

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 parentheses.

- 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 parentheses 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 parentheses. 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 output type is string.

According to the former steps, in the postfix expression we got, every number and operator are separated 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 contain 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 result.

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 divide 10.

```
1  for(int i=0; i<len; i++)
2  {
3      if ( i < lena)
4      {
5          res[i] = res[i] + (str_a[i] - '0');
6      }
7      if (i < lenb)
8      {
9          res[i] = res[i] + (str_b[i] - '0');
10     }
11
12     if(res[i] >= 10)
13     {
14         res[i+1] = res[i+1] + res[i]/10;
15         res[i] = res[i] % 10;
16     }
17 }
```

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

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

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, separate 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 comparison of integer parts above.

Third, go through each bit of the two strings. Use an int array res to store the intermediate result. 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`.

```
1  for (int i = 0; i < len; i++)
2  {
3      if (str_a[i] >= '0' && str_a[i] <= '9')
4      {
5          res[i] = res[i] + (str_a[i] - '0');
6      }
7
8      if(str_b[i] >= '0' && str_b[i] <= '9')
9      {
10         res[i] = res[i] - (str_b[i] - '0');
11     }
12
13     if(res[i] < 0)
14     {
15         res[i+1] = res[i+1] - 1;
```

```

16         res[i] = res[i] + 10;
17     }
18 }

```

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.

```

1  if (result[0]!='-')
2  {
3      for (int i = 0; i < p; i++)
4      {
5          if (result[i]=='0' && result[i+1]!='.')
6          {
7              zero = zero + 1;
8              continue;
9          }
10         else break;
11     }
12 }
13 else
14 {
15     for (int i = 1; i < p; i++)
16     {
17         if (result[i]=='0' && result[i+1]!='.')
18         {
19             zero = zero + 1;
20             continue;
21         }
22         else break;
23     }
24 }
25
26 if (zero>0)
27 {
28     if (result[0]=='-')
29         result = result.erase(1,zero);
30     else
31         result = result.erase(0,zero);
32 }

```

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.

```

1  if ((str_a0.find('.')==string::npos) && (str_b0.find('.')==string::npos))
2  {
3      while(res[len-1]==0 && len>1)
4      {
5          len = len - 1;
6      }
7      if (len==1)
8      {
9          result = result + to_string(res[0]);
10         return result;

```

```

11     }
12     for (int i = len-1; i >= 0; i--)
13     {
14         result = result + to_string(res[i]);
15     }
16 }
17 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 negative 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 contain point. After that, add the length of the fraction parts of the two strings, which will be the fraction length of the final 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 separate elements.

```

1  for(int i=0; i<lena; i++)
2  {
3      for(int j=0; j<lenb; j++)
4      {
5          re[i+j] = re[i+j] + (str_a[i]-'0')*(str_b[j]-'0');
6      }
7  }
8  for(int i=0; i<re.size()-1; i++)
9  {
10     re[i+1] = re[i+1] + re[i]/10;
11     re[i] = re[i] % 10;
12 }
13 int t = re.back();
14 re.back() = t%10;
15 t = t / 10;
16 while(t)
17 {
18     re.push_back(t%10);
19     t = t / 10;
20 }
21 while(re.size() > 1 && re.back() == 0)
22     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.

```
1  if (a[0]=='-' && b[0]!='-')
2  {
3      result = "-" + result;
4  }
5
6  if (a[0]!='-' && b[0]=='-')
7  {
8      result = "-" + result;
9  }
```

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 have uploaded the code onto Github.

[CS205-Projects-2022Fall/Project 2 at main · Micromalusx/CS205-Projects-2022Fall \(github.com\)](https://github.com/Micromalusx/CS205-Projects-2022Fall/blob/main/Project%202%20at%20main)

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.

```
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$ make
[100%] Built target calculator
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:
2+3
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:
16+15/3-10+2*5
21.000000
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:
3*9/3+1
10.000000
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

Test case #2

The user can use parentheses to enforce the priorities.

