

CS205 C/C++Program Design - Project 1

姓名: 张琦

SID: 12010123

Part 1 - 分析问题

1.1 基本思路:

题目要求我们设计一个整数乘法计算器，且该计算器要能够识别用户的输入是否为整数，还要能对非常大的整数进行乘法计算。

- 解决步骤:

1.为了能够在命令行传入参数，将main函数设置为：`int main(int argc, char **argv)`

2.为了能够接收用户的输入并打印出结果，我们使用 `iostream`。

3.对于乘积不大于 $2^{63}-1$ 且不小于 -2^{63} 的数我们可以选择使用 `long long` 类型整数直接对它们进行乘法，实现一个小范围计算器。

4.编写一个函数来判断一个字符是否为整数。判断一个字符串是否为合法整数实际上就是判断字符串中的每一个字符是否都为整数。其中首位是负号的整数是合法整数，但是首位为0但字符串长度不为1是非法整数（会对后续的大整数乘法造成影响）。在循环中对字符逐个检验，只要有不是0-9之间数字的字符，立即跳出循环并作出判断。

5.字符'0'至'9'对应的ASCII编码依次是48至57，连续的编码可以很好地帮助我们判断该字符是否是整数字符，48和57则是两个判断边界条件。

6.如果要字符串转化为 `long long` 类型整数可以利用 `string` 中的 `c_str()` 方法返回一个指向string对应的char类型数组地址的指针，用 `cstdlib` 中的 `atoll` 函数。

7.在 `while` 循环中执行输入输出可以让用户在输入错误后继续输入直到输入正确。

8.对于非常大的整数，我们可以导入 `string` 利用字符串模拟人工计算乘法的方式对字符串进行乘法，因为此方法的根本原理是将答案的每一个位数分开计算，再对字符进行拼接操作，所以只要用户的整数输入是合法的（000001也是非法输入），无论整数的位数多大，都可以实现两数相乘。

1.2 关于实现超大整数乘法的方法

```
string Multiplication(string a, string b)
```

实现的具体步骤:

思路来源：博客：https://blog.csdn.net/weixin_41376979/article/details/79197186
(仅参考了思路)

1.此函数的目的是传入两个合法的整数字符串后，能够返回他们乘积的字符串。

(由数学知识知：两个整数a和b相乘后得到的整数c的位数一定会比a和b位数的和加一的值小，故设置存储结果的数组的长度为 `int length = a.length() + b.length() + 1;`)

2.先将a和b的根据长度分开，方便后续的循环赋值

3.设置两个bool类型数 judge1 和 judge2，利用 isNegative 函数判断传入的整数字符串是否为负数，利用两个bool类型数储存结果，再利用 string 中的 erase 和 remove 函数将符号从字符串中剔除。

4.利用中的 reverse 函数将两个字符串倒置，使得下标0对应结果的个位，下标1对应结果的十位，以此类推。

5.利用循环模拟人工计算的方法，在数组中从低位向高位乘，在竖式计算中，将乘数第一位与被乘数的每一位相乘，记录结果之后，用第二位相乘，记录结果并且左移一位，以此类推，直到计算完最后一位，再将各项结果相加，得出最后结果。

6.计算完乘法后，对数组进行进位操作。

7.根据前面的判断对结果字符串选择性添加符号。

1.3 使用的头文件及方法：

1.3.1 iostream 的 cin、cout

1.3.2 algorithm 的 erase

1.3.3 cstdlib 的 atoi

1.3.4 string 的 at、c_str、length()、remove

1.3.5 sstream 的 str()

Part 2 - 源代码

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <sstream>
#include <algorithm> //调用字符串反转的函数

using namespace std;

bool isInteger(string a); //函数声明
long long transform(string a);
string IntToString(int &i);
bool isNegative(string a);
string Multiplication(string a, string b);

bool isInteger(string a) //用于判断传入的字符串是否为整数
{
    bool judge = false;
    for (int i = 0; i < a.length(); ++i)
    {
        if (a.at(0) == '0' && a.length() != 1) //排除不正当输入的情况
        {
            return false;
        }
        if (a.at(i) == '-' && a.length() > 1) //a可能会出现负数
        {
            continue;
        }
    }
}
```

