

Project1: A Simple Calculator

需求分析

输入格式与输出格式

运行命令格式: `./mul a b`, 其中 a, b 既可以为文件地址, 也可是是实数。

输出答案格式: `x * y = z`, 其中 x, y, z 分别表示三个实数。

输入参数的约束

本项目实现了两个大实数相乘的乘法计算器。

“大实数”，就是一个实数，它可以是正数、负数或者零，并且其有效位数可以很多。

输入的“大整数”应该满足下列任意一种情况：

1. 整数：一个任意符号的长整数，可以带 1 位符号位（正数也可以带符号位），如 `-1189203`, `1232`, `0`, `+123`。
2. 非科学记数法小数：形如 `整数.非负整数` 的小数，如 `-11.123`、`2.0`、`+3.14`。
3. 科学记数法小数：形如 `整数e整数` 或者 `非科学计数法小数e整数` 的小数，如 `-1.2380192e-12830`、`-12.31e128301`、`0.190123e-1238902`、`+0.008190238e123`。

正因为如此，输入命令中的大实数 a, b 应满足较为宽松的限制：

1. 符号相对自由：既支持正实数，同样也支持负实数或者零，不局限于非负整数。
2. 数据范围扩大：本项目可处理数据范围足够大的大实数，不受 C++ 内部普通数的数据类型范围（如 `int`、`long long`、`_int128`、`double`、`long double` 等）数据范围的限制。
3. 数据计算精度准确：本项目准确计算两个大实数的乘法，不受 C++ 内部实数数据类型（如 `float`、`double`、`long double` 等）丢失有效位数以外精度的限制。

项目提供两种输入参数 a, b 的方法：一种是通过终端直接传入数值；第二种是通过终端传入一个文本文件，项目程序读取文本文件中的内容获得数值。

如果项目当前目录中存在文本文件 `a.txt`（其中的内容是：`2`）和 `b.txt`（其中的内容是：`3`），那么下列两种方式都可以成功运行项目程序。

- 方法 1：在项目所在终端执行命令 `./mul 2 3`。
- 方法 2：在项目所在终端执行命令 `./mul a.txt b.txt`。

并且运行效果都相同，都是 `2 * 3 = 6`。

问题 输出 调试控制台 终端

- root@MaysternLaptop:/home/cpp_fall2022/project01# g++ calculator.cpp -o mul
- root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 2 3
`2 * 3 = 6`
- root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul a.txt b.txt
`2 * 3 = 6`

当输入参数给出的 a, b 均满足：

`a, b` 为可读文本文件，且其中读出的数据（除去文本文件末尾非数字字符以外的所有文本内容）是“大实数”或者 a, b 本身就是“大实数”。

则项目程序认为输入合法，会按照 `x * y = z` 的格式，计算并输出 $a \times b$ 的值。

否则项目程序认为输入不合法，会提示下列错误：

- `The entered numbers should be missing!` 输入的参数缺失。
- `Too many numbers entered!` 输入的参数过多。
- `The input cannot be interpret as numbers!` 输入的数字格式错误，不是两个数。
- `The entered number has too many significant digits!` 为了避免由于输入整数有效位数过大（项目程序默认最大有效位数 `longestSignDigit = 1e6`）导致程序运行缓慢，当输入的数字有效位数过大时，程序会报出此错误并终止运行。
- `The filename "result.txt" cannot be entered as one of the arguments!` 作为输入参数的文件名不能是 `result.txt`，因为该文件保存计算的答案文件。

输出答案的约束

形如 `x * y = z` 格式的输出，将会先写入项目文件目录下的 `result.txt`，然后在终端输出。

Hint: 由于文件读写速度远远快于在终端上输出数字的速度，因此当终端打印结果较慢时，建议您查看项目文件目录下的 `result.txt` 文件查看结果。

对于 x, y, z 中的每一个数字分别考虑，有两种输出方法：

1. 常规记数法输出：输出一个整数，或者一个形如 `整数.非负数` 的非科学计数法小数。
2. 科学计数法输出：形如 `整数e整数` 或者 `非科学计数法小数e整数`，并且 `e` 之前的 `整数` 或者 `非科学计数法小数` 部分的 `绝对值` 在 $[1, 10)$ 。

如果这个数字本身的类型就是整数类型，那么我们将会直接采用常规记数法输出。

否则，项目程序会首先计算按照常规记数法输出需要的字符数，若该字符数不大于设定的最长字符数（程序中设置的默认值为 `decimalMeanLen = 20`），将优先采用常规记数法输出。否则，将会采用科学计数法输出。

另外，在输出数字时，项目程序会自动忽略小数点最后的连续 `0` 的部分，以及大整数的前导 `0`，以便于更加精简的表示输出。

项目思路及代码实现

更多代码请见随报告一并提交的 `calculator.cpp`

程序常数设定

本程序依赖于三个常数，`longestSignDigit`（设置有效位数长度）、`decimalMeanLen`（设置使用常规记数法输出小数时的最长字符数）、`PI`（设置 π 的值用于快速傅里叶变换 FFT）

其默认值为：`longestSignDigit = 1e6`、`decimalMeanLen = 20`、`PI = acos(-1)`。



```
1 const int longestSignDigit = 1e6; // 设置有效位数长度
2 const int decimalMeanLen = 20;    // 设置使用常规记数法输出小数时的最长字符数
3 const double PI = acos(-1);      // 设置PI的值用于快速傅里叶变换
```

输入数据及预处理模块

此模块主要进行参数输入、错误检测、数据存储的过程，主要以字符串的形式输入参数，对参数进行错误检测后，将两个数字存储到程序中大实数类中，为后续的两数相乘模块进行前期准备。

参数输入

主函数应当采用如下的格式，可以在运行程序时获得输入的参数：



```
1 int main(int argc, char* argv[]) {  
2     // ...  
3     return 0;  
4 }  
5
```

参数检测

以字符串的形式输入参数，依次进行如下检测：

1. 输入参数的个数是否恰好为 2 个，是否过多或者过少。
 1. 参数过多：输出 `Too many numbers entered!`。
 2. 参数过少：输出 `The entered numbers should be missing!`
2. 如果输入参数是文件地址，首先检测该文件地址是否为 `result.txt`，如果是，则输出 `The filename "result.txt" cannot be entered as one of the arguments!`；否则将这个文本文件打开，并读入数据作为输入参数。
3. 输入参数的格式是否为整数、小数点表示法表示的小数、科学计数法表示的小数。如果输入参数的格式出现错误，则输出 `The input cannot be interpret as numbers!`
4. 判断输入数字的实际有效位数是否过长（项目程序默认最大有效位数 `longestSignDigit = 1e6`）。如果任一输入的数字的有效位数大于 `longestSignDigit`，则输出 `The entered number has too many significant digits!`

在主函数中，可以使用这样的代码进行参数检测与错误信息输出：



```
1 // 参数错误检测，并输出错误信息。
2 int paramState = paramTest(argc, argv);
3 switch (paramState) {
4     case 1:
5         std::cout << "Too many numbers entered!" << std::endl;
6         break;
7     case 2:
8         std::cout << "The entered numbers should be missing!" << std::endl;
9         break;
10    case 3:
11        std::cout << "The input cannot be interpret as numbers!" << std::endl;
12        break;
13    case 4:
14        std::cout << "The entered number has too many significant digits!" << std::endl;
15        break;
16    case 5:
17        std::cout << "The filename \"result.txt\" cannot be entered as one of the arguments!" << std::endl;
18        break;
19 }
20 if (paramState != 0) return 0;
```

