PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 12

Tema: Introdução à programação IV
Atividade: Grupos de dados heterogêneos

01.) Editar e salvar um esboço de programa em C++, cujo nome será mymatrix.hpp, que conterá definições para uso posterior:

```
/*
  mymatrix.hpp  - v0.0. - __ / __ / _____
  Author: _____

 */

// ------------------------------------------------ definicoes globais

#ifndef  _MYMATRIX_H_
#define _MYMATRIX_H_

// dependencias

#include <iostream>
using std::cin  ;          // para entrada
using std::cout;           // para saida
using std::endl;           // para mudar de linha

#include <iomanip>
using std::setw;           // para definir espacamento

#include <string>
using std::string;         // para cadeia de caracteres

#include <fstream>
using std::ofstream;       // para gravar arquivo
using std::ifstream;       // para ler    arquivo


template < typename T >
class Matrix
{
 private:                  // area reservada
   T   optional;
   int rows     ;
   int columns;
   T** data     ;
```

```
public                          // area aberta
 Matrix ( )
 {
  // definir valores iniciais
     this->rows      = 0;
     this->columns = 0;
  // sem reservar area
     data                = nullptr;
 } // end constructor

 Matrix ( int rows, int columns, T initial )
 {
  // definir dado local
     bool OK           = true;
  // definir valores iniciais
     this->optional = initial     ;
     this->rows      = rows       ;
     this->columns = columns;
  // reservar area
     data        = new T* [ rows ];
     if ( data != nullptr )
     {
       for ( int x = 0; x < rows; x=x+1 )
       {
         data [x] = new T  [ columns ];
         OK = OK && ( data [x] != nullptr );
       } // end for
       if ( ! OK )
       {
         data = nullptr;
       } // end if
     } // end if
 } // end constructor

~Matrix ( )
{
    if ( data != nullptr )
    {
      for ( int x = 0; x < rows; x=x+1 )
      {
         delete ( data [ x ] );
      } // end for
      delete ( data );
      data = nullptr;
    } // end if
} // end destructor ( )

 void set ( int row, int column, T value )
 {
  if ( row      < 0 || row       >= rows     ||
       column < 0 || column >= columns )
  {
    cout << "\nERROR: Invalid position.\n";
  }
  else
  {
    data [ row ][ column ] = value;
  } // end if
 } // end set ( )
```

```
  T    get ( int row, int column )
  {
   T value = optional;
   if ( row      < 0 || row      >= rows      ||
        column < 0 || column >= columns )
   {
     cout << "\nERROR: Invalid position.\n";
   }
   else
   {
     value = data [ row ][ column ];
   } // end if
   return ( value );
  } // end get ( )

  void print ( )
  {
    cout << endl;
    for ( int x = 0; x < rows; x=x+1 )
    {
       for ( int y = 0; y < columns; y=y+1 )
       {
          cout << data[ x ][ y ] << "\t";
       } // end for
       cout << endl;
    } // end for
    cout << endl;
  } // end print ( )
}; // end class

#endif
```

Editar outro programa em C++, na mesma pasta, cujo nome será Exemplo1200.cpp, para mostrar dados em matriz:

```
/**
  Method_01 - Mostrar certa quantidade de valores.
 */
void method_01 ( )
{
 // definir dados
   Matrix <int> int_matrix ( 2, 2, 0 );

   int_matrix.set ( 0, 0, 1 );         int_matrix.set ( 0, 1, 2 );
   int_matrix.set ( 1, 0, 3 );         int_matrix.set ( 1, 1, 4 );

 // identificar
   cout << "\nMethod_01 - v0.0\n" << endl;

 // mostrar dados
   int_matrix.print ( );

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_01 ( )
```

OBS.:
As referências para matrizes são duplas e precisarão valores iniciais em ambas.
A reciclagem do espaço será feita automaticamente de acordo com à definição do destrutor.

02.) Compilar o programa.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
Em caso de dúvidas, consultar a apostila, recorrer aos monitores ou apresentá-las ao professor.

