Universidade Federal de Santa Maria
Prof. Cesar Tadeu Pozzer
Lógica e Algoritmo – ELC1064
Autores: Guilherme Schardong e Cesar Pozzer
11/05/2022

# Representação e Manipulação de Dados

Este módulo trata de formas de representação de dados mais complexos e em grande quantidade. São apresentados vetores, matrizes e estruturas (conjunto de dados heterogêneos).

## 1. Vetores

Muitas vezes é necessário fazer o processamento de muitos dados. Suponha o caso de encontrar o maior valor lido de um conjunto de 5 valores. O seguinte algoritmo resume esse problema

```
Início
   Var: A, B, C, D, E
   Ler A, B, C, D, E
   Se( A > B e A > C e A > D e A > E ) então
      Escreva( A )
   Senão Se( B > A e B > C e B > D e B > E) então
      Escreva( B )
   Senão Se( C > A e C > B e C > D e C > E) então
      Escreva( C )
   Senão Se( D > A e D > B e D > C e D > E ) então
      Escreva( D )
   Senão
      Escreva ( E )
   Fim se
Fim.
```

Imagine como ficaria esse exemplo se fosse necessário encontrar o maior valor de um conjunto de 100 valores. Para tratar esse problema, faz-se de uso de vetores. Um vetor é semelhante a uma variável, porém pode armazenar vários valores de um mesmo tipo, que são indexados por um índice inteiro.

Abaixo temos um exemplo de como definir um vetor de 100 elementos e como acessar cada um de seus elementos.

```
Início
   Var: vet[100], cont
   cont = 0

   Enquanto( cont < 100 ) repita
      vet[cont] = ler()
      cont = cont + 1
   Fim enquanto
Fim.
```

No exemplo acima, a variável `cont` é um número inteiro que varia entre 0 e 99, por isso, ela pode ser usada para acessar os elementos do vetor `vet`. Esse exemplo demonstra a leitura de dados para dentro de um vetor. Para exibir os dados, o princípio é o mesmo, basta criar um laço com um contador variando de 0 a 99 e escrever `vet[cont]`.

A indexação de um vetor começa pelo número 0 e termina em n-1, sendo n o número de elementos do vetor. Ou seja, no caso de um vetor de 8 elementos, o primeiro elemento é o vet[0] e o último elemento é o vet[7]. A figura a seguir ilustra esse exemplo.

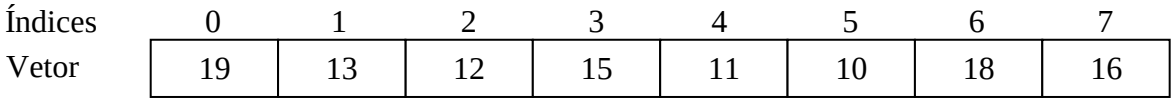



**Figura 1: Exemplo de um vetor de 8 posições**

Observe que o vetor ilustrado na figura acima possui 8 elementos e que os seus índices variam entre 0 e 7. Esse conceito é muito importante, pois grande parte dos erros nos algoritmos surgem da indexação errada dos vetores!

Abaixo temos exemplos em código e imagem da troca de dois valores dentro de um vetor. Trocaremos o valor v[0] com o valor em v[5]. Os valores em vermelho são aqueles que foram modificados a cada passo do algoritmo.

