**《数据结构》实验报告**

**学号： 09022107 姓名 ：梁耀欣**

**实验题号：p223.6 实验日期：\_**2023.10.19**\_\_ 实验类型：** 必做

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

这个问题主要围绕建立稀疏矩阵的十字链表来表示一个非零元素很少的非稠密矩阵，用这个方法可以节省矩阵的空间，便于遍历和存储矩阵。矩阵的元素用节点表示，节点之间形成两个循环链表：一个是行链表，一个是列链表。每个节点包含 rowLink、colLink、row、col 和 value 五个数据成员。这些节点以行和列的顺序链接起来，然后再链接到一个通用的头节点。同时，还需要实现一些函数来进行矩阵操作，如加法、减法、乘法、拷贝构造、析构、转置等。在老师课上的讲解后我理解结点有头结点和普通结点两种，头结点通过next连接，right和down分别是对应行和列的指针，由于空间结构的实现过于复杂，我把这个方法改进简化为书上的形式，即分成行链表和列链表两个头结点的指针，具体实现在下面问题中阐明。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

**输入运算符的重载:将元素读取和输入到十字链表中，插入适当的位置，用到了遍历稀疏矩阵链表的算法、循环读取的算法。**

**输出运算符的重载:遍历十字链表，按照给定的输出格式将原本的矩阵输出。**

**拷贝构造:用深拷贝的形式遍历原矩阵的链表并把节点逐个复制到新的矩阵中，与赋值运算符的算法类似。**

**+-\*等运算符重载：首先创建一个新的稀疏矩阵result对象，分别遍历当前矩阵和传入a矩阵的链表，对行和列进行匹配，执行相应的减法、加法、乘法操作。**

**析构函数:遍历链表，首先逐个释放普通节点的内存，然后释放头结点和链表头的内存。**

**转置函数:创建一个新的稀疏矩阵Matrix对象，和此对象的行列数交换，非零元素不变；遍历此稀疏矩阵链表，把行列交换后插入新的稀疏矩阵中，得到矩阵的转置。**

**为了方便写代码，我在矩阵类中添加了一个函数来添加结点，便于在构建矩阵或插入结点时运用大量重复的代码，主要的算法思想就是传入行、列、值，创建一个新节点后赋值，通过行指针找到对应的行位置，依靠列数在适当的位置插入新节点，之后连接列链表，节点的插入就完成了。**

**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

Matrix(int numRows, int numCols, int numElements)：构造函数，用于初始化 Matrix 类的对象。它接受矩阵的行数、列数和非零元素个数，然后动态分配行链表和列链表的头指针向量，并初始化这两个数组。

~Matrix()：析构函数，销毁 Matrix 类的对象。在析构函数中，释放分配的内存，包括节点和头指针向量。

AddNode(int row, int col, ElemType value)：用于向稀疏矩阵中添加一个新节点。根据给定的行、列和值，它会创建一个新节点，并将其插入到正确的位置，同时更新行链表和列链表。

friend istream& operator>>(istream& is, Matrix& matrix)：输入运算符重载，用于从标准输入流中读取矩阵的信息。这个函数首先从输入中读取行数、列数和非零元素个数，然后使用 AddNode 函数将元素添加到矩阵中。

friend ostream& operator<<(ostream& os, const Matrix& matrix)：输出运算符重载，用于将矩阵的内容输出到标准输出流。它遍历稀疏矩阵的行和列，并输出相应的元素。

Matrix operator+(const Matrix& other) const：加号运算符重载，用于执行矩阵相加操作。这个函数遍历两个矩阵并将它们的对应元素相加，然后返回结果矩阵。

Matrix(const Matrix& other)：拷贝构造函数，用于创建一个矩阵的深拷贝。它会复制原始矩阵的数据，并创建一个新的对象。

Matrix& operator=(const Matrix& other)：等号运算符重载，用于执行矩阵的赋值操作。它首先清除当前矩阵的数据，然后复制另一个矩阵的数据。

