Unidade IV

7 ESTRUTURAS DE ARMAZENAMENTO DE DADOS

As variáveis simples alocam um endereço de memória para armazenar um único dado.

Lembrete

A forma mais simples de armazenar um dado na elaboração de um algoritmo é utilizar uma variável ou uma constante.

Em algumas situações, a lógica requer estruturas capazes de armazenar grandes quantidades de dados durante a execução do código, especialmente quando podem ser identificadas com o mesmo nome.

Duas estruturas para armazenamento de dados durante a execução do algoritmo são os vetores e as matrizes, também conhecidas como **estruturas de dados homogêneas** porque guardam vários dados do mesmo tipo em uma única variável, ou seja, usam um único identificador para várias posições de memória.

As estruturas de dados homogêneas são representadas por linhas e colunas que caracterizam suas dimensões como demonstrado a seguir:

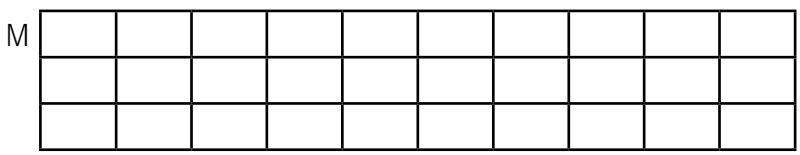

Figura 143 – Representação de uma matriz

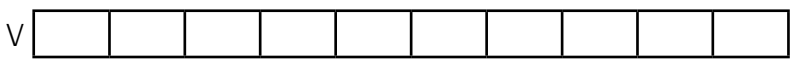

Figura 144 – Representação de um vetor

As dimensões da matriz são definidas em termos de linhas e colunas. A matriz M da figura 143 é uma matriz 3x10, ou seja, com 3 linhas e 10 colunas, capaz de armazenar até trinta dados do mesmo tipo.

O vetor pode ser entendido como uma matriz unidimensional, pois sempre terá apenas uma única linha e uma ou mais colunas. As matrizes unidimensionais são sempre referenciadas como vetores. Um vetor tem dimensão 1xn (um por n) e tem a sua declaração simplificada.

Cada posição da matriz ou do vetor deve ser referenciada por um índice. Vetores possuem índices que representam as colunas e as matrizes possuem índices para linhas e colunas.

7.1 Vetores

Um vetor, também conhecido como matriz unidimensional, é uma alocação de memória com um identificador único, indexada e dimensionada com um tamanho específico.

A variável V da figura anterior possui 10 endereços de memória referenciados por um único identificador e é capaz de armazenar 10 diferentes valores, todos do mesmo tipo, ou seja, ou todos os valores são inteiros, reais, caracteres ou lógicos. Vetores e matrizes possuem índices que fazem referência às posições de memória alocadas para a variável a fim de acessar dados nela armazenados.

7.1.1 Declaração de variáveis compostas homogêneas unidimensionais: vetores

Ao declarar uma variável como um vetor, deve-se informar o tamanho do vetor.

```
    var
1.  x[7] : inteiro
2.  vint[5] = {10, 23, 45, 65, 18} : inteiro
3.  vreal[] = {10.5, 20.5, 4.5, 6.5, 1.8} : real
4.  y: vetor [0..3] de inteiro
```

Figura 145 – Formas de declarar vetores

A declaração do vetor x apresentada na linha 1:

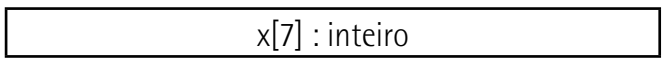

Figura 146

aloca 10 endereços de memória do tamanho de um tipo inteiro e pode ser representada na figura seguinte. Nessa declaração, x é uma variável composta homogênea capaz de armazenar até 7 valores do tipo inteiro, mas nenhum valor lhe foi atribuído.

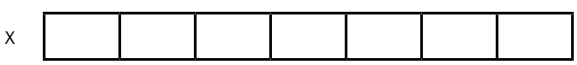

Figura 147

Na linha 2, o vetor é declarado e inicializado ao mesmo tempo. A variável **vint** é um vetor de inteiros de tamanho 5, cujos valores lhes foram atribuídos no ato da declaração. Após a execução da linha 2, o vetor não estará vazio e os valores serão alocados na ordem em que foram listados. O vetor contém um conjunto finito de valores e esse conjunto poderá ser vazio, unitário ou com um número definido de valores.

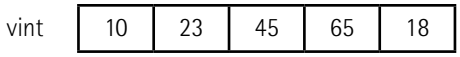

Figura 148

Na linha 3, o vetor é declarado e inicializado, contudo o tamanho não é explicitamente definido. O tamanho do vetor será interpretado pelo compilador como o tamanho da lista. Neste caso, **vreal[] = {10.5, 20.5, 7.9, 4.5, 6.5, 1.8}**, a lista contém seis valores do tipo real, logo, serão alocados seis endereços de memória, todos identificados por vreal. Ao tipo real é reservado o dobro de memória destinada ao tipo inteiro, por isso a representação com quadrados maiores.

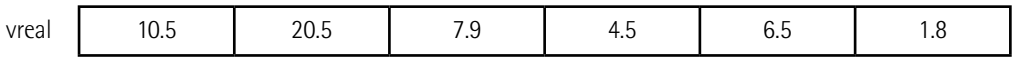

Figura 149

A linha 4 contém a única declaração aceita pelo VisuAlg, utilizada nos exemplos deste livro-texto. As demais são sintaxes presentes nas linguagens de programação. No VisuAlg a declaração define o intervalo de índices aceitos no vetor. Neste caso,

```
y: vetor [0..3] de inteiro
```

Figura 150

define que y é um vetor de inteiros cujos índices são identificados por números no intervalo de 0 a 3, ou seja, a primeira posição do vetor será referenciada pelo índice 0 e a última pelo índice 3, o que representa um vetor com 4 posições. Os números abaixo do vetor representam os índices.

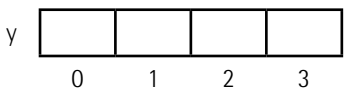

Figura 151

O primeiro índice de vetores e matrizes é sempre zero, ou seja, a primeira linha e a primeira coluna correspondem ao índice 0 e se o vetor possui 10 colunas, os índices variam de 0 a 9. O mesmo conceito é aplicado às matrizes, ou seja, a matriz M da figura 143 tem as linhas de 0 a 2 e as colunas de 0 a 9.

