

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

- 01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo0800.c, para mostrar dados em arranjo:

```
/**
  printIntArray - Mostrar arranjo com valores inteiros.
  @param n      - quantidade de valores
  @param array  - grupo de valores inteiros
*/
void printIntArray ( int n, int array [ ] )
{
  // definir dado local
  int x = 0;

  // mostrar valores no arranjo
  for ( x=0; x<n; x=x+1 )
  {
    // mostrar valor
    IO_printf ( "%2d: %d\n", x, array [ x ] );
  } // end for
} // end printIntArray ( )

/**
  Method_01 - Mostrar certa quantidade de valores.
*/
void method_01 ( )
{
  // definir dado
  int array [ ] = { 1, 2, 3, 4, 5 };

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

  // executar o metodo auxiliar
  printIntArray ( 5, array );

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

OBS.:

A atribuição direta de todos os valores ao arranjo só é permitida quando da sua definição.

- 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.

```
/**
    readIntArray - Ler arranjo com valores inteiros.
    @param n      - quantidade de valores
    @param array  - grupo de valores inteiros
*/
void readIntArray ( int n, int array [ ] )
{
    // definir dados locais
    int x = 0;
    int y = 0;
    chars text = IO_new_chars ( STR_SIZE );

    // ler e guardar valores em arranjo
    for ( x=0; x<n; x=x+1 )
    {
        // ler valor
        strcpy ( text, STR_EMPTY );
        y = IO_readint ( IO_concat (
                                IO_concat ( text, IO_toString_d ( x ) ), " : " ) );
        // guardar valor
        array [ x ] = y;
    } // end for
} // end readIntArray ( )

/**
    Method_02.
*/
void method_02 ( )
{
    // definir dados
    int n = 5;           // quantidade de valores
    int array [ n ];

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

    // ler dados
    readIntArray ( n, array );

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

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

OBS.:

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 n        - quantidade de valores
    @param array    - grupo de valores inteiros
*/
void fprintfIntArray ( chars fileName, int n, int array [ ] )
{
    // definir dados locais
    FILE* arquivo = fopen ( fileName, "wt" );
    int x = 0;

    // gravar quantidade de dados
    IO_printf ( arquivo, "%d\n", n );

    // gravar valores no arranjo
    for ( x=0; x<n; x=x+1 )
    {
        // gravar valor
        IO_printf ( arquivo, "%d\n", array [ x ] );
    } // end for

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

/**
    Method_03.
*/
void method_03 ( )
{
    // definir dados
    int n = 5;           // quantidade de valores
    int array [ n ];

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

    // ler dados
    readIntArray ( n, array );

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

    // 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" );
```

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

```
    if ( n <= 0 )
    {
        IO_println ( "ERRO: Valor invalido." );
        n = 0;
    } // end if
```

```
    // retornar dado lido
    return ( n );
} // end freadArraySize ( )
```

```
/**
 freadIntArray      - Ler arranjo com valores inteiros.
 @param fileName - nome do arquivo
 @param n          - quantidade de valores
 @param array      - grupo de valores inteiros
 */
```

```
void freadIntArray ( chars fileName, int n, int array [ ] )
```

```
{
    // definir dados locais
    int x = 0;
    int y = 0;
    FILE* arquivo = fopen ( fileName, "rt" );
```

```
    // ler a quantidade de dados
    IO_fscanf ( arquivo, "%d", &x );
```

```

    if ( n <= 0 || n > x )
    {
        IO_println ( "ERRO: Valor invalido." );
    }
    else
    {
        // ler e guardar valores em arranjo
        x = 0;
        while ( ! feof ( arquivo ) && x < n )
        {
            // ler valor
            IO_fscanf ( arquivo, "%d", &y);
            // guardar valor
            array [ x ] = y;
            // passar ao proximo
            x = x+1;
        } // end while
    } // end if

} // end freadIntArray ( )

