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
   int length;
   T  *data;

 public:        // area aberta
   Array ( int n )
   {
    // definir valor inicial
      length  = n;
      data    = NULL;

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

```
   void free ( )
   {
     if ( data != NULL )
     {
       delete ( data );
       data = NULL;
     } // 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 = 0;              // return value has type dependency

    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 ou privadas.

Editar outro programa em C++, na mesma pasta, cujo nome será Exemplo1101.cpp, para mostrar dados em arranjo:

```
/*
  Exemplo1101 - 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

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

```
/**
   Method01 - Mostrar certa quantidade de valores.
 */
void method01 ( )
{
 // definir dados
    Array  <int> int_array ( 5 );

    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 << "\nEXEMPLO1101 - Method01 - v0.0\n" << endl;

 // mostrar dados
    int_array.print ( );

 // reciclar espaco
    int_array.free ( );

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

/**
   Method02.
 */
void method02 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method02 - v0.0" << endl;

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

/**
   Method03.
 */
void method03 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method03 - v0.0" << endl;

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

```
/**
  Method04.
 */
void method04 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method04 - v0.0" << endl;

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

/**
  Method05.
 */
void method05 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method05 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method05 ( )

/**
  Method06.
 */
void method06 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method06 - v0.0" << endl;

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

/**
  Method07.
 */
void method07 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method07 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method07 ( )

/**
  Method08.
 */
void method08 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method08 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method08 ( )
```

```
/**
  Method09.
 */
void method09 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method09 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method09 ( )


/**
  Method10.
 */
void method10 ( )
{
 // identificar
    cout <<  endl << "EXEMPLO1101 - Method10 - v0.0" << endl;

 // encerrar
    pause ( "Apertar ENTER para continuar" );
} // fim method10 ( )

// ------------------------------------------------ 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:
          method00 ( );
         break;
        case 1:
          method01 ( );
         break;
        case 2:
          method02 ( );
         break;
        case 3:
          method03 ( );
         break;
        case 4:
          method04 ( );
         break;
        case 5:
          method05 ( );
         break;
        case 6:
          method06 ( );
         break;
        case 7:
          method07 ( );
         break;
        case 8:
          method08 ( );
         break;
        case 9:
          method09 ( );
         break;
        case 10:
          method10 ( );
         break;
        default:
          cout << endl << "ERRO: Valor invalido." << endl;
      } // fim escolher
    }
    while ( x != 0 );

  // encerrar
    pause  ( "Apertar ENTER para terminar" );
    return ( 0 );
} // fim 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.) Copiar a versão atual do programa para outra nova – Exemplo1102.cpp.

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

```
/**
   Method02.
 */
void method02 ( )
{
 // definir dados
    Array  <int> int_array ( 5 );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method02 - v0.0" << endl;

 // ler dados
    int_array.read  ( );

 // mostrar dados
    int_array.print ( );

 // reciclar espaco
    int_array.free ( );

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

OBS.:
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 – Exemplo1103.cpp.

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

```
/**
  Method03.
 */
void method03 ( )
{
 // definir dados
   Array  <int> int_array ( 5 );

 // identificar
   cout <<  endl << "EXEMPLO1110 - Method03 - 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" );
} // fim method03 ( )
```

OBS.:

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

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 – Exemplo1104.cpp.

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

```
/**
   Method04.
 */
void method04 ( )
{
 // definir dados
    Array  <int> int_array ( 5 );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method04 - 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" );
} // 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 – Exemplo1105.cpp.

17.) Editar mudanças no nome do programa e versão.
Acrescentar na biblioteca outros construtores e um método para criar um objeto com dados copiados de um arranjo comum.

```
Array ( )                  // construtor padrao
{
 // definir valor inicial
    length  = 0;
 // reservar area
    data     = NULL;
} // 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 ( )
```

```
  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 o novo.

```
/**
   Method05.
 */
void method05 ( )
{
 // definir dados
    int other [ ] = { 1, 2, 3, 4, 5 };
    Array  <int> int_array ( 5, other );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method05 - v0.0" << endl;

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

 // reciclar espaco
    int_array.free ( );

 // encerrar
    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 – Exemplo1106.cpp.