03.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

04.) Acrescentar na biblioteca outro método para ler e guardar dados em matriz.

```
void read ( )
{
  cout << endl;
  for ( int x = 0; x < rows; x=x+1 )
  {
    for ( int y = 0; y < columns; y=y+1 )
    {
      cout << setw( 2 ) << x << ", "
           << setw( 2 ) << y << " : ";
      cin  >> data[ x ][ y ];
    } // end for
  } // end for
  cout << endl;
} // end read ( )
```

Na parte principal, incluir a chamada do método para testar o novo.

```
/**
  Method_02.
 */
void method_02 ( )
{
 // definir dados
   Matrix <int> matrix ( 2, 2, 0 );

 // identificar
   cout << endl << "Method_02 - v0.0" << endl;

 // ler dados
   matrix.read  ( );

 // mostrar dados
   matrix.print ( );

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_02 ( )
```

OBS.:
Só poderá ser mostrado o arranjo em que existir algum conteúdo
(diferente de **nullptr** = inexistência de dados).

05.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

06.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

07.) Acrescentar na biblioteca outro método para gravar em arquivo dados na matriz.

```
void fprint ( string fileName )
{
  ofstream afile;

  afile.open ( fileName );
  afile << rows      << endl;
  afile << columns << endl;
  for ( int x = 0; x < rows; x=x+1 )
  {
     for ( int y = 0; y < columns; y=y+1 )
     {
        afile << data[ x ][ y] << endl;
     } // end for
  } // end for
  afile.close ( );
} // end fprint ( )
```

Na parte principal, incluir a chamada do método para testar o novo.

```
/**
  Method_03.
 */
void method_03 ( )
{
 // definir dados
   Matrix <int> matrix ( 2, 2, 0 );

 // identificar
   cout << endl << "Method_03 - v0.0" << endl;

 // ler dados
   matrix.read  ( );

 // mostrar dados
   matrix.print ( );

 // gravar  dados
   matrix.fprint( "MATRIX1.TXT" );

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_03 ( )
```

OBS.:
As quantidades de linhas e colunas serão gravadas nas primeiras linhas do arquivo.

08.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

09.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

10.) Acrescentar na biblioteca outro método para ler arquivo e guardar dados em matriz.

```
  void fread ( string fileName )
  {
    ifstream afile;
    int m = 0;
    int n  = 0;

    afile.open ( fileName );
    afile >> m;
    afile >> n;
    if ( m <= 0 || n <= 0 )
    {
      cout << "\nERROR: Invalid dimensions for matrix.\n" << endl;
    }
    else
    {
     // guardar a quantidade de dados
       rows    = m;
       columns = n;
     // reservar area
       data    = new T* [ rows ];
       for ( int x = 0; x < rows; x=x+1 )
       {
          data [x] = new T  [ columns ];
       } // end for
     // ler dados
       for ( int x = 0; x < rows; x=x+1 )
       {
          for ( int y = 0; y < columns; y=y+1 )
          {
             afile >> data[ x ][ y ];
          } // end for
       } // end for
    } // end if
    afile.close ( );
  } // end fread ( )
```

Na parte principal, incluir a chamada do método para testar o novo.

```
/**
   Method_04.
 */
void method_04 ( )
{
 // definir dados
    Matrix <int> matrix ( 1, 1, 0 );

 // identificar
    cout <<  endl << "Method_04 - v0.0" << endl;

 // ler dados
    matrix.fread  ( "MATRIX1.TXT" );

 // mostrar dados
    matrix.print ( );

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_04 ( )
```

OBS.:
Só poderá ser guardada a mesma quantidade de dados lida no início do arquivo, se houver.
Haverá redimensionamento da área reservada para armazenar os valores.

11.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

12.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

13.) Acrescentar na biblioteca outros construtores e
um método para criar um objeto com dados copiados de outras matriz.

```
Matrix& operator= ( const Matrix <T> &other )
{
  if ( other.rows == 0 || other.columns == 0 )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
     this->rows    = other.rows  ;
     this->columns = other.columns;
     this->data    = new T* [ rows ];
     for ( int x = 0; x < rows; x=x+1 )
     {
        this->data [ x ] = new T  [ columns ];
     } // end for
     for ( int x = 0; x < this->rows; x=x+1 )
     {
        for ( int y = 0; y < this->columns; y=y+1 )
        {
           data [ x ][ y ] = other.data [ x ][ y ];
        } // end for
     } // end for
  } // end if
  return ( *this );
} // end operator= ( )
```

Na parte principal, incluir a chamada do método para testar o novo.

```
/**
  Method_05.
 */
void method_05 ( )
{
 // definir dados
    Matrix <int> int_matrix1 ( 1, 1, 0 );
    Matrix <int> int_matrix2 ( 1, 1, 0 );

 // identificar
    cout << endl << "Method_05 - v0.0" << endl;

 // ler dados
    int_matrix1.fread ( "MATRIX1.TXT" );

 // mostrar dados
    cout << "\nOriginal\n" << endl;
    int_matrix1.print ( );

 // copiar  dados
    int_matrix2 = int_matrix1;

 // mostrar dados
    cout << "\nCopia\n" << endl;
    int_matrix2.print ( );

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_05 ( )
```

OBS.:
Só poderá ser copiada a mesma quantidade de dados, se houver espaço suficiente.

