

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

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo1001.c, para mostrar dados em arranjo:

```
/*
  Exemplo1010 - 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;
 } // fim se
 } // fim se

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

/**
 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
 if ( tmpArray != NULL )
 {
 free ( tmpArray->data );
 free ( tmpArray );
 } // fim se
} // fim free_int_Array ( )

// ----- metodos

/**
 Method00 - nao faz nada.
 */
void method00 ( )
{
 // nao faz nada
} // fim method00 ( )

```

```

/**
 printIntArray - Mostrar arranjo com valores inteiros.
 @param array - grupo de valores inteiros
 */
void printIntArray ( int_Array array )
{
 // mostrar valores no arranjo
 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 ] );
 } // fim repetir
} // printIntArray ( )

/**
 Method01 - Mostrar certa quantidade de valores.
 */
void method01 ( )
{
 // definir dado
 int_Array array;

 // montar arranjo em estrutura
 array.length = 5;
 array.data = (ints) malloc (array.length * sizeof(int));
 array.data [ 0 ] = 1;
 array.data [ 1 ] = 2;
 array.data [ 2 ] = 3;
 array.data [ 3 ] = 4;
 array.data [ 4 ] = 5;

 // identificar
 IO_id ( "EXEMPLO1010 - Method01 - v0.0" );

 // executar o metodo auxiliar
 printIntArray ( array );

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

```

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.) Copiar a versão atual do programa para outra nova – Exemplo1002.c.

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

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 espaco 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 ) ), " : " ) );
        } // fim repetir
    } // fim se

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

```

/**
  Method02.
 */
void method02 ( )
{
  // definir dados
  int_Array array;

  // identificar
  IO_id ( "EXEMPLO1010 - Method02 - v0.0" );

  // ler dados
  array = IO_readIntArray ( );

  // mostrar dados
  IO_printf ( "\n" );
  printIntArray ( array );

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

```

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).

06.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

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

08.) Copiar a versão atual do programa para outra nova – Exemplo1003.c.

- 09.) Editar mudanças no nome do programa e versão.  
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
    for ( array.ix=0; array.ix<array.length; array.ix=array.ix+1 )
    {
        // gravar valor
        fprintf ( arquivo, "%d\n", array.data [ array.ix ] );
    } // fim repetir

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

/**
    Method03.
*/
void method03 ( )
{
    // definir dados
    int_Array array;

    // identificar
    IO_id ( "EXEMPLO0803 - Method03 - v0.0" );

    // ler dados
    array = IO_readIntArray ( );

    // mostrar dados
    IO_printf ( "\n" );
    fprintfIntArray ( "ARRAY1.TXT", array );

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

OBS.:  
Se existir dados no arranjo 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 – Exemplo1004,c.

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

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" );

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

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

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

// 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 );

// 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;
} // fim repetir
} // fim se

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

```



```

/**
  Method04.
 */
void method04 ( )
{
  // definir dados
  int_Array array; // arranjo sem tamanho definido

  // identificar
  IO_id ( "EXEMPLO1010 - Method04 - v0.0" );

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

  // mostrar dados
  IO_printf ( "\n" );
  printIntArray ( array );

  // 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 – Exemplo1005,c.

- 17.) Editar mudanças no nome do programa e versão.  
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 copia do arranjo
 @param fileName - nome do arquivo
 @param array    - grupo de valores inteiros
 */
int_Array copyIntArray ( int_Array array )
{
    // definir dados locais
    int    x = 0;
    int    y = 0;
    static int_Array copy;

    if ( array.length <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
        array.length = 0;
    }
    else
    {
        // reservar area
        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 ];
            } // fim repetir
        } // fim se
    } // fim se