```
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:
3*(4+5)-7
20
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:
16+9-2*(2+1*(3+5))
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:
16+(9-2)*(2+1*(3+5))
86
wanghaoyu@LAPTOP-WHY:/mnt/c/Users/WHY-SUSTech/Desktop/2022-2023-1/cpp/CPP_Project_2/Improved_Calculator$
```

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.

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 correspondence. 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 irstart = 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

```
1  #pragma once
2  #include <string>
3  #include <string.h>
4
5  //函数声明1: 判断是否是操作符
6  bool isOperator(char ch);
7
8  //函数声明2: 获取优先级
9  int getPriority(char ch);
10
11 //函数声明3: 后缀表达式计算
12 //double calculate(const std::string str);
13
14 //函数声明4: 判断是否为纯数字计算
15 bool allnumber(const std::string str);
16
17 //函数声明5: 判断是否用到数学函数
18 bool mathfunction(const std::string str);
19
20 //函数声明6: 判断是否是字母
21 bool isLetter(char ch);
22
23 //函数声明7: 判断是否是变量运算
24 bool isVariable(const std::string str);
```

calculator.cpp

```
1  #include "calculator.hpp"
2  #include <iostream>
3  #include <string.h>
4  #include <stack>
5  #include <math.h>
6  using namespace std;
7
8  //函数定义1: 判断是否是操作符
9  bool isOperator(char ch)
10 {
11     if(ch=='+' || ch=='-' || ch=='*' || ch=='/')
12         return true;
13     else
14         return false;
15 }
16
17 //函数定义2: 获取优先级
18 int getPriority(char ch)
19 {
20     int level = 0;
21     switch (ch)
22     {
23     case '(':
24         level = 1;
25         break;
```

```

26     case '+':
27         level = 2;
28         break;
29     case '-':
30         level = 2;
31         break;
32     case '*':
33         level = 3;
34         break;
35     case '/':
36         level = 3;
37         break;
38     default:
39         level = 0;
40         break;
41 }
42 return level;
43 }
44
45 //函数定义3: 后缀表达式的计算
46
47 //函数定义4: 判断是否为纯数字计算
48 bool allnumber(const std::string str)
49 {
50     int a = 0;
51     for (int i = 0; i < str.length(); i++)
52     {
53         char ch = str[i];
54         if ((ch>=48 && ch<=57) || isOperator(ch) || ch=='.' || ch=='(' ||
ch==')')
55             a = a + 1;
56         else break;
57     }
58     if (a==str.length())
59         return true;
60     else
61         return false;
62 }
63
64 //函数定义5: 判断是否用到数学函数
65 bool mathfunction(const std::string str)
66 {
67     string s1 = "abs"; //绝对值
68     string s2 = "floor"; //向下取整
69     string s3 = "ceil"; //向上取整
70     string s4 = "sqrt"; //平方根
71     string s5 = "log"; //自然对数, e为底
72     string s6 = "exp"; //e的多少次方
73     string s7 = "sin"; //sin
74     string s8 = "cos"; //cos
75     string s9 = "tan"; //tan
76
77     if (str.find(s1)==string::npos && str.find(s2)==string::npos &&
str.find(s3)==string::npos
78         && str.find(s4)==string::npos && str.find(s5)==string::npos &&
str.find(s6)==string::npos
79         && str.find(s7)==string::npos && str.find(s8)==string::npos &&
str.find(s9)==string::npos)

```

```

80     {
81         return false;
82     }
83     else
84         return true;
85 }
86
87 //函数定义6: 判断是否是字母
88 bool isLetter(char ch)
89 {
90     if ((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
91     {
92         return true;
93     }
94     else
95         return false;
96 }
97
98 //函数定义7: 判断是否是变量运算
99 bool isVariable(const std::string str)
100 {
101     if (str.find_first_of('=')!=string::npos)
102     {
103         int equal = str.find_first_of('=');
104         if (equal==1)
105         {
106             char v = str[0];
107             if (isLetter(v))
108             {
109                 return true;
110             }
111             else
112                 return false;
113         }
114         else
115             return false;
116     }
117     else
118         return false;
119 }

```

main.cpp

```

1  #include "calculator.hpp"
2  #include <iostream>
3  #include <string>
4  #include <string.h>
5  #include <stack>
6  #include <math.h>
7  #include <iomanip>
8  #include <vector>
9  #include <algorithm>
10 using namespace std;
11
12 const int maxn = 100005;
13
14 string ap_add(const std::string a, const std::string b)

```



