



Desenvolvimento de Sistemas

Estrutura de dados: conceitos, aplicação e usabilidade; vetores, matrizes

Como visto durante o estudo desta unidade curricular, para programar é possível – e muitas vezes necessário – armazenar dados para usá-los posteriormente. Quando são armazenados na memória do computador ou do dispositivo, podendo inclusive ser alterados no decorrer do programa, eles são chamados de **variáveis**. Quando se mantêm inalterados do início ao fim do programa, são chamados de **constants**.

Esse dados variáveis e constantes são simples e únicos e permitem usar o armazenamento em memória do sistema, mas também apresentam algumas limitações importantes que impossibilitam melhorar a eficiência nos sistemas.

Logo, para incrementar sistemas e principalmente a lógica das aplicações, podem-se utilizar estruturas de dados, que nada mais são do que uma organização de múltiplos dados organizados na memória do sistema, de forma coerente, para que este possa ser armazenado e usado corretamente.



Conceitos

O conceito de estruturas de dados pode ser aplicado de diversas maneiras, havendo assim uma série de tipos de estruturas diferentes para diferentes aplicações. Dependendo do algoritmo, é possível trabalhar com um volume muito grande de informações, como aplicações de bancos de dados, busca, ordenação, entre outros.

As principais estruturas de dados usadas na maioria das linguagens de programação são:

- ◆ Vetores e matrizes
- ◆ Registro
- ◆ Lista
- ◆ Pilha
- ◆ Fila
- ◆ Árvore
- ◆ Dicionário

Contudo, nem todas são apresentadas em todas as linguagens de programação, então é preciso saber que tais estruturas existem e que não podem ser usadas em qualquer linguagem de programação. Veja alguns exemplos de estruturas:

Vetores e matrizes

Vetores e matrizes (ou *arrays*) são estruturas de dados simples que podem auxiliar quando, em algum algoritmo, são necessárias muitas variáveis do mesmo tipo.

- ◆ Vetor é uma variável que armazena várias variáveis do mesmo tipo.
- ◆ Matriz é um vetor que armazena vários vetores.
- ◆ Um vetor tem só uma dimensão, só uma linha de coordenadas. Já a matriz tem mais de uma dimensão e, consequentemente, mais de um índice de coordenadas.

Mais adiante no conteúdo, você verá mais detalhes sobre o assunto.

Registros

Um vetor pode armazenar muitos dados do mesmo tipo; já um registro pode armazenar dados de tipos diferentes, normalmente representando algo específico por meio desse conjunto. Em vez de posições, os registros têm campos que especificam cada informação. Por exemplo, um registro de **usuário** poderia ter os campos “Nome”, “Telefone”, “E-mail” e “Senha”.

Uma estrutura de registro sempre terá um nome (por exemplo: livro, carro, editora, produto) e campos, os quais, na maioria das linguagens de programação, são acessados com um ponto entre o nome e o campo (por exemplo: livro.autor,

livro.editora).



Listas

As listas como estruturas de dados são semelhantes aos vetores, podendo armazenar apenas um tipo de dado, mas com algumas diferenças. O primeiro ponto que difere é que, nas listas, a ordem dos valores e, por padrão, os vetores têm um tamanho fixo, e as listas têm tamanhos variáveis.

As listas podem ser incrementadas, ou seja, podem-se acrescentar novos valores no início ou no fim da lista. A retirada dos elementos pode ser feita a partir de qualquer ponto da estrutura.

Aliás, existem dois tipos de lista: lista ligada e lista duplamente ligada.

- ◆ **Lista ligada:** em uma estrutura de dados do tipo lista, cada valor está dentro de um nó, o qual funciona semelhantemente às variáveis vistas anteriormente, guardando valores na memória. Porém, além de conhecer o próprio valor, uma lista ligada conhece o valor seguinte.

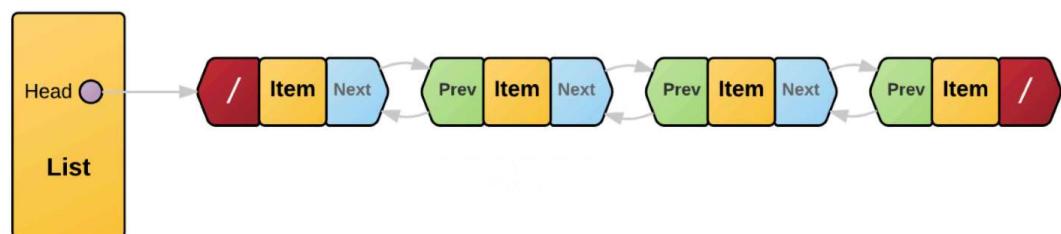


Figura 1 – Lista ligada

A imagem mostra um retângulo com as escritas “Head” e “List”. O retângulo é ligado por uma seta a um conjunto de três figuras. Na primeira figura há uma barra; na segunda, a palavra “Item”; e, na terceira, a palavra “Next”. Esse conjunto é ligado por setas a um novo conjunto de três figuras. Na primeira figura há a palavra “Prev”; na segunda, a palavra “Item”; e, na terceira, a palavra “Next”. Esse processo se repete no último conjunto, no qual a última figura contém uma barra.

