CS205 Assignment 2 Report

SID: 11910718 Name: 陆彦青

Part 1. Introduction & Features

1. This is a simple calculator that supports the operations of addition, subtraction, multiplication, division, exponentiation, square root, mod and factorial for positive and negative integers and decimals.
2. It also supports the definition of variables(all the letters can be the name of variables) and the assignment of variable's value by the values of other variables.
3. Specifically, it can support arbitrary precision for addition, subtraction and multiplication.
4. Some math functions such as **sqrt()** are supported.
5. The postfix expression and recursion are used to complete the functions.

Part 2. Result & Verification

Test case #1:

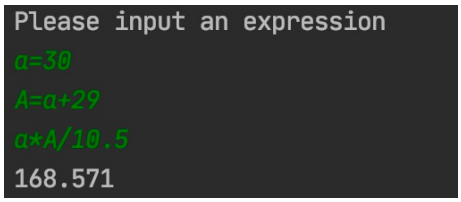
```
Input:  (5+2)*3^2%10
Output: 3
```

Screen-shot for case #1:

Test case #2:

```
Input:  a=30
        A=a+29
        a*A/10.5
Output: 168.571
```

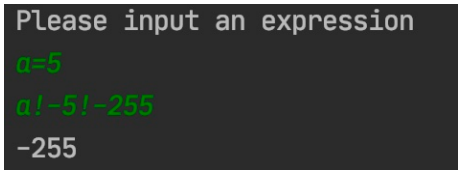
Screen-shot for case #2:



Test case #3:

```
Input:  a=5
        a!-5!-255
Output: -255
```

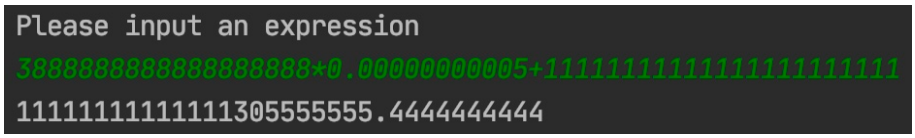
Screen-shot for case #3:



Test case #4:

```
Input:  3888888888888888888*0.00000000005+11111111111111111111111
Output: 11111111111111305555555.4444444444
```

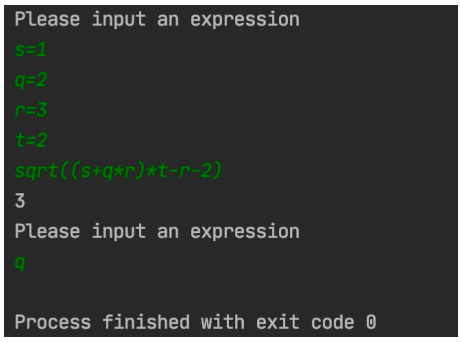
Screen-shot for case #4:



Test case #5:

```
Input:  s=1
        q=2
        r=3
        t=2
        sqrt((s+q*r)*t-r-2)
Output: 3
```

Screen-shot for case #5:



Part 3. Difficulties & Solutions

1. Switch the infix expression into postfix expression. My solution is to classify the different cases of input and use the **stack** to handle them separately.

2. Support the definition of variables and assign the value of a variable with the value of existing variables. I create an array to store the value of different letters and use **recruison** in the function **toPostfix**.

3. Compute the output value of the expression. According to the laws of calculating a postfix expression, I use the **stack** again.

4. Support the operation of arbitrary precision. Since the range of **int,long** is limited, I create the new functions **add**, **sub** and **mul** to switch the numbers to **string** and compute them according to the fundamental laws of operation.

5. Recognize math functions. The function **replaceAll** is used to replace all the math functions in an expression to unused characters.

6. Support the operations of multibit numbers. My solution is to insert the separators(',') between complete numbers and operators. Then I create a function **split(const string &src)** to split the return value of **toPostfix** into a **vector<string>**. Finally, I can use the function **compute(string &exp)** to get the output value.

