

CPP_Project1_Multiplication@Jiko

- Name: 纪可鸣
- SID: 12112813

仓库网址: https://github.com/JikoSchnee/SUSTECH_CS205_C-C-Project/tree/main/Project1/code

目录

CPP_Project1_Multiplication@Jiko

目录

I 程序启动

启动方式

方式一: 命令行带输入

方式二: 命令行无输入 (或不规范)

II 数据的输入与处理

数据的输入

数据的存放

数据类型的判断

不同数据类型的处理

翻译

return -1: 非法输入

return 0: 合法字符

return 1: 中文字符

return 2: 科学计数法

III 计算

数组做乘法

正负判断

小数点处理

数组乘法

掐头去尾

IV 输出

V 一些测试数据

I 程序启动

启动方式

方式一: 命令行带输入

情况①: 正常输入

```
1 /mul 1.1234 346435
```

情况②：无输入或输入字符串个数不为2，转到启动方式二

```
1 /mul 12 34 245
```

方式二：命令行无输入（或不规范）

```
1 /mul 12 34 245
```

此时程序会输出特定语句，输入语句后的前两个字符串，

```
1 /mul 12 34 245
2 Please input two numbers need to be multiply(split by " "):
3 123 35 123
```

即第一个数为”123“，第二个数为”35“

代码实现：

```
1 if (argc == 3){ //方式1
2     a = argv[1];
3     b = argv[2];
4 }else{ //方式2
5     cout << "Please input two numbers need to be multiply(split by \" \"):"
6     << endl;
7     string A,B;
8     cin >> A >> B;
9     a = const_cast<char*>(A.c_str()); //把String转成Char*再存入全局变量a、b中
10    b = const_cast<char*>(B.c_str());
11 }
```

II 数据的输入与处理

数据的输入

启动方式1使用 `int main(int argc, char* argv[])` 直接输入两个数据；启动方式2先使用 `string` 存放两个数据再利用 `a = const_cast<char*>(A.c_str());` 转为 `char*`。

数据的存放

因为之前只学过java，所以不知道怎么选择最优的存放方式，在经过一段时间的探索之后决定使用两个全局的 `char*` 来存放输入的字符串，并在之后直接在此上操作。

数据类型的判断

在这一步中使用 `int judgeType(char* obj)` 来判断输入数据的类型，因为输入的类型是无穷无尽的，所以我仅挑选了其中几个比较常见的典型案例，例如中文（一千一百五十一、陆拾壹）、科学计数法（ $1e8$ ， $1e-5$ ， $4e-20$ ）等等，不同类型的数据对应不同的数字输出。

后续有更多的类型只要补充正则以及对应的输出数字就行。

代码实现：

```

1  int judgeType(string obj){
2      int Ecounter = 0;
3      for (int i = 0; i < obj.length(); ++i) {
4          if (obj[i]>=0 && obj[i]<=127){//在ascii码表内
5              if (obj[i]<'0' || obj[i]>'9'){//不为数字
6                  if (obj[i] == 'e'){//计算e的数量
7                      Ecounter++;
8                  }
9              }
10             } else{//含全角字符
11                 return 1;
12             }
13         }
14         if (Ecounter == 1){
15             return 2;
16         }
17         return 0;
18     }

```

不同数据类型的处理

翻译

将所有数据变为只由“0-9”以及小数点“.”和负号“-”组成，将这一数据定为**标准数据**。

return -1: 非法输入

使程序直接输出“The input cannot be interpret as numbers!”

例如：

```

1  /mul 我不是数字 我是汉字
2  The input cannot be interpret as numbers!

```

```

1  /mul %*# 2
2  The input cannot be interpret as numbers!

