CS205 C/ C++ Programming - A Simple Calculator

name：刘家宝 (Liu Jiabao)

SID：12110416

Part 1 -Analysis

1.1 总体目标

实现一个可以将两个数字相乘的计算器。

1.2 具体需求

整数相乘概述

符号的判断

这一步较为简单，对于整数而言，正负号只能处于index=0的位置（鲁棒性将会过滤不符合的情况），故我们用如下代码实现结果符号的判定，其中，isNegative变量标定了结果字符串是否需要在前添加负号：

```
    //判断结果的正负
    bool isNegative = false;

    if ((first_string[0] == '-' && second_string[0] != '-') || (first_string[0]
!= '-' && second_string[0] == '-'))
    {
        isNegative = true;
    }

    ll first_string_size = first_string.size();
    ll second_string_size = second_string.size();

    if (first_string[0] == '-')
    {
        first_string_size--;
        first_string = first_string.substr(1, first_string_size - 1); //截取除了负
号后面的一段字符串做之后的计算
    }
    if (second_string[0] == '-')
    {
        second_string_size--;
        second_string = second_string.substr(1, second_string_size - 1); //截取除
了负号后面的一段字符串做之后的计算
    }
```

高精度的实现

开辟三个个vector数组A、B、C存放两个长字符串和结果，我们很容易知道A和B相乘之后的位数不会超过A+B字符串位数的总和，因此我们可以开辟一个长A+B的vector空间，依次将乘数的第i位乘以被乘数的第j位，并将产生的结果存入vector数组的第i+j位，之后遍历C中的每一个元素进行进位操作，同时对C因为多开辟空间而多出来的初始化的0进行删除，最后产生的C中的元素依次放入字符串res输出即为乘法的结果，由此，我们可以实现高精度乘法

相关代码如下：

```
string multiply1(string num1, string num2)
```

```
{
    string res;
    vector<int> A, B, C;

    for (int i = num1.size() - 1; i >= 0; i--)
    {
        A.push_back(num1[i] - '0');// 将num1中每个元素转为int类型
    }
    for (int i = num2.size() - 1; i >= 0; i--)
    {
        B.push_back(num2[i] - '0');// 将num2中每个元素转为int类型
    }

    C = mul(A, B);// 调用mul方法
    for (int i = C.size() - 1; i >= 0; i--)
    {
        res += to_string(C[i]);//累加乘法结果
    }
    return res;
}

vector<int> mul(vector<int> A, vector<int> B)
{
    vector<int> C(A.size() + B.size(), 0);// 开辟一个长A+B的vector空间

    for (int i = 0; i < A.size(); ++i)
    {
        for (int j = 0; j < B.size(); ++j)
        {
            C[i + j] += A[i] * B[j];//将产生的结果存入vector数组的第i+j位
        }
    }

    int temp = 0;
    for (int &i : C)
    {
        temp += i;
        i = temp % 10;// 进位操作
        temp /= 10;
    }

    while (C.size() > 1 && C.back() == 0)
    {
        C.pop_back();// 去掉C中多余初始化的0
    }
    return C;
}
```

浮点数相乘概述

笔者认为浮点数相乘主要有三个主要问题需处理：

1.将能转为小数的科学计数法转为小数计算

2.将过大的浮点数直接以科学计数法的形式计算

3.高精度浮点数相乘的实现

首先，在输入每个数据时，我们将进行数据类型的判断，得到一个数据是否为浮点数，整数，或并非我们想要的数据，标记此时数据的类型之后，进入下列问题的解决。

（其中内置的**isFloatPoint**、**isFloatE**、**isInt**方法将于程序鲁棒性部分展示）

```
int judgeType(const string &str)
{
    if (isFloatPoint(str.c_str()))// 是浮点数
        return 1;
    if (isFloatE(str.c_str()))// 是科学计数法
        return 2;
    if (isInt(str.c_str()))// 是整数
        return 3;

    return -1;// 错误输入
}
```

将科学计数法转为小数

此方法会将能转为小数的科学计数法转为小数计算（如1.1e-2等）由于前置条件已经过滤了非法输入，我们直接调用库实现如下两个方法：

```
double stringToDouble(string s) // 将字符串转为16位以内的double 若超出范围依然会保持数学计数法
{
    double d;
    stringstream ss;
    ss << s;
    ss >> setprecision(16) >> d;
    ss.clear();
    return d;
}

string doubleToString(double d) // 将未能转为double的字符串保持为string类型，便于后序字符串相乘
{
    string s;
    stringstream ss;
    ss << setprecision(16) << d;
    s = ss.str();
    ss.clear();
    return s;
}
```

