

CS205 C/ C++ Programming - Lab Assignment2

name:李冰 (Libing)

SID:12110141

My codes are provided on the github

<https://github.com/2069958859/Project2.git> ,please check it there.

Part1-Analysis

1、 This Project is to implement a much better calculator, I implement 6 operators: + - * / % ^ and 3 types of parentheses: () [] {}. Besides, it can calculate math functions "sqrt(x)" and be defined using "x=a".

When I input a line of arithmetic expression, it can output the correct results.

2、 Because there are several types of operators(+ - * / % ^ () [] {}) with different priority, I will get the correct answer after turn them into a postfix expression, and then calculate the answer using postfix expression.

Postfix expression has an advantage that the operator is evaluated in the order in which the operator appear, and the parentheses are removed, so the evaluation is easy to implement.

3、 To support arbitrary precision, I will use string to calculate.

Part2-Code

Here are source.cpp and arithmetic.cpp two cpp files, and source.hpp one hpp file.

The followings are some important data names:

```
stack<char> symbolStack;    //存符号的栈
stack<string> fingerStack;  //存数字的栈
stack<double> doubleFinger; //以double类型存数字的栈
vector<string> postfix;     // expression前缀表达式形式
vector<string> dataName;    //存代数式的数据名称
vector<string> Data;        //存代数式的数据值
```

In source.cpp, I will implement following methods:

Mainly used to get postfix expressions and calculate results using stack.

```
bool isalgExp(string exp); //检查是否为"x=1"的形式
void algexp(string algexp); //处理代数式
int getPriority(char c);
string processSymbol(string exp); //将负数转化为0-a
bool isDouble(string x);          //返回是否是浮点数（整数也判断为true）
void getPostfix(string exp);      //将表达式转化为后缀表达式
```

```
string erasePostZero(string str); //去掉小数点后多余的0
void assignNum();
void calculate(vector<string> post); //计算后缀表达式``
```

In arithmetic.cpp, I will implement following methods:

Mainly used to auxiliary calculate result in source.cpp, based on Project1, the method of dealing with decimal point is simplified.

```
string findDot(string str); //查找是否有小数点, 如果有, 就记录小数有几位并抹去小数点
bool isNegative(string x);
bool allZero(string str); //检查是否为0
void process(string mm, string nn); //处理输入的数据, 记录有几位小数, 将小数位较少的用0补齐
string sub(string mm, string nn); // double 减法运算, 计算mm-nn
string add(string mm, string nn); // double 加法的运算
string multiply(string mm, string nn); //double 乘法运算
```

1、The code of turning the expression to the postfix expression:

Here are several situations and the corresponding processing method of the expression:

- 1、operators(+ - * / ^ %): Eject elements from the stack until a lower priority element is found (or the stack is empty). After these elements are popped, the encountered operators are pushed onto the stack.
- 2、open parentheses(([{): process like the operators, but only be popped when met the corresponding close parentheses.
- 3、close parentheses()] }): pop the stack element, printing the pop operator until an open parenthesis is encountered. Note that the open parenthesis only pops up and does not output.
- 4、numbers: print it directly.
- 5、sqrt(x) function: recognize the function and put "S" on the stack as the operator.
- 6、invalid input: exit directly.

```
int getPriority(char c) // get 优先级
{
    if (c == '+' || c == '-')
    {
        return 1;
    }
    else if (c == '*' || c == '/')
    {
        return 2;
    }
    else if (c == '%' || c == '^' || c == 'S')
    {
        return 3;
    }
    else
    {
        return 0;
    }
}
```

```

    { //数字的情况
        return 0;
    }
}

```

```

void getPostfix(string exp) //将表达式转化为后缀表达式
{
    string temp;
    for (size_t i = 0; i < exp.size(); i++)
    {
        char flag;
        switch (exp[i])
        {
            case ')':
                flag = '(';
                break;
            case ']':
                flag = '[';
                break;
            case '}':
                flag = '{';
                break;
            default:
                flag = ' ';
        }
        temp = temp + flag;
        if (exp[i] == ' ')
        { //处理输入的空格
            continue;
        }
        else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' || exp[i] == '/' || exp[i] == '%' || exp[i] == '^')
        {
            if (exp[i + 1] == '+' || exp[i + 1] == '-' || exp[i + 1] == '*' || exp[i + 1] == '/' || exp[i + 1] == '%' || exp[i + 1] == '^')
            {
                cout << "not valid input (multiple symbols) !" << endl;
                exit(0);
            }
            else
            {
                if (symbolStack.empty() || symbolStack.top() == '(' || symbolStack.top() == '[' || symbolStack.top() == '{')
                {
                    symbolStack.push(exp[i]);
                }
                else
                {
                    while (!symbolStack.empty() && (getPriority(exp[i]) <= getPriority(symbolStack.top())))
                    {
                        temp += symbolStack.top();
                        symbolStack.pop();
                        postfix.push_back(temp);
                    }
                }
            }
        }
    }
    temp += exp[i];
    postfix.push_back(temp);
}

