

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

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo1000.c, para mostrar dados em arranjo:

```
/*
   Exemplo1000 - v0.0. - __ / __ / ____
   Author: _____

*/
// dependencias
#include "io.h"           // para definicoes proprias

// ----- definicoes globais

/**
   Definicao de tipo arranjo com inteiros
   baseado em estrutura
*/
typedef
struct s_int_Array
{
    int length;
    ints data ;
    int ix ;
}
int_Array;

/**
   Definicao de referencia para arranjo com inteiros
   baseado em estrutura
*/
typedef int_Array* ref_int_Array;
```

```

/**
 new_int_Array - Reservar espaco para arranjo com inteiros
 @return referencia para arranjo com inteiros
 @param n - quantidade de dados
 */
ref_int_Array new_int_Array ( int n )
{
 // reserva de espaco
 ref_int_Array tmpArray = (ref_int_Array) malloc (sizeof(int_Array));

 // estabelecer valores padroes
 if ( tmpArray == NULL )
 {
 IO_printf ( "\nERRO: Falta espaco.\n" );
 }
 else
 {
 tmpArray->length = 0;
 tmpArray->data = NULL;
 tmpArray->ix = -1;
 if ( n>0 )
 {
 // guardar a quantidade de dados
 tmpArray->length = n;
 // reservar espaco para os dados
 tmpArray->data = (ints) malloc (n * sizeof(int));
 // definir indicador do primeiro elemento
 tmpArray->ix = 0;
 } // end if
 } // end if

 // retornar referencia para espaco reservado
 return ( tmpArray );
} // end new_int_Array ( )

/**
 free_int_Array - Dispensar espaco para arranjo com inteiros
 @param tmpArray - referencia para grupo de valores inteiros
 */
void free_int_Array ( ref_int_Array tmpArray )
{
 // testar se ha' dados, antes de reciclar o espaco
 if ( tmpArray != NULL )
 {
 if ( tmpArray->data != NULL )
 {
 free ( tmpArray->data );
 } // end if
 free ( tmpArray );
 } // fim se
} // end free_int_Array ( )

```

```

/**
 printIntArray - Mostrar arranjo com valores inteiros.
 @param array - grupo de valores inteiros
 */
void printIntArray ( int_Array array )
{
 // mostrar valores no arranjo
 if ( array.data )
 {
 for ( array.ix=0; array.ix<array.length; array.ix=array.ix+1 )
 {
 // mostrar valor
 printf ( "%2d: %d\n", array.ix, array.data [ array.ix ] );
 } // end for
 } // end if
 } // end printIntArray ( )

/**
 Method_01 - Mostrar certa quantidade de valores.
 */
void method_01 ( )
{
 // definir dado
 int_Array array;

 // montar arranjo em estrutura
 array.length = 5;
 array.data = (ints) malloc (array.length * sizeof(int));

 // testar a existência de dados
 if ( array.data )
 {
 array.data [ 0 ] = 1;
 array.data [ 1 ] = 2;
 array.data [ 2 ] = 3;
 array.data [ 3 ] = 4;
 array.data [ 4 ] = 5;
 } // fim se

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

 // executar o metodo auxiliar
 printIntArray ( array );

 // reciclar o espaco
 if ( array.data )
 {
 free ( array.data );
 } // fim se

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

```

OBS.:

As definições iniciais servirão para especificar um tipo de armazenador composto por vários tipos de dados, os quais serão usados sempre em conjunto.

Um desses dados será a quantidade de valores armazenados; outro, uma referência para onde serão guardados; e um terceiro para permitir o acesso a cada um desses valores.

Dois métodos acompanharão o uso desse novo tipo de armazenador: o que servirá para proceder a reserva de espaço e estabelecer os valores iniciais (construir a identidade), e o que servirá para liberar e reciclar o espaço reservado, quando esse não tiver mais utilidade para o programa.

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 arranjo.