这里使用到了参数合法性检测函数 `paramTest`：



```
1 int paramTest(int argc, char *argv[]) {
2     /*
3         本函数实现参数合法性检测：
4         1. 输入参数过多：返回1
5         2. 输入参数过少：返回2
6         3. 输入参数格式错误，不为数字或者可打开的文件：返回3
7         4. 输入数据的有效位数大于longestSignDigit，返回4
8         5. 输入数据为文件名"result.txt"，返回5
9     */
10    if (argc > 3) return 1;
11    if (argc < 3) return 2;
12    for (int i = 1; i < argc; i++) {
13        if ((std::string) argv[i] == "result.txt") return 5;
14        int fileState = fileTest(argv[i]);
15        if (fileState == 0) continue;
16        if (fileState == 1) return 3;
17        if (fileState == 2) return 4;
18    }
19    return 0;
20 }
```

参数合法性检测函数 `paramTest` 中使用到了 `fileTest` 函数（检测文件内文件是否存在，其内文本是否为实数）



```
1 int fileTest(char *numFile) {
2     /*
3         检测字符数组 file 表示文件是否存在，其内容是否为合法的数字：
4         1. 若 file 表示文件不存在：返回1。
5         2. 若 file 文件中的内容不是数字：返回1。
6         3. 若 file 文件中的数字有效位数大于longestSignDigit：返回2。
7         4. 若 file 文件存在且其内容是合法的数字：返回0。
8     */
9     std::string fileName = numFile, num_str = "";
10    if (FILE *file = fopen(fileName.c_str(), "r")) {
11        char ch;
12        while (!feof(file)) {
13            ch = fgetc(file);
14            num_str += ch;
15        }
16        fclose(file);
17        int endNumPos = 0;
18        for (int i = 0; i < num_str.length(); i++)
19            if (isdigit(num_str[i])) endNumPos = i;
20        num_str = num_str.substr(0, endNumPos + 1);
21    } else {
22        num_str = numFile;
23    }
24    return numTest(num_str);
25 }
```

`fileTest` 需要判断一个字符串是否能够表示为一个实数，所以需要使用到 `numTest` 函数（分四种情况讨论字符串是否表示为一个实数）。



```
1 int numTest(std::string num) {
2     /*
3         检测一个字符串 num 是否为数字，即满足如下格式：
4         1. 整数：一个长整数，如-1189203, 1232
5         2. 小数1：整数.非负整数，如-11.123, 2.0
6         3. 小数2：整数e整数，如1e-12390
7         4. 小数3：小数e整数，如-1.2380192e-12830, -12.31e128301, 0.190123e-1238902, 0.008190238e123
8         如果字符串 num 不满足上述两种格式，返回1；
9         如果字符串 num 表示的数字有效位数大于longestSignDigit，返回2。
10        否则返回0。
11    */
12    int num_e = 0, pos_e = -1;
13    int num_dot = 0, pos_dot = -1;
14    for (int i = 0; i < num.length(); i++) {
15        if (num[i] == 'e') num_e++, pos_e = i;
16        if (num[i] == '.') num_dot++, pos_dot = i;
17    }
18    if (num_e > 1 || num_dot > 1) return 1;
19    if (num_e == 0 && num_dot == 0 && intTest(num, false)) {
20        return signDigitTest(num, pos_e);
21    }
22    if (num_e == 0 && num_dot == 1) {
23        std::string part1 = num.substr(0, pos_dot), part2 = num.substr(pos_dot + 1);
24        if (intTest(part1, false) && intTest(part2, true)) {
25            return signDigitTest(num, pos_e);
26        }
27    }
28    if (num_e == 1 && num_dot == 0) {
29        std::string part1 = num.substr(0, pos_e), part2 = num.substr(pos_e + 1);
30        if (intTest(part1, false) && intTest(part2, false)) {
31            return signDigitTest(num, pos_e);
32        }
33    }
34    if (num_e == 1 && num_dot == 1) {
35        std::string part1 = num.substr(0, pos_e), part2 = num.substr(pos_e + 1);
36        if (floatTest(part1) && intTest(part2, false)) {
37            return signDigitTest(num, pos_e);
38        }
39    }
40    return 1;
41 }
```

`numTest` 函数基于三个函数实现，`intTest`（检测字符串是否为整数） `floatTest` （检测字符串是否为使用常规记数法表示的实数） `signDigitTest` （检测合法数字的有效位数是否超过设定值）

这三个函数分别实现如下：



```
1 bool intTest(std::string num, bool sign) {
2     /*
3         检测字符串num是否为整数,
4         输入参数sign = false 表示对符号无限制, sign = true 表示符号必须为正号 (或者0)
5         返回值为true, 则表示数字为满足sign限制的整数,
6         返回值为false, 则表示不是。
7     */
8     if (num == "-" || num == "+") return false;
9     if (num.length() == 0) return false;
10    if (sign == true && num[0] == '-') return false;
11    if (!(num[0] == '+' || num[0] == '-' || isdigit(num[0]))) return false;
12    for (int i = 1; i < num.length(); i++)
13        if (!isdigit(num[i])) return false;
14    return true;
15 }
16 bool floatTest(std::string num) {
17     /*
18         检测字符串num是否为采用常规记数法表示的小数,
19         返回值为true, 则表示时采用常规记数法表示的小数,
20         返回值为false, 则表示不是。
21     */
22     if (num.length() == 0) return false;
23     int pos_dot;
24     for (int i = 0; i < num.length(); i++)
25         if (num[i] == '.') {
26             pos_dot = i;
27             break;
28         }
29     std::string part1 = num.substr(0, pos_dot), part2 = num.substr(pos_dot + 1);
30     return (intTest(part1, false) && intTest(part2, true));
31 }
32 int signDigitTest(std::string num, int pos_e) {
33     /*
34         检测大实数num的有效位数是否大于设定的最长有效位数长度longestSignDigit
35         如果 num 的有效位数大于longestSignDigit的值则返回2, 否则返回0
36     */
37     if (pos_e == -1) pos_e = num.length();
38     int signDigit = 0, staPos = -1;
39     for (int i = 0; i < pos_e; i++) {
40         if (num[i] >= '1' && num[i] <= '9') {
41             staPos = i;
42             break;
43         }
44     }
45     if (staPos == -1) return 0;
46     for (int i = staPos; i < pos_e; i++)
47         if (num[i] >= '0' && num[i] <= '9')
48             signDigit++;
49     if (signDigit > longestSignDigit) return 2;
50     return 0;
51 }
```

数据存储

参数通过检测后, 将需要相乘的两个数存在大实数类 **bigReal** 中。



```
1 // 数据存储
2 bigReal num1(loadFileData(argv[1]));
3 bigReal num2(loadFileData(argv[2]));
```

