矩阵相加和相乘实验报告

目录

[一：需求分析 2](#_Toc87619164)

[题目的目的 2](#_Toc87619165)

[输入的正确形式 2](#_Toc87619166)

[输入的错误形式 2](#_Toc87619167)

[输出样例 3](#_Toc87619168)

[二: 概要设计 5](#_Toc87619169)

[问题解决思路 5](#_Toc87619170)

[主函数伪代码 5](#_Toc87619171)

[class matrix 6](#_Toc87619172)

[函数调用关系 10](#_Toc87619173)

[三：调试分析报告 11](#_Toc87619174)

[输入检测 - getElement() 11](#_Toc87619175)

[四：设计实验的回顾讨论 11](#_Toc87619176)

[五：用户使用说明 12](#_Toc87619177)

[六：测试结果 12](#_Toc87619178)

[七：实验结果 14](#_Toc87619179)

题目：矩阵相加和相乘

班级：2020211306 姓名：马天成 学号：2020211376 分工：撰写代码+实验报告

班级：2020211306 姓名：黄洪建 学号：2020211371 分工：撰写代码

班级：2020211306 姓名：马紫薇 学号：2020211392 分工：撰写实验报告

# 一：需求分析

## 题目的目的

前提：假设稀疏矩阵A和B均以三元组表作为存储结构。

1. 判断是否可以相加相乘。如果不可以，直接返回不能计算。

2. 假如可以相加，另设三元组表C存放相加结果，并输出结果矩阵。

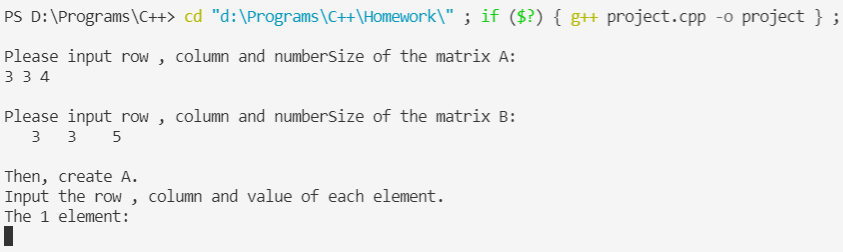
3. 假如可以相乘，另设三元组表C存放相乘结果，并输出结果矩阵。

## 输入的正确形式

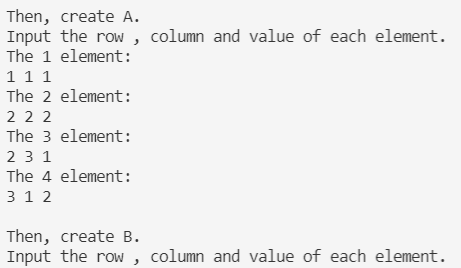
首先，永远不能相信用户的输入。

所以对于所有的输入，我们加了智能的输入机制：忽略前置、后置空格，只需输入整型数据；并且，我们对于溢出的数据会报错并要求重新输入。

