

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

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo0900.c, para mostrar dados em matriz:

```
/**
    printIntMatrix    - Mostrar arranjo bidimensional com valores inteiros.
    @param rows      - quantidade de linhas
    @param columns    - quantidade de colunas
    @param matrix     - grupo de valores inteiros
*/
void printIntMatrix ( int rows, int columns, int matrix[ ][columns] )
{
    // definir dado local
    int x = 0;
    int y = 0;

    // mostrar valores na matriz
    for ( x=0; x<rows; x=x+1 )
    {
        for ( y=0; y<columns; y=y+1 )
        {
            // mostrar valor
            IO_printf ( "%3d\t", matrix [ x ][ y ] );
        } // end for
        IO_printf ( "\n" );
    } // end for
} // end printIntMatrix ( )

/**
    Method_01 - Mostrar certa quantidade de valores.
*/
void method_01 ( )
{
    // definir dado
    int matrix [ ][3] = {
                                {1, 2, 3},
                                {4, 5, 6},
                                {7, 8, 9}
                            };

    // identificar
    IO_id ( "Method_01 - v0.0" );

    // executar o metodo auxiliar
    printIntMatrix ( 3, 3, matrix );

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

OBS.:

A atribuição direta de todos os valores à matriz só é permitida quando da sua definição.

A ordem dos parâmetros recomendada deve trazer a quantidade de linhas e colunas antes do armazenador da matriz, e a quantidade de colunas deve ser indicada.

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 outro método para ler e guardar dados em matriz.

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

```
/**
    readIntMatrix    - Ler arranjo bidimensional com valores inteiros.
    @param rows      - quantidade de linhas
    @param columns    - quantidade de colunas
    @param matrix     - grupo de valores inteiros
*/
void readIntMatrix ( int rows, int columns, int matrix[ ][columns] )
{
    // definir dados locais
    int x = 0;
    int y = 0;
    int z = 0;
    chars text = IO_new_chars ( STR_SIZE );

    // ler e guardar valores em arranjo
    for ( x=0; x<rows; x=x+1 )
    {
        for ( y=0; y<columns; y=y+1 )
        {
            // ler valor
            strcpy ( text, STR_EMPTY );
            z = IO_readint ( IO_concat (
                                IO_concat ( IO_concat ( text, IO_toString_d ( x ) ), " , " ),
                                IO_concat ( IO_concat ( text, IO_toString_d ( y ) ), " : " ) ) );

            // guardar valor
            matrix [ x ][ y ] = z;
        } // end for
    } // end for
} // end readIntMatrix ( )
```

```

/**
    Method_02.
*/
void method_02 ( )
{
    // definir dados
    int n = 2;           // quantidade de valores
    int matrix [ n ][ n ];

    // identificar
    IO_id ( "Method_02 - v0.0" );

    // ler dados
    readIntMatrix ( n, n, matrix );

    // mostrar dados
    IO_printf ( "\n" );
    printIntMatrix ( n, n, matrix );

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

```

OBS.:

Só poderá ser mostrado a matriz em que existir algum conteúdo  
(diferente de **NULL** = 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 outro método para gravar em arquivo dados na matriz.  
Na parte principal, incluir a chamada do método para testar o novo.

```

/**
    fprintIntMatrix    - Gravar arranjo bidimensional com valores inteiros.
    @param fileName   - nome do arquivo
    @param rows        - quantidade de linhas
    @param columns     - quantidade de colunas
    @param matrix      - grupo de valores inteiros
*/
void fprintIntMatrix ( chars fileName, int rows, int columns, int matrix[ ][columns] )
{
    // definir dados locais
    FILE* arquivo = fopen ( fileName, "wt" );
    int x = 0;
    int y = 0;

    // gravar quantidade de dados
    IO_fprintf ( arquivo, "%d\n", rows );
    IO_fprintf ( arquivo, "%d\n", columns );

```

```

// gravar valores no arquivo
for ( x=0; x<rows; x=x+1 )
{
    for ( y=0; y<columns; y=y+1 )
    {
        // gravar valor
        IO_printf ( arquivo, "%d\n", matrix [ x ][ y ] );
    } // end for
} // end for