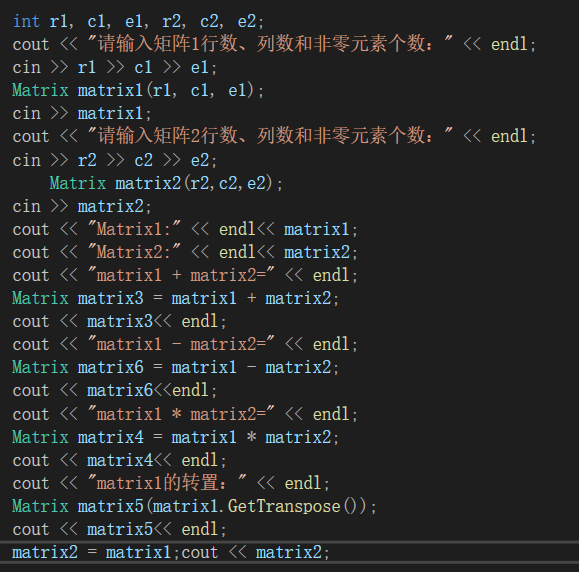
Matrix operator\*(const Matrix& other)：乘号运算符重载，用于执行矩阵相乘操作。它将两个矩阵相乘，然后返回结果矩阵。

Matrix GetTranspose()：求转置函数，用于获取矩阵的转置。它创建一个新矩阵，并将矩阵的行和列互换，然后返回转置后的矩阵。

Matrix operator-(const Matrix& other) const：减号运算符重载，用于执行矩阵相减操作。这个函数遍历两个矩阵并将它们的对应元素相减，然后返回结果矩阵。

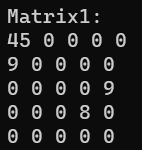
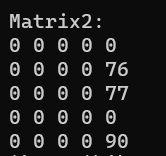
**4．测试数据：**设计测试数据，或具体给出测试数据。

在main函数中创立了几个对象来测试输出结果。



**其中我们通过对matrix1和matrix2的赋值，实现上述功能函数的测试。**

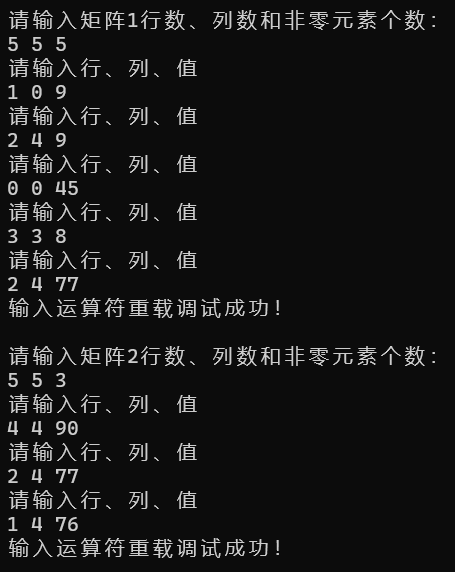
**例如，我们输入两个矩阵：**

其中输入必须用三元组的形式，例如第一个矩阵第一个数：（0，0，45）

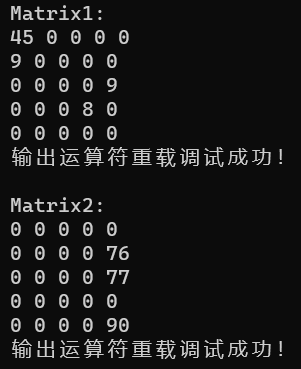
**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

