

# Algoritmos e Lógica de Programação

Profª. Eliane Oliveira Santiago



# Vetores

# Estrutura de Dados Homogênea

Quando desenvolvemos um algoritmo ou programa estruturado, muitas vezes, precisamos usar várias informações de um mesmo tipo de dado. Podemos fazer isso criando inúmeras variáveis, uma para cada informação, mas isso é inviável, pois é demasiado complexo desenvolvermos e manipularmos um algoritmo que utiliza inúmeras variáveis para várias informações de um mesmo tipo. Para solucionar este problema, podemos utilizar vetores e matrizes para armazenar essas informações, isto é, uma **estrutura de dados homogênea**.

# Variável simples

Var

x : inteiro

x 

10
----

Início

x  $\leftarrow$  10

Guarda um único valor de  
um único tipo

# Estrutura de dados homogênea

Uma estrutura de dados homogênea é uma estrutura capaz de armazenar várias informações de um mesmo tipo de dado. Assim, com um único nome declarado para esta estrutura, podemos manipular várias informações.

Temos dois tipos de estrutura de dados homogênea: **vetores** e **matrizes**. No capítulo seguinte, veremos matrizes e a seguir veremos como e quando trabalhar com um vetor, bem como, declarar e manipular um vetor.

# Variável composta homogênea

Var

x[0..9] : inteiro

Início

~~x ← -10~~

x[0] ← 10

x[3] ← 20

	0	1	2	3	4	5	6	7	8	9
x	10			20						

Guarda vários valores de um único tipo

Sintaxe da declaração no Visual G

Var

x : **vetor**[0..9] de inteiro

# Variável composta homogênea

Var

x[0..25] : caracter

Início

x[0] ← 'A'

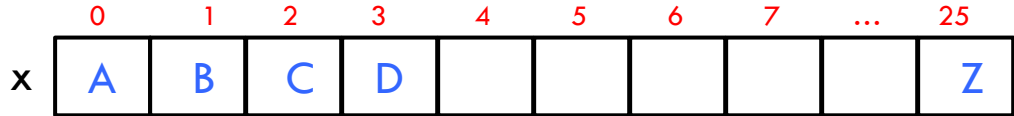
x[1] ← 'B'

x[2] ← 'C'

:

:

x[25] ← 'Z'



Sintaxe da declaração no Visual G

Var

x : **vetor**[0..25] **de caracter**

# Variável composta homogênea

Var

x : vetor[0..25] de char

Início

x[0] ← 'A'

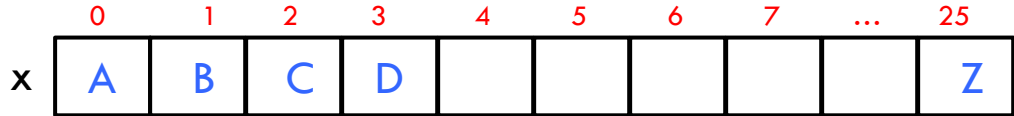
x[1] ← 'B'

x[2] ← 'C'

:

:

x[25] ← 'Z'





# Vetores

**Um vetor é um conjunto de informações de um mesmo tipo de dado.** Note que vetor é um nome particular dado à matriz unidimensional. Por exemplo, considere uma fileira de 5 carteiras de uma sala de aula representando uma estrutura de dados e cada uma das carteiras representando partições dessa estrutura. Em outras palavras, a fileira de carteiras é um vetor e cada carteira um elemento deste vetor. Podemos representar, graficamente, este vetor da seguinte forma:

	0	1	2	3	4
Vet	carteira1	carteira2	carteira3	carteira4	carteira5

Neste exemplo, possuímos um vetor chamado **Vet**, com dimensão ou tamanho do vetor igual a 5 e com cinco posições: 0, 1, 2, 3 e 4. Na posição 0 do vetor **Vet**, temos a carteira1, na posição 1 do vetor **Vet**, temos a carteira2 e, assim por diante, até que na posição 4 do vetor **Vet**, temos a carteira5.

# Declaração de Vetores

**var:**

`notas : vetor[0..9] de real`

Vamos exemplificar, graficamente, um vetor chamado **notas**, de 10 posições, contendo notas finais de 10 alunos. Como estamos considerando notas, então determinamos que essas informações são do tipo **real**.