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

```
/**
  IO_readintArray - Ler arranjo com valores inteiros.
  @return arranjo com valores lidos
 */
int_Array IO_readintArray ( )
{
  // definir dados locais
  chars text = IO_new_chars ( STR_SIZE );
  static int_Array array;

  // ler a quantidade de dados
  do
  {
    array.length = IO_readint ( "\nlength = " );
  }
  while ( array.length <= 0 );

  // reservar espaço para armazenar
  array.data = IO_new_ints ( array.length );
```

```

// testar se ha' espaco
if ( array.data == NULL )
{
    array.length = 0; // nao ha' espaco
}
else
{
    // ler e guardar valores em arranjo
    for ( array.ix=0; array.ix<array.length; array.ix=array.ix+1 )
    {
        // ler valor
        strcpy ( text, STR_EMPTY );
        array.data [ array.ix ]
            = IO_readint ( IO_concat (
                IO_concat ( text, IO_toString_d ( array.ix ) ), " : " ) );
    } // end for
} // end if

// retornar arranjo
return ( array );
} // end IO_readintArray ( )

/**
    Method_02.
*/
void method_02 ( )
{
    // definir dados
    int_Array array;

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

    // ler dados
    array = IO_readintArray ( );

    // testar a existência de dados
    if ( array.data )
    {
        // mostrar dados
        IO_printf ( "\n" );
        printIntArray ( array );
        // reciclar o espaco
        free ( array.data );
    } // end if

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

```

OBS.:

Reparar que as definições para uso são mais simples que outras anteriormente apresentadas. Uma definição estática (**static**) preservará a existência do dado fora do contexto de declaração. Só poderá ser mostrado o arranjo 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 no arranjo.
Na parte principal, incluir a chamada do método para testar o novo.

```
/**
    fprintfIntArray    - Gravar arranjo com valores inteiros.
    @param fileName - nome do arquivo
    @param array     - grupo de valores inteiros
*/
void fprintfIntArray ( chars fileName, int_Array array )
{
    // definir dados locais
    FILE* arquivo = fopen ( fileName, "wt" );

    // gravar quantidade de dados
    fprintf ( arquivo, "%d\n", array.length );

    // gravar valores no arquivo, se existirem
    if ( array.data )
    {
        for ( array.ix=0; array.ix<array.length; array.ix=array.ix+1 )
        {
            // gravar valor
            fprintf ( arquivo, "%d\n", array.data [ array.ix ] );
        } // end for
    } // end if

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

/**
    Method_03.
*/
void method_03 ( )
{
    // definir dados
    int_Array array;

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

```

// ler dados
array = IO_readIntArray ( );

// testar a existência de dados
if ( array.data )
{
    // mostrar e gravar dados
    IO_printf ( "\n" );
    printIntArray ( array );
    fprintfIntArray ( "ARRAY1.TXT", array );
    // reciclar o espaço
    free ( array.data );
} // end if

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

```

OBS.:

Se existir dados no arranjo 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 outro método para ler arquivo e guardar dados em arranjo.
Na parte principal, incluir a chamada do método para testar o novo.

```

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

```

```

// testar a existencia
if ( arquivo )
{
    // ler a quantidade de dados
    fscanf ( arquivo, "%d", &n );

    if ( n <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
        n = 0;
    } // end if

    fclose ( arquivo );
} // end if

// retornar dados lidos
return ( n );
} // end freadArraySize ( )

/**
fIO_readintArray - Ler arranjo com valores inteiros.
@return arranjo com os valores lidos
@param fileName - nome do arquivo
@param array - grupo de valores inteiros
*/
int_Array fIO_readintArray ( chars fileName )
{
    // definir dados locais
    int x = 0;
    int y = 0;
    FILE* arquivo = fopen ( fileName, "rt" );
    static int_Array array;