O nome “lista ligada” deriva da união dos nós, pois cada nó é amarrado ao seguinte, indicando qual seria o próximo nó.

- ◆ **Lista duplamente ligada:** sabendo que a lista ligada é amarrada ao nó seguinte, a diferença entre lista ligada e lista duplamente ligada é que esta, além de ser amarrada ao nó seguinte, é também amarrada ao nó anterior, sendo assim bidirecional. Então, pode-se “andar” para frente e para trás dentro das informações da lista.

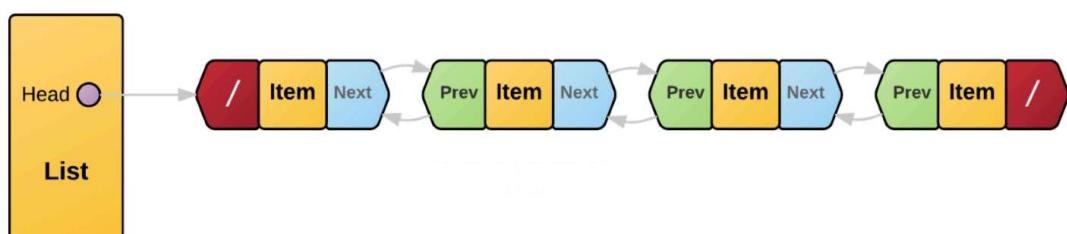


Figura 2 – Lista duplamente ligada



A imagem mostra um retângulo com as escritas “Head” e “List”. O retângulo é ligado por uma seta a um conjunto de três figuras. Na primeira figura há uma barra; na segunda, a palavra “Item”; e, na terceira, a palavra “Next”. Esse conjunto é ligado por setas a um novo conjunto de três figuras. Na primeira figura há a palavra “Prev”; na segunda, a palavra “Item”; e, na terceira, a palavra “Next”. Esse processo se repete no último conjunto, no qual a última figura contém uma barra.

Para exemplificar, basta imaginar uma alcateia, na qual os lobos se organizam com os mais velhos à frente e o alfa ao fim da fila, observando tudo. Cada lobo sabe, além da própria posição na fila, a posição de quem vai à frente e de quem vai atrás.



Figura 3 – Alcateia

Pilhas

Pilha é mais uma estrutura de dados que funciona como uma coleção de elementos permitindo acesso a somente um dado armazenado. Como o acesso aos dados é restrito e somente um item da pilha pode ser lido de cada vez, a inserção e a remoção dos valores também são afetadas, podendo ser feita apenas uma de cada vez.

- ◆ **Pilha LIFO:** a estrutura de dados pilha LIFO (*last in first out* – em tradução livre: “o último que entra é o primeiro que sai”) traz exatamente este princípio: só se pode retirar o último elemento que foi inserido na pilha, ou seja, o mais recente.

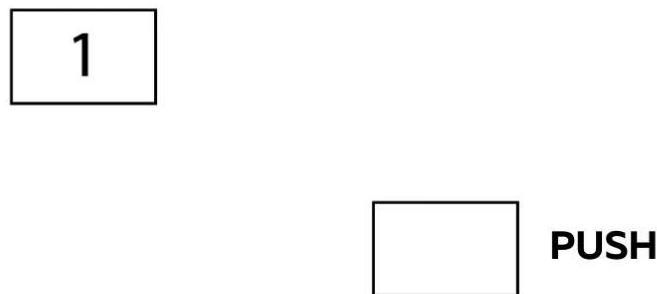


Figura 4 – Pilha *push*

A animação mostra um conjunto de quadrados, cada um com um número sequencial a partir de zero. Os quadrados vão sendo empilhados, um acima do outro na sequência.

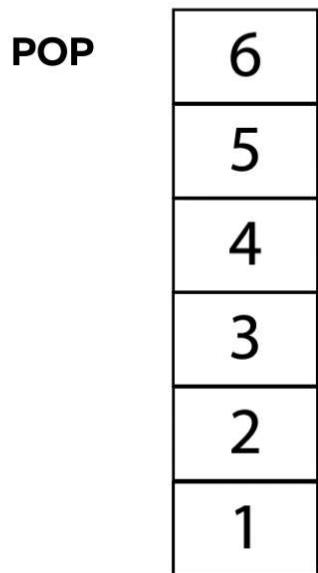


Figura 5 – Pilha *pop*

A animação mostra um conjunto de seis retângulos, empilhados cada um com um número de seis a um. Os retângulos vão sendo retirados um de cada vez, começando pelo maior.



Figura 6 – Depósito de caixas

Um bom exemplo é quando caixas ou outros objetos sem uso são organizados em prateleiras. Para acessar algum item, é preciso sempre tirar primeiro o que está mais em cima, ou seja, o que foi colocado por último.

