Lógica de Programação

Aula 11 - Estruturas de Dados Homogêneas - Vetores

Apresentação da Aula

Na aula anterior, você exercitou o **aninhamento** das estruturas de repetição. Nesta aula, você aprenderá o que são as estruturas de dados homogêneas e conhecerá um dos seus tipos. Além disso, terá a oportunidade de construir algoritmos utilizando esse importante elemento para a construção de algoritmos.

Objetivos

Explorar o conceito de **estruturas de dados homogêneas**;

Conhecer a sintaxe da utilização de vetores;

Distinguir quando é conveniente utilizar vetores;

Construir algoritmos por meio de vetores.

Variáveis Primitivas - Os tipos de variáveis

Você viu, no início de nossa disciplina, que na construção de programas utiliza-se variáveis para armazenar algum dado que, geralmente, é do tipo inteiro, real, cadeia de caracteres (texto) ou booleano (verdadeiro ou falso). Viu, também, que cada uma das variáveis pode armazenar um único dado por vez. Veja o código abaixo:

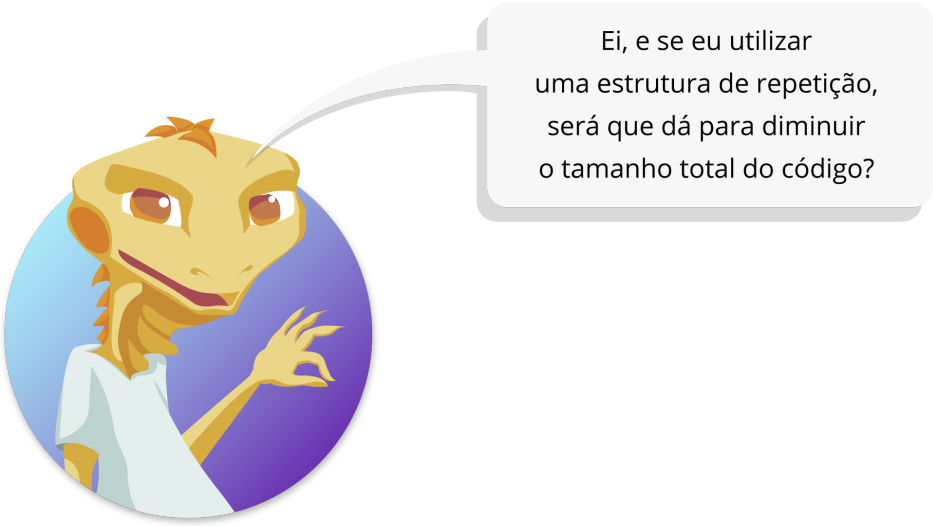
```
1  var i := 1
2  var j := 8.1
3  var nome := "Meu texto"
4  var v_ou_f := verdadeiro
5
6  escreva "O valor de i é {i} e é inteiro, pois não tem casa decimal."
7  escreva "O valor de j é {j} e é real, pois tem casa decimal."
8  escreva "Já o texto da variável nome é {nome}."
9  escreva "A variável v_ou_f tem o valor {v_ou_f}."
10
```

Assim, cada variável é capaz de receber e de armazenar um único valor por vez. No que diz respeito ao exemplo acima, se você atribuir um novo valor à variável **j**, na linha 5, o valor **8** será substituído pelo novo valor.

Agora que você já revisou rapidamente seus conhecimentos sobre as variáveis, pense no seguinte problema: "Você precisará ler 5 nomes de alunos e, em seguida, imprimir o nome desses alunos na mesma ordem que foram lidos". Qual seria uma das soluções para esse pequeno problema?

```
escreva "Digite o 1º aluno:"
var aluno1 := leia_texto
escreva "Digite o 2º aluno:"
var aluno2 := leia_texto
escreva "Digite o 3º aluno:"
var aluno3 := leia_texto
escreva "Digite o 4º aluno:"
var aluno4 := leia_texto
escreva "Digite o 5º aluno:"
var aluno5 := leia_texto

escreva "Aluno: {aluno1}"
escreva "Aluno: {aluno2}"
escreva "Aluno: {aluno3}"
escreva "Aluno: {aluno4}"
escreva "Aluno: {aluno5}"

```

Observe que a solução para o problema resultou na construção de um programa bem simples. No entanto, o código ficou bem grande, não foi mesmo? Agora, imagine que o programa deveria ler o nome de 40 alunos (o tamanho das turmas de alunos do IMD)! Certamente, o programa seria muito maior, mesmo que o algoritmo fosse simples.



