conv[ci] = s_op.top();
            ci = ci + 1;
            s_op.pop();
        }

        ch_conv[ci] = '\0';
        //至此，获得了后缀表达式

        //后缀表达式计算,先将char*转成string类型
        string str_cal;
```

```
693         str_cal = ch_conv;
694 
695         string result = ap_calculate(str_cal);
696         //double result = calculate(str_cal);
697 
698         cout << result << endl;
699     }
700     else if (mathfunction(str)) //简单的数学函数计算
701     {
702         string str_fun;
703         string str_num;
704         int n1 = str.find_first_of('(');
705         int n2 = str.find_first_of(')');
706 
707         str_fun = str.substr(0, n1);
708         str_num = str.substr(n1+1, n2-n1-1);
709 
710         double n = stod(str_num);
711         double result = 0;
712         if (str_fun == "abs")
713         {
714             result = abs(n);
715         }
716         else if (str_fun == "floor")
717         {
718             result = floor(n);
719         }
720         else if (str_fun == "ceil")
721         {
722             result = ceil(n);
723         }
724         else if (str_fun == "sqrt")
725         {
726             result = sqrt(n);
727         }
728         else if (str_fun == "log")
729         {
730             result = log(n);
731         }
732         else if (str_fun == "exp")
733         {
734             result = exp(n);
735         }
736         else if (str_fun == "sin")
737         {
738             result = sin(n);
739         }
740         else if (str_fun == "cos")
741         {
742             result = cos(n);
743         }
744         else if (str_fun == "tan")
745         {
746             result = tan(n);
747         }
748 
749         cout << result << endl;
750     }
```

```
751     else
752     {
753         cout << "Please input your expression in an appropriate format." <<
    endl;
754     }
755 
756     return 0;
757 
758 }
```

**CMakeLists.txt**

```
1 cmake_minimum_required(VERSION 3.10)
2 
3 project(calculator)
4 
5 aux_source_directory(. DIR_SRCS)
6 
7 add_executable(calculator ${DIR_SRCS})
```