

# CS205 C/ C++ Programming - Project1

---

**Name:** 王习之 (Wang Xizhi)

**SID:** 11911818

## Part1 - Analysis

---

The first is to consider the input of the problem. After analyzing several types of input of the problem, I decided to divide the input into three types: integer, floating-point number and standard scientific counting method. Given the possibility of value overflow, I chose String to store values and perform operations. I also used regular expressions to split my input to simplify the code. `#include <string> #include <regex>`

Later in the computation, I converted both integers and floating-point number into a scientific representation of e0. In this operation, I multiply the things before e, and add the things after e, which simplifies a lot.

In the String multiplication process, I use the method of traversing each digit, noting the position of the decimal point, and inserting it later in the result.

I used integer or decimal expressions for the final results, and optimized the results. For the results with endings like.0, I optimized the results by removing the decimal point.

## Part2 - Code

---

```
#include <iostream>
#include <string>
#include <regex>

using namespace std;
string a1, a2, b1, b2, mulans;
long long point1, point2, point;
int sign, ae, be, enumber;

int main(int argc, char *argv[])
{
    for (int i = 1; i < argc; i++)
    {
        string a = argv[i];
        regex reg1("\\-?\\d\\.\\d*e\\-?\\d+");
        bool ret1 = regex_match(a, reg1);
        regex reg2("\\-?\\d+\\.\\d+");
        bool ret2 = regex_match(a, reg2);
        regex reg6("\\d+");
        bool ret6 = regex_match(a, reg6);
        if (ret1 == false && ret2 == false and ret6 == false)
        {
            printf("The input cannot be interpret as numbers!\n");
        }
    }
}
```

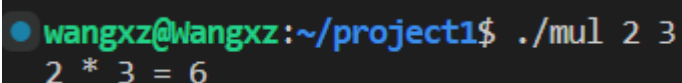
```
        return 0;
    }
    if (a[0] == '-')
    {
        sign++;
        a.erase(a.begin());
    }
    regex reg3("(.*?)\\e(.*?)");
    smatch matchResult;
    bool ret3 = regex_match(a, matchResult, reg3);
    if (i == 1)
    {
        a1 = (ret3 == true) ? matchResult[1] : a;
        a2 = (ret3 == true) ? matchResult[2] : string("0");
        ae = stoi(a2);
        regex reg4("(.*?)\\.(.*?)");
        smatch match;
        bool ret4 = regex_match(a1, match, reg4);
        if (ret4)
        {
            a1 = match[1];
            a1 += match[2];
            point1 = match[2].length();
        }
    }
    else if (i == 2)
    {
        b1 = (ret3 == true) ? matchResult[1] : a;
        b2 = (ret3 == true) ? matchResult[2] : string("0");
        be = stoi(b2);
        regex reg4("(.*?)\\.(.*?)");
        smatch match;
        bool ret4 = regex_match(b1, match, reg4);
        if (ret4)
        {
            b1 = match[1];
            b1 += match[2];
            point2 = match[2].length();
        }
    }
    point = point1 + point2;
    enumber = ae + be;
    int a = a1.size(), b = b1.size();
    vector<int> mul(a + b);
    for (int i = a - 1; i >= 0; i--)
    {
        for (int j = b - 1; j >= 0; j--)
        {
            int answer = int(a1[i] - '0') * int(b1[j] - '0');
            mul[i + j + 1] += answer;
        }
    }
    for (int i = a + b - 1; i >= 1; i--)
```

```
{
    mul[i - 1] += mul[i] / 10;
    mul[i] = mul[i] % 10;
}
for (int i = 0; i < a + b; i++)
{
    mulans += to_string(mul[i]);
}
if (mulans[0] == '0')
{
    mulans.erase(mulans.begin());
}
string Sign = (sign == 1) ? "-" : "";
enumber = enumber - point;
if (enumber >= 0)
{
    mulans += string(enumber, '0');
}
else
{
    long long dis = enumber + mulans.size();
    regex reg10("[0.]*$");
    if (dis <= 0)
    {
        mulans = "0." + string(-dis, '0') + mulans;
        mulans = regex_replace(mulans, reg10, "");
    }
    else
    {
        mulans.insert(dis, ".");
        mulans = regex_replace(mulans, reg10, "");
    }
}
cout << argv[1] << " * " << argv[2] << " = " << Sign << mulans << endl;
return 0;
}
```

## Part3 - Result & Verification

---

Test case #1: Integer multiplication



```
wangxz@wangxz:~/project1$ ./mul 2 3
2 * 3 = 6
```

Test case #2: Floating-point number multiplication

### Test case #3: Not number

### Test case #4: Big integer multiplication

### Test case #5: Big floating-point number multiplication

## Part4 - Difficulties & Solutions

Then after I split the string, I did the multiplication of the string traversal method, and the effect of the decimal point on me was also quite big. I recorded the position of the decimal point, and finally determined whether the decimal point was in the middle, before, or after the end of the result, the output of different processing.

The most useful thing I've learned in this whole process is the regular expression. It's a very convenient way to know if a string is what we want, and it's also a very useful way to extract only what we want.