- ◆ **Pilha FIFO:** a estrutura de dados pilha FIFO (*first in first out* – em tradução livre: “o primeiro que entra é o primeiro que sai) traz exatamente este princípio: só se pode retirar o primeiro elemento que foi inserido na pilha, ou seja, o mais antigo.

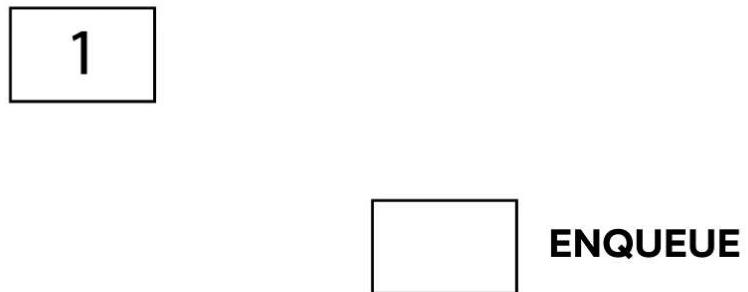


Figura 7 – Pilha **enqueue**

A animação mostra um conjunto de quadrados, cada um com um número sequencial a partir de zero. Os quadrados vão sendo empilhados, um acima do outro na sequência.

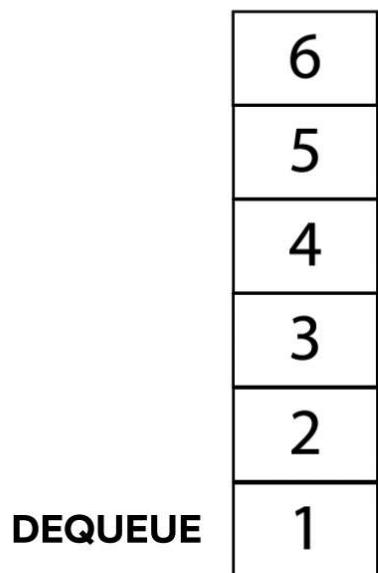


Figura 8 – Pilha **dequeue**

A animação mostra um conjunto de seis retângulos empilhados, cada um com um número de seis a um. Os retângulos vão sendo retirados um de cada vez, começando pelo menor.

Um exemplo fácil de ser lembrado são as filas de atendimento no banco ou no caixa do supermercado, nas quais quem chega primeiro é o primeiro a ser atendido.



Figura 9 – Fila em um caixa de supermercado

Árvores

Árvores são estruturas de dados que distribuem seus elementos em uma hierarquia na qual existem um elemento inicial, conhecido como **nó raiz**, e elementos subordinados a ele. Os nós eventualmente são chamados de **galhos**, **folhas** ou **nós filhos**.

Os nós filhos podem ter um ou mais nós filhos ou até mesmo nenhum nó. Os nós filhos que contêm outros nós filhos são chamados de galhos, e os que não contêm nó algum são chamados de folhas.

As árvores são uma das estruturas mais comumente usadas. Tudo no computador está dentro de uma pasta raiz (`/` ou `c:`) e, dentro dela, estendem-se os nós que também são pastas.



Figura 10 – Árvore de diretórios e arquivos

A figura mostra uma estrutura de pastas e, dentro da pasta principal, outras duas pastas. Dentro dessas duas pastas, há mais pastas e conteúdos.

- ◆ **Árvores binárias:** considerando que uma estrutura de dados do tipo árvore é um conjunto de nós, dentro deste pode haver subconjuntos ou subárvores que contêm um próprio conjunto de nós. Uma árvore pode ter seu conjunto de nós dividido em três subconjuntos distintos: o primeiro subconjunto é o nó raiz; o segundo, a subárvore da direita; e o terceiro, a subárvore da esquerda.