1. 输入三元组矩阵的行、列、有效元素个数



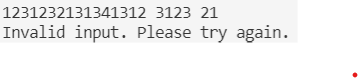
2. 若矩阵不为0矩阵，则依次输入合法元素的行、列、值



## 输入的错误形式

因为每次输入是三元组形式，所以会对溢出的整型、非三个的输入报错并要求重新输入。

**1. 整型溢出**

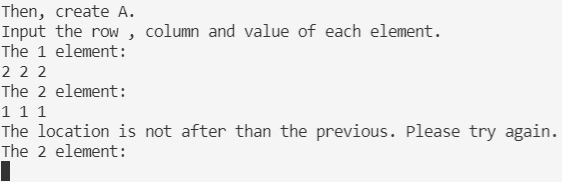


**2. 个数不对**

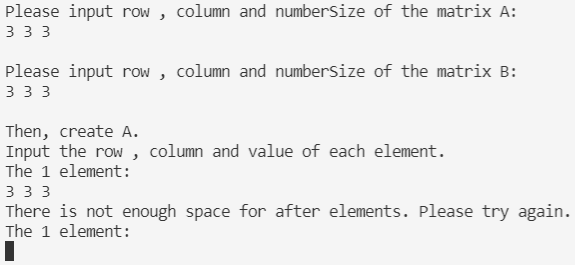




**3. 顺序不对**



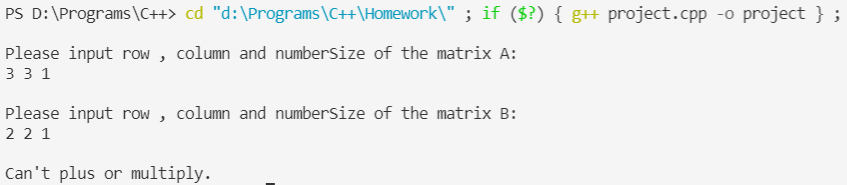
**4. 空间不足**



## 输出样例

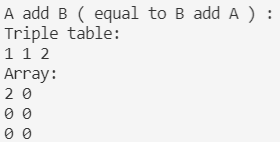
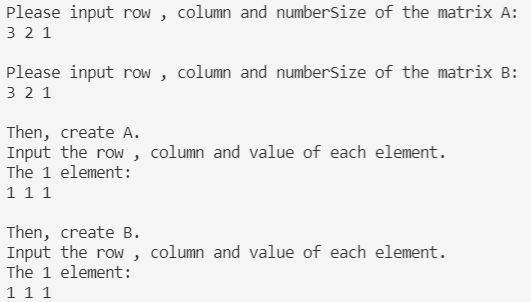
**1. 不可以相加相乘**

ex:不可以相加相乘则不用录入数据，直接返回。



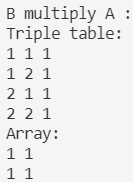
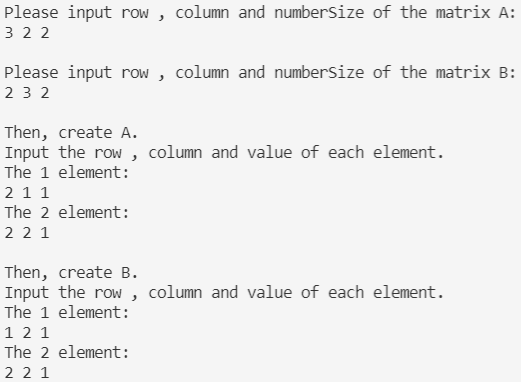
output\_1

**2. 只能相加**



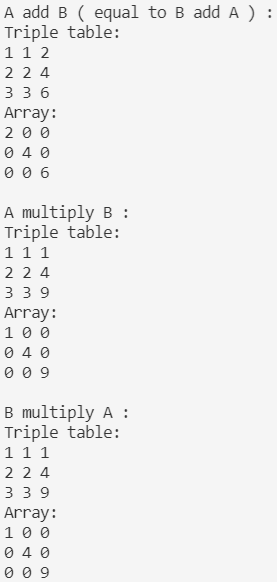
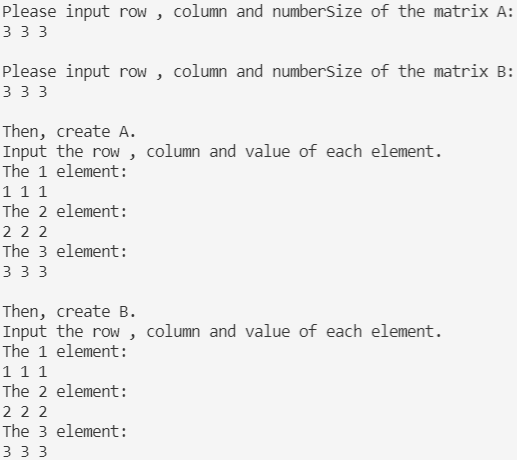
iutput\_2 output\_2

**3. 只能相乘**



input\_3 output\_3

**4. 能相加能相乘**

****

input\_4 output\_4

# 二: 概要设计

## 问题解决思路

1.从键盘输入稀疏矩阵A和B

2.检测A和B能否相加/相乘

3.如能，做矩阵相加和相乘运算，并打印运算结果

4.如不能，应显示出原因

## 主函数伪代码

int main()

{

获取矩阵A和B的大小 ;

if ( 不可计算 ) {

return 0 ;

}

创建矩阵A ;

创建矩阵B ;