其中 `loadFileData` 函数是将文件或者数字转换为 `string` 类型，并返回。



```
1 std::string loadFileData(char *numFile) {
2     /*
3         将字符串实数信息
4         (如果检测到是可以打开的文件名, 读出文件中的所有值作为返回值; 否则直接返回这个数字)
5         转换为string类型。
6     */
7     std::string fileName = numFile, num_str = "";
8     if (FILE *file = fopen(fileName.c_str(), "r")) {
9         char ch;
10        while (!feof(file)) {
11            ch = fgetc(file);
12            num_str += ch;
13        }
14        fclose(file);
15        int endNumPos = 0;
16        for (int i = 0; i < num_str.length(); i++) {
17            if (isdigit(num_str[i])) endNumPos = i;
18            num_str = num_str.substr(0, endNumPos + 1);
19        } else {
20            num_str = numFile;
21        }
22        return num_str;
23    }
```

大实数类 `bigReal` 用来存储一个大实数，需要包含如下信息：

```
struct bigReal {
    /*
    | 实现一个有符号大实数类 bigReal, 支持乘法操作。
    */
    bool type;           // 保存大实数的数据类型, type = 0 表示是整数, type = 1 表示是小数。
    bigInt effNum = 0;   // 保存所有有效位数表示的大整数。
    bigInt power = 0;   // 保存该大实数的幂次值。
    void setNumber(std::string num) { ... }
    void show() { ... }
    std::string to_decimal_string() { ... }
    std::string to_scinote_string() { ... }
    std::string to_string() { ... }
    bigReal(std::string number) { ... }
};

bigReal operator * (bigReal num1, bigReal num2) { ... }
```

- 变量

1. `bool type` 用来保存数字的类型，整数 `type = 0`、小数 `type = 1`。
2. `bigInt effNum` 用来保存大实数的所有有效数字，其中 `bigInt` 是一个大整数类（后面会进一步说明）。

3. `bigInt power` 用来保存大实数 10 的幂指数，其中 `bigInt` 是一个大整数类（后面会进一步说明）。



```
1 bool type;           // 保存大实数的数据类型, type = 0 表示是整数, type = 1 表示是小数。
2 bigInt effNum = 0;   // 保存所有有效位数表示的大整数。
3 bigInt power = 0;    // 保存该大实数的幂次值。
```

对于数字 `-3.1415926e-100` 上述信息分别是：

```
type = 1 effNum = (bigInt) "-31415926" power = (bigInt) "-107"
```

- 过程

1. `void show()` 用来打印变量 `type` 、 `effNum` 、 `power` 等，用于调试。
2. `void setNumber(std::string num)` 传入一个 `string` 类型的参数，将当前的大实数设置成表示 `string` 所表示实数的值。



```
1 void setNumber(std::string num) {
2     /*
3         将一个用string表示的大实数存储到当前大实数中
4     */
5     int num_e = 0, pos_e = -1;
6     int num_dot = 0, pos_dot = -1;
7     for (int i = 0; i < num.length(); i++) {
8         if (num[i] == 'e') num_e++, pos_e = i;
9         if (num[i] == '.') num_dot++, pos_dot = i;
10    }
11    if (num_e == 0 && num_dot == 0) {
12        type = 0;
13        effNum.setNumber(num);
14    } else {
15        type = 1;
16        if (num_e == 0 && num_dot == 1) {
17            std::string tmp = "";
18            for (int i = 0; i < num.length(); i++)
19                if (num[i] != '.') tmp += num[i];
20            effNum.setNumber(tmp);
21            power.setNumber(-num.length() + (pos_dot + 1));
22        }
23        if (num_e == 1 && num_dot == 0) {
24
25            effNum.setNumber(num.substr(0, pos_e));
26            power.setNumber(num.substr(pos_e + 1));
27        }
28        if (num_e == 1 && num_dot == 1) {
29            std::string tmp = "";
30            for (int i = 0; i < pos_e; i++)
31                if (num[i] != '.') tmp += num[i];
32            effNum.setNumber(tmp);
33            power.setNumber(-pos_e + (pos_dot + 1));
34            bigInt powerAdd = 0;
35            powerAdd.setNumber(num.substr(pos_e + 1));
36            power += powerAdd;
37        }
38    }
39 }
40 void show() {
41     /*
42         将 bigReal 内部值输出，用于调试方便。
43     */
44     std::cout << "sign = " << effNum.sign << std::endl;
45     std::cout << "type = " << type << std::endl;
46     std::cout << "effNum = " << effNum.to_string() << std::endl;
47     std::cout << "power = " << power.to_string() << std::endl;
48 }
```

- 函数

1. `std::string to_decimal_string()` 以常规记数法方式返回需要输出的字符串。
2. `std::string to_scinote_string()` 以科学记数法方式返回需要输出的字符串。
3. `std::string to_string()` 判断需要使用那种方法输出字符串，并生成输出字符串。



```
1 std::string to_decimal_string() {
2     /*
3         将 bigReal 按照常规记数法输出。
4     */
5     if (effNum.num.size() == 1 && effNum.num[0] == 0) return "0";
6     std::string res = "";
7     int pow = 0;
8     for (int i = (int) power.num.size() - 1; i >= 0; i--)
9         pow = pow * 10 + power.num[i];
10    if (power.sign == 0) {
11        for (int i = (int) effNum.num.size() - 1; i >= 0; i--)
12            res += '0' + effNum.num[i];
13        for (int i = 1; i <= pow; i++)
14            res += '0';
15    } else {
16        std::vector<int> tmp;
17        std::vector<int>().swap(tmp);
18        for (int i = 0; i < effNum.num.size(); i++)
19            tmp.push_back(effNum.num[i]);
20        for (int i = 1; i <= pow - ((int)effNum.num.size() - 1); i++)
21            tmp.push_back(0);
22        int pos;
23        for (int i = 0; i < pow; i++)
24            if (tmp[i] != 0 || i == pow - 1) {
25                pos = i;
26                break;
27            }
28        for (int i = pos; i < pow; i++) res += tmp[i] + '0';
29        res += '.';
30        for (int i = pow; i < tmp.size(); i++)
31            res += tmp[i] + '0';
32        reverse(res.begin(), res.end());
33    }
34    if (effNum.sign == 1) res = "-" + res;
35    return res;
36}
37 std::string to_scientific_string() {
38     /*
39         将 bigReal 按照科学记数法输出。
40     */
41     if (effNum.num.size() == 1 && effNum.num[0] == 0) return "0";
42     bigInt pow = power + effNum.num.size() - 1;
43     std::string res = "";
44     bool check_no_dot = true;
45     for (int i = (int) effNum.num.size() - 2; i >= 0; i--)
46         if (effNum.num[i] != 0)
47             check_no_dot = false;
48     res = std::to_string(effNum.num[effNum.num.size() - 1]);
49     if (!check_no_dot) {
50         res += '.';
51         int pos = 0;
52         for (int i = 0; i <= (int) effNum.num.size() - 2; i++)
53             if (effNum.num[i] != 0) {
54                 pos = i;
55                 break;
56             }
57         for (int i = (int) effNum.num.size() - 2; i >= pos; i--)
58             res += effNum.num[i] + '0';
59     }
```

```

60     res += 'e';
61     res += pow.to_string();
62     if (effNum.sign == 1) res = "-" + res;
63     return res;
64 }
65 std::string to_string() {
66     /*
67         自动选择将 bigReal 是按常规记数法还是科学记数法输出。
68     */
69     if (effNum.num.size() == 1 && effNum.num[0] == 0) return "0";
70     if (type == 0) return effNum.to_string();
71     bigInt pow = power, ws = 0;
72     if (power.sign == 0) {
73         ws = effNum.num.size() + pow;
74     } else {
75         pow.sign = 0;
76         if (absCmp(pow, effNum.num.size()) >= 0) ws = pow + 2;
77         else ws = effNum.num.size() + 1;
78         int cnt = 0;
79         for (int i = 0; i < (int) effNum.num.size() - 1; i++) {
80             if (effNum.num[i] != 0) break;
81             cnt++;
82         }
83         if (absCmp(pow, cnt) <= 0) ws = ws - pow;
84         else ws = ws - cnt;
85     }
86     if (absCmp(ws, decimalMeanLen) <= 0) return to_decimal_string();
87     else return to_scinote_string();
88 }