Vou deixar você pensar um pouco e tentar encontrar uma solução utilizando uma das estruturas de repetição.

Atividade 01

Alunos da turma

Construa um algoritmo que leia 10 nomes de alunos e, após ler todos os nomes, imprima-os na mesma ordem que foram lidos.

Estruturas de Dados Homogêneas

Você deve ter reparado que o exercício anterior não possui uma solução trivial. E se tivéssemos uma variável que permitisse armazenar vários valores? Será que isso tornaria a solução mais fácil? Imagine uma única variável que armazena vários valores ao mesmo tempo! Dessa forma, quando fosse necessário criar um algoritmo que precisasse guardar o nome de 40 alunos (ou mais), teríamos apenas uma única variável para todos os nomes, em vez de 40 variáveis. Isso certamente tornaria tudo mais fácil, não é mesmo?

Felizmente, a maioria das linguagens de programação possuem recursos que permitem a declaração de variáveis com capacidade múltipla de armazenamento. Essas variáveis são chamadas de **estruturas de dados homogêneas**.

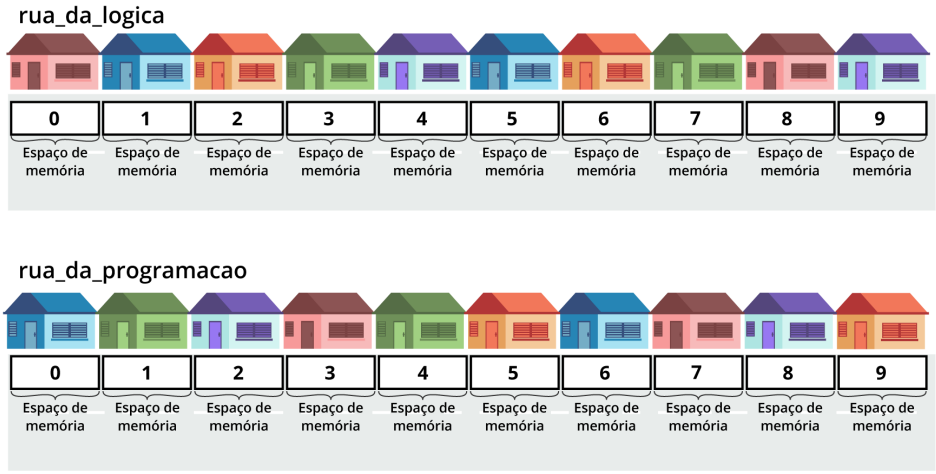
As **estruturas de dados homogêneas** permitem armazenar um conjunto de dados de um mesmo tipo (por isso o nome *homogêneo*) em uma única variável. Essas variáveis também recebem o nome de **variáveis compostas homogêneas** ou **variáveis compostas indexadas**.

Podemos dizer, por exemplo, que uma variável homogênea é como uma rua, e os espaços para armazenar os dados são como as casas dessa rua. Assim como uma rua, as estruturas de dados homogêneas possuem um nome para identificá-las. Esse

nome deve seguir as mesmas regras das variáveis primitivas que você utiliza até este momento.

Já para identificar cada um dos espaços (casas da rua) onde serão armazenados os dados, é necessário utilizar um número que recebe o nome de **índice** e que permite a identificação desses espaços. Se você pensar bem, cada casa de uma rua possui um número e este não pode se repetir nessa mesma rua. Já em outra rua (que em nosso algoritmo seria equivalente a outra variável do tipo estrutura homogênea), você poderá ter casas com os mesmos números da primeira rua.

Figura 01 - Endereços em Vetores



Você pode estar se perguntando, agora, qual a capacidade de armazenamento de uma variável homogênea. A quantidade de espaços para armazenamento é, geralmente, definida no momento da declaração da variável. E, se pensarmos bem, quando uma rua é construída, ela também tem um tamanho definido, não é verdade?