    // testar a existencia
    if ( arquivo )
    {
        // ler a quantidade de dados
        fscanf ( arquivo, "%d", &array.length );

        // testar se ha' dados
        if ( array.length <= 0 )
        {
            IO_printf ( "\nERRO: Valor invalido.\n" );
            array.length = 0; // nao ha' dados
        }
        else
        {
            // reservar espaco
            array.data = IO_new_ints ( array.length );

```



```

// testar a existência
if ( array.data )
{
    // ler e guardar valores em arranjo
    array.ix = 0;
    while ( ! feof ( arquivo ) &&
            array.ix < array.length )
    {
        // ler valor
        fscanf ( arquivo, "%d", &(array.data [ array.ix ] ) );
        // passar ao proximo
        array.ix = array.ix + 1;
    } // end while
} // end if
} // end if

// retornar dados lidos
return ( array );
} // end fIO_readintArray ( )

/**
    Method_04.
*/
void method_04 ( )
{
    // definir dados
    int_Array array; // arranjo sem tamanho definido

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

    // ler dados
    array = fIO_readintArray ( "ARRAY1.TXT" );

    // testar a existência de dados
    if ( array.data )
    {
        // mostrar dados
        IO_printf ( "\n" );
        printIntArray ( array );
        // reciclar o espaco
        free ( array.data );
    } // end if

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

```

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 um método para copiar dados de um arranjo para outro.
Na parte principal, incluir a chamada do método para testar o novo.

```
/**
 copyIntArray    - Copiar arranjo com valores inteiros.
 @return referencia para copia do arranjo
 @param fileName - nome do arquivo
 @param array    - grupo de valores inteiros
 */
ref_int_Array copyIntArray ( int_Array array )
{
    // definir dados locais
    int    x = 0;
    int    y = 0;
    ref_int_Array copy;

    if ( array.length <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
        array.length = 0;
    }
    else
    {
        // reservar area
        copy = new_int_Array ( array.length );
        // testar se ha' descritor
        if ( copy )
        {
            copy->length = array.length;
            copy->data = IO_new_ints ( copy->length );

            // testar se ha' espaco e dados
            if ( copy->data == NULL || array.data == NULL )
            {
                printf ( "\nERRO: Falta espaco ou dados." );
            }
            else
            {
                // ler e copiar valores
                for ( array.ix=0; array.ix<array.length; array.ix=array.ix+1 )
                {
                    // copiar valor
                    copy->data [ array.ix ] = array.data [ array.ix ];
                } // end for
            } // end if
        } // end if

        // retornar dados lidos
        return ( copy );
    } // end copyIntArray ( )
}
```

```

/**
  Method_05.
 */
void method_05 ( )
{
  // definir dados
  int_Array array; // arranjo sem tamanho definido
  ref_int_Array other; // referencia para arranjo sem tamanho definido

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

  // ler dados
  array = fIO_readIntArray ( "ARRAY1.TXT" );

  // copiar dados
  other = copyIntArray ( array );

  // testar a existência de dados
  if ( array.data )
  {
    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntArray ( array );

    // mostrar dados
    IO_printf ( "\nCopial\n" );
    printIntArray ( *other ); // dereferenciar a copia