前者能将字符串转为16位以内的double，若超出范围则依然会保持数学计数法模式；后者能将经过转换的double类型重新保持为string类型，便于后序实现科学计数法相乘或高精度字符串相乘，同时，在执行完相应方法后，我们改变数据对应的标记，至此我们就实现了将能转为小数的科学计数法字符串转为小数字符串的方法

大浮点数计算的实现

对于过大浮点数的计算（如1e200 * 1e200），我们实现的方式很简单，将两个字符串e前的数字分别相乘，并考虑进位；将两个字符串e后的数字相加

实现代码如下：

```
    // 在经过处理以后仍然是科学计数法表达形式
    if (judge1 == 2 && judge2 == 2)
    {
        int carry = 0;// 处理进位的变量
        int new_first_part = 0; // 得到的新浮点数的整数部分
        int new_second_part = 0; // 得到的新浮点数的指数部分

        // 进行第一个数据的处理
        for (int i = 0; i < first_string.size(); ++i)
        {
            if (first_string[i] == '.')
            {
                point1 = i;
            }
            if (first_string[i] == 'E' || first_string[i] == 'e')
            {
                E1 = i;
            }
        }
        first_part1 = stoi(first_string.substr(0, E1));// 截取
        second_part1 = stoi(first_string.substr(E1 + 1));

        // 进行第二个数据的处理
        for (int i = 0; i < first_string.size(); ++i)
        {
            if (first_string[i] == '.')
            {
                point2 = i;
            }
            if (first_string[i] == 'E' || first_string[i] == 'e')
            {
                E2 = i;
            }
        }
        first_part2 = stoi(first_string.substr(0, E2));// 截取
        second_part2 = stoi(first_string.substr(E2 + 1));

        new_second_part = second_part1 + second_part2;
        new_first_part = first_part1 * first_part2;

        if (new_first_part >= 10)// 判断是否需要进位
        {
            new_first_part %= 10;
            carry++;
        }
        new_second_part += carry;
        cout << first_string + " * " + second_string + " = ";
        cout << new_first_part << 'e' << new_second_part << endl;
```

```
        goto loop;// 循环完毕 再次请求输入
    }
```

小数相乘的实现

基于整数相乘的基础，小数相乘的实现变得尤为简单，我们只需要标记两个小数点的位置，然后删除两个小数点，此时我们就得到了两个整数，结合整数相乘的方法，我们只需要将小数点放在正确的位置即可

实现代码如下：

```
    if (judge1 == 1 || judge2 == 1)
        {
            int PointA = 0;
            int PointB = 0;
            int sum;
            for (int i = 0; i < first_string_size; ++i)
            {
                if (first_string[i] == '.')
                {
                    // 记录PointA的下标
                    PointA = first_string_size - (i + 1);
                    first_string.erase(i, 1);
                }
            }
            for (int i = 0; i < second_string_size; ++i)
            {
                if (second_string[i] == '.')
                {
                    // 记录PointB的下标
                    PointB = second_string_size - (i + 1);
                    second_string.erase(i, 1);
                }
            }

            sum = PointA + PointB;

            res = multiply1(first_string, second_string);

            if (res.size() - sum != 0)
                res.insert(res.size() - sum, 1, '.');// 将小数点放在合适的位置

            if (isNegative)
            {
                res = '-' + res;
            }
            cout << res << endl;
        }

        goto loop; //运算完成之后回到程序起始，再次进行新一轮运算
    }
```

程序的鲁棒性

用户输入错误的提示

该部分程序的鲁棒性在此次project中的体现在：

1.用户输入的两个字符字符串不是两个数据（科学计数法输入有误，浮点数输入有误，不是浮点数或整数如字母）

2.用户输入的数据个数不是2（没有输入数据 或输入数据多于两个）

一旦发现有上述情况，将根据不同的情况返回对应的输出

程序鲁棒性的实现

程序鲁棒性的判断主要用到以下四个函数

```
// 判断部分声明
bool isInt(const char *str);// 判断一个字符串是否为int类型
bool isFloatE(const char *str);// 判断一个字符串是否为科学计数法的浮点数
bool isFloatPoint(const char *str);// 判断一个字符串是否为浮点数
bool isNumber(const string&);// 进行isInt，isFloatE，isFloatPoint方法的封装，判断是否为数字
```

isInt

判断较为简单，对于每个字符串中的每个char指针，如果指向的是‘+’或‘-’，他们必须处于字符串的最前端，否则就并非整数；如果指向的是数字，将初始化为fales的isInteger设为true，便于后续判断，在处理完这两步之后，其余遇到的所有非这两者的指向都会返回false