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

```
/**
   Method06.
 */
void method06 ( )
{
 // definir dados
    Array  <int> int_array1 ( 1 );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method06 - 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" );
} // fim method06 ( )
```

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

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 – Exemplo1107.cpp.

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

```
/**
   Method07.
 */
void method07 ( )
{
 // definir dados
    Array  <int> int_array1 ( 1 );
    Array  <int> int_array2 ( 1 );

 // identificar
    cout << endl << "EXEMPLO1110 - Method07 - 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" );
} // fim method07 ( )
```

OBS.:
Só poderão ser copiados os dados correspondentes à quantidade, se houver espaço e 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.) Copiar a versão atual do programa para outra nova – Exemplo1108.cpp.

29.) Editar mudanças no nome do programa e versão.
Acrescentar um método para testar a igualdade de dois arranjos, posição por posição.

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

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

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

```
/**
  Method08.
 */
void method08 ( )
{
 // definir dados
   int   other [ ] = { 1, 2, 3 };
   Array <int> int_array1 ( 3, other );
   Array <int> int_array2 ( 3, other );
   int x;

 // identificar
   cout <<  endl << "EXEMPLO1110 - Method08 - 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 ( );

// mostrar dados
   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" );
} // 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 – Exemplo1109.cpp.

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

```
/**
   Method09.
 */
void method09 ( )
{
 // definir dados
    Array  <int> int_array1 ( 1 );
    Array  <int> int_array2 ( 1 );
    Array  <int> int_array3 ( 1 );

 // identificar
    cout <<  endl << "EXEMPLO1110 - Method09 - 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" );
} // fim method09 ( )
```

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

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 – Exemplo1110.cpp.

37.) Editar mudanças no nome do programa e versão.
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 = 0;   // return value has type dependency

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

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

```
/**
  Method10.
 */
void method10 ( )
{
 // definir dados
   int other [ ] = { 1, 2, 3, 4, 5 };
   Array  <int> int_array ( 5, other );
   int x;

 // identificar
   cout <<  endl << "EXEMPLO1110 - Method10 - 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" );
} // fim method10 ( )
```

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

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 (Exemplo1111)
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.gerarInt (inferior, superior);

02.) Incluir em um programa (Exemplo1112) uma função para
procurar o maior valor 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 como referência inicial.

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
maior = arranjo.acharMaior ( );

03.) Incluir em um programa (Exemplo1113) uma função para
procurar o menor valor 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 maior valor como referência inicial.

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
menor = arranjo.acharMenor ( );

04.) Incluir em um programa (Exemplo1114) uma função para
somar todos os valores 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");
soma = arranjo.somarValores ( );

05.) Incluir em um programa (Exemplo1115) uma função para
achar a média dos valores 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");
media = arranjo.mediaValores( );

06.) Incluir em um programa (Exemplo1116) uma função para
verificar se todos os valores são iguais a zero em um arranjo.
Para testar, ler o arquivo ("DADOS.TXT")
armazenar os dados em um arranjo.

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
teste = arranjo.zeros ( );

07.) Incluir em um programa (Exemplo1117) uma função para
dizer se está ordenado, ou não, em ordem crescente.
DICA: Testar se não está desordenado, ou seja,
se existe algum valor que seja menor que o seguinte.
Não usar ***break*** !

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
teste = arranjo.crescente ( );

08.) Incluir em um programa (Exemplo1118) uma função 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.

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
existe = arranjo.acharValor (procurado, 5, 10);

09.) Incluir em um programa (Exemplo1119) uma função para
escalar o arranjo, 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 = lerArquivo ("DADOS.TXT");
novo = arranjo.escalar(constante);

10.) Incluir em um programa (Exemplo1120) um método para
colocar valores em arranjo em ordem crescente,
mediante trocas de posições, até ficar totalmente ordenado.
Para testar, ler o arquivo ("DADOS.TXT"),
e armazenar os dados em arranjo.

Exemplo: arranjo = lerArquivo ("DADOS.TXT");
arranjo.ordenar ( );

Tarefas extras

E1.) Incluir em um programa (Exemplo11E1) um operador (!=) para
dizer se dois arranjos são diferentes, pelo menos em uma posição.

E2.) Incluir em um programa (Exemplo11E2) um operador (-) para
calcular as diferenças entre dois arranjos, posição por posição.