// fechar arquivo
fclose ( arquivo );
} // end fprintfIntMatrix ( )

/**
    Method_03.
*/
void method_03 ( )
{
    // definir dados
    int rows    = 0;
    int columns = 0;

    // identificar
    IO_id ( "Method_03 - v0.0" );

    // ler dados
    rows = IO_readint ( "\nrows    = " );
    columns = IO_readint ( "\ncolumns = " );
    IO_printf ( "\n" );

    if ( rows <= 0 || columns <= 0 )
    {
        IO_println ( "\nERRO: Valor invalido." );
    }
    else
    {
        // reservar espaco
        int matrix [ rows ][ columns ];
        // ler dados
        readIntMatrix ( rows, columns, matrix );
        // mostrar dados
        IO_printf ( "\n" );
        printIntMatrix ( rows, columns, matrix );
        // gravar dados
        IO_printf ( "\n" );
        fprintfIntMatrix( "MATRIX1.TXT", rows, columns, matrix );
    } // end if

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

```

OBS.:

Se existir dados na matriz original, eles serão sobrescritos.

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 outra função para ler arquivo e guardar dados em matriz.  
Na parte principal, incluir a chamada do método para testar o novo.

```
/**
 freadMatrixRows - Ler tamanho (linhas) da matriz com valores inteiros.
 @return quantidade de linhas da matriz
 @param fileName - nome do arquivo
 */
int freadMatrixRows ( chars fileName )
{
    // definir dados locais
    int n = 0;
    FILE* arquivo = fopen ( fileName, "rt" );
    ints array = NULL;

    // ler a quantidade de dados
    IO_fscanf ( arquivo, "%d", &n );

    if ( n <= 0 )
    {
        IO_println ( "ERRO: Valor invalido." );
        n = 0;
    } // end if

    // retornar dado lido
    return ( n );
} // end freadMatrixRows ( )

/**
 freadMatrixColumns - Ler tamanho (colunas) da matriz com valores inteiros.
 @return quantidade de colunas da matriz
 @param fileName - nome do arquivo
 */
int freadMatrixColumns ( chars fileName )
{
    // definir dados locais
    int n = 0;
    FILE* arquivo = fopen ( fileName, "rt" );

    // ler a quantidade de dados
    IO_fscanf ( arquivo, "%d", &n );
    IO_fscanf ( arquivo, "%d", &n );

    if ( n <= 0 )
    {
        IO_println ( "ERRO: Valor invalido." );
        n = 0;
    } // end if

    // retornar dado lido
    return ( n );
} // end freadMatrixColumns ( )
```

```

/**
freadIntMatrix    - Ler arranjo bidimensional com valores inteiros.
@param fileName  - nome do arquivo
@param rows      - quantidade de valores
@param columns   - quantidade de valores
@param matrix    - grupo de valores inteiros
*/
void freadIntMatrix ( chars fileName, int rows, int columns, int matrix[ ][columns] )
{
    // definir dados locais
    int x = 0;
    int y = 0;
    int z = 0;
    FILE* arquivo = fopen ( fileName, "rt" );

    // ler a quantidade de dados
    IO_fscanf ( arquivo, "%d", &x );
    IO_fscanf ( arquivo, "%d", &y );

    if ( rows <= 0 || rows > x ||
        columns <= 0 || columns > y )
    {
        IO_println ( "ERRO: Valor invalido." );
    }
    else
    {
        // ler e guardar valores em arranjo
        x = 0;
        while ( ! feof ( arquivo ) && x < rows )
        {
            y = 0;
            while ( ! feof ( arquivo ) && y < columns )
            {
                // ler valor
                IO_fscanf ( arquivo, "%d", &z );
                // guardar valor
                matrix [ x ][ y ] = z;
                // passar ao proximo
                y = y+1;
            } // end while
            // passar ao proximo
            x = x+1;
        } // end while
    } // end if