相关代码如下：

```
bool isInt(const char *str)
{
    bool isInteger = false;

    int index = 0;
    while (*str != '\0')
    {
        switch (*str)
        {
        case '0' ... '9':
            isInteger = true;
            break;
        case '+':
        case '-':
            if (index != 0) // 符号index只能为0
            {
                return false;
            }
            break;
        default:
            return false;
        }
        str++;
```

```
        index++;
    }

    if (isInteger) // 如果有整数存在返回true 反之返回false
    {
        return true;
    }
    return false;
}
```

类似的，我们可以写出**isFloatE**和**isFloatPoint**方法，前者为判断浮点数是否满足科学计数法的形式，后者为判断浮点数是否满足非科学计数法的浮点数表达形式

isFloatPoint

首先，对于非科学计数法小数，我们一定要有小数点，所以设置isPoint变量，并且小数点前和小数点后都要有数字，故设置numBefore和numBehind变量为false，经过char指针遍历的过程中，在isPoint位置前遇到数字，则将numBefore设置为true；在isPoint位置后遇到数字，则将numBehind设置为true，同时在最后判断是否三者皆为true

相关代码如下：

```
bool isFloatPoint(const char *str)
{
    int isPoint = 0;
    int index = 0;
    bool numBefore = false;
    bool numBehind = false;

    while (*str != '\0')
    {
        switch (*str)
        {
        case '0' ... '9':
            if (isPoint)
            {
                numBehind = true;
            }
            else
            {
                numBefore = true;
            }
            break;
        case '.':
            if (isPoint || !numBefore) // 如果已经有小数点或小数点前面没有数字
            {
                return false;
            }
            else
            {
                isPoint = index;// 反之记录小数点的index
            }
            break;
        default:
            return 0;
```

```
        }
        index++;
        str++;
    }

    if (!numBefore)
    {
        return 0;
    }
    else if (isPoint && !numBehind)
    {
        return 0;
    }

    return isPoint;
}
```

isFloatE

其次，对于科学计数法的小数而言，我们可以直接理解其为 A + E/e + B 的形式 ， 其中 A 可以为整数 也可以为浮点数（但不能为科学计数法类型） ， B可以为任意整数，此两者的判断方式类似于前两种方法的判断。基于此，我们的判断逻辑是：对于字符串每个char指针进行遍历，判断E/e之前和之后是否有相应的A和B，实现代码类似于前两个方法，只不过我们加入了isE变量和symbol变量，前者用来记录isE是否存在及其位置，后者用来记录符号的位置是否正确

相关代码如下：

```
bool isFloatE(const char *str)
{
    int isE = 0;
    int isPoint = -1;
    int symbol = 1;
    bool numBefore = false;
    bool numBehind = false;

    int index = 0;
    while (*str != '\0')
    {
        switch (*str)
        {
        case '.':
            isPoint = index;
            break;
        case '0' ... '9':
            if (isE)
            {
                numBehind = true;
            }
            else
            {
                numBefore = true;
            }
            break;
        case '+':
        case '-':
```

```
            if (index != 0 && index != isE + 1) // ‘+’和‘-’既可以在index=0的位置，也可以在isE+1的位置
            {
                symbol = 0;
            }
            break;
        case 'e':
        case 'E':
            if (isE || !numBefore)// 如果已经有E或前面没有数字
            {
                return false;
            }
            else
            {
                isE = index;// 反之记录E的index
            }
            break;
        default:
            return 0;
        }
        index++;
        str++;
    }

    if (!numBefore)// 如果E前面没有数字
    {
        return 0;
    }
    else if (!symbol)// 如果符号的位置不对
    {
        return 0;
    }
    else if (isE && !numBehind) // 如果E存在且E后面没有数字
    {
        return 0;
    }

    return isE;
}
```

