DOCUMENTAÇÃO TRABALHO PRÁTICO II

VÍTOR ALVES DE ANDRADE E AGUIAR

2019

ÍNDICE

1 INTRODUÇÃO

Este trabalho prático propõe que seja desenvolvida, em C++, uma classe para manipulação de matrizes. A classe deve permitir operações matemáticas e a manuseio de matrizes com elementos do tipo *double*. O trabalho ainda tem o requisito de que o acesso aos elementos das matrizes desta classe deve ser feito a partir do índice 1, e não partir do índice 0.

---

2 COMO COMPILAR E EXECUTAR

Para utilizar e/ou avaliar este trabalho prático, será necessário que o código-fonte da classe Matriz (Matriz.h e Matriz.cpp) seja importado para algum projeto na linguagem C++. Feita a importação da classe, basta incluí-la no código da aplicação e explorar a interface da classe.

---

3 IMPLEMENTAÇÃO DO CÓDIGO

O código fonte da classe Matriz foi inteiramente implementado no Microsoft Visual Studio 2019.

3.1 Atributos

Linhas
Número inteiro que representa o número de linhas da matriz.

Colunas
Número inteiro que representa o número de colunas da matriz.

Dados
Estrutura de dado onde são armazenados os ponteiros que apontam para os elementos da matriz.

3.2 Construtores e Destrutor

Matriz ( )

Construtor vazio que inicializa uma matriz vazia, com número de linhas e colunas iguais a 0.

Matriz (*int* linhas, *int* colunas, const *double* & valor)

Construtor que cria um objeto Matriz com determinado número de linhas e colunas. Pode receber a referência de um valor para inicializar todos os elementos da matriz. Se este argumento for nulo, é utilizado seu valor default 0.

Matriz (const Matriz &m)

Construtor de cópia que recebe a referência de um objeto Matriz e instancia uma matriz idêntica à recebida.

~Matriz();

Destrutor que libera a memória alocada dinamicamente para armazenar os elementos do objeto Matriz.

3.3 Interface

getColunas ( ) const { return Colunas; }

Método inline que retorna o número de colunas da matriz.

getLinhas ( ) const { return Linhas; }

Método inline que retorna o número de linhas da matriz.

Matriz operator+ (const Matriz a) const

Sobrecarga do operador + que irá realizar a soma de 2 matrizes, desde que estas tenham as mesmas dimensões. Senão, retorna a própria matriz.

Matriz operator- (const Matriz a) const

Sobrecarga do operador - que irá realizar a subtração de 2 matrizes, desde que estas tenham as mesmas dimensões. Senão, retorna a própria matriz.

void operator+= (const Matriz a)

Sobrecarga do operador += que irá realizar a soma e a atribuição da soma de 2 matrizes ao operando que estiver ao lado esquerdo, desde que estas tenham as mesmas dimensões.

void operator-= (const Matriz a)

Sobrecarga do operador -= que irá realizar a subtração e a atribuição da subtração de 2 matrizes ao operando que estiver ao lado esquerdo, desde que estas tenham as mesmas dimensões.

Matriz operator*(const Matriz a) const

Sobrecarga do operador * que irá realizar a multiplicação de matriz por outra matriz. suas dimensões sejam compatíveis. Retorna a matriz resultado, desde que suas dimensões sejam compatíveis, se não, retorna a própria matriz.

Matriz operator ~()

Sobrecarga do operador ~que irá retornar a matriz transposta do operando que estiver à sua direita.

