Tarea 1 – Repaso C++

Profesora: Nancy Hitschfeld K.
Auxiliar: Vicente González
Ayudante: Juan Flores

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 el tipo de dato `double` y deberá realizar múltiples operaciones, como lo son:

- Suma
- Resta
- Multiplicación
- Obtener el máximo/mínimo
- Cálculo de la transpuesta
- Guardar la matriz a un archivo
- Crear una matriz desde 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 vió en la auxiliar 2). 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:

Listado 1: Archivo Matrix.h

```
#include <cstddef>
#include <memory>
#include <ostream>
#include <string>
#include <tuple>

class Matrix {
 private:
  std::unique_ptr<double[]> 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 std::string &filename);  // Constructor that reads from a file,
                                          // any format is valid
  Matrix(const Matrix &
             matrix);  // Copy constructor,
                       // https://www.geeksforgeeks.org/copy-constructor-in-cpp/
```

```
  ~Matrix();           // Destructor

  // Setters and getters
  double &operator[](std::size_t x,
                     std::size_t y);  // Set value to (i,j) <row,column>
  const double &operator[](
      std::size_t x,
      std::size_t y) const;  // Get value from (i,j) <row,column>
  void fill(double value);   // Fill all the matrix with a value

  // Dimensions
  std::tuple<int, 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
  double max() const;  // Maximum value of the matrix
  double min() const;  // Minimum value of the matrix

  // Utilitary functions
  friend std::ostream &operator<<(
      std::ostream &os, const Matrix &mat);  // Display matrix to console
  void save_to_file(const std::string &filename)
      const;  // Save matrix to a file, any format is valid

  // Booleans
  bool operator==(const Matrix &matrix) const;  // Equal operator
  bool operator≠(const Matrix &matrix) const;  // Not equal operator

  // Mathematical operation
  Matrix &operator=(const Matrix &matrix);   // Assignment operator (copy)
  Matrix &operator*=(const Matrix &matrix);  // Multiplication
  Matrix &operator*=(double a);              // Multiply by a constant
  Matrix &operator+=(const Matrix &matrix);  // Add
  Matrix &operator-=(const Matrix &matrix);  // Substract
  Matrix &transpose();                      // Transpose the matrix
};
```

Ejemplo de uso:

Listado 2: Archivo example.cpp

```
#include <iostream>
#include "Matrix.h"

int main(int argc, char *argv[]) {
  Matrix mat(3, 2); /* 3 filas, 2 columnas */
  mat[0, 1] = 3; /* set(fila, columna, valor) */
  mat[1, 1] = 2; /* indización comienza en cero */
  mat[2, 0] = 1;
  std::cout << mat;
  /**
  0 3
  0 2
  1 0
  **/
```

```
    return 0;
}
```

Algunas consideraciones:

- 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 y restas con distintos tamaños de la matriz, etc. Las funciones que imprimen en pantalla no son necesarias 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.
- Usted debe crear una **librería** no un **ejecutable**, ya que la idea es que su código sea usado por otras personas.
- Para que su tarea compile debe utilizar el **estándar C++ 23**, esto lo puede definir dentro de cmake cambiando la variable `CMAKE_CXX_STANDARD`.

En el readme conteste las siguientes preguntas:

1. ¿Afectaría en algo el tipo de dato de su matriz?, ¿Qué pasa si realiza operaciones de multiplicación tipo de dato *integer* en vez de *double*?
2. Si se empezaran a usar números muy pequeños o muy grandes y principalmente números primos, ¿Qué ocurre en términos de precisión?
3. ¿Pueden haber problemas de precisión si se comparan dos matrices idénticas pero con diferente tipo? (`Matrix p1 == p2`).

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