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

```

/**
  Method05.
 */
void method05 ( )
{
  // definir dados
  int_Array array; // arranjo sem tamanho definido
  int_Array other; // arranjo sem tamanho definido

  // identificar
  IO_id ( "EXEMPLO1010 - Method05 - v0.0" );

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

  // copiar dados
  other = copyIntArray ( array );

  // mostrar dados
  IO_printf ( "\nOriginal\n" );
  printIntArray ( array );

  // mostrar dados
  IO_printf ( "\nCopia\n" );
  printIntArray ( other );

  // 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 – Exemplo1006,c..

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

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    lines ;
    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 lines - quantidade de dados
 @param columns - quantidade de dados
 */
ref_int_Matrix new_int_Matrix ( int lines, int columns )
{
    // reserva de espaco
    ref_int_Matrix tmpMatrix = (ref_int_Matrix) malloc (sizeof(int_Matrix));

    // estabelecer valores padroes
    if ( tmpMatrix != NULL )
    {
        tmpMatrix->lines    = 0;
        tmpMatrix->columns = 0;
        tmpMatrix->data     = NULL;
        // reservar espaco
        if ( lines>0 && columns>0 )
        {
            tmpMatrix->lines    = lines;
            tmpMatrix->columns = columns;
            tmpMatrix->data     = malloc (columns * sizeof(int));
            for ( tmpMatrix->ix=0;
                  tmpMatrix->ix<tmpMatrix->lines;
                  tmpMatrix->ix=tmpMatrix->ix+1 )
            {
                tmpMatrix->data [ tmpMatrix->ix ] = (ints) malloc (columns * sizeof(int));
            } // fim repetir
        } // fim se
        tmpMatrix->ix    = 0;
        tmpMatrix->iy    = 0;
    } // fim se
    return ( tmpMatrix );
} // fim 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 tmpMatrix )
{
    // testar se ha' dados
    if ( tmpMatrix != NULL )
    {
        for ( tmpMatrix->ix=0;
              tmpMatrix->ix<tmpMatrix->lines;
              tmpMatrix->ix=tmpMatrix->ix+1 )
        {
            free ( tmpMatrix->data [ tmpMatrix->ix ] );
        } // fim repetir
        free ( tmpMatrix );
    } // fim se
} // fim free_int_Matrix ( )

// ----- metodos

/**
    printIntMatrix - Mostrar matrix com valores inteiros.
    @param array - grupo de valores inteiros
*/
void printIntMatrix ( ref_int_Matrix matrix )
{
    // mostrar valores na matriz
    for ( matrix->ix=0; matrix->ix<matrix->lines; 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 ] );
        } // fim repetir
        printf ( "\n" );
    } // fim repetir
} // printIntArray ( )

```

```

/**
  Method06.
 */
void method06 ( )
{
  // definir dado
  ref_int_Matrix matrix = new_int_Matrix ( 3, 3 );

  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;

  // identificar
  IO_id ( "EXEMPLO1010 - Method06 - v0.0" );

  // executar o metodo auxiliar
  printIntMatrix ( matrix );

  // reciclar espaco
  free_int_Matrix ( matrix );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // fim method06 ( )

```

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.

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 – Exemplo1007,c..

- 25.) Editar mudanças no nome do programa e versão.  
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 lines = 0;
    int columns = 0;
    int z = 0;
    chars text = IO_new_chars ( STR_SIZE );

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

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

    // testar se ha' espaco
    if ( matrix->data == NULL )
    {
        // nao ha' espaco
        matrix->lines = 0;
        matrix->columns = 0;
        matrix->ix = 0;
        matrix->iy = 0;
    }
    else
    {
        // ler e guardar valores na matriz
        for ( matrix->ix=0; matrix->ix<matrix->lines; 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 ) ), " : " ) ) );
            } // fim repetir
            printf ( "\n" );
        } // fim repetir
    } // fim se

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

```

/**
  Method07.
 */
void method07 ( )
{
  // definir dados
  ref_int_Matrix matrix = NULL;

  // identificar
  IO_id ( "EXEMPLO1010 - Method07 - 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" );
} // fim method07 ( )

```

OBS.:

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

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 – Exemplo1008,c..



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

Acrescentar um método para gravar dados em matriz, posição por posição.

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

```
/**
    fprintfIntMatrix    - Gravar arranjo bidimensional com valores inteiros.
    @param fileName    - nome do arquivo
    @param matrix      - grupo de valores inteiros
*/
void fprintfIntMatrix ( 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->lines );
        fprintf ( arquivo, "%d\n", matrix->columns );

        // gravar valores no arquivo
        for ( matrix->ix=0; matrix->ix<matrix->lines; 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 ] );
            } // fim repetir
        } // fim repetir
        // fechar arquivo
        fclose ( arquivo );
    } // fim se
} // fprintfIntMatrix ( )
```

```

/**
  Method08.
 */
void method08 ( )
{
  // definir dados
  ref_int_Matrix matrix = NULL;

  // identificar
  IO_id ( "EXEMPLO1010 - Method08 - 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" );
} // fim method08 ( )

```

OBS.:

Só poderão ser operados arranjos com mesma quantidade de dados.

- 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 – Exemplo1009,c..

- 33.) Editar mudanças no nome do programa e versão.  
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
    int    lines    = 0;
    int    columns = 0;
    FILE* arquivo = fopen ( fileName, "rt" );

    // ler a quantidade de dados
    fscanf ( arquivo, "%d", &lines );
    fscanf ( arquivo, "%d", &columns );

    if ( lines <= 0 || columns <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
    }
    else
    {
        // reservar espaco para armazenar
        ref_int_Matrix matrix = new_int_Matrix ( lines, columns );

        // testar se ha' espaco
        if ( matrix->data == NULL )
        {
            // nao ha' espaco
            matrix->lines    = 0;
            matrix->columns = 0;
            matrix->ix       = 0;
            matrix->iy       = 0;
        }
        else
        {
            // ler e guardar valores na matriz
            matrix->ix = 0;
            while ( ! feof ( arquivo ) && matrix->ix < matrix->lines )
            {
                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;
                } // fim repetir
                // passar ao proximo
                matrix->ix = matrix->ix+1;
            } // fim repetir
            matrix->ix = 0;
            matrix->iy = 0;
        } // fim se
    } // fim se
} // fim freadintMatrix ( )
```

