Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2019

Homework 5 – Operator overloading for integer matrix class

Due: 17/04/2019, Wednesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement a class for *integer matrices* and overload some operators. The details of these operators and the functions that you will implement are given below. Our focus in this homework is on the class design and implementation. The usage of this class in the main program will be given to you.

Matrix Class Design

You have to keep the *rowNumber*, *columnNumber* and *elements* of the matrix as private data members of the class. Elements of the matrix must be stored in a **dynamic matrix of integers** with *rowNumber* rows and *columnNumber* elements per row. Since the matrix is dynamic one, as *elements*, you will need to use a pointer to pointer. Dynamic matrices were covered in week 4 of the course. Please refer to lecture notes to recall.

Your class implementation must include the following basic class functions:

- Default constructor: creates a *Matrix* object with zero rows and zero columns This constructor does not allocate memory for the elements of the matrix.
- Constructor with *rowNum*, *columnNum*, *init* parameters: creates a *Matrix* object with *rowNum* rows and *columnNum* columns, and allocates enough memory. Initial values of all of the elements must be set to *init*. Please refer to week 4 material about creating a dynamic matrix via pointers. If *rowNum* and/or *columnNum* is/are non-positive, then an empty matrix with zero rows and zero columns must be created.
- Deep copy constructor: takes a *Matrix* object as const-reference parameter and creates a new *Matrix* object as the deep copy of the parameter.
- Destructor: By definition, destructor deletes all dynamically allocated memory of the object and returns them to the heap.

In this homework, you will overload several operators for the *Matrix* class. These operators and the details of overloading are given below. You can overload operators as member or free functions.

+ This operator will be overloaded such that it calculates and returns the summation of two *Matrix* operands. The mathematical definition of + operation on matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ is given below. Here assume that the dimensions of the operands are the same; you do not need to check the dimension equality in the operator implementation.

$$\boldsymbol{X}+\boldsymbol{Y}=\begin{bmatrix} x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{1,n} \\ x_{2,1}, x_{2,2}, x_{2,3}, \dots, x_{2,n} \\ \cdot \\ \cdot \\ x_{k,1}, x_{k,2}, x_{k,3}, \dots, x_{k,n} \end{bmatrix}+\begin{bmatrix} y_{1,1}, y_{1,2}, y_{1,3}, \dots, y_{1,n} \\ y_{2,1}, y_{2,2}, y_{2,3}, \dots, y_{2,n} \\ \cdot \\ \cdot \\ y_{k,1}, y_{k,2}, y_{k,3}, \dots, y_{k,n} \end{bmatrix}$$

$$=\begin{bmatrix} x_{1,1}+y_{1,1}, x_{1,2}+y_{1,2}, x_{1,3}+y_{1,3}, \dots, x_{1,n}+y_{1,n} \\ x_{2,1}+y_{2,1}, x_{2,2}+y_{2,2}, x_{2,3}+y_{2,3}, \dots, x_{2,n}+y_{2,n} \\ \cdot \\ \cdot \\ x_{k,1}+y_{k,1}, x_{k,2}+y_{k,2}, x_{k,3}+y_{k,3}, \dots, x_{k,n}+y_{k,n} \end{bmatrix}$$

− This operator will be overloaded such that it calculates and returns the difference of two *Matrix* operands. The mathematical definition of − operation on matrixes $\boldsymbol{X}$ and $\boldsymbol{Y}$ is given below. Here assume that the dimensions of the operands are the same; you do not need to check the dimension equality in the operator implementation.

$$\boldsymbol{X}-\boldsymbol{Y}=\begin{bmatrix} x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{1,n} \\ x_{2,1}, x_{2,2}, x_{2,3}, \dots, x_{2,n} \\ \cdot \\ \cdot \\ x_{k,1}, x_{k,2}, x_{k,3}, \dots, x_{k,n} \end{bmatrix}-\begin{bmatrix} y_{1,1}, y_{1,2}, y_{1,3}, \dots, y_{1,n} \\ y_{2,1}, y_{2,2}, y_{2,3}, \dots, y_{2,n} \\ \cdot \\ \cdot \\ y_{k,1}, y_{k,2}, y_{k,3}, \dots, y_{k,n} \end{bmatrix}$$

$$=\begin{bmatrix} x_{1,1}-y_{1,1}, x_{1,2}-y_{1,2}, x_{1,3}-y_{1,3}, \dots, x_{1,n}-y_{1,n} \\ x_{2,1}-y_{2,1}, x_{2,2}-y_{2,2}, x_{2,3}-y_{2,3}, \dots, x_{2,n}-y_{2,n} \\ \cdot \\ \cdot \\ x_{k,1}-y_{k,1}, x_{k,2}-y_{k,2}, x_{k,3}-y_{k,3}, \dots, x_{k,n}-y_{k,n} \end{bmatrix}$$

= This operator will be overloaded such that it will assign the *Matrix* object on the right hand-side of the operator to the *Matrix* object on the left hand-side. This function must be implemented in a way to allow cascaded assignments. Due to C++ language rules, this operator must be a member function (cannot be free function).