    // reciclar os espacos
    free ( array.data );
    free ( other->data );
    free ( 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 definição no início, próxima à feita anteriormente para os arranjos. Acrescentar um método para mostrar dados em arranjos bidimensionais (matrizes). Na parte principal, incluir a chamada do método para testar o novo.

```
/**
 * Definicao de tipo arranjo bidimensional com inteiros baseado em estrutura
 */
typedef
struct s_int_Matrix
{
    int rows ;
    int columns;
    ints* data ;
    int ix, iy ;
}
int_Matrix;

/**
 * Definicao de referencia para arranjo bidimensional com inteiros baseado em estrutura
 */
typedef int_Matrix* ref_int_Matrix;

/**
 * new_int_Matrix - Reservar espaco para arranjo bidimensional com inteiros
 * @return referencia para arranjo com inteiros
 * @param rows - quantidade de dados
 * @param columns - quantidade de dados
 */
ref_int_Matrix new_int_Matrix ( int rows, int columns )
{
    // reserva de espaco
    ref_int_Matrix tmpMatrix = (ref_int_Matrix) malloc (sizeof(int_Matrix));

    // estabelecer valores padroes
    if ( tmpMatrix != NULL )
    {
        tmpMatrix->rows = 0;
        tmpMatrix->columns = 0;
        tmpMatrix->data = NULL;
        // reservar espaco
        if ( rows>0 && columns>0 )
        {
            tmpMatrix->rows = rows;
            tmpMatrix->columns = columns;
            tmpMatrix->data = malloc (rows * sizeof(int));
            if ( tmpMatrix->data )
            {
                for ( tmpMatrix->ix=0;
                     tmpMatrix->ix<tmpMatrix->rows;
                     tmpMatrix->ix=tmpMatrix->ix+1 )
                {
                    tmpMatrix->data [ tmpMatrix->ix ] = (ints) malloc (columns * sizeof(int));
                } // end for
            } // end if
        } // end if
        tmpMatrix->ix = 0;
        tmpMatrix->iy = 0;
    } // end if
    return ( tmpMatrix );
} // end new_int_Matrix ( )
```

```

/**
    free_int_Matrix    - Dispensar espaco para arranjo com inteiros
    @param tmpMatrix - referencia para grupo de valores inteiros
*/
void free_int_Matrix ( ref_int_Matrix matrix )
{
    // testar se ha' dados
    if ( matrix != NULL )
    {
        if ( matrix->data != NULL )
        {
            for ( matrix->ix=0;
                  matrix->ix<matrix->rows;
                  matrix->ix=matrix->ix+1 )
            {
                free ( matrix->data [ matrix->ix ] );
            } // end for
            free ( matrix->data );
        } // end if
        free ( matrix );
    } // end if
} // end free_int_Matrix ( )

/**
    printIntMatrix    - Mostrar matrix com valores inteiros.
    @param array    - grupo de valores inteiros
*/
void printIntMatrix ( ref_int_Matrix matrix )
{
    // testar a existencia
    if ( matrix != NULL && matrix->data != NULL )
    {
        // mostrar valores na matriz
        for ( matrix->ix=0; matrix->ix<matrix->rows; matrix->ix=matrix->ix+1 )
        {
            for ( matrix->iy=0; matrix->iy<matrix->columns; matrix->iy=matrix->iy+1 )
            {
                // mostrar valor
                printf ( "%3d\t", matrix->data [ matrix->ix ][ matrix->iy ] );
            } // end for
            printf ( "\n" );
        } // end for
    } // end if
} // end printIntArray ( )

```

```

/**
  Method_06.
 */
void method_06 ( )
{
  // definir dado
  ref_int_Matrix matrix = new_int_Matrix ( 3, 3 );

  if ( matrix != NULL && matrix->data != NULL )
  {
    matrix->data [0][0] = 1;  matrix->data [0][1] = 2;  matrix->data [0][2] = 3;
    matrix->data [1][0] = 3;  matrix->data [1][1] = 4;  matrix->data [1][2] = 5;
    matrix->data [2][0] = 6;  matrix->data [2][1] = 7;  matrix->data [2][2] = 8;
  } // fim se

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

  // executar o metodo auxiliar
  printIntMatrix ( matrix );

  // reciclar espaco
  free_int_Matrix ( matrix );

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

```

OBS.:

As definições iniciais servirão para especificar um tipo de armazenador composto por vários tipos de dados, os quais serão usados sempre em conjunto, tal como nos arranjos unidimensionais.

Dentre esses dados estarão a quantidade de linhas e de colunas; uma referência para onde serão guardados; e facilitadores para o acesso.

