PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 11

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

01.) Editar e salvar um esboço de programa em C++, cujo nome será myarray.hpp, que conterá definições para uso posterior:

```
/*
  myarray.hpp   - v0.0. - __ / __ / _____
  Author: _____________________
*/

// ------------------------------------------------ definicoes globais

#ifndef  _MYARRAY_HPP_
#define _MYARRAY_HPP_

// dependencias

#include <iostream>
using std::cin  ;          // para entrada
using std::cout;           // para saida
using std::endl;           // para mudar de linha

#include <iomanip>
using std::setw;           // para definir espacamento

#include <string>
using std::string;         // para cadeia de caracteres

#include <fstream>
using std::ofstream;    // para gravar arquivo
using std::ifstream ;   // para ler       arquivo


template < typename T >
class Array
{
 private:       // area reservada
   T   optional;
   int length;
   T  *data;
```

```
  public:         // area aberta
   Array ( int n, T initial )
   {
    // definir valores iniciais
       optional = initial;
       length   = 0;
       data     = nullptr;

    // reservar area
       if ( n > 0 )
       {
           length  = n;
           data    = new T [ length ];
       }
   } // end constructor

   void free ( )
   {
     if ( data != nullptr )
     {
       delete ( data );
       data = nullptr;
     } // end if
   } // end free ( )

  void set ( int position, T value )
   {
     if ( 0 <= position && position < length )
     {
       data [ position ] = value;
     } // end if
   } // end set ( )

   T    get ( int position )
   {
     T value = optional;

     if ( 0 <= position && position < length )
     {
       value = data [ position ];
     } // end if

     return ( value );
   } // end get ( )

   void print ( )
   {
     cout << endl;
     for ( int x = 0; x < length; x=x+1 )
     {
        cout << setw( 3 ) << x << " : "
             << setw( 9 ) << data[ x ]
             << endl;
     } // end for
     cout << endl;
   } // end print ( )
};

#endif
```

OBS.:
Uma classe é uma outra forma de definir dados e métodos relacionados.
Pode haver partes públicas, compartilhadas ou privadas.

Editar outro programa em C++, na mesma pasta, cujo nome será Exemplo1100.cpp, para mostrar dados em arranjo:

```
/*
  Exemplo1100 - v0.0. - __ / __ / _____
  Author: _____

 */
// dependencias
#include <iostream>    // std::cin, std::cout, std:endl
#include <limits>      // std::numeric_limits
#include <string>      // para cadeias de caracteres

// ------------------------------------------------ definicoes globais

void pause ( std::string text )
{
   std::string dummy;
   std::cin.clear ( );
   std::cout << std::endl << text;
   std::cin.ignore( );
   std::getline(std::cin, dummy);
   std::cout << std::endl << std::endl;
} // end pause ( )


// ------------------------------------------------ classes / pacotes

#include "myarray.hpp"

using namespace std;

// ------------------------------------------------ metodos

/**
  Method_00 - nao faz nada.
 */
void method_00 ( )
{
 // nao faz nada
} // end method_00 ( )
```

```
/**
  Method_01 - Mostrar certa quantidade de valores.
 */
void method_01 ( )
{
 // definir dados
    Array  <int> int_array ( 5, 0 );

    int_array.set ( 0, 1 );
    int_array.set ( 1, 2 );
    int_array.set ( 2, 3 );
    int_array.set ( 3, 4 );
    int_array.set ( 4, 5 );

 // identificar
    cout << "\nMethod_01 - v0.0\n" << endl;

 // mostrar dados
    int_array.print ( );

 // reciclar espaco
    int_array.free ( );

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

/**
  Method_02.
 */
void method_02 ( )
{
 // identificar
    cout <<  endl << "Method_02 - v0.0" << endl;

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

/**
  Method_03.
 */
void method_03 ( )
{
 // identificar
    cout <<  endl << "Method_03 - v0.0" << endl;

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

```
/**
  Method_04.
 */
void method_04 ( )
{
 // identificar
    cout <<  endl << "Method_04 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_04 ( )

/**
  Method_05.
 */
void method_05 ( )
{
 // identificar
    cout <<  endl << "Method_05 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // end method_05 ( )