**我把几个功能函数末尾加上一行输出，以显示调用了此函数：**

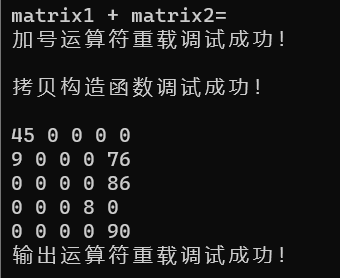


**首先对两个矩阵进行输入，先输入矩阵的规格，即行数、列数、非零元素个数，显示输入运算符调用。**

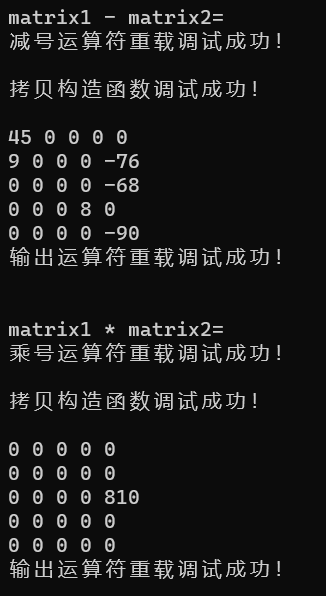
**然后调用输出运算符，矩阵以原本的形式输出：**



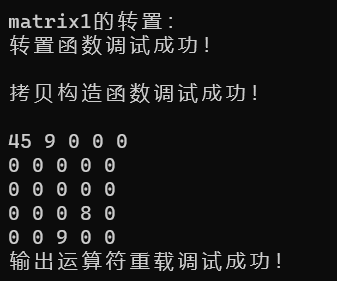
**接着使用加号和等号运算，把m1和m2相加的值赋给新对象，实现赋值运算：**



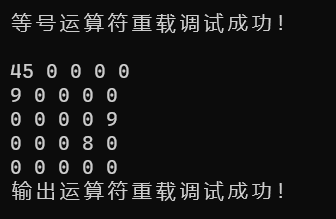
**重复上面的调用，分别调用减法、乘法：**



**最后同样接受matrix1的转置：**



**最终我们调试m2=m1的赋值，有以下的结果：**



**虽然以前的赋值都是用=实现的，但每个赋值都相当于新的构造，因此实际上调用的是拷贝构造函数，在最后一次对已有对象的赋值才调用等号的重载。**

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

**实验过程遇到的最主要问题就是对空间的想象，对行指针和列指针的遍历总会出现内存溢出的问题，因此我选择了分别构建存行和列指针的数组，这样寻找行列的速度会变快而无需遍历，同时相对于普通矩阵，去掉了非零元素，每添加一个结点就创建一个对象，也节省了空间。**