== This operator will be overloaded such that it will check two *Matrix* objects for equality. If the sizes of these two matrices are not the same, then this operator directly returns false. If the sizes are the same, then all of the corresponding elements of the matrices must be the same in order to return true. Otherwise, the operator returns false.

! This operator will be overloaded such that it will return the transpose of the matrix operand. In linear algebra, the transpose of a matrix is an operator which switches the rows with columns.

Note that this is a unary operator (i.e. it takes a single operand). That means, if you implement it as a member function, the function does not take any parameters and processes the object on which the operator function is called. Alternatively, you can implement it as a free function; in this case, the function takes only one const-reference *Matrix* parameter and processes it. In either case, the transposed value must be returned from the function as value (i.e. not reference) and you will not change the content of its operand.

The mathematical definition of transpose is given below.

$$\boldsymbol{X} = \begin{bmatrix} x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{1,n} \\ x_{2,1}, x_{2,2}, x_{2,3}, \dots, x_{2,n} \\ x_{3,1}, x_{3,2}, x_{3,3}, \dots, x_{3,n} \\ x_{4,1}, x_{4,2}, x_{4,3}, \dots, x_{4,n} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_{k,1}, x_{k,2}, x_{k,3}, \dots, x_{k,n} \end{bmatrix} \Rightarrow \; !\boldsymbol{X} = \begin{bmatrix} x_{1,1}, x_{2,1}, x_{3,1}, x_{4,1} \; \dots, x_{k,1} \\ x_{1,2}, x_{2,2}, x_{3,2}, x_{4,2} \; \dots, x_{k,2} \\ x_{1,3}, x_{2,3}, x_{3,3}, x_{4,3} \; \dots, x_{k,3} \\ \cdot \\ \cdot \\ \cdot \\ x_{1,n}, x_{2,n}, x_{3,n}, x_{4,n} \; \dots, x_{k,n} \end{bmatrix}$$

Other than these operators, you will need to implement the following accessors (getters) and mutator (setter) as member functions.

`getRowNumber`	returns the number of rows of the *Matrix* object. No parameters.
`getColumnNumber`	returns the number of columns of the *Matrix* object. No parameters.
`getElementAt`	takes row and column indices as parameters and returns the matrix element at this position
`setElementAt`	takes row and column indices, and an integer value as parameters. The function does not return anything but assigns the parameter value to the corresponding matrix element.

You also need to write member function named `print` that does not take any parameters, does not return anything, but displays the content of the matrix.

A life-saving advice: The getters above need to be **`const` member functions** so that they work fine with const-reference parameters. If you do not remember what "`const` member function" is, please refer to CS201 notes or simply Google it!

In order the see the usage and the effects of these operators, please see **sample runs** and the provided **main.cpp**.

In this homework, you are allowed to implement some other helper member functions, accessors and mutators, if needed. However, you are **not allowed to use friend functions and friend classes**.

You have to have separate header and implementation files, in addition to the main.cpp.

Please do not make use any standard or non-standard *container* or *matrix* classes in your homework; you will implement your own class from scratch.

main.cpp

In this homework, **main.cpp** file is given to you within this homework package and we will test your codes with this main function with different inputs. You are not allowed to make any modifications in the main function (of course, you can edit the file to add `#includes`, etc. to the beginning of the file). Inputs are explained with appropriate prompts so that you do not get confused. We strongly recommend you to examine main.cpp file and sample runs before starting your homework.

Sample Runs

Some sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Especially try different matrix values (other than the ones given in the sample runs) and extreme cases.