Part 4. Appendix(Source Code)

```
#include <iostream>
#include <vector>
#include <string>
#include <stack>
#include <cmath>

using namespace std;

vector<string> split(const string &src)//将后缀表达式分割为vector
{
    vector<string> dest;
    string::size_type start = 0, index;
    string substring;
    index = src.find_first_of(',', start);
    while (index != string::npos && start < src.size() - 1) {
        substring = src.substr(start, index - start);
        dest.push_back(substring);
        start = index + 1;
        index = src.find_first_of(',', start);

    }
    if (start < src.size()) {
        substring = src.substr(start);
        dest.push_back(substring);
    }
    return dest;
}

string& replaceAll(string& str, const string& old, const string& new_value){//替换全部指定字符串，
    便于计算某些math function
    while (str.find(old) != string::npos){
        str.replace(str.find(old),old.size(),new_value);
    }
    return str;
}

double alphabet[52] = {0};

void defineVar(char letter, double num) {
    if (letter - 'a' < 0) {
        alphabet[letter - 'A'] = num;

    } else {
        alphabet[letter - 'a' + 26] = num;
    }
}

bool isNumber(const string& str) { //判断表达式中的数字
    for (int i = 0; i < str.size(); i++) {
```

```
        if (str[i] == '.' && i < str.size() - 1 && str[i + 1] >= '0' && str[i + 1] <= '9') {
            continue;
        }
        if (str[i] == '-' && str.size() > 1) {
            continue;
        }
        if (str[i] > '9' || str[i] < '0')
            return false;
    }
    return true;
}

bool isInt(const string& str){ //判断是否为整数
    if (str.find('.') == string::npos) return true;
    else return false;
}

string add(string& a, string& b){ //高精度加法
    string sub(string& a, string b);
    string str;
    if (a[0] == '-'){
        if (b[0] == '-'){
            str = '-'+ add(a.erase(0,1),b.erase(0,1));
            return str;
        } else {
            return sub(b,a.erase(0,1));
        }
    } else {
        if (b[0] == '-'){
            return sub(a,b.erase(0,1));
        } else {
            string :: size_type a_int = a.find('.'),b_int = b.find('.'); //补0使位数相等
            string :: size_type a_dec,b_dec;
            if (a_int == string::npos){
                a_int = a.size();
                a_dec = 0;
            } else {
                a_dec = a.size()-a_int-1;
            }
            if (b_int == string::npos){
                b_int = b.size();
                b_dec = 0;
            } else {
                b_dec = b.size()-b_int-1;
            }
            if (a_int > b_int){
                for (int i = 0; i < a_int-b_int; ++i)
                    b = '0' + b;
            } else if (a_int < b_int){
                for (int i = 0; i < b_int-a_int; ++i)
                    a = '0' + a;
            }
```

```
            if (a_dec > b_dec){
                if (b_dec == 0) b += '.';
                for (int i = 0; i < a_dec-b_dec; ++i)
                    b += '0';
            } else if (a_dec < b_dec){
                if (a_dec == 0) a += '.';
                for (int i = 0; i < b_dec-a_dec; ++i)
                    a += '0';
            }

            int carry = 0,res;
            for (int i = a.size()-1; i > -1; --i) {
                if (a[i] == '.'){
                    str = '.' + str;
                } else {
                    char c = a[i], d = b[i];
                    res = (c - '0' + d - '0' + carry) % 10;
                    carry = (c - '0' + d - '0' + carry) / 10;
                    str = char(res + '0') + str;
                }
            }
            if (carry != 0) str = char(carry + '0') + str;

            return str;
        }
    }
}
string sub(string& a, string b){ //高精度减法
    string str;
    if (b[0] == '-'){
        return add(a,b.erase(0,1));
    } else {
        if (a[0] == '-'){
            str = '-' + add(a.erase(0,1),b);
            return str;
        } else {
            if (stod(a) < stod(b)) return '-' + sub(b,a);
            else {
                string::size_type a_int = a.find('.'), b_int = b.find('.'); //补0使位数相等
                string::size_type a_dec, b_dec;
                if (a_int == string::npos) {
                    a_int = a.size();
                    a_dec = 0;
                } else {
                    a_dec = a.size() - a_int - 1;
                }
                if (b_int == string::npos) {
                    b_int = b.size();
                    b_dec = 0;
                } else {
