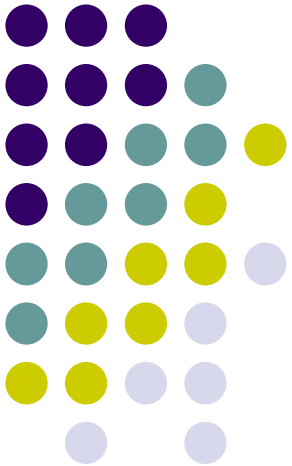
Algoritmos e Estruturas de Dados I

Vetores

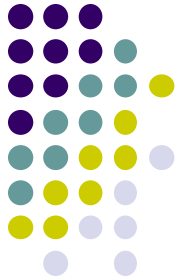
Profa. Márcia Cristina Moraes

Profa. Milene Selbach Silveira



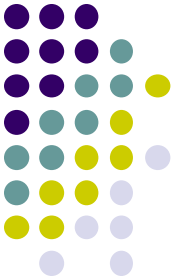
Material para estudo:
Forbellone, A. e Eberspächer, H. (2005)
➔ capítulo 4 (conceitos de variáveis compostas homogêneas e EXERCÍCIOS)

Variável SIMPLES Homogênea



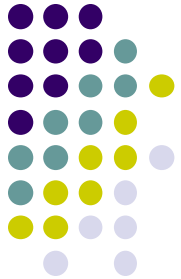
- Corresponde a uma posição de memória, identificada por um único nome e cujo conteúdo é de um único tipo.
- Por exemplo:
  - Inteiro: Idade, NroAlunos
  - Real: Nota1, Peso, Altura
  - Literal: Nome

Variável COMPOSTA Homogênea



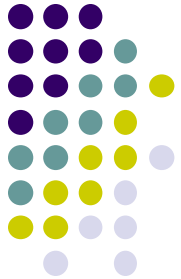
- Correspondem a posições de memória, identificadas por um único nome, individualizadas através de índices e cujo conteúdo é de um mesmo tipo.

Variáveis Compostas Homogêneas



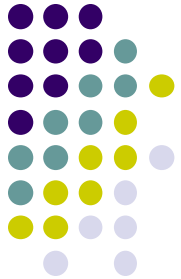
- O nome de uma variável composta é um identificador que obedece as mesmas regras de formação de identificadores de variáveis simples.
- Este nome refere-se a todos os elementos da variável composta.
- Para referência a um elemento é necessário colocar o nome da variável seguido de um ou mais índices entre colchetes.

Variáveis Compostas Homogêneas



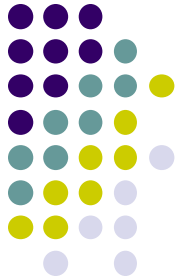
- Este tipo de variável consiste em localizações contínuas de memória. O menor endereço corresponde ao primeiro elemento e o maior endereço ao último.
- Estas variáveis podem ter uma dimensão (conhecidas como Vetores) ou muitas dimensões (conhecidas como Matrizes).

Variáveis Compostas Unidimensionais - Vetores



- Variáveis compostas unidimensionais dão chamadas de vetores.
- Elas são utilizadas para armazenar um conjunto de dados cujos elementos podem ser endereçados por um único índice.
- Podemos ter vetores de inteiros, reais, literais.
- Vetores são matrizes de uma única dimensão.