最后，我们将上述三者方法封装，得到isNumber方法，以用来判断字符串是否为我们想要的数据

```
// 封装方法 判断字符串是否为一个数
bool isNumber(const string &s)
{
    if (isInt(s.c_str()) || isFloatPoint(s.c_str()) || isFloatE(s.c_str()))
    {
        return true;
    }

    return false;
}
```

Part 2 - Some improvement

用户二次输入的实现

尽管只是作为一个简单的乘法器，考虑到代码的复用性以及用户使用这一种程序的感受，肯定不希望运行一次就只能做一次乘法运算，一定是希望自己能够控制它停止或者继续，重复不断地、直到完成自己想要的乘法运算。为了满足用户的这一需要， 本程序通过使用 goto语句控制程序的运行顺序，分别在程序运算结束之后与提示用户输入不符合规范之后回到程序的初始阶段重新开始 读取用户的输入，直到用户输入了“quit”为止，实现了重复次数的乘法运算。

字符串乘法的竖式计算逻辑

通过分析两数通过竖式作乘法运算的过程可以发现，每次错位相加都可以省略被加数的个位进行运算，这样就可以很大程度节省运算时间与空间，也方便这一字符串相加程序的编写，也不用考虑错位的位数关系，每次都是直接进行运算即可，而且运算的最大位数即为两字符串位数的相加值。

Part 3 - Result and Verification

整数相乘

（以下为经过科学计算器验证的正确结果）

case1：

12342116546151567890 * 123456789045641614498 5
=1523718078814957004509913355386731053165 0

case2：

1234567890*1234567890 = 1524157875019052100

case3：

12342116546112634551567890 *12345678911561569104564161449 85 =
1523718080174143149715390067217889869036758630541310531650

case4：

2*3 = 6

```
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ g++ main.cpp -o caculator
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 12342116546151567890 123456789045641614498 5
 12342116546151567890 * 123456789045641614498 5 = 1523718078814957004509913355386731053165 0
 Please inter two Numbers, or enter " quit " to quit the loop:
 quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1234567890 1234567890
 1234567890 * 1234567890 = 1524157875019052100
 Please inter two Numbers, or enter " quit " to quit the loop:
 quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 12342116546112634551567890 12345678911561569159104564161449 85
 12342116546112634551567890 * 12345678911561569159104564161449 85 = 1523718080174143149715390067217889869036757863054131053165 0
 Please inter two Numbers, or enter " quit " to quit the loop:
 quit
```

```
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 2 3
 2 * 3 = 6
 Please inter two Numbers, or enter " quit " to quit the loop:
 quit
```

浮点数相乘

case1：

1.1e-2 * 1.1e2 = 0.011 * 110 = 1.210

case2：

1e200 * 1e200 = 1e+200 * 1e+200 = 1e400

case3：

3.1416 * 2 = 6.2832

```
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ g++ main.cpp -o caculator
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1.1e-2 1.1e2
0.011 * 110 = 1.210
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1.0e200 1.0e200
1e+200 * 1e+200 = 1e400
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 2 3
2 * 3 = 6
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 3.1416 2
3.1416 * 2 = 6.2832
Please inter two Numbers, or enter " quit " to quit the loop:
quit
```

鲁棒性测试

case1：

输入：./cacular 1 a

输出：Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:

case2：

输入：./cacular e1 1e-2

输出：Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:

case3：

输入：./cacular 1 1 1 1 1

输出：Please inter two proper Numbers, or enter " quit " to quit the loop:

case4：

输入：./cacular

输出：You haven't put any number in command

```
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ g++ main.cpp -o caculator
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1 a
Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator e1 1e-2
Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1 1 1 1 1
Sorry, you should enter two proper numbers, please try again or enter " quit " to quit the loop:
Please inter two Numbers, or enter " quit " to quit the loop:
quit
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator
You haven't put any number in command
Please inter two Numbers, or enter " quit " to quit the loop:
```

二次输入测试

经检验，二次输入符合我们的预期，其中整数与整数运算，浮点数运算，以及非法输入判断结果如下：

```
nian@仅此而已:/mnt/c/Users/nian5/Desktop/b$ ./caculator 1 1
1 * 1 = 1
Please inter two Numbers, or enter " quit " to quit the loop:
123456789 1234567890
123456789 * 1234567890 = 152415787501905210
Please inter two Numbers, or enter " quit " to quit the loop:
1.23456789 1.234567890
1.23456789 * 1.234567890 = 1.52415787501905210
Please inter two Numbers, or enter " quit " to quit the loop:
e1 1e200
Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:
Please inter two Numbers, or enter " quit " to quit the loop:
123456789 1.234567890
123456789 * 1.234567890 = 152415787.501905210
Please inter two Numbers, or enter " quit " to quit the loop:
1a a
Wrong input! Please inter two Numbers, or enter " quit " to quit the loop:
```