7.1.2 Atribuindo valores a vetores

Ao vetor poderão ser atribuídos valores no ato da declaração, como apresentado nas linhas 2 e 3 da figura 145. Adicionalmente, podem ser atribuídas constantes de maneira manual, ou por força bruta, simplesmente informando em qual posição do vetor o dado deve ser armazenado, conforme a linha 2 da figura a seguir, ou por comando de entrada de dados com leitura da variável via teclado.

```
1.  Inicio
2.  x[0]<-24
3.  leia(vint[4])
4.  para i de 0 até 3 passo 1 faça
5.          y[i]<-i
6.      fimpara
7.  Fimalgoritmo
```

Figura 152 – Formas de atribuição com vetores

Na linha 2, o comando armazenará, no vetor x, na posição 1, o valor 24. Esse vetor foi declarado com 7 posições e, após a atribuição, ficará da seguinte forma:

Figura 153

As demais posições poderão ser preenchidas fazendo referência ao índice, da seguinte forma:

x[0]<-24

x[1]<-30

x[2]<-36

x[3]<-42

x[4]<-48

x[5]<-54

x[6]<-60

e o resultado é o vetor **x** totalmente preenchido, conforme ilustrado a seguir:

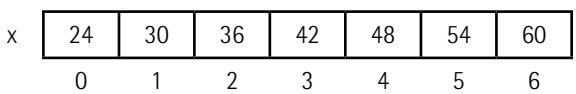

Figura 154

Se o vetor tivesse 100 posições, seriam 100 linhas de atribuições e isso não seria uma boa prática. Como o índice do vetor varia de maneira incremental, o algoritmo pode ser projetado para que a atribuição ocorra dentro de um laço de repetição, da forma apresentada nas linhas de 5 a 7 da figura 152. Nesse exemplo, a variável **i** tem o propósito de variar o índice do vetor.

A cada iteração do laço PARA um índice do vetor é incrementado e a instrução dentro do laço fará a entrada de dados. Para um vetor com 100, 1.000 ou mais posições, a codificação não aumenta, apenas a quantidade de vezes que o laço será repetido, como mostrado no código da figura a seguir:

```
para i de 1 até 100 passo 1 faça
    escreva("V[", i, "] = ")
    leia(V[i])
fimpara
```

Figura 155

7.1.3 Pesquisando um determinado elemento no vetor

Uma funcionalidade muito importante é pesquisar se um determinado valor existe no vetor. Se um vetor armazenasse uma lista de nomes de amigos para o churrasco de domingo, a pesquisa responderia à pergunta: **"Fulano está na lista?"**. O trecho de código a seguir pesquisa um valor no vetor e, se o encontrar, retornará o índice do vetor, se não, retornará -1.

```
1.  Algoritmo "Pesquisa elemento num vetor"
2.  Var
3.          i, x, vet: vetor [0..9] de inteiro
4.
5.  Inicio
6.          //preenchendo o vetor
7.          para i de 0 até 9 passo 1 faça
8.            escreva("V[", i, "] = ")
9.            leia(V[i])
10.         fimpara
11.
12.         //pesquisando um dado elemento no vetor
13.         escreva("Qual valor deseja pesquisar: ")
14.         leia(x)
15.         para i de 0 até 9 passo 1 faça
16.           se (vet[i]=x)
17.                                   escreva("Valor encontrado na posição ", i)
18.                       fimse
19.         fimpara
20. Fimalgoritmo
```

Figura 156 – Formas de atribuição com vetores

Nas linhas de 7 a 10 do algoritmo da figura anterior, o vetor será preenchido com dados informados via teclado. Nas linhas de 12 a 19, o valor x com entrada na linha 14 será pesquisado no vetor. Será

escrita a mensagem **Valor encontrado na posição i** quando o valor for encontrado. Se a execução do algoritmo for finalizada e nenhuma mensagem for exibida, significa que o valor da variável x não existe no vetor.

7.1.4 Escrevendo os elementos de um vetor

A estrutura de repetição da figura a seguir, nas linhas 13 a 14, mostra todos os elementos do vetor **vet** preenchido pelo usuário nos comandos das linhas de 7 a 10.

```
1.   Algoritmo "Mostrando os elementos de um vetor"
2.   Var
3.         i, vet: vetor [0..9] de inteiro
4.
5.   Inicio
6.         //preenchendo o vetor
7.         para i de 0 ate 9 passo 1 faca
8.           escreva("Vet[", i, "] = ")
9.           leia(Vet[i])
10.        fimpara
11.
12.        //exibindo os elementos do vetor
13.        para i de 0 até 9 passo 1 faca
14.          escreva("Vet[", i, "] = ", vet[i])
15.        fimpara
16.  Fimalgoritmo
```

Figura 157 – Formas de atribuição com vetores

No exemplo da figura seguinte, para cada posição do vetor x está sendo atribuído o valor da variável contadora do laço **for**, a variável **i + 1**.

```
1.   Algoritmo "Exemplo Vetor"
2.   Var
3.     //declaracao das variaveis
4.     x: vetor [0..3] de inteiro
5.     i: inteiro
6.
7.   Inicio
8.     //Laco de repeticao para preencher o vetor.
9.     para i de 0 ate 3 passo 1 faca
10.      ...<COMANDOS>...
11.      x[i] <- i + 1
12.      ...<COMANDOS>...
13.    Fimpara
14.  Fimalgoritmo
```

Figura 158 – Exemplo de pseudocódigo para criar vetor

Primeiro é necessário criar a variável do vetor, no caso, **x**, junto com ela informamos a quantidade de posições que ela terá, assim determinamos um valor finito a ela: **vetor[0..3]**, isso indica que o vetor terá 4 posições iniciando em 0 e terminando em 3. O passo seguinte é informar o tipo de dado que o vetor armazenará.

Depois de criar o vetor através de um laço de repetição, usa-se a instrução PARA. A instrução seguinte **x[i] <- i + 1** faz com que o valor de **i + 1** seja armazenado na posição **x[i]**. Como **i** é um contador, em cada passo será somado o valor 1 a cada repetição **para i de 0 ate 3 +1 faca**.