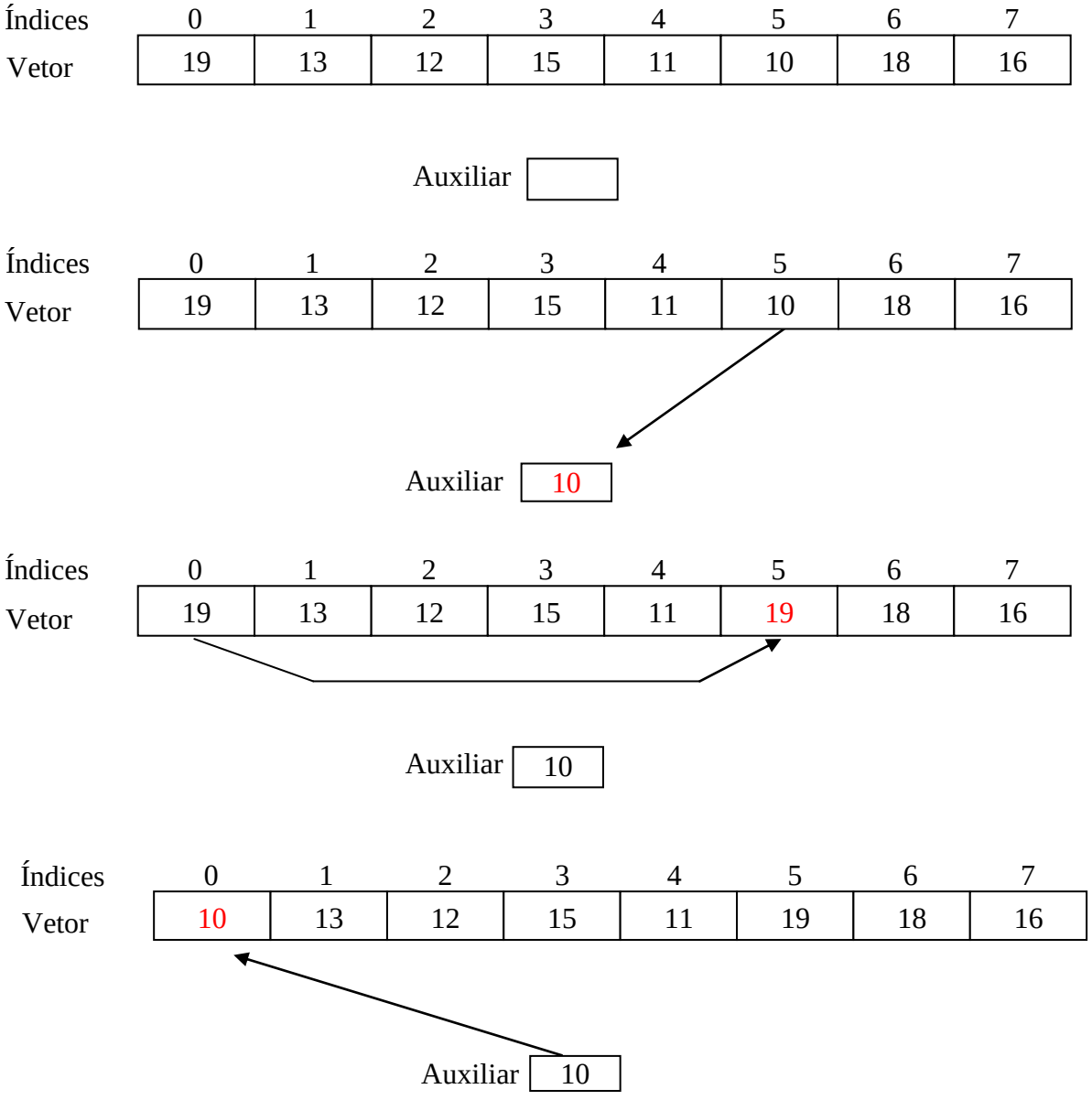



**Figura 2: Exemplo de troca de valores dentro de um vetor.**

Em código, o exemplo acima tomaria a seguinte forma:

```
Início
   Var: vet[8], aux
   aux = vet[5]
   vet[5] = vet[0]
   vet[0] = aux
Fim.
```

Para trocar dois valores dentro de um vetor, é necessária uma variável auxiliar para armazenar temporariamente o valor de uma das posições. Agora considere o seguinte exemplo com o vetor preenchido com os valores da figura 2.

```
Início
   Var: vet[8]
   vet[0] = vet[5]
   vet[5] = vet[0]
Fim.
```

Quais seriam os valores presentes nas posições 0 e 5 do vetor? Ocorreria a troca de posição dos valores 10 e 19?

Para encontrar o maior valor de um conjunto com N elementos, armazenados em um vetor, tem-se uma solução bem simples e compacta. Observem como foi inicializada a variável max.

```
Início
   Var: vet[100], cont, max
   max = vet[0]
   Para cont = 1 até 99 repita
      Se ( vet[cont] > max ) então
          max = vet[cont]
      Fim se
      cont = cont + 1
   Fim para
   Escrever("o maior valor é " + max)
Fim.
```

Outro exemplo interessante envolve ler um conjunto de valores, armazená-los em um vetor e escrevê-los junto com os seus índices. Por exemplo, para o vetor a seguir, o resultado do algoritmo é exibido abaixo.