**构造函数：**

**时间复杂度：O(numElements)**

**空间复杂度：O(numRows + numCols)**

**析构函数：**

**时间复杂度：O(numElements)**

**空间复杂度：O(numRows + numCols)**

**AddNode：**

**时间复杂度：O(1) - O(iRow) 取决于插入的位置**

**空间复杂度：O(1)**

**operator>>：**

**时间复杂度：O(numElements)**

**空间复杂度：O(1)**

**operator<<：**

**时间复杂度：O(iRow \* iCol)**

**空间复杂度：O(1)**

**Operator=/-：**

**时间复杂度：O(numElements)**

**空间复杂度：O(numElements)**

**拷贝构造函数：**

**时间复杂度：O(numElements)**

**空间复杂度：O(numRows + numCols)**

**Operator=：**

**时间复杂度：O(numElements)**

**空间复杂度：O(numRows + numCols)**

**operator\*：**

**时间复杂度：O(iRow \* iCol \* other.iCol)**

**空间复杂度：O(iRow \* iCol + other.iRow \* other.iCol)**

**GetTranspose()：**

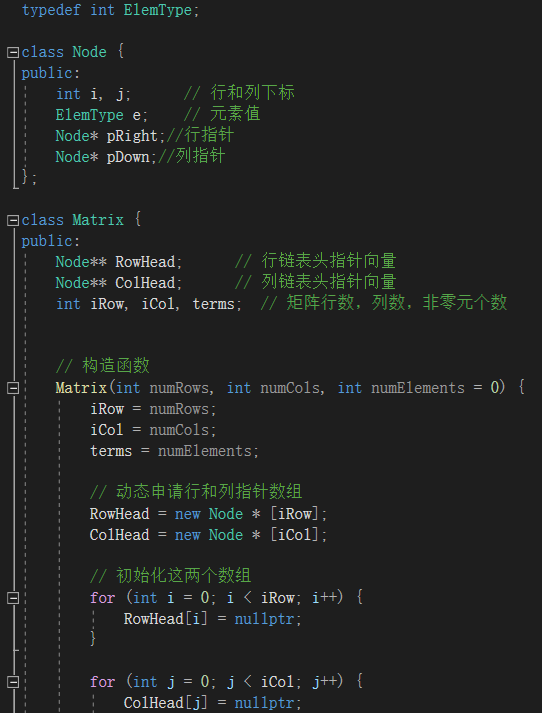
**时间复杂度：O(numElements)**

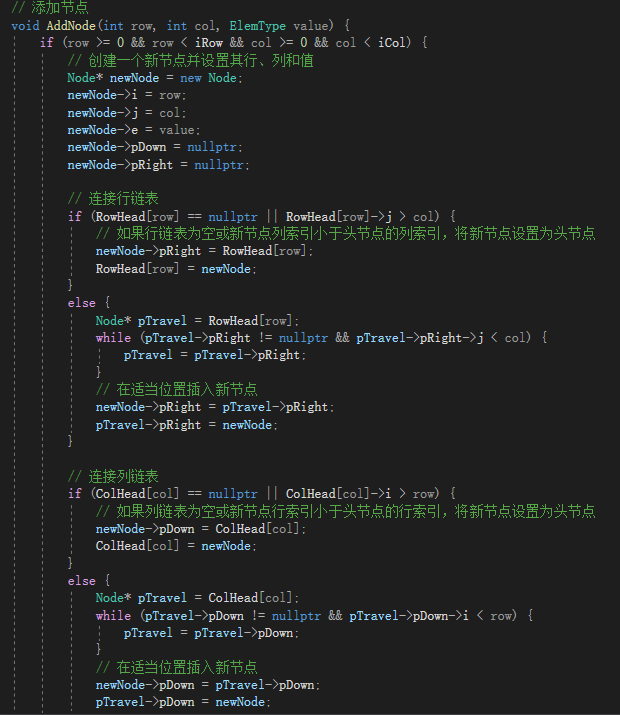
**空间复杂度：O(numElements)**

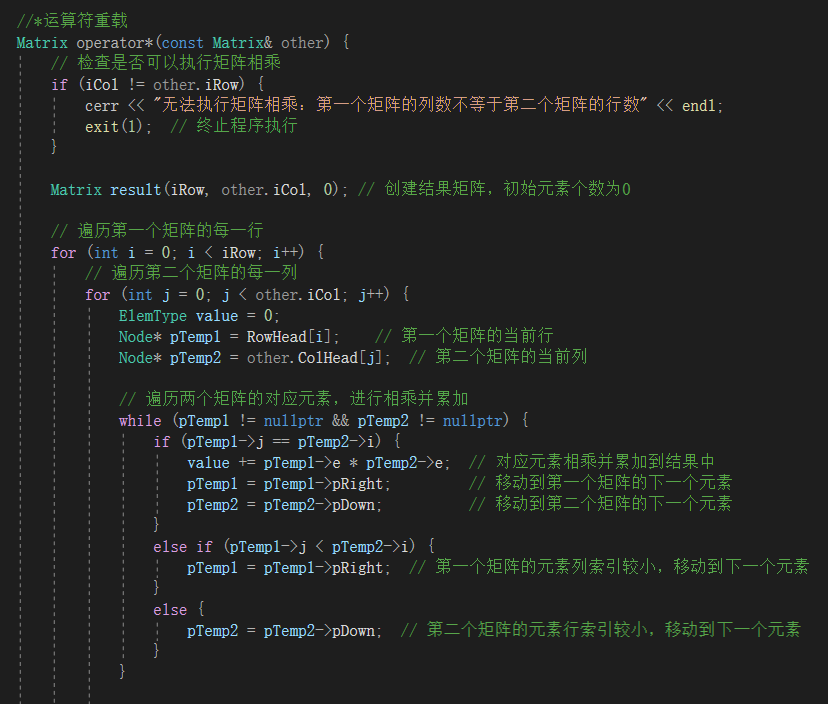
**因此总的时间复杂度是O(iRow \* iCol \* other.iCol)，空间复杂度是O(iRow \* iCol + other.iRow \* other.iCol)**

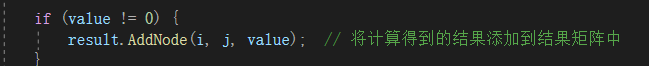
**心得体会：这次的实验比上次复杂得多，用到了许多c++里难点知识，比如链表的遍历、深拷贝与浅拷贝应用、运算符重载及友元函数的应用等，很多知识点都忘了，在写实验的过程中边复习边进行，感觉有很多收获和锻炼。**

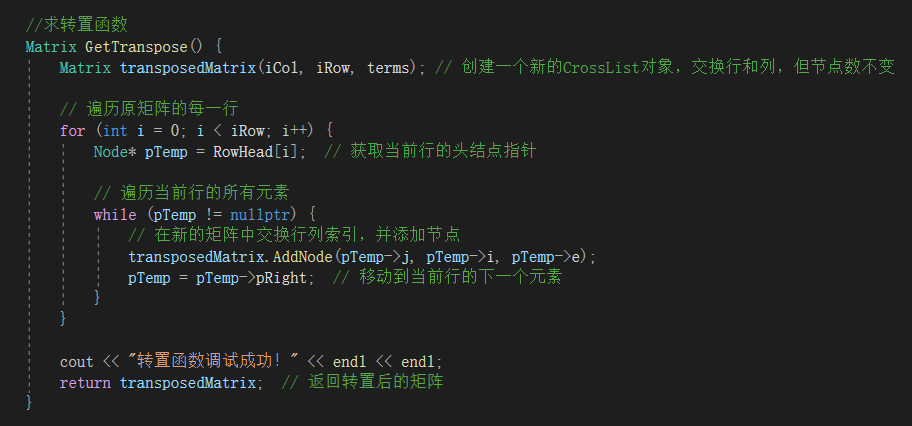
**7. 主要源代码：**











**8. 实验源程序：**附件清单

(1) pro2.cpp

…