Observação

A palavra "faca" na instrução **para i de 0 ate 3 +1 faca** significa "faça".

Exemplo 1

Elabore 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.

Solução

```
1.  Algoritmo "Vetor_Tabuada"
2.  Var
3.    //Declaracoes das variaveis
4.    x: vetor [0..9] de inteiro
5.    i, numero: inteiro
6.
7.  Inicio
8.    //Entrada
9.    escreval("Calculadora de Tabuada.")
10.   escreval("Digite o valor do número que deseja calcular: ")
11.   leia(numero)
12.   //Laco para calcular e armazenar dados no vetor
13.   para i de 0 ate 9 passo 1 faca
14.     x[i] <- numero * (i + 1)
15.     escreval(numero, " * ", i+1, " = ", x[i])
16.   fimpara
17.
18.  Fimalgoritmo
```

Figura 159 – Pseudocódigo para calcular tabuada e armazenar em um vetor

A criação do vetor é feita na declaração da variável x como um vetor de inteiros, junto a ela informamos a quantidade de posições que ela terá, assim determinamos um valor finito a ela: **vetor[0..9]**, isso indica que o vetor terá 10 posições iniciando em 0 e terminando em 9. Depois é informado o tipo de dado que o vetor armazenará.

O próximo passo é preencher o vetor usando um laço de repetição. Neste exemplo, foi usada a instrução PARA, que, dentro do bloco **x[i] <- num * (i + 1)** faz com que o valor de **num * (i + 1)** seja armazenado na posição **x[i]**. Como **i** é um contador, em cada passo será somado o valor 1 a cada repetição **para i de 0 ate 9 +1 faca**. A linha seguinte formata o texto que será exibido e exibe o resultado do cálculo.

O próximo exemplo é mais elaborado e é voltado a converter a temperatura e armazenar os dados no vetor. Nele, o algoritmo converte graus Celsius em graus Fahrenheit e em graus Kelvin. Para fazê-lo, serão criados três vetores para armazenar cada uma das conversões.

Exemplo 2

Desenvolva um algoritmo que receba 15 valores referentes às temperaturas em graus Celsius e armazene-as 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.

Solução

```
1.  Algoritmo "Vetor_Conversor_de_Temperatura"
2.  Var
3.    //Declaracoes das variaveis
4.    x: vetor [0..14] de real
5.    y: vetor [0..14] de real
6.    z: vetor [0..14] de real
7.    i: inteiro
8.
9.  Inicio
10.   escreval("Conversor de Temperatura.")
11.   //Laco de repeticao para criar o vetor, calcular e armazenar.
12.   para i de 0 ate 14 passo 1 faca
13.     //Entrada de dados.
14.     leia(x[i])
15.     //Conversao para Graus Fahrenheit.
16.     y[i] <- (x[i]*9/5) + 32
17.     //Conversao para Graus Kelvin.
18.     z[i] <- x[i]+ 273.15
19.     //Exibindo resultados.
20.     escreval(x[i], " Graus Celsius")
21.     escreval(y[i], " Graus Fahrenheit")
22.     escreval(z[i], " Graus Kelvin")
23.   fimpara
24.
25. Fimalgoritmo
```

Figura 160 – Pseudocódigo converte temperatura e armazena dados nos vetores

Nesse algoritmo, foram criados 3 vetores, declarados como **x**, **y** e **z**. Na declaração, devem ser informadas as quantidades de posições que cada um terá, assim é determinado um valor finito com 15 posições com o intervalo declarado dentro dos colchetes: **vetor[0..14]**.

Isso indica que o vetor terá 15 posições iniciando em 0 e terminando em 14. Depois é informado o tipo de dado que o vetor armazenará. Em seguida é criado o vetor através de um laço de repetição: foi usada a instrução PARA. A instrução seguinte **leia(x[i])** efetuará a leitura de 15 números informados pelo usuário, em seguida **y[i] <- (x[i]*9/5) + 32** e **z[i] <- x[i]+ 273.15** farão a conversão de graus Celsius para graus Fahrenheit e para graus Kelvin, respectivamente. Por fim, os resultados serão exibidos.

A partir dos exemplos citados, pode-se concluir que vetores são ideais para armazenar uma certa quantidade de dados e que podemos utilizar não apenas um, mas muitos vetores.

Saiba mais

Leia mais sobre vetores no capítulo:

ASCENCIO, A. F. G.; CAMPOS, E. A. V. Vetores. *In:* ASCENCIO, A. F. G.; CAMPOS, E. A. V. *Fundamentos da programação de computadores*. São Paulo: Pearson, 2012.

7.2 Matrizes

Uma matriz é uma alocação de memória com um identificador único, indexada em termos de linhas e colunas, daí o conceito de estrutura de dados composta, homogênea e bidimensional, cujos valores devem ser específicos.

A variável **m** da figura 143 possui 30 endereços de memória referenciados por um único identificador e capaz de armazenar 30 diferentes valores, todos do mesmo tipo. Essa matriz possui índices distintos para linhas e colunas.

Lembrete

Afirmar que todos os valores são do mesmo tipo quer dizer que todos os valores são inteiros, reais, caracteres ou lógicos.

A seguir serão apresentadas as operações mais comuns com matrizes.

7.2.1 Declaração de variáveis compostas homogêneas bidimensionais: matrizes

Como exemplo de uma matriz homogênea, pode-se pensar num tabuleiro tal como o tabuleiro de jogo de xadrez, que tem a dimensão 8x8, totalizando 64 posições, conforme ilustrado na figura a seguir:

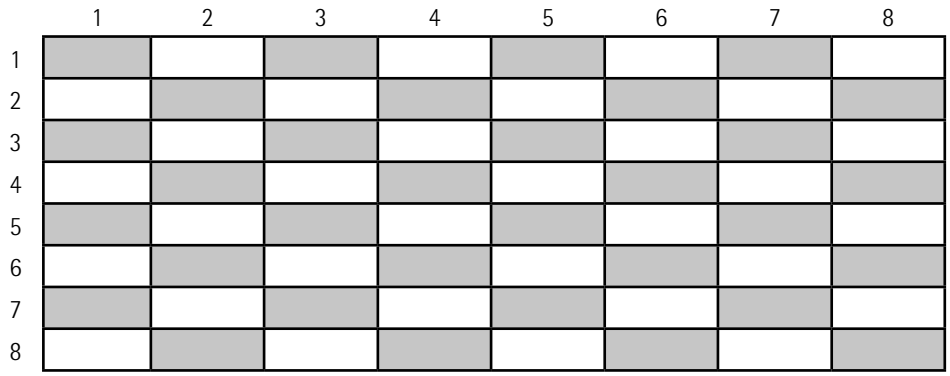

Figura 161 – Exemplo de matriz homogênea – tabuleiro de xadrez

Cada linha ou coluna da matriz é indexada com números inteiros que, na lógica do algoritmo, devem ser computados por variáveis que representam linhas ou colunas. Ao declarar uma variável como uma matriz, devem-se informar as dimensões, ou seja, o número de linhas e de colunas conforme os exemplos da figura seguinte.

```
1.  x[5][3] : inteiro
2.  tabuleiro[8][8] : caractere
3.  vint[2][3] = {{10, 20, 30}, {40, 50,60}} : inteiro
4.  vreal[][] = {{10, 20, 30}, {40, 50,60}} : inteiro
5.  z: matriz [1..10, 1..2] de inteiro
```

Figura 162 – Formas de declarar matrizes

A declaração da matriz x apresentada na linha 2 possui 5 linhas e 3 colunas, foi declarada e não inicializada, logo a alocação de memória é feita, mas está vazia, conforme ilustrado a seguir.

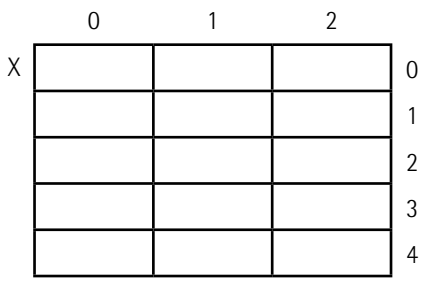

Figura 163

A linha 2 **tabuleiro[8][8]: caractere** declara o tabuleiro como uma matriz quadrada 8x8, ou seja, 8 linhas e 8 colunas.

A linha 3 da figura 162 mostra como declarar e inicializar uma matriz ao mesmo tempo, **vint[2][3] = {{10, 20, 30}, {40, 50,60}) : inteiro**. Uma matriz armazena um conjunto de valores ou vários vetores. Cada linha é separada por {} e os elementos da linha são separados por vírgula. A declaração e inicialização simultânea consiste de um conjunto de conjuntos de dados.

A declaração da linha 4 não tem especificação do tamanho da matriz, logo, será alocada memória suficiente para armazenar os dados já estabelecidos.

```
Vreal[][] = {{10, 20, 30}, {40, 50,60}) : inteiro
```

Figura 164

A linha 6 apresenta a única sintaxe de declaração aceita no VisuAlg, a qual será utilizada em nossos exemplos. O pseudocódigo para criar a matriz da figura 161 é apresentado na figura subsequente.

```
1.  Algoritmo "Tabuleiro de Xadrez"
2.  Var
3.    //declaracao das variaveis
4.    matriz_tabuleiro: vetor[0..7,0..7] de inteiro
5.    i, j: inteiro
6.
7.  procedimento tabuleiro()
8.  Inicio
9.    //Laço de repetição para criar o Eixo X.
10.   para i de 0 ate 7 passo 1 faca
11.     //Laço de repetição para criar o Eixo Y.
12.     para j de 0 ate 7 passo 1 faca
13.       matriz_tabuleiro[i,j] <- i + j
14.       //Saida para exibir cada uma das posições da MATRIZ.
15.       escreval("Coluna: ",m, " = ", matriz_tabuleiro[i,j])
16.     fimpara
17.   fimpara
18. fimprocedimento
19.
20. Inicio
21.
22. //Voz de Comando
23.   tabuleiro()
24.
25. Fimalgoritmo
```

Figura 165 – Exemplo de algoritmo para montar um tabuleiro de xadrez

7.2.2 Atribuindo valores a matrizes

À matriz poderão ser conferidos valores no ato da declaração, conforme apresentado nas linhas 3 e 4 da figura 162. Adicionalmente, podem ser atribuídas constantes de maneira manual, ou por força bruta, simplesmente informando em qual posição da matriz o dado deve ser armazenado, conforme a linha 1 da figura seguinte, ou por comando de entrada de dados com leitura da variável via teclado, conforme a linha 3.

```
1.  Inicio
2.  x[0,1]<-24
3.  leia(tabuleiro[4,4])
4.  para linha de 0 até 9 passo 1 faça
5.          para coluna de 0 até 3 passo 1 faca
6.                  y[linha, coluna]<-i
7.          fimpara
8.  Fimalgoritmo
```

Figura 166 – Formas de atribuição em matriz

As operações mais comuns para manipular dados em vetores ou matrizes são:

- preencher a matriz;
- escrever os elementos da matriz;
- pesquisar se um determinado elemento existe no vetor ou na matriz.

Essas operações podem percorrer todas as posições das estruturas de dados vetores ou matrizes e a implementação requer a aplicação de estruturas de repetição encadeadas, visto que precisará variar linha e coluna.

A declaração da matriz x na figura seguinte, **x[5,3] : inteiro**, estabelece 5 linhas e 3 colunas. A atribuição **x[0,1]<-24** força a atribuição do valor 24 na posição linha 0, coluna 1, conforme imagem a seguir:

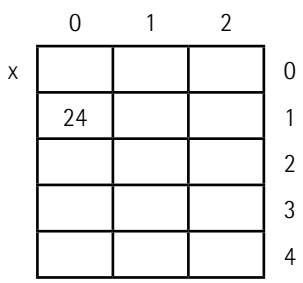

Figura 167 – Matriz com 3 colunas e 5 linhas após a execução do comando x[0,1]<-24

A atribuição da linha 3 da figura 162 preenche a matriz, conforme demonstra a próxima figura:

	0	1	2	
vint	10	20	30	0
	40	50	60	1

Figura 168 – Matriz com 3 colunas e 2 linhas após a execução do comando

vint[2][3] = {{10, 20, 30}, {40, 50,60}) : inteiro.