14.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

15.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

16.) Acrescentar na biblioteca uma função para testar se a matriz só contém zeros.

```
bool isZeros ( )
{
  bool result = false;
  int x = 0;
  int y = 0;
  if ( rows > 0 && columns > 0 )
  {
    result = true;
    while ( x < rows && result )
    {
        y = 0;
        while ( y < columns && result )
        {
            result = result && ( data [ x ][ y ] == 0 );
            y = y + 1;
        } // end while
        x = x + 1;
    } // end while
  } // end if
  return ( result );
} // end isZeros ( )
```

Na parte principal, incluir a chamada do método para testar a função.

```
/**
  Method_06.
 */
void method_06 ( )
{
 // definir dados
   Matrix <int> int_matrix ( 2, 2, 0 );

   int_matrix.set ( 0, 0, 0 );    int_matrix.set ( 0, 1, 0 );
   int_matrix.set ( 1, 0, 0 );    int_matrix.set ( 1, 1, 0 );

 // identificar
   cout << endl << "Method_06 - v0.0" << endl;

 // mostrar dados
   int_matrix.print ( );

 // testar  condicao
   cout << "Zeros = " << int_matrix.isZeros ( ) << endl;

 // ler dados
   int_matrix.fread ( "MATRIX1.TXT" );

 // mostrar dados
   int_matrix.print ( );

 // testar  condicao
   cout << "Zeros = " << int_matrix.isZeros ( ) << endl;

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_06 ( )
```

17.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

18.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

19.) Acrescentar na biblioteca um operador para testar se matrizes são diferentes.

```
bool operator!= ( const Matrix <T> &other )
{
  bool result = false;
  int    x       = 0;
  int    y       = 0;

  if ( other.rows      == 0 || rows       != other.rows     ||
       other.columns == 0 || columns != other.columns )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
    x = 0;
    while ( x < rows && ! result )
    {
        y = 0;
        while ( y < columns && ! result )
        {
            result = ( data [ x ][ y ] != other.data [ x ][ y ] );
            y = y + 1;
        } // end while
        x = x + 1;
    } // end while
  } // end if
  return ( result );
} // end operator!= ( )
```

Na parte principal, incluir a chamada do método para testar a função.

```
/**
  Method_07.
 */
void method_07 ( )
{
 // definir dados
    Matrix <int> int_matrix1 ( 1, 1, 0 );
    Matrix <int> int_matrix2 ( 1, 1, 0 );

 // identificar
    cout << endl << "Method_07 - v0.0" << endl;

 // ler dados
    int_matrix1.fread ( "MATRIX1.TXT" );

 // mostrar dados
    cout << "\nMatrix_1\n";
    int_matrix1.print ( );

 // copiar  dados
    int_matrix2 = int_matrix1;

 // mostrar dados
    cout << "\nMatrix_2\n";
    int_matrix2.print ( );

 // testar  condicao
    cout << "Diferentes = " << (int_matrix1!=int_matrix2) << endl;

 // alterar dados
    int_matrix2.set ( 0, 0, (-1) );

 // mostrar dados
    cout << "\nMatrix_1\n";
    int_matrix1.print ( );

 // mostrar dados
    cout << "\nMatrix_2\n";
    int_matrix2.print ( );

 // testar  condicao
    cout << "Diferentes = " << (int_matrix1!=int_matrix2) << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_07 ( )
```

OBS.:
Só poderão ser comparadas matrizes com as mesmas dimensões.