Variáveis Compostas Unidimensionais - Vetores



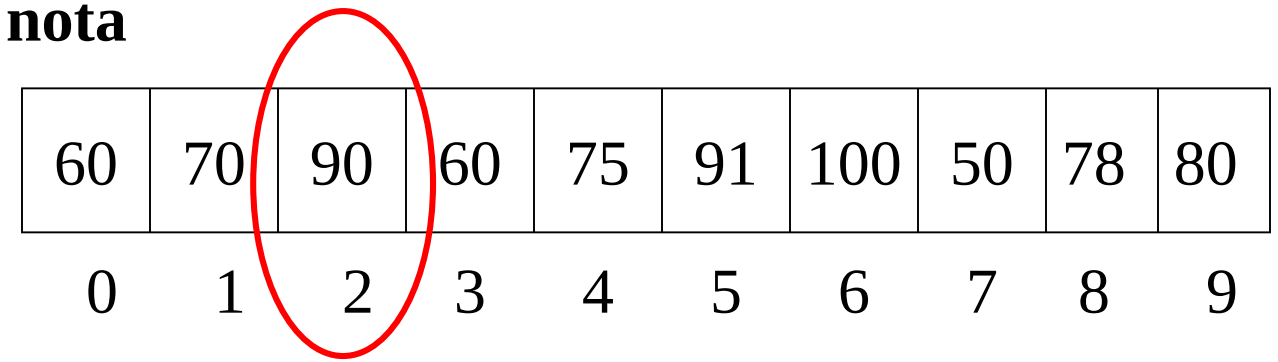
- Supondo que as notas de 10 alunos estejam armazenadas em uma variável composta identificada por nota, o vetor teria a seguinte representação:

nota

60	70	90	60	75	91	100	50	78	80
0	1	2	3	4	5	6	7	8	9

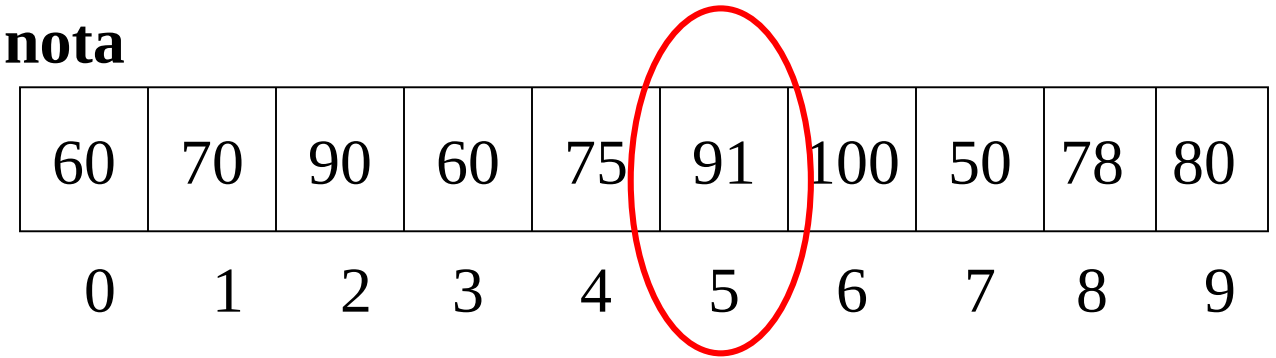
Variáveis Compostas Unidimensionais - Vetores

- Para referenciar o terceiro elemento desta variável seria:
  - nota[2]
  - o conteúdo armazenado nesta posição é 90 e o índice é a constante inteira 2, pois os índices dos vetores começam a contar a partir de 0.



Variáveis Compostas Unidimensionais - Vetores

- Utilizando-se a variável **i** como índice de nota pode-se ter acesso a qualquer uma das notas armazenadas, através da notação
  - nota[i]
  - seja o valor de i igual a 5 em um determinado instante, nota[i] seria substituída por nota[5], cujo valor é 91

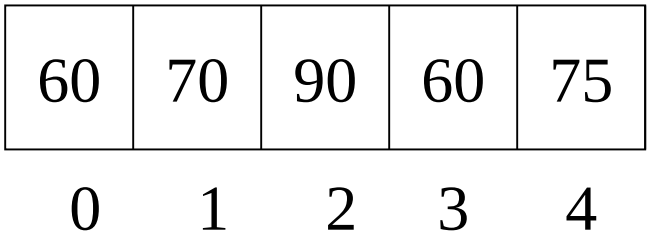


Variáveis Compostas Unidimensionais - Vetores

- São utilizadas para armazenar conjuntos de dados cujos elementos podem ser endereçados por um único índice.
- Exemplo de declaração:

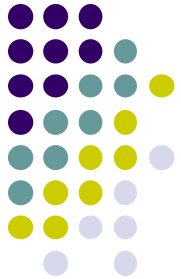
inteiro: vet[5] ⟷ **Vetor de inteiros de 5 posições de nome vet (índice do vetor sempre inicia em 0)**

vet



vet[0] ← 60
vet[1] ← 70
vet[2] ← 90
vet[3] ← 60
vet[4] ← 75

Exemplo 1



- Fazer um algoritmo que declara um vetor vet com 6 elementos do tipo inteiro, inicializa vet com o valor 2 e depois escreve o seu conteúdo.

```
Algoritmo MeuVetor
inteiro: vet[6]
Início
    vet[0] ← 2
    vet[1] ← 2
    vet[2] ← 2
    vet[3] ← 2
    vet[4] ← 2
    vet[5] ← 2
    Escreva(vet[0])
    Escreva(vet[1])
    Escreva(vet[2])
    Escreva(vet[3])
    Escreva(vet[4])
    Escreva(vet[5])
Fim
```

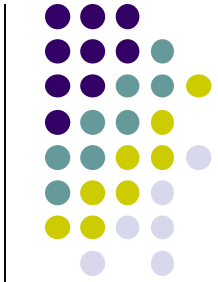
Exemplo 1

- Fazer um algoritmo que declara um vetor vet com 6 elementos do tipo inteiro, inicializa vet com o valor 2 e depois escreve o seu conteúdo.

```
Algoritmo MeuVetor
inteiro: vet[6]
Início
    vet[0] ← 2
    vet[1] ← 2
    vet[2] ← 2
    vet[3] ← 2
    vet[4] ← 2
    vet[5] ← 2
    Escreva(vet[0])
    Escreva(vet[1])
    Escreva(vet[2])
    Escreva(vet[3])
    Escreva(vet[4])
    Escreva(vet[5])
Fim
```

O que temos aqui?

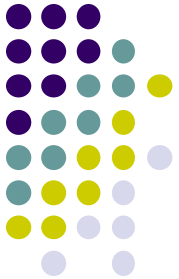
Exemplo 1 – com repetição



- Fazer um algoritmo que declara um vetor vet com 6 elementos do tipo inteiro, inicializa vet com o valor 2 e depois escreve o seu conteúdo.

```
Algoritmo MeuVetor
inteiro: vet[6], i
Início
    Para i de 0 até 5
     Início
       vet[i] ← 2
     Fim_para_i
    Para i de 0 até 5
     Início
       Escreva(vet[i])
     Fim_para_i
Fim
```

Exemplo 1 – com repetição



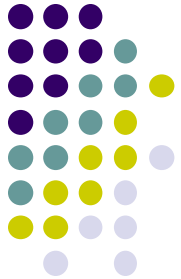
- Fazer um algoritmo que declara um vetor vet com 6 elementos do tipo inteiro, inicializa vet com o valor 2 e depois escreve o seu conteúdo.

```
Algoritmo MeuVetor
inteiro: vet[6], i
Início
    Para i de 0 até 5
     Início
       vet[i] ← 2
     Fim_para_i
    Para i de 0 até 5
     Início
       Escreva(vet[i])
     Fim_para_i
Fim
```

ou

```
Para i de 0 até 5
    Início
       vet[i] ← 2
       Escreva(vet[i])
    Fim_para_i
```

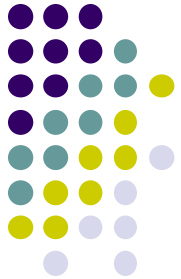
Exemplo 2



- Fazer um algoritmo que leia 5 valores numéricos inteiros, os armazene em um vetor e os imprima.

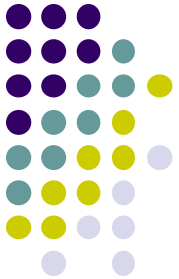
```
Algoritmo MeuOutroVetor
Inteiro: vetor[5], i
Início
    Para i de 0 até 4
    Início
        Leia(vetor[i])
        Escreva(vetor[i])
    Fim_para_i
Fim
```

Exercício



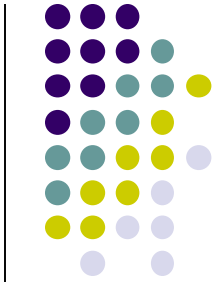
- Fazer um algoritmo que leia 100 valores numéricos inteiros e armazene-os em um vetor. Após, verifique – dentre estes valores - se existem valores iguais a 30. Se existirem, escrever as posições em que estes valores estão armazenados.