Figura 169

A atribuição da linha 4 da figura 162 preenche a matriz:

	0	1	
Vreal	10.5	20.3	0
	30.0	40.9	1
	50.0	60.0	2

Figura 170 – Matriz com 2 colunas e 3 linhas após a execução do comando

vreal[][] = {{10.5, 20.3}, {30.0, 40.9},{50.0, 60.0}) : inteiro

Figura 171

A atribuição manual, posição por posição, seria:

Tabuleiro[0][0] <- "Torre"

Tabuleiro[0][1] <- "Cavalo"

Tabuleiro[0][2] <- "Bispo"

Tabuleiro[0][3] <- "Dama"

Tabuleiro[0][4] <- "Rei"

Tabuleiro[0][5] <- "Bispo"

Tabuleiro[0][6] <- "Cavalo"

Tabuleiro[0][7] <- "Torre"

Para fazer atribuições de maneira mais automática, o uso de estruturas de repetição é imprescindível. A linha dos piões no tabuleiro de xadrez será realizada no seguinte laço PARA:

```
para o de 0 até 7 passo 1 faça
        tabuleiro[1][i] <- "Peão"
        tabuleiro[6][i] <- "Peão"
fimpara
```

Figura 172

Nesse laço, as linhas 1 e 6 serão preenchidas em todas as colunas com "pião". Apenas um laço foi necessário, porque as linhas foram fixadas na referência da coluna na atribuição da matriz tabuleiro.

Observação

Os exemplos da figura 162 ilustrados nas figuras de 167 a 172 não funcionam no VisuAlg. Para executar, será necessário traduzir para alguma linguagem de programação, tal como Java ou C/C++.

7.2.3 Pesquisando um determinado elemento na matriz

A pesquisa de dados em matrizes é tão relevante quanto a pesquisa de dados em vetores. O trecho de código seguinte pesquisa um valor no vetor e, se encontrar, escreverá a linha e a coluna da matriz.

```
1.  Algoritmo "Pesquisa elemento numa matriz"
2.  Var
3.          linha, coluna, x: inteiro
4.      M: vetor [0..4, 0..4] de inteiro
5.
6.  Inicio
7.          //preenchendo a matriz
8.          para linha de 0 até 4 passo 1 faca
9.            para coluna de 0 até 4 passo 1 faca
10.                   escreva("M[", linha, "][", coluna, "] = " )
11.                   leia(M[linha,coluna])
12.           fimpara
13.         fimpara
14.
15.         //pesquisando um dado elemento no vetor
16.         escreva("Qual valor deseja pesquisar: ")
17.         leia(x)
18.
19.         para linha de 0 até 4 passo 1 faca
20.           para coluna de 0 ate 4 passo 1 faca
21.                   se(M[linha,coluna] = x) entao
22.                   escreval("valor encontrado na linha", linha, " e ", coluna)
23.             fimse
24.           fimpara
25.         fimpara
26. Fimalgoritmo
```

Figura 173 – Pesquisando um elemento em uma matriz

O laço PARA encadeado percorrerá todas as linhas e todas as colunas da matriz, escrevendo as posições onde o valor de x for igual. Percorrer os elementos de uma matriz é uma tarefa que possui custo de processamento proporcional a n^2 no caso de uma matriz quadrada. No caso geral, é igual ao número de linhas vezes o número de colunas. O custo de processamento crescerá proporcionalmente ao tamanho da matriz.

É importante lembrar que vetores e matrizes são estruturas de dados estáticas e lineares. A principal limitação no uso de matrizes ou vetores está no fato de que uma vez declarados, seu tamanho não poderá ser alterado e, por consequência, se todas as posições não forem necessárias, haverá desperdício de memória, se houver necessidade de mais posições, não será possível aumentar o seu tamanho.

Porém, uma vantagem para o uso de matrizes e vetores é que o acesso ao dado é muito rápido quando se sabe o índice.

7.2.4 Escrevendo os elementos de uma matriz

Escrever os elementos da matriz é mostrar todos os valores dentro dela.

```
1.  Algoritmo "Mostrando os elementos de uma matriz"
2.  Var
3.          linha, coluna: inteiro
4.          M: vetor [0..9, 0..9] de inteiro
5.
6.  Inicio
7.          //preenchendo a matriz
8.          para linha de 0 até 9 passo 1 faca
9.            para coluna de 0 até 9 passo 1 faca
10.                       escreva("M[", linha, "][", coluna, "] = " )
11.                       leia(M[linha,coluna])
12.           fimpara
13.         fimpara
14.
15.         //exibindo os elementos da matriz
16.         para linha de 0 até 9 passo 1 faca
17.           para coluna de 0 até 9 passo 1 faca
18.                       escreva("M[", linha, "][", coluna, "] = ",
19.                                                 M[linha,coluna])
20.           fimpara
21.         fimpara
22. Fimalgoritmo
```

Figura 174 – Algoritmo para escrever os elementos de uma matriz

Um fator importante a observar é que o laço PARA externo controlará as linhas da matriz e o laço interno, as colunas. Para cada iteração na linha, o laço da coluna executará dez vezes.

Saiba mais

Leia mais sobre matrizes no capítulo:

ASCENCIO, A. F. G.; CAMPOS, E. A. V. Matriz. *In:* ASCENCIO, A. F. G.; CAMPOS, E. A. V. *Fundamentos da programação de computadores*. São Paulo: Pearson, 2012.

8 APLICAÇÕES COM VETORES E MATRIZES

Suponha que você tenha em uma matriz M1 5x10, compreendendo os dados de 50 peças em produção, e outra matriz M2 3x10, com os dados de outras 30 peças, e deseje juntar todos os dados numa única matriz M3 8x10, conforme representado nas figuras de 175 a 177.