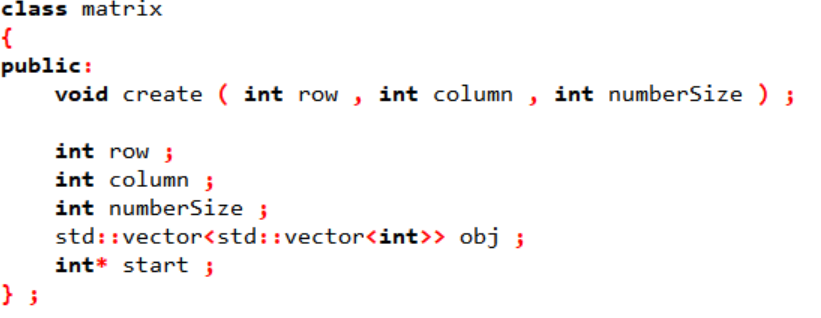
计算 ;

输出结果 ;

return ;

}

## class matrix



* matrix::create()是matrix的生成函数，根据参数调用getElement()来进行合法元素的录入;
* int\* start 是记录行元素起始位置的数组头指针。

1. 矩阵S与T的加法运算



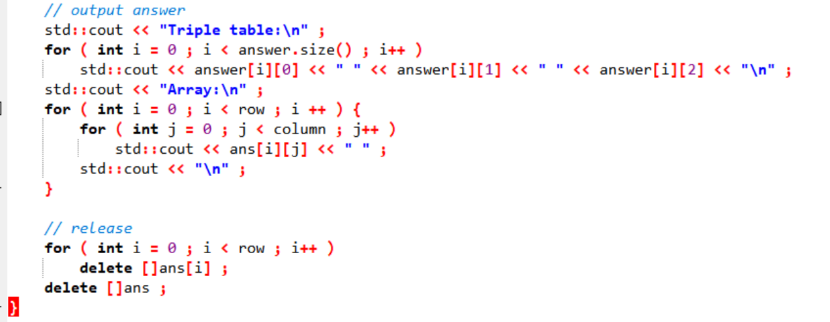
**加法过程**

* 假如A的位置在B之前，则对应位置加上A的值；
* 假如B的位置在A之前，则对应位置加上B的值；
* 假如A和B位置相等，则加上A和B的值。

**输出结果**

* 三元组形式
* 矩阵形式

这会让我们更直观地理解三元组的答案。



**判断条件**

加法运算判断条件是 

如果结果为真，则进行上面的加法运算。

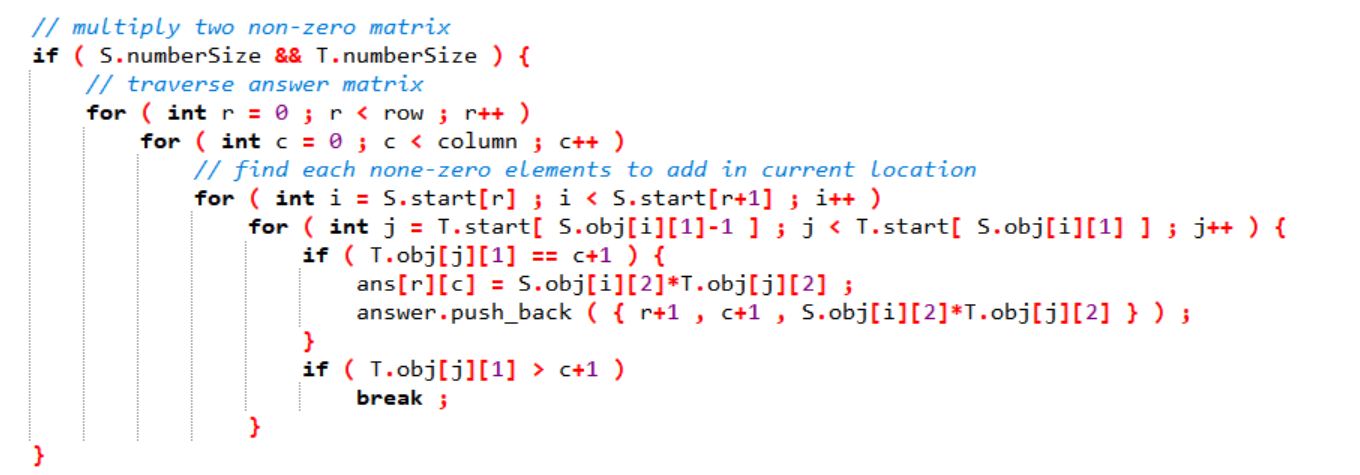
1. 矩阵S与矩阵T的乘法运算

**乘法过程**

对于每一个矩阵的位置 ( i , j ) :