| Índices | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Vetor | 2000 | 1321 | 1009 | 2314 | 9008 | 2314 | 1223 | 2555 |

| | |
|---|---|
| Início<br>   Var: vet[8], cont<br><br>   Para cont = 0 até 7 repita<br>      Escreva (vet[cont], cont)<br>      cont = cont + 1<br>   Fim para<br>Fim. | 2000, 0<br>1321, 1<br>1009, 2<br>2314, 3<br>9008, 4<br>2314, 5<br>1223, 6<br>2555, 7 |

**Figura 3: Exemplo da escrita dos valores de um vetor juntamente com os seus índices.**

Para escrever os valores de um vetor de trás para frente não é necessário inverter o vetor, basta iterar de trás para frente. O exemplo a seguir demonstra isso.

```
Início
   Var: vet[20], cont

   Para cont = 19 até 0 repita
      Escreva vet[cont]
      cont = cont - 1
   Fim para
Fim.
```

Observe que a variável `cont` começa com o valor 19 (última casa do vetor) e decresce até chegar a 0 (primeira casa do vetor). Assim, não é necessário inverter o vetor, basta iterar nele de trás para frente.

O exemplo a seguir demonstra o deslocamento dos valores para a esquerda em um vetor. O primeiro valor, como é o valor mais a esquerda é enviado para a última casa do vetor dando a impressão de que o vetor é circular.

```
Início
   Var: vet[5], cont, tmp

   tmp = vet[0]
   Para cont = 1 até 4 repita
      vet[cont - 1] = vet[cont]
      cont = cont + 1
   Fim para
   vet[4] = tmp
Fim.
```

**Exercícios:**

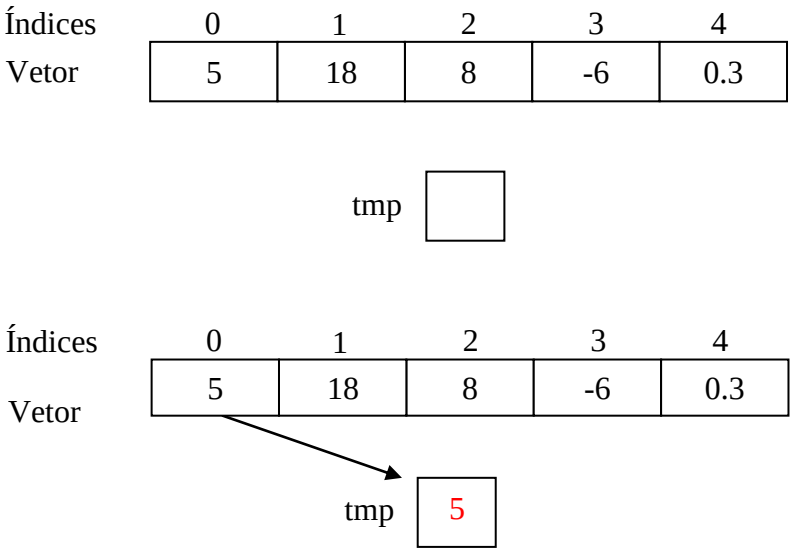
Função Acha(vet, elem, tam)
- Imprime achou/naoachou
- retorna idx

Funcao Remove(vet, ele, tam)
- shift esquerda
- troca com último removido
- marca como não válido

Funcao principal()
  idx = Acha()
  tam = Remove()

Observe que no exemplo acima é necessário armazenar o valor de `vet[0]` em uma variável temporária, caso contrário ele será sobrescrito na primeira iteração do laço, onde `vet[0] = vet[1]`.

A figura 4 ilustra o processo do algoritmo anterior.