    // fechar arquivo
    fclose ( arquivo );
} // end freadIntMatrix ( )

```

```

/**
    Method_04.
*/
void method_04 ( )
{
    // definir dados
    int rows = 0;
    int columns = 0;

    // identificar
    IO_id ( "Method_04 - v0.0" );

    // ler dados
    rows = freadMatrixRows ( "MATRIX1.TXT" );
    columns = freadMatrixColumns ( "MATRIX1.TXT" );

    if ( rows <= 0 || columns <= 0 )
    {
        IO_println ( "\nERRO: Valor invalido." );
    }
    else
    {
        // definir armazenador
        int matrix [ rows ][ columns ];

        // ler dados
        freadIntMatrix ( "MATRIX1.TXT", rows, columns, matrix );

        // mostrar dados
        IO_printf ( "\n" );
        printIntMatrix ( rows, columns, matrix );
    } // end if

    // encerrar
    IO_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.

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 uma função para copiar dados de uma matriz para outra.  
Na parte principal, incluir a chamada do método para testar o novo.

```
/**
    copyIntMatrix    - Copiar matriz com valores inteiros.
    @param rows      - quantidade de valores
    @param columns    - quantidade de valores
    @param matrix     - grupo de valores inteiros
*/
void copyIntMatrix ( int rows, int columns,
                    int matrix2[ ][columns], int matrix1[ ][columns] )
{
    // definir dados locais
    int x = 0;
    int y = 0;

    if ( rows <= 0 || columns <= 0 )
    {
        IO_println ( "ERRO: Valor invalido." );
    }
    else
    {
        // copiar valores em matriz
        for ( x = 0; x < rows; x = x + 1 )
        {
            for ( y = 0; y < columns; y = y + 1 )
            {
                // copiar valor
                matrix2 [ x ][ y ] = matrix1 [ x ][ y ];
            } // end for
        } // end for
    } // end if
} // end copyIntMatrix ( )

/**
    Method_05.
*/
void method_05 ( )
{
    // definir dados
    int rows    = 0;
    int columns = 0;

    // identificar
    IO_id ( "Method_05 - v0.0" );

    // ler dados
    rows    = freadMatrixRows ( "MATRIX1.TXT" );
    columns = freadMatrixColumns ( "MATRIX1.TXT" );
}
```



```

if ( rows <= 0 || columns <= 0 )
{
    IO_printf ( "\nERRO: Valor invalido." );
}
else
{
    // definir armazenadores
    int matrix [ rows ][ columns ];
    int other  [ rows ][ columns ];

    // ler dados
    freadIntMatrix ( "MATRIX1.TXT", rows, columns, matrix );

    // copiar dados
    copyIntMatrix ( rows, columns, other, matrix );

    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntMatrix ( rows, columns, matrix );

    // mostrar dados
    IO_printf ( "\nCopial\n" );
    printIntMatrix ( rows, columns, other );
} // end if

// encerrar
IO_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 outra função para fazer a transposição de uma matriz.  
Na parte principal, incluir a chamada do método para testar a função.

```

/**
transposeIntMatrix - Transpor valores inteiros em matriz.
@param rows       - quantidade de valores
@param columns    - quantidade de valores
@param matrix2    - grupo de valores inteiros (transposto)
@param matrix1    - grupo de valores inteiros
*/
void transposeIntMatrix ( int rows, int columns,
                        int matrix2[ ][rows] , int matrix1[ ][columns] )
{
    // definir dados locais
    int x    = 0;
    int y    = 0;

```

```

// percorrer a matriz
for ( x = 0; x<rows; x=x+1 )
{
    for ( y = 0; y<columns; y=y+1 )
    {
        matrix2[ y ][ x ] = matrix1 [ x ][ y ];
    } // end for
} // end for
} // end transposeIntMatrix ( )

/**
 * Method_06.
 */
void method_06 ( )
{
    // definir dados
    int matrix1 [ ][2] = { {1, 0} ,
                           {0, 1} };
    int matrix2 [ ][2] = { {0, 0} ,
                           {0, 0} };
    int matrix3 [ ][3] = { {1, 2, 3} ,
                           {4, 5, 6} };
    int matrix4 [ ][3] = { {1, 2, 3} ,
                           {4, 5, 6} ,
                           {7, 8, 9} };