Passando Vetores para Funções e Procedimentos



- Podemos passar vetores para funções e procedimentos por:
  - Valor
  - Referência

Passagem de Vetores por Valor – entendendo a lógica...

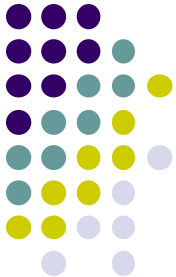


```
Procedimento Inicializa_vetor(inteiro: vet[6])
Início
  inteiro: indice
  Para indice de 0 até 5
     Início
        vet[indice] ← 1
        Escreva(vet[indice])
     Fim_para_indice
  Fim
```

O que significa passar **vet[6]** por **valor**?

Que significado/impacto isto terá no algoritmo principal?

Passagem de Vetores por Valor – entendendo a lógica...



```
Procedimento Inicializa_vetor(inteiro: vet[6])
Início
  inteiro: indice
  Para indice de 0 até 5
     Início
        vet[indice] ← 1
        Escreva(vet[indice])
     Fim_para_indice
  Fim
```

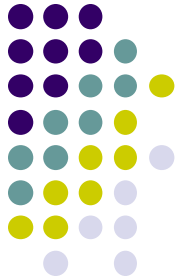
O que acontece com vetor quando ele é enviado para o procedimento?

A alteração que ele sofre dentro do procedimento se reflete no seu valor no algoritmo principal?

```
Algoritmo Principal
inteiro: i, vetor[6]
Início
   Para i de 0 até 5
     Início
       vetor[i] ← i * 5
     Fim_para_i
   Inicializa_vetor(vetor)
   Para i de 0 até 5
     Início
         Escreva(vetor[i])
      Fim_para_i
   Fim
```

Só o nome

Passagem de Vetores por Referência – entendendo a lógica...

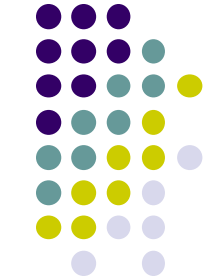


```
Procedimento Inicializa_vetor(inteiro: ref vet[6])
Início
  inteiro: indice
  Para indice de 0 até 5
     Início
        vet[indice] ← 1
        Escreva(vet[indice])
     Fim_para_indice
  Fim
```

O que significa passar **vet[6]** por referência?

Que significado/impacto isto terá no algoritmo principal?

Passagem de Vetores por Referência – entendendo a lógica...



```
Procedimento Inicializa_vetor(inteiro: ref vet[6])
Início
  inteiro: indice
  Para indice de 0 até 5
     Início
        vet[indice] ← 1
        Escreva(vet[indice])
     Fim_para_indice
  Fim
```

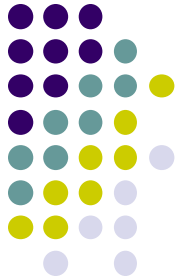
O que acontece com vetor quando ele é enviado para o procedimento?

A alteração que ele sofre dentro do procedimento se reflete no seu valor no algoritmo principal?

```
Algoritmo Principal
inteiro: i, vetor[6]
Início
   Para i de 0 até 5
     Início
       vetor[i] ← i * 5
     Fim_para_i
   Inicializa_vetor(vetor)
   Para i de 0 até 5
     Início
         Escreva(vetor[i])
      Fim_para_i
   Fim
```

Só o nome

Passagem de Vetores – Procedimentos Genéricos



```
Procedimento le_vetor(inteiro ref vet[ ], tam)
Início
  inteiro: indice
  Para indice de 0 até tam-1
     Início
        leia(vet[indice])
     Fim_para_indice
Fim
```

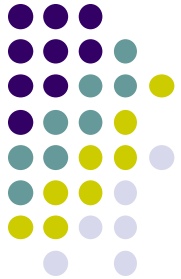
```
Procedimento escreve_vetor(inteiro vet[ ], tam)
Início
 inteiro: indice
  Para indice de 0 até tam-1
     Início
        escreva(vet[indice])
     Fim_para_indice
Fim
```

```
Algoritmo Principal
inteiro: vetor[6]
Início
   le_vetor(vetor,6)
   escreve_vetor(vetor,6)
Fim
```

Por quê fazer **genérico**?