/**
    Method_04.
*/
void method_04 ( )
{
    // definir dados
    int n = 0;           // quantidade de valores

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

    // ler dados
    n = freadArraySize ( "ARRAY1.TXT" );

    if ( n <= 0 )
    {
        IO_printf ( "\nERRO: Valor invalido.\n" );
    }
    else
    {
        // definir armazenador
        int array [ n ];
        // ler dados
        freadIntArray ( "ARRAY1.TXT", n, array );
        // mostrar dados
        IO_printf ( "\n" );
        printIntArray ( n, array );
    } // end if

    // encerrar
    IO_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 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.
    @param n      - quantidade de valores
    @param copy   - copia do grupo de valores inteiros
    @param array  - grupo de valores inteiros
*/
void copyIntArray ( int n, int copy [ ], int array [ ] )
{
    // definir dados locais
    int x = 0;
    int y = 0;

    if ( n <= 0 )
    {
        IO_println ( "ERRO: Valor invalido." );
        n = 0;
    }
    else
    {
        // copiar valores em arranjo
        for ( x = 0; x < n; x = x + 1 )
        {
            // copiar valor
            copy [ x ] = array [ x ];
        } // end for
    } // end if
} // end copyIntArray ( )

/**
    Method_05.
*/
void method_05 ( )
{
    // definir dados
    int n = 0;           // quantidade de valores

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

    // ler a quantidade de dados
    n = freadArraySize ( "ARRAY1.TXT" );
```

```

if ( n <= 0 )
{
    IO_printf ( "\nERRO: Valor invalido.\n" );
}
else
{
    // definir armazenador
    int array [ n ];
    int other [ n ];
    // ler dados
    freadIntArray ( "ARRAY1.TXT", n, array );
    // copiar dados
    copyIntArray ( n, other, array );
    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntArray ( n, array );
    // mostrar dados
    IO_printf ( "\nCopia\n" );
    printIntArray ( n, 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 função para somar os dados em um arranjo.
Na parte principal, incluir a chamada do método para testar a função.

```

/**
    sumIntArray - Somar valores em arranjo com inteiros.
    @return     - soma dos valores
    @param n    - quantidade de valores
    @param array - grupo de valores inteiros
*/
int sumIntArray ( int n, int array [ ] )
{
    // definir dados locais
    int soma = 0;
    int x    = 0;
    // mostrar valores no arranjo
    for ( x=0; x<n; x=x+1 )
    {
        // somar valor
        soma = soma + array [ x ];
    } // end for
    // retornar resposta
    return ( soma );
} // end sumIntArray ( )

```

```

/**
  Method_06.
 */
void method_06 ( )
{
  // definir dados
  int n = 0;           // quantidade de valores

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

  // ler a quantidade de dados
  n = freadArraySize ( "ARRAY1.TXT" );

  if ( n <= 0 )
  {
    IO_printf ( "\nERRO: Valor invalido.\n" );
  }
  else
  {
    // definir armazenador
    int array [ n ];
    // ler dados
    freadIntArray ( "ARRAY1.TXT", n, array );
    // mostrar a soma dos valores no arranjo
    IO_printf ( "\nSoma = %d\n", sumIntArray ( n, array ) );
  } // end if

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

```

OBS.:

Só poderão ser somados os dados correspondentes à quantidade, se houver.