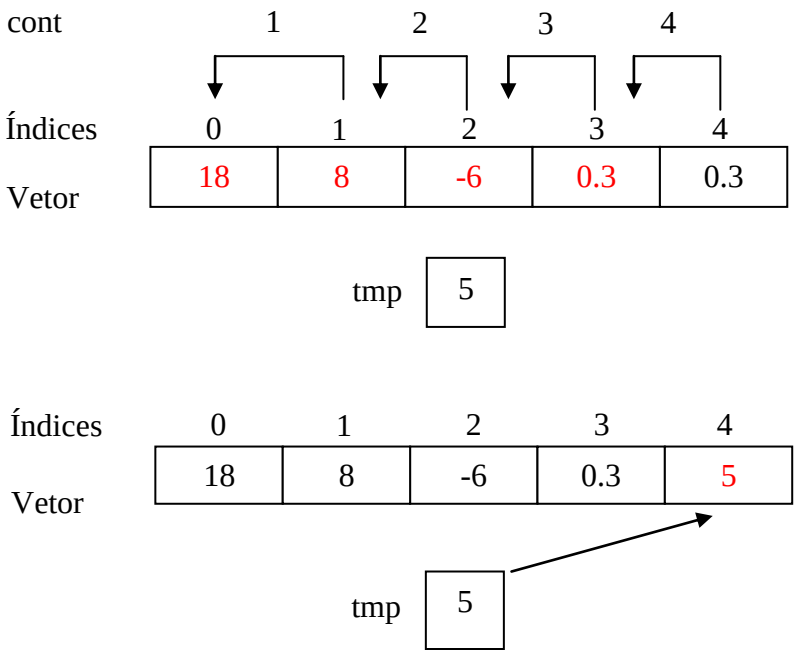



**Figura 4: Exemplo o deslocamento para a esquerda de todos os elementos de um vetor.**

Abaixo temos um exemplo da eliminação de valores repetidos em uma string (vetor de caracteres), com no máximo 20 caracteres, juntamente com a entrada do algoritmo e a saída esperada para a entrada. Os caracteres repetidos serão substituídos por um "-". Note que no exemplo abaixo, a primeira ocorrência da letra não é substituída, e que o algoritmo substitui todas as ocorrências posteriores da mesma letra.

| | | |
|---|---|---|
| `Início`<br>`  Var: str[20], i, j`<br>`  Para i=0 até 19 repita`<br>`  Ler(str[i])`<br>`  i++`<br>`  Fim para`<br>`  Para i=0 até 18 repita`<br>`  Para j=i+1 até 19 repita`<br>`  Se str[i] == str[j] então`<br>`  str[j] = '-'`<br>`  Fim se`<br>`  j++`<br>`  Fim para`<br>`  i++`<br>`  Fim para`<br>`Fim` | Entradas:<br>`TEMPERA`<br>`PINGUIM`<br>`GARRAFA`<br>`BATATA`<br>`MORCEGO`<br>`PANO` | Saídas:<br>`TEMP-RA`<br>`PINGU-M`<br>`GAR--F-`<br>`BAT---`<br>`MORCEG-`<br>`PANO` |

O exemplo acima atua da seguinte forma: Escolhe um elemento do vetor e compara-o com todos os elementos seguintes. Se encontrar alguma repetição, substitui a letra repetida por um "-". O algoritmo não precisa comparar uma determinada letra com as anteriores, pois ele garante que não haverão caracteres repetidos até o ponto atual. E o algoritmo não pode comparar uma posição do vetor com ela mesma, pois ele acusará repetição e substituirá o primeiro caractere, o que não é desejado, por isso o valor de `j` começa sempre em `i + 1`.

Também pode ser feito um teste dentro do primeiro laço que verifica se o caractere já foi substituído por um "-" e apenas se não foi substituído ele procura por repetições, senão, ele pula para o próximo elemento. O algoritmo ficaria da seguinte forma:

```
Início
   Var: str[20], i, j
   Para i=0 até 19 repita
      Ler(str[i])
      i++
   Fim para
   Para i=0 até 18 repita
     Se str[i] != '-' então
       Para j=i+1 até 19 repita
         Se str[i] == str[j] então
           str[j] = '-'
         Fim se
         j++
       Fim para
     Fim se
     i++
   Fim para
Fim.
```

Vetores também podem ser passados para funções, como mostrado no seguinte exemplo, que troca o valor de duas posições do vetor. Assume-se que o tamanho do vetor na função troca() seja conhecido. Caso não seja (caso genérico), deve-se passar uma variável indicando o tamanho do vetor, como mostrado nos exercícios abaixo.

```
Início
   Var: vet[20]
   vet[2] = 6
   vet[4] = 10
   troca(vet, 2, 4)
   Escreva(vet[2], vet[4])
Fim
```

```
Funcao troca(v[20], i, j)
   Var: aux
   aux = v[i]
   v[i] = v[j]
   v[j] = aux
Fim Funcao.
```

Existe uma diferença em relação a passagem de variáveis convencionais. No exemplo anterior, ao realizar a troca dos elementos i e j do vetor v[] na função troca() faz com que os valores do vetor vet[] na função principal sejam trocados, ou seja, o vetor v[] é o mesmo vetor vet[]. Alterar v[] é equivalente a alterar vet[]. A função escreva() vai imprimir 10,6.