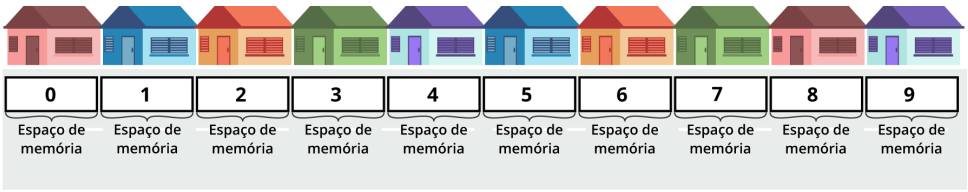
O tamanho de cada estrutura de dados homogênea é, na maioria das linguagens de programação existentes, definido no momento de se declarar a variável, mas há linguagens nas quais não é necessário determinar a quantidade de espaços iniciais. Na linguagem **Potigol** adotada em nosso material didático, é necessário definir o tamanho das variáveis homogêneas.

Classificação das Estruturas de Dados Homogêneas

As estruturas de dados homogêneas são classificadas em dois tipos: vetores (também conhecidos como *arrays*) e matrizes. Nesta aula, você conhecerá os vetores; as matrizes, por sua vez, serão estudadas nas aulas seguintes desta disciplina.

Nos vetores (*arrays*), a estrutura é do tipo **unidimensional**, isto é, elas possuem apenas uma dimensão. Poderíamos dizer que a estrutura é semelhante à de uma rua, como visto no exemplo anterior.

Figura 02 - Índices em Vetores Unidimensionais



Como na declaração das variáveis primitivas, ao se declarar uma variável do tipo vetor, no **Potigol**, não é necessário determinar previamente qual o tipo de dado que será armazenado nesse vetor. Mas lembre-se sempre que ele é **homogêneo** e portanto, somente dados do mesmo tipo do declarado na criação da variável poderão ser armazenados nesses espaços, por esse motivo, esses dados recebem o nome de **estruturas de dados homogêneas**. Essa característica não é válida para todas as linguagens de programação, algumas permitem armazenar tipos de dados diferentes (inteiro, real ou texto) em um mesmo vetor.

Veja a sintaxe, no **Potigol**, relativa à declaração da variável **alunos** do tipo vetor, cuja capacidade é para armazenar até **10** nomes.

```
1 var alunos := vetor[10]
```

A variável **alunos** criada possui **10** posições para armazenar até "10 elementos". Essa variável foi definida para armazenar os nomes de alunos. Sendo assim, vamos, agora, armazenar os nomes de **10** alunos nela.

Atenção

Na linguagem de programação que estamos utilizando, a capacidade de armazenamendo de uma variável do tipo vetor ou matriz **não** é dinâmica, ou seja, o tamanho defininido durante sua definição é fixo.

Algumas linguagens de programação exigem que seja informado o tamanho do vetor ou matriz ao se declarar esse tipo de variável. Já outras possuem tamanho dinâmico, não sendo necessário definir o tamanho (capacidade) da variável.

```
var alunos := vetor[10]

escreva "Informe o nome do aluno 1º aluno:"
alunos[0] := leia_texto
escreva "Informe o nome do aluno 2º aluno:"
alunos[1] := leia_texto
escreva "Informe o nome do aluno 3º aluno:"
alunos[2] := leia_texto
escreva "Informe o nome do aluno 4º aluno:"
alunos[3] := leia_texto
escreva "Informe o nome do aluno 5º aluno:"
alunos[4] := leia_texto
escreva "Informe o nome do aluno 6º aluno:"
alunos[5] := leia_texto
escreva "Informe o nome do aluno 7º aluno:"
alunos[6] := leia_texto
escreva "Informe o nome do aluno 8º aluno:"
alunos[7] := leia_texto
escreva "Informe o nome do aluno 9º aluno:"
alunos[8] := leia_texto
escreva "Informe o nome do aluno 10º aluno:"
alunos[9] := leia_texto

escreva "1º Aluno: {alunos[0]}"
escreva "2º Aluno: {alunos[1]}"
escreva "3º Aluno: {alunos[2]}"
escreva "4º Aluno: {alunos[3]}"
escreva "5º Aluno: {alunos[4]}"
escreva "6º Aluno: {alunos[5]}"
escreva "7º Aluno: {alunos[6]}"
escreva "8º Aluno: {alunos[7]}"
escreva "9º Aluno: {alunos[8]}"
escreva "10º Aluno: {alunos[9]}"

```

O código acima declarou o vetor **alunos** com 10 posições e, em seguida, foram atribuídos os nomes de 10 alunos, cada em uma das posições existentes no vetor. Você recorda que, assim como as casas de uma rua, os espaços devem ter endereços? Nesse caso, o vetor possui **índices**. Na maioria das linguagens de programação o primeiro espaço de um vetor é o endereço de **índice 0** (zero).