- 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 dizer se um arranjo é todo igual a zero.
Na parte principal, incluir a chamada do método para testar a função.

```

/**
  isAllZeros    - Testar valores inteiros em arranjo.
  @return       - true, se todos os dados forem iguais a zero;
                 false, caso contrario
  @param n      - quantidade de valores
  @param array  - grupo de valores inteiros
 */
bool isAllZeros ( int n, int array [ ] )
{
  // definir dados locais
  bool result = true;
  int x       = 0;

```



```

// mostrar valores no arranjo
x=0;
while ( x<n && result )
{
    // testar valor
    result = result && ( array [ x ] == 0 );
    // passar ao proximo
    x = x + 1;
} // end while

// retornar resposta
return ( result );
} // end isAllZeros ( )

/**
    Method_07.
*/
void method_07 ( )
{
    // definir dados
    int n = 5;           // quantidade de valores
    int array1 [ ] = { 0,0,0,0,0 };
    int array2 [ ] = { 1,2,3,4,5 };
    int array3 [ ] = { 1,2,0,4,5 };

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

    // testar dados
    IO_println ( "\nArray1" );
    printIntArray ( n, array1 );
    IO_printf ( "isAllZeros (array1) = %d", isAllZeros (n, array1) );

    IO_println ( "\nArray2" );
    printIntArray ( n, array2 );
    IO_printf ( "isAllZeros (array2) = %d", isAllZeros (n, array2) );

    IO_println ( "\nArray3" );
    printIntArray ( n, array3 );
    IO_printf ( "isAllZeros (array3) = %d", isAllZeros (n, array3) );

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

```

OBS.:

Só deverá ser verificado o arranjo que possuir dados (não ser vazio).

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 somar dados em dois arranjos, posição por posição.
Na parte principal, incluir a chamada do método para testar a função.

```
/**
    addIntArray - Somar arranjos com inteiros.
    @return      - arranjo com a soma resultante
    @param n      - quantidade de valores
    @param array3 - soma de grupo de valores inteiros
    @param array1 - grupo de valores inteiros (1)
    @param k      - constante para multiplicar o segundo arranjo
    @param array2 - grupo de valores inteiros (2)
*/
void addIntArray ( int n, int array3 [ ],
                  int array1 [ ], int k, int array2 [ ] )
{
    // definir dados locais
    int x = 0;

    // mostrar valores no arranjo
    for ( x=0; x<n; x=x+1 )
    {
        // somar valor
        array3 [ x ] = array1 [ x ] + k * array2 [ x ];
    } // end for
} // end addIntArray ( )

/**
    Method_08.
*/
void method_08 ( )
{
    // definir dados
    int n = 0;           // quantidade de valores

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

    // ler a quantidade de dados
    n = freadArraySize ( "ARRAY1.TXT" );
}
```

```

if ( n <= 0 )
{
    IO_printf ( "\nERRO: Valor invalido.\n" );
}
else
{
    // definir armazenador
    int array [ n ];
    int other [ n ];
    int sum [ n ];
    // ler dados
    freadIntArray ( "ARRAY1.TXT", n, array );
    // copiar dados
    copyIntArray ( n, other, array );
    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntArray ( n, array );
    // mostrar dados
    IO_printf ( "\nCopia\n" );
    printIntArray ( n, other );

    // operar soma de arranjos
    addIntArray ( n, sum, array, (-2), other );

    // mostrar resultados
    IO_printf ( "\nSoma\n" );
    printIntArray ( n, sum );
} // end if

// 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 testar a igualdade de dois arranjos, posição por posição.
Na parte principal, incluir a chamada do método para testar a função.

```
/**
isEqual      - Testar arranjos com inteiros sao iguais.
@return      - true, se todos os dados forem iguais;
              false, caso contrario
@param n     - quantidade de valores
@param array1 - grupo de valores inteiros (1)
@param array2 - grupo de valores inteiros (2)
*/
bool isEqual ( int n, int array1 [ ], int array2 [ ] )
{
    // definir dados locais
    bool result = true;
    int x      = 0;

    // mostrar valores no arranjo
    x = 0;
    while ( x < n && result )
    {
        // testar valor
        result = result && ( array1 [ x ] == array2 [ x ] );
        // passar ao proximo
        x = x + 1;
    } // end while

    // retornar resposta
    return ( result );
} // end isEqual ( )

/**
    Method_09.
*/
void method_09 ( )
{
    // definir dados
    int n = 0;           // quantidade de valores