/**
  Method_06.
 */
void method_06 ( )
{
 // identificar
    cout <<  endl << "Method_06 - v0.0" << endl;

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

/**
  Method_07.
 */
void method_07 ( )
{
 // identificar
    cout <<  endl << "Method_07 - v0.0" << endl;

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

/**
  Method_08.
 */
void method_08 ( )
{
 // identificar
    cout <<  endl << "Method_08 - v0.0" << endl;

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

```
/**
   Method_09.
 */
void method_09 ( )
{
 // identificar
    cout <<  endl << "Method_09 - v0.0" << endl;

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


/**
   Method_10.
 */
void method_10 ( )
{
 // identificar
    cout <<  endl << "Method_10 - v0.0" << endl;

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

// ------------------------------------------------- acao principal

/*
  Funcao principal.
  @return codigo de encerramento
*/
int main ( int argc, char** argv )
{
 // definir dado
    int x = 0;                  // definir variavel com valor inicial

 // repetir até desejar parar
    do
    {
     // identificar
        cout << "EXEMPLO1101 - Programa - v0.0\n       " << endl;

     // mostrar opcoes
        cout << "Opcoes                                 " << endl;
        cout << " 0 - parar                             " << endl;
        cout << " 1 - mostrar dados inteiros em arranjo " << endl;
        cout << " 2 -                                   " << endl;
        cout << " 3 -                                   " << endl;
        cout << " 4 -                                   " << endl;
        cout << " 5 -                                   " << endl;
        cout << " 6 -                                   " << endl;
        cout << " 7 -                                   " << endl;
        cout << " 8 -                                   " << endl;
        cout << " 9 -                                   " << endl;
        cout << "10 -                                   " << endl;

     // ler do teclado
        cout << endl << "Entrar com uma opcao: ";
        cin  >> x;
```

```
    // escolher acao
      switch ( x )
      {
        case 0:   method_00 ( );   break;
        case 1:   method_01 ( );   break;
        case 2:   method_02 ( );   break;
        case 3:   method_03 ( );   break;
        case 4:   method_04 ( );   break;
        case 5:   method_05 ( );   break;
        case 6:   method_06 ( );   break;
        case 7:   method_07 ( );   break;
        case 8:   method_08 ( );   break;
        case 9:   method_09 ( );   break;
        case 10: method_10 ( );   break;
        default:
          cout << endl << "ERRO: Valor invalido." << endl;
      } // end switch
    }
    while ( x != 0 );

  // encerrar
    pause  ( "Apertar ENTER para terminar" );
    return ( 0 );
} // end main ( )

/*
---------------------------------------------- documentacao complementar

---------------------------------------------- notas / observacoes / comentarios

---------------------------------------------- previsao de testes

---------------------------------------------- historico

Versao      Data              Modificacao
 0.1        __/__             esboco

 ---------------------------------------------- testes

Versao      Teste
 0.1        01. ( OK )        identificacao de programa

*/
```

OBS.:
Entradas e saídas utilizarão as primitivas da linguagem C++ para sequências (**streams**).

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 na biblioteca outro método para ler e guardar dados em arranjo.

```
void read ( )
{
  cout << endl;
  for ( int x = 0; x < length; x=x+1 )
  {
    cout << setw( 3 ) << x << " : ";
    cin   >> data[ x ];
  } // end for
  cout << endl;
} // end read ( )
```

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

```
/**
  Method_02.
 */
void method_02 ( )
{
 // definir dados
   Array  <int> int_array ( 5, 0 );

 // identificar
   cout <<  endl << "Method_02 - v0.0" << endl;

 // ler dados
   int_array.read  ( );

 // mostrar dados
   int_array.print ( );

 // reciclar espaco
   int_array.free ( );

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

OBS.:
Só poderá ser mostrado o arranjo em que existir algum conteúdo
(diferente de **nullptr** = 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 na biblioteca outro método para gravar em arquivo dados no arranjo.