```

- 构造函数

1. **bigReal(std::string number)** 传入一个 `string` 类型的参数，构造当前的大实数设置成表示 `string` 所表示实数的值。



```

1  bigReal(std::string number) {
2      /*
3          将一个用string表示的大实数构造到当前大实数中
4      */
5      setNumber(number);
6 }

```

- 运算符

1. **bigReal operator * (bigReal num1, bigReal num2)** 支持两个 `bigReal` 做乘法操作。



```
1 bigReal operator * (bigReal num1, bigReal num2) {
2     /*
3         计算两个大实数 bigReal 的乘法
4     */
5     num1.type |= num2.type;
6     num1.effNum *= num2.effNum;
7     num1.power += num2.power;
8     return num1;
9 }
10
```

大实数类 `bigReal` 中所使用到的大整数类 `bigInt` 是用来存储一个大整数的，需要包含如下信息：

```
struct bigInt {
    /*
        实现一个有符号大整数类 bigInt，支持加、乘操作。
    */
    bool sign; // 大整数的符号，sign = true 表示该整数为负数，sign = false 表示该整数的0或者负数
    std::vector<int>num; // 大整数的有效值
    bigInt& checkCarry() { ... }
    void clear() { ... }
    void setNumber(long long number) { ... }
    void setNumber(std::string number) { ... }
    bigInt(long long number) { ... }
    bigInt(std::string number) { ... }
    std::string to_string() { ... }
};

int absCmp(const bigInt &a, const bigInt &b) { ... }
bigInt absSub(bigInt a, bigInt b) { ... }
struct complex { ... };
void FFT(std::vector<complex> &y, int len, int on) { ... }
bigInt absMul(bigInt a, bigInt b) { ... }
bigInt& operator += (bigInt &a, const bigInt &b) { ... }
bigInt& operator *= (bigInt &a, const bigInt &b) { ... }
bigInt operator + (bigInt a, const bigInt b) { ... }
bigInt operator - (bigInt a, bigInt b) { ... }
bigInt operator * (bigInt a, bigInt b) { ... }
```

- 变量

1. `bool sign` 用来保存大整数的符号，`sign = 0` 表示正数，`sign = 1` 表示负数。
2. `std::vector<int>num` 用来保存大整数的数字。



```
1 bool sign; // 大整数的符号，sign = true 表示该整数为负数，sign = false 表示该整数的0或者负数
2 std::vector<int>num; // 大整数的有效值
```

- 过程

1. `void clear()` 用来清空当前这个大整数，并释放内存。



```
1 void clear() {
2     /*
3         清除大整数本身并释放空间。
4     */
5     sign = 0;
6     std::vector<int>().swap(num);
7 }
```

2. `void setNumber(long long number)` 传入一个 `long long` 类型的参数, 将当前的大整数设置成表示 `long long` 所表示实数的值。



```
1 void setNumber(long long number) {
2     /*
3         将该大整数的值设为 long long 数 number 的值。
4     */
5     clear();
6     if (number == 0) {
7         sign = 0;
8         num.push_back(0);
9         return;
10    }
11    if (number < 0)
12        sign = 1, number = -number;
13    else
14        sign = 0;
15    while (number > 0) {
16        num.push_back(number % 10);
17        number /= 10;
18    }
19    checkCarry();
20 }
```

3. `void setNumber(std::string number)` 传入一个 `string` 类型的参数, 将当前的大整数设置成表示 `string` 所表示实数的值。



```
1 void setNumber(std::string number) {
2     /*
3         将该大整数的值设为 string 数 number的值。
4     */
5     clear();
6     if (number[0] != '-' && number[0] != '+') number = "+" + number;
7     if (number[0] == '-') sign = 1;
8     else sign = 0;
9     for (int i = 1; i < number.size(); i++) {
10        num.push_back(number[i] - '0');
11    }
12    reverse(num.begin(), num.end());
13    checkCarry();
14 }
```

- 函数

1. `bigInt& checkCarry()` 检查并且更新当前大整数的进位情况，将自己的地址传出。



```
1 bigInt& checkCarry() {
2     /*
3         检查进位、去除前导0，返回bigInt本身
4         请注意，0在此处表示为 +0
5     */
6     while (!num.empty() && !num.back()) num.pop_back();
7     if (num.empty()) {
8         sign = 0;
9         num.push_back(0);
10        return *this;
11    }
12    for (int i = 1; i < num.size(); i++) {
13        num[i] += num[i - 1] / 10;
14        num[i - 1] %= 10;
15    }
16    while (num.back() >= 10) {
17        num.push_back(num.back()/10);
18        num[num.size() - 2] %= 10;
19    }
20    return *this;
21 }
```

2. `std::string to_string()` 以 `string` 类型输出当前大整数表示的数。



```
1 std::string to_string() {
2     /*
3         以string类型格式将表示的整数返回
4     */
5     std::string tmp = "";
6     for (int i = 0; i < num.size(); i++)
7         tmp += num[i] + '0';
8     reverse(tmp.begin(), tmp.end());
9     if (sign) tmp = "-" + tmp;
10    return tmp;
11 }
```

- 构造函数

1. `bigInt(long long number)` 传入一个 `long long` 类型的参数，构造当前的大实数设置成表示 `long long` 所表示实数的值。
2. `bigInt(std::string number)` 传入一个 `string` 类型的参数，构造当前的大实数设置成表示 `string` 所表示实数的值。



```
1 bigInt(long long number) {
2     /*
3         构造该大整数的值为 long long 数 number 的值
4     */
5     setNumber(number);
6 }
7 bigInt(std::string number) {
8     /*
9         构造该大整数的值为 string 数 number 的值
10 */
11    setNumber(number);
12 }
```

- 运算符

1. `imbigReal& operator += (bigInt &a, const bigInt &b)` 计算大整数 $a + b$ 的值，并将其赋值到 a 中。



```
1 bigInt& operator += (bigInt &a, const bigInt &b) {
2     /*
3         计算 a + b 并将其值存放到a中
4     */
5     if (a.sign == b.sign) {
6         if (a.num.size() < b.num.size()) a.num.resize(b.num.size());
7         for (int i = 0; i < b.num.size(); i++) a.num[i] += b.num[i];
8         return a.checkCarry();
9     } else {
10         if (a.sign == 1) {
11             int sign;
12             if (absCmp(a, b) <= 0) sign = 0;
13             else sign = 1;
14             a = absSub(b, a);
15             a.sign = sign;
16             return a;
17         } else {
18             int sign;
19             if (absCmp(a, b) >= 0) sign = 0;
20             else sign = 1;
21             a = absSub(a, b);
22             a.sign = sign;
23             return a;
24         }
25     }
26 }
```