	0	1	2	3	4	5	6	7	8	9
notas	9.5	10.0	8.5	5	8	7.5	6	9	8.5	7

notas	0	1	2	3	4	5	6	7	8	9
	9.5	10.0	8.5	5	8	7.5	6	9	8.5	7

**notas** é o nome do vetor;

o vetor **notas** possui dez informações, por isso, tem tamanho 10, ou seja, 10 posições: de 0 até 9;

a posição 0 do vetor, notas[0], possui a nota 9.5;

a posição 1 do vetor, notas[1], possui a nota 10.0;

a posição 2 do vetor, notas[2], possui a nota 8.5;

a posição 3 do vetor, notas[3], possui a nota 5.0;

a posição 4 do vetor, notas[4], possui a nota 8.0;

a posição 5 do vetor, notas[5], possui a nota 7.5;

a posição 6 do vetor, notas[6], possui a nota 6.0;

a posição 7 do vetor, notas[7], possui a nota 9.0;

a posição 8 do vetor, notas[8], possui a nota 8.5;

a posição 9 do vetor, notas[9], possui a nota 7.0.

# Atribuição

**<nome do vetor> [<posição>] ← <valor>;**

**notas [0] ← 9.5;**

**Atribuir no momento da declaração do vetor**

**<nome vetor> [<nro dados>] ← {<vlr1> , <vlr2> , ... , <vlrn>} <tipo dados do vetor>;**

**notas[] ← {9.5, 10, 3.5} : real**

# Mostrando os elementos de um vetor

Quando mostramos valores do vetor, precisamos saber qual o tipo de dados das informações que foram armazenadas no vetor e o número de elementos desse vetor.

Os valores são mostrados para cada posição do vetor e a forma de mostrar os elementos na tela é a mesma de qualquer variável ou constante.

Sintaxe: `escreva(<nome do vetor>[<posição>])`

Exemplos: `escreva(notas[0])`

`escreva("A nota é: ", notas[0])`

# 1. Desenvolva um algoritmo que receba 25 valores inteiros e mostre esses números.

**Algoritmo** Mostrar

**var**

    i : inteiro

    vet : vetor[0..24] de inteiro //⇔ Declaração VisualG

**inicio**

**para** i **de** 0 **até** 24 **passo** 1 **faça** //processamento de dados mensagem ao usuário

**escreva**("Digite um valor inteiro")

**leia**(Vet[i]) // entrada de dados

**escreva**("Vet[", i, "] = ", Vet[i]) // saída de resultados

**fimpara**;

**fimalgoritmo**.

# Vamos somar os elementos do Vetor?

	0	1	2	3	4	5	6	7	8	9
x	10	20	30	40	50	60	70	80	90	100

Var

x: vetor [0..9] de inteiro

Inicio

Soma  $\leftarrow$  0

Soma  $\leftarrow$  soma + x[0]

Soma  $\leftarrow$  soma + x[1]

:

Soma  $\leftarrow$  soma + x[9]

## 2. Desenvolva um algoritmo que receba 100 valores numéricos inteiros e mostre a soma desses 100 números.

**Algoritmo** Somar

**var**

VetSoma[0..99] , i , soma  $\leftarrow$  0 : **inteiro**;

**Início**

**// processamento de dados**

**para** i **de** 0 **até** 99 **passo** + 1 **faça**

**escreva**("Digite um valor inteiro"); **// mensagem ao usuário**

**leia**(VetSoma[i]) **// entrada de dados**

    soma  $\leftarrow$  soma + VetSoma[i]


**fimpara**;

**escreva**("A soma dos 100 valores digitados é: ", soma);

**Fimalgoritmo.**



3. Desenvolva um algoritmo que receba 4 notas bimestrais, armazena-as em um vetor de tamanho 4, calcule e mostre a média aritmética destas 4 notas.



### 3. Desenvolva um algoritmo que receba 50 notas bimestrais, calcule e mostre a média aritmética destas 50 notas.

```
algoritmo MediaAritmetica
var
    i : inteiro;
    Notas[0..49] , media, soma ← 0 : real;

Inicio
    para i de 0 até 49 passo + 1 faça
        escreva("Digite uma nota") // mensagem ao usuário
        leia(Notas[i])              // entrada de dados
        soma ← soma + Notas[i]
    fim_para
    media ← soma/50;
    escreva("A média das 50 notas digitadas é: " , media);

Fim_algoritmo.
```

Desenvolva um algoritmo que receba 10 valores inteiros num vetor A e 10 valores inteiros num vetor B. Construa um vetor C com 10 posições, onde cada posição possua a soma dos elementos dos vetores A e B em suas respectivas posições. Mostrar os elementos dos três vetores.