* 遍历左矩阵的i行；
* 对于左矩阵i行每个元素，匹配有矩阵j列的元素，如果乘法位置匹配，则乘法结果加到(i,j) ;
* 如果找到下标过了的，直接跳出循环，匹配左矩阵i行下一个元素。

如果有个矩阵为0矩阵，则直接返回0矩阵。

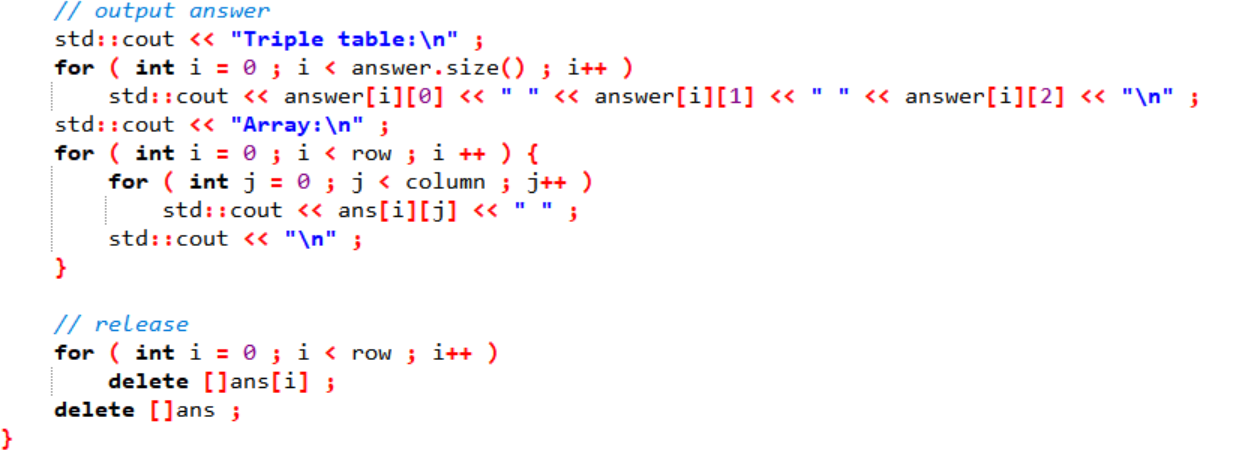


**输出结果**

* 三元组形式
* 矩阵形式

如果可以A\*B则输出A\*B结果；可以B\*A输出B\*A结果。

（下列输出是通用输出，只需把参数顺序调换即可表示两种运算顺序）



**判断条件**

A可以乘B 

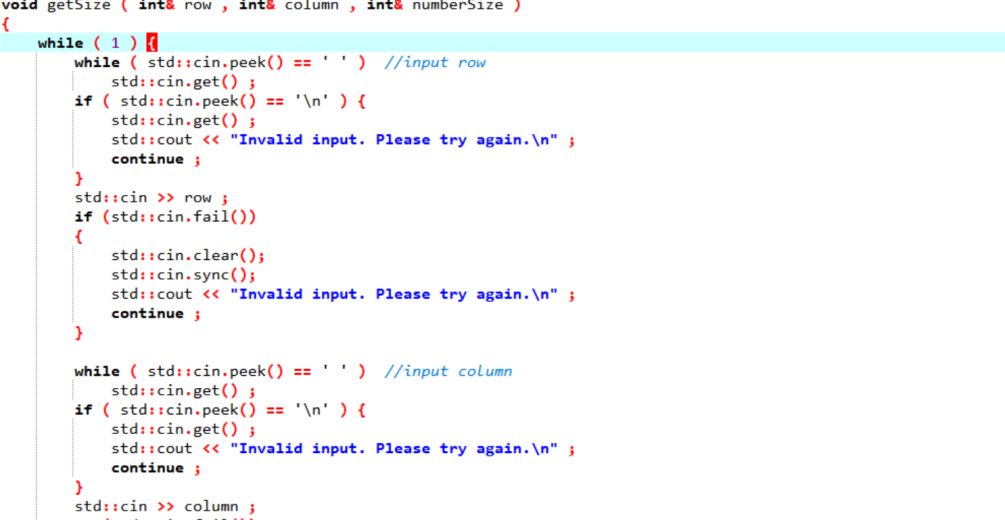
B可以乘A 