```

return 0: 合法字符

即输入数据已经满足为**标准数据**，直接进入下一步的运算。

return 1: 中文字符

使用了网上的代码进行转换

<https://blog.csdn.net/u010412858/article/details/80354996>

```

1  int chineseToInt(wstring s)
2  {
3      map<wchar_t, int> chineseNum;
4      chineseNum.insert(pair<wchar_t, int>(L'零', 0));
5      chineseNum.insert(pair<wchar_t, int>(L'一', 1));
6      chineseNum.insert(pair<wchar_t, int>(L'二', 2));
7      chineseNum.insert(pair<wchar_t, int>(L'两', 2));
8      chineseNum.insert(pair<wchar_t, int>(L'俩', 2));
9      chineseNum.insert(pair<wchar_t, int>(L'三', 3));

```

```

10     chineseNum.insert(pair<wchar_t, int>(L'四', 4));
11     chineseNum.insert(pair<wchar_t, int>(L'五', 5));
12     chineseNum.insert(pair<wchar_t, int>(L'六', 6));
13     chineseNum.insert(pair<wchar_t, int>(L'七', 7));
14     chineseNum.insert(pair<wchar_t, int>(L'八', 8));
15     chineseNum.insert(pair<wchar_t, int>(L'九', 9));
16     chineseNum.insert(pair<wchar_t, int>(L'十', 10));
17     chineseNum.insert(pair<wchar_t, int>(L'百', 100));
18     chineseNum.insert(pair<wchar_t, int>(L'千', 1000));
19     chineseNum.insert(pair<wchar_t, int>(L'万', 10000));
20     chineseNum.insert(pair<wchar_t, int>(L'亿', 100000000));
21     chineseNum.insert(pair<wchar_t, int>(L'壹', 1));
22     chineseNum.insert(pair<wchar_t, int>(L'贰', 2));
23     chineseNum.insert(pair<wchar_t, int>(L'叁', 3));
24     chineseNum.insert(pair<wchar_t, int>(L'肆', 4));
25     chineseNum.insert(pair<wchar_t, int>(L'伍', 5));
26     chineseNum.insert(pair<wchar_t, int>(L'陆', 6));
27     chineseNum.insert(pair<wchar_t, int>(L'柒', 7));
28     chineseNum.insert(pair<wchar_t, int>(L'捌', 8));
29     chineseNum.insert(pair<wchar_t, int>(L'玖', 9));
30     chineseNum.insert(pair<wchar_t, int>(L'拾', 10));
31     chineseNum.insert(pair<wchar_t, int>(L'玖', 100));
32     chineseNum.insert(pair<wchar_t, int>(L'仟', 1000));
33     chineseNum.insert(pair<wchar_t, int>(L'萬', 10000));
34     chineseNum.insert(pair<wchar_t, int>(L'億', 100000000));
35
36     int result=0, tmp = 0, hnd_mln=0;
37     wchar_t curr_char;
38     int curr_digit;
39     for (int i = 0; i < s.length(); ++i)
40     {
41         curr_char = s.at(i);
42         if (chineseNum.find(curr_char) == chineseNum.end())
43             return NULL;
44         curr_digit = chineseNum.at(curr_char);
45
46         if (curr_digit == pow(10, 8))//meet 「亿」 or 「億」
47         {
48             result = result + tmp;
49             result = result * curr_digit;
50             //get result before 「亿」 and store it into hnd_mln
51             //reset `result`
52             hnd_mln = hnd_mln * pow(10, 8) + result;
53             result = 0;
54             tmp = 0;
55         }
56         else
57         {
58             if (curr_digit == pow(10, 4))//meet 「万」 or 「萬」
59             {
60                 result = result + tmp;
61                 result = result * curr_digit;
62                 tmp = 0;
63             }
64             else
65             {

```

```

66         if (curr_digit >= 10) //meet 「十」, 「百」, 「千」 or their
traditional version
67         {
68             if (tmp == 0)
69                 tmp = 1;
70             result = result + curr_digit * tmp;
71             tmp = 0;
72         }
73         else
74         {
75             tmp = tmp * 10 + curr_digit;
76             /*if (curr_digit != NULL)
77
78             else
79             {
80                 return result;
81             }*/
82         }
83     }
84 }
85 }
86 result = result + tmp;
87 result = result + hnd_mln;
88 return result;
89 }

```

将对应的字赋予对应的值最后相乘相加，在此基础上我加入了其他口语表达习惯例如“俩”“两”。需要注意的是中文输入需要非常遵守规范，十百千万都需说明清楚，否则判为无法转换。例如：

```

1  /mul 一十 十一
2  The input cannot be interpret as numbers!
3  /mul 二百五 二百五十
4  205 * 250 = 51250
5  /mul 贰佰 叁佰五十
6  200 * 350 = 70000