[illegible]

# 4.Solução do exercício 4

**Algoritmo** SomaVetoresA\_e\_B\_em\_C

**var**

    i : inteiro;

    A[0..9] , B[0..9], C[0..9] : inteiro;

**Inicio**

**para** i de 0 até 9 **passo** + 1 **faça**

**escreva**("A[" , i, "]" = ") // mensagem ao usuário

**leia**(A[i]) // entrada de dados do vetor A

**escreva**("B[" , i, "]" = ") // mensagem ao usuário

**leia**(B[i]) // entrada de dados do vetor B

        C[i] ← A[i] + B[i]

**escreva**("C[" , i, "]" = " , C[i])

**fim\_para**

**Fim\_algoritmo.**

## 5. Façam o exercício proposto

Desenvolva um algoritmo que receba 5 valores reais num vetor X e 3 valores reais num vetor Y. Construa um vetor Z de 8 posições com a concatenação dos vetores X e Y, ou seja, os elementos das 5 primeiras posições são os mesmos elementos do vetor X e os elementos das 3 últimas posições são os mesmos do vetor Y. Mostre os elementos dos três vetores.

## Algoritmo Concatenação\_Vetores

var

i : inteiro;

X[0..49] , Y[0..29], Z[0..79] : inteiro;

Inicio

para i de 0 até 49 passo +1 faça

    escreva("X[", i, "] = ") // mensagem ao usuário

    leia(X[i]) // entrada de dados do vetor X

    Z[i] ← X[i]

    if (i<30)

        escreva("Y[", i, "] = ") // mensagem ao usuário

        leia(Y[i]) // entrada de dados do vetor Y

        Z[i+50] ← Y[i]

    fim\_se

fim\_para

para i de 0 até 79 passo +1 faça

    escreva("Z[", i, "] = ", Z[i]) // mensagem ao usuário

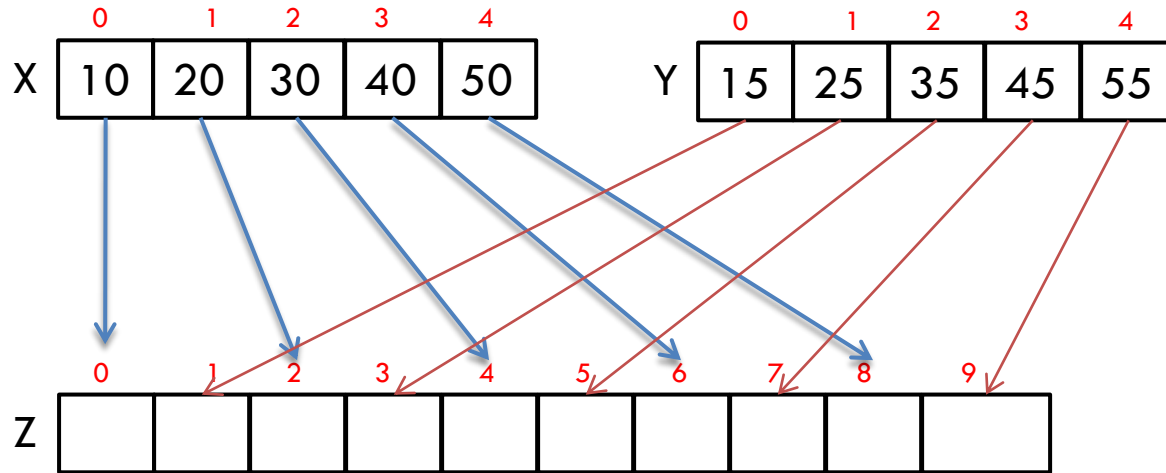
fim\_para

Fim algoritmo.