```

        else
        {
            if (!(a.at(i) <= '0' || a.at(i) >= '9')) //字符0-9在ASCII中连续出现，包含不在0-9之间的字符就一定不是整数
            {
                judge = true;
                break;
            }
        }
    }
    return judge;
}

long long transform(string a) //将字符串转化为long long类型
{
    long long answer = atoll(a.c_str()); //atoll接收char数组的地址，返回对应的long long类型整数
    return answer; //c_str()函数返回一个指向正规C字符串的指针，内容与本string串相同
}

string IntToString(int &i) //此方法将传入整数类型地址返回相应字符串，摘自博客：
http://www.voidcn.com/article/p-onwlyush-ut.html
{
    string s;
    stringstream ss(s);
    ss << i;
    return ss.str();
}

bool isNegative(string a) //判断一个整数字符串是否是负数
{
    return a.at(0) == '-';
}

string Multiplication(string a, string b) //此时的string a和b一定是正确输入
{
    string longStr = a.length() >= b.length() ? a : b;
    string shortStr = a.length() < b.length() ? a : b;
    bool judge1 = false; //利用逻辑运算用于判断最后的结果是否包含符号
    bool judge2 = false;
    if (isNegative(longStr))
    {
        judge1 = true;
        longStr.erase(remove(longStr.begin(), longStr.end(), '-'),
longStr.end());
    }
    if (isNegative(shortStr))
    {
        judge2 = true;
        shortStr.erase(remove(shortStr.begin(), shortStr.end(), '-'),
shortStr.end());
    }

    reverse(longStr.begin(), longStr.end());
    reverse(shortStr.begin(), shortStr.end());
    int length = a.length() + b.length() + 1;
    int array[length];

```

```

for (int i = 0; i < length; i++) //先把数组完全置0
{
    array[i] = 0;
}

int temp = 0;

for (int i = 0; i < shortStr.length(); i++) //得到位数还没有进位的数组
{
    for (int j = 0; j < longStr.length(); j++)
    {
        array[j + temp] = array[j + temp] + (shortStr.at(i) - 48) *
(longStr.at(j) - 48);
    }
    temp++;
}

for (int i = 0; i < length; i++) //进位操作
{
    if (array[i] >= 10)
    {
        int carry = array[i] / 10;
        array[i] = array[i] % 10;
        array[i + 1] = array[i + 1] + carry;
    }
}

temp = length - 1; //找到第一个非零的数，即为乘积的最大位，temp变量重复使用，节约空间
while (array[temp] == 0)
{
    temp--;
}
string answer;
if (judge1 || judge2) //存在负数
{
    if ((judge1 && !judge2) || (!judge1 && judge2)) //一负一正的情况
    {
        answer = answer + "-";
    }
}

for (temp; temp >= 0; temp--)
{
    answer = answer + IntToString(array[temp]);
}
return answer;
}

int main(int argc, char **argv) //
{
    string a, b;
    if (argc > 1)
    {
        char *var1 = argv[1];
        char *var2 = argv[2];
        a = var1;
        b = var2;
    }
}

```

```

    }
    else
    {
        cout << "Please input two integers" << endl;
        cin >> a;
        cin >> b;
    }

    bool check = true; //尝试直到输入正确为止
    while (check)
    {
        if (isInteger(a) && isInteger(b))
        {
            check = false;
            long long a_Number = transform(a);
            long long b_Number = transform(b);
            cout << "低精度计算器: " << endl;
            cout << a_Number << " * " << b_Number << "= " << a_Number * b_Number
            << endl;