    // identificar
    IO_id ( "Method_06 - v0.0" );

    // testar dados
    IO_println ( "\nMatrix1 " );
    printIntMatrix( 2, 2, matrix1 );

    transposeIntMatrix ( 2, 2, matrix2, matrix1 );

    IO_println ( "\nMatrix2" );
    printIntMatrix( 2, 2, matrix2 );

    IO_println ( "\nMatrix3" );
    printIntMatrix( 2, 3, matrix3 );

    transposeIntMatrix ( 2, 3, matrix4, matrix3 );

    IO_println ( "\nMatrix4" );
    printIntMatrix( 3, 2, matrix4 );

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

```

OBS.:

As quantidades de linha e colunas estarão trocadas na matriz transposta.

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 uma função para dizer se uma matriz é identidade.  
Na parte principal, incluir a chamada do método para testar a função.

```
/**
  isidentity      - Testar se matriz identidade.
  @return         - true, se todos os dados forem iguais a zero;
                  false, caso contrario
  @param rows     - quantidade de valores
  @param columns  - quantidade de valores
  @param matrix   - grupo de valores inteiros
*/
bool isidentity ( int rows, int columns, int matrix[ ][columns] )
{
  // definir dados locais
  bool result = false;
  int  x      = 0;
  int  y      = 0;

  // testar condicao de existencia
  if ( rows <= 0 || columns <= 0 ||
      rows != columns )
  {
    IO_printf ( "\nERRO: Valor invalido.\n" );
  }
  else
  {
    // testar valores na matriz
    x = 0;
    result = true;
    while ( x < rows && result )
    {
      y = 0;
      while ( y < columns && result )
      {
        // testar valor
        if ( x == y )
        {
          result = result && ( matrix [ x ][ y ] == 1 );
        }
        else
        {
          result = result && ( matrix [ x ][ y ] == 0 );
        } // end if
        // passar ao proximo
        y = y + 1;
      } // end while
      // passar ao proximo
      x = x + 1;
    } // end while
  } // end if

  // retornar resposta
  return ( result );
} // end isidentity ( )
```

```

/**
  Method_07.
 */
void method_07 ( )
{
  // definir dados
  int matrix1 [ ][2] = { {0, 0} ,
                        {0, 0} };
  int matrix2 [ ][3] = { {1, 2, 3} ,
                        {4, 5, 6} };
  int matrix3 [ ][2] = { {1, 0} ,
                        {0, 1} };

  // identificar
  IO_id ( "Method_07 - v0.0" );

  // testar dados
  IO_println ( "\nMatrix1" );
  printIntMatrix( 2, 2, matrix1 );
  IO_printf ( "isIdentity (matrix1) = %d", isIdentity (2, 2, matrix1) );

  IO_println ( "\nMatrix2" );
  printIntMatrix( 2, 3, matrix2 );
  IO_printf ( "isIdentity (matrix2) = %d", isIdentity (2, 3, matrix2) );

  IO_println ( "\nMatrix3" );
  printIntMatrix( 2, 2, matrix3 );
  IO_printf ( "isIdentity (matrix3) = %d", isIdentity (2, 2, matrix3) );

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

```

OBS.:

Só deverá ser verificado a matriz for quadrada (quantidade de linhas e colunas iguais).

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 uma função para testar a igualdade de dados em duas matrizes, posição por posição.  
Na parte principal, incluir a chamada do método para testar a função.