# Demais Exercícios Propostos (1)

6. Desenvolva um algoritmo que receba 10 valores numéricos inteiros num vetor Num, calcule e mostre os números primos e suas respectivas posições.
7. Desenvolva um algoritmo que receba 15 valores numéricos inteiros num vetor X, receba 15 valores numéricos inteiros num vetor Y e concatene alternadamente os elementos dos vetores X e Y num terceiro vetor Z de 30 posições. Os elementos das posições ímpares do vetor Z são os mesmos do vetor X e os elementos das posições pares do vetor Z são os mesmos do vetor Y. Mostrar os elementos dos três vetores.
8. Desenvolva um algoritmo que receba 30 valores numéricos inteiros num vetor, calcule e armazene, num segundo vetor, o quadrado dos 30 valores do primeiro vetor. Mostre os valores dos dois vetores.
9. Desenvolva um algoritmo que receba 45 valores numéricos inteiros num vetor, calcule e armazene, num segundo vetor, o fatorial de cada elemento do primeiro vetor. Mostre os dois vetores.
10. Desenvolva um algoritmo que receba 90 valores numéricos inteiros positivos num vetor, calcule e armazene, num segundo vetor, o negativo de cada elemento do primeiro vetor. Mostre os dois vetores.

© 2015 Pearson Education, Inc. or its affiliate(s). All rights reserved. Pearson Education, Inc., publishing as Pearson Benjamin Cummings, 101 Philip Drive, Assinippi Park, New York, NY 10986-1997. Printed in the United States of America. 0-321-92623-6





# Exercícios propostos (2)

11. Desenvolva um algoritmo que receba 85 valores numéricos reais num vetor, calcule e armazene, num segundo vetor, a metade de cada elemento do primeiro vetor. Mostre os dois vetores.

12. Desenvolva um algoritmo que calcule a tabuada de um número determinado pelo usuário e armazene, num vetor de 10 posições, o resultado da tabuada. Mostre os elementos do vetor.

13. Desenvolva um algoritmo que receba 15 valores referentes às temperaturas em graus Celsius e armazene num vetor de 15 posições. Calcule e armazene, num segundo vetor de 15 posições, os valores de cada temperatura em graus Celsius convertidos para graus Fahrenheit. Calcule e armazene, num terceiro vetor de 15 posições, os valores de cada temperatura em graus Celsius convertidos para graus Kelvin. Mostre os elementos dos três vetores.

## Exercícios propostos (3)

14. Desenvolva um algoritmo que receba 35 valores numéricos inteiros num vetor, calcule e armazene o triplo dos valores divisíveis por 3 num segundo vetor, armazenando inalterados os valores que não são divisíveis por 3. Mostre os elementos dos dois vetores.

# Exercícios propostos (4)

15. Desenvolva um algoritmo que receba 35 valores numéricos reais num vetor e classifique em ordem crescente os elementos desse vetor, utilizando a seguinte regra:

- selecione o menor elemento do vetor de 35 posições;
- troque este elemento pelo primeiro elemento do vetor;
- repita os dois primeiros itens, considerando agora os 34 elementos restantes do vetor, trocando o menor elemento com o segundo elemento do vetor;
- repita os dois primeiros itens, considerando agora os 33 elementos restantes do vetor, trocando o menor elemento com o terceiro elemento do vetor;
- continue até que se considere apenas o vetor com a última posição.

Mostre os elementos ordenados do vetor.

## Exercícios propostos (5)

16. Desenvolva um algoritmo que receba 100 valores numéricos inteiros num vetor. Armazene os restos das divisões dos elementos das posições pares por suas posições, num segundo vetor, e os quocientes das divisões dos elementos das posições ímpares por suas posições neste segundo vetor. Mostre os elementos dos dois vetores.

# Bibliografias

## BÁSICA

- GOMES, Ana Fernanda A. Campos, Edilene Aparecida V. Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. Prentice Hall, 2007.
- CARBONI, Irenice de Fátima. Lógica de Programação. Thomson.
- XAVIER, Gley Fabiano Cardoso. Lógica de Programação - Cd-rom. Senac São Paulo – 2007.

## COMPLEMENTAR

- FORBELLONE, André Luiz Villar. Eberspache, Henri Frederico. Lógica de Programação – A construção de Algoritmos e Estrutura de Dados. Makron Books, 2005.
- LEITE, Mário - Técnicas de Programação – Brasport - 2006.
- PAIVA, Severino – Introdução à Programação – Ed. Ciência Moderna – 2008.
- PAULA, Everaldo Antonio de. SILVA, Camila Ceccatto da. Lógica de Programação –Viena – 2007.
- CARVALHO, Fábio Romeu, ABE, Jair Minoru. Tomadas de decisão com ferramentas da lógica paraconsistente anotada: Método Paraconsistente de Decisão (MPD), Editora Edgard Blucher Ltda. - 2012.