```

转换后再进入下一步计算。

return 2: 科学计数法

将科学计数法分为两类，一类是+一类是-。

主要思路还是把科学计数法转成**标准数据**，将数据分为两部分，前一部分为part1，后一部分为part2。part2先把后面的数字拼起来然后使用 `stoi()` 变成int类型，名字叫 `Enumber`，如果是正的就开一个长度为Enumber+1的数组，初始化所有格子为0，从前往后填充part1；如果是负的就开一个长度为Enumber+2的数组，初始化所有格子为0，第二位改为小数点，从后往前填充part1。如此一来就可以得到**标准数据**之后再进行下一步计算。

代码实现：

```

1  else if (type == 2) //科学计数法
2      int eLocation = 0;
3      for (int i = 0; i < strlen(obj); ++i) {
4          if (obj[i] == 'e'){
5              eLocation = i;
6          }
7      }

```

```

8         if (obj[eLocation+1] == '-') {
9             char* linshi;
10            for (int i = eLocation, j=0; i < strlen(obj); ++i) {
11                if (obj[i] >= '0' && obj[i] <= '9') {
12                    linshi[j++] = obj[i];
13                }
14            }
15            int Enumber = stoi(linshi);
16            char result[Enumber+2] = {0};
17            result[1] = '.';
18            for (int i = eLocation, j=Enumber+1; i >=0; --i) {
19                if (obj[i] >= '0' && obj[i] <= '9') {
20                    result[j--] = obj[i];
21                }
22            }
23            obj = result;
24        } else {
25            char* linshi;
26            for (int i = eLocation, j=0; i < strlen(obj); ++i) {
27                if (obj[i] >= '0' && obj[i] <= '9') {
28                    linshi[j++] = obj[i];
29                }
30            }
31            int Enumber = stoi(linshi);
32            char result[Enumber+1] = {0};
33            for (int i = 0, j=0; i < eLocation; ++i) {
34                if (obj[i] >= '0' && obj[i] <= '9') {
35                    result[j++] = obj[i];
36                }
37            }
38            obj = result;
39        }

```

四计算

数组做乘法

正负判断

翻译之后所有数据将变为只由“0-9”以及小数点“.”和负号“-”，此时再使用函数来处理数据的正负，若为正，则函数返回true，若为负，则函数返回false，并移除负号。在这时候数据变为无符号数，接下来将两数进行计算。最后在打印时如果两数对应的布尔值不一致则先打印一个负号，反之不打印。