20.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

21.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

22.) Acrescentar um método para subtrair dados em matrizes, posição por posição.

```
Matrix& operator- ( const Matrix <T> &other )
{
  static Matrix <T> result ( 1, 1, 0 );
  int     x     = 0;
  int     y     = 0;

  if ( other.rows      == 0 || rows       != other.rows ||
       other.columns == 0 || columns != other.columns )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
    result.rows      = rows;
    result.columns = other.columns;
    result.data        = new T* [ result.rows ];
    for ( int x = 0; x < result.rows; x=x+1 )
    {
       result.data [x] = new T  [ result.columns ];
    } // end for

    for ( int x = 0; x < result.rows; x=x+1 )
    {
       for ( int y = 0; y < result.columns; y=y+1 )
       {
          result.data [ x ][ y ] = data [ x ][ y ] - other.data [ x ][ y ];
       } // end for
    } // end for
  } // end if
  return ( result );
} // end operator- ( )
```

Na parte principal, incluir a chamada do método para testar a operação.

```
/**
  Method_08.
 */
void method_08 ( )
{
 // definir dados
    Matrix <int> int_matrix1 ( 1, 1, 0 );
    Matrix <int> int_matrix2 ( 1, 1, 0 );
    Matrix <int> int_matrix3 ( 1, 1, 0 );

 // identificar
    cout << endl << "Method_08 - v0.0" << endl;

 // ler dados
    int_matrix1.fread ( "MATRIX1.TXT" );

 // mostrar dados
    cout << "\nMatrix_1\n";
    int_matrix1.print ( );

 // copiar  dados
    int_matrix2 = int_matrix1;

 // mostrar dados
    cout << "\nMatrix_2\n";
    int_matrix2.print ( );

 // operar  dados
    int_matrix3 = int_matrix1 - int_matrix2;

 // mostrar dados
    cout << "\nMatrix_3\n";
    int_matrix3.print ( );

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_08 ( )
```

OBS.:
Só poderão ser operadas matrizes com as mesmas dimensões.

23.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

24.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

25.) Acrescentar um operador para calcular o produto de matrizes.

```
Matrix& operator* ( const Matrix <T> &other )
{
  static Matrix <T> result ( 1, 1, 0 );
  int x     = 0;
  int y     = 0;
  int z     = 0;
  int sum = 0;

  if (          rows <= 0 ||          columns == 0 ||
      other.rows <= 0 || other.columns == 0 ||
            columns    != other.rows               )
  {
    cout << endl << "ERROR: Invalid data." << endl;
    result.data [ 0 ][ 0 ] = 0;
  }
  else
  {
    result.rows       = rows;
    result.columns = other.columns;
    result.data         = new T* [ result.rows ];
    for ( int x = 0; x < result.rows; x=x+1 )
    {
       result.data [x] = new T  [ result.columns ];
    } // end for

    for ( x = 0; x < result.rows; x = x + 1 )
    {
       for ( y = 0; y < result.columns; y = y + 1 )
       {
          sum = 0;
          for ( z = 0; z < columns; z = z + 1 )
          {
             sum = sum + data [ x ][ z ] *  other.data [ z ][ y ];
          } // end for
          result.data [ x ][ y ] = sum;
       } // end for
    } // end for
  } // end if
  return ( result );
} // end operator* ( )
```

Na parte principal, incluir a chamada do método para testar a operação.

```
/**
  Method_09.
 */
void method_09 ( )
{
 // definir dados
    Matrix <int> int_matrix1 ( 2, 2, 0 );

    int_matrix1.set ( 0, 0, 1 );
    int_matrix1.set ( 0, 1, 0 );
    int_matrix1.set ( 1, 0, 0 );
    int_matrix1.set ( 1, 1, 1 );

    Matrix <int> int_matrix2 ( 1, 1, 0 );
    Matrix <int> int_matrix3 ( 1, 1, 0 );

 // identificar
    cout << endl << "Method_09 - v0.0" << endl;

 // ler dados
    int_matrix2.fread ( "MATRIX1.TXT" );

 // mostrar dados
    cout << "\nMatrix_1\n";
    int_matrix1.print ( );

 // mostrar dados
    cout << "\nMatrix_2\n";
    int_matrix2.print ( );

 // operar  dados
    int_matrix3 = int_matrix1 * int_matrix2;

 // mostrar dados
    cout << "\nMatrix_3\n";
    int_matrix3.print ( );

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_09 ( )
```

OBS.:
Só poderão ser operadas matrizes com dimensões compatíveis.