                    b_dec = b.size() - b_int - 1;
                }
```

```
                if (a_int > b_int) {
                    for (int i = 0; i < a_int - b_int; ++i)
                        b = '0' + b;
                } else if (a_int < b_int) {
                    for (int i = 0; i < b_int - a_int; ++i)
                        a = '0' + a;
                }
                if (a_dec > b_dec) {
                    if (b_dec == 0) b += '.';
                    for (int i = 0; i < a_dec - b_dec; ++i)
                        b += '0';
                } else if (a_dec < b_dec) {
                    if (a_dec == 0) a += '.';
                    for (int i = 0; i < b_dec - a_dec; ++i)
                        a += '0';
                }

                int carry = 0, res;
                for (int i = a.size() - 1; i > 0; --i) {
                    char c = a[i], d = b[i];
                    if (a[i] == '.') {
                        str = '.' + str;
                    } else {
                        res = c - d - carry;
                        if (res < 0) {
                            res += 10;
                            carry = 1;
                        } else carry = 0;
                        str = char(res + '0') + str;
                    }
                }
                if (a[0] != '.') {
                    res = a[0] - b[0] - carry;
                    str = to_string(res) + str;
                } else if (carry == 0) {
                    str = '.' + str;
                } else {
                    str = "-1." + str;
                }
                return str;
            }
        }
    }
}

string mul(string& a, string& b){ //高精度乘法
    if (a[0] == '-'){
        if (b[0] == '-'){
            return mul(a.erase(0,1),b.erase(0,1));
        } else {
            return '-' + mul(a.erase(0,1),b);
        }
```

```
    } else {
        if (b[0] == '-'){
            return '-' + mul(a,b.erase(0,1));
        } else {
            string::size_type a_int = a.find('.'), b_int = b.find('.');
            string::size_type a_dec = 0, b_dec = 0;
            if (a_int != string::npos){
                a = a.erase(a_int,1);
                a_dec = a.size()-a_int;
            }
            if (b_int != string::npos){
                b = b.erase(b_int,1);
                b_dec = b.size()-b_int;
            }
            string rows[b.size()];
            for (int i = b.size()-1; i >= 0; --i) {
                string row;
                int over = 0;
                for (int j = a.size()-1; j >= 0; --j) {
                    string tmp = to_string((a[j]-'0') * (b[i]-'0') + over);
                    if (tmp.size() > 1){
                        row = tmp[1] + row;
                        over = tmp[0]-'0';
                    } else {
                        row = tmp + row;
                        over = 0;
                    }
                }
                if (over != 0) row = char(over + '0') + row;
                int k = i;
                while ((b.size()-k-1) > 0){
                    row += '0';
                    k++;
                }
                rows[i] = row;
            }
            string result = "0";
            for (int i = 0; i < b.size(); ++i) {
                result = add(result,rows[i]);
            }
            string::size_type num = a_dec + b_dec;
            if (num != 0){
                result = result.insert(result.size()-num,".");
                while (result[result.size()-1] == '0' || result[result.size()-1] == '.') result.
                    pop_back(); //去掉多余的0
            }
            while (result[0] == '0') result.erase(0,1);
            if (result[0] == '.') result = '0'+result;
            return result;
        }
    }
}
```

```
string div(string& a, string& b){
    return to_string(stod(a) / stod(b));
}
string mod(string& a, string& b){
    return to_string((int)stod(a) % (int)stod(b));
}

string factorial(const string& in) {
    string in1 = in;
    string out = "1";
    if (in[0] != '-' && isInt(in)) { //负数、小数无法进行阶乘运算
        while (sub(in1, "1") != "0") {
            out = mul(out,in1);
            in1 = sub(in1,"1");
        }
        return out;
    }
    return in;
}

bool flag = true;

string compute(string &exp) //根据后缀表达式的规则计算
{
    vector<string> transform = split(exp);
    stack<string> tmp;
    string a, b;
    if (transform.size() == 1) {
        return transform[0];
    }