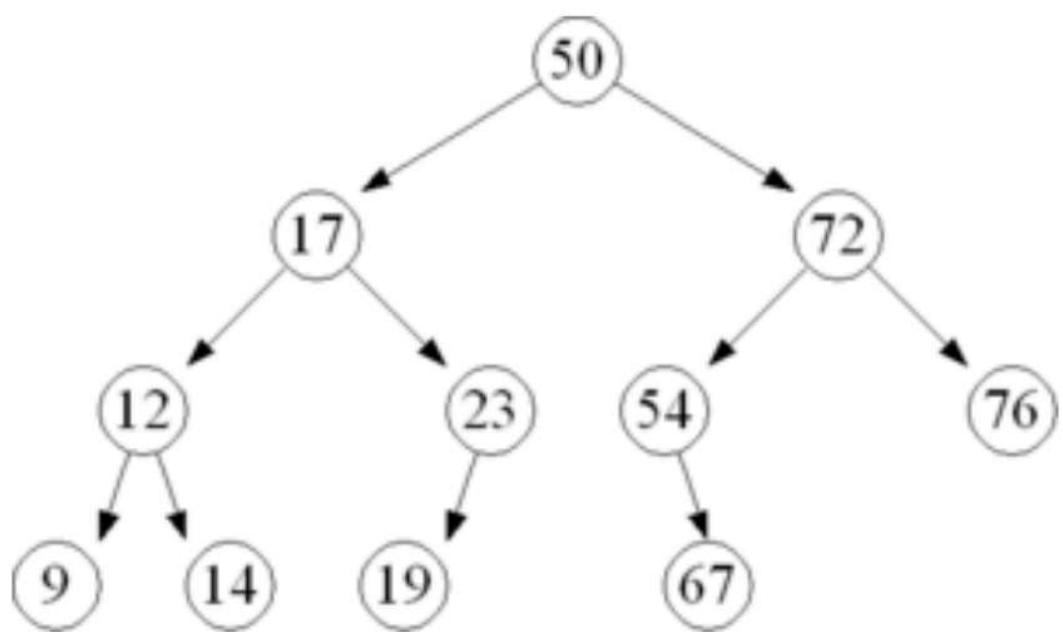


Figura 11 – Árvores binárias

A imagem mostra uma sequência de figuras com números aleatórios. Essa sequência tem uma figura principal ligada a duas figuras secundárias, e cada figura é ligada a mais uma ou duas figuras.

Dicionário

O dicionário é uma estrutura de dados que funciona como um vetor, uma lista ou uma pilha, com a diferença de que não é indexado automaticamente, e sim mapeado, ou seja, o dicionário é indexado manualmente por uma chave, com a qual

se acessa um elemento da estrutura (semelhante ao nome de uma variável), obtendo-se, assim, o valor armazenado nesse elemento (semelhante ao valor de uma variável).

Essa estrutura permite o acesso, a adição ou a remoção de elemento sem considerar a ordem; na verdade, o dicionário não tem ordem, pois não tem índice automático. Somente a chave importa.

Aplicação e usabilidade

Agora que você já viu a teoria a respeito das principais estruturas de dados, que tal aprender como elas funcionam na prática? Os exemplos serão elaborados no Portugol, então você deve testá-los no Portugol Studio.

Veja a seguir como criar as principais estruturas de dados.

Vetores

Como visto anteriormente, os vetores são estruturas de dados homogêneas, ou seja, só podem armazenar o mesmo tipo de dado. Se você criar duas variáveis do tipo inteiro, elas estarão em qualquer parte da memória do computador sem nenhuma espécie de relação entre si. Já ao criar um vetor com 50 posições, além de reservadas na memória, tais posições estarão juntas, por serem uma estrutura sequencial. Assim, percorrer um vetor é mais ágil do que percorrer um conjunto de variáveis.

Outra vantagem é poder usar estruturas de repetição para acessar as posições do vetor em sequência, o que será visto mais adiante neste conteúdo.

Iniciando um vetor

A declaração de um vetor em Portugol segue o padrão de declaração de uma variável, mas com uma diferença: depois do nome, usa-se um par de colchetes informando o tamanho do vetor.

Veja exemplos:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando um vetor de números inteiros com cinco posições  
        inteiro vetor[5]  
  
        // Declarando um vetor de caractere com duzentas posições  
        caractere vetor2[200]  
  
        // Declarando um vetor de números fracionários com trinta e quatro posições  
        real vetor3[34]  
  
        // Declarando um vetor de palavras com cem posições  
        cadeia vetor4[100]  
  
        // Declarando um vetor de verdadeiro ou falso com cinco posições  
        logico vetor5[5]  
  
    }  
}
```

Os vetores são autoindexados, ou seja, já são criados com uma chave para cada posição. Essas chaves são números inteiros, começando com o número zero.

Preenchendo o vetor

Para acessar uma posição, deve-se chamar o nome do vetor com a posição que se quer acessar dentro dos colchetes. Então, pode-se atribuir um valor acessando diretamente a posição do vetor, conforme segue:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando um vetor de números inteiros com cinco posições  
        inteiro vetor[5]  
  
        // Acrescentando um valor a uma posição  
        vetor[0] = 34  
  
        // Declarando um vetor de caracter inteiros com cinco duzentas posições  
        caracter vetor2[200]  
  
        // Acrescentando um valor a uma posição  
        vetor2[20] = 'w'  
  
        // Declarando um vetor de números fracionários com trinta e quatro posições  
        real vetor3[34]  
  
        // Acrescentando um valor a uma posição  
        vetor3[2] = 34.32  
  
        // Declarando um vetor de palavras com cem posições  
        cadeia vetor4[100]  
  
        // Acrescentando um valor a uma posição  
        vetor4[99] = "Mauro"  
  
        // Declarando um vetor de verdadeiro ou falso com cinco posições  
        logico vetor5[5]  
  
        // Acrescentando um valor a uma posição  
        vetor5[2] = verdadeiro  
  
    }  
}
```

Lembre-se de que o vetor tem o tamanho que você decidir, mas deve começar com a posição zero. Então, se você criar um vetor com cinco posições, a primeira delas é zero; e a última, quatro.