Relembrem... Qual a principal vantagem de usar um subalgoritmo (função ou procedimento)?

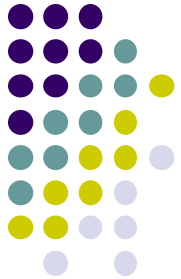
Considerações



- Quando passamos um vetor por valor estamos passando uma cópia do vetor original para o parâmetro que recebe o vetor.

- Quando passamos um vetor por referência estamos passando um apontador para o endereço de memória do vetor. Se faz um endereçamento de memória direto, ou seja, seria como se estivéssemos passado o próprio vetor que está no algoritmo principal.

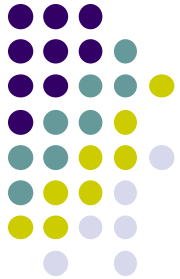
Considerações



- Uma função **nunca pode retornar um vetor através do seu nome**, pois uma função retorna **um único valor** e um vetor é uma variável composta, ou seja, armazena **mais de um valor**. Deste modo ele não pode ser retornado por uma função.

- Vetores somente podem ser “retornados” quando são passados por **referência**.

Exercício

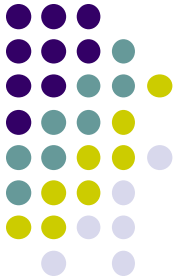


- Faça um procedimento que recebe um vetor de inteiros, o seu tamanho e um número inteiro e inicializa o vetor com a tabuada do número (este vetor deve voltar modificado ao algoritmo que o chamou). Por exemplo, **se** o número for 6 e o vetor de inteiros tiver tamanho 6 o vetor ficaria inicializado como:

0	6	12	18	24	30
0	1	2	3	4	5

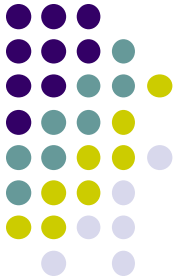
No algoritmo principal faça a criação de um vetor de inteiros de tamanho 6, leia o número inteiro (para o qual será criado o vetor com a tabuada) e chame o procedimento criado neste exercício. Ao final, chame o procedimento definido em aula para escrever o vetor resultante.

Algoritmos de Ordenação de Vetores



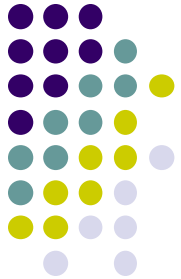
- Veremos dois tipos de algoritmos de ordenação de vetores:
  - Método da bolha
  - Método do maior ou menor elemento do conjunto

Método da Bolha



- A estratégia de ordenação é a comparação entre pares de elementos adjacentes.
- Os pares são trocados quando estiverem fora de ordem.
- Repete-se o algoritmo até que todos estejam ordenados.

Método da Bolha



- Supondo a leitura de um vetor V [6] com os seguintes valores:

15	7	1	4	17	10
0	1	2	3	4	5

- Para fazer um algoritmo Bolha Crescente, se compara os pares adjacentes e troca quando o anterior for maior do que o próximo.

Método Bolha Crescente

- Trecho do algoritmo BolhaCrescente

Por quê 4 e não 5 se é um vetor de 6 posições?

```
Para i de 0 até 4
    se v[i] > v[i+1] então
        Início
            x ← v[i]
            v[i] ← v[i+1]
            v[i+1] ← x
        Fim
Fim_para_i
```

Problema:
Só faz uma passagem no vetor.

Precisa percorrer o vetor até garantir que ele esteja completamente ordenado.

Para ordenar sempre são necessários dois laços de repetição!!

