PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 09

Preparação

Vídeos recomendados:
Antes de iniciar as atividades, recomenda-se assistir aos seguintes vídeos:

https://www.youtube.com/watch?v=3TP0e-bfdfw&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=26
https://www.youtube.com/watch?v=7YdzpGWTiSM&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=27
https://www.youtube.com/watch?v=sTYLxyPszWQ&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=28
https://www.youtube.com/watch?v=p2ihD9uDZs4&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=61
https://www.youtube.com/watch?v=iU9CL5d-P5U&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=62
https://www.youtube.com/watch?v=34uZMXVQd08&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=63
https://www.youtube.com/watch?v=W4vbwEJn11U&list=PL8iN9FQ7_jt4DJbeQqv--jpTy-2gTA3Cp&index=65

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

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo0901.c, para mostrar dados em matriz:

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

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

```
/**
  Method01 - Mostrar certa quantidade de valores.
 */
void method01 ( )
{
 // definir dado
   int matrix [ ][3] = { {1, 2, 3},
                         {4, 5, 6},
                         {7, 8, 9}
                       };

 // identificar
   IO_id ( "EXEMPLO0910 - Method01 - v0.0" );

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

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

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.) Copiar a versão atual do programa para outra nova – Exemplo0902.c.

05.) Editar mudanças no nome do programa e versão.
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 lines        - quantidade de linhas
  @param columns - quantidade de colunas
  @param matrix      - grupo de valores inteiros
 */
void readIntMatrix ( int lines, 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<lines; 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;
       } // fim repetir
    } // fim repetir
} // readIntMatrix ( )

/**
   Method02.
 */
void method02 ( )
{
 // definir dados
    int n = 2;                  // quantidade de valores
    int matrix [ n ][ n ];

 // identificar
    IO_id ( "EXEMPLO0910 - Method02 - v0.0" );

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

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

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

OBS.:
Só poderá ser mostrado a matriz em que existir algum conteúdo
(diferente de **NULL** = 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.

09.) Copiar a versão atual do programa para outra nova – Exemplo0903.c.

09.) Editar mudanças no nome do programa e versão.
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 lines        - quantidade de linhas
  @param columns  - quantidade de colunas
  @param matrix      - grupo de valores inteiros
 */
void fprintIntMatrix ( chars fileName, int lines, 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", lines       );
   IO_fprintf ( arquivo, "%d\n", columns );

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

 // fechar arquivo
   fclose ( arquivo );
} // fprintIntMatrix ( )

/**
  Method03.
 */
void method03 ( )
{
 // definir dados
   int lines       = 0;
   int columns = 0;

 // identificar
   IO_id ( "EXEMPLO0910 - Method03 - v0.0" );

 // ler dados
   lines       = IO_readint ( "\nlines   = "      );
   columns = IO_readint ( "\ncolumns = " );
   IO_printf ( "\n" );
```

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

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

OBS.:
Se existir dados na matriz original, eles serão sobrescritos.

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 – Exemplo0904,c.

13.) Editar mudanças no nome do programa e versão.
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;
    } // fim se

 // retornar dado lido
    return ( n );
} // 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;
    } // fim se

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

/**
   freadIntMatrix       - Ler arranjo bidimensional com valores inteiros.
   @param fileName  - nome do arquivo
   @param lines       - quantidade de valores
   @param columns  - quantidade de valores
   @param matrix      - grupo de valores inteiros
 */
void freadIntMatrix ( chars fileName, int lines, 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 ( lines      <= 0 || lines      > x  ||
        columns <= 0 || columns > y )
   {
     IO_println ( "ERRO: Valor invalido." );
   }
   else
   {
    // ler e guardar valores em arranjo
       x = 0;
       while ( ! feof ( arquivo ) && x < lines )
       {
          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;
          } // fim repetir
        // passar ao proximo
          x = x+1;
       } // fim repetir
   } // fim se
} // freadIntMatrix ( )

/**
   Method04.
 */
void method04 ( )
{
 // definir dados
   int  lines      = 0;
   int  columns = 0;

 // identificar
   IO_id ( "EXEMPLO0910 - Method04 - v0.0" );

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

   if ( lines <= 0 || columns <= 0 )
   {
     IO_println ( "\nERRO: Valor invalido." );
   }
   else
   {
    // definir armazenador
       int matrix [ lines ][ columns ];
    // ler dados
       freadIntMatrix ( "MATRIX1.TXT", lines, columns, matrix );
    // mostrar dados
       IO_printf      ( "\n" );
       printIntMatrix ( lines, columns, matrix );
   } // fim se

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

OBS.:
Só poderá ser guardada a mesma quantidade de dados lida no início do arquivo, se houver.