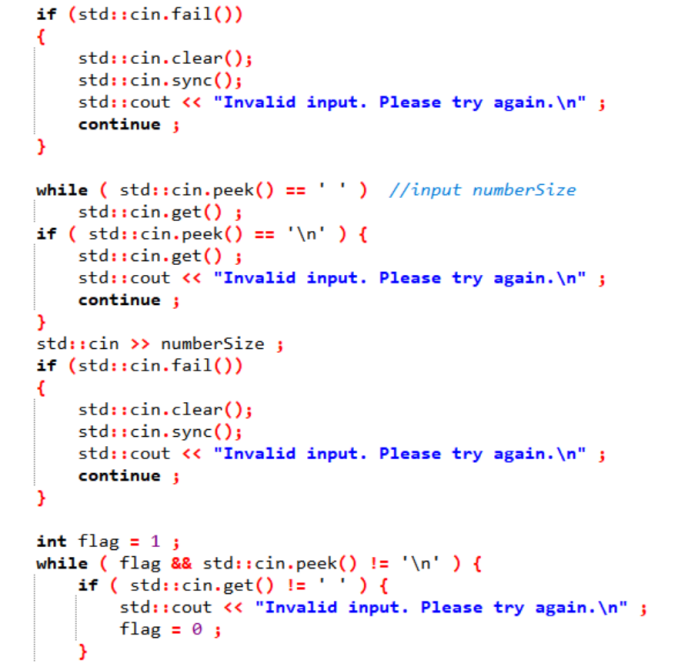
如果可以计算，再计算输出结果。

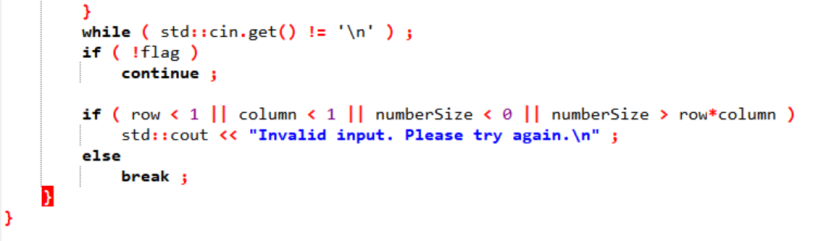
1. 对于矩阵，获取其行、列和有效数字个数

**功能**

* 会依次录入行、列和有效数字个数；
* 会忽略前导0和后导0；
* 对于溢出的整型会报错并要求重新输入；
* 对于大于或小于三个的输入会报错并要求重新输入。







**说明**

我们在输入上花了很大功夫，因为要检测输入。

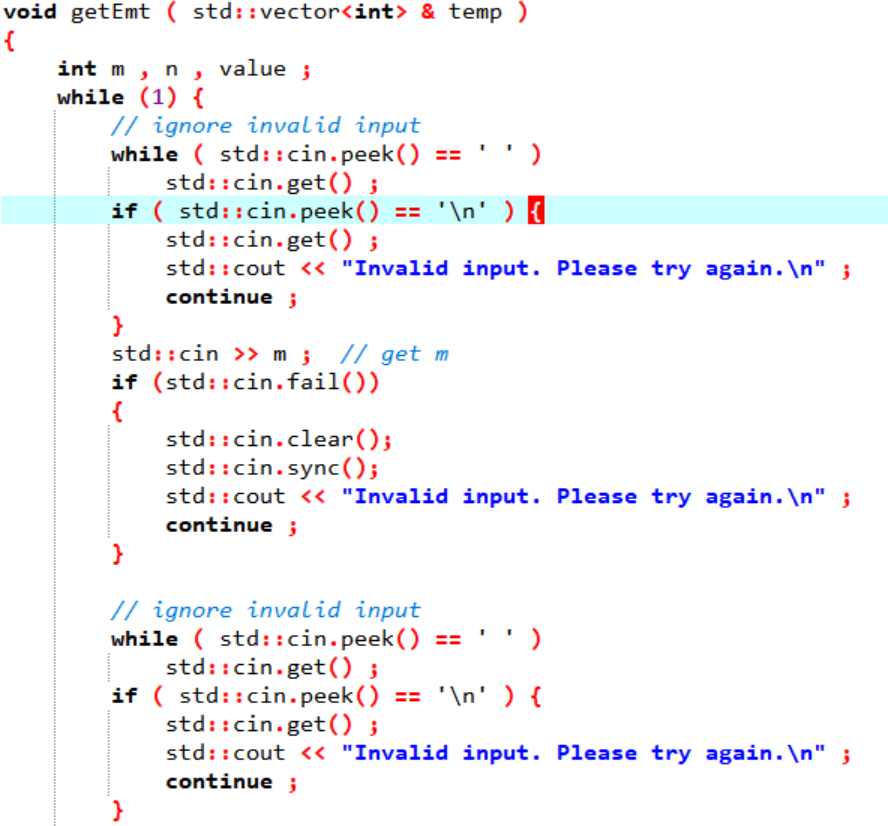
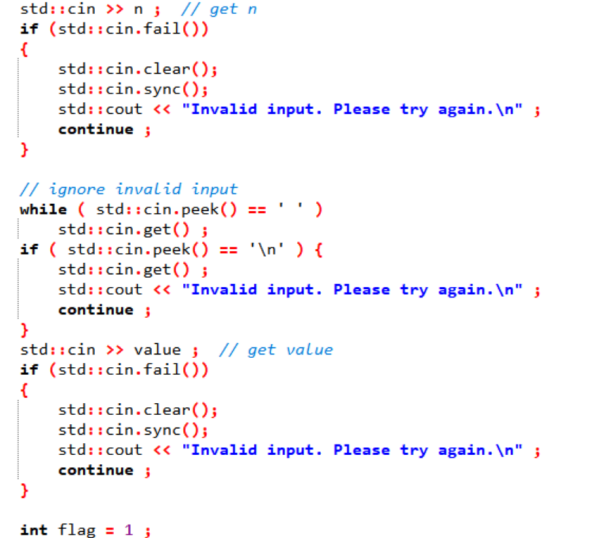
**结果**