Observe o trecho a seguir, que apresenta um erro:

```
programa {  
    funcao inicio() {  
  
        // Declarando um vetor de palavras com cem posições  
        cadeia vetor[100]  
  
        // Acrescentando um valor a uma posição  
        vetor[100] = "Mauro"  
    }  
}
```

Saída:

Erro de execução: Você tentou acessar um índice de vetor ou matriz inválido.

O índice, no caso dos vetores, deve ser menor que o número de elementos que o vetor tem. Por exemplo, se foi declarado um vetor com cinco elementos (inteiro vetor[5]), o maior índice possível é quatro (4). O mesmo erro ocorreria com qualquer valor fora do escopo do vetor, isto é, com valores maiores que o último número antes do tamanho ou menor que zero.

Para descobrir a última posição do vetor, basta subtrair 1 do tamanho. Logo, nesse caso, um vetor de tamanho 5 terá como sua última posição 4; um vetor de tamanho 7432 terá como sua última posição 7431. Então, um vetor padrão de **N** posições tem índices de posições que vão de zero a **N -1**.

Atribuição direta

Outra possibilidade é, ao criar um vetor, já atribuir os valores de suas posições diretamente na linha da declaração do vetor, o que funciona muito bem com vetores menores. Para fazer uma atribuição direta, você deve iniciar a declaração do vetor normalmente e, no fim da linha, acrescentar o sinal de igual para atribuição e depois um par de chaves. Os valores são colocados separados por vírgula dentro das chaves.

Confira um exemplo:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando um vetor de números inteiros com cinco posições e atribuindo seus valores  
        inteiro  vetor[5] = {13,4,87,2,22}  
  
        // Declarando um vetor de caractere com cinco posições e atribuindo seus valores  
        caractere vetor2[5] = {'x','y','z','a','b'}  
  
        // Declarando um vetor de números fracionários com cinco posições e atribuindo seus valores  
        real      vetor3[5] = {1.5,8.1,3.0,15.1587,5.1}  
  
        // Declarando um vetor de palavras com cinco posições e atribuindo seus valores  
        cadeia   vetor4[5] = {"Hugo","José","Luiz","Patricia","Mauro Duarte"}  
  
        // Declarando um vetor de valores verdadeiro ou falso com cinco posições e atribuindo seus valores  
        logico   vetor5[5] = {verdadeiro,falso,falso,verdadeiro,verdadeiro}  
  
    }  
}
```

Lendo o vetor

A leitura do vetor é, a princípio, feita pela leitura da sua posição, isto é, o nome do vetor é seguido de um par de colchetes e, dentro dos colchetes, vai o número da posição do vetor.

Por exemplo:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando um vetor de números inteiros com cinco posições e atribuindo seus valores  
        inteiro  vetor[5] = {13,4,87,2,22}  
  
        // Mostrando a posição 0 do vetor  
        escreva(vetor[0])  
        escreva("\n")  
  
        // Declarando um vetor de caractere com cinco posições e atribuindo seus valores  
        caracter vetor2[5] = {'x','y','z','a','b'}  
  
        // Mostrando a posição 1 do vetor2  
        escreva(vetor2[1])  
        escreva("\n")  
  
        // Declarando um vetor de números fracionários com cinco posições e atribuindo seus valores  
        real      vetor3[5] = {1.5,8.1,3.0,15.1587,5.1}  
  
        // Mostrando a posição 0 do vetor3  
        escreva(vetor3[0])  
        escreva("\n")  
  
        // Declarando um vetor de palavras com cinco posições e atribuindo seus valores  
        cadeia   vetor4[5] = {"Hugo","José","Luiz","Patricia","Mauro Duarte"}  
  
        // Mostrando a posição 3 do vetor4  
        escreva(vetor4[3])  
        escreva("\n")  
  
        // Inicializando um vetor de valores verdadeiro ou falso com cinco posições e atribuindo seus valores  
        logico   vetor5[5] = {verdadeiro,falso,falso,verdadeiro,verdadeiro}  
  
        // Mostrando a posição 0 do vetor5  
        escreva(vetor5[0])  
        escreva("\n")  
    }  
}
```

Lembre-se de que o vetor é uma estrutura de memória sequencial, então é relativamente simples percorrê-lo. Usando uma estrutura de laço de repetição, pode-se facilmente percorrer o vetor do início ao fim.

Caso você não se lembre, reveja o conteúdo **Repetições: estruturas de repetição condicional pré-teste, pós-teste e com variável de controle**, disponível nesta unidade curricular.

