# ELE078 - Programação Orientada a Objetos

## Atividade Prática 04

### Parte 01: Construtor de Cópia

Modifique a classe *Ponto2D* , implementada na Atividade Prática 03, para que seja possível, além das formas usais, inicializar objetos da seguinte forma:

In [ ]:

```
// formas usuais (já implementadas)
Ponto2D p1;
Ponto2D p2(3.0, 4.0);

// inicializações usando o construtor de cópia
Ponto2D p3(p1);      // Ponto2D p3 é inicializado com as coordenadas do ponto p1
Ponto2D p4 = p2;     // Ponto2D p4 é inicializado com as coordenadas do ponto p2
```

### Parte 02: Revisitando a classe *Matrix* (Gerência de Memória e Construtor de Cópia)

Readequar o código do TAD **Matrix**, implementado na Atividade Prática 02. A classe deve ser implementada de forma a permitir que os usuários **manipulem seus índices (linhas e colunas) começando de 1 (um) e não de 0 (zero)**. Os elementos devem ser armazenados com estruturas criadas dinamicamente na memória através do uso de ponteiros duplos.

A interface da classe **Matrix** encontra-se a seguir. Você deve usar os recursos aprendidos até o momento para tornar o programa eficiente:

- inicialização e destruição de objetos
- gerência de memória de forma adequada (sem vazamento)
- sobrecarga de construtores e de funções membro
- argumentos _default_ em construtores e funções membro, funções membro _inline_
- passagem de argumentos por referência sempre que possível
- argumentos e funções membro constantes ( _const_ ) sempre que possível

In [ ]:

```
// matrix.h (header file)

#include <iostream>

class Matrix {
    private:

        double** m; // m é um array 2D a ser implementado como um pointer de poi
nters
        int nRows;  // numero de linhas
        int nCols;  //  numero de colunas

    public:

        // Construtores
        Matrix();
        Matrix(int rows, int cols, const double &value = 0.0);
        Matrix(ifstream &myFile)
        Matrix(const Matrix& that)
        // destrutor
        ~Matrix();

        // basic getters
        int getRows() const;
        int getCols() const;
        double get(int row, int col) const;

        // other methods
        void print() const;
        void unit();
        void zeros();
        void ones();

};
```

In [ ]:

```
// matrix.cpp
#include "matrix.h"

// contrutor default - cria uma matriz vazia com nRows = nCols = 0
Matrix::Matrix(){
    ...
}


// contrutor parametrico 1 - cria uma matriz com nRows  = rows, nCols = cols e
// com todos os elementos iguais a 0.0 (double)
Matrix::Matrix(int rows, int cols, const double &value = 0.0){
    ...
}

// contrutor parametrico 2 - cria uma matriz com os dados fornecidos pelo arquiv
o texto myFile.
Matrix::Matrix(fstream &myFile){
    ...
}


// contrutor de copia
Matrix::Matrix(const Matrix& that){
    ...
}


// destrutor
Matrix::~Matrix() {
    ...
}


// obtem o numero de linhas
int Matrix::getRows() const {
    ...
}

// obtem o numero de colunas
int Matrix::getCols() const {
    ...
}


// obtem um elemento específico na posição (row,col). Obs: deve checar consisten
cia
double Matrix::get(int row, int col) const {
    ...
}

// imprime o conteudo da matriz
void Matrix::print() {
    ...
}

// faz com que a matriz torne-se uma matriz identidade
void Matrix::unit(){
```

```
}

// faz com que a matriz torne-se uma matriz nula
void Matrix::zeros(){

}

// faz com que a matriz torne-se uma matriz cujos elementos sao iguaia a 1
void Matrix::ones(){

}
```

**Programa Teste:**

Complemente o programa teste a seguir para que todas as operações implementadas na classe possam ser testadas.

In [ ]:

```
int main()
{
    ifstream in("myMatrix.txt");
    Matrix Y;
    Matrix X(3,1);
    Matrix Z(3,2,7.0);
    Matrix W(in);

    ...

    return 0;
}
```