小数点处理

先移除小数点，最后在数组乘法结束后再加上。

例如：346.134613471 * 2313461.234234 = 346134613471 * 2313461234234 / 1e15

数组乘法

在经过前面的**数据处理**、**正负判断**、**小数点处理**后余下的只有0-9共10个数字组成的数，因为可能遇到数据类型无法存放的超大数据，所以我决定采用数组来计算乘法。其原理与列乘法竖式类似，将两数分别存入两个数组，长度分别为 `len1` 和 `len2`，`n`位数与`m`位数相乘最大为`m+n`位数（使用计算器 `99*999`简单验证），因此可以开一个存放结果的数组3，长度为`m+n`，数1的`i`位与数2的`j`位相乘，结果加在数3的`i+j`位，每次运算后把多于10的部分进位，个位数保留，依此类推可以计算 **非常大的数字**。经过小数点处理后也可以处理 **高精度的浮点数**。代码实现：

```
1 void mul(char* number1,char* number2){
2     int len1 = strlen(number1);
3     int len2 = strlen(number2);
4     int len3 = len1+len2;
5     bool n1 = true;
6     bool n2 = true;
7     if (number1[0] == '-') {
8         number1[0] = '0';
9         n1 = false;
10    }
11    if (number2[0] == '-') {
12        number2[0] = '0';
13        n2 = false;
14    }
15    revstr(number1); //翻转char
16    revstr(number2);
17    //查找小数点位置
18    int point = 0;
19    point = findPoint(number1,len1) + findPoint(number2,len2);
20    if (point == 0){
21        ifPoint = false;
22    }else{
23        ifPoint = true;
24    }
25
26    int first[len1] = {0};
27    int second[len2] = {0};
28    int result[len3] = {0};
29    for (int i = 0,j = 0; i < len1; ++i) {
30        if (number1[i] != '.'){
31            first[j++] = number1[i] - '0';
32        }
33    }
34    for (int i = 0,j = 0; i < len2; ++i) {
35        if (number2[i] != '.'){
36            second[j++] = number2[i] - '0';
37        }
38    }
39
40    for (int i = 0; i < len1; ++i) {
41        for (int j = 0; j < len2; ++j) {
42            result[i+j] += first[i] * second[j];
43        }
44    }
45    int up = 0;
46    for (int i = 0; i < len3; ++i) {
```

```

47         result[i] += up;
48         up = result[i] / 10;
49         result[i] = result[i] % 10;
50     }
51
52     char solution[len3]; // 最终结果的数组
53     for (int i = 0, j = 0; i < len3; ++i) {
54         if (point == 0) {
55             solution[i] = result[j++] + '0';
56         } else {
57             if (i != point) {
58                 solution[i] = result[j++] + '0';
59             } else {
60                 solution[i] = '.';
61             }
62         }
63     }
64     // 掐头去尾
65     if (ifPoint) {
66         cut(solution, len3);
67     }
68     revstr(solution);
69     cut(solution, len3);
70     if (n1 != n2) {
71         cout << "-";
72     }
73     for (int i = 0; i < strlen(solution); ++i) {
74         if (solution[i] >= '0' && solution[i] <= '9') {
75             cout << solution[i];
76         } else if (solution[i] == '.') {
77             cout << solution[i];
78         }
79     }
80     cout << endl;
81
82     revstr(number1);
83     revstr(number2);
84     return;
85 }

```

案例：

```

1  /mul 12345 457458536589
2  12345 * 457458536589 = 5647325634191205
3  /mul 346.134613471 2313461.234234
4  346.134613471 * 2313461.234234 = 800769010.091728182766214

```

掐头去尾

数组乘法结束后头尾会有多余的0，此时需要从头到尾和从尾到头一直去除直到该0的下一位是小数点。

代码实现：


```

1 void cut(char* solution,int len3){
2     for (int i = 0; i < len3; ++i) {
3         if ((solution[i] <= '0' || solution[i] > '9' )&& solution[i+1] != '.'){
4             solution[i] = '&';
5         }else{
6             break;
7         }
8     }
9 }

```

刚经过数组乘法的结果还没有翻转，此时使用cut()即从尾到头去0。

之后使用revstr翻转结果，此时使用cut()即从头到尾去0。

需要注意的是若存在小数点且小数点后全部为0，会保留十分位的0来说明进行了浮点运算。

例子：

```

1 /mul 0.2 5
2 0.2 * 5 = 1.0
3 /mul 1.23 23.2
4 1.23 * 23.2 = 28.536

```

IV输出

先使用 `printAB()` 来输出经过 `converter()` 处理后的两个数据及乘号，接着使用 `mul()`（参照数组乘法中的代码，在计算完后会直接输出结果）来输出整个算式：

```

1 void printAB(){
2     for (int i = 0; i < strlen(a); ++i) {
3         cout << a[i];
4     }
5     cout << " * ";
6
7     for (int i = 0; i < strlen(b); ++i) {
8         cout << b[i];
9     }
10    cout << " = ";
11 }

```

V一些测试数据

```

1 /mul 2.1e5 1.1e5
2 210000 * 110000 = 23100000000

```

```

1 /mul 3450s 029845f
2 The input cannot be interpret as numbers!

```

```

1 /mul 134561032451451235 96029436023945620
2 134561032451451235 * 96029436023945620 = 12921820057112706828443579721840700

```

```
1  /mul 六千五百一十三 五千两百二十
2  6513 * 5220 = 33997860
```

```
1  /mul 1e5 2.5
2  100000 * 2.5 = 250000.0
```