Veja primeiramente um exemplo preenchendo o vetor com informações informadas pelo usuário:

```
programa
{
    funcao inicio()
    {
        // Declarando um vetor de inteiros
        inteiro vetor[10]

        // preenche o vetor com números informados pelo usuário
        para (inteiro posicao = 0; posicao < 10; posicao++)
        {
            escreva("Informe um número inteiro:")
            leia(vetor[posicao])
        }
    }
}
```

Agora aumente o exemplo adicionando a leitura do vetor, também utilizando o laço de repetição **para**:

```
programa
{
    funcao inicio()
    {
        // Declarando um vetor de inteiros
        inteiro vetor[10]

        // preenche o vetor com numeros informados pelo usuário
        para (inteiro posicao = 0; posicao < 10; posicao++)
        {
            escreva("Informe um número inteiro:")
            leia(vetor[posicao])
        }

        // Lê e exibe o vetor conteúdo do vetor
        escreva ("Exibindo o conteúdo do vetor item por item\n")

        para(inteiro posicao = 0; posicao < 10; posicao++)
        {
            escreva (vetor[posicao], " - ")
        }
    }
}
```

Que tal realizar alguns desafios?

Crie um vetor e preencha-o com valores. Após, exiba o conteúdo na tela na ordem inversa ao que foi preenchido. Por exemplo: se o vetor foi preenchido com [1,2,3,4,5], deve aparecer na tela [5,4,3,2,1].

Elabore um algoritmo que solicite ao usuário a entrada de cinco números e que exiba o somatório deles na tela. Após exibir a soma, o programa deve mostrar também os números que o usuário digitou, um por linha.

Escreva um algoritmo que solicite ao usuário a entrada de 5 nomes, e que exiba a lista desses nomes na tela. Após exibir essa lista, o programa deve mostrar também os nomes na ordem inversa em que o usuário os digitou, um por linha.

Como exemplo para um dos desafios, faça um programa que receba o nome de cinco amigos aos quais você emprestou dinheiro e também o valor emprestado. Depois, calcule e exiba o total emprestado.

```
programa{  
  
    funcao inicio(){  
  
        //declaração das variáveis que vamos usar  
        inteiro indice  
        real total = 0.0  
        cadeia nome[5]  
        real valor[5]  
  
        // Início do programa  
        escreva("Programa de controle do dinheiro emprestado:")  
        escreva("\n")  
  
        // Preenchendo as informações  
        para(indice = 0; indice < 5; indice++){  
            escreva("Informe o nome do Amigo: ")  
            leia(nome[indice])  
            escreva("informe quanto o Amigo deve: ")  
            leia(valor[indice])  
        }  
  
        // Mostrando o resumo  
        escreva("\n")  
        escreva("Resumo:")  
        escreva("\n")  
  
        para(indice = 0; indice < 5; indice++){  
            escreva("Meu amigo ",nome[indice]," me deve R$",valor[indice])  
            escreva("\n")  
        }  
  
        //Calculando e mostrando valor total  
  
        escreva("\n")  
        escreva("Total emprestado: R$")  
  
        para(indice = 0; indice < 5; indice++){  
            total = total + valor[indice]  
        }  
        escreva(total)  
    }  
}
```



Matrizes

Matrizes são estruturas de dados multidimensionais, ou seja, têm mais de uma dimensão, como se fossem um gráfico cartesiano no qual há o eixo **x** e o eixo **y**. Só que, diferentemente de um gráfico, cada posição dentro da matriz pode receber um valor individual, funcionando como uma tabela de dados ou, em outras palavras, uma série de vetores empilhados.

Em uma representação da matriz como uma tabela, tem-se o equivalente a linhas e colunas. Lembre-se de que a figura 12 é apenas uma representação para melhor compreensão dos conceitos e que os dados na memória não estão necessariamente organizados assim fisicamente no *hardware*.

		COLUNAS	
		0	1
LINHAS	0	45	32
	1	26	14
2	40	43	65
3	74	79	48
		95	77
		55	

Figura 12 – Matriz

A figura mostra uma tabela de quatro linhas e quatro colunas com números aleatórios em cada célula. Fora da tabela, textos identificam as linhas e as colunas e também o número de cada linha, de zero a três, e o número de cada coluna, de zero a três.

As matrizes – assim como os vetores – ajudam, mas contam com uma vantagem a mais: elas têm mais dimensões. Imagine, por exemplo, que você quer armazenar as notas da prova que os alunos de uma turma fizeram. Um vetor simples pode tranquilamente armazenar todas as notas, representado da seguinte forma:

Notas	7.5	8	4	5.5	7	...
-------	-----	---	---	-----	---	-----

Figura 13 – Vetor de notas

A figura mostra a palavra “Notas” seguida de uma série de quadrados sequenciais com as notas dos alunos.

E se, para cada aluno, precisassem ser armazenadas três notas de três provas? Com vetores simples, até poderiam ser criados três vetores, um para cada prova, mas, com uma matriz, podem-se armazenar colunas e linhas que representam as provas e os alunos da seguinte forma:

Notas	7.5	8	4	5.5	7	...
	7	6	5	6	7	...
	8	9	8	5.5	7.5	...

Figura 14 – Matriz de notas

A figura mostra a palavra “Notas” seguida de uma tabela de quadrados sequenciais, com três linhas e um número indefinido de colunas. Cada quadrado traz as notas das provas dos alunos.