            cout << "高精度计算器: " << endl;
            cout << a << " * " << b << "= " << Multiplication(a, b) << endl;
        }
        else
        {
            cout << "Illegal input , please try again" << endl;
            cin >> a;
            cin >> b;
        }
    }
    return 0;
}

```

Part 3 - 结果检验

测试样例#1:

执行程序后传入两个较小的正整数:

输入: 100 100

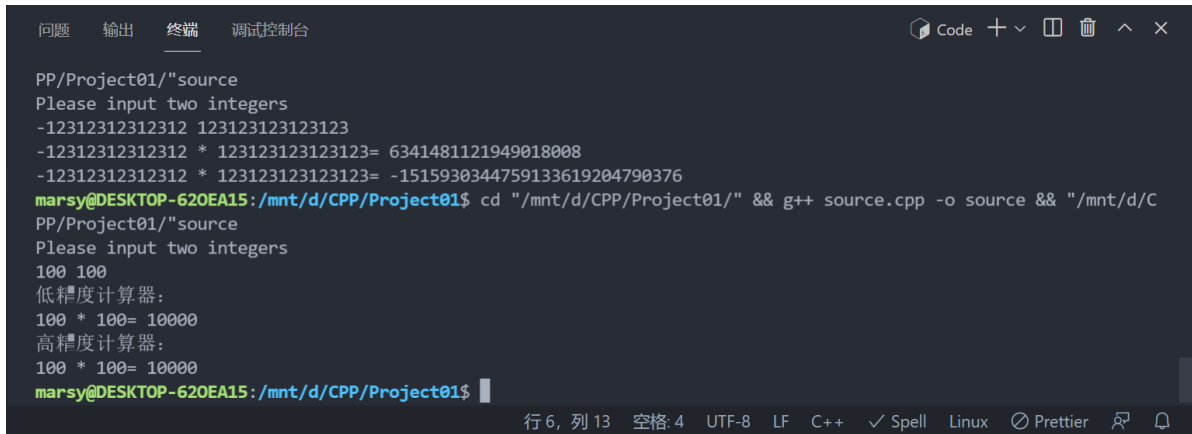
输出: 低精度计算器:

100 * 100= 10000

高精度计算器:

100 * 100= 10000

终端验证：



A terminal window with tabs for '问题', '输出', '终端', and '调试控制台'. The '终端' tab is active. The terminal shows the execution of a C++ program. It prompts for two integers, receives '123123123123' and '100', and then displays the results of calculations for both low and high precision. The high precision result is a large number. The terminal status bar at the bottom shows '行 6, 列 13' and various icons for spell checking and prettier.

```
PP/Project01/"source
Please input two integers
-12312312312312 123123123123123
-12312312312312 * 123123123123123= 6341481121949018008
-12312312312312 * 123123123123123= -1515930344759133619204790376
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$ cd "/mnt/d/CPP/Project01/" && g++ source.cpp -o source && "/mnt/d/C
PP/Project01/"source
Please input two integers
100 100
低精度计算器：
100 * 100= 10000
高精度计算器：
100 * 100= 10000
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

测试样例#2:

执行程序后传入一个较大的正整数，一个较小的正整：

输入：123123123123 100

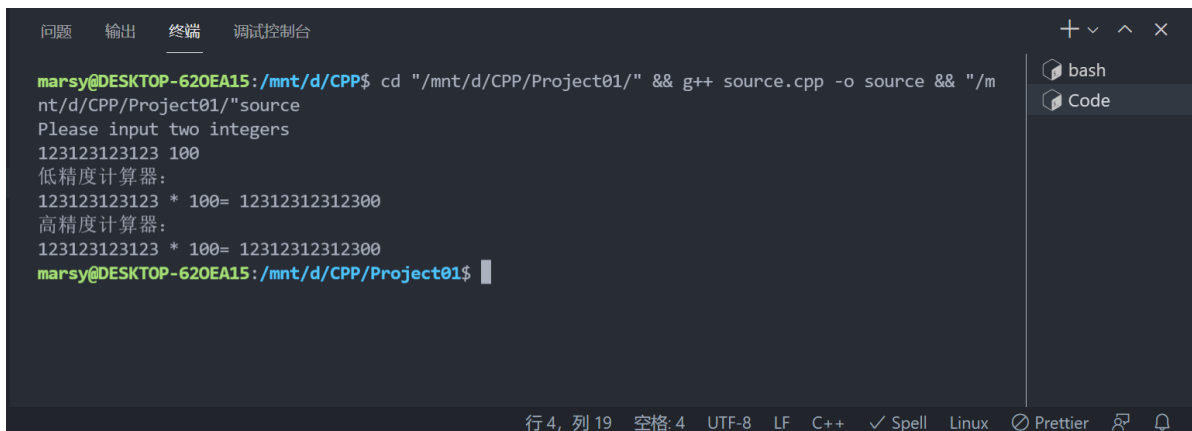
输出：低精度计算器：

123123123123 * 100= 12312312312300

高精度计算器：

123123123123 * 100= 12312312312300

终端验证：



A terminal window showing the execution of the same C++ program with different input. It prompts for two integers, receives '123123123123' and '100', and displays the results. The high precision result is '12312312312300'. The terminal status bar at the bottom shows '行 4, 列 19'.

```
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd "/mnt/d/CPP/Project01/" && g++ source.cpp -o source && "/m
nt/d/CPP/Project01/"source
Please input two integers
123123123123 100
低精度计算器：
123123123123 * 100= 12312312312300
高精度计算器：
123123123123 * 100= 12312312312300
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

测试样例#3:

执行程序后传入两个绝对值较小的负整数：

输入：-123 -234

输出：低精度计算器：

-123 * -234= 28782

高精度计算器：

-123 * -234= 28782

终端验证:



```
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd "/mnt/d/CPP/Project01/" && g++ source.cpp -o source && "/mnt/d/CPP/Project01/"source
Please input two integers
-123 -234
低精度计算器:
-123 * -234= 28782
高精度计算器:
-123 * -234= 28782
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

测试样例#4:

执行程序后传入一个绝对值较大的负整数, 一个绝对值较大的正整数 (没有超过 $2^{63} - 1$)

输入: -123321789 23412636

输出: 低精度计算器:

-123321789 * 23412636= -2887288156725804

高精度计算器:

-123321789 * 23412636= -2887288156725804

终端验证:



```
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd "/mnt/d/CPP/Project01/" && g++ source.cpp -o source && "/mnt/d/CPP/Project01/"source
Please input two integers
-123321789 23412636
低精度计算器:
-123321789 * 23412636= -2887288156725804
高精度计算器:
-123321789 * 23412636= -2887288156725804
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

测试样例#5:

执行程序后传入一个绝对值较大的负整数, 一个绝对值较大的正整数 (超过 $2^{63} - 1$, 此时低精度计算器失效, 高精度计算器仍保持准确)

输入: -123321789123213123 234126363123123123123

低精度计算器：

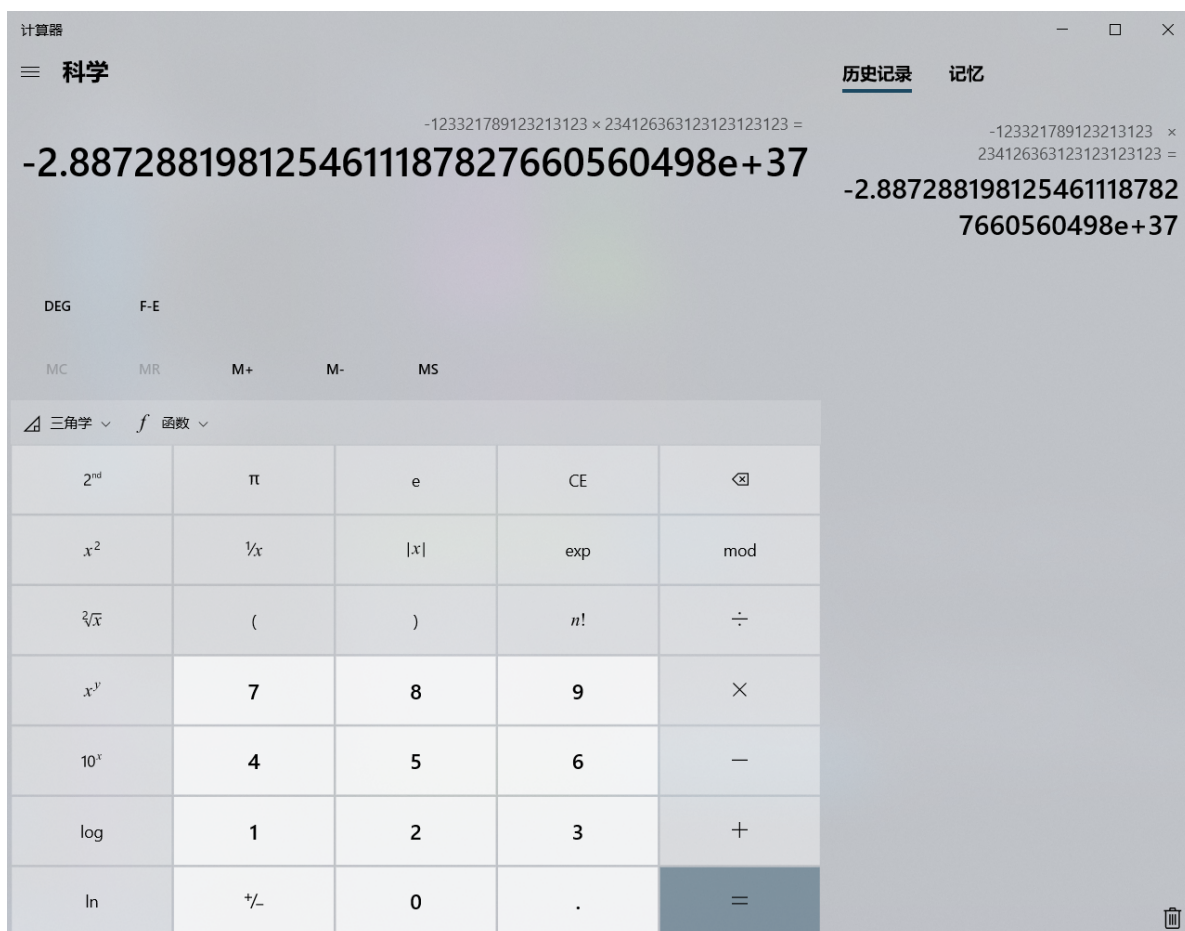
高精度计算器:

-28872881981254611187827660560498343129 (准确)

终端验证:

```
问题 输出 终端 调试控制台
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd "/mnt/d/CPP/Project01/" && g++ source.cpp -o source && "/mnt/d/CPP/Project01/"source
Please input two integers
-123321789123213123 234126363123123123123
低精度计算器:
-123321789123213123 * 9223372036854775807 = -9100050247731562685
高精度计算器:
-123321789123213123 * 234126363123123123123 = -28872881981254611187827660560498343129
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

Win10系统内置科学计算器验证:



测试样例#6:

直接在命令行传入两个很小的正整数（同AS1演示中的2和3）

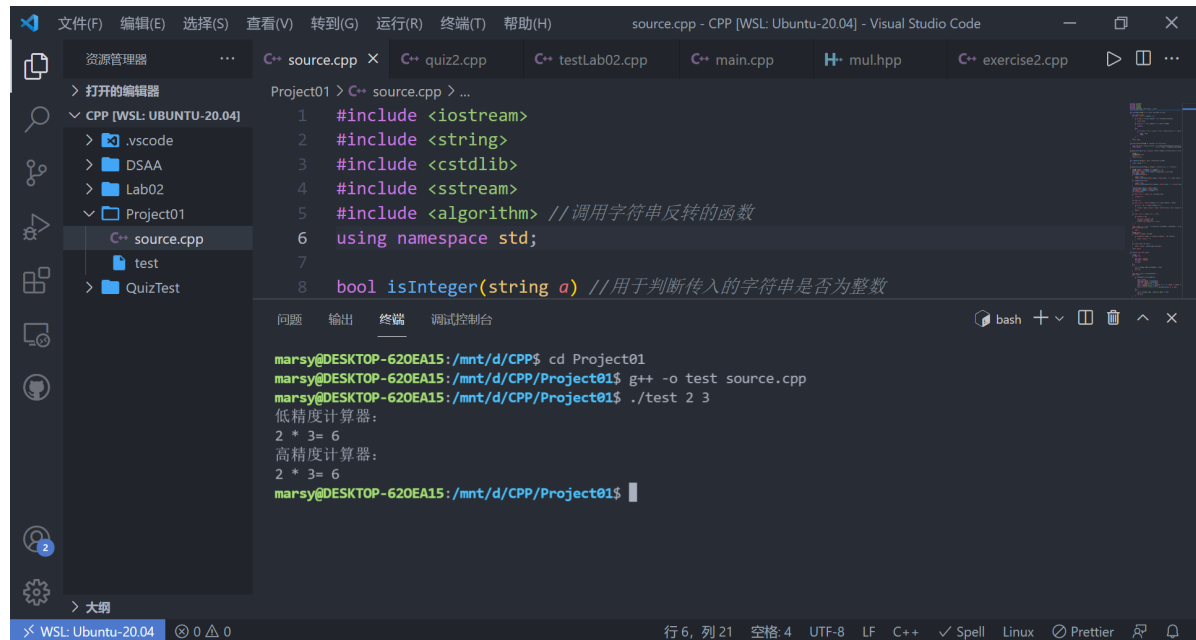
输入: 2 3

输出: 低精度计算器:

$2 * 3 = 6$

高精度计算器:

$2 * 3 = 6$



The screenshot shows the Visual Studio Code interface with a C++ project. The editor displays the following code in `source.cpp`:

```
1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4 #include <sstream>