```

```

        temp = "";
    }
    symbolStack.push(exp[i]);
}
}
else if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
{
    symbolStack.push(exp[i]);
}
else if (exp[i] == ')' || exp[i] == ']' || exp[i] == '}')
{
    while (!symbolStack.empty() && symbolStack.top() != flag)
    {
        temp += symbolStack.top();
        symbolStack.pop();
        postfix.push_back(temp);
        temp = "";
    }
    if (!symbolStack.empty() && symbolStack.top() == flag)
    {
        symbolStack.pop();
    }
}
else
{
    if (isdigit(exp[i])) //数字的情况
    {
        temp += exp[i];
        while (exp.length() > i + 1 && (isdigit(exp[i + 1]) || exp[i + 1]
== '.'))
        {
            temp = temp + exp[i + 1];
            i++;
        }
        if (!isDouble(temp)) //检验数字是否合法
        {
            cout << "not correct number!" << endl;
            exit(0);
        }
        postfix.push_back(temp);
    }
    else if (exp[i] == 's' && exp[i + 1] == 'q' && exp[i + 2] == 'r' &&
exp[i + 3] == 't')
    { // math function sqrt(x)
        temp += exp[i + 5];
        while (exp.length() > i + 6 && (isdigit(exp[i + 6]) || exp[i + 6]
== '.'))
        {
            temp = temp + exp[i + 6];
            i++;
        }
        i = i + 6;
        postfix.push_back(temp);
    }
}

```

```

        symbolStack.push('S'); //作为sqrt函数的记号
    }
    else
    { //非法输入
        cout << "not valid input!" << endl;
        exit(0); //直接结束程序
    }
}
}
while (!symbolStack.empty())
{
    temp = "";
    temp += symbolStack.top();
    symbolStack.pop();
    postfix.push_back(temp);
}
}

```

2、The implement of calculating postfix expression:

Since division is still difficult to achieve large precision even we using strings to calculate, the division is evaluated directly as a double, so the doubleFinger stack and fingerStack are used to record values synchronously.

If the divisor is 0, it can get an error hint.

```

string num1, num2;
void assignNum() //从栈中取值用来计算
{
    if (!fingerStack.empty())
    {
        num2 = fingerStack.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    if (!fingerStack.empty())
    {
        num1 = fingerStack.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
}

```

```

void calculate(vector<string> post) //计算后缀表达式
{
    string temp;
    double d1 = 0, d2 = 0, num = 0;
    for (int i = 0; i < post.size(); i++)
    {
        temp = post[i]; // string
    }
}

```

```
if (isdigit(temp[0]))
{
    num = atof(temp.c_str()); //将该字符对应的字符串转化成double类型
    doubleFinger.push(num);    //以double类型存入
    fingerStack.push(temp);    //以字符串类型存入
}
else if (temp == "+")
{
    assignNum();
    fingerStack.push(add(num1, num2));
    doubleFinger.push(stod(add(num1, num2)));
}
else if (temp == "-")
{
    assignNum();
    fingerStack.push(sub(num1, num2));
    doubleFinger.push(stod(sub(num1, num2)));
}
else if (temp == "*")
{
    assignNum();
    fingerStack.push(multiply(num1, num2));
    doubleFinger.push(stod(multiply(num1, num2)));
}
else if (temp == "/")
{
    if (!fingerStack.empty())
    {
        d2 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    if (!fingerStack.empty())
    {
        d1 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    if (d2 == 0)
    {
        cout << "错误: 除数为0" << endl;
        exit(0);
    }
    else
    {
        fingerStack.push(erasepostZero(to_string(d1 / d2)));
        doubleFinger.push(d1 / d2);
    }
}
else if (temp == "^")
{
    if (!fingerStack.empty())
    {
        d2 = doubleFinger.top();
```

```

        fingerStack.pop();
        doubleFinger.pop();
    }
    if (!fingerStack.empty())
    {
        d1 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    fingerStack.push(erasepostZero(to_string(pow(d1, d2))));
    doubleFinger.push(pow(d1, d2));
}
else if (temp == "%")
{
    if (!fingerStack.empty())
    {
        d2 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    if (!fingerStack.empty())
    {
        d1 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    fingerStack.push(erasepostZero(to_string(fmod(d1, d2))));
    doubleFinger.push(fmod(d1, d2)); // fmod可以小数求余
}
else if (temp == "S")
{ // sqrt(x) function
    if (!fingerStack.empty())
    {
        d2 = doubleFinger.top();
        fingerStack.pop();
        doubleFinger.pop();
    }
    fingerStack.push(erasepostZero(to_string(sqrt(d2))));
    doubleFinger.push(sqrt(d2)); // sqrt 函数开平方
}
}
}