```

15 {
16     string str_a = a;
17     string str_b = b;
18     string str_a0 = a;
19     string str_b0 = b;
20     string frac_a;
21     string frac_b;
22     int pointa = 0;
23     int pointb = 0;
24     int fraclen = 0;
25
26     if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
27     {
28         if (str_a.find('.')!=string::npos)
29         {
30             pointa = str_a.find_first_of('.');
31             frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
32         }
33         else
34         {
35             str_a = str_a + ".0";
36             pointa = str_a.find_first_of('.');
37             frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
38         }
39
40         if (str_b.find('.')!=string::npos)
41         {
42             pointb = str_b.find_first_of('.');
43             frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
44         }
45         else
46         {
47             str_b = str_b + ".0";
48             pointb = str_b.find_first_of('.');
49             frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
50         }
51
52         if (str_a.find('.')!=string::npos && str_b.find('.')!=string::npos)
53         {
54             if (frac_a.length() > frac_b.length())
55             {
56                 fraclen = frac_a.length();
57                 for (int i = 1; i <= (frac_a.length()-frac_b.length());
i++)
58                 {
59                     str_b = str_b + '0';
60                 }
61                 pointb = str_b.find_first_of('.');
62                 frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
63             }
64             else if (frac_b.length() > frac_a.length())
65             {
66                 fraclen = frac_b.length();
67                 for (int i = 1; i <= (frac_b.length()-frac_a.length());
i++)
68                 {
69                     str_a = str_a + '0';

```

```

70         }
71         pointa = str_a.find_first_of('.');
72         frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
73     }
74     else fraclen = frac_a.length();
75
76     str_a = str_a.erase(str_a.find('.'),1);
77     str_b = str_b.erase(str_b.find('.'),1);
78 }
79 }
80
81 int res[maxn] = {0};
82 int lena = str_a.length();
83 int lenb = str_b.length();
84 reverse(str_a.begin(), str_a.end()); //翻转数组
85 reverse(str_b.begin(), str_b.end());
86 int len = max(lena, lenb);
87 for(int i=0; i<len; i++)
88 {
89     if ( i < lena)
90     {
91         res[i] = res[i] + (str_a[i] - '0');
92     }
93     if (i < lenb)
94     {
95         res[i] = res[i] + (str_b[i] - '0');
96     }
97
98     if(res[i] >= 10)
99     {
100         res[i+1] = res[i+1] + res[i]/10;
101         res[i] = res[i] % 10;
102     }
103 }
104
105 string result = {};
106
107 if(res[len] > 0)
108 {
109     result = result + to_string(res[len]);
110 }
111 for(int i=len-1; i>=0; i--)
112 {
113     result = result + to_string(res[i]);
114 }
115
116 if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
117 {
118     string result1 = result.substr(0, result.length()-fraclen);
119     string result2 = result.substr(result.length()-fraclen, fraclen);
120     result = result1 + "." + result2;
121 }
122
123 return result;
124 }
125
126 string ap_sub(const std::string a, const std::string b)

```

```

127 {
128     string str_a = a;
129     string str_b = b;
130     string str_a0 = a;
131     string str_b0 = b;
132     string frac_a;
133     string frac_b;
134     string int_a;
135     string int_b;
136     int pointa = 0;
137     int pointb = 0;
138     int fraclen = 0;
139     string result = {};
140
141     if ((str_a0.find('.')!=string::npos) ||
142         (str_b0.find('.')!=string::npos))//a-b
143     {
144         if (str_a.find('.')!=string::npos)
145         {
146             pointa = str_a.find_first_of('.');
147             frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
148         }
149         else
150         {
151             str_a = str_a + ".0";
152             pointa = str_a.find_first_of('.');
153             frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
154         }
155
156         if (str_b.find('.')!=string::npos)
157         {
158             pointb = str_b.find_first_of('.');
159             frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
160         }
161         else
162         {
163             str_b = str_b + ".0";
164             pointb = str_b.find_first_of('.');
165             frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
166         }
167
168         if (str_a.find('.')!=string::npos && str_b.find('.')!=string::npos)
169         {
170             if (frac_a.length() > frac_b.length())
171             {
172                 fraclen = frac_a.length();
173                 for (int i = 1; i <= (frac_a.length()-frac_b.length());
174                     i++)
175                 {
176                     str_b = str_b + '0';
177                 }
178                 pointb = str_b.find_first_of('.');
179                 frac_b = str_b.substr( pointb+1, str_b.length()-pointb-1 );
180             }
181             else if (frac_b.length() > frac_a.length())
182             {
183                 fraclen = frac_b.length();

```