5 #include <algorithm> // 调用字符串反转的函数
6 using namespace std;
7
8 bool isInteger(string a) // 用于判断传入的字符串是否为整数
```

The terminal window shows the execution of the program:

```
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd Project01
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$ g++ -o test source.cpp
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$ ./test 2 3
低精度计算器:
2 * 3= 6
高精度计算器:
2 * 3= 6
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

测试样例#7:

直接在命令行传入一个很小的负整数，一个很大的正整数（乘积小于 -2^{63} ，此时低精度计算器失效，高精度计算器仍保持准确）

输入: -71623876123712 234623462348352342134

输出: 低精度计算器:

$-71623876123712 * 9223372036854775807 = 71623876123712$ (错误)

高精度计算器:

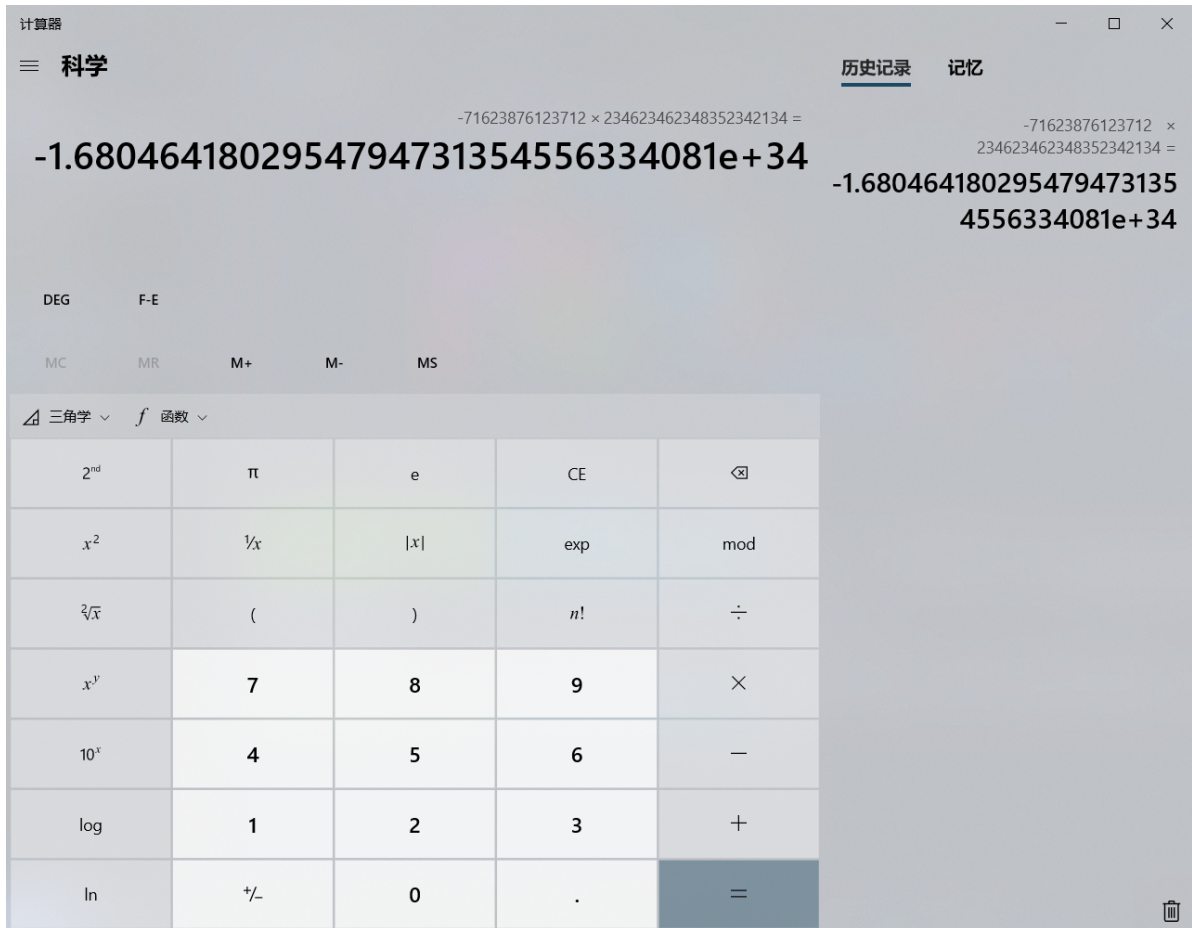
$-71623876123712 * 234623462348352342134 =$

$-16804641802954794731354556334081408$ (正确)