M1	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50

Figura 175 – Representação do problema para concatenar duas matrizes de tamanhos diferentes: matriz M1 5x10

M2	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	9
1	9	8	7	6	5	4	3	2	1	0
2	1	2	3	4	5	6	7	8	9	9

Figura 176 – Representação do problema para concatenar duas matrizes de tamanhos diferentes: matriz M2 3x10

M3	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	9
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	1	2	3	4	5	6	7	8	9	10
6	11	12	13	14	15	16	17	18	19	20
7	21	22	23	24	25	26	27	28	29	30

Figura 177 – Representação do problema para concatenar duas matrizes de tamanhos diferentes: matriz M3 8x10

Na matriz M3, devem ser armazenados os valores de M1 e M2.

```
1.  Algoritmo "Concatenar Matrizes"
2.  var
3.    //declaracao das variaveis
4.          M1: vetor [1..5, 1..10] de inteiro
5.          M2: vetor [1..3, 1..10] de inteiro
6.          M3: vetor [1..8, 1..10] de inteiro
7.
8.          i, linha, coluna: inteiro
9.  Inicio
10.   i ← 0;
11.
12.   //preencher a matriz M1
13.   para linha de 1 a 5 passo 1 faca){
14.      para coluna de 0 a 10 passo 1 faca){
15.         M1[linha,coluna] ← i+1;
16.            i ← i+1
17.            escreva(" "
18.      fimpara
19.       escreval("  ", M1[linha,coluna]);
20.   fimpara
21.
22.   //preencher a matriz M2
23.   para linha de 1 a 3 passo 1 faca){
24.      para coluna de 0 a 10 passo 1 faca){
25.            i ← i+1
26.         M2[linha,coluna] ← i;
27.         escreval("  ", M2[linha,coluna]);
28.      fimpara
29.      escreval("\n ");
30.   fimpara
31.
32.   //preencher a Matriz M3 conforme enunciado
33.   para linha de 1 a 5 passo 1 faca){
34.      para coluna de 0 a 10 passo 1 faca){
35.         se(linha<3) entao
36.            M3[linha,coluna] = M1[linha,coluna];
37.            M3[linha+5,coluna] = M2[linha,coluna];
38.         }
39.         senao{
40.            M3[linha,coluna] = M1[linha,coluna];
41.          fimse
42.      fimpara
43.   fimpara
44.
45.   //mostrar todos os dados da matriz M3
46.   para linha de 1 a 8 passo 1 faca){
47.      para coluna de 0 a 10 passo 1 faca){
48.         escreva(M3[linha][coluna]);
49.      fimpara
50.     escreval(" ");
      fimpara
    Fimalgoritmo
```

Figura 178 – Pseudocódigo para elaborar a matriz do calendário

No algoritmo da figura anterior, são apresentados blocos de códigos para preencher as matrizes e para escrever os dados da matriz M3. Observe que as matrizes possuem tamanhos diferentes e, por esse motivo, expressões aritméticas foram inseridas para calcular o índice da matriz 3 onde cada dado deve ser organizado.

Exemplo 1

Desenvolva um algoritmo que receba 49 valores inteiros, calcule e mostre os números pares, suas posições e a soma dos números pares, bem como os números ímpares, suas posições e a quantidade de números ímpares.

```
1.    Algoritmo "VetorParesImpares"
2.    var
3.      x: vetor [0..48] de inteiro
4.      i, somap, somai: inteiro
5.    Inicio
6.      somap <-0
7.      somai <-0
8.      escreval(" ")
9.      para i de 0 ate 48 passo 1 faca
10.       leia (x[i])
11.       se(x[i] mod 2 = 0) entao
12.         somap <- somap + x[i]
13.       senao
14.         somai <- somai + x[i]
15.       fimse
16.     fimpara
17.
18.     escreval("Vetor X:")
19.     para i de 0 ate 48 passo 1 faca
20.       escreval("soma pares = ", somap)
21.       escreval("soma ímpares = ", somai)
22.
23.     fimpara
24.
25.   Fimalgoritmo
```

Figura 179 – Pseudocódigo solução com vetor

O algoritmo da figura anterior pode ser reescrito a fim de ser trabalhado com uma matriz.

```
1.  Algoritmo "MatrizParesImpares"
2.  Var
3.    x: vetor [0..6,0..6] de inteiro
4.    i, j, somap,somai: inteiro
5.  Inicio
6.    escreval("")
7.    para i de 0 ate 6 passo 1 faca
8.      para j de 0 ate 6 passo 1 faca
9.        leia (x[i,j])
10.       se(x[i,j] mod 2 = 0) entao
11.         somap <- somap + x[i,j]
12.       senao
13.         somai <- somai + x[i,j]
14.       fimse
15.     fimpara
16.   fimpara
17.   escreval("")
18.   escreval("Matriz X:")
19.   para i de 0 ate 6 passo 1 faca
20.     para j de 0 ate 6 passo 1 faca
21.       escreva(x[i,j])
22.     fimpara
23.   fimpara
24.
25. Fimalgoritmo
```