2. `bigInt& operator *= (bigInt &a, const bigInt &b)` 计算大整数 $a \times b$ 的值，并将其赋值到 a 中。该运算符的重载依赖于额外的函数 `FFT` 和自定义的额外复数类 `complex`（代码会在后面给出）。



```
1 bigInt& operator *= (bigInt &a, const bigInt &b) {
2     /*
3         计算 a * b 并将值存放到a中
4     */
5     int sign = a.sign ^ b.sign;
6     a = absMul(a, b);
7     a.sign = sign;
8     return a;
9 }
```

3. `bigInt operator + (bigInt a, const bigInt b)`
4. `bigInt operator - (bigInt a, bigInt b)`
5. `bigInt operator * (bigInt a, bigInt b)`



```
1  bigInt operator + (bigInt a, const bigInt b) {
2      // 计算 a + b
3      return a += b;
4  }
5  bigInt operator - (bigInt a, bigInt b) {
6      // 计算 a - b
7      b.sign = 1 - b.sign;
8      return a += b;
9  }
10 bigInt operator * (bigInt a, bigInt b) {
11     // 计算 a * b
12     return a *= b;
13 }
```

此外，由于上面 `bigInt` 类中的数要做一些运算，我们定义了一些额外的函数：

- `int absCmp(const bigInt &a, const bigInt &b)` 比较两个大整数 `bigInt` 绝对值的大小，如果 $|a| < |b|$ 则返回 `-1`，如果 $|a| = |b|$ 则返回 `0`，如果 $|a| > |b|$ 则返回 `1`。



```
1  int absCmp(const bigInt &a, const bigInt &b) {
2      /*
3          比较两个bigInt的绝对值的大小：
4          1. |a| < |b| 返回 -1
5          2. |a| = |b| 返回 0
6          3. |a| > |b| 返回 1
7      */
8      if (a.num.size() > b.num.size()) return 1;
9      if (a.num.size() < b.num.size()) return -1;
10     for (int i = (int)a.num.size() - 1; i >= 0; i--) {
11         if (a.num[i] > b.num[i]) return 1;
12         if (a.num[i] < b.num[i]) return -1;
13     }
14     return 0;
15 }
```

- `bigInt absSub(bigInt a, bigInt b)` 计算两个大整数 `bigInt` 绝对值差的绝对值，即 $||a| - |b||$ 。



```
1  bigint absSub(bigInt a, bigInt b) {
2      /*
3          求两个bigInt绝对值差的绝对值，即 ||a| - |b|| 
4      */
5      if (absCmp(a, b) < 0) std::swap(a, b);
6      for (int i = 0; i < b.num.size(); a.num[i] -= b.num[i], i++)
7          if (a.num[i] < b.num[i]) {
8              int j = i + 1;
9              while (!a.num[j]) j++;
10             while (j > i) {
11                 --a.num[j];
12                 a.num[--j] += 10;
13             }
14         }
15     return a.checkCarry();
16 }
```

- `bigint absMul(bigInt a, bigInt b)` 计算两个大整数 `bigInt` 绝对值的乘积，即 $|a| \times |b|$ 。
此函数依赖于手写的复数类 `complex` 和一个快速傅里叶变换函数 `FFT`。



```
1 struct complex {
2     /*
3         手动实现一个复数类，支持复数加减乘三种操作。
4     */
5     double r; // 表示复数的实部
6     double i; // 表示复数的虚部
7     complex() { }
8     complex(double x, double y) {
9         r = x;
10        i = y;
11    }
12    inline complex operator *(const complex&x) const {
13        return complex(r*x.r - i*x.i, r*x.i + i*x.r );
14    }
15    inline void operator *=(const complex&x) {
16        *this = *this * x;
17    }
18    inline complex operator +(const complex&x) const {
19        return complex(r + x.r, i + x.i);
20    }
21    inline complex operator -(const complex&x) const {
22        return complex(r - x.r, i - x.i);
23    }
24 };
25 void FFT(std::vector<complex> &y, int len, int on) {
26     /*
27         快速傅里叶变换，用于实现快速乘法。
28     */
29     for (int i = 1, j = len / 2; i < len - 1; i++) {
30         if (i < j) std::swap(y[i], y[j]);
31         int k = len / 2;
32         while (j >= k) {
33             j -= k;
34             k /= 2;
35         }
36         if (j < k) j += k;
37     }
38     for (int h = 2; h <= len; h <<= 1) {
39         complex wn(cos(-on * 2.0 * PI / h), sin(-on * 2.0 * PI / h));
40         for (int j = 0; j < len; j += h) {
41             complex w(1, 0);
42             for (int k = j; k < j + h / 2; k++) {
43                 complex u = y[k];
44                 complex t = w * y[k + h / 2];
45                 y[k] = u + t;
46                 y[k + h / 2] = u - t;
47                 w *= wn;
48             }
49         }
50     }
51     if (on == -1) {
52         for (int i = 0; i < len; i++)
53             y[i].r = y[i].r / len;
54     }
55 }
56 bigInt absMul(bigInt a, bigInt b) {
57     /*
58         采用快速傅里叶变换实现两个大整数 bigInt 绝对值的相乘，即 |a| * |b|，返回一个大整数 bigInt。
59     */
60     int len1 = a.num.size(), len2 = b.num.size(), len = 1;
61     std::vector<complex>x1, x2;
62     std::vector<complex>().swap(x1);
63     std::vector<complex>().swap(x2);
64     while (len < len1 * 2 || len < len2 * 2)
65         len <<= 1;
66     for (int i = 0; i < len1; i++)
67         x1.push_back(complex(a.num[i], 0));
68     for (int i = len1; i < len; i++)
69         x1.push_back(complex(0, 0));
70     for (int i = 0; i < len2; i++)
71         x2.push_back(complex(b.num[i], 0));
```

```

72     for (int i = len2; i < len; i++)
73         x2.push_back(complex(0, 0));
74     FFT(x1, len, 1);
75     FFT(x2, len, 1);
76     for (int i = 0; i < len; i++)
77         x1[i] = x1[i] * x2[i];
78     FFT(x1, len, -1);
79     std::vector<int> sum;
80     std::vector<int>().swap(sum);
81     for (int i = 0; i < len; i++)
82         sum.push_back((int)(x1[i].r + 0.5));
83     for (int i = 0; i < len; i++) {
84         sum[i + 1] += sum[i] / 10;
85         sum[i] %= 10;
86     }
87     len = len1 + len2 - 1;
88     while (sum[len] <= 0 && len > 0) len--;
89     std::vector<int>().swap(a.num);
90     for (int i = 0; i <= len; i++)
91         a.num.push_back(sum[i]);
92     a.sign = 0;
93     return a.checkCarry();
94 }

```

大实数乘法模块

此模块主要进行两个大实类 `bigReal` 的数字 `num1, num2` 进行乘法，通过快速傅里叶变化来实现。

在主函数中，大整数乘法模块只需要使用如下语句即可实现：



```

1 // 数据处理，计算两个大实数的乘法
2 bigReal num3 = num1 * num2;

