CS205 C/ C++ Programming - Project2 Report

Name: 周三泰（Zhou Santai)

SID: 12112008

Part 1 - Analysis

> First, since we allow multiple inputs, so we not use command line to get the input, use `cin` instead. Also, since the input of numbers can be in various shapes, we should construct a `number` class to consistently cope with them. In `number`class, we have different member variables, such as `number.positive` to denote the positiveness of the number, `number.decimals` to denote number's decimal digits and `number.integers`to store all the Arabic numerals it consists of. Also, calculate method should be implemented in class `number` such as addition, substraction, multiplication ,division, sin(), cos(), tan(), sqrt(), ln(), exp() as the basic operations of numbers.

i) To iterpret the input, two vectors for numbers and operators,respectively, are constructed, after carefully checked. And apply the math rules to operators according to the priority level.
ii) To allow arbitrary precision, we use arrays to store the Arabic numerals and calculate them with arrays , appling addition, substraction, multiplication, division in high precision.
iii) To enable define variable, a array to store the bijections between variables and numbers is constructed.
main method is quite concise since more specific methods are implemented in number.cpp since there are some restrictions on the input, we provide with a helpful announcement if the user enter '-h'.

Part 2 - Code

> use method clear to link operations and functions:(No worries for vector 'numbers' is empty since it is checked before enter this method.)

```
void number::clear(vector<number> &numbers, vector<char> &operators, char
a) {
    number temp;
    number number1 = numbers.back();
    numbers.pop_back();
    switch (a) {
```

```
        case '+': {
            number number2 = numbers.back();
            numbers.pop_back();
            temp = add(number2, number1);
            break;
        }
        case '-': {
            number number2 = numbers.back();
            numbers.pop_back();
            temp = minus(number2, number1, 0);
            break;
        }
        case '*': {
            number number2 = numbers.back();
            numbers.pop_back();
            temp = mul(number2, number1);
            break;
        }
        case '/': {
            number number2 = numbers.back();
            numbers.pop_back();
            temp = div(number2, number1);
            break;
        }
        case 's': {
            temp = sin(number1);
            break;
        }
        case 'c': {
            temp = cos(number1);
            break;
        }
        case 't': {
            temp = tan(number1);
            break;
        }
        case 'e': {
            temp = exp(number1);
            break;
        }
        case 'l': {
            temp = ln(number1);
            break;
```

```
        }
        case 'S': {
            temp = sqrt(number1);
            break;
        }
        default: {
            cerr << "invalid operators!";
            numbers.push_back(number1);
        }
    }
    numbers.push_back(temp);
}
```

then the method add,minus,mul,div,exp(),ln(),sin(),cos(),tan() are done respectively.

use the method function to manage all functions in a unified way.

```
number number::function(const number &num1, char a) {
    number result;
    if (num1.integers.empty()) {
        cerr << "invalid input for cos" << endl;
    }
    int ints[1000] = {};
    convertNumToInts(num1, ints);
    int k = beginIndexFinding(ints);
    long long integer_part = 0;
    long double fraction_part = 0;
    for (int i = ints[0]; i <= 500; ++i) {
        integer_part = integer_part * 10 + ints[i];
        if (integer_part > 10000000000000)cerr << "we can not calculate
such a huge number" << endl;
    }
    for (int i = k; i >= 501; --i) {
        fraction_part = fraction_part * 0.1 + 0.1 * ints[i];
    }
    long double num = integer_part + fraction_part;
    long double answer;
    switch (a) {
        case 's': {
            answer = sinl(num);
            break;
        }
        case 'c': {
```

```
            answer = cosl(num);
            break;
        }
        case 'l': {
            answer = log(num);
            break;
        }
        case 'e': {
            answer = expl(num);
            break;
        }
        case 'S': {
            answer = sqrtl(num);
            break;
        }
    }
    if (answer < 0)result.positive = 0;
    else result.positive = true;
    answer = abs(answer);

    std::stringstream ss;
    ss << std::setprecision(15) << answer;
    string str = ss.str();

    int j = 0;
    if (answer < 1)j += 2;
    result.decimals = 9 + j;
    if (ceil(answer) == floor(answer))
        for (int i = 0; i < answer; ++i) {
            result.integers.push_back(str[i] - '0');
            result.decimals = 0;
            return result;
        }//把整数的直接返回了，有小数的再取9位小数
    for (int i = j; i < j + 500 - ints[0] + 11; ++i) {
        if (str[i] == '.')continue;
        result.integers.push_back((str[i] - '0'));
    }
    return result;
}
```

