CSCE 221 Cover Page
Programming Assignment #1
Due by February 4 midnight to eCampus

First Name Last Name UIN

User Name E-mail address

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name Date

Programming Assignment 1 (100 points)

- You want to implement a data structure that allows to tabulate data coming from many different models such as communication or social networks. The entries in a table may express a relation between two groups of people, e.g. the number 1 could denote friends and 0 otherwise. This type of data structure could be also used to represent a location of an object in a two-dimensional space using its coordinates, e.g. a pair (i, j) to refer to a particular element of the table. These tables are called two-dimensional arrays or matrices. The **Background** section of this assignment provides some basic information about matrices and their operations.

- The purpose of this individual programming assignment is to learn about an elementary design, implementation, and testing of a simple C++ class called `My_matrix`. The class implementation allows you to understand and overview the basic C++ concepts like pointers, memory allocation, deallocation, dynamic arrays, constructors, copy or move constructors and assignments, destructors, operator overloading, reading from and writing to a file.

- In the **first phase** of the assignment, implement in C++ a class `My_matrix` that can hold data of integer type (`int`). The two parameters representing a matrix dimensions are usually not known in advance so it is necessary to allocate the arrays dynamically. The description of dynamic matrix type for data manipulation is provided in the course textbook, see the section "*Dynamic Allocation of Matrices*," pp. 112-113. The first part of the assignment is **due by January 28** with pre-grading done in the labs.

- In the **second phase,** which is **due by February 4th**, you will need to write a generic version of the class `My_matrix` that can handle different types of data.

- The *private* members of the class `My_matrix` should contain at least these elements (do not use the STL class `vector`):

  - `n` is the number of rows in the matrix (`n >= 0`)
  - `m` is the number of columns in the matrix (`m >= 0`)
  - `ptr` is a pointer to the dynamic array of rows of type `int**` (a pointer to a pointer of `int`)
  - `ptr[i]` is a pointer to the `i`th row of type `int*`.
  - Matrix elements will be accessed by indices `i` and `j` using `ptr[i][j]`.

  See the file `My_matrix.h`.

- The *public interface* of the class `My_matrix` should contain at least the operations listed below.

  - `number_of_rows()` returns the number of rows.
  - `number_of_columns()` returns the number of columns.
  - `elem(i,j)` returns or sets a matrix element at row `i` and column `j` with an array bound checking. An `out_of_range` exception is thrown if `i` and/or `j` is not in range of the matrix dimensions.
  - `operator()(i)` returns an access to the `i`th row of the matrix, without performing array bound checking.
  - `operator()(i,j)` returns the `(i,j)` element of the matrix, without performing array bound checking.
  - default constructor with 0 rows and 0 column, and `ptr` is `nullptr`.
  - a constructor with two positive arguments `n` and `m` creates a matrix with `n` rows and `m` columns.
  - copy constructor makes a copy of an object of type `My_matrix`.
  - destructor deallocates allocated memory of an `My_matrix` object.
  - copy assignment assigns an object of type `My_matrix` to another object of the same type.
  - move copy operator and move assignment operator used to optimize object copying code.

  See the file `My_matrix.h`.

- Non-member overloaded operators:

- `operator>>` (input operator) reads input, row by row, to a `My_matrix` object. The first line should have only two numbers: the number of rows and columns. The remaining lines contain matrix rows. If the input format is not correct your program should throw the user-defined exception `incorrect_input()`. Read about exceptions in "*Programming Principles and Practices using C++*", pp. 1138-1139. Also, read the Section 2.4 (pp. 93-) in the course textbook.
- `operator<<` (output operator) prints the content of an `My_matrix` object, row by row, and each row is displayed in a separate line.
- overloading the addition (+) and product (∗) operators for matrices. Both the operators work on `My_matrix` objects. That is, `C=A+B` is equivalent to `C[i][j]=A[i][j]+B[i][j]`, and `C=A*B` is equivalent `C[i][j]`$=\sum_{k=1}^{n}$`A[i][k]*B[k][j]`, see the **Background** section of this assignment for more details. Notice that the number of rows must be the same as the number of columns for both the matrix arguments for the addition operator. If this condition is not satisfied an exception should be thrown. Also, the number of columns of the first argument must be the same as the number of rows of the second argument for the product operator. If this condition is not satisfied an exception should be thrown.