Para variáveis convencionais, como mostrado no seguinte exemplo, a alteração da variável a na função troca não altera o valor da variável a da função principal, pois são variáveis ***locais*** em cada função.

```
Início
   Var: a
   a = 5
   troca(a)
   Escreva(a)
Fim.
```

```
Funcao troca(a)
   a = 20
Fim Funcao
```

# TODO: exemplos de funções recursivas

***Exercícios:*** Elabore algoritmos para os seguintes problemas:

1. Função procura(vetor[], valor, tam) que retorna 1 se *valor* é igual a algum elemento de *vetor*. Deve retornar 0 caso contrário. O parâmetro tam representa o tamanho do vetor.

2. Função repetido(vetor[], tam) que retorna 1 se existem algum valor repetido dentro de *vetor*. Deve retornar 0 caso contrário
3. Função maisRepete(vetor[], tam) que retorna o valor que mais se repete dentro de *vetor*.
4. Funcao ordena(vetor[], tam) que faz a ordenação do vetor de tamanho tam em ordem crescente.
5. Função para calcular o número de repetições de cada elemento.
6. Os exercícios anteriores em versão recursiva

## 2. Matrizes

Um fato a ser observado é que também podem ser declarados vetores de vetores, também conhecidos como matrizes, de um tipo de dados. Ou seja, o programador pode declarar uma matriz de inteiros, ou de números ponto-flutuante, caracteres, estruturas, etc. Um exemplo dessa declaração segue abaixo.

```
Início
   Var: mat[5][5], l, c  //uso variável l para linha e c para coluna
   l = 0
   Enquanto(l < 5) repita
      c = 0
      Enquanto(c < 5) repita
         mat[l][c] = ler()
         c = c + 1
      Fim Enquanto
      l = l + 1
   Fim Enquanto
Fim.
```

No exemplo anterior, foi declarada uma matriz de 5x5 elementos e duas dimensões e o seu conteúdo foi lido usando o teclado. Na figura a seguir temos o formato da matriz do exemplo anterior.

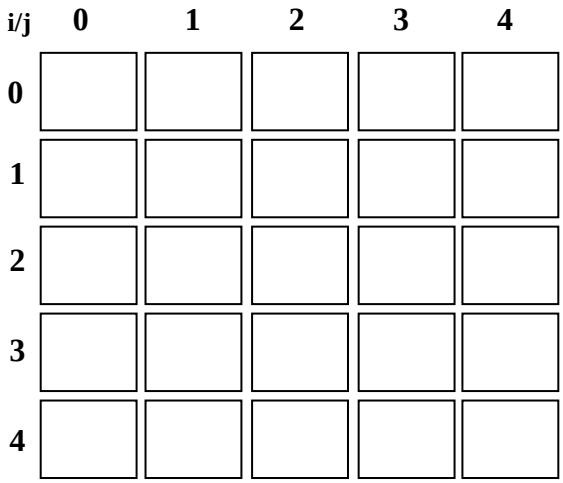



**Figura 4: Exemplo de uma matriz de 5x5 elementos**

Observe que as regras de indexação são as mesmas que para os vetores. Para a matriz de 5x5 elementos acima, o número de linhas, bem como o número de colunas varia entre 0 e 4, pois a matriz tem 5 linhas e 5 colunas.

O exemplo anterior demonstrou o uso de uma matriz de dimensões 5x5, mas podem ser usados quaisquer números para o número de linhas e colunas, contanto que sejam inteiros e maiores que zero. Como exercício, repita essa mesma solução usando laços de repetição para.

```
Início
   Var: mat[3][300], l, c, acc
   l = 0
   acc = 0
   Enquanto(l < 3) repita
      c = 0
      Enquanto(c < 300) repita
         mat[l][c] = ler()    #Lê todos os valores para a matriz.
         c = c + 1
      Fim Enquanto
      l = l + 1
   Fim Enquanto

   l = 0
   Enquanto(l < 3) repita
      c = 0
      Enquanto(c < 300) repita
         acc = acc + mat[l][c]
         c = c + 1
      Fim Enquanto
      l = l + 1
   Fim Enquanto

   acc = acc / 900    #Calcula a média de todos os valores da matriz.
   Escreva("A média dos valores da matriz é: " + acc)
Fim.
```

Matrizes são muito utilizadas, pois alguns problemas e tipos de dados são representados mais intuitivamente com esse formato. Por exemplo, o formato mais intuitivo para uma tabela de dados é uma matriz de duas dimensões, enquanto os cubos de um rubik's cube são melhor representados por uma matriz de três dimensões.