Since the limited space, only substraction , multiplication and division part will be shown.

```
number number::minus(const number num1, number num2, int i) {//正常顺序是0,
反着输入过是1

    if (num1.positive == num2.positive) {
        bool flag = abs_com(num1, num2);
        if (equal(num1, num2)) {
            number temp;
            temp.positive = 1;
            temp.decimals = 0;
            temp.integers = vector<int>{0};
            return temp;
        }

        //    if (flag)result.positive = num1.positive;
        //     else result.positive = !num1.positive;
        number result;
        if (flag) {//namely num1 is larger than num2
            result = tempMinus(num1, num2);
        } else {
            //result = minus(num2, num1,1);
            result = tempMinus(num2, num1);
            result.positive = !result.positive;
        }
        return result;
    } else {
        num2.positive = !num2.positive;
        number temp = add(num1, num2);
        temp.positive = num1.positive;
        return temp;
    }
}
```

```
number number::mul(const number &num1, const number &num2) {
    bool positiveness = num1.positive == num2.positive;
    number result;
    int ans[1000] = {0};
    result.positive = positiveness;
    int cal1[1000] = {0};//cal【500】
    int cal2[1000] = {0};//cal【500】
    number::convertNumToInts(num1, cal1);
    number::convertNumToInts(num2, cal2);
    for (int i = 500 + num1.decimals; i >= 500 - (num1.integers.size() -
num1.decimals) + 1; --i) {
```

```
        for (int j = 500 + num2.decimals; j >= 500 - (num2.integers.size()
- num2.decimals) + 1; --j) {
            ans[i + j - 500] += cal1[i] * cal2[j];
            ans[i + j - 501] += ans[i + j - 500] / 10;
            ans[i + j - 500] = ans[i + j - 500] % 10;
        }
    }
    for (int i = 500 - (num1.integers.size() - num1.decimals)
                - ((num2.integers.size() - num2.decimals)) - 5//多减几个 没
事
            ; i < 1000; ++i) {
        if (ans[i] != 0) {
            ans[0] = i;
            break;
        }
    }
    int d = -1000;
    for (int i = 999; i >= ans[0]; --i) {
        if (ans[i] != 0) {
            d = i;
            break;
        }
    }
    if (d != -1000)result.decimals = d - 500;//注意decimal可能是负的;
    else cerr << "decimal mistake!" << endl;
    convertIntsToNum(result, ans);

    return result;
}
```

```
number number::div(const number &num1, number num2) {
    number result;
   if (num2.integers.size() - num2.decimals >= 10 || num2.integers.size()
>= 12) {
       cerr << "we can not div such a complex number";
    }
    if ((num2.integers.size() == 1 && num2.integers.back() == 0) ||
num2.integers.empty())
        cerr << "invalid input for div. we can not divide zero." << endl;

    int cal1[1000] = {0};
    convertNumToInts(num1, cal1);
    int cal2[1000] = {0};
```

```
    convertNumToInts(num2, cal2);

    int stopIndexForNum1 = stopIndexFinding(cal1);
    int stopIndexForNum2 = stopIndexFinding(cal2);
    int length = num2.integers.size();
    int indexDifference = stopIndexForNum2 - stopIndexForNum1;
    int caltemp[1000] = {0};
    caltemp[0] = cal1[0];
    int ans[1000] = {0};
    for (int i = stopIndexForNum2; i < length + stopIndexForNum2; ++i) {
        caltemp[i - indexDifference] = cal2[i];//对齐
    }
    int deviation = 0;
    for (int i = cal1[0]; i < cal1[0] + num1.integers.size() +
num2.integers.size() + 10; ++i) {
        while (true) {
            bool flag = false;
            int count = 0;
            for (int j = i; j < i + length; ++j) {
                if (cal1[j] > caltemp[j - deviation]) {
                    flag = true;
                    break;
                } else if (cal1[j] == caltemp[j - deviation])count++;
                else flag = false;
            }//比出是可以减的，就继续
            if (flag || count == length) {//可以减
                for (int j = i; j < i + length; ++j) {
                    cal1[j] -= caltemp[j - deviation];
                    if (cal1[j] < 0) {
                        cal1[j] += 10;
                        int k = j;
                        while (cal1[--k] <= 0) {}
                        cal1[k]--;
                        for (int l = k + 1; l < j; ++l) {
                            cal1[l] += 9;
                        }
                    }
                }
                ans[i + length - 1]++;
            } else break;//假如不可以减了，就跳到下一位了
        }
        cal1[i + 1] += cal1[i] * 10;
        cal1[i] = 0;//到下一位了，那个首位数就要归0 给到后面
```

```
            deviation++;
        }
        for (int i = cal1[0]; i < 999; ++i) {
            if (ans[i] != 0) {
                ans[0] = i;
                break;
            }
        }
        int beginIndex = 0;
        for (int i = 999; i > 0; --i) {
            if (ans[i] != 0) {
                beginIndex = i;
                break;
            }

        }

        for (int i = ans[0]; i <= beginIndex; i++) {
            result.integers.push_back(ans[i]);
        }
        result.decimals = beginIndex - 500 - num2.decimals;

        bool positiveness = num1.positive == num2.positive;
        result.positive = positiveness;
        return result;
    }
```