Método Bolha Crescente – versão 1 – Procedimento

```
Procedimento BolhaCrescente(inteiro: ref v[], tam)
Inicio
 inteiro: n, x, i
laço 1 ➔ Para n de 0 até tam-2
         Início
laço 2 ➔  Para i de 0 até tam-2
           Inicio
              se v[i] > v[i+1] então
                Início
                    x ← v[i]
                    v[i] ← v[i+1]
                    v[i+1] ← x
                Fim
           Fim_para_i
    Fim_para_n
Fim
```

Problema:
Nem sempre é necessário repetir – o trecho da ordenação - quantas vezes for o tamanho do vetor.

Método Bolha Crescente – versão 2 - Procedimento

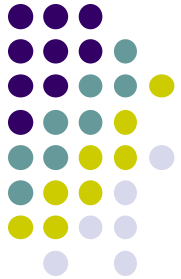
```
Procedimento BolhaCrescente(inteiro: ref v[], tam)
Inicio
Inteiro: i, x, k
k ← 0
laço 1 ➔ Enquanto (k == 0)
          Início
           k ← 1
laço 2 ➔   Para i de 0 até tam-2
            Inicio
             se v[i] > v[i+1] então
               Início
                  x ← v[i]
                  v[i] ← v[i+1]
                  v[i+1] ← x
                  k ← 0
               Fim
          Fim_para_i
Fim_Enquanto
Fim
```

Precisou trocar? Pode ser que algo tenha se perdido pela frente... Repete outra vez!!!

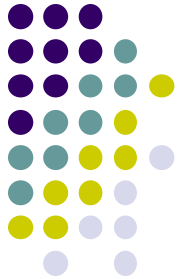
Esta versão passa somente o número de vezes necessárias para ordenação do vetor.

Exercício

- Faça o procedimento correspondente ao do método Bolha Decrescente.

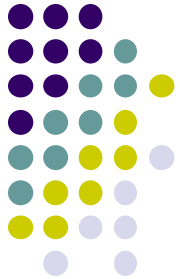


Método do Menor Elemento do Conjunto



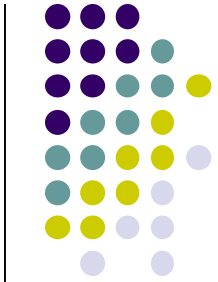
- Ordena em ordem crescente de acordo com a seguinte estratégia:
  - Seleciona o elemento do vetor que apresenta o menor valor.
  - Troque este elemento pelo primeiro.
  - Repita esta operação, agora envolvendo os 5 elementos restantes (trocando o de menor valor com o da segunda posição), e assim por diante.

Método Troca Menor Elemento



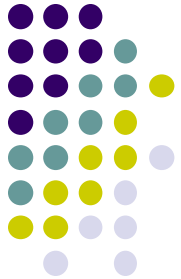
```
Procedimento TrocaMenor(inteiro: ref v[], tam)
Inicio
inteiro: n, menor, p, i, aux
laço 1 ➔ Para n de 0 até tam-2
  Início
    menor ← v[n]
    p ← n
laço 2 ➔ Para i de n+1 até tam-1
      Início
        se menor > v[i] então
                              Início
                                menor ← v[i]
                                p ← i
                              Fim
    Fim_para_i
    aux ← v[n]
    v[n] ← v[p]
    v[p] ← aux
  Fim_para_n
Fim
```

Exercício



- Faça o procedimento que ordena em ordem decrescente de acordo com a seguinte estratégia:
  - Seleciona o elemento do vetor que apresenta o maior valor.
  - Troque este elemento pelo primeiro.
  - Repita esta operação, agora envolvendo os 5 elementos restantes (trocando o de maior valor com o da segunda posição), e assim por diante.

Considerações



- Método da Bolha e do Maior ou Menor valor servem para casos onde se tem um conjunto **pequeno** de valores a ser ordenado.
- Para conjuntos grandes de valores deve-se utilizar métodos mais eficientes como QuickSort, MergeSort, etc, que serão vistos posteriormente no curso!

Bibliografia

- Orth, Afonso Inácio. Algoritmos e Programação. Editora AIO. 2001.
- **Forbellone, A. e Eberspacher, H. Lógica de Programação: A Construção de Algoritmos e Estruturas de Dados. Makron Books, São Paulo, 3ª edição. 2005.**