```
void fprint ( string fileName )
{
  ofstream afile;       // output file

  afile.open ( fileName );
  afile << length << endl;
  for ( int x = 0; x < length; x=x+1 )
  {
     afile << data[ x ] << endl;
  } // end for
  afile.close ( );
} // end fprint ( )
```

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

```
/**
  Method_03.
 */
void method_03 ( )
{
 // definir dados
   Array  <int> int_array ( 5, 0 );

 // identificar
   cout <<  endl << "Method_03 - v0.0" << endl;

 // ler dados
   int_array.read  ( );

 // mostrar dados
   int_array.fprint ( "INT_ARRAY1.TXT" );

 // reciclar espaco
   int_array.free ( );

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

OBS.:

Entradas e saídas para arquivos utilizarão as primitivas da linguagem C++ para sequências (**fstreams**).

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 na biblioteca outro método para ler arquivo e guardar dados em arranjo.

```
void fread ( string fileName )
{
  ifstream afile;        // input file
  int n = 0;
  afile.open ( fileName );
  afile >> n;
  if ( n <= 0 )
  {
    cout << "\nERROR: Invalid length.\n" << endl;
  }
  else
  {
   // guardar a quantidade de dados
     length  = n;
   // reservar area
     data    = new T [ n ];
   // ler dados
     for ( int x = 0; x < length; x=x+1 )
     {
       afile >> data[ x ];
     } // end for
  } // end if
  afile.close ( );
} // end fread ( )
```

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

```
/**
  Method_04.
 */
void method_04 ( )
{
 // definir dados
   Array  <int> int_array ( 5, 0 );

 // identificar
   cout <<  endl << "Method_04 - v0.0" << endl;

 // ler dados
   int_array.fread ( "INT_ARRAY1.TXT" );

 // mostrar dados
   int_array.print ( );

 // reciclar espaco
   int_array.free ( );

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_04 ( )
```

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

11.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

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

13.) Acrescentar na biblioteca outros construtores e
um método para criar um objeto com dados copiados de um arranjo comum.

```
Array ( )                  // construtor padrao
{
 // definir valores iniciais
    length  = 0;
 // reservar area
    data    = nullptr;
} // end constructor

                           // contrutor baseado em copia
Array ( int length, int other [ ] )
{
  if ( length <= 0 )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
   // criar copia
     this->length = length;
   // reservar area
     data   = new T [ this->length ];
   // ler dados
     for ( int x = 0; x < this->length; x=x+1 )
     {
        data [ x ] = other [ x ];
     } // end for
  } // end if
} // end constructor ( )
```

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

```
/**
   Method_05.
 */
void method_05 ( )
{
 // definir dados
    int other [ ] = { 1, 2, 3, 4, 5 };
    Array  <int> int_array ( 5, other );

 // identificar
    cout <<  endl << "Method_05 - v0.0" << endl;

 // mostrar dados
    cout << "\nCopia\n" << endl;
    int_array.print (  );

 // reciclar espaco
    int_array.free ( );

 // encerrar
    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 na biblioteca mais um construtor para criar um objeto a partir de outro existente.

```
Array ( const Array& other )
{
  if ( other.length <= 0 )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
   // criar copia
     length = other.length;
   // reservar area
     data    = new T [ other.length ];
   // ler dados
     for ( int x = 0; x < length; x=x+1 )
     {
        data [ x ] = other.data [ x ];
     } // end for
  } // end if
} // end constructor ( )
```

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

```
/**
  Method_06.
 */
void method_06 ( )
{
 // definir dados
    Array  <int> int_array1 ( 1, 0 );

 // identificar
    cout <<  endl << "Method_06 - v0.0" << endl;

 // ler dados
    int_array1.fread ( "INT_ARRAY1.TXT" );

 // mostrar dados
    cout << "\nOriginal\n" << endl;
    int_array1.print ( );

 // criar   copia
    Array <int> int_array2 ( int_array1 );

 // mostrar dados
    cout << "\nCopia\n" << endl;
    int_array2.print ( );

 // reciclar espaco
    int_array1.free ( );
    int_array2.free ( );

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

OBS.:
Só poderão ser copiados os dados correspondentes à quantidade, se houver espaço e dados.