Part 3 - Result & Verification

Test case for trigonometric function:

```
please enter your equation, type "-h" for help and restriction and "quit" to quit
sin(1)
0.84147098480
cos(1)
0.54030230586
sin(0.12*2.1+1)
0.94961336569
cos(0.11)
0.99395609795
```

Test case for basic operations and brackets:

```
please enter your equation, type "-h" for help and restriction and "quit" to quit
1000000000000000000.11*2
2000000000000000000.22
9.3456789-1.2817263
8.0639526
0.45678-1023.2
-1022.74322
111.1/2
55.55
8763.3/3
2921.1
1234.2*11.293
13937.8206
3+(1+3.1)*3.33
16.653
sin(1)*2
1.6829419696
sin(1+1)
0.90929742682
1-sqrt(4)*2.222222
-3.444444
```

```
please enter your equation, type "-h" for help and restriction and "quit" to quit
356478.1234129438*1.18237
421489.038779762360806
```

Test case for exp(),ln(),sqrt():

```
please enter your equation, type "-h" for help and restriction and "quit" to quit
exp(1)
2.718281828
exp(2)
7.389056098
exp(1)*2+sqrt(4)
7.436563656
sqrt(3)
1.732050807
sqrt(4)
2
sqrt(3.3)
1.816590212
ln(2)
0.69314718055
ln(3)
1.098612288
```

Test case for giving numbers to variable

```
please enter your equation, type "-h" for help and restriction and "quit" to quit
x=1.112

y=3.33*sqrt(4)

x+y
7.772
x=9.33

y=sin(1)+cos(1)

x+y
10.71177329066
x=(2.32782323+1)*2-1

y=exp(1)

x-y
2.937364632
```

All results are verified by my casio fx-991CN X.

Part 4 - Difficulties & Solutions

1.Since many operations have something in common, to avoid the redundance, many codes are packaged as a single method, which lead to the conciseness of the program. The control stream of the program is well organized and visible. To enable arbitrary precisions, use arrays to store the Arabic numerals and successfully implement it.
2.Utilize const reference variable appropriately to avoid not inevitable copy.
3.Careful check for input and print cerr to tell user, and given a corresponding announcement(help) for user to get known of the rules.
4.Declaration of functions before the specific method in number.cpp file to free me from manage the order of detailed method.
5.enable functions as follows: sin cos tan exp ln sqrt and give numbers to variables.

Detailed code will be provided in https://github.com/zhousantai/proj2/tree/master