Fall 2022

CPSC2620 Assignment 3

Due: (October 28), 2022 11:55pm

(30 marks)

Please note that due to the potential COVID situation, the requirements of this assignment might be changed and also the compilation and submission procedure might be changed. You will be notified should a situation arise.

Notes:
(*) Please note that no collaboration is allowed for this assignment. Should any plagiarism be identified, all the involved students will get zero for the assignment and will be reported to the Chair of the department for further action. The marker has been asked to check your submissions carefully.
(*) Your submission will be marked on the correctness and readability (including comments) of your program.
(*) Please follow the guidelines to name your files (see below).
(*) Late submission policy: 1 day late: 5% off, 2 days late: 10% off, more than 2 days late: you will get 0 for the assignment.

(*) Write and submit a single Makefile that is used to compile your programs in Questions 1 and 2.

Question 1 (15 marks)

Note: Files to be created and submitted for Question 1 are:

MatrixExt.cc
MatrixExt.h
testMatrixExt.cc (which is used to test the class *CMatrixExt* you create)

The executable for this question is called *testMatrixExt*. Please see below for more details.

Continuing writing the *CMatrix* class in Question 2 from Assignment 2, we will expand it in this question. Let's call the new class *CMatrixExt*. All requirements regarding element indices are the same as before. Data members are the same as before. The following is the list of member functions. Note that some of them can be borrowed from your solution to Question 2 in Assignment 2.

int **allocateMatrixMemory(unsigned int n):to allocate an n x n array of integers.
void deallocateMatrixMemory(int **M, unsigned int n):to deallocate an n x n array of integers.
CMatrixExt(int n = 2): constructor.

CMatrixExt(const CMatrixExt &M): copy constructor: copy matrix M to this current matrix using **deep copying**.
~CMatrixExt(): destructor.
int getDimension(void): to return the dimension: n for the matrix n x n.
int getElementAt(int i, int j): to return the element at (i, j). Check whether the indices are OK. The indices start from 1 and will be checked.
int replaceElementAt(int i, int j, int newint): to replace the element at (i, j) with the new element. Check whether the indices are OK. Return the old element.
int setElementAt(int i, int j): set the element at (*i, j*) to be 1 and return the old element at (*i, j*). Use assert to ensure that the indices are within the valid range.
int clearElementAt(int i, int j): set the element at (*i, j*) to be 0 and return the old element at (*i, j*). Use assert to ensure that the indices are within the valid range.
void swapElementsAt(int i1, int j1, int i2, int j2): to swap the element at (i1, j1) with the element at (i2, j2). Chech whether the indices are OK.
void resizeMatrix(int newsize): to resize the matrix with the new size Copy the original elements if necessary.
void readMatrix(void):
void printMatrix(void): to read or print out the matrix from cin or to cout. Read elements and print out elements by following the row-major order.
void makeIdentity(): to make the current matrix an identity, i.e., change all the elements on the diagonal to 1 and change all the other element to 0.
void copyMatrix(const CMatrixExt &M) : to take one CMatrixExt parameter and copy it to this current CMatrixExt object. Use assert to ensure that the dimensions of the two matrices are compatible.
CMatrixExt addMatrix(const CMatrixExt &M) : to take one CMatrixExt parameter and return the result matrix that is the sum between the current CMatrixExt object and the supplied CMatrixExt object in a pair-wise manner. Use assert to ensure that the dimensions of the two matrices are compatible. Don't change the current matrix.
CMatrixExt addMatrix(int x): to add x to each element of the current CMatrixExt object and return the result matrix. Don't change the current matrix. **(**Note: this function replaces the member function **addConstant** in Question 2 in Assignment 2.)
CMatrixExt subtractMatrix(const CMatrixExt &M) : to take one CMatrixExt parameter and return the result matrix that is the difference between the current object and the supplied object in a pair-wise manner. Don't change the current matrix. Use assert to ensure that the dimensions of the two matrices are compatible.
CMatrixExt subtractMatrix(int x): to subtract x from each element of the current CMatrixExt object and return the result matrix. Don't change the current matrix.
CMatrixExt multiplyMatrix(const CMatrixExt &M) : to take one CMatrixExt parameter, compute the product of the current CMatrixExt object with it, and return the result matrix. Don't change the current matrix. Use assert to ensure that the dimensions of the two matrices are compatible.
CMatrixExt multiplyMatrix(int x): to multiply each element of the current CMatrixExt object by x and return the result matrix. Don't change the current matrix.
bool isDiagonal(void): to teturn true if all the elements not on the diagonal are zeros Otherwise return false.
bool isIdentity(void): to return true if all the elements on the diagonal are ones and all the elements elsewhere are zeros. Otherwise return false.

bool isBigger(const CMatrixExt &M): For two matrices A and B, we say A > B, if every element of A is bigger than the corresponding element of B. Return true if the current matrix > the input matrix M. Otherwise return false.
bool isSmaller(const CMatrixExt &M): For two matrices A and B, we say A < B, if every element of A is smaller than the corresponding element of B. Return true if the current matrix < the input matrix M. Otherwise return false.
bool isEqual(const CMatrixExt &M): For two matrices A and B, we say A = B, if every element of A is equal to the corresponding element of B. Return true if the current matrix = input matrix M. Otherwise return false.

Create a test program called *testMatrixExt.cc*. Your test should show that the member functions of the matrix class work as expected. One good way to do this: create two 3 x 3 matrices, read elements in them, do additions, subtractions and multiplication between the two matrices or between a matrix and an integer. Each time print the result out to verify (show) that your program works.

Question 2 (15 marks)

Note: Files to be created and submitted for Question 2 are:

MatrixPro.cc
MatrixPro.h
testMatrixPro.cc (which is used to test the class *CMatrixPro* you create)

The executable for this question is called *testMatrixPro*. Please see below for more details.

Let us make some semi-professional class such that you can show it to your teammates and let them use it. Continuing writing the *CMatrixExt* class in Question 1, we will modify it in this question. Let's call the new class *CMatrixPro*. All requirements regarding element indices are the same as before. Data members are the same as before. Modify the following member functions into overloaded operators, while keeping the other members functions as before.

readMatrix	>> (implemented as a friend)
printMatrix	<< (implemented as a friend)
copyMatrix	= (implemented as a member operator)
addMatrix	+ (implemented as member operators for both versions)
subtractMatrix	**-** (implemented as member operators for both versions)
multiplyMatrix	* (implemented as member operators for both versions)
isBigger	> (implemented as a member operator)
isSmaller	< (implemented as a member operator)
isEqual	== (implemented as a member operator)

Create a test program called *testMatrixPro.cc*. Your test should show that the member functions/operators of the matrix class work as expected. One good way to do this: create two 3 x 3 matrices, read elements in them, do additions, subtractions and multiplication between the two matrices or between a matrix and an integer using the corresponding operators. Each time print the result out to verify (show) that your program works.