Dois métodos acompanharão o uso desse novo tipo de armazenador: o que servirá para proceder a reserva de espaço e estabelecer os valores iniciais (construir a identidade), e o que servirá para liberar e reciclar o espaço reservado, quando esse não tiver mais utilidade para o programa.

Destaca-se a necessidade de se lidar individualmente com cada linha de dados.

Diferente do exemplo com arranjo unidimensional, destaca-se aqui também o uso da referência, a necessidade da reserva de espaço e a liberação de seu uso para a reciclagem.

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 ler e armazenar dados em arranjo bidimensional (matriz).
Na parte principal, incluir a chamada do método para testar a função.

```
/**
    IO_readintMatrix - Ler arranjo bidimensional com valores inteiros.
    @return referencia para o grupo de valores inteiros
*/
ref_int_Matrix IO_readintMatrix ( )
{
    // definir dados locais
    int rows = 0;
    int columns = 0;
    chars text = IO_new_chars ( STR_SIZE );

    // ler a quantidade de dados
    do
    { rows = IO_readint ( "\nrows = " ); }
    while ( rows <= 0 );
    do
    { columns = IO_readint ( "\ncolumns = " ); }
    while ( columns <= 0 );

    // reservar espaco para armazenar valores
    ref_int_Matrix matrix = new_int_Matrix ( rows, columns );

    // testar se ha' espaco
    if ( matrix != NULL )
    {
        if ( matrix->data == NULL )
        {
            // nao ha' espaco
            matrix->rows = 0;
            matrix->columns = 0;
            matrix->ix = 0;
            matrix->iy = 0;
        }
        else
        {
            // ler e guardar valores na matriz
            for ( matrix->ix=0; matrix->ix<matrix->rows; matrix->ix=matrix->ix+1 )
            {
                for ( matrix->iy=0; matrix->iy<matrix->columns; matrix->iy=matrix->iy+1 )
                {
                    // ler e guardar valor
                    strcpy ( text, STR_EMPTY );
                    matrix->data [ matrix->ix ][ matrix->iy ]
                    = IO_readint ( IO_concat (
                                IO_concat ( IO_concat ( text, IO_toString_d ( matrix->ix ) ), ", " ),
                                IO_concat ( IO_concat ( text, IO_toString_d ( matrix->iy ) ), " : " ) ) );
                } // end for
                printf ( "\n" );
            } // end for
        } // end if
    } // end if

    // retornar dados lidos
    return ( matrix );
} // end IO_readintMatrix ( )
```

```

/**
  Method_07.
 */
void method_07 ( )
{
  // definir dados
  ref_int_Matrix matrix = NULL;

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

  // ler dados
  matrix = IO_readintMatrix ( );

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

  // reciclar espaco
  free_int_Matrix ( matrix );

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

```

OBS.:

Diferente do exemplo com arranjo unidimensional, destaca-se aqui o uso da referência.

- 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 gravar dados em matriz, posição por posição.
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 matrix - grupo de valores inteiros
 */
void fprintIntMatrix ( chars fileName, ref_int_Matrix matrix )
{
  // definir dados locais
  FILE* arquivo = fopen ( fileName, "wt" );

```



```

// testar se ha' dados
if ( matrix == NULL )
{
    printf ( "\nERRO: Nao ha' dados." );
}
else
{
    // gravar quantidade de dados
    fprintf ( arquivo, "%d\n", matrix->rows );
    fprintf ( arquivo, "%d\n", matrix->columns );

    if ( matrix->data != NULL )
    {
        // gravar valores no arquivo
        for ( matrix->ix=0; matrix->ix<matrix->rows; matrix->ix=matrix->ix+1 )
        {
            for ( matrix->iy=0; matrix->iy<matrix->columns; matrix->iy=matrix->iy+1 )
            {
                // gravar valor
                fprintf ( arquivo, "%d\n", matrix->data [ matrix->ix ][ matrix->iy ] );
            } // end for
        } // end for
    } // end if
    // fechar arquivo
    fclose ( arquivo );
} // end if
} // end fprintfIntMatrix ( )