    for (string str : transform) {
        if (isNumber(str)) {
            tmp.push(str);
        } else if (str == "!") {
            a = tmp.top();
            tmp.pop();
            tmp.push(factorial(a));
        } else if (str == "@") {
            double x = stod(tmp.top());
            tmp.pop();
            tmp.push(to_string(sqrt(x)));
        }else {
            char opr = str[0];
//            cout << opr << endl;
            b = tmp.top();
            tmp.pop();
            a = tmp.top();
            tmp.pop();
            switch (opr) {
                case '+':
                    tmp.push(add(a,b));
                    break;
```

```
                case '-':
                    tmp.push(sub(a,b));
                    break;
                case '*':
                    tmp.push(mul(a,b));
                    break;
                case '/':
                    tmp.push(div(a,b));
                    break;
                case '%':
                    if (!isInt(b)) {
                        cout << "This kind of operation is invalid" << endl;
                        flag = false;
                        break;
                    }
                    tmp.push(mod(a,b));
                    break;
                case '^':
                    if (!isInt(a) || isInt(b)) {
                        cout << "This kind of operation is invalid" << endl;
                        flag = false;
                        break;
                    }
                    double c,d;
                    c = stod(a), d = stod(b);
                    tmp.push(to_string(pow(c, d)));
                    break;
                default :
                    cout << "Your input contains invalid character" << endl;
                    flag = false;
            }
        }
    }

    return tmp.top();

}

string toPostfix(string &in) { //将中缀表达式转化为后缀表达式(只在赋值时使用此方法)
    stack<char> s1;
    string out;
    in = replaceAll(in,"sqrt", "@");

    for (int i = 0; i < in.size(); i++) {
        if ((in[i] > 64 && in[i] < 91) || (in[i] > 96 && in[i] < 123)) {
            string letter;
            if (in[i] - 'a' < 0) {
                letter = to_string(alphabet[in[i] - 'A']);
            } else {
                letter = to_string(alphabet[in[i] - 'a' + 26]);
            }
            if (letter[0] == '-') {
```

```
                out += "0,";
                out += letter.substr(1);
                out += ',';
                s1.push('-');
            } else if (letter[0] == '+') {
                out += "0,";
                out += letter.substr(1);
                out += ',';
                s1.push('+');
            } else {
                out += letter;
                out += ',';
            }
            //cout << out << endl;
        } else
            switch (in[i]) {
                case ')':
                    while (!s1.empty() && s1.top() != '(') {
                        out += s1.top();
                        out += ',';
                        s1.pop();
                    }
                    s1.pop();
                    if (s1.top() == '@'){
                        out += "@,";
                        s1.pop();
                    }
                    break;

                case '(':
                    s1.push(in[i]);
                    break;
                case '!':
                    out += "!,";
                    break;
                case '*':
                case '/':
                case '%':
                    while (!s1.empty() && s1.top() != '+' && s1.top() != '-' && s1.top() != '(') {
                        out += s1.top();
                        out += ',';
                        s1.pop();
                    }
                    s1.push(in[i]);
                    break;
                case '-':
                case '+':
                    if (i == 0 || in[i - 1] == '(' || in[i - 1] == '*' || in[i - 1] == '/' ||
                                  in[i - 1] == '+' || in[i - 1] == '-') //判断+-代表正负or运算符
                    {
                        out += in[i];
                        break;
```

```
                    }
                    while (!s1.empty() && s1.top() != '(') {
                        out += s1.top();
                        out += ',';
                        s1.pop();
                    }
                    s1.push(in[i]);
                    break;
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    out += in[i];
