Tarea 1 - Repaso C++

Prof: Nancy Hitschfeld K.
Auxiliar: Pablo Pizarro R. Sergio P. Salinas

1. Introducción

Esta tarea inicial está orientada para los que no han aún programado en C++ den los primeros pasos en esa dirección y para los que ya saben, repasar las características más importantes, e implementar en C++ una clase que permita manejar matrices como lo hace Matlab/Octave, aunque una versión reducida.

2. Requerimientos de la tarea

Se debe implementar la clase **Matrix** la cual aceptará cualquier tipo de valor numérico (i.e. se debe implementar con templates). Esta clase debe realizar múltiples operaciones, como lo son:

- Suma
- Resta
- Multiplicación
- Obtener el máximo/mínimo
- Cálculo del determinante
- Cálculo de la transpuesta
- Cálculo de la inversa
- Guardar la matriz a un archivo
- Imprimir en consola

Se le proveerá un archivo .h con las declaraciones de la clase. Usted debe implementar todos los métodos, los cuales deben ser testeados con Google Test (como se verá en la auxiliar 3). El objetivo de la tarea es familiarizarse con el lenguaje de programación, la creación de objetos, el cálculo matricial y el *testing*. La clase debe tener como mínimo lo siguiente:

Código 1: Archivo de encabezado Matrix.h

```
#include <iostream>
#include <string>
#include <stdexcept>

template<class T>
class Matrix {
private:
    T *mat; // Store the matrix
    int n = 0; // Number of rows
    int m = 0; // Number of columns
public:
    Matrix(); // Empty constructor
    Matrix(int n); // Constructor, vector like [1xn]
```

```
    Matrix(int n, int m); // Constructor [nxm], n:rows, m: columns
    Matrix(const Matrix<T> &matrix); // Copy constructor, https://www.geeksforgeeks.org/copy-
     ↪ constructor-in-cpp/
    ~Matrix(); // Destructor

    // Setters and getters
    void set(int i, int j, T value); // Set value to (i,j) <row,column>
    T get(int i, int j) const; // Get value from (i,j) <row,column>
    void fill(T value); // Fill all the matrix with a value

    // Dimensions
    int *size() const; // Returns a list of the size of the matrix, e.g. [2,4], 2 rows, 4 columns
    int length() const; // Return max dimension, usefull for vectors, e.g. [2,4] -> 4

    // Values
    T max() const; // Maximum value of the matrix
    T min() const; // Minimum value of the matrix
    T det() const; // Calculate the determinant of the matrix

    // Utilitary functions
    void disp() const; // Display matrix to console
    void save_to_file(std::string filename) const; // Save matrix to a file, any format is valid
    Matrix<T> &clone() const; // Clone the matrix

    // Booleans
    bool is_diag() const; // Matrix is diagonal
    bool is_identity() const; // Check if matrix is identity
    bool is_symmetric() const; // Check if matrix is symmetric
    bool is_square() const; // Check if matrix is square nxn
    bool is_vector() const; // Check if matrix is vector
    bool operator==(const Matrix<T> &matrix) const; // Equal operator
    bool operator!=(const Matrix<T> &matrix) const; // Not equal operator

    // Mathematical operation
    Matrix<T> &operator=(const Matrix<T> &matrix); // Assignment operator
    Matrix<T> &transpose() const; // Transpose the matrix
    Matrix<T> &operator*=(const Matrix<T> &matrix); // Multiplication
    Matrix<T> &operator*=(T a); // Multiply by a constant
    Matrix<T> &operator*+=(const Matrix<T> &matrix); // Add
    Matrix<T> &operator*-=(const Matrix<T> &matrix); // Substract
    Matrix<T> &inverse() const; // Matrix inversion [M] -> [M]^-1
    void normalize(); // Normalize the matrix, divide all the elements by the max number, [1,2;3,4] ->
     ↪  [0.25,0.5;0.75,1]
};
```

Ejemplo de uso:

Código 2: Ejemplo de uso

```
auto *mat = new Matrix<double>(3, 2); /* 3 filas, 2 columnas */
mat->set(0, 1, 3); /* set(fila, columna, valor) */
mat->set(1, 1, 2); /* indización comienza en cero */
mat->set(2, 0, 1);
```

```
mat->disp(); /**
0    3
0    2
1    0
**/
delete mat;
```

Algunas consideraciones:

- Recuerde usar templates.
- El constructor debe inicializar la matriz en cero.
- Los índices de filas y columnas parten desde cero.
- Se espera un test de cada una de las operaciones, pruebe multiplicación, sumas, restas y determinante con distintos tamaños de la matriz, etc. Las funciones que hacen **disp** no es necesario testearlas.
- Si se realiza una operación incorrecta (como multiplicar matrices con incorrecto orden) se espera que usted lance una excepción. Para ello puede usar **throw std::logic_error("[MATRIX] Mensaje de error");**. Los tests también deben ser capaces de esperar excepciones.
- Usted puede añadir tantos métodos privados como estime conveniente, puede incluso cambiar los nombres de las variables privadas. Lo que NO puede hacer es renombrar las operaciones **públicas** ya que su código será testeado con un archivo externo.

En el readme conteste las siguientes preguntas:

1. ¿Afecta en algo el tipo de dato de su matriz?, ¿Qué pasa si realiza operaciones de multiplicación o inversa con un tipo de dato *integer* o *double*.
2. ¿Qué pasa si se normaliza la matriz con un integer o con un double?
3. Si se empezaran a usar números muy pequeños o muy grandes y principalmente números primos, ¿Qué ocurre en términos de precisión?
4. ¿Pueden haber problemas de precisión si se comparan dos matrices idénticas pero con diferente tipo? (`Matrix p1 == p2`).
5. Grafique el tiempo que tarda el proceso de inversión de la matriz vs el número de elementos (tamaño) y comente sus resultados.

3. Información extra

- **Se debe incluir README sobre cómo ejecutar su programa.** Si no se incluye un README se realizará 1.0 punto de descuento en su tarea.
- Su tarea debe realizarse con tests. Se evaluará que cada una de sus implementaciones haya sido testeada. Tareas sin tests serán evaluadas con la nota mínima.
- Usted puede redifinir los operadores de *overloading* si estima conveniente usando **friends**, tambien puede redefinir el cin/cout. Esto no es obligatorio, pero lo puede hacer.
- No modifique el nombre de las funciones, ya que la evaluación de la tarea se hará con un archivo de test que llamará a las mismas funciones que posee el archivo de encabezado .h. Tampoco puede cambiar el nombre de la clase.
- Se recomienda la utilización del IDE CLion de Jetbrains, ya que al ser estudiantes de la Universidad de Chile disponen de cuentas profesionales gratuitas. Para esto deben ingresar a https://www.jetbrains.com/shop/eform/students y utilizando su correo @dcc.uchile.cl deberían recibir de forma casi instantánea la aprobación. CLion funciona tanto en Unix como en Windows (ahí deberán instalar primero MinWin o CygWin y CLion lo reconocerá). A menos que usted haga uso intenso de Visual Studio, este no se recomienda tanto por la dificultad de la curva de aprendizaje así como también el uso en espacio (pesa alrededor de 30GB).
- Como compilador en Windows puede usar gcc de cygwin. https://www.jetbrains.com/help/clion/quick-tutorial-on-configuring-clion-on-windows.html
- La fecha de entrega es aquella que salga en U-Cursos.

4. Links de interés

- Editor CLion para C++
- Tutorial C++ (1)
- Curso de C++ (2)
- Copy constructor in C++
- Operator overloading
- What is “Object-Oriented Programming”? (1991 revised version)