```

确定符号

`bool effNum.sign` 表示数字的符号，正号 `effNum.sign = 0`，负号 `effNum.sign = 1`。

为了确定大实数 `num1` 乘以 `num2` 结果 `ans` 的符号，可以列出如下的真值表：

<code>num1.effNum.sign</code>	<code>num2.effNum.sign</code>	<code>ans.effNum.sign</code>
0	0	0
0	1	1
1	0	1
1	1	0

所以得到公式 `ans.effNum.sign = num1.effNum.sign ^ num2.effNum.sign` (其中 `^` 为异或运算)。

确定类型

`bool type` 为大实数的数字类型，整数 `type = 0`、小数 `type = 1`。

- 当两个数字都是整数类型，那么答案的类型也应当是整数类型。
- 当两个数字至少有一个是小数类型，那么答案的类型也应当是小数类型。

为了确定大实数 `num1` 乘以 `num2` 结果 `ans` 的类型，可以列出如下的真值表：

<code>num1.type</code>	<code>num2.type</code>	<code>ans.type</code>
0	0	0
0	1	1
1	0	1
1	1	1

所以得到公式 `ans.type = num1.type | num2.type` (其中 `|` 符号为或运算)。

确定有效位数和有效数字

`effNum` 表示大实数的有效数字。

为了确定大实数 `num1` 乘以 `num2` 结果 `ans` 的结果有效值，我们可以将两个数的有效值相乘，得到结果的有效值，即 `ans.effNum = num1.effNum * num2.effNum`，并根据实际得到的结果 `ans.effNum` 计算答案有效值的长度 `ans.effLen`。

考虑到这里 `num1.effNum` 和 `num2.effNum` 长度可能较长。如果采用时间复杂度为 $O(n^2)$ 的朴素做法，会导致项目程序在此处存在 **性能瓶颈**，因此在计算两个大整数乘法时，项目程序采用时间复杂度为 $O(n \log_2 n)$ 的 **快速傅里叶变换算法** 进行优化。（其中 n 表示数字有效值的长度）

确定幂指数

大实数 `num1, num2` 可以分别表示为：

- $\text{num}_1 = \text{num}_1.\text{effNum} \times 10^{\text{num}_1.\text{power}}$
- $\text{num}_2 = \text{num}_2.\text{effNum} \times 10^{\text{num}_2.\text{power}}$

所以，大实数 `num1` 乘以 `num2` 结果 `ans` 的值可表示为：

- $\text{ans} = (\text{num}_1.\text{effNum} \times \text{num}_2.\text{effNum}) \times 10^{\text{num}_1.\text{power} + \text{num}_2.\text{power}}$

所以：`ans.power = num1.power + num2.power`

上述 2.3.2 对应下列代码的第5行，

上述 2.3.1 和 2.3.3 对应下列代码的第5行，

上述2.3.4 对应下列代码的第7行。



```
1 bigReal operator * (bigReal num1, bigReal num2) {
2     /*
3         计算两个大实数 bigReal 的乘法
4     */
5     num1.type |= num2.type;
6     num1.effNum *= num2.effNum;
7     num1.power += num2.power;
8     return num1;
9 }
```

数据输出模块

此模块主要进行计算结果的输出，既在终端输出，也在目录中以文件形式输出。



```
1 // 数据输出
2 FILE *file = fopen("result.txt", "w");
3 fprintf(file, "%s * %s = %s\n", num1.to_string().c_str(),
4         num2.to_string().c_str(),
5         num3.to_string().c_str());
6 fclose(file);
7 std::cout << num1.to_string() << " * " << num2.to_string()
8                         << " = " << num3.to_string()
9                         << std::endl;
10 return 0;
11 }
```

见 2.3 节内容，这里摘抄如下：

形如 $x * y = z$ 格式的输出，将会先写入项目文件目录下的 `result.txt`，然后在终端输出。

Hint: 由于文件读写速度远远快于在终端上输出数字的速度，因此当终端打印结果较慢时，建议您查看项目文件目录下的 `result.txt` 文件查看结果。

对于 x, y, z 中的每一个数字分别考虑，有两种输出方法：

1. 常规记数法输出：输出一个整数，或者一个形如 **整数.非负整数** 的非科学计数法小数。
2. 科学计数法输出：形如 **整数e整数** 或者 **非科学计数法小数e整数**，并且 **e** 之前的 **整数** 或者 **非科学计数法小数** 部分的 **绝对值** 在 $[1, 10)$ 。

如果这个数字本身的类型就是整数类型，那么将会直接采用常规记数法输出。

否则，项目程序会首先计算按照常规记数法输出需要的字符数，若该字符数不大于设定的最长字符数（程序中设置的默认值为 `decimalMeanLen = 20`），将优先采用常规记数法输出。否则，将会采用科学计数法输出。

另外，在输出数字时，项目程序会自动忽略小数点最后的连续 0 的部分，以及大整数的前导 0，以便于更加精简的表示输出。

程序实现效果

项目文件原代码请见 `calculator.cpp`，使用 `g++ calculator.cpp -o mul` 即可生成可执行程序 `mul`。

在终端输入形如 `.\mul a b` 的命令，即可复现程序。

您可以在项目目录下的 `result.txt` 查看运行结果，或者在直接在 `终端` 中查看结果。

接下来的例子是在项目说明文档 `project1.pdf` 所列出的 6 个例子，我们都得到了比较合理的输出。

- 例子 1：两个小正整数的乘法操作

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 2 3
2 * 3 = 6
```

- 例子 2：两个小非科学计数法表示的实数的乘法操作

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 3.1416 2
3.1416 * 2 = 6.2832
```

- 例子 3：两个科学计数法表示的小实数的乘法操作

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 3.1415 2.0e-2
3.1415 * 0.02 = 0.06283
```

- 例子 4：错误：输入不为数字

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul a 2
The input cannot be interpreted as numbers!
```

- 例子 5：两个稍大正整数的乘法操作

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 1234567890 1234567890
1234567890 * 1234567890 = 1524157875019052100
```

- 例子 6：两个科学记数法表示的实数的乘法操作