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 – Exemplo0905,c.

17.) Editar mudanças no nome do programa e versão.
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 lines        - quantidade de valores
   @param columns      - quantidade de valores
   @param matrix       - grupo de valores inteiros
 */
void copyIntMatrix ( int lines, int columns,
                     int matrix2[ ][columns], int matrix1[ ][columns] )
{
 // definir dados locais
    int x = 0;
    int y = 0;

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

```
/**
  Method05.
 */
void method05 ( )
{
 // definir dados
   int  lines      = 0;
   int  columns = 0;

 // identificar
   IO_id ( "EXEMPLO0910 - Method05 - v0.0" );

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

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

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

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

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

    // mostrar dados
      IO_printf     ( "\nCopia\n" );
      printIntMatrix ( lines, columns, other   );
   } // fim se

 // encerrar
   IO_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 – Exemplo0906,c..

21.) Editar mudanças no nome do programa e versão.
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 lines        - quantidade de valores
  @param columns      - quantidade de valores
  @param matrix2      - grupo de valores inteiros (transposto)
  @param matrix1      - grupo de valores inteiros
 */
void transposeIntMatrix ( int lines, int columns,
                          int matrix2[ ][lines] , int matrix1[ ][columns] )
{
 // definir dados locais
   int x      = 0;
   int y      = 0;

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

/**
  Method06.
 */
void method06 ( )
{
 // 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} };

 // identificar
   IO_id ( "EXEMPLO0910 - Method06 - 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" );
} // fim method06 ( )
```

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

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 – Exemplo0907,c..

25.) Editar mudanças no nome do programa e versão.
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 lines       - quantidade de valores
  @param columns  - quantidade de valores
  @param matrix      - grupo de valores inteiros
 */
bool isIdentity (  int lines, int columns, int matrix[ ][columns] )
{
 // definir dados locais
   bool result = true;
   int   x       = 0;
   int   y       = 0;
```

```
// testar condicao de existencia
   if ( lines <= 0 || columns <= 0 ||
        lines != columns )
   {
     IO_printf ( "\nERRO: Valor invalido.\n" );
   }
   else
   {
    // testar valores na matriz
      x = 0;
      while ( x<lines && 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 );
            } // fim se
          // passar ao proximo
            y = y + 1;
         } // fim repetir
       // passar ao proximo
         x = x + 1;
      } // fim repetir
   } // fim se

 // retornar resposta
   return ( result );
} // isIdentity ( )

/**
  Method07.
 */
void method07 ( )
{
 // 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 ( "EXEMPLO0910 - Method07 - 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" );
} // fim method07 ( )
```

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

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 – Exemplo0908,c..

29.) Editar mudanças no nome do programa e versão.
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 lines        - 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 lines, 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<lines && result )
    {
       y = 0;
       while ( y<columns && result )
       {
        // testar valor
          result = result &&
                   ( matrix1 [ x ][ y ] == matrix2 [ x ][ y ] );
        // passar ao proximo
          y = y + 1;
       } // fim repetir
     // passar ao proximo
       x = x + 1;
    } // fim repetir
 // retornar resposta
    return ( result );
} // isEqual ( )

/**
   Method09.
 */
void method09 ( )
{
 // 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 ( "EXEMPLO0910 - Method08 - 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" );
} // fim method09 ( )
```

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

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 – Exemplo0909,c..