```
Project01 > C++ source.cpp > main(int, char **)
140 {
141     if (isInteger(a) && isInteger(b))
142     {
143         check = false;
144         long long a_Number = transform(a);
145         long long b_Number = transform(b);
146         cout << "低精度计算器: " << endl;
147         cout << a_Number << " * " << b_Number << " = " << a_Number * b_Number << endl;
148         cout << "高精度计算器: " << endl;
149         cout << a << " * " << b << " = " << Multiplication(a, b) << endl;
150     }
}
```

```
marsy@DESKTOP-620EA15:/mnt/d/CPP$ cd Project01
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$ g++ -o test source.cpp
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$ ./test -71623876123712 234623462348352342134
低精度计算器:
-71623876123712 * 9223372036854775807= 71623876123712
高精度计算器:
-71623876123712 * 234623462348352342134= -16804641802954794731354556334081408
marsy@DESKTOP-620EA15:/mnt/d/CPP/Project01$
```

Win10系统内置科学计算器验证：



Part 4 - 遇到的困难及解决方法

1. 在实现大整数乘法的函数中

1.1 如何判断结果的符号

解决方法：设置了两个bool类型数，利用短路求值帮助判断

1.2 如何打印出超过longlong最大值的整数

解决方法：使用数组存储整数的每一个位数，再利用字符串拼接生成结果，避免了越界问题

1.3 如何避免大整数函数接收到非法输入

解决方法：在传入字符串前对字符串进行筛选，保证传入的字符串一定合法

1.4 如何简化模拟乘法算法的设计

解决方法：将传入的字符串在去除符号后倒置，保证下标随位数增大而增大

1.5 如何将整数类型数组转化成字符串

解决方法：利用编写一个传入整数类型地址返回相应字符串的函数，再在循环中调用该函数，对字符进行拼接

2. 在实现判断某个字符串是否为整数的函数中

2.1 判断整数和负数

解决方法：判断字符串的首部是否为符号，且长度是否不为1。

2.2 判断某个字符是否为整数

解决方法：字符0-9在ASCII中连续出现，包含ASCII码不在0-9之间的字符的字符串就一定不是整数。

3.在实现通过命令行传入参数

3.1 区分直接通过命令行传入参数和执行程序后再传参

解决方法：通过argc的大小作判断，大于一则为命令行传参，否则为执行程序后再传参