Figura 180 – Pseudocódigo solução com matriz

Exemplo 2

Um usuário precisa armazenar informações das contas de aluguel, água, luz, gás e telefone de 12 meses (janeiro a dezembro) para saber quanto ele gasta por ano. Elabore um algoritmo que armazene essas informações, calcule e exiba o valor gasto durante o ano para cada uma das contas e dê uma opção a mais para exibir todos os valores de uma única vez.

```
1.  Algoritmo "Orçamento Anual"
2.  Var
3.    opcao, opcaoConsulta, contaMes, tipoConta, a, b : inteiro
4.    valor : real
5.    valorConta: vetor [0..11,0..4] de real
6.
7.  procedimento menuPrincipal()
8.  Inicio
9.    escreva("-------------------------------------------------")
10.   escreva("|                                               |")
11.   escreva("|          Orçamento Anual                      |")
12.   escreva("|                                               |")
13.   escreva("-------------------------------------------------")
14.   escreva("|                                               |")
15.   escreva("| O que deseja fazer?                           |")
16.   escreva("| [1] Inserir dados para orçamento              |")
17.   escreva("| [2] Consultar orçamento                       |")
```

```
18.  escreva("| [3] Sair                                          |")
19.  escreva("|                                                   |")
20.  escreval("--------------------------------------------------- ")
21.  escreval("")
22. fimprocedimento
23. procedimento menuConsulta()
24.
25. Inicio
26.  escreva("-----------------------------------------------------")
27.  escreva("|                                                   |")
28.  escreva("|          Consultar Gastos                         |")
29.  escreva("|                                                   |")
30.  escreva("-----------------------------------------------------")
31.  escreva("|                                                   |")
32.  escreva("| Qual conta deseja consultar?                      |")
33.  escreva("| [1] Todas                                         |")
34.  escreva("| [2] Só Água                                       |")
35.  escreva("| [3] Só Luz                                        |")
36.  escreva("| [4] Só Gás                                        |")
37.  escreva("| [5] Só Telefone                                   |")
38.  escreva("| [6] Só Aluguel                                    |")
39.  escreva("|                                                   |")
40.  escreval("--------------------------------------------------- ")
41.  escreval("")
42. fimprocedimento
43.
44. procedimento msgErr()
45. Inicio
46.  escreva("-----------------------------------------------------")
47.  escreva("|                                                   |")
48.  escreva("|          Opção Inválida !!!                       |")
49.  escreva("|                                                   |")
50.  escreval("--------------------------------------------------- ")
51.  escreval("")
52. fimprocedimento
53.
54. funcao trocaMes(a: inteiro) : caracter
55. Var
56.  mes : caracter
57. Inicio
58.  escolha(a)
59.    caso 0
60.      mes <- "Janeiro"
61.      retorne(mes)
62.    caso 1
63.      mes <- "Fevereiro"
64.      retorne(mes)
65.    caso 2
66.      mes <- "Março"
67.      retorne(mes)
68.    caso 3
69.      mes <- "Abril"
70.      retorne(mes)
71.    caso 4
72.      mes <- "Maio"
73.      retorne(mes)
74.    caso 5
75.      mes <- "Junho"
76.      retorne(mes)
77.    caso 6
```

```
78.      mes <- "Julho"
79.      retorne(mes)
80.    caso 7
81.      mes <- "Agosto"
82.      retorne(mes)
83.    caso 8
84.      mes <- "Setembro"
85.      retorne(mes)
86.    caso 9
87.      mes <- "Outubro"
88.      retorne(mes)
89.    caso 10
90.      mes <- "Novembro"
91.      retorne(mes)
92.    caso 11
93.      mes <- "Dezembro"
94.      retorne(mes)
95.  fimescolha
96. fimfuncao
97.
98. funcao trocaConta(b: inteiro) : caracter
99. Var
100.   nomeConta : caracter
101. Inicio
102.   escolha(b)
103.     caso 0
104.       nomeConta <- "Aluguel"
105.       retorne(nomeConta)
106.     caso 1
107.       nomeConta <- "Água"
108.       retorne(nomeConta)
109.     caso 2
110.       nomeConta <- "Luz"
111.       retorne(nomeConta)
112.     caso 3
113.       nomeConta <- "Gás"
114.       retorne(nomeConta)
115.     caso 4
116.       nomeConta <- "Telefone"
117.       retorne(nomeConta)
118.   fimescolha
119. fimfuncao
120.
121. procedimento cadastroContas()
122. Inicio
123.   para contaMes de 0 ate 11 faca
124.     escreva("Informe os Gastos do Mês de ", trocaMes(contaMes), " : ")
125.     para tipoConta de 0 ate 4 faca
126.       escreva("Valor da Conta de ", trocaConta(tipoConta), " : ")
127.       leia(valor)
128.       valorConta[contaMes,tipoConta] <- valor
129.     fimpara
130.   fimpara
131. fimprocedimento
132.
133. procedimento totalGastoanoTodas()
134. Var
135.   totalGasto: vetor [0..4] de real
136.   i, j : inteiro
137. Inicio
```

```
138.   para i de 0 ate 4 faca
139.     para j de 0 ate 11 faca
140.       totalGasto[i] <- totalGasto[i] + valorConta[j,i]
141.     fimpara
142.     escreva("O valor gasto no ano com a Conta de ", trocaConta(i), " é: ",totalGasto[i])
143.   fimpara
144. fimprocedimento
145.
146. procedimento totalGastoanoAluguel()
147. Var
148.   totalAluguel : real
149.   i, conta : inteiro
150. Inicio
151.   conta <- 0
152.   totalAluguel <- 0
153.   para i de 0 ate 11 faca
154.       totalAluguel <- totalAluguel + valorConta[i,conta]
155.   fimpara
156.   escreva("O valor gasto no ano com a Conta de ", trocaConta(conta), " é: ",totalAluguel)
157. fimprocedimento
158.
159. procedimento totalGastoanoAgua()
160. Var
161.   totalAgua : real
162.   i, conta : inteiro
163. Inicio
164.   conta <- 1
165.   totalAgua <- 0
166.   para i de 0 ate 11 faca
167.       totalAgua <- totalAgua + valorConta[i,conta]
168.   fimpara
169.   escreva("O valor gasto no ano com a Conta de ", trocaConta(conta), " é: ",totalAgua)
170. fimprocedimento
171.
172. procedimento totalGastoanoLuz()
173. Var
174.   totalLuz : real
175.   i, conta : inteiro
176. Inicio
177.   conta <- 2
178.   totalLuz <- 0
179.   para i de 0 ate 11 faca
180.       totalLuz <- totalLuz + valorConta[i,conta]
181.   fimpara
182.   escreva("O valor gasto no ano com a Conta de ", trocaConta(conta), " é: ",totalLuz)
183. fimprocedimento
184.
185. procedimento totalGastoanoGas()
186. Var
187.   totalGas : real
188.   i,conta : inteiro
189. Inicio
190.   conta <- 3
191.   totalGas <- 0
192.   para i de 0 ate 11 faca
193.       totalGas <- totalGas + valorConta[i,conta]
194.   fimpara
195.   escreva("O valor gasto no ano com a Conta de ", trocaConta(conta), " é: ",totalGas)
196. fimprocedimento
197.
```

```
198. procedimento totalGastoanoTelefone()
199. Var
200.   totalTelefone : real
201.   i,conta : inteiro
202. Inicio
203.   conta <- 4
204.   totalTelefone <- 0
205.   para i de 0 ate 11 faca
206.       totalTelefone <- totalTelefone + valorConta[i,conta]
207.   fimpara
208.   escreva("O valor gasto no ano com a Conta de ", trocaConta(conta), " é: ",totalTelefone)
209. fimprocedimento
210.
211.  //Modulo Principal
212.  Inicio
213.  repita
214.   menuPrincipal()
215.   leia(opcao)
216.   se((opcao=1) ou (opcao=2) ou (opcao=3)) entao
217.     escolha(opcao)
218.       caso 1
219.         cadastroContas()
220.       caso 2
221.         menuConsulta()
222.         leia(opcaoConsulta)
223.           se((opcaoConsulta=1) ou (opcaoConsulta=2) ou (opcaoConsulta=3) ou (opcaoConsulta=4) ou (opcaoConsulta=5) ou
      (opcaoConsulta=6)) entao
224.             escolha(opcaoConsulta)
225.               caso 1
226.                 totalGastoanoTodas()
227.               caso 2
228.                 totalGastoanoAgua()
229.               caso 3
230.                 totalGastoanoLuz()
231.               caso 4
232.                 totalGastoanoGas()
233.               caso 5
234.                 totalGastoanoTelefone()
235.               caso 6
236.                 totalGastoanoAluguel()
237.             fimescolha
238.           senao
239.             msgErr()
240.           fimse
241.
242.     fimescolha
243.   senao
244.     msgErr()
245.   fimse
246.
247. ate(opcao=3)
248. Fimalgoritmo
```