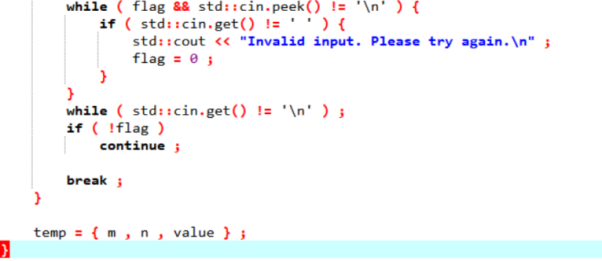
录入matrix的行、列和有效数字个数

1. 对于一个矩阵，得到它的每个元素，包括loc\_row、loc\_col和value

**功能**

* 会依次录入每个元素的loc\_row、loc\_col和value；
* 会忽略前导0和后导0；
* 对于溢出的整型、大于或小于三个的输入会报错并要求重新输入；
* 假如输入位置是在矩阵界外、在前一个元素之前或者没有足够空间留给剩余元素时会报错并要求重新输入。



**结果**

得到matrix的每个元素，包括loc\_row、loc\_col和value

## 函数调用关系

int main()

mul()

matrix::create()

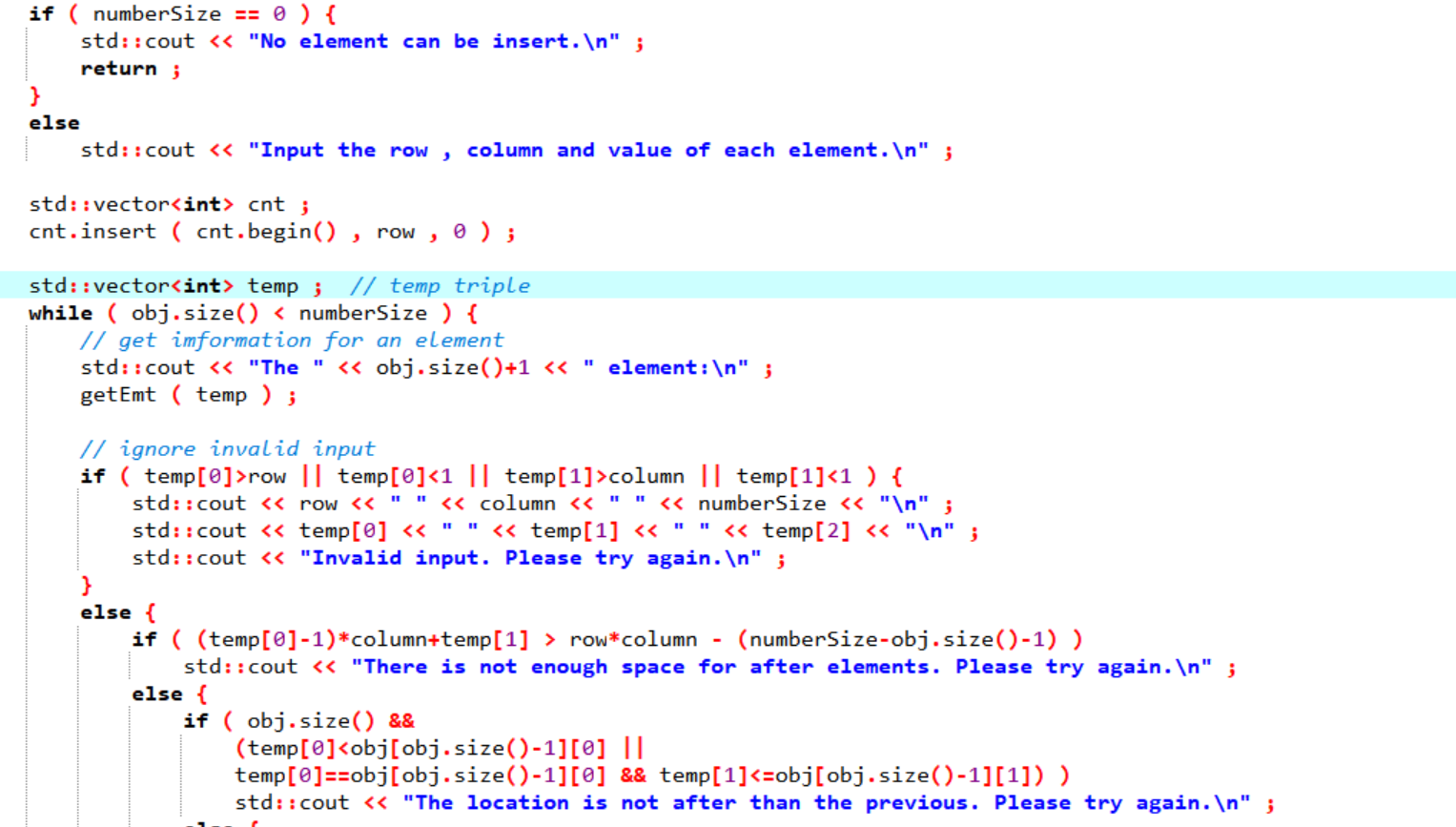
getSize()