Part 4 - Code

mathod.h

```
#include <string>
#include <vector>

using namespace std;

// 判断
int judgeType(const string &);
bool isInt(const char *str);
bool isFloatE(const char *str);
bool isFloatPoint(const char *str);
bool isNumber(const string &);
// 乘法
string multiply1(string num1, string num2);
vector<int> mul(vector<int> A, vector<int> B);
// 转换
double stringToDouble(string s);
string doubleToString(double d);
```

main.cpp

(已经展示过的mathod.h中的方法不再放入以节省空间)

```
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <sstream>
#include "mathod.h"

using namespace std;

typedef long long ll;

int main(int argc, char **argv)
{

    string first_string, second_string; // 用两个字符串储存起始输入的两个数
    bool first_loop = true;             // 对是否第一次进入程序做标记
    int judge1, judge2;                 // 分别标记两个字符串的类型

    // 如果有科学计数法需要的话 以下变量将储存变量信息
    int E1 = 0;
    int point1 = -1;
```

```
    int first_part1 = 0;
    int second_part1 = 0;

    int E2 = 0;
    int point2 = -1;
    int first_part2 = 0;
    int second_part2 = 0;

loop:
    // 1.用命令行运行程序时加载数据的输入
    if (first_loop)
    {
        if (argc <= 1)
        {
            cout << "You haven't put any number in command" << endl;
            first_loop = false; //将first_loop设置为false
            goto loop;
        }
        else if (argc != 3)
        {
            cout << "Sorry, you should enter two numbers, please try again or
enter \" quit \" to quit the loop:" << endl;
            first_loop = false;
            goto loop;
        }

        first_string = argv[1];
        second_string = argv[2];
        first_loop = false;

        if ((!isNumber(first_string) || !isNumber(second_string)))
        {
            cout << "Wrong input! Please inter two Numbers, or enter \" quit \"
to quit the loop:" << endl;
            goto loop;
        }
    }
    else
    {
        // 2.用命令行输入参数计算
        cout << "Please inter two Numbers, or enter \" quit \" to quit the
loop:" << endl;
        cin >> first_string;

        //无论是否为第一次输入程序 当输入的第一个字符串为quit时，退出程序
        if (first_string == "quit")
        {
            return 0;
        }
        cin >> second_string;
    }

    //无论是否为第一次输入程序 当输入的第一个字符串为quit时，退出程序
    if (first_string == "quit")
    {
```

```
        return 0;
    }

    // 判断两个字符串的数据类型
    judge1 = judgeType(first_string);
    judge2 = judgeType(second_string);

    // 进行字符串转换操作，将可以转为double的科学计数法浮点数转为浮点数字符串，否则保持字符串形式
    first_string = doubleToString(stringToDouble(first_string));
    second_string = doubleToString(stringToDouble(second_string));

    // 任意一个数字并非数字
    if (judge1 == -1 || judge2 == -1)
    {
        cout << "Sorry, you should enter two numbers, please try again" << endl;
        goto loop;
    }

    // 判断两个字符串的数据类型
    judge1 = judgeType(first_string);
    judge2 = judgeType(second_string);

    // 在经过处理以后仍然是科学计数法表达形式
    if (judge1 == 2 && judge2 == 2)
    {
        int carry = 0;
        int new_first_part = 0;
        int new_second_part = 0;

        for (int i = 0; i < first_string.size(); ++i)
        {
            if (first_string[i] == '.')
            {
                point1 = i;
            }
            if (first_string[i] == 'E' || first_string[i] == 'e')
            {
                E1 = i;
            }
        }
        first_part1 = stoi(first_string.substr(0, E1));
        second_part1 = stoi(first_string.substr(E1 + 1));

        for (int i = 0; i < first_string.size(); ++i)
        {
            if (first_string[i] == '.')
            {
                point2 = i;
            }
            if (first_string[i] == 'E' || first_string[i] == 'e')
            {
                E2 = i;
            }
        }
```

```
        first_part2 = stoi(first_string.substr(0, E2));
        second_part2 = stoi(first_string.substr(E2 + 1));

        new_second_part = second_part1 + second_part2;
        new_first_part = first_part1 * first_part2;

        if (new_first_part >= 10)
        {
            new_first_part %= 10;
            carry++;
        }
        new_second_part += carry;
        cout << first_string + " * " + second_string + " = ";
        cout << new_first_part << 'e' << new_second_part << endl;
        goto loop;
    }
    else
    {
        cout << first_string + " * " + second_string + " = ";
        //判断结果的正负
        bool isNegative = false;
        if ((first_string[0] == '-' && second_string[0] != '-') ||
(first_string[0] != '-' && second_string[0] == '-'))
        {
            isNegative = true;
        }

        ll first_string_size = first_string.size();
        ll second_string_size = second_string.size();

        if (first_string[0] == '-')
        {
            first_string_size--;
            first_string = first_string.substr(1, first_string_size - 1); //截取
除了负号后面的一段字符串做之后的计算
        }
        if (second_string[0] == '-')
        {
            second_string_size--;
            second_string = second_string.substr(1, second_string_size - 1); //截
取除了负号后面的一段字符串做之后的计算
        }

        judge1 = judgeType(first_string);
        judge2 = judgeType(second_string);
        string res;

        if (judge1 == 3 && judge2 == 3)
        {
            res = multiply1(first_string, second_string);
            if (isNegative)
            {
                res = '-' + res;
            }
```

```
            cout << res << endl;
        }

        if (judge1 == 1 || judge2 == 1)
        {
            int PointA = 0;
            int PointB = 0;
            int sum;
            for (int i = 0; i < first_string_size; ++i)
            {
                if (first_string[i] == '.')
                {
                    // 记录PointA的下标
                    PointA = first_string_size - (i + 1);
                    first_string.erase(i, 1);
                }
            }
            for (int i = 0; i < second_string_size; ++i)
            {
                if (second_string[i] == '.')
                {
                    // 记录PointA的下标
                    PointB = second_string_size - (i + 1);
                    second_string.erase(i, 1);
                }
            }

            sum = PointA + PointB;

            res = multiply1(first_string, second_string);

            if (res.size() - sum != 0)
                res.insert(res.size() - sum, 1, '.');

            if (isNegative)
            {
                res = '-' + res;
            }
            cout << res << endl;
        }

        goto loop; //运算完成之后回到程序起始，再次进行新一轮运算
    }
}
```