```

3、Deal with algebraic expression

```

void algexp(string algexp)
{ //处理代数式
    string dataname = "";
    string temp = "";
    double data;

```

```
    if (isalgExp(algexp))
    {
        dataName.push_back(algexp.substr(0, equalmark));
        Data.push_back(algexp.substr(equalmark + 1, algexp.size()));
    }
}
```

4、main function

If the assignment is not complete, it will keep input line, it will terminate when we input "exit", or it will wait for the next expression.

```
int main()
{
    string expression;
    while (true)//可以一直计算直到遇到“exit”
    {
        getline(cin, expression);
        algexp(expression);
        if (expression == "")//输入回车的情况
        {
        }
        else if (isExit(expression))
        {
            break;
        }
        else if (expression == "help")//打印帮助文档
        {
            help();
        }
        else
        { //替换常数
            int placePI = expression.find("PI");
            while (placePI != -1)
            {
                expression.replace(placePI, 2, "3.14159265358");
                placePI = expression.find("PI");
            }
            int placeE = expression.find("E");
            while (placeE != -1)
            {
                expression.replace(placeE, 1, "2.718281828459");
                placeE = expression.find("E");
            }

            while (isalgExp(expression)) //是给变量赋值的式子
            {
                getline(cin, expression);
                algexp(expression);
            }
        }
    }
}
```



```
        for (int i = 0; i < dataName.size(); i++)
        { //将式子中的变量名替换成值
            int place = expression.find(dataName[i]);
            while (place != -1)
            {
                expression.replace(place, dataName[i].length(), Data[i]);
                place = expression.find(dataName[i]);
            }
        }
        expression = processSymbol(expression);
        getPostfix(expression);
        calculate(postfix);
        cout << fingerStack.top() << endl;

        dataName.clear(); //清空之前的信息
        Data.clear();
        postfix.clear();
        clearStack(); //将栈清空
    }
}
return 0;
}
```

Part 3 - Result & Verification

1、I use cmake to manage the source files and get the right answers for the given test cases, it will stop when input "exit" :

```
project(hello)
add_executable(cal source.cpp arithmetic.cpp )
```

2、 other test cases:

Variables can be defined with any names made up of letters: x, y, zz, hello .etc, and variables can be used many times. Besides, it can be defined by an expression:

```

libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
hello=10
world=2.7
cc=5
hello*world/cc+world
8.1
exit
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
x=1.5+1
y=6/2
x+y-10
-4.5

```

And it can support negative calculating:

```

libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
-2*2.4+(-3)/2
-6.3

```

It can also detect invalid situations:

```

libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
8+66/0
错误：除数为0
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
12+uu
not valid input!
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
xx=0
yy+3
variables are not be defined
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
3+-2
not valid input (multiple symbols) !
libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
2..+6
not correct number!

