ELE078 - Programação Orientada a Objetos

Atividade Prática 02

Parte 01: Readequando a TAD Matriz:

Readequar o código do TAD **Matriz**, implementado na Atividade Prática 01, para o formato de um programa orientado a objetos. A interface da classe **Matriz** encontra-se a seguir.

In [ ]:

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

#include <iostream>

class Matrix {
    private:

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

    public:

        // Construtores
        Matrix();
        Matrix(int rows, int cols, double elem);
        // destrutor
        ~Matrix();

        // basic getters
        int getRows() const;
        int getCols() const;

        // other methods
        Matrix transpose();
        void print() const;


};
```

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 elem (double)
Matrix::Matrix(int rows, int cols, double elem){
    ...
}


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


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

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

// retorna uma matriz transposta
Matrix Matrix::transpose() {
    ...
}

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

Programa Teste:

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

In [ ]:

```
// main.cpp

int main()
{
    Matrix Y(3,2);
    Matrix X(4,1);

    std::cout << "Y:: " << std::endl;
    Y.print();
    std::cout << "Numero de linhas de Y:: " << Y.getRows();
    std::cout << "Numero de colunas de Y:: " << Y.getCols();

    std::cout << "Z é transposta de Y:: " << std::endl;
    Matrix Z = Y.transpose();
    Z.print();
    std::cout << "Numero de linhas de Z:: " << Z.getRows();
    std::cout << "Numero de colunas de Z:: " << Z.getCols();

    std::cout << std::endl << "X:: " << std::endl;
    X.print();

    std::cout << "Numero de linhas de X:: " << X.getRows();
    std::cout << "Numero de colunas de X:: " << X.getCols();

    return 0;
}
```

Parte 02: Manipulando o Tempo:

Crie uma classe denominada **Tempo** que contenha 4 atributos inteiros chamados dia (variação de 0 a um valor máximo não limitado) hora (variação de 0 a 23), min (0 a 59) e seg (0 a 59). Separe a definição da classe em um *.h* e a implementação das funções em um *.cpp*.

- (a) Crie uma função membro da classe para checar a consistência dos atributos e modificá-los caso não sejam consistentes. Por exemplo, caso *dia* = 45, *hora* = 30, *min* = 56 e *seg* = 65, estes valores devem ser transformados em *dia* = 46 , *hora* = 6, *min* = 57 e *seg* = 5. Nas funções a seguir a consistência do objeto Tempo resultante deve ser sempre mantida (use esta função para isto)!
- (b) Crie um único construtor para esta classe, que possa ser usado para criar variáveis do tipo Tempo através de declarações como: *Tempo t, t1(72), t2(90,3), t3(4, 7,55), t4(45, 30, 56, 65)*. O primeiro parâmetro é o número de dias, o segundo o número de horas, o terceiro o de minutos e o quarto o de segundos. Repare que no objeto *t4*, os parâmetros passados não configuram um objeto válido. Você deve usar a função de (a) para torná-lo válido.
- (c) Crie uma função membro para somar dois objetos do tipo Tempo. Use a função da letra (a) para manter a consistência do resultado;
- (d) Crie funções membro para leitura e impressão de dados via entrada padrão. (novamente, letra (a) ...);
- (e) Crie funções membro que permitam incrementar e decrementar 1 segundo a um objeto do tipo tempo. (novamente, letra (a) ...);

Dica: Você deve usar os recursos aprendidos até o momento para tornar o programa eficiente:

- encapsulamento
- inicialização e destruição de objetos com contrutores e destrutor
- sobrecarga de construtores e funções membro
- argumentos default em construtores e funções membro