As matrizes podem possuir um número arbitrário de dimensões, não se restringindo a apenas 1 ou 2 dimensões. Mas deve ser levado em consideração que quanto maior o número de dimensões da matriz, maior a complexidade do algoritmo, pois é necessário um número maior de laços encadeados para percorrê-la.
O exemplo a seguir ilustra a busca do maior valor em uma matriz de 3 dimensões.

```
Início
   Var: mat[10][10][10], maior, i, j, k
   i = 0
   Enquanto(i < 10) repita
      j = 0
      Enquanto(j < 10) repita
         k = 0
         Enquanto(k < 10) repita
            mat[i][j][k] = ler()
            k = k + 1
         Fim Enquanto
         j = j + 1
      Fim Enquanto
      i = i + 1
   Fim Enquanto

   maior = mat[0][0][0]

   i = 0
   Enquanto(i < 10) repita
```

```
      j = 0
      Enquanto(j < 10) repita
         k = 0
         Enquanto(k < 10) repita
            Se(mat[i][j][k] > maior) repita
                maior = mat[i][j][k]
            Fim Se
            k = k + 1
         Fim Enquanto
         j = j + 1
      Fim Enquanto
      i = i + 1
   Fim Enquanto

   Escrever(maior)
Fim.
```

Note que no exemplo anterior, a matriz possui 1.000 elementos, pois cada uma de suas dimensões possui 10 elementos e cada elemento é formado por outros 10 elementos que são formados por outros 10 elementos, assim por diante. Isso faz com que seja muito fácil possuir uma matriz com mais dados que a memória seja capaz de suportar.

O seguinte exemplo ilustra a passagem de matrizes (e vetores) para funções.

```
Inicio
    Var: mat[3][4], vet[10], ret[10]
    ret = Func(mat, vet)
Fim.

Funcao Func(m[][], v[])
   Var: aux[10], i
   Para i = 0 até 9 repita
         aux[i] = v[i]
         i++
   Fim para
   Retorna aux
Fim Funcao
```

Como **exercício**, elabore solução para, dada uma matriz de dimensão NxM (linhas e colunas) de números inteiros, dizer se existirem duas colunas cuja soma de todos os valores seja igual.

# 3. Estruturas de Dados Heterogêneos

Conforme os programas se tornam mais complexos, é desejável que seja possível agrupar de alguma forma dados relacionados. Considere o seguinte trecho de código

```
Início
   Var: nomeAluno1, nomeAluno2, nomeAluno3, nomeAluno4, nomeAluno5
   Var: notaAluno1, notaAluno2, notaAluno3, notaAluno4, notaAluno5
   Var: mediaTurma = 0

   Ler nomeAluno1, nomeAluno2, nomeAluno3, nomeAluno4, nomeAluno5
   Ler notaAluno1, notaAluno2, notaAluno3, notaAluno4, notaAluno5

   mediaTurma=(notaAluno1+notaAluno2+notaAluno3+notaAluno4+notaAluno5)/5
   Se mediaTurma >= 7 então
```

```
      Escreva "A turma está acima da média"
   Senão então
     Escreva "A turma possui média abaixo de 7"
   Fim se
Fim.
```

O exemplo acima ilustra a leitura dos nomes de cinco alunos e suas notas, então ele calcula a média desses alunos e se ela for maior ou igual a 7 uma mensagem informativa é exibida, senão uma mensagem de que a turma possui média abaixo de 7 é exibida.

Agora imagine seja necessário adicionar também o número de faltas de cada aluno, isso implicaria na criação de cinco novos registros, cada um para armazenar o número de faltas de cada aluno. Imagine a mesma situação na universidade, uma variável "`nomeAluno`", bem como uma variável "`notaAluno`" para cada um dos 19000 alunos.

Seria mais conveniente se pudéssemos agrupar o nome do aluno com a sua nota, criando um único tipo de dado que encapsule as informações de cada aluno.

O mecanismo mais utilizado para agrupar esses dados é chamado de **Estrutura** e um exemplo da declaração de uma estrutura segue abaixo.

```
Estrutura Aluno
   Var: nome, nota
```

Ou seja, cada aluno possui um "`nome`" e uma "`nota`" associados ao seu registro. Caso seja necessário adicionar o número de faltas ao registro do aluno, basta adicionar uma variável "`nrFaltas`" à estrutura aluno.