Atenção

Tentar armazenar ou acessar dados em uma posição (índice) maior que o tamanho da variável homogênea (vetor ou matriz), em algumas linguagens de programação, provoca comportamento anômalo ou erro de execução de seu programa. Portanto, fique sempre atento ao tamanho dos vetores e matrizes que você utilizar em seus algoritmos.

No código acima, foram inseridos os nomes em 10 posições do vetor **alunos**, através dos **índices** de **0** até **9** indicados logo após o nome da variável com o uso dos colchetes '**[ ]**'. Assim, se você desejar acessar a primeira posição do vetor **alunos**, deverá indicar o índice **[0]**, porém, se desejar acessar a quinta posição deverá indicar o índice **[4]**. A última posição desse vetor com 10 elementos é acessada pelo índice **[9]**.

Devido a essa necessidade de utilizar os índices nas variáveis homogêneas, para indicar qual posição irá acessar, seja para armazenar um novo dado ou obter um dado já armazenado, você sempre precisará indicar para qual posição do vetor deseja obter o acesso.

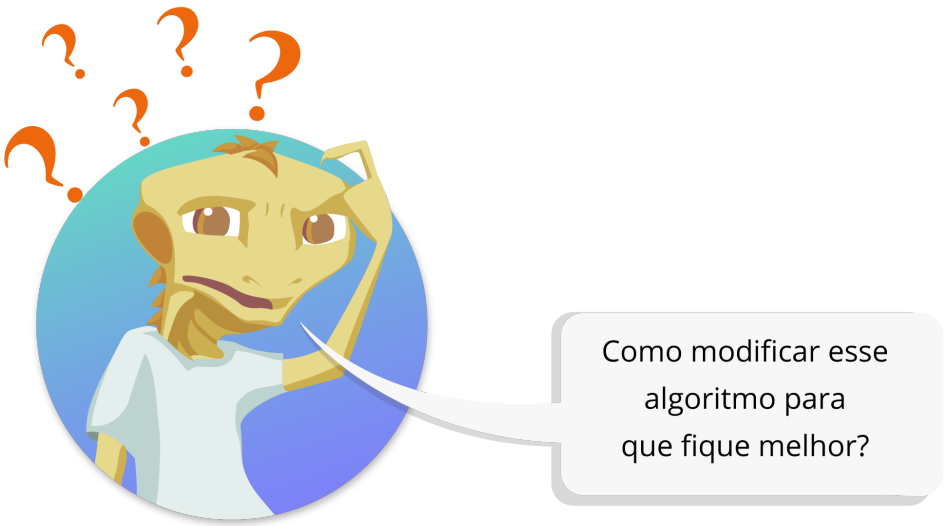
Exercício Resolvido - Sintaxe de Vetores

Você recorda do exercício realizado no início da aula? Nele, foi solicitado que você definisse um algoritmo que recebesse o nome de 10 alunos e que, em seguida, realizasse a impressão desses nomes na ordem em que foram lidos pelo algoritmo. Veja abaixo uma possível solução para esse problema, mas agora utilizando um único vetor.

```
var alunos := vetor[10]

alunos[0] := leia_texto
alunos[1] := leia_texto
alunos[2] := leia_texto
alunos[3] := leia_texto
alunos[4] := leia_texto
alunos[5] := leia_texto
alunos[6] := leia_texto
alunos[7] := leia_texto
alunos[8] := leia_texto
alunos[9] := leia_texto

escreva "{alunos[0]}"
escreva "{alunos[1]}"
escreva "{alunos[2]}"
escreva "{alunos[3]}"
escreva "{alunos[4]}"
escreva "{alunos[5]}"
escreva "{alunos[6]}"
escreva "{alunos[7]}"
escreva "{alunos[8]}"
escreva "{alunos[9]}"

```

O que achou dessa solução? Ela é bem fácil de implementar, não é verdade? Apesar disso, esse simples programa possui mais de 20 linhas de código! Por esse motivo, certamente, essa não é a melhor solução para esse exercício.