    // identificar
    IO_id ( "Method_09 - v0.0" );

    // ler a quantidade de dados
    n = freadArraySize ( "ARRAY1.TXT" );
}
```

```

if ( n <= 0 )
{
    IO_printf ( "\nERRO: Valor invalido.\n" );
}
else
{
    // definir armazenador
    int array [ n ];
    int other [ n ];
    // ler dados
    freadIntArray ( "ARRAY1.TXT", n, array );
    // copiar dados
    copyIntArray ( n, other, array );
    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntArray ( n, array );
    // mostrar dados
    IO_printf ( "\nCopia\n" );
    printIntArray ( n, other );

    // mostrar resultado da comparacao
    IO_printf ( "\nIguais = %d\n", isEqual ( n, array, other ) );

    // modificar um valor
    other [ 0 ] = (-1) * other [ 0 ];

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

    // mostrar resultado da comparacao
    IO_printf ( "\nIguais = %d\n", isEqual ( n, array, other ) );
} // end if

// encerrar
IO_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 uma função para procurar por certo valor em um arranjo.
Na parte principal, incluir a chamada do método para testar a função.

```
/**
searchArray - Procurar valor em arranjo com inteiros.
@return      - true, se encontrar;
              false, caso contrario
@param value - valor a ser procurado
@param n     - quantidade de valores
@param array - grupo de valores inteiros
*/
bool searchArray ( int value, int n, int array [ ] )
{
// definir dados locais
bool result = false;
int x      = 0;

// mostrar valores no arranjo
x = 0;
while ( x < n && ! result )
{
// testar valor
result = ( value == array [ x ] );
// passar ao proximo
x = x + 1;
} // end while

// retornar resposta
return ( result );
} // end searchArray ( )

/**
Method_10.
*/
void method_10 ( )
{
// definir dados
int n      = 0; // quantidade de valores
int value = 0; // valor a ser procurado

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

// ler a quantidade de dados
n = freadArraySize ( "ARRAY1.TXT" );
```

```

if ( n <= 0 )
{
    IO_printf ( "\nERRO: Valor invalido.\n" );
}
else
{
    // definir armazenador
    int array [ n ];
    // ler dados
    freadIntArray ( "ARRAY1.TXT", n, array );
    // mostrar dados
    IO_printf ( "\nOriginal\n" );
    printIntArray ( n, array );

    // mostrar resultados de procuras
    value = array [ 0 ];
    IO_printf ( "\nProcurar por (%d) = %d\n",
        value, searchArray ( value, n, array ) );

    value = array [ n / 2 ];
    IO_printf ( "\nProcurar por (%d) = %d\n",
        value, searchArray ( value, n, array ) );

    value = array [ n - 1 ];
    IO_printf ( "\nProcurar por (%d) = %d\n",
        value, searchArray ( value, n, array ) );

    value = (-1);
    IO_printf ( "\nProcurar por (%d) = %d\n",
        value, searchArray ( value, n, array ) );
} // end if

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

```

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 **while**, exceto as que tiverem que percorrer todos os dados, e deverão ser usadas expressões lógicas para interrompê-las.
- Testes deverão usar expressões lógicas e usar funções próprias em lugar das disponíveis em bibliotecas nativas.

01.) Incluir um método (0811) para

ler o tamanho de um arranjo para inteiros do teclado,

bem como todos os seus elementos, garantindo que só tenha valores positivos e ímpares.

Verificar se o tamanho (ou dimensão) não é nulo ou negativo.

Para testar, ler diferentes quantidades de dados.

DICA: Retornar no parâmetro (n) a quantidade lida.

Exemplo: lerImpares (&n, arranjo);

02.) Incluir um método (0812) para

ler um arranjo com inteiros de arquivo.

Valores negativos e também os pares e os negativos deverão ser descartados.