```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 1.0e200 1.0e200
1e200 * 1e200 = 1e400
```

事实上，项目说明文档 `project1.pdf` 所列出的 6 个例子，并不足以说明项目所做的所有工作。

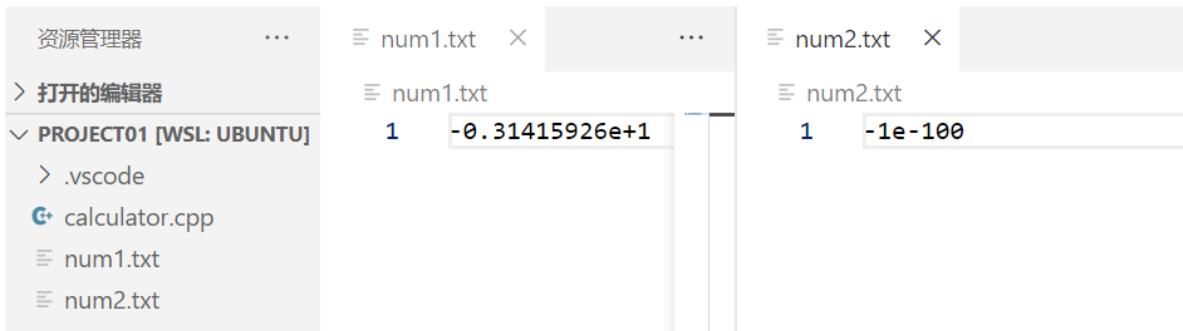
接下来我将会列出几个更加复杂的例子，来更好的说明项目所做的所有工作。

- 例子 7：程序支持一位符号位（正数也可以支持一位的符号位）的输入，使用有效位数更多，值域更大。

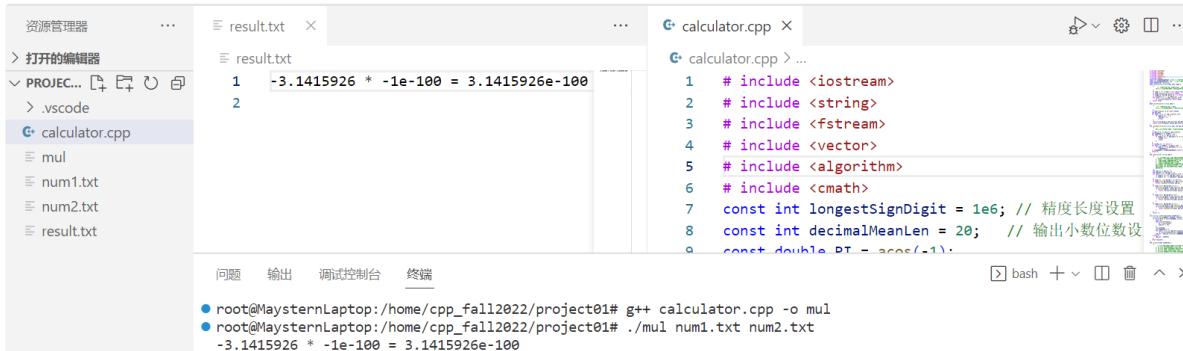
```
问题    输出    调试控制台    终端
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul -1.1283190e28398123 +1231.1231e120381
-1.128319e28398123 * 1.2311231e120384 = -1.3890995850689e28518507
```

- 例子 8：程序支持使用文件打开数据。

如下图所示，在项目目录下，新建有 `num1.txt`（其中内容为 `-0.31415926e+1`）`num2.txt`（其中内容为 `-1e-100`）。

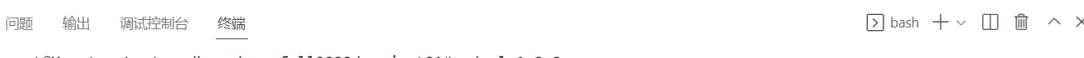


使用命令 `g++ calculator.cpp -o mul` 产生可执行文件 `mul`，并使用命令 `\mul num1.txt num2.txt` 读入数据。可以发现在终端和 `result.txt` 中同时出现计算结果。

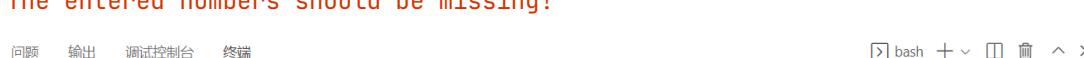


- 例子 9：项目程序认为输入不合法，可能会提示 5 种错误：

Too many numbers entered!



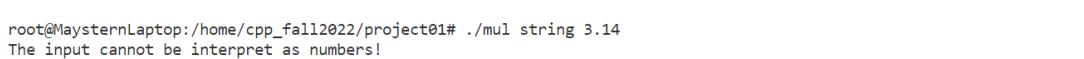
Too many numbers entered!



```
root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 1
```



问题 输出 调试控制台 终端





```
calculator.cpp > ...  
calculator.cpp > ...  
1 # include <iostream>  
2 # include <string>  
3 # include <fstream>  
4 # include <vector>  
mul  
num1.txt  
num2.txt  
result.txt  
问题 输出 调试控制台 终端  
● root@MayternLaptop:/home/cpt/fall2022/project01# ./mul num1.txt string.txt  
The input cannot be interpreted as numbers!
```

The filename "result.txt" cannot be entered as one of the arguments!



The entered number has too many significant digits!

注意这里我们将 `longestSignDigit` 参数的值，从默认值 `1e6` 改成了 `10`，由于输入的 `3.1415926535` 有 `11` 位有效位数，超过了最长的精度长度，所以会报错。

```

1 # include <iostream>
2 # include <string>
3 # include <fstream>
4 # include <vector>
5 # include <algorithm>
6 # include <cmath>
7 const int longestSignDigit = 10; // 精度长度设置
8 const int decimalMeanLen = 20; // 输出小数位数设置

```

问题 输出 调试控制台 终端

root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 3.1415926535 123
The entered number has too many significant digits!

- 例子 10：计算两个大整数（有效位数为 **1e6** 的大整数）的乘法，由于在终端输入的方式太花费时间，这里我们采用使用文件输入的方式输入这两个大整数。

```

num1.txt
1 6095622564079801010888323802902103
9761227417340551351996711813126014
4581000563036687505416439578804253
57582115473446061608039729724197774
515881517134494416135851066934459
2329700266481717842716976811480635
7259715136843908614580311714559496
367892586096857427718624395711070
89259088683537504608207502216408491
0830608173825406615139545966469662

num2.txt
1 139420631180941692028256223903928181374484470539953
880422914236145272842289264352832994396819625364568
9898255507649535821862709948527436834902517869308
1471470856232616044656809715767772406116798234331726
509400176145385118707520256548390250499203733367337
100347192153271440460945393226958915724161689743981
18563615196888937498016024473587565027563641798118
3259725613479426192659030861925471596368727661493
880983062374911281949912890252530538156386799028112
325704921194359364295884786729173644238378003941682

result.txt
1 163578736771791768968088045259290699061913080195501342453779374541551361525246919573945406333869
732027752806071042494509678523562144749594248545434701628278631288920359848178915438504798316424
2286764605394732733277178435296188996854861604490641906947766524455408649679230613421281171808607876
09287472351292508189206758200752799428546980529514788992937291454910788375231886859683147
3115408980377352492399127378610268215228758593108863223001577979160784857502216399923235945405548
07699129067695566854367196321699877302128312068482215306940218831134475331610298697068618213075
2765440937254588636785991183259941381852402682185759703136029181579957254109730326662954462158307
8503175524935451399955782644632133154492098843731395

```

执行计算命令 `./mul num1.txt num2.txt` 后，在终端和 `result.txt` 都很快的输出了结果。

```

result.txt
1 163578736771791768968088045259290699061913080195501342453779374541551361525246919573945406333869
732027752806071042494509678523562144749594248545434701628278631288920359848178915438504798316424
2286764605394732733277178435296188996854861604490641906947766524455408649679230613421281171808607876
09287472351292508189206758200752799428546980529514788992937291454910788375231886859683147
3115408980377352492399127378610268215228758593108863223001577979160784857502216399923235945405548
07699129067695566854367196321699877302128312068482215306940218831134475331610298697068618213075
2765440937254588636785991183259941381852402682185759703136029181579957254109730326662954462158307
8503175524935451399955782644632133154492098843731395

```

值得一提的是，使用 **bigReal** 类的数字进行乘法，可以通过洛谷（一个DSAA 推荐的刷题网站）的 **1e6** 级别的非负整数乘法的题目。

P1919 【模板】A*B Problem 升级版 (FFT 快速傅里叶变换)

[提交答案](#)[加入题单](#)

题目背景

[M+ 复制Markdown](#) [展开](#)

本题数据已加强，请使用 FFT/NTT，不要再交 Python 代码浪费评测资源。

题目描述

给你两个正整数 a, b , 求 $a \times b$.

输入格式

第一行一个正整数，表示 a ;

第二行一个正整数，表示 b 。

输出格式

输出一行一个整数表示答案。

输入输出样例

输入 #1

[复制](#)

