

## Aula 9

**Objectivos:** (Continuação da aula anterior.)

### Problema A1

Continuando o problema das datas das aulas anteriores, implemente os operadores de relação de ordem (<, >, <= e >=) para as datas.

### Problema A2

Continuando o problema 1 da aula 2 (e reutilizando as classes `DATA`), acrescente a informação de data de nascimento à classe `PESSOAS`.

**A2.1.** Defina o operador `[]` na classe `LISTA_PESSOAS` de forma a que o índice 1 corresponda à primeira pessoa da lista, o 2 à segunda, etc.

(**Nota:** Construa este método de forma a garantir correcção na sua utilização. Ou seja, o método só deve ser aplicável com índices válidos:  $i \geq 1$  and  $i \leq num\_pessoas$ .)

**A2.2.** Acrescente o método `ordena` à classe `LISTA_PESSOAS`. Por defeito, este método deve ordenar a lista pela data de nascimento das pessoas, utilizando o algoritmo de ordenação sequencial.

**A2.3.** Estude as possibilidades de se poder escolher quer o critério de ordenação (por nome, por idade, etc.), quer o algoritmo de ordenação (sequencial, bolha, etc.).

## Problema B1

**B1.1.** Crie uma classe abstracta pura `CFigura` que contenha os métodos `double Area()`, `double Perimetro()`. Faça com que a classe `CFigura` possua o destrutor puro virtual. Tente criar na função `main` alguma instância da classe `CFigura`. O que acontece?

**B1.2.** Construa as classes `CCirculo`, `CRectangulo`, e `CQuadrado`. Pense qual tem que ser a hierarquia das classes e que membros estas devem incluir. Implemente os métodos `Area` e `Perimetro`. Para calcular a área e o perímetro dos círculos, precisará do valor de  $PI = 3.14159$ . Este valor é uma constante partilhada por todos os objectos da classe, portanto declare-o como `static const double PI`; Faça com que se saiba quantas instâncias de classes `CCirculo`, `CRectangulo` e `CQuadrado` existem (implemente para o efeito a função estática `Count`). Redefina o operador `>` para comparar duas figuras (com base na sua área). Pense que funções podem e devem ser declaradas como constantes.

**B1.3.** Implemente duas funções globais cada uma das quais recebe como argumento um array de ponteiros para `CFigura` e um valor `unsigned` que indica o número de ponteiros. A função `DisplayAreas` deve imprimir a área de cada figura e a função `DisplayPerimeters` visualiza o perímetro de cada figura. Teste as suas classes e funções com a seguinte função `main`:

```
int main()
{
    {
        CCirculo c1(5); CRectangulo r1(2, 3); CQuadrado q1(4);

        CFigura* pContainer1[] = { &c1, &r1, &q1 } ;
        CFigura::Count();

        CCirculo c2(15); CRectangulo r2(3, 5);
        CFigura* pContainer2[] = { &c2, &r2 } ;
        CFigura::Count();

        DisplayAreas(pContainer1, sizeof(pContainer1) / sizeof(*pContainer1));
        DisplayAreas(pContainer2, sizeof(pContainer2) / sizeof(*pContainer2));

        cout << (*pContainer1[0] > *pContainer1[1]) << endl;
        cout << (*pContainer1[1] > *pContainer1[2]) << endl;
        DisplayPerimeters(pContainer1, sizeof(pContainer1)/sizeof(*pContainer1));
        DisplayPerimeters(pContainer2, sizeof(pContainer2)/sizeof(*pContainer2));
    }
}
```

```

}

CFigura::Count();

return 0;
}

```

## Problema B2

**B2.1.** Crie uma classe `CVector` que representa um vector de inteiros. O tamanho máximo do vector é desconhecido pelo que a memória deve ser reservada dinamicamente.

O construtor deve ter a forma `CVector(unsigned e1)`; onde `e1` é o número de elementos no vector; o vector deve ser preenchido com zeros. Redefina o operador `[]` que possibilita a leitura e escrita de elementos individuais.