É importante ressaltar que os campos (ou membros) em uma estrutura não precisam ser do mesmo tipo, ou seja, um campo pode ser um número real, outro pode ser um número inteiro, ou um conjunto de caracteres, etc. Isso dá flexibilidade às estruturas, pois permite que qualquer tipo de variável, até mesmo outras estruturas, sejam membros de uma estrutura de dados mais complexa.

A vantagem principal das estruturas é que elas permitem encapsular os dados relevantes a uma determinada entidade, por exemplo, nome e notas de um aluno, informações básicas de um computador (quantidade de memória, número de HDs presentes, número de GPUs, CPUs, etc), dados sobre os clientes e fornecedores de uma empresa, etc. Cada um desses dados dentro de uma "caixa".

| | |
|---|---|
| `Estrutura Aluno`<br>`  Var: nome, nota, faltas` | `Estrutura Turma`<br>`  Var: alunos[50], media, período,`<br>`  professor` |
| `Estrutura HD`<br>`  Var: nrCilindros, nrTrilhas,`<br>`  nrSetores, tamTotal` | `Estrutura Computador`<br>`  Var: memoria, CPUs, GPUs, HDs,`<br>`  idade, nrUsuarios` |

Agora, como usar a estrutura aluno, e como acessar o nome e a nota de cada registro? O exemplo abaixo responde a essas perguntas. Utiliza-se, neste texto, a convenção **Var NomeEstrutura: variáveis** para informar que as variáveis criadas são de um tipo estrutura. No seguinte exemplo, al1, al2, etc são do tipo Estrutura Aluno.

```
Início
   Var Aluno: al1, al2, al3, al4, al5
   Var: mediaTurma = 0
```

```
   Ler al1.nome, al2.nome, al3.nome, al4.nome, al5.nome
   Ler al1.nota, al2.nota, al3.nota, al4.nota, al5.nota

   mediaTurma = (al1.nota+al2.nota+al3.nota+al4.nota+al5.nota)/5
   Se mediaTurma >= 7 então
      Escreva "A turma está acima da média"
   Senão então
      Escreva "A turma possui média abaixo de 7"
   Fim se
Fim.
```

Será adotada uma forma mais compacta de leitura dos dados para a estrutura, considere o seguinte trecho de código idêntico ao trecho acima.

```
Início
   Var Aluno: al1, al2, al3, al4, al5
   Var: mediaTurma = 0

   Ler al1, al2, al3, al4, al5

   mediaTurma = (al1.nota+al2.nota+al3.nota+al4.nota+al5.nota)/5
   Se mediaTurma >= 7 então
      Escreva "A turma está acima da média"
   Senão então
      Escreva "A turma possui média abaixo de 7"
   Fim se
Fim.
```

O comando "`Ler`" executará a leitura de todos os dados da estrutura, ou seja, para as variáveis `al1`, `al2`, `al3`, `al4` e `al5` serão lidos o nome e a nota correspondentes a cada uma.

Para acessar um membro de uma estrutura basta usar "`nomeDaVariavel.nomeDoCampo`". Esse formato funciona para qualquer estrutura. No caso do exemplo acima, é fácil adicionar um campo "`nrFaltas`", basta adicioná-lo à estrutura `Aluno`. O exemplo abaixo demonstra a nova estrutura `Aluno`.

```
Estrutura Aluno
   Var: nome, nota, nrFaltas
```

Usando o mesmo conceito, é possível adicionar vários campos à estrutura `Aluno`, como o número de vezes que o aluno foi expulso de sala, o número de vezes que repetiu o 7º ano, quantas vezes ele faltou dizendo que estava doente, etc.

Outro conceito útil é o aninhamento de estruturas, ou seja, declarar uma estrutura que possua outra estrutura como membro. Veja o exemplo a seguir, nele definimos a estrutura `Turma` contento vários alunos e a média de notas deles. Para este exemplo nós continuamos usando a estrutura `Aluno` definida anteriormente.

```
Estrutura Turma
   Var Aluno: aluno1, aluno2, aluno3, aluno4, aluno5
   Var: mediaTurma
```

É possível notar que a estrutura `Turma` possui cinco alunos e a média da turma como membros. Cada aluno é uma estrutura do tipo `Aluno` definida nos exemplos anteriores. A variável `mediaTurma` é uma variável ponto-flutuante.