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 na biblioteca um operador para copiar arranjo para outro.

```
Array& operator= ( const Array <T> other )
{
  if ( other.length <= 0 )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
     this->length = other.length;
     this->data    = new T [ other.length ];
     for ( int x = 0; x < this->length; x=x+1 )
     {
        data [ x ] = other.data [ x ];
     } // end for
  } // end if
  return ( *this );
} // end operator= ( )
```

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

```
/**
  Method_07.
 */
void method_07 ( )
{
 // definir dados
   Array  <int> int_array1 ( 1, 0 );
   Array  <int> int_array2 ( 1, 0 );

 // identificar
   cout << endl << "Method_07 - v0.0" << endl;

 // ler dados
   int_array1.fread ( "INT_ARRAY1.TXT" );

 // mostrar dados
   cout << "\nOriginal\n" << endl;
   int_array1.print ( );

 // copiar  dados
   int_array2 = int_array1;

 // mostrar dados
   cout << "\nCopia\n" << endl;
   int_array2.print ( );

 // reciclar espaco
   int_array1.free ( );
   int_array2.free ( );

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

OBS.:
Só poderão ser copiados os dados correspondentes à quantidade, se houver espaço e dados.

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 testar a igualdade de dois arranjos, posição por posição.

```
bool operator== ( const Array <T> other )
{
  bool result = false;
  int  x      = 0;

  if ( other.length == 0 || length != other.length )
  {
    cout << "\nERROR: Missing data.\n" << endl;
  }
  else
  {
    x = 0;
    result = true;
    while ( x < this->length && result )
    {
      result = result &&  (data [ x ] == other.data [ x ] );
      x = x + 1;
    } // end while
  } // end if
  return ( result );
} // end operator== ( )
```

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

```
/**
  Method_08.
 */
void method_08 ( )
{
 // definir dados
   int   other [ ] = { 1, 2, 3 };
   Array <int> int_array1 ( 3, other );
   Array <int> int_array2 ( 3, other );
   int x;

 // identificar
   cout <<  endl << "Method_08 - v0.0" << endl;

 // mostrar dados
   cout << endl;
   cout << "Array_1";
   int_array1.print ( );
```

```
// mostrar dados
   cout << "Array_2";
   int_array2.print ( );

// mostrar comparacao
   cout << "Igualdade = " << (int_array1==int_array2) << endl;

// alterar dado
   int_array2.set ( 0, (-1) );

// mostrar dados
   cout << endl;
   cout << "Array_1" << endl;
   int_array1.print ( );

   cout << "Array_2" << endl;
   int_array2.print ( );

// mostrar comparacao
   cout << "Igualdade = " << (int_array1==int_array2) << endl;

// reciclar espaco
   int_array1.free ( );
   int_array2.free ( );

// encerrar
   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 um operador para somar valores em dois arranjos, posição por posição.

```
Array& operator+ ( const Array <T> other )
{
   static Array <T> result ( other );

   if ( other.length <= 0 )
   {
      cout << "\nERROR: Missing data.\n" << endl;
   }
   else
   {
      for ( int x = 0; x < this->length; x=x+1 )
      {
         result.data [ x ] = result.data [ x ] + this->data [ x ];
      } // end for
   } // end if
   return ( result );
} // end operator+ ( )
```

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

```
/**
  Method_09.
 */
void method_09 ( )
{
 // definir dados
    Array  <int> int_array1 ( 1, 0 );
    Array  <int> int_array2 ( 1, 0 );
    Array  <int> int_array3 ( 1, 0 );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method_09 - v0.0" << endl;

 // ler dados
    int_array1.fread ( "INT_ARRAY1.TXT" );

 // copiar  dados
    int_array2 = int_array1;

 // somar   dados
    int_array3 = int_array2 + int_array1;

 // mostrar dados
    cout << endl;
    cout << "Original" << endl;
    int_array1.print ( );

 // mostrar dados
    cout << "Copia"    << endl;
    int_array2.print ( );

 // mostrar dados
    cout << "Soma"     << endl;
    int_array3.print ( );

 // reciclar espaco
    int_array1.free ( );
    int_array2.free ( );
    int_array3.free ( );

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

OBS.:
Só poderão ser operados arranjos com mesma quantidade de dados.

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 na biblioteca para acessos externos aos valores no arranjo.