Part 5 - Summary

困难及实现

起初在实现字符串相乘时首先想到将字符串直接转为某种数据类型，后来发现存在两个问题，一是有些字符串过大难以直接转为某种数据类型，存在数据溢出的情况；二是无法准确判断数据类型及大小，难以运用合适的数据类型以存放，基于分治的思想，我们可以对字符串的类型进行判断，编写了对字符串类型的判断方法（isInt()，isFloatE()，isFloatPoint() 方法），保证程序的鲁棒性（judgeType()），之后通过调用库将数据范围足够储存在某种数据类型的数据储存（stringToDouble()，doubleToString() 方

法），直接进行计算；将难以转为某种数据类型的字符串数据实现整数，小数和科学计数法的三种大数乘法（整数和小数均基于mutiply() 方法，后者需要基于下标找到小数点位置，科学计数法基于分部相乘方法），乘法的内核为vector数据类型的运用

总结

通过编写这段乘法器代码，我了解了c++基本语法的运用，深刻地了解了乘法的运算机制，实现了高精度相乘的方法，也更深刻地了解c++数据类型的范围，起初我只使用了基本类型来实现乘法器，发现即便是使用最大范围的数据类型，精度依然很低，当输入的数字位数稍微大一点程序就近乎崩溃，所以通过这次写乘法器我也更加深刻了解了c++数据类型的魅力，同时在编写程序的过程当中也更加注意到要完善自己的编码风格，我也更加深入地了解了c++代码编写的规范性，编写了足够的注释以让别人看懂，起了一些能让人一眼看懂的变量名。在鲁棒性方面，也防止了一些不合法的输入等等。我也更加为用户着想，例如用户可能不小心输入 了不合法的数值，就要及时提醒他重新输入数值，用户可能并不想一次执行只能运算一次乘法，所以程序也设置了循环运行的机制。

综上，这次项目带给我诸多收获，感谢阅读！