Assim, cada coluna representa o aluno e cada linha representa uma das provas. A interseção entre alunos e provas é a nota do aluno na respectiva prova:

Notas	Aluno 1	Aluno 2	Aluno 3	Aluno 4	Aluno 5	
Prova 1	7.5	8	4	5.5	7	...
Prova 2	7	6	5	6	7	...
Prova 3	8	9	8	5.5	7.5	...

Figura 15 – Matriz de notas

A figura mostra a palavra “Notas” seguida de uma tabela de quadrados sequenciais, com três linhas e um número indefinido de colunas. Cada quadrado traz as notas das provas dos alunos. A figura também apresenta as legendas das linhas, como “Prova 1”, “Prova 2” e “Prova 3”, e das colunas, como “Aluno 1”, “Aluno 2”, “Aluno 3”, e assim por diante.

Iniciando uma matriz

Para criar uma matriz, deve-se seguir o mesmo princípio de criação de um vetor, inserindo o tipo, o nome e, na sequência, os tamanhos, desta vez com dois pares de colchetes e os tamanhos dentro destes.

Por exemplo:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando uma matriz de números inteiros com 5 colunas e duas linhas, dez posições  
        inteiro matriz[5][2]  
  
        // Declarando uma matriz de caracteres com 20 colunas e 20 linhas, 400 posições  
        caracter matriz2[20][20]  
  
        // Declarando uma matriz de números fracionários com 3 linhas e 9 colunas, vinte e sete posições  
        real matriz3[3][9]  
  
        // Declarando uma matriz de palavras com 10 linhas e 10 colunas, cem posições  
        cadeia matriz4[10][10]  
  
        // Declarando uma matriz de verdadeiro ou falso com cinco linhas e 3 colunas, 5 posições  
        logico matriz5[5][3]  
  
    }  
}
```

Assim como os vetores, as matrizes são autoindexadas, isto é, já são criadas com uma chave para cada posição. Essas chaves são pares de números inteiros, começando com o par **zero, zero**.

Preenchendo a matriz

Para acessar uma posição, deve-se chamar o nome da matriz com a posição que se quer acessar dentro do par de colchetes. Então, pode-se atribuir um valor acessando diretamente a posição da matriz, conforme segue:

```
programa {  
  
    funcao inicio() {  
  
        // Declarando uma matriz de números inteiros com 5 colunas e duas linhas, dez posições  
        inteiro matriz[5][2]  
  
        // Acrescentando um valor a uma posição  
        matriz[0][1] = 34  
  
        // Declarando uma matriz de caracteres com 20 colunas e 20 linhas, 400 posições  
        caracter matriz2[20][20]  
  
        // Acrescentando um valor a uma posição  
        matriz2[2][1] = 'w'  
  
        // Declarando uma matriz de números fracionários com 3 linhas e 9 colunas, vinte e sete posições  
        real matriz3[3][9]  
  
        // Acrescentando um valor a uma posição  
        matriz3[2][8] = 34.32  
  
        // Declarando uma matriz de palavras com 10 linhas e 10 colunas, cem posições  
        cadeia matriz4[10][10]  
  
        // Acrescentando um valor a uma posição  
        matriz4[9][9] = "Mauro"  
  
        // Declarando uma matriz de verdadeiro ou falso com cinco linhas e 3 colunas, 5 posições  
        logico matriz5[5][3]  
  
        // Acrescentando um valor a uma posição  
        matriz5[2][2] = verdadeiro  
  
    }  
}
```

Atribuição direta



Tal como ocorre com o vetor, também é possível criar a matriz e já atribuir os valores de suas posições diretamente na linha da declaração.

Para fazer uma atribuição direta, você deve iniciar a declaração da matriz normalmente e, no fim da linha, acrescentar o sinal de igual para atribuição e depois um par de chaves. Cada “linha” da matriz deve ser preenchida com os valores colocados separados por vírgula. Assim, a matriz deve ser “montada” uma linha por vez.

Por exemplo:

```
/*
Criando uma matriz de inteiros chamada mat com 2 linhas e 3 colunas e fornecendo
um conjunto de valores.

Primeira linha: {34, 87, 25}
Segunda linha: {56, 90, 58}
*/
programa {

    funcao inicio() {
        inteiro mat[2][3] = {{34,56,25},{87,90,58}};

    }
}
```

Lendo a matriz

A leitura da matriz é feita pelas coordenadas da sua posição, ou seja, o nome da matriz é seguido de dois pares de colchetes e, dentro de cada par de colchetes, vai o número da linha e da coluna.