Sample Run 1:

```
Please enter the row number of Matrix m1:
2
Please enter the column number of Matrix m1:
3
Please enter the init value of Matrix m1:
1

Matrix m1:
1 2 3
2 3 4

Please enter the row number of Matrix m2:
2
Please enter the column number of Matrix m2:
3
Please enter the init value of Matrix m2:
1

Matrix m2:
1 2 3
2 3 4

Please enter the row number of Matrix m3:
3
Please enter the column number of Matrix m3:
3
Please enter the init value of Matrix m3:
1

Matrix m3:
1 2 3
2 3 4
3 4 5

Matrix m4:
1 2 3
2 3 4
3 4 5
```

```
m1 is equal to m2.

m3 = m1 + m2 + m1:
3 6 9
6 9 12

m4 = m2 - m1 - m2:
-1 -2 -3
-2 -3 -4

Transpose of m3:
3 6
6 9
9 12

Assigning m4 = m2 = m3.
Matrix m4:
3 6
6 9
9 12

Matrix m2:
3 6
6 9
9 12

Matrix m3:
3 6
6 9
9 12
```

Sample Run 2:

```
Please enter the row number of Matrix m1:
2
Please enter the column number of Matrix m1:
5
Please enter the init value of Matrix m1:
1

Matrix m1:
1 2 3 4 5
2 3 4 5 6

Please enter the row number of Matrix m2:
2
Please enter the column number of Matrix m2:
2
Please enter the init value of Matrix m2:
2

Matrix m2:
2 4
```

```
4 6

Please enter the row number of Matrix m3:
1
Please enter the column number of Matrix m3:
1
Please enter the init value of Matrix m3:
1

Matrix m3:
1

Matrix m4:
1

m1 is not equal to m2.

m3 = m1 + m2 + m1:
Matrix m1 and m2 do not have the same dimensions. Cannot be added.

m4 = m2 - m1 - m2:
Matrix m1 and m2 do not have the same dimensions. Cannot be subtracted.

Transpose of m3:
1

Assigning m4 = m2 = m3.
Matrix m4:
1

Matrix m2:
1

Matrix m3:
1
```

Sample Run 3:

```
Please enter the row number of Matrix m1:
-1
Please enter the column number of Matrix m1:
2
Please enter the init value of Matrix m1:
1

Matrix m1:

Please enter the row number of Matrix m2:
0
Please enter the column number of Matrix m2:
0
Please enter the init value of Matrix m2:
3
```

```
Matrix m2:

Please enter the row number of Matrix m3:
2
Please enter the column number of Matrix m3:
2
Please enter the init value of Matrix m3:
1

Matrix m3:
1 2
2 3

Matrix m4:
1 2
2 3

m1 is equal to m2.

m3 = m1 + m2 + m1:

m4 = m2 - m1 - m2:

Transpose of m3:

Assigning m4 = m2 = m3.
Matrix m4:

Matrix m2:

Matrix m3:
```

Some Important Rules

You should prepare (or at least test) your program using MS Visual Studio 2012 C++. We will use the standard C++ compiler and libraries of the abovementioned platform while testing your homework. It'd be a good idea to write your name and last name in the program (as a comment line of course).

Submissions guidelines are below. Some parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade. Name your solution, project, cpp file that contains your main program using the following convention (the necessary file extensions such as .sln, .cpp, etc, are to be added to it):

"SUCourseUserName_YourLastname_YourName_HWnumber"

Your SUCourse user name is actually your SUNet user name which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

Cago_Ozbugsizkodyazaroglu_Caglayan_hw5

In some homework assignments, you may need to have more than one .cpp or .h files to submit. In this case add informative phrases after the hw number. However, do not add any other character or phrase to the file names.

Now let us explain which files will be included in the submitted package. Visual Studio 2012 will create two *debug* folders, one for the solution and the other one for the project. You should delete these two *debug* folders. Moreover, if you have run your program in release mode, Visual Studio may create *release* folders; you should delete these as well. Apart from these, Visual Studio 2012 creates a file extension of *.sdf*; you will also delete this file. The remaining content of your solution folder is to be submitted after compression. Compress your solution and project folders using WINZIP or WINRAR programs. Please use "zip" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up correctly and it contains all of the solution, project and source code files that belong to the latest version of your homework. Especially double-check that the zip file contains your cpp and (if any) header files that you wrote for the homework.

Moreover, we strongly recommend you to check whether your zip file will open up and run correctly. To do so, unzip your zip file to another location. Then, open your solution by clicking the file that has a file extension of .sln. Clean, build and run the solution; if there is no problem, you could submit your zip file. Please note that the deleted files/folders may be regenerated after you build and run your program; this is normal, but do not include them in the submitted zip file.

You will receive no credits if your compressed zip file does not expand or it does not contain the correct files. The naming convention of the zip file is the same. The name of the zip file should be as follows:

SUCourseUserName_YourLastname_YourName_HWnumber.zip

For example, zubzipler_Zipleroglu_Zubeyir_hw5.zip is a valid name, but

Hw4_hoz_HasanOz.zip, HasanOzHoz.zip

are **NOT** valid names.

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Taha Atahan Akyıldız, Albert Levi