```
/**
    isEqual          - Testar se matrizes iguais.
    @return          - true, se todos os dados forem iguais;
                    - false, caso contrario
    @param rows      - quantidade de valores
    @param columns    - quantidade de valores
    @param matrix1    - grupo de valores inteiros (1)
    @param matrix2    - grupo de valores inteiros (2)
*/
bool isEqual ( int rows, int columns,
               int matrix1[ ][columns], int matrix2[ ][columns] )
{
    // definir dados locais
    bool result = true;
    int  x      = 0;
    int  y      = 0;

    // mostrar valores na matriz
    x = 0;
    while ( x < rows && result )
    {
        y = 0;
        while ( y < columns && result )
        {
            // testar valor
            result = result &&
                ( matrix1 [ x ][ y ] == matrix2 [ x ][ y ] );
            // passar ao proximo
            y = y + 1;
        } // end while
        // passar ao proximo
        x = x + 1;
    } // end while

    // retornar resposta
    return ( result );
} // end isEqual ( )

/**
    Method_09.
*/
void method_09 ( )
{
    // definir dados
    int matrix1 [ ][2] = { {0, 0} ,
                          {0, 0} };
    int matrix2 [ ][2] = { {1, 2} ,
                          {3, 4} };
    int matrix3 [ ][2] = { {1, 0} ,
                          {0, 1} };

    // identificar
    IO_id ( "Method_08 - v0.0" );
}
```

```

// testar dados
IO_println ( "\nMatrix1" );
printIntMatrix( 2, 2, matrix1 );

IO_println ( "\nMatrix2" );
printIntMatrix( 2, 2, matrix2 );

IO_printf ( "isEqual (matrix1, matrix2) = %d",
            isEqual (2, 2, matrix1, matrix2) );

copyIntMatrix ( 2, 2, matrix1, matrix3 );
copyIntMatrix ( 2, 2, matrix2, matrix3 );

IO_println ( "\nMatrix1" );
printIntMatrix( 2, 2, matrix1 );

IO_println ( "\nMatrix3" );
printIntMatrix( 2, 2, matrix2 );

IO_printf ( "isEqual (matrix1, matrix2) = %d",
            isEqual (2, 2, matrix1, matrix2) );

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

```

OBS.:

Só poderão ser comparadas matrizes com mesma quantidade de linhas e colunas.

- 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 método para somar dados em duas matrizes, posição por posição.  
Na parte principal, incluir a chamada do método para testar o novo.

```

/**
addIntMatrix    - Somar matrizes com inteiros.
@param rows     - quantidade de valores
@param columns  - quantidade de valores
@param matrix3  - grupo de valores inteiros resultante
@param matrix1  - grupo de valores inteiros (1)
@param k        - constante para multiplicar o segundo termo
@param matrix2  - grupo de valores inteiros (2)
*/
void addIntMatrix ( int rows, int columns,
                    int matrix3[ ][columns],
                    int matrix1[ ][columns], int k, int matrix2[ ][columns] )
{
    // definir dados locais
    int x = 0;
    int y = 0;

```

```

// mostrar valores na matriz
for ( x=0; x<rows; x=x+1 )
{
    for ( y = 0; y < columns; y = y + 1 )
    {
        // somar valores
        matrix3 [ x ][ y ] = matrix1 [ x ][ y ] + k * matrix2 [ x ][ y ];
    } // end for
} // end for

} // end addIntMatrix ( )

/**
 * Method_09.
 */
void method_09 ( )
{
    // definir dados
    int matrix1 [ ][2] = { { 1, 2},
                          { 3, 4} };
    int matrix2 [ ][2] = { { 1, 0},
                          { 0, 1} };
    int matrix3 [ ][2] = { { 0, 0},
                          { 0, 0} };

    // identificar
    IO_id ( "Method_09 - v0.0" );

    // testar dados
    IO_println ( "\nMatrix1" );
    printIntMatrix( 2, 2, matrix1 );

    IO_println ( "\nMatrix2" );
    printIntMatrix( 2, 2, matrix2 );

    // soamr matrizes
    addIntMatrix ( 2, 2, matrix3, matrix1, (-2), matrix2 );

    IO_println ( "\nMatrix3" );
    printIntMatrix( 2, 2, matrix3 );

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

```

OBS.:

Só poderão ser operadas matrizes com mesma quantidade de linhas e colunas.

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 um método para calcular o produto de matrizes.

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