Mas como acessar o `nome` e a `nota` do `aluno1` na estrutura `Turma`? O exemplo abaixo demonstra esse caso.

```
Início
   Var Turma: turma
   Ler turma.aluno1.nome, turma.aluno1.nota

   Se turma.aluno1.nota >= 7 então
      Escreva "O aluno1 está acima da média"
   Senão
      Escreva "O aluno1 está em exame"
   Fim se
Fim.
```

### 3.1 Vetor de Estruturas

É interessante perceber que é possível usar vetores dentro de estruturas, o que facilita o encapsulamento de grandes conjuntos de dados. A nossa estrutura `Turma` pode ser reescrita da seguinte forma.

```
Estrutura Turma
   Var Aluno: alunos[50]  #alunos é um vetor de estrutura do tipo Aluno
   Var: mediaTurma
```

Observe que essa nova forma de escrever a estrutura `Turma` permitiu que o número de alunos fosse aumentado de 5 para 50 sem esforço, bastou aumentar o número de posições do vetor de alunos. Um algoritmo que calcule a média de todos os alunos da turma e guarde-a na variável `mediaTurma` da estrutura `Turma` segue abaixo.

```
Início
   Var Turma: turma    //não confundir Turma com turma
   Var: i
   Para i=0 até 49 repita
      Ler turma.alunos[i]
      i++
   Fim para

   Para i=0 até 49 repita
      turma.mediaTurma = turma.mediaTurma + turma.alunos[i].nota
      i++
   Fim para
   turma.mediaTurma = turma.mediaTurma / 50
   Escreva("A média de notas da turma é: " + turma.mediaTurma)
Fim.
```

Pode-se também criar um vetor de turmas, para representar, por exemplo, uma universidade, como no seguinte exemplo. Observa-se que neste caso tem-se uma **matriz de dados**, onde cada linha pode representar uma turma e cada coluna um aluno.

```
Início
   Var Turma: turma[100]
   Var: t, a
   Para t=0 até 99 repita  → para cada turma. Usei a variável t de turma
      Para a=0 até 49 repita   → para cada aluno de cada turma
         Ler turma[t].alunos[a]
         a++
      Fim Para
```

```
      t++
   Fim para
Fim.
```

Como **exercício**, defina estruturas para representar uma universidade composta por várias turmas. Após, repita um programa para fazer a declaração de um vetor de universidades e sua respectiva inicialização.

### 3.2 Matriz de Estruturas

Da mesma forma, pode-se criar matrizes de estruturas, como no seguinte exemplo, onde `ponto` representa uma matriz de 10x10 pontos. Em termos espaciais, temos a representação de um espaço 3D.

```
Estrutura Ponto
   Var: x, y, z
```

```
Início
   Var Ponto: ponto[10][10] //não confundir Ponto com ponto
   Var: lin, col
   Para lin=0 até 9 repita
      Para col=0 até 9 repita
         ponto[lin][col] = ler() #le as 3 coordenadas de cada ponto
         col = col + 1
      Fim para
      lin = lin + 1
   Fim para
Fim.
```

De forma mais detalhada, a leitura poderia ser realizada da seguinte forma:

```
Início
   Var Ponto: ponto[10][10]
   Var: lin, col
   Para lin=0 até 9 repita
      Para col=0 até 9 repita
         ponto[lin][col].x = ler()
         ponto[lin][col].y = ler()
         ponto[lin][col].z = ler()
         col = col + 1
      Fim para
      lin = lin + 1
   Fim para
Fim.
```

Definindo a estrutura ponto como um vetor de números temos:

```
Estrutura Ponto
   Var: coord[3]
```

```
Início
   Var Ponto: ponto[10][10]
   Var: lin, col, idx
   Para lin=0 até 9 repita
      Para col=0 até 9 repita
```

```
         Para idx=0 até 2 repita
            ponto[lin][col].coord[idx] = ler()
            idx = idx + 1
         Fim Para
         col = col + 1
      Fim para
      lin = lin + 1
   Fim para
Fim.
```

Pode-se fazer a passagem de estruturas ou vetores de estruturas para funções, como mostrado no seguinte exemplo.

```
Estrutura Ponto
   Var: coord[3]

Início
   Var Ponto: p[10][10]
   imprime(p, 10)
Fim.
```

```
Funcao imprime(Ponto: p[][], dim)
   Var: l, c, idx
   Para(l=0 ate dim-1)
      Para(c=0 ate dim-1)
         Para(idx=0 ate 2)
             Escreva(p[l][c].coord[idx]
             idx = idx + 1
         Fim para
         c = c + 1
      Fim para
      l = l + 1
   Fim para
Fim Funcao
```