Matriz operator* (const *double* a) const

Sobrecarga do operador * que irá realizar a multiplicação elemento a elemento da matriz por um número do tipo *double*

void operator*= (const *double* a)

Sobrecarga do operador *= que irá realizar a multiplicação elemento a elemento da matriz por um número de precisão real *double* e a atribuição deste resultado à matriz operanda.

void operator*= (const Matriz &a)

Sobrecarga do operador *= que irá realizar a multiplicação das duas matrizes operandas e a atribuição deste resultado à matriz à esquerda.

bool operator== (const Matriz &a) const

Sobrecarga do operador == que irá retornar o booleano resultado da comparação de equidade entre os operandos da esquerda e da direita.

bool operator!= (const Matriz &a) const

Sobrecarga do operador != que irá retornar o booleano resultado da comparação de inequidade entre os operandos da esquerda e da direita.

ostream& operator« (ostream& out, const & a)

Sobrecarga do operador « que irá formatar e realizar o fluxo de output da matriz operanda.

istream& operator» (istream& in, const Matriz& a)

Sobrecarga do operador » que irá formatar e realizar o fluxo de input da matriz operanda. Por exemplo, o usuário, utilizando do std:cin « <Matriz> , poderá inserir, um a um, os elementos da matriz operanda.

operator= (const Matriz &a)

Sobrecarga do operador = que irá realizar a atribuição dinâmica da matriz operanda da direita à matriz da esquerda, independentemente da estrutura da matriz que receberá a atribuição.

void zeros()

Método que escreve em todos os elementos da matriz o valor de 0.

void inicializaValores (*double* valor)

Método que escreve em todos os elementos da matriz o valor recebido por argumento.

double getDado (*int* linha, *int* coluna) const

Método que retorna o valor do elemento de posição [linha][coluna] da matriz, indexada a partir da posição 1.

void setDado (*int* linha, *int* coluna, *double* valor)

Método que escreve o valor recebido por argumento no elemento de posição [linha][coluna] da matriz, indexada a partir da posição 1.

3.4 Testes

O código abaixo foi utilizado para realizar testes nos métodos implementados na classe Matriz :

```
#include "pch.h"
#include <iostream>
#include "Matriz.h"

int main()
{

  Matriz Y;
  Matriz X(3, 3, 7), A(3, 3, 5), C(3, 3, 3);
  Y = X;
  Matriz W = C;
  Matriz Z(A);
  int numeroLinhas = A.getLinhas();
  int numeroColunas = A.getColunas();
  cout << "A\n" << A;
  A.setDado(2, 1, 10); // altera o valor de uma posi  de A
  Y.zeros(); // modifica todos os elementos de Y para o valor zero
  cout << "A\n" << A;
  C = A + A; // Soma
  cout << "C\n" << C;
  C -= A; // Subtra
  cout << "C\n" << C;
  A = C - A; // Subtra
  cout << "A\n" << A;
  A += A; // Soma
  cout << "A\n" << A;
  A = ~C; // A  igual a transposta de C
  cout << "A\n" << A;
  X = A;
  X *= 2; // multiplica por uma constante
  cout << "X\n" << X;
  C = A * X; // multiplica de matrizes
  cout << "C \n" << C;
  C *= A; // multiplica de matrizes
  cout << "C \n" << C;
  if (A == C) {
    printf("A e igual a C");
  }
  else printf("A e diferente de C");
  // verifica a igualdade entre A e C
  if (X != Y) {
    printf("X e diferente de Y");
  }
  else printf("X e igual Y"); // verifica a desigualdade entre A e C

  cout << C << endl; // Impresso de matrizes
```

```
47    cin >> Y;
48    cout << "Y\n" << Y;
49
50    return 0;
51  }
```

Observações:

- O método utilizado para setar um valor de um elemento da matriz foi implementado de forma diferente da apresentada no código-teste exemplo na especificação do Trabalho Prático. No enunciado, a forma :

  A(2,1)=10;

  foi apresentada.Entretanto, a forma implementada acessa o atributo Dados do objeto, como no exemplo a seguir :

  A.Dados[2][1] = 10;

---

4 CONCLUSÃO

O desenvolvimento do trabalho prático foi muito produtivo, tendo em vista que explorou vastamente a sobrecarga de operadores, dos mais variados tipos, bem como a manipulação de ponteiros duplos. Ambos tópicos apresentaram peculiaridades, dificultando um pouco o desenvolvimento. Além disso, também foram exploradas as funções membro constantes, que colaboram com a robustez do código e a passagem de argumento por referência, que evita a cópia desnecessária de objetos, tornando o código mais eficiente.