

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=0, int columns=0, 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 Exemplo1201.cpp, para mostrar dados em matriz:

```

/**
 * Method01 - Mostrar certa quantidade de valores.
 */
void method01 ( )
{
    // 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 << "\nEXEMPLO1210 - Method01 - v0.0\n" << endl;

    // mostrar dados
    int_matrix.print ( );

    // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method01 ( )

```

OBS.:

As referências para matrizes são duplas e precisarão valores iniciais em ambas.

A reciclagem do espaço será feita automaticamente de acordo com a 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.) Copiar a versão atual do programa para outra nova – Exemplo1202.cpp.

05.) Editar mudanças no nome do programa e versão.

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.

```
/**
    Method02.
*/
void method02 ( )
{
    // definir dados
    Matrix <int> matrix ( 2, 2, 0 );

    // identificar
    cout << endl << "EXEMPLO1210 - Method02 - v0.0" << endl;

    // ler dados
    matrix.read ( );

    // mostrar dados
    matrix.print ( );

    // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method02 ( )
```

OBS.:

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

- 06.) 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.
- 07.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 08.) Copiar a versão atual do programa para outra nova – Exemplo1203.cpp.
- 09.) Editar mudanças no nome do programa e versão.
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.

```
/**
    Method03.
*/
void method03 ( )
{
    // definir dados
    Matrix <int> matrix ( 2, 2, 0 );

    // identificar
    cout << endl << "EXEMPLO1210 - Method03 - v0.0" << endl;

    // ler dados
    matrix.read ( );

    // mostrar dados
    matrix.print ( );

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

    // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method03 ( )
```

OBS.:

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

- 10.) 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.
- 11.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 12.) Copiar a versão atual do programa para outra nova – Exemplo1204.cpp.
- 13.) Editar mudanças no nome do programa e versão.
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.

```
/**
    Method04.
*/
void method04 ( )
{
    // definir dados
    Matrix <int> matrix ( 1, 1, 0 );

    // identificar
    cout << endl << "EXEMPLO1210 - Method04 - v0.0" << endl;

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

    // mostrar dados
    matrix.print ( );

    // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method04 ( )
```

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.

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.) Copiar a versão atual do programa para outra nova – Exemplo1205.cpp.

- 17.) Editar mudanças no nome do programa e versão.
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.

```
/**
 * Method05.
 */
void method05 ( )
{
    // definir dados
    Matrix <int> int_matrix1 ( 1, 1, 0 );
    Matrix <int> int_matrix2 ( 1, 1, 0 );

    // identificar
    cout << endl << "EXEMPLO1210 - Method05 - 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" );
} // fim method05 ( )
```

OBS.:

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

18.) 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.

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

20.) Copiar a versão atual do programa para outra nova – Exemplo1206.cpp.

21.) Editar mudanças no nome do programa e versão.

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 for
            x = x + 1;
        } // end while
    } // end if
    return ( result );
} // end isZeros (
```

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

```
/**
    Method06.
*/
void method06 ( )
{
    // 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 << "EXEMPLO1210 - Method06 - 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" );
} // fim method06 (
```

- 22.) 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.
- 23.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 24.) Copiar a versão atual do programa para outra nova – Exemplo1207.cpp.
- 25.) Editar mudanças no nome do programa e versão.
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 for
            x = x + 1;
        } // end for
    } // end if
    return ( result );
} // end operator!= ( )
```

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

```
/**
  Method07.
 */
void method07 ( )
{
  // definir dados
  Matrix <int> int_matrix1 ( 1, 1, 0 );
  Matrix <int> int_matrix2 ( 1, 1, 0 );

  // identificar
  cout << endl << "EXEMPLO1210 - Method07 - 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" );
} // fim method07 ( )
```

OBS.:

Só poderão ser comparadas matrizes com as mesmas dimensões.

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.) Copiar a versão atual do programa para outra nova – Exemplo1208.cpp.

29.) Editar mudanças no nome do programa e versão.

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.

```
/**
    Method08.
*/
void method08 ( )
{
    // 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 << "EXEMPLO1210 - Method08 - 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" );
} // fim method08 ( )
```

OBS.:

Só poderão ser operadas matrizes com as mesmas dimensões.

30.) 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.

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

32.) Copiar a versão atual do programa para outra nova – Exemplo1209.cpp.

- 33.) Editar mudanças no nome do programa e versão.
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.

```
/**
 * Method09.
 */
void method09 ( )
{
    // 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 << "EXEMPLO1210 - Method09 - 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" );
} // fim method09 ( )
```

OBS.:

Só poderão ser operadas matrizes com dimensões compatíveis.

34.) 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.

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

36.) Copiar a versão atual do programa para outra nova – Exemplo1210.c.

37.) Editar mudanças no nome do programa e versão.

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.

```
/**
    Method10.
*/
void method10 ( )
{
    // definir dados
    Matrix <int> int_matrix ( 3, 3, 0 );
    int x = 0;
    int y = 0;

    // identificar
    cout << endl << "EXEMPLO1210 - Method10 - 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" );
} // fim method10 ( )
```

OBS.:

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

38.) 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.

39.) 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.

Prever, realizar e registrar todos os testes efetuados.

Integrar as chamadas de todos os programas em um só.

01.) Incluir em um programa (Exemplo1211)

ler a quantidade de elementos ($M \times N$) a serem gerados;

gerar essa quantidade ($M \times N$) 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 muito grandes.

Exemplo: `matrix.gerarInt (inferior, superior);`

02.) Incluir em um programa (Exemplo1212) uma função 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 = lerArquivo ("DADOS1.TXT");`

`matrix2 = matrix1.escalar (2);`

03.) Incluir em um programa (Exemplo1213) uma função 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 = lerArquivo ("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 = lerArquivo ("DADOS1.TXT");`

`matrix2 = lerArquivo ("DADOS2.TXT");`

`teste = (matrix1 == matrix2);`

- 05.) Incluir em um programa (Exemplo1215) um operador para somar duas matrizes.

Para testar, receber um nome de arquivo como parâmetro e aplicar a função sobre o arranjo com os valores lidos.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        matrix2 = lerArquivo ( "DADOS2.TXT" );  
        soma    = matrix1 + matrix2;
```

- 06.) Incluir em um programa (Exemplo1216) uma função para operar duas linhas da matriz, guardando no lugar da primeira, as somas de cada elemento da primeira com o respectivo da segunda multiplicados por uma constante.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        matrix1.somarLinhas ( 0, 1, (-1) );
```

- 07.) Incluir em um programa (Exemplo1217) uma função para operar duas colunas da matriz, guardando no lugar da primeira, as diferenças de cada elemento da primeira com o respectivo da segunda multiplicado por uma constante.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        matrix1.subtrairLinhas ( 0, 1, (2) );
```

- 08.) Incluir em um programa (Exemplo1218) uma função para dizer em qual linha da matriz se encontra certo valor, se houver.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        teste    = matrix1.procurarLinhas ( procurado );
```

- 09.) Incluir em um programa (Exemplo1219) uma função para dizer em qual coluna da matriz se encontra certo valor, se houver.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        teste    = matrix1.procurarColunas ( procurado );
```

- 10.) Incluir em um programa (Exemplo1220) uma função para transpor os dados em uma matriz.

```
Exemplo: matrix1 = lerArquivo ( "DADOS1.TXT" );  
        matrix1.transpor ( );
```

Tarefas extras

E1.) Incluir em um programa (Exemplo12E1) uma função para dizer se uma matriz apresenta a característica abaixo.

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

E2.) Incluir em um programa (Exemplo12E2) uma função para montar uma matriz com 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