```
/**
    mulIntMatrix      - Multiplicar matrizes com inteiros.
    @param rows3      - quantidade de linhas da matriz (3)
    @param columns3    - quantidade de colunas da matriz (3)
    @param matrix3     - grupo de valores inteiros resultante
    @param rows1       - quantidade de linhas da matriz (1)
    @param columns1    - quantidade de colunas da matriz (1)
    @param matrix1     - grupo de valores inteiros (1)
    @param rows2       - quantidade de linhas da matriz (2)
    @param columns2    - quantidade de colunas da matriz (2)
    @param matrix2     - grupo de valores inteiros (2)
*/
void mulIntMatrix ( int rows3, int columns3,
                    int matrix3[ ][columns3],
                    int rows1, int columns1,
                    int matrix1[ ][columns1],
                    int rows2, int columns2,
                    int matrix2[ ][columns2] )
{
    // definir dados locais
    int x    = 0;
    int y    = 0;
    int z    = 0;
    int soma = 0;

    if ( rows1 <= 0 || columns1 <= 0 ||
        rows2 <= 0 || columns2 <= 0 ||
        rows3 <= 0 || columns3 <= 0 ||
        columns1 != rows2 ||
        rows1    != rows3 ||
        columns2 != columns3 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
    }
    else
    {
        // percorrer valores na matriz resultante
        for ( x=0; x<rows3; x=x+1 )
        {
            for ( y = 0; y < columns3; y = y + 1 )
            {
                // somar valores
                soma = 0;
                for ( z = 0; z < columns1; z = z + 1 ) // ou (z < rows2)
                {
                    soma = soma + matrix1 [ x ][ z ] * matrix2 [ z ][ y ];
                } // end for
                // guardar a soma
                matrix3 [ x ][ y ] = soma;
            } // end for
        } // end for
    } // end if
} // end mulIntMatrix ( )
```



```

/**
    Method_10.
*/
void method_10 ( )
{
    // identificar
    IO_id ( "Method_10 - v0.0" );

    // definir dados
    int matrix1 [ ][2] = { {1, 2},
                           {3, 4} };
    int matrix2 [ ][2] = { {1, 0},
                           {0, 1} };
    int matrix3 [ ][2] = { {0, 0},
                           {0, 0} };

    // identificar
    IO_id ( "EXEMPLO0910 - Method_09 - v0.0" );

    // testar produto
    IO_println ( "\nMatrix1" );
    printIntMatrix( 2, 2, matrix1 );
    IO_println ( "\nMatrix2" );
    printIntMatrix( 2, 2, matrix2 );

    // multiplicar matrizes
    mullIntMatrix ( 2, 2, matrix3, 2, 2, matrix1, 2, 2, matrix2 );
    IO_println ( "\nMatrix3 = Matrix1 * Matrix2" );
    printIntMatrix( 2, 2, matrix3 );

    // outro teste
    IO_println ( "\nMatrix2" );
    printIntMatrix( 2, 2, matrix2 );
    IO_println ( "\nMatrix1" );
    printIntMatrix( 2, 2, matrix1 );

    // multiplicar matrizes
    mullIntMatrix ( 2, 2, matrix3, 2, 2, matrix2, 2, 2, matrix1 );
    IO_println ( "\nMatrix3 = Matrix2 * Matrix1" );
    printIntMatrix( 2, 2, matrix3 );

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

```

OBS.:

Só poderão ser operadas as matrizes com dimensões compatíveis, ou seja, cuja a quantidade de colunas da primeira, for igual à quantidade de linhas da segunda. A matriz resultante terá a mesma quantidade de linhas da primeira matriz, e a mesma quantidade de colunas da segunda matriz.

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.

Prever, realizar e registrar todos os testes efetuados.

Integrar as chamadas de todos os programas em um só.

Supor que as dimensões de uma matriz não passarão de 10,  
e serão as mesmas caso a matriz for quadrada.

01.) Incluir um método (0911) para

ler as dimensões (quantidade de linhas e de colunas) de uma matriz real do teclado,  
bem como todos os seus elementos (apenas valores positivos).

Verificar se as dimensões não são nulas ou negativas.

Para testar, ler dados e mostrá-los na tela por outro método.