/**
    Method_08.
*/
void method_08 ( )
{
    // definir dados
    ref_int_Matrix matrix = NULL;

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

    // ler dados
    matrix = IO_readintMatrix ( );

    // gravar dados
    fprintfIntMatrix( "MATRIX1.TXT", matrix );

    // reciclar espaco
    free_int_Matrix ( matrix );

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

```

OBS.:

Só poderão ser operados arranjos com mesma quantidade de dados.

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 uma função para ler dados de arquivo para armazenar em matriz.
Na parte principal, incluir a chamada do método para testar a função.

```
/**
freadintMatrix    - Ler arranjo bidimensional com valores inteiros.
@return referencia para o grupo de valores inteiros
@param fileName - nome do arquivo
*/
ref_int_Matrix freadintMatrix ( chars fileName )
{
    // definir dados locais
    ref_int_Matrix matrix = NULL;
    int    rows    = 0;
    int    columns = 0;
    FILE* arquivo  = fopen ( fileName, "rt" );
    // ler a quantidade de dados
    fscanf ( arquivo, "%d", &rows    );
    fscanf ( arquivo, "%d", &columns );
    if ( rows <= 0 || columns <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
    }
    else
    {
        // reservar espaco para armazenar
        matrix = new_int_Matrix ( rows, columns );
        // testar se ha' espaco
        if ( matrix != NULL && matrix->data == NULL )
        {
            // nao ha' espaco
            matrix->rows    = 0;
            matrix->columns = 0;
            matrix->ix      = 0;
            matrix->iy      = 0;
        }
    }
}
```

```

else
{
// testar a existência
if ( matrix != NULL )
{
// ler e guardar valores na matriz
matrix->ix = 0;
while ( ! feof ( arquivo ) && matrix->ix < matrix->rows )
{
matrix->iy = 0;
while ( ! feof ( arquivo ) && matrix->iy < matrix->columns )
{
// guardar valor
fscanf ( arquivo, "%d", &(matrix->data [ matrix->ix ][ matrix->iy ] ) );
// passar ao proximo
matrix->iy = matrix->iy+1;
} // end while
// passar ao proximo
matrix->ix = matrix->ix+1;
} // end while
matrix->ix = 0;
matrix->iy = 0;
} // end if
} // end if
} // end if
// retornar matriz lida
return ( matrix );
} // end freadintMatrix ( )

/**
Method_09.
*/
void method_09 ( )
{
// identificar
IO_id ( "Method_09 - v0.0" );

// ler dados
ref_int_Matrix matrix = freadintMatrix ( "MATRIX1.TXT" );

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

// reciclar espaco
free_int_Matrix ( matrix );

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

```

OBS.:

A leitura de dados foi utilizada na definição da referência para o armazenamento.

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 uma função para copiar dados em uma estrutura semelhante à da matriz.
Na parte principal, incluir a chamada do método para testar a função.

```
/**
    copyIntMatrix - Copiar matriz com valores inteiros.
    @return referencia para o grupo de valores inteiros
 */
ref_int_Matrix copyIntMatrix ( ref_int_Matrix matrix )
{
    // definir dados locais
    ref_int_Matrix copy = NULL;

    if ( matrix == NULL || matrix->data == NULL )
    {
        IO_printf ( "\nERRO: Faltam dados.\n" );
    }
    else
    {
        {
            if ( matrix->rows <= 0 || matrix->columns <= 0 )
            {
                IO_printf ( "\nERRO: Valor invalido.\n" );
            }
            else
            {
                // reservar espaco
                copy = new_int_Matrix ( matrix->rows, matrix->columns );