26.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

27.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

28.) Acrescentar na biblioteca para acessos externos aos valores em matriz.

```
const int getRows ( )
{
  return ( rows );
} // end getRows ( )

const int getColumns ( )
{
  return ( columns );
} // end getColumns ( )
```

Na parte principal, incluir a chamada do método para testar a função.

```
/**
  Method_10.
 */
void method_10 ( )
{
 // definir dados
   Matrix <int> int_matrix ( 3, 3, 0 );
   int x = 0;
   int y = 0;

 // identificar
   cout <<  endl << "Method_10 - v0.0" << endl;

 // ler dados
   int_matrix.fread ( "MATRIX1.TXT" );

 // mostrar dados
   cout << "\nMatrix\n";
   int_matrix.print ( );

 // operar  dados
   for ( int x = 0; x < int_matrix.getRows ( ); x=x+1 )
   {
      for ( int y = 0; y < int_matrix.getColumns ( ); y=y+1 )
      {
         int_matrix.set ( x, y, int_matrix.get ( x, y ) * (-1) );
      } // end for
   } // end for

 // mostrar dados
   cout << "\nMatrix\n";
   int_matrix.print ( );

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_10 ( )
```

OBS.:
Só poderá haver acesso se houver dados e somente serão acessadas posições válidas.

29.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

30.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

Exercícios

DICAS GERAIS: Consultar o Anexo C 02 na apostila para outros exemplos.
Não usar métodos ou funções já prontos em bibliotecas nativas da linguagem.
Prever, realizar e registrar todos os testes efetuados.
Integrar as chamadas de todos os programas em um só.

01.) Incluir métodos (1211) para
ler a quantidade de elementos (MxN) a serem gerados;
gerar essa quantidade (MxN) de valores aleatórios
dentro do intervalo e armazená-los em matriz;
gravá-los, um por linha, em um arquivo ("DADOS.TXT").
A primeira linha do arquivo deverá informar a quantidade
de números aleatórios (N) que serão gravados em seguida.
DICA: Usar a função **rand**( ), mas tentar limitar valores ao intervalo [1:100].

Exemplo: matrix.randomIntGenerate (inferior, superior);

02.) Incluir uma função (1212) para
escalar uma matriz, multiplicando todos os seus valores por uma constante.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre a matriz com os valores lidos.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix2 = matrix1.scalar (3); // multiplicar cada valor pelo argumento

03.) Incluir uma função (1213) para
testar se uma matriz é a identidade.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre a matriz com os valores lidos.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
teste = matrix1.identidade ( );

04.) Incluir em um programa (Exemplo1214) um operador para
testar a igualdade de duas matrizes.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix2 = readMatrixFromFile ("DADOS2.TXT");
teste = (matrix1 == matrix2);

]

05.) Incluir uma função (1215) para
somar duas matrizes e mostrar o resultado.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix2 = readMatrixFromFile ("DADOS2.TXT");
soma = matrix1.add (matrix2);

06.) Incluir uma função (1216) para
operar duas linhas da matriz, guardando no lugar da primeira,
as somas de cada elemento da primeira linha com o respectivo da segunda linha
multiplicados por uma constante.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix1.addRows (0, 1, (-1));

07.) Incluir uma função (1217) para
operar duas linhas da matriz, guardando no lugar da primeira,
as diferenças de cada elemento da primeira linha com o respectivo da segunda linha
multiplicado por uma constante.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix1.subtractRows (0, 1, (2));

08.) Incluir uma função (1218) para
dizer em qual linha da matriz se encontra certo valor, se houver.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
teste = matrix1.searchRows (procurado);

09.) Incluir uma função (1219) para
dizer em qual coluna da matriz se encontra certo valor, se houver.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
teste = matrix1.searchColumns (procurado);

10.) Incluir uma função (1220) para
transpor os dados em uma matriz.

Exemplo: matrix1 = readMatrixFromFile ("DADOS1.TXT");
matrix1.transpose ( );

Tarefas extras

E1.) Incluir uma função (12E1) para
dizer se uma matriz apresenta a característica abaixo.

					1	5	9	13
		1	4	7	2	6	10	14
1	3	2	5	8	3	7	11	15
2	4	3	6	9	4	8	12	16

E2.) Incluir uma função (12E2) para
montar uma matriz com a característica abaixo.

					16	12	8	4
		9	6	3	15	11	7	3
4	2	8	5	2	14	10	6	2
3	1	7	4	1	13	9	5	1