```
/*
Primeira linha: {34, 87, 25}
Segunda linha: {56, 90, 58}
*/
programa {

funcao inicio() {

// Declarando a Matriz e atribuindo valores
inteiro mat[2][3] = {{34,56,25},{87,90,58}};

// Lendo uma posição da matriz e escrevendo na tela
// linha 0 (primeiro) coluna 2 (terceira)
escreva(mat[0][2])

}

}
```

Nunca se esqueça de que tanto a matriz quanto o vetor iniciam em zero, então a primeira posição é [0][0].

Usando uma estrutura de laço de repetição, pode-se percorrer a matriz do início até o fim, o que, de fato, é um pouco mais complicado em comparação ao vetor. É preciso, no caso da matriz, percorrer linha por linha.

Caso você não se lembre de como utilizar laços **para** aninhados, reveja o conteúdo **Repetições: estruturas de repetição condicional pré-teste, pós-teste e com variável de controle**, disponível nesta unidade curricular.

Veja um exemplo preenchendo a matriz com informações informadas pelo usuário:

```
programa {  
  
    funcao inicio() {  
  
        inteiro linha, coluna  
  
        // Declarando a Matriz  
        inteiro mat[2][3]  
  
        // Percorrendo a Matriz e inserindo valores  
  
        // iniciamos acessando a primeira linha (depois de passar pela primeira  
        // linha vai repetindo)  
        para (linha = 0; linha < 2; linha++) {  
            // Dentro de cada linha vamos percorrer as colunas  
            para (coluna = 0; coluna < 3; coluna++){  
                escreva("Informe um número inteiro:")  
                leia(mat[linha][coluna])  
            }  
        }  
    }  
}
```

Agora aumente o exemplo adicionando a leitura da matriz, também utilizando o laço de repetição **para**:

```
programa {

    funcao inicio() {

        inteiro linha, coluna

        // Declarando a Matriz
        inteiro mat[2][3]

        // Percorrendo a Matriz e inserindo valores

        // iniciamos acessando a primeira linha (depois de passar pela primeira linha vai repetindo)
        para (linha = 0; linha < 2; linha++){
            // Dentro de cada linha vamos percorrer as colunas
            para (coluna = 0; coluna < 3; coluna++){
                escreva("Informe um número inteiro:")
                escreva(mat[linha][coluna])
            }
        }

        // iniciamos acessando a primeira linha (depois de passar pela primeira linha vai repetindo)
        para (linha = 0; linha < 2; linha++){
            // Dentro de cada linha vamos percorrer as colunas
            para (coluna = 0; coluna < 3; coluna++){

                escreva(mat[linha][coluna])
                escreva(",")
            }
            // Vamos quebrar uma linha pra ficar mais legal
            escreva("\n")
        }
    }
}
```

Como exemplo, elabore um programa que receba do usuário o nome de cinco alunos e, na sequência, cadastre notas de três provas para cada aluno. Após o cadastro, exiba para cada aluno as notas deles.

```
programa {  
  
    funcao inicio() {  
  
        inteiro linha, coluna  
  
        // Declarando um vetor de nomes  
        cadeia nome[5]  
        // Declarando a Matriz para as notas  
        real notas[5][3]  
  
  
        para (linha = 0; linha < 5; linha++)  
        {  
            escreva("Informe o nome do Aluno: ")  
            leia(nome[linha])  
  
  
            para (coluna = 0; coluna < 3; coluna++){  
                escreva("Informe a nota",coluna+1,: "")  
                leia(notas[linha][coluna])  
            }  
        }  
  
  
        // iniciamos acessando a primeira linha (depois de passar pela primeira lin  
        ha vai repetindo)  
        para (linha = 0; linha < 5; linha++)  
        {  
            // Mostrando o nome do aluno e o inicio da linha  
            escreva("O Aluno ",nome[linha]," obteve as notas: ")  
  
            // Dentro de cada linha vamos percorrer as colunas  
            // Montando o resto da linha  
            para (coluna = 0; coluna < 3; coluna++){  
                escreva(notas[linha][coluna])  
                escreva(" ")  
            }  
            // Vamos quebrar uma linha pra ficar mais legal  
            escreva("\n")  
        }  
    }  
}
```

Que tal realizar mais alguns desafios?



Atualize o algoritmo anterior para que, além das notas, mostre também as médias e o *status* de aprovado ou reprovado. Como auxílio, releia o conteúdo **Condicionais: Lógica booliana e estruturas condicionais simples e compostas**.

Em uma festa, há uma mesa com 6 lugares de cada lado, totalizando 12 lugares. Elabore um programa que receba o nome dos 12 convidados e que armazene esses nomes em uma matriz que representa a mesa. Depois, mostre na tela os pares dos nomes dos convidados que estão de frente um para o outro em cada linha até mostrar todos.

Encerramento

Agora que você já viu as estruturas de dados e a forma de aplicá-las em algoritmos, pode elaborar algoritmos cada vez mais complexos, aproveitando tais estruturas para simplificar alguns processos (por exemplo, em vez de criar uma série de variáveis, você pode criar apenas um vetor ou uma matriz).

As estruturas abordadas neste conteúdo o acompanharão em toda a sua trajetória como programador, como analista de sistemas, como gerente de banco de dados ou como qualquer outra profissão possível que você queira seguir dentro do universo da tecnologia da informação.

Assista ao vídeo com exemplos do conteúdo estudado até aqui.