```

/**
  Method09.
 */
void method09 ( )
{
  // identificar
  IO_id ( "EXEMPLO1010 - Method09 - 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" );
} // fim method09 ( )

```

OBS.:

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

- 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 – Exemplo1010,c..

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

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->lines <= 0 || matrix->columns <= 0 )
        {
            IO_printf ( "\nERRO: Valor invalido.\n" );
        }
        else
        {
            // reservar espaco
            copy = new_int_Matrix ( matrix->lines, 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->lines; 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 ];
                    } // fim repetir
                } // fim repetir
            } // fim se
        } // fim se
    } // fim se

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

```

/**
  Method10.
 */
void method10 ( )
{
  // definir dados
  ref_int_Matrix matrix = NULL;
  ref_int_Matrix other = NULL;

  // identificar
  IO_id ( "EXEMPLO1010 - Method10 - 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" );
} // fim method10 ( )

```

- 38.) Compilar o programa novamente.  
 Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.  
 Se não houver erros, seguir para o próximo passo.
- 39.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

## Exercícios

DICAS GERAIS: Consultar o Anexo C 02 na apostila para outros exemplos.

Prever, realizar e registrar todos os testes efetuados.

Integrar as chamadas de todos os programas em um só.

- 01.) Incluir em um programa (Exemplo1011) um método 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( )**, mas tentar limitar valores muito grandes.

Exemplo: valor = gerarInt ( inferior, superior );

- 02.) Incluir em um programa (Exemplo1012) uma função para procurar certo valor em um arranjo.  
Para testar, receber um nome de arquivo como parâmetro e aplicar a função sobre o arranjo com os valores lidos;

Exemplo: arranjo = lerArquivo ( "DADOS.TXT" );  
menor = procurar ( arranjo, n );

- 03.) Incluir em um programa (Exemplo1013) uma função 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 os tamanhos são compatíveis.

Exemplo: arranjo1 = lerArquivo ( "DADOS1.TXT" );  
arranjo2 = lerArquivo ( "DADOS2.TXT" );  
resposta = comparar ( arranjo1, arranjo2 );

- 04.) Incluir em um programa (Exemplo1014) uma função para operar a soma de dois arranjos, com o segundo escalado 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.

Exemplo: arranjo1 = lerArquivo ( "DADOS1.TXT" );  
arranjo2 = lerArquivo ( "DADOS2.TXT" );  
soma = somar ( arranjo1, arranjo2 );

- 05.) Incluir em um programa (Exemplo1015) uma função para dizer se um arranjo está em ordem crescente.  
Para testar, receber um nome de arquivo como parâmetro e aplicar a função sobre o arranjo com os valores lidos;

```
Exemplo: arranjo1 = lerArquivo ( "DADOS1.TXT" );  
         resposta = ordenado ( arranjo );
```

- 06.) Incluir em um programa (Exemplo1016) uma função 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.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         matriz2 = transpostaMatriz ( matriz1 );
```

- 07.) Incluir em um programa (Exemplo1017) uma função 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.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         resposta = zeroMatriz ( matriz1 );
```

- 08.) Incluir em um programa (Exemplo1018) uma função para testar a igualdade de duas matrizes pela.  
Para testar, receber dados de arquivos e aplicar a função sobre as matrizes com os valores lidos;  
DICA: Verificar se os tamanhos são compatíveis.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         matriz2 = lerMatrizDeArquivo ( "DADOS2.TXT" );  
         resposta = compararMatriz ( matriz1, matriz2 );
```

- 09.) Incluir em um programa (Exemplo1019) uma função para operar a soma de duas matrizes, com a segunda escalado 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.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         matriz2 = lerMatrizDeArquivo ( "DADOS2.TXT" );  
         soma = somarMatriz ( matriz1, matriz2 );
```



10.) Incluir em um programa (Exemplo1020) uma função 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.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         matriz2 = lerMatrizDeArquivo ( "DADOS2.TXT" );  
         soma    = multiplicarMatriz  ( matriz1, matriz2 );
```

#### Tarefas extras

E1.) Incluir em um programa (Exemplo10E1) uma função 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.

```
Exemplo: arranjo1 = lerArquivo ( "DADOS1.TXT" );  
         ordenado = ordenar    ( arranjo );
```

E2.) Incluir em um programa (Exemplo10E2) uma função 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.

```
Exemplo: matriz1 = lerMatrizDeArquivo ( "DADOS1.TXT" );  
         matriz2 = lerMatrizDeArquivo ( "DADOS2.TXT" );  
         resposta = identidadeMatriz  ( multiplicar (matriz1, matriz2) );
```