**B2.2.** Construa a classe `CMatrix`. Esta classe é composta por um conjunto de vectores que representam as linhas da matriz. Para além disso a matriz possui um nome representado por um objecto da classe `string`: `m_sNome`. O construtor da classe deve receber três argumentos (o nome, o número de linhas e o número de colunas) e preencher a matriz com zeros. Redefina o operador `*` (multiplicação). Este é definido da seguinte maneira:

Para as matrizes  $A(m \times n)$  e  $B(n \times l)$  o resultado da multiplicação é uma matriz  $X(m \times l)$  cujos elementos são:

$$x_i^j = \sum_{k=0}^{k < n} a_i^k * b_k^j, \quad i = 1, \dots, m; \quad j = 1, \dots, l$$

**B2.3.** Construa a classe `CBinaryMatrix` que representa uma matriz booleana. Redefina o operador `*` (multiplicação). Este é definido da forma seguinte:

Para as matrizes booleanas  $A(m \times n)$  e  $B(n \times l)$  o resultado da multiplicação é uma matriz booleana  $X(m \times l)$  cujos elementos são:

$$x_i^j = \bigvee_{k=0}^{k < n} a_i^k \wedge b_k^j, \quad i = 1, \dots, m; \quad j = 1, \dots, l$$

**B2.4.** Redefina os operadores `=` para ambas as classes. O operador `=` em vez de simplesmente copiar o nome da matriz, deve acrescentar que a matriz

corrente foi actualizada a partir de outra: “*nome1 actualizada da nome2*”. Na implementação do operador \* também componha o nome da matriz nova de maneira seguinte: “*nome1 multiplicada por nome2*”. Pense que funções podem e devem ser declaradas como constantes.

**B2.5.** Implemente a seguinte função main:

```
int main()
{
    using namespace std;

    unsigned r1 = 3, c1 = 2;
    CMatrix m1("m1", r1, c1);
    m1[0][0] = 2;  m1[0][1] = 3;
    m1[1][0] = 4;  m1[1][1] = 5;
    m1[2][0] = 1;  m1[2][1] = 4;

    unsigned r2 = 2, c2 = 4;
    CMatrix m2("m2", r2, c2);
    m2[0][0] = 1;  m2[0][1] = 2;  m2[0][2] = 3;  m2[0][3] = 4;
    m2[1][0] = 4;  m2[1][1] = 3;  m2[1][2] = 2;  m2[1][3] = 1;

    cout << m1 << endl;
    cout << m2 << endl;
    cout << m1 * m2 << endl;

    CBinaryMatrix mb1("mb1", r1, c1);
    mb1[0][0] = 1;  mb1[1][1] = 1;
    mb1[2][0] = 1;  mb1[2][1] = 1;

    CBinaryMatrix mb2("mb2", r2, c2);
    mb2[0][2] = 1;  mb2[0][3] = 1;
    mb2[1][1] = 1;  mb2[1][3] = 1;

    cout << mb1 << endl;
    cout << mb2 << endl;
    cout << mb1 * mb2 << endl;

    cout << (mb1 = mb2) << endl;

    CMatrix* mm1 = &mb1;
    CMatrix* mm2 = &mb2;
    *mm1 = *mm2;

    return 0;
}
```

**B2.6.** Implemente as funções necessárias para que a função `main` execute correctamente.

**B2.7.** Os resultados devem ser os seguintes:

```
m1:   2   3
      4   5
      1   4
```

```
m2:   1   2   3   4
      4   3   2   1
```

```
m1 multiplicada por m2:  14  13  12  11
                       24  23  22  21
                       17  14  11   8
```

```
mb1:   1   0
       0   1
       1   1
```

```
mb2:   0   0   1   1
       0   1   0   1
```

```
mb1 multiplicada por mb2:  0   0   1   1
                          0   1   0   1
                          0   1   1   1
```

```
mb1 actualizada da mb2:  0   0   1   1
                          0   1   0   1
```

## Problema B3

Indique, em cada uma das afirmações seguintes, se elas são verdadeiras ou falsas.

**B3.1.** A implementação do construtor de cópia seguinte para uma classe `type` que contém dois membros: o número de elementos `nElements` e o `array` de inteiros `array_Elements`, está correcta:

```
type::type(const type& r)
{
    delete [] array_Elements;
```

```
nElements = r.nElements;
array_Elements = new int[nElements];
for (unsigned i = 0; i < nElements; i++)
    array_Elements[i] = r.array_Elements[i];
}
```

**B3.2.** A implementação do operador = para a mesma classe `type` está correcta:

```
type& type::operator=(const type& rv)
{
    nElements = rv.nElements;
    array_Elements = new int[nElements];
    for (unsigned i = 0; i < nElements; i++)
        array_Elements [i] = rv. array_Elements [i];
    return *this;
}
```

**B3.3.** A função `operator X()` definida na classe `Y` sem a especificação do valor de retorno, permite a conversão automática de `Y` para `X`.