```
Exemplo: double positiveMatrix [10][10];  
          readPositiveDoubleMatrix ( 3, 3, positiveMatrix );  
          printDoubleMatrix        ( 3, 3, positiveMatrix );
```

02.) Incluir um método (0912) para

gravar uma matriz real em arquivo.

A matriz e o nome do arquivo serão dados como parâmetros.

Para testar, usar a leitura da matriz do problema anterior.

Usar outro método para ler e recuperar a matriz do arquivo, antes de mostrá-la na tela.

```
Exemplo: double positiveMatrix [10][10];  
          readPositiveDoubleMatrix      ( 3, 3, positiveMatrix );  
          fprintfDoubleMatrix ( "MATRIX_01.TXT", 3, 3, positiveMatrix );
```

03.) Incluir um método (0913) para

mostrar somente os valores na diagonal principal de uma matriz real, se for quadrada.

```
Exemplo: double positiveMatrix [10][10];  
          readPositiveDoubleMatrix ( 3, 3, positiveMatrix );  
          printDiagonalDoubleMatrix ( 3, 3, positiveMatrix );
```

04.) Incluir um método (0914) para

mostrar somente os valores na diagonal secundária de uma matriz real, se for quadrada.

```
Exemplo: double positiveMatrix [10][10];  
          readPositiveDoubleMatrix ( 3, 3, positiveMatrix );  
          printSDiagonalDoubleMatrix ( 3, 3, positiveMatrix );
```

05.) Incluir um método (0915) para

mostrar somente os valores abaixo e na diagonal principal de uma matriz real, se for quadrada.

```
Exemplo: double positiveMatrix [10][10];  
          readPositiveDoubleMatrix ( 3, 3, positiveMatrix );  
          printLDTriangleDoubleMatrix ( 3, 3, positiveMatrix );
```

06.) Incluir um método (0916) para  
mostrar somente os valores acima e na diagonal principal de uma matriz real, se for quadrada.

Exemplo: `double positiveMatrix [10][10];`  
`readPositiveDoubleMatrix ( 3, 3, positiveMatrix );`  
`printLUTriangleDoubleMatrix ( 3, 3, positiveMatrix );`

07.) Incluir um método (0917) para  
mostrar somente os valores abaixo e na diagonal secundária de uma matriz real, se for quadrada.

Exemplo: `double positiveMatrix [10][10];`  
`readPositiveDoubleMatrix ( 3, 3, positiveMatrix );`  
`printSLDTriangleDoubleMatrix ( 3, 3, positiveMatrix );`

08.) Incluir um método (0918) para  
mostrar somente os valores acima e na diagonal secundária de uma matriz real, se for quadrada.

Exemplo: `double positiveMatrix [10][10];`  
`readPositiveDoubleMatrix ( 3, 3, positiveMatrix );`  
`printSLUTriangleDoubleMatrix ( 3, 3, positiveMatrix );`

09.) Incluir uma função (0919) para  
testar se não são todos zeros os valores abaixo da diagonal principal de uma matriz real quadrada.

Exemplo: `double positiveMatrix [10][10];`  
`readNegativeDoubleMatrix ( 3, 3, positiveMatrix );`  
`bool result = allZerosLTriangleDoubleMatrix ( 3, 3, positiveMatrix );`

10.) Incluir uma função (0920) para  
testar se são zeros os valores acima da diagonal principal de uma matriz real quadrada.

Exemplo: `double positiveMatrix [10][10];`  
`readPositiveDoubleMatrix ( 3, 3, positiveMatrix );`  
`bool result = allZerosUTriangleDoubleMatrix ( 3, 3, positiveMatrix );`

## Tarefas extras

E1.) Incluir definições e testes (09E1) testes para ler do teclado as quantidades de linhas e colunas de uma matriz, e montar uma matriz com a característica abaixo, a qual deverá ser gravada em arquivo, após o retorno.

Exemplos:

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

E2.) Incluir definições e testes (09E2) para ler do teclado as quantidades de linhas e colunas de uma matriz, e montar uma matriz com a característica abaixo, a qual deverá ser gravada em arquivo, após o retorno.

Exemplos:

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