33.) Editar mudanças no nome do programa e versão.
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 lines       - 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 lines, 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<lines; x=x+1 )
   {
      for ( y = 0; y < columns; y = y + 1 )
      {
       // somar valores
          matrix3 [ x ][ y ] = matrix1 [ x ][ y ] + k * matrix2 [ x ][ y ];
      } // fim repetir
   } // fim repetir

} // addIntMatrix ( )
```

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

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

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 – Exemplo0910,c..

37.) Editar mudanças no nome do programa e versão.
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 lines3      - quantidade de linhas     da matriz (3)
  @param columns3 - quantidade de colunas da matriz (3)
  @param matrix3     - grupo de valores inteiros resultante
  @param lines1      - quantidade de linhas     da matriz (1)
  @param columns1 - quantidade de colunas da matriz (1)
  @param matrix1     - grupo de valores inteiros (1)
  @param lines2      - quantidade de linhas     da matriz (2)
  @param columns2 - quantidade de colunas da matriz (2)
  @param matrix2     - grupo de valores inteiros (2)
 */
void mulIntMatrix ( int lines3, int columns3,
                    int matrix3[ ][columns3],
                    int lines1, int columns1,
                    int matrix1[ ][columns1],
                    int lines2, int columns2,
                    int matrix2[ ][columns2] )
{
 // definir dados locais
   int x       = 0;
   int y       = 0;
   int z       = 0;
   int soma = 0;

   if ( lines1 <= 0 || columns1 == 0 ||
        lines2 <= 0 || columns2 == 0 ||
        lines3 <= 0 || columns3 == 0 ||
        columns1  != lines2 ||
        lines1       != lines3 ||
        columns2  != columns3 )
   {
     IO_printf ( "\nERRO: Valor invalido.\n" );
   }
   else
   {
     // percorrer valores na matriz resultante
        for ( x=0; x<lines3; x=x+1 )
        {
           for ( y = 0; y < columns3; y = y + 1 )
           {
            // somar valores
              soma = 0;
              for ( z = 0; z < columns1; z = z + 1 )
              {
                 soma = soma + matrix1 [ x ][ z ] *  matrix2 [ z ][ y ];
              } // fim repetir
            // guardar a soma
              matrix3 [ x ][ y ] = soma;
           } // fim repetir
        } // fim repetir
   } // fim se

} // mulIntMatrix ( )
```

```
/**
  Method10.
 */
void method10 ( )
{
 // identificar
   IO_id ( "EXEMPLO0910 - Method10 - 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 - Method09 - v0.0" );

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

 // multiplicar matrizes
   mulIntMatrix  ( 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
   mulIntMatrix  ( 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" );
} // fim method10 ( )
```

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.

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

01.) Incluir em um programa (Exemplo0911) um método 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.

02.) Incluir em um programa (Exemplo0912) um método 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.

03.) Incluir em um programa (Exemplo0913) um método para
mostrar somente os valores pares na diagonal principal de uma matriz real,
se for quadrada.

04.) Incluir em um programa (Exemplo0914) um método para
mostrar somente os valores ímpares na diagonal secundária de uma matriz real,
se for quadrada.

05.) Incluir em um programa (Exemplo0915) um método para
mostrar somente os valores múltiplos de 3, abaixo da diagonal principal de uma matriz real,
se for quadrada.

06.) Incluir em um programa (Exemplo0916) um método para
mostrar somente os valores múltiplos de 5, acima da diagonal principal de uma matriz real,
se for quadrada.

07.) Incluir em um programa (Exemplo0917) um método para
mostrar somente os valores múltiplos de 3 e ímpares, abaixo da diagonal secundária
de uma matriz real, se for quadrada.

08.) Incluir em um programa (Exemplo0918) um método para
mostrar somente os valores múltiplos de 5 e pares, acima da diagonal secundária
de uma matriz real, se for quadrada.

09.) Incluir em um programa (Exemplo0919) uma função para
testar se não são todos zeros os valores abaixo da diagonal principal de uma matriz real quadrada.

10.) Incluir em um programa (Exemplo0920) uma função para
testar se são zeros os valores acima da diagonal principal de uma matriz real quadrada.

Tarefas extras

E1.) Incluir em um programa (Exemplo09E1) definições e 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.

					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

E2.) Incluir em um programa (Exemplo09E2) definições e 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.

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