                // testar se ha' espaco e dados
                if ( copy == NULL || copy->data == NULL )
                {
                    printf ( "\nERRO: Falta espaco." );
                }
                else
                {
                    // copiar valores
                    for ( copy->ix = 0; copy->ix < copy->rows; copy->ix = copy->ix + 1 )
                    {
                        for ( copy->iy = 0; copy->iy < copy->columns; copy->iy = copy->iy + 1 )
                        {
                            // copiar valor
                            copy->data [ copy->ix ][ copy->iy ]
                                = matrix->data [ copy->ix ][ copy->iy ];
                        } // end for
                    } // end for
                } // end if
            } // end if
        } // end if
    } // end if

    // retornar copia
    return ( copy );
} // end copyIntMatrix ( )
```

```

/**
  Method_10.
 */
void method_10 ( )
{
  // definir dados
  ref_int_Matrix matrix = NULL;
  ref_int_Matrix other = NULL;

  // identificar
  IO_id ( "Method_10 - v0.0" );

  // ler dados
  matrix = freadIntMatrix ( "MATRIX1.TXT" );

  // copiar dados
  other = copyIntMatrix ( matrix );

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

  // mostrar dados
  IO_printf ( "\nCopied\n" );
  printIntMatrix ( other );

  // reciclar espaço
  free_int_Matrix ( matrix );
  free_int_Matrix ( other );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method10 ( )

```

- 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.

Restrições:

- As repetições deverão, de preferência, usar **for**, exceto as que tiverem que não forem percorrer todos os dados e, nesses casos deverão ser usadas expressões lógicas para interrompê-las.
- Testes deverão usar expressões lógicas e usar funções próprias em lugar das disponíveis em bibliotecas nativas.
- Os tratamentos de erros e condições excepcionais deverão usar **else**, e prover mensagens adequadas.

01.) Incluir um método (1011) para

gerar um valor inteiro aleatoriamente dentro de um intervalo, cujos limites de início e de fim serão recebidos como parâmetros.

Para para testar, ler os limites do intervalo do teclado;

ler a quantidade de elementos (N) a serem gerados;

gerar essa quantidade (N) de valores aleatórios

dentro do intervalo e armazená-los em arranjo;

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()** e limitar a valores menores ou iguais a 10^6 .

Exemplo: valor = RandomIntGenerate (inferior, superior);

02.) Incluir uma função (1012) para

procurar certo valor inteiro em um arranjo.

Para testar, receber um nome de arquivo como parâmetro e

aplicar a função sobre o arranjo com os valores lidos.

DICA: Usar o modelo de arranjo em estrutura proposto nos exemplos.

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");

resposta = arraySearch (valor, arranjo);

03.) Incluir uma função (1013) para

operar a comparação de dois arranjos.

Para testar, receber dados de arquivos e

aplicar a função sobre os arranjos com os valores lidos.

DICA: Verificar se, e somente se, os tamanhos forem iguais.

Usar o modelo de arranjo em estrutura proposto nos exemplos.

Exemplo: arranjo1 = readArrayFromFile ("DADOS1.TXT");

arranjo2 = readArrayFromFile ("DADOS2.TXT");

resposta = arrayCompare (arranjo1, arranjo2);

- 04.) Incluir uma função (1014) para
operar a soma de dois arranjos, com os elementos do segundo multiplicados por uma constante.
Para testar, receber dados de arquivos e
aplicar a função sobre os arranjos com os valores lidos;
DICA: Verificar se os tamanhos são compatíveis.
Usar o modelo de arranjo proposto nos exemplos.

```
Exemplo: arranjo1 = readArrayFromFile ( "DADOS1.TXT" );  
         arranjo2 = readArrayFromFile ( "DADOS2.TXT" );  
         soma     = arrayAdd           ( arranjo1, 1, arranjo2 );
```

- 05.) Incluir uma função (1015) para
dizer se um arranjo está em ordem decrescente.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos.
DICA: Usar o modelo de arranjo em estrutura proposto nos exemplos.
Não usar *break*!