Implementation and Testing

1. Download the supplementary tar file with a sample code from the class webpage.
2. Your files should be arranged as follows
   (a) Declaration of `My_matrix` class in `My_matrix.h`
   (b) Definition (implementation) of `My_matrix` class in `My_matrix.cpp`
   (c) Reading from a file, writing to a file and testing `My_matrix` operations in `main.cpp`
   (d) `Makefile` is called by `make`
3. (50 points) Implement, compile, run and test Phase 1 of the assignment.
   Compile your program using the following Linux machine command line:
   ```
   g++ -std=c++11 *.cpp
   ```
   or
   ```
   make all
   ```
   And then run your program by executing
   ```
   ./main
   ```
4. (20 pt) Implement, compile, run and test Phase 2 of the assignment: A generic version of `My_matrix`
   (a) The templated class `My_matrix` uses data type as a parameter. Recall the templated vector material, slides 16-22 and follow the instructions below
      i. Templates should be declared and defined in the `TemplatedMy_matrix.h` file. Move the content of `My_matrix.cpp` and `My_matrix.h` to `TemplatedMy_matrix.h`
      ii. Replace `int` type by generic type `T`. Later, in the main function, `T` could be specified as any numeric type: `double`, `float`, `long`, or possibly a user-defined type.
      iii. To create a templated class with generic type `T`, you must replace declaration and/or return type `int` by `T`.
      iv. Use the generic type `T` anywhere throughout the class `TemplatedMy_matrix`.
      v. Add the keyword `template <typename T>` before a class declaration.
      vi. If a member function is defined outside the class declaration, change the function signature by replacing `My_matrix::` with `TemplatedMy_matrix<T>::`
   (b) Compile and run the generic version similarly as in Part 1 of the assignment.
   (c) Test all the operations for the generic version using at least three different types of objects.
5. (30 points) Prepare a report (PDF and LyX) and cover page (PDF and LyX) in the electronic version.

(a) (1 point) Program description and purpose of the assignment

(b) (1 point) Description of data structures

(c) (1 point) Instructions to compile and run your program including input and output specifications (if any).

(d) (2 point) Logical exceptions (and possible bug descriptions)

(e) (1 point) C++ object oriented or generic programming features, including C++11 features.

(f) (24 points) Evidences of testing of the part 1 and 2 operations.

Bonus (20 points) Implement Part 1 and Part 2 of the assignment using the STL class `vector` instead of two-dimensional array (`ptr`) and provide the overloaded operations for `My_matrix` type of objects.

Submission to eCampus no latter than February 4 and follow the steps.

1. Create a folder for the assignment report files, in LyX and PDF formats.
2. Create a folder for each phase (1 and 2) of the class `My_matrix`.
3. Create a directory named `FirstName_SecondName_PA1_19A` and move the three directories created in the step 1 and 2 into this one.
4. Compress the directory `FirstName_SecondName_PA1_19A` using the tar program, see the instructions link. Name the tar file `FirstName_SecondName_PA1_19A.tar`
5. Submit the `FirstName_SecondName_PA1_19A.tar` file obtained in step 4 to eCampus.

Background

- An example of a 3×2 matrix with 3 rows and 2 columns: $\begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \end{bmatrix}$
- An example of a $n \times m$ matrix, n rows and m columns, n and m are positive integers:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix}$$

- The ith row of A is the $1 \times m$ matrix $[a_{i1}, a_{i2}, \dots, a_{im}]$.
- The jth column of A is the $n \times 1$ matrix $\begin{bmatrix} a_{1j} \\ \dots \\ a_{nj} \end{bmatrix}$.
- The (i, j)th element or entry of A is the element a_{ij}. Further on, we will use the notation $A = [a_{ij}]$ to denote that the matrix consists of these elements.

Matrix Arithmetic: Addition

Let $A = [a_{ij}]$ and $B = [b_{ij}]$ be $m \times n$ matrices. The *sum* of A and B, denoted by $A + B$, is the $m \times n$ matrix that has $a_{ij} + b_{ij}$ as its (i, j)th element. In other words, $A + B = [a_{ij} + b_{ij}]$. Example:

$$\begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 5 \\ 0 & 3 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 0 & 5 \\ 4 & 7 \end{bmatrix}.$$

Note that matrices of different sizes cannot be added.

Matrix Arithmetic: Product

Let A be an $n \times k$ matrix and B be a $k \times m$ matrix. The *product* of A and B, denoted by AB, or sometimes $A * B$, is the $n \times m$ matrix that has its (i, j)th element equal to the sum of the products of the corresponding elements from the ith row of A and the jth column of B. In other words, if $AB = [c_{ij}]$ then $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{ik}b_{kj}$.

Fact: The product of two matrices is *undefined* when the number of columns in the first matrix is not the same as the number of rows in the second matrix. Example:

$$\begin{bmatrix} 1 & 0 & 4 \\ 2 & 1 & 1 \\ 3 & 1 & 0 \\ 0 & 2 & 2 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 1 & 1 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 14 & 4 \\ 8 & 9 \\ 7 & 13 \\ 8 & 2 \end{bmatrix}$$

where $c_{21} = 2 \cdot 2 + 1 \cdot 1 + 1 \cdot 3 = 8$.