```
const int getLength ( )
{
  return ( length );
} // end getLength ( )

T& operator[]( const int position )
{
  static T value = optional;

  if ( position < 0 || length <= position )
  {
    cout << endl << "\nERROR: Invalid position.\n" << endl;
  }
  else
  {
   value = data [ position ];
  } // end if
  return ( value );
} // end operator [ ]
```

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

```
/**
  Method_10.
 */
void method_10 ( )
{
 // definir dados
   int other [ ] = { 1, 2, 3, 4, 5 };
   Array  <int> int_array ( 5, other );
   int x;

 // identificar
   cout <<  endl << "Method_10 - v0.0" << endl;

 // mostrar dados
   cout << "\nAcesso externo" << endl;
   for ( x=0; x<int_array.getLength( ); x=x+1 )
   {
      cout << endl << setw( 3 ) << x << " : " << int_array [ x ];
   } // fim repetir
   cout << endl << "\nFora dos limites:";
   cout << endl << "[-1]: " << int_array.get(-1) << endl;
   cout << endl << "["      << int_array.getLength( ) << "]: "
                            << int_array [ int_array.getLength( ) ] << endl;

 // reciclar espaco
   int_array.free ( );

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

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

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.
NÃO usar métodos ou funções já prontos em bibliotecas/classes nativas da linguagem.
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 métodos (1111) para
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 = arranjo.randomIntGenerate (inferior, superior);

02.) Incluir uma função (1112) para
procurar o maior valor par 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 primeiro valor par, se houver, como referência inicial.

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

03.) Incluir uma função (1113) para
procurar o maior valor par múltiplo de 3 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 primeiro valor par múltiplo de 3, se houver, como referência inicial.

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");
menor = arranjo.searchFirstEvenx3 ( );

04.) Incluir uma função (1114) para
somar todos os valores em um arranjo entre duas posições dadas.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos;

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");
soma = arranjo.addInterval (inicio, fim);

05.) Incluir uma função (1115) para
achar a média dos valores em um arranjo entre duas posições dadas.
Para testar, receber um nome de arquivo como parâmetro e
aplicar a função sobre o arranjo com os valores lidos;

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");
media = arranjo.averageInterval (inicio, fim);

06.) Incluir uma função (1116) para
verificar se todos os valores são positivos e menores que 100 em um arranjo.
Para testar, ler o arquivo ("DADOS.TXT")
armazenar os dados em um arranjo.

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

07.) Incluir uma função (1117) para
dizer se está ordenado, ou não, em ordem decrescente.
DICA: Testar se não está desordenado, ou seja,
se existe algum valor que seja maior que o seguinte.
Não usar *break* !

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

08.) Incluir uma função (1118) para
dizer se determinado valor está presente em arranjo,
entre duas posições indicadas.
Para testar, ler o arquivo ("DADOS.TXT"),
e armazenar os dados em arranjo;
ler do teclado um valor inteiro para ser procurado;
determinar se o valor procurado existe no arranjo.
DICA: Usar repetição com expressão lógica para parar quando encontrar.
Não usar *break* !

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");
existe = arranjo.searchInterval (procurado, inicio, fim);

09.) Incluir uma função (1119) para
escalar dados em arranjo, entre duas posições dadas,
multiplicando cada valor por uma constante.
Para testar, ler o arquivo ("DADOS.TXT"),
e armazenar os dados em arranjo;
ler do teclado um valor inteiro para indicar a constante.

Exemplo: arranjo = readArrayFromFile ("DADOS.TXT");
novo = arranjo.scalar(constante, inicio, fim);

10.) Incluir um método (1120) para
colocar valores em arranjo em ordem decrescente,
mediante trocas de posições, até ficar totalmente ordenado.
Para testar, ler o arquivo ("DADOS.TXT"),
e armazenar os dados em arranjo.

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

Tarefas extras

E1.) Incluir uma função/operador (11E1) para
dizer se dois arranjos são diferentes, pelo menos em uma posição.

E2.) Incluir uma função/operador (11E2) para
calcular a quantidade de diferenças entre dois arranjos,
considerados os valores posição por posição,
caso tenham tamanhos iguais.