Ao analisar o código, você consegue notar algo em comum entre as linhas? Repare que há 10 linhas quase idênticas em que o comando **leia_texto** é utilizado e há mais 10 linhas em que se utiliza o comando escreva. A diferença entre elas é somente o índice que varia de uma linha para outra.



O índice que é utilizado entre os colchetes de um vetor e que indica a posição que se deseja acessar pode ser substituído por uma variável primitiva do tipo **inteiro**. Assim, você pode utilizar estruturas de repetição e simplificar a manipulação dos dados de um vetor. Veja a mesma solução apresentada acima, agora, utilizando uma estrutura com variável de controle e algumas mensagens de orientação ao usuário.

```
1  var alunos := vetor[10]
2
3  # ler 10 nomes.
4  para i de 0 até 9 faça
5      escreva "O nome do {i+1}º aluno:"
6      alunos[i] := leia_texto
7  fim
8
9  # imprimir os 10 nomes lidos
10 para i de 0 até 9 faça
11     escreva "{i+1}º aluno: {alunos[i]}"
12 fim
13
```

O que achou dessa nova solução? Ela conta com pouco mais de 10 linhas de código e atende ao solicitado no exercício, além de ficar mais funcional e elegante do que a primeira solução apresentada.

Esse primeiro exemplo mostrou como utilizar uma variável dotipo **vetor** para armazenar texto, mas, em outras situações, você poderá precisar utilizar outro tipo de dado, como o **inteiro** e o **real**. Em todos os casos, você precisará indicar, na

declaração da variável, que deseja criar um vetor. Veja, no exemplo abaixo, a declaração de dois vetores — um para armazenar dados do tipo **inteiro** e outro do tipo **real**.

```
var faltas := vetor[40]
var notas := vetor[40]

# ler as faltas e notas de 40 alunos.
para i de 0 até 39 faça
   escreva "Digite a quantidade de faltas do {i+1}º aluno:"
   faltas[i] := leia_inteiro
   escreva "Digite a nota do {i+1}º aluno:"
   notas[i] := leia_real
fim

# imprimir as faltas e notas de 40 alunos.
para i de 0 até 39 faça
   escreva "O {i+1}º aluno tem {faltas[i]} faltas e sua nota é {notas[i]}."
fim

```

Atividade 02

Lendo e imprimindo dados organizados

Construa um algoritmo que receba os dados de 4 alunos. Para cada aluno, o programa recebe os dados na seguinte ordem: nome, faltas e média.

Após o algoritmo receber os dados, o seu programa deverá escrever a listagem dos alunos com base no modelo abaixo:

Formato da Saída esperada:

```
Nome – Nº de Faltas – Média
João Almeida – 12 – 7.1
Simbião Chronos – 0 – 8.9
Alice Lovelace – 0 – 9.1
Matoso Matos – 9 – 6.1
```

Atividade 03

Trabalhando com vetores

Crie um programa que receba 100 valores e armazena-os em um vetor. Em seguida, o programa deverá apresentar todas as posições do vetor em que há um valor menor ou igual a 10 armazenado, informando o índice da posição.

Para cada valor presente no vetor que seja menor ou igual a 10, escreva "**v[i] = x**", onde **i** é a posição do vetor e **x** é o valor armazenado na posição. Exemplo:

```
V[0] = -5.1
V[12] = 0.6
V[33] = 8.5
V[51] = 9.9
V[87] = 10.0
V[99] = -6.1
```

Agora que você conheceu as estruturas de dados homogêneas, acredito que já deve estar pensando em como melhorar os algoritmos que você criou na disciplina, não é mesmo? A cada novo recurso que você conhecer, imagine como utilizá-lo para solucionar os variados problemas. Não deixe de refazer os comandos com atenção, pois essa repetição te ajudará a fixar a sintaxe e a desenvolver cada vez mais sua habilidade na programação.

Até a próxima aula!

Resumo

Nesta aula, você conheceu as estruturas de dados homogêneas e trabalhou com as estruturas unidimensionais, além de realizar a construção de algoritmos por meio das estruturas de repetição e vetores. Nas próximas aulas, você conhecerá as estruturas homogêneas bidimensionais. Portanto, caso você tenha encontrado alguma dificuldade, converse com seu mediador para poder sanar o quanto antes as suas dúvidas.

O completo entendimento das estruturas homogêneas unidimensionais é importante para a construção dos seus próximos conhecimentos acerca do desenvolvimento de algoritmos.