```

182         for (int i = 1; i <= (frac_b.length()-frac_a.length());
i++)
183         {
184             str_a = str_a + '0';
185         }
186         pointa = str_a.find_first_of('.');
187         frac_a = str_a.substr( pointa+1, str_a.length()-pointa-1 );
188     }
189     else fraclen = frac_a.length();
190
191     int_a = str_a.substr(0, pointa);
192     int_b = str_b.substr(0, pointb);
193
194     if(int_a.length() < int_b.length())
195     {
196         result = result + "-";
197         swap(str_a, str_b);
198     }
199
200     if(int_a.length() == int_b.length())
201     {
202         if (int_a==int_b)
203         {
204             for(int i = 0; i < fraclen; i++)
205             {
206                 if(frac_b[i] < frac_a[i]) break;
207                 else if(frac_b[i] == frac_a[i]) continue;
208                 else if(frac_b[i] > frac_a[i])
209                 {
210                     result = result + "-";
211                     swap(str_a, str_b);
212                     break;
213                 }
214             }
215         }
216         else
217         {
218             for(int i = 0; i<int_a.length(); i++)
219             {
220                 if(int_b[i] < int_a[i]) break;
221                 else if(int_b[i] == int_a[i]) continue;
222                 else if(int_b[i] > int_a[i])
223                 {
224                     result = result + "-";
225                     swap(str_a, str_b);
226                     break;
227                 }
228             }
229         }
230     }
231
232     str_a = str_a.erase(str_a.find('.'),1);
233     str_b = str_b.erase(str_b.find('.'),1);
234 }
235 }
236
237 int res[maxn] = {0};
238 int lena = str_a.length();

```

```

239     int lenb = str_b.length();
240     reverse(str_a.begin(), str_a.end()); //翻转数组
241     reverse(str_b.begin(), str_b.end());
242     int len = max(lena, lenb);
243
244     if((str_a0.find('.')==string::npos) &&
(str_b0.find('.')==string::npos))
245     {
246         if(lena < lenb)
247         {
248             result = result + "-";
249             swap(str_a, str_b);
250         }
251
252         if(lena == lenb)
253         {
254             for(int i = lena-1; i>=0; i--)
255             {
256                 if(str_b[i] < str_a[i]) break;
257                 else if(str_b[i] == str_a[i]) continue;
258                 else if(str_b[i] > str_a[i])
259                 {
260                     result = result + "-";
261                     swap(str_a, str_b);
262                     break;
263                 }
264             }
265         }
266     }
267
268     for (int i = 0; i < len; i++)
269     {
270         if (str_a[i]>='0' && str_a[i]<='9')
271         {
272             res[i] = res[i] + (str_a[i] - '0');
273         }
274
275         if(str_b[i]>='0' && str_b[i]<='9')
276         {
277             res[i] = res[i] - (str_b[i] - '0');
278         }
279
280         if(res[i] < 0)
281         {
282             res[i+1] = res[i+1] - 1;
283             res[i] = res[i] + 10;
284         }
285     }
286
287
288     if ((str_a0.find('.')!=string::npos) ||
(str_b0.find('.')!=string::npos))
289     {
290         for (int i = len-1; i >= 0; i--)
291         {
292             result = result + to_string(res[i]);
293         }
294         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         {

```

```

408         re[i+j] = re[i+j] + (str_a[i]-'0')*(str_b[j]-'0');
409     }
410 }
411 for(int i=0; i<re.size()-1; i++)
412 {
413     re[i+1] = re[i+1] + re[i]/10;
414     re[i] = re[i] % 10;
415 }
416 int t = re.back();
417 re.back() = t%10;
418 t = t / 10;
419 while(t)
420 {
421     re.push_back(t%10);
422     t = t / 10;
423 }
424 while(re.size() > 1 && re.back() == 0)
425     re.pop_back();
426
427 for (int i = re.size()-1; i>=0; i--)
428 {
429     result = result + to_string(re[i]);
430 }
431
432 if (a.find('.')!=string::npos || b.find('.')!=string::npos)
433 {
434     string re_int = result.substr(0, result.length()-fraclen);
435     string re_frac = result.substr(result.length()-fraclen, fraclen);
436     result = re_int + "." + re_frac;
437 }
438
439
440 if (a[0]=='-' && b[0]!='-')
441 {
442     result = "-" + result;
443 }
444
445 if (a[0]!='-' && b[0]=='-')
446 {
447     result = "-" + result;
448 }
449
450 return result;
451 }
452
453 string div(const std::string a, const std::string b)
454 {
455     double da = stod(a);
456     double db = stod(b);
457     double re = db/da;
458     string res = to_string(re);
459     return res;
460 }
461
462 string ap_calculate(const std::string str)
463 {
464     stack<std::string> s_cal;
465     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);

```