                    if (i == in.size() - 1 || in[i + 1] != '.' && (in[i + 1] < '0' || in[i + 1] > '9')) {
                        out += ',';
                    }
                    break;
                case '.':
                    out += in[i];
                    break;
                default:
                    s1.push(in[i]);
                    break;
            }

    }
    while (!s1.empty()) {
        out += s1.top();
        out += ',';
        s1.pop();
    }
//    cout << out << endl;
    return out;
}

string toPostfix() { //重载上一个方法,function中输入便于实现递归
    cin.clear();
    string in;
    cin >> in;
    if (in == "q") //输入q表示退出程序
    {
        return "end";
    }
    stack<char> s1;
    string out;
```

```
    in = replaceAll(in,"sqrt", "@");

    for (int i = 0; i < in.size(); i++) {
        if ((in[i] > 64 && in[i] < 91) || (in[i] > 96 && in[i] < 123)) {
            string letter;
            //cout << "letterMode" << endl;
            if (in[i] - 'a' < 0) {
                letter = to_string(alphabet[in[i] - 'A']);
            } else {
                letter = to_string(alphabet[in[i] - 'a' + 26]);
            }
            if (letter[0] == '-') {
                out += "0,";
                out += letter.substr(1);
                out += ',';                 // ','是分隔符
                s1.push('-');
            } else if (letter[0] == '+') {
                out += "0,";
                out += letter.substr(1);
                out += ',';
                s1.push('+');
            } else {
                out += letter;
                out += ',';
            }
        } else
            switch (in[i]) {
                case '=': //实现定义变量及连环定义，利用了递归的思想
                    if (i > 0 || i < in.size() - 1) {
                        string subStr = in.substr(i + 1);
                        string subRes = toPostfix(subStr);
                        string tmpVal = compute(subRes);
                        defineVar(in[i - 1], stod(tmpVal));
                        return toPostfix();
                    }
                    break;
                case ')':
                    while (!s1.empty() && s1.top() != '(') {
                        out += s1.top();
                        out += ',';
                        s1.pop();
                    }
                    s1.pop();
                    if (s1.top() == '@'){
                        out += "@,";
                        s1.pop();
                    }
                    break;

                case '(':
                    s1.push(in[i]);
                    break;
```

```
            case '!':
                out += "!,";
                break;
            case '*':
            case '/':
            case '%':
                while (!s1.empty() && s1.top() != '+' && s1.top() != '-' && s1.top() != '(') {
                    out += s1.top();
                    out += ',';
                    s1.pop();
                }
                s1.push(in[i]);
                break;
            case '-':
            case '+':
                if (i == 0 || in[i - 1] == '(' || in[i - 1] == '*' || in[i - 1] == '/' ||
                    in[i - 1] == '+' || in[i - 1] == '-') //判断+-代表正负or运算符
                {
                    out += in[i];
                    break;
                }
                while (!s1.empty() && s1.top() != '(') {
                    out += s1.top();
                    out += ',';
                    s1.pop();
                }
                s1.push(in[i]);
                break;
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                out += in[i];
                if (i == in.size() - 1 || in[i + 1] != '.' && (in[i + 1] < '0' || in[i + 1] > '9')) {
                    out += ',';//加入分隔符，便于实现多位数运算
                }
                break;
            case '.':
                out += in[i];
                break;
            default:
                s1.push(in[i]);
                break;

        }
```

```
    }
    while (!s1.empty()) {
        out += s1.top();
        out += ',';
        s1.pop();
    }
//    cout << out << endl;
    return out;
}

int main() {
    while (true) {
        cout << "Please input an expression" << endl;
        string res = toPostfix();
        if (res == "end") {
            break;
        }
        string result = compute(res);
        if (flag) {
            cout << result << endl;
        }
        flag = true;
    }

    return 0;
}
```