O arranjo e o nome do arquivo serão dados como parâmetros.

Guardar, em arquivo primeiro o tamanho, depois os elementos, um dado por linha.

Para testar, ler diferentes nomes e quantidade de dados.

Exemplo: gravarImparesPositivos (n, arranjo);

03.) Incluir um método e uma função (0813) 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 aberto 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 (usar **mod=%**).

Exemplo: valor = gerarInt (inferior, superior);

- 04.) Incluir um método e uma função (0814) para procurar o menor valor ímpar em um arranjo.
Para testar, receber um nome de arquivo como parâmetro e aplicar a função sobre o arranjo com os valores lidos;
DICA: Procurar o primeiro valor ímpar da tabela como referência inicial para o menor valor, se houver.

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

- 05.) Incluir um método e uma função (0815) para procurar o maior valor ímpar, múltiplo de 3 e não de 5, 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 ímpar múltiplo de 3 e não de 5 na tabela, se houver, como referência inicial.

Exemplo: arranjo = lerArquivo (n, "DADOS.TXT");
maior = acharMaiorImparDivisivelPorTres (n, arranjo);

- 06.) Incluir um método e uma função (0816) para determinar a média valores em um arranjo.
Para testar, ler o arquivo ("DADOS.TXT") armazenar os dados em um arranjo;
separar em dois outros arquivos, os valores menores ou iguais à média, e os maiores que ela.

Exemplo: arranjo = lerArquivo (n, "DADOS.TXT");
media = acharMedia (n, arranjo);

DICA: Considerar a possibilidade de que a quantidade de dados possa ser igual a zero.

- 07.) Incluir um método e uma função (0817) para receber como parâmetro um arranjo e 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 !

08.) Incluir um método e uma função (0818) para procurar por determinado valor em arranjo e dizer se esse valor pode ser nele encontrado, indicada a posição inicial para se começar a procura. 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, a partir da posição indicada.

Exemplo: arranjo = lerArquivo (n, "DADOS.TXT");
 existe = acharValor (procurado, 0, n, arranjo);

09.) Incluir um método e uma função (0819) para procurar por determinado valor em arranjo e dizer onde se encontra esse valor, indicada a posição inicial para começar a procura. Para testar, ler o arquivo ("DADOS.TXT"), e armazenar os dados em arranjo; ler do teclado um valor inteiro para ser procurado; determinar a primeira posição onde esse valor ocorrer no arranjo, se houver.

Exemplo: arranjo = lerArquivo (n, "DADOS.TXT");
 onde = acharPosicao (procurado, 0, n, arranjo);

10.) Incluir um método e uma função (0820) para procurar por determinado valor em arranjo e dizer quantas vezes esse valor pode ser encontrado, indicada a posição inicial para começar a procura. Para testar, ler o arquivo ("DADOS.TXT"), e armazenar os dados em arranjo; ler do teclado um valor inteiro para ser procurado; determinar quantas vezes o valor procurado aparece no arranjo, se ocorrer.

Exemplo: arranjo = lerArquivo (n, "DADOS.TXT");
 vezes = acharQuantos (procurado, 0, n, arranjo);

Tarefas extras

- E1.) Incluir um método e uma função (08E1) para ler um valor inteiro do teclado, e mediante funções para calcular e retornar a quantidade e seus divisores ímpares guardados em arranjo, o qual deverá ser mostrado na tela após o retorno, bem como gravado em arquivo ("DIVISORES.TXT").
DICA: Supor que a quantidade de divisores será, no máximo, igual ao próprio número.
- E2.) Incluir um método e uma função (08E1) para ler um arquivo ("PALAVRAS.TXT"), e mediante uma função retornar as dez primeiras palavras que não comecem ou terminem com a letra 'D' (ou 'd'), se houver. As palavras encontradas deverão ser exibidas na tela, após retorno.
DICA: Supor que a quantidade de palavras não ultrapassará 100.