```
Exemplo: arranjo1 = readArrayFromFile ( "DADOS1.TXT" );  
         resposta = isArrayDecrescent ( arranjo );
```

- 06.) Incluir uma função (1016) para
obter a transposta de uma matriz.
Para testar, receber dados de arquivos e
aplicar a função sobre as matrizes com os valores lidos.
DICA: Verificar se os tamanhos são compatíveis.
Usar o modelo de matriz em estrutura proposto nos exemplos.

```
Exemplo: matriz1 = readMatrixFromFile ( "DADOS1.TXT" );  
         matriz2 = matrixTranspose   ( matriz1 );
```

- 07.) Incluir uma função (1017) para
testar se uma matriz só contém valores iguais a zero.
Para testar, receber dados de arquivos e
aplicar a função sobre as matrizes com os valores lidos.
DICA: Verificar se os tamanhos são compatíveis.
Usar o modelo de matriz em estrutura proposto nos exemplos.
Não usar *break*!

```
Exemplo: matriz1 = readMatrixFromFile ( "DADOS1.TXT" );  
         resposta = matrixZero         ( matriz1 );
```

08.) Incluir uma função (1018) para

testar a igualdade de duas matrizes.

Para testar, receber dados de arquivos e

aplicar a função sobre as matrizes com os valores lidos.

DICA: Verificar se os tamanhos são compatíveis.

Usar o modelo de matriz em estrutura proposto nos exemplos.

Não usar *break*!

Exemplo: `matriz1 = readMatrixFromFile ("DADOS1.TXT");`

`matriz2 = readMatrixFromFile ("DADOS2.TXT");`

`resposta = matrixCompare (matriz1, matriz2);`

09.) Incluir uma função (1019) para

operar a soma de duas matrizes, com os elementos da segunda multiplicados por uma constante.

Para testar, receber dados de arquivos e

aplicar a função sobre as matrizes com os valores lidos.

DICA: Verificar se os tamanhos são compatíveis.

Usar o modelo de matriz em estrutura proposto nos exemplos.

Exemplo: `matriz1 = readMatrixFromFile ("DADOS1.TXT");`

`matriz2 = readMatrixFromFile ("DADOS2.TXT");`

`soma = matrixAdd (matriz1, -1, matriz2);`

10.) Incluir uma função (1020) para

obter o produto de duas matrizes.

Para testar, receber dados de arquivos e

aplicar a função sobre as matrizes com os valores lidos.

DICA: Verificar se os tamanhos são compatíveis.

Usar o modelo de matriz em estrutura proposto nos exemplos.

Exemplo: `matriz1 = readMatrixFromFile ("DADOS1.TXT");`

`matriz2 = readMatrixFromFile ("DADOS2.TXT");`

`soma = matrixProduct (matriz1, matriz2);`

Tarefas extras

- E1.) Incluir uma função (10E1) para
colocar um arranjo em ordem decrescente, pelo método de trocas de posição.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos.
DICA: Usar o modelo de arranjo em estrutura proposto nos exemplos.

```
Exemplo: arranjo1 = readArrayFromFile ( "DADOS1.TXT" );  
         ordenado = sortArrayDown    ( arranjo );
```

- E2.) Incluir uma função (10E2) para
testar se o produto de duas matrizes é igual à matriz identidade.
Para testar, receber dados de arquivos e
aplicar a função sobre as matrizes com os valores lidos;
DICA: Verificar se os tamanhos são compatíveis.
Usar o modelo de matriz em estrutura proposto nos exemplos.
Não usar *break*!

```
Exemplo: matriz1 = readMatrixFromFile ( "DADOS1.TXT" );  
         matriz2 = readMatrixFromFile ( "DADOS2.TXT" );  
         resposta = identityMatrix      ( matrixProduct (matriz1, matriz2) );
```