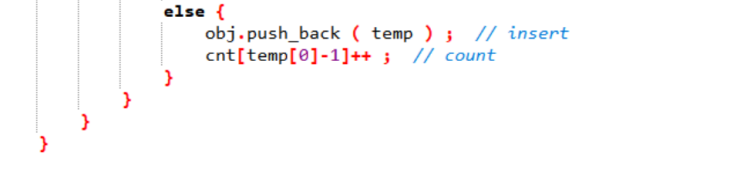
add()

getElement()

# 三：调试分析报告

## 输入检测 - getElement()

****

****

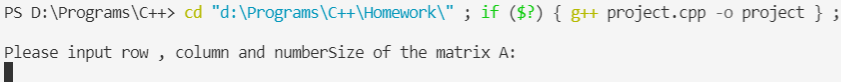
**会分别检测是否元素越界，是否有足够空间和顺序是否规范。**

# 四：设计实验的回顾讨论

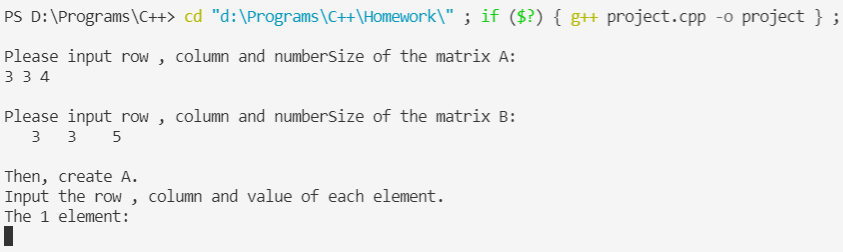
* 在实验之初由于对三元组表存储不熟悉，因此出现了部分错误处理。
* 在实验之初由于对顺序存储的不熟悉，而导致在实现三元组表存储和运用时导致算法低效。
* 在实验过程中计算矩阵加法时由于考虑不周全导致输出时出现低级错误。
* 在测试时发现我们对输入没有考虑周全，仍有许多错误输入没有检测。
* 在测试时发现部分情况下输出会因数值溢出而无法输出。