```
114514  
1919810
```

输出 #1

[复制](#)

```
219845122340
```

说明/提示

【数据范围】

$1 \leq a, b \leq 10^{1000000}$

可能需要一定程度的常数优化。

数据由 NaCly_Fish 重造

所需要的改动仅仅是将代码的 `int main()` 改变为：

```
1 int main() {  
2     std::string a, b;  
3     std::cin >> a >> b;  
4     bigReal num1(a), num2(b);  
5     std::cout << (num1 * num2).to_string() << std::endl;  
6     return 0;  
7 }
```

下面是通过该题目的截图：

The screenshot shows the submission interface for the problem. It includes tabs for '测试点信息' (Test Point Information) and '源代码' (Source Code). The '测试点信息' tab is active, displaying a table of 10 test points (#1 to #10) with their status as 'AC' (Accepted) and execution time. The '源代码' tab shows the C++ code provided above. To the right, there is a summary box for the submission:

所属题目	P1919 【模板】A*B Problem 升级版 (FFT 快速傅里叶变换)
评测状态	Accepted
评测分数	100
提交时间	2022-09-16 02:59:16

源代码为：

```
calculator.cpp luogu.cpp X
G+ luogu.cpp > main()
505     BigNumber(std::string number) {
506         setNumber(number);
507     }
508 }
509
510 BigNumber operator * (BigNumber num1, BigNumber num2) {
511     num1.type |= num2.type;
512     num1.effNum *= num2.effNum;
513     num1.power += num2.power;
514     return num1;
515 }
516
517 int main() {
518     std::string a, b;
519     std::cin >> a >> b;
520     BigNumber num1(a), num2(b);
521     std::cout << (num1 * num2).to_string() << std::endl;
522     return 0;
523 }
524
```

问题 输出 调试控制台 终端

- root@MaysternLaptop:/home/cpp_fall2022/project01# g++ luogu.cpp -o run && ./run
114514
1919810
219845122340

您可以在这里查阅本题的通过记录: <https://www.luogu.com.cn/record/86711910>

这说明使用 **FFT** 进行的优化是非常有效的，这也可以说明项目程序运行对于大实数有很好的效果。

- 例子 11：程序在输出时，能根据数据的数值，自行选择输出方式是 **常规记数法** 还是 **科学计数法**。这依赖于设置的常规计数法输出的最多字符数 `decimalMeanLen`（默认值为 `decimalMeanLen = 20`）

在下面的例子中，我们更改了最多字符数 `decimalMeanLen` 的值为 5，即如按照常规记数法输出该数字的长度不大于 5，将采用常规记数法输出；否则将会采用科学计数法输出。

需要指出的是，在计算按照常规记数法输出实数需要的字符数的时，算法的时间复杂度只和有效数字的长度线性相关。

```
calculator.cpp X
calculator.cpp > intTest(std::string, bool)
1 # include <iostream>
2 # include <string>
3 # include <fstream>
4 # include <vector>
5 # include <algorithm>
6 # include <cmath>
7 const int longestSignDigit = 1e6; // 精度长度设置
8 const int decimalMeanLen = 5;    // 输出小数位数设置
9 const double PI = acos(-1);
```

问题 输出 调试控制台 终端

- root@MaysternLaptop:/home/cpp_fall2022/project01# g++ calculator.cpp -o mul
- root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 3.1 3.14
3.1 * 3.14 = 9.734
- root@MaysternLaptop:/home/cpp_fall2022/project01# ./mul 12391 123.12
12391 * 1.2312e2 = 1.52557992e6

上述的一些例子可以说明本项目程序的功能。

您也可以尝试更多数据，以测试项目程序的鲁棒性。

项目中出现的难题及解决方案

本次大作业是本课程的第一个大作业，总体而言完成任务的难度不大。

如果只采用 `C++` 中自带的简单数字类型，仅仅是完成任务，不是很有意思。

为了解决 `C++` 中的简单数字类型会存在的有限范围以及精度较低的问题，我设计了大实数类型，可以在 $O(n \log_2 n)$ 的时间复杂度内（其中 n 为大实数类型的有效位数）完成两个大实数的乘法运算。

我将项目主要分成四个部分：数据输入、数据存储、数据处理、数据输出。

在数据输入部分中，最大的难题在于**支持读取文件以获取长度较长的数据和如何判断数据的合法性情况**。

- 支持读取文件以获取长度较长的数据方面，我先将输入的参数视为文件路径，尝试打开这个文件路径，如果不存在该文件，则将该参数视为数字，判断合法性。
- 如何判断数据的合法性情况方面，我设计了 5 种错误，方便进行数据处理前就检查出可能出现错误的情况，提高了程序的鲁棒性。

在数据存储部分中，最大的难题在于**存储大实数的有效值**。我将一个具有有限位数的大实数视为一个大整数乘以 10 的大整数幂次。存储大实数的有效值的问题就转化为存储两个大整数的问题。我还使用 `vector` 方便进行内存管理，尽量减小无效和重复的内存开销。

在数据处理部分中，最大的难题在于**如何快速计算两个大整数的乘法**。这个问题是本项目的时间瓶颈，我查阅了相关资料，采用了快速傅里叶变换的方式，代替朴素的做法，突破了瓶颈，将项目的时间复杂度瓶颈从 $O(n^2)$ 减小到了 $O(n \log_2 n)$ （其中 n 为大实数类型的有效位数）并且在测试中取得了不错的效果。

在数据输出部分中，最大的难题在于**如何让实数按照合理的方式选择输出方式**。我们知道，表示大实数的方式有小数点表示法和科学记数表示法。后者对于任何的大实数都能够工作，而前者与 10 的大整数幂次这个可能很大的值有关。因此，我先尝试采用小数点表示法，如果发现输出非常大（字符数大于设定的 `decimalMeanLen` 默认值为 20）那么采用科学记数表示法，否则有限选用小数点表示法。这样数据显示适应于使用者的阅读习惯，还是非常合理的。

一些问题

虽然此项目在功能上我实现了自认为比较满意的效果。但我认为自己还是在一些方面存在不足，需要在后续的学习和代码时间中进行继续学习和补充：

1. 关于码风：我对 `Google C++ Style` 的理解还并不全面，或许只停留在代码格式的层面上，在后续的学习中我会学习如何写出更加规范的代码，而不是仅仅只注重解决问题。
2. 关于项目管理：由于本项目中“使用 1 个 `C++` 文件进行提交”作为其中一个要求，在未来项目的探索中，为进一步维护项目，应将代码按照功能分为不同的部分，再使用 `makefile` 生成相关可执行文件。
3. 关于指针和内存管理：本项目大量的采用 `vector` 进行数据处理。接下来的学习中，应该避免自己使用过多的 `STL` 容器，而是采用更一般的数据类型，锻炼自己对于指针和内存管理的能力。

后记

因为项目编程和报告撰写所花时间有限，本次项目可能存在许多瑕疵和不足，

非常欢迎您对此提出宝贵的批评与意见。

非常感谢您能看到这里！