```

if there are some blanks, it can calculate correctly:

```

libing@DESKTOP-ASMAD70:/mnt/c/Users/20699/Desktop/c++/nkdProject2/Project2$ ./cal
2 + 3/ 2
3.5

```

Here is a help documentation which can tell users how to use it when input "help":(I will show only part of it)

```

help
This is an instruction to the calculator user.
constant value:
E=2.718281828459
PI=3.14159265358

Supported functions:
1. Basic mathematical calculation: + - * / % ^
example-1.1:
3*4
12
example-1.2:
3.45/3.2
1.078125

2. calculation with brackets: ( ) [ ] { }
example-2.1:
(4.5+6.8)*1.2
13.56
example-2.2:
{2+3*[5+6/(2+3)]*(2+3)}*2
190.0

3. Definition with any names made up of letters: x, y, zz, hello ,and can be defined by an expression
example-3.1:
x=3
y=6
x+2*y
21

```

Part 4 - Problems & Summary

1、problem

When I calculate with double, the string stack can't record the result, so in the calculate method I calculate with both stacks in parallel, using `to_string()` to convert double to string, but in this way, the result will keep 6 decimals. Like 1.000000, so I write a method `erasepostZero` to get rid of any extra zeros after the decimal point:

```

string erasepostZero(string str) //去掉小数点后多余的0
{
    string ans;
    int dot = str.find(1, '.');
    int zeros = 0;
    for (int i = str.size() - 1; i >= dot; i--)
    {
        if (str[i] == '0' || str[i] == '.')
        {
            zeros++;
        }
        else
        {
            break;
        }
    }
    ans = str.substr(0, str.size() - zeros);
    return ans;
}

```

and use it in the calculate method:

```
fingerStack.push(erasepostZero(to_string(d1 / d2)));
doubleFinger.push(d1 / d2);
```

2、summary

I gained a lot from this project. First of all, I learned how to use stacks, and the conversion and meaning between infix and postfix expressions. Secondly, I also improved the problem that the decimal point was very jumbled (冗杂) in the last project calculation, to solve it, I handled it directly in the method. As followings, take multiplication as an example.

```
string findDot(string str) //查找是否有小数点，如果有，就记录小数有几位并抹去小数点
{
    string a = str;
    dot = a.find(".", 1);
    if (dot != -1)
    {
        a.erase(dot, 1);
        dot = str.length() - dot - 1; //计算小数点后有几位
    }
    else
    { //无小数点，即有0位小数
        dot = 0;
    }
    return a;
}
```

```
string multiply(string mm, string nn)
{
    int dotsum = 0;
    string m = findDot(mm);
    dotsum = dot;
    string n = findDot(nn);
    dotsum += dot;
    bool negative = false;
    if (isNegative(mm))
    {
        m = m.substr(1, mm.length());
    }
    if (isNegative(nn))
    {
        n = n.substr(1, nn.length());
    }
    if ((isNegative(mm) && !isNegative(nn)) || (!isNegative(mm) && isNegative(nn)))
    {
        negative = true;
    }
    if (m.size() < n.size() || (m.size() == n.size() && m < n))
    { //保证m大
```

```

        swap(m, n);
    }
    int a = m.size();
    int b = n.size();
    int flag = 0;
    string ans;
    for (int i = b - 1; i >= 0; i--)
    {
        flag = 0;
        string aa(b - i - 1, '0');
        for (int j = a - 1; j >= 0; j--)
        {
            flag = flag + (m[j] - '0') * (n[i] - '0');
            aa.append(to_string(flag % 10));
            flag = flag / 10;
        }
        if (flag != 0)
        {
            aa.append(to_string(flag));
        }

        reverse(aa.begin(), aa.end());
        ans = add(ans, aa);
    }

    if (dotsum != 0)
    {
        flag = 0;
        for (int i = 0; i < ans.size() - dotsum - 1; i++)
        {
            if (ans[i] == '0')
            {
                flag++;
            }
        }
        ans = ans.substr(flag, ans.size()); //除去前面的0
        ans = ans.insert(ans.length() - dotsum, ".");
    }
    if (negative)
    {
        ans = "-" + ans;
    }
    return ans;
}

```

When I do subtraction and addition, I will process it first:

```

void process(string mm, string nn)
{ //处理输入的数据, 记录有几位小数, 将小数位较少的用0补齐
    m = findDot(mm);
    dot1 = dot;

```

```
n = findDot(nn);
dot2 = dot;
if (dot1 != 0 || dot2 != 0)
{ //补齐位数
    if (dot1 >= dot2)
    { // m的小数位多
        n.append(dot1 - dot2, '0');
        dot = dot1;
    }
    else if (dot1 < dot2)
    {
        m.append(dot2 - dot1, '0');
        dot = dot2;
    }
}
}
```