# 五：用户使用说明

**1.**打开程序，首先会出现如下界面：

****

1. 用户输入合法时将会继续显示：



1. 当输入不合法时，会出现：





用户可根据提示，重新输入相应数值并回车即可。

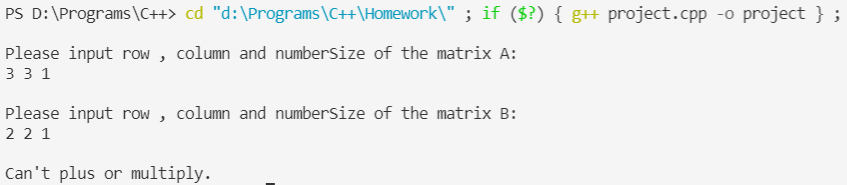
我们花了很长的时间在输入检测上；

现在除了数据相加相乘后结果溢出的问题会存在，其他的应该已经没有问题了；

简而言之，就是：除了溢出，这程序随便怎么输，只要不合法都会报错并要求重新输入。

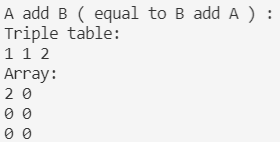
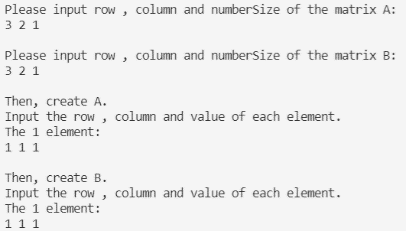
# 六：测试结果

1. **不可以相加相乘**



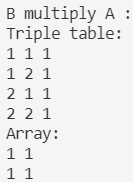
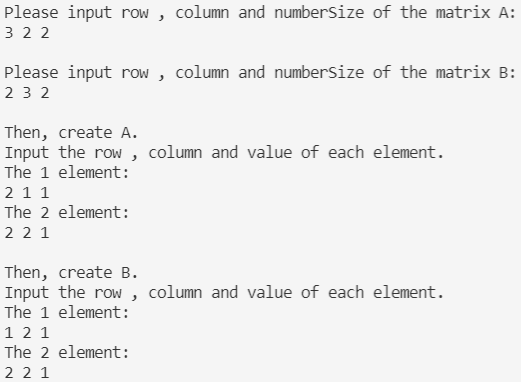
output\_1

**2. 只能相加**



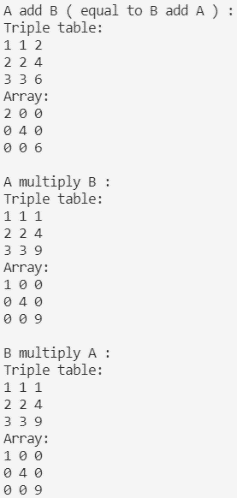
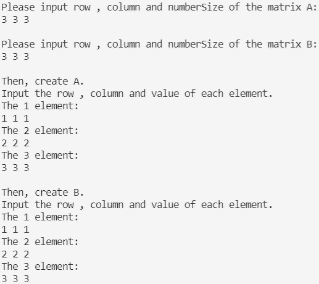
iutput\_2 output\_2

**3. 只能相乘**



input\_3 output\_3

**4. 能相加能相乘**

****

input\_4 output\_4

# 七：实验结果

1. 加深了我对稀疏矩阵和三元组表的理解，成功将理论与实践相结合起来，利用这两个工具成功解决了矩阵相加相乘问题，对于稀疏矩阵的节省空间的特点有了进一步的了解。

2. 深入研究了数组的存储表示和实现技术，熟悉广义表存储结构的特性。

在实验过程中遇到了一些问题，由于对于顺序存储的不熟悉而导致的三元数组的存储和运用的算法效率低问题，在计算矩阵的加法时由于考虑不全面而导致输出时出现低级错误。

3. 这些问题在随后的小组讨论中都得到了及时的纠正，并且在交流中通过交换彼此的思路，拓宽了大家的视野。这次的实验使我们了解到细节的重要性，有时会因为一个极小的错误而导致程序出现低级错误，而且体会到了调试程序的必要性，通过这一步骤，可以使得最终程序得到优化，提高计算效率。