```

523         }
524         else
525         {
526             num = div(num1, num2);
527             s_cal.push(num);
528         }
529     }
530     else
531     {
532         //double tempnum = stod(tempstr);
533         s_cal.push(tempstr);
534     }
535     i = end + 1;
536 }
537
538 return s_cal.top();
539 }
540
541 int main()
542 {
543     cout << "Please input what you want to calculate:" << endl;
544     char ch[maxn] = {};
545     cin.getline(ch, maxn);
546
547     string str = ch;
548     int varN = 0;
549
550     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
551     {
552         varN = 1;
553         string str_v;
554         string str_vcal; //含变量的计算式
555         string str_value = str.substr(2, str.length()-2); //储存变量值
556         vector<char> var_store;
557         vector<string> value_store;
558
559         var_store.push_back(str[0]);
560         value_store.push_back(str_value);
561
562         while (true)
563         {
564             char ch_v[maxn] = {};
565             cin.getline(ch_v, maxn);
566             str_v = ch_v;
567
568             if (isvariable(str_v))
569             {
570                 varN = varN + 1;
571                 string value = str_v.substr(2, str_v.length()-2);
572                 var_store.push_back(str_v[0]);
573                 value_store.push_back(value);
574                 continue;
575             }
576             else
577             {
578                 str_vcal = str_v;
579                 break;

```

```

580     }
581 }
582
583 string str_nca1 = str_vca1; //代入数值后的计算式（不含变量）
584 int irstart = 0;
585
586 for (int i = 0; i < str_vca1.length(); i++)
587 {
588     if (isLetter(str_vca1[i]))
589     {
590         for (int j = 0; j < varN; j++)
591         {
592             if (var_store[j]==str_vca1[i])
593             {
594                 str_nca1.replace(str_nca1.find(str_vca1[i]), 1,
value_store[j]);
595             }
596             else continue;
597         }
598     }
599     else continue;
600 }
601
602 str = str_nca1;
603 }
604
605
606 if (allnumber(str)) //纯算数计算
607 {
608     //获取字符串长度
609     int len = str.length();
610     //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
611     stack<char> s_op;
612     stack<char> s_conv;
613     //输出目标, 后缀表达式
614     char ch_conv[len] = {};
615     //遍历字符串, 按优先级放入中间结果栈
616     for (int i = 0; i < len; i++)
617     {
618         if ((str[i]>=48 && str[i]<=57) || str[i]=='.' ) //数字-直接放入栈
619         {
620             s_conv.push(str[i]);
621         }
622         else if (isOperator(str[i])) //操作符-判断优先级
623         {
624
625             while (true)
626             {
627                 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
628                 {
629                     s_op.push(str[i]);
630                     s_conv.push(' '); //如果输入的数超过个位无法区分, 加空格
来区分
631                     break;
632                 }
633                 else //优先级小或相等的时候拿出栈顶, 再进行循环
634                 {

```

```

635         s_conv.push(' ');
636         char c1 = s_op.top();
637         s_op.pop();
638         s_conv.push(c1);
639     }
640 }
641 }
642 else //括号
643 {
644     if (str[i]=='(')
645     {
646         s_op.push(str[i]);
647     }
648     else //str[i]==')'的情况
649     {
650         while (s_op.top()!='(')
651         {
652             s_conv.push(' ');
653             char c2 = s_op.top();
654             s_op.pop();
655             s_conv.push(c2);
656         }
657         s_op.pop(); //丢掉'('
658     }
659 }
660 }
661
662 while (s_op.empty()==0)
663 {
664     s_conv.push(' ');
665     char c3 = s_op.top();
666     s_op.pop();
667     s_conv.push(c3);
668 }
669
670 //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
671 //可以利用s_op将s_conv中数据的顺序颠倒，再赋给ch_conv
672
673 while (s_conv.empty()==0)
674 {
675     char c4 = s_conv.top();
676     s_conv.pop();
677     s_op.push(c4);
678 }
679
680 int ci = 0;
681 while (s_op.empty()==0)
682 {
683     ch_conv[ci] = s_op.top();
684     ci = ci + 1;
685     s_op.pop();
686 }
687
688 ch_conv[ci] = '\0';
689 //至此，获得了后缀表达式
690
691 //后缀表达式计算，先将char*转成string类型
692 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})
```