Object-Oriented Programming

Spring Semester 2022

Homework Assignment 2.

28/03/2021

Contents

Introduction

1. Try to provide clear and careful solutions.
2. You should provide comments for your code so it will be completely clear what you are trying to achieve. WARNING! Lack of comments might lead to points reduction.
3. Please note the following few points which may lead to points reduction during the submission check:
   (a) Avoid using *magic numbers*. For example: "if (i>17)", 17 is a magic number. If 17 is representing, for example, the number of shoes, then instead you should write: "if (i>shoesNumber)".
   (b) Try to avoid code duplication as much as possible.
   (c) You should not globally enable *std* namespace usage or any other namespaces, e.g. *using namespace std;*.

Definitions

During the second homework, you will learn how to overload different operators in C++, as well you are going to practice your skills working with templates.

In this assignment you will implement a class that is capable to manage basic matrix manipulations (**Matrix** class), and a class that represents some container of matrices (**MatrixContainer** class).

Learning from past experience, please note that some (small) updates to the definitions and requirement of the assigment might be published in next few days and that following said updates and the HW forum is required.

Like in the last assignment, The methods signature is up to you, but :

1. Use the **const** keyword where possible and needed.
2. Use the **friend** functions where needed.
3. Use **reference** variables where possible and needed.
4. Use **static** variables and functions where possible and needed.

You need to understand where to use what, but if you'r unsure, feel free to ask me for help.

Your goals are:

- Having the main files, you should provide declaration and implementation while keeping in mind the basic *OOP* concept of *encapsulation*.

  Keep in mind that the included main files are very basic and do not cover all possible scenarios and end cases.

- Take care of correct dynamic memory management. Be aware of both allocating and releasing resources.

  ☒ In this submission, you are **RESTRICTED** from using *STL* library regardless of prior knowledge you have.

  ☑ You **can** assume that the input for the functions you write will be legal.

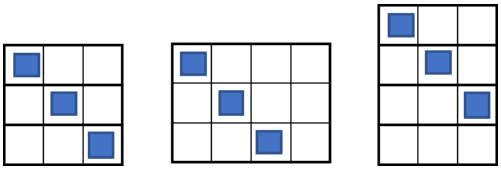
Matrix class

You have a free choice in the implementation of the class attributes and methods, but you must implement the following methods:

1. Operator+/- - by scalar (same type as the marix) or other matrix. Ex : mat+5 : adds 5 to each cell in the matrix
2. Operator* – just for matrix*scalar
3. Operator* - matrix by matrix (of the exact same size) **5 BONUS POINTS**
    - Not required, if you do not wish to implement the function, write the appropriate signature and return the identity matrix.
4. getIdentityMatrix – return the identity matrix .
5. Operator++/-- – prefix and postfix. Same functionality as mat+1, mat-1.
6. getDiag – returning the matrix main diagonal (Example in the next page).
7. Operator==/!= - return if 2 matrices are not/equal (equal : if $\forall i,j\ mat1[i,j] == mat2[i,j]$, more on floating point comparison in appendix 1)
8. To matrix type covertion – returning the trace of the matrix. Ex : int trace = int(mat)
9. operator<<
10. operator() – with arguments i,j should return a reference to cell [i,j] , example – mat1(x,y)

Hint – To save work and time (and code duplication) - use operators to implement different operators, for example : use == to implement !=, use + to implement (-) etc.

main diagonals:



*the class methods should be implemented only in the header file as inline methods. (to avoid troubleshooting that can appear with template and classes)

MatrixContainer class

You have a free choice in the implementation of the class attributes and methods.

To be time and space complexity efficient- the data structure that holds the matrices should be a dynamic array (Explantion in appendix 2).

You must implement the following methods:

- addMatrix – add a matrix to the container (you can add an unlimited amount of matrices)
- removeLastMatrix – removing the last matrix from the container.
- mulMatAtIndexByScalar(i,n) – returning the multiplication result of the matrix in index i with the scalar n
- addMatricesAtIndexes(i1,i2) – returning the result of the addition of the matrix in index i1 and the matrix in index i2.
- operator<< – printing all the matrices in the container.
- size() – return the amount of matrices in the container.
- limit() – return the size of the dynamic array.

Evaluation

Homework exercise provided with the following example program files and corresponding outputs:

1. matrix_main.cpp
2. matrix _output.txt
3. main_matrix_container.cpp
4. output_matrix_container.txt

You should be able to compile your code with “matrix_main.cpp” and “main_matrix_container.cpp” files, and receive, respectively, the correct output which placed in “matrix_output.txt” and “output_matrix_container.txt” file.

Submission should only include the following files :

1. Martix.h
2. MatrixContainer.h

App1. Floating point comparison

Floating point comparison (double==double) can be inaccurate, this is due to the way decimal values are stored in the computer.
One way to compare floats is by tolerance to difference, i.e , checking to see if the difference is lower that known epsilon value (tolerance).
In this assignment, your matrix might hold floating point value, so you need the make a robust comparison.
Use the `DBL_EPSILON` const defined value, and compare like so:

$$if\big((val1 - val2) < DBL\_EPSILON\big)\,\{\,return\ equal\}$$

This compare will work in Integer values as well.
This is not the most robust way, there are more robust comaprisons, but for this assignment, it will suffice.

For more information on floating point comparison : Link - http://realtimecollisiondetection.net/blog/?p=89

App2. Dynamic array data structure

Implementing a dynamic array should be as follows:

The size of the array starts from size = 1.

When full, that is, n (number of elements in the array) is equal to size: the array doubles his size by creating a new array twice as big, and copies his elements into the new array.

When $n = \frac{size}{4}$, the array halves his size by creating a new array half as big , and copies his elements into the new array.

When implemention the array, don't forget to free allocated memory!

This guarantees amortized $O(1)$ per insert/delete last operation.

Make sure you keep the C++ syntax convention and submit the files exactly as described in the "Oop – Cpp – Conventions and Requirements" file in Moodle.

GOOD LUCK! ☺