Figura 181 – Algoritmo orçamento anual

Neste exemplo de algoritmo, tem-se a aplicação de diversos conceitos, como: decisão, repetição, escolha-caso, encadeamento, procedimentos, funções, vetores e matrizes. Para exibir informações ao usuário como menus e resultados, foram criados procedimentos que permitem a interação com ele de modo que ele entenda o que está ocorrendo e o que é solicitado.

Duas funções de troca de valor foram estabelecidas: para o usuário não necessitar digitar os nomes dos meses (função **trocaMes**, linha 54) nem os nomes das contas (função **trocaConta**, linha 98), isso também facilita a elaboração do código, uma vez que números são mais adequados para trabalhar com laços de repetição. Dessa forma, o **mês 0** indica o mês de **janeiro** e a conta segue até o **mês 11**, que indica **dezembro**, da mesma forma que a **conta 0** indica **aluguel**, **conta 1** indica **água**, **conta 2**, **luz**, **conta 3**, **gás** e **conta 4**, **telefone**.

Para armazenar os dados, criou-se um procedimento chamado **cadastroContas**, na linha 121, que é chamado na linha 226 do módulo principal de **voz de comando**. Contudo, para que esse procedimento seja iniciado, na linha 221 o **menuPrincipal** é carregado para que o usuário escolha o que deseja fazer: cadastrar, consultar ou sair. Ao escolher uma opção, primeiro é verificado o valor inserido através do comando **SE**, condicionado a validar os números 1, 2 e 3, sendo a correspondência: **1** para **cadastro**, **2** para **consulta** e **3** para **sair**.

O módulo principal é iniciado com o comando **REPITA... ATÉ**, isso fará com que esse módulo repita até que o usuário digite a opção 3 **sair**. Enquanto ele não informar essa opção, o programa repetirá toda sua estrutura abrindo, primeiramente, o menu principal. Além disso, se o usuário escolher uma opção diferente de 1, 2 ou 3, uma mensagem de erro será exibida, através da chamada do procedimento **msgErr**. Ao observar o módulo principal, é possível notar que esse procedimento é utilizado duas vezes.

Outro ponto de destaque desse algoritmo é a variável global **valorConta**. Essa variável é uma matriz declarada como uma variável global para ser reutilizada nos diversos módulos criados.

Lembrete

O escopo de uma variável define se ela é local ou global. Uma variável é global quando definida fora de todas as funções do programa, ou seja, na seção de declaração de variáveis do algoritmo.

Variáveis globais são acessíveis em qualquer ponto do algoritmo, ou seja, no bloco principal ou dentro de módulos especificados como procedimentos ou funções.

Variáveis locais são declaradas dentro do módulo e são conhecidas apenas no escopo da função (ou procedimento).

Para o VisuAlg e em outras linguagens, tais como C, C++ ou Java, por exemplo, as boas práticas recomendam a utilização de passagem de valores ou passagem de vetores/matrizes diretamente pelo procedimento ou função, mas o VisuAlg não dá suporte a isso.

Quando uma variável é utilizada dentro de um procedimento ou função, a declaração e utilização só vale dentro do próprio procedimento ou função, assim como foram feitos nos procedimentos e funções: **trocaConta**, **trocaMes**, **cadastroContas**, **totalGastoanoTodas**, **totalGastoanoAluguel,**

totalGastoanoAgua, **totalGastoanoLuz**, **totalGastoanoGas** e **totalGastoanoTelefone**, em que há variáveis locais. Para a consulta, também optou-se por utilizar **ESCOLHA-CASO** e chamar **procedimentos**.

A quantidade de linhas de código não foi reduzida, mas o algoritmo está organizado. A finalidade de se utilizar procedimentos e/ou funções é reaproveitar linhas que serão eventualmente repetidas, reduzindo linhas de código à medida que cada procedimento ou função é reaproveitada. Além disso, os vetores ou matrizes também podem ser reutilizados, desde que a sobrescrita de dados seja possível, caso contrário, os dados serão perdidos quando uma das posições for sobrescrita.

8.1 Ordenando vetores

Algoritmos importantes foram escritos para ordenação de dados armazenados em vetores. Assim, o algoritmo de ordenação denominado Bolha (Bubble sort) percorre todas as posições do vetor, trocando os elementos de posição de dois elementos consecutivos cada vez que estiverem fora de ordem.

Ao final de cada iteração, o elemento maior estará na última posição do vetor e pode ficar fora da próxima iteração, ao mesmo tempo, elementos menores mover-se-ão em direção ao início do vetor.

```
1.  Algoritmo "_5.4.1 OrdenaBolha"
2.  var
3.    V : vetor [1..5] de inteiro
4.          valor, j, i, aux: inteiro
5.          N : inteiro const
6.
7.  Inicio
8.    // entrada
9.    // preencher o vetor V com dados aleatórios
10.   aleatorio on
11.   para i de 1 ate 5 passo 1 faca
12.       leia(valor)
13.       V[i] <- valor
14.       escreval("V[", i, "] = ", V[i])
15.   fimpara
16.   aleatorio off
17.
18.   // atribuicao da constante
19.   N<-5
20.   // processamento: ordenacao dos dados pelo método da bolha
21.         para j de 1 ate N-1 passo 1 faca
22.           para i de 1 ate N-1 passo 1 faca
23.                   se (V[i]>V[i+1]) entao
24.                           aux<-V[i]
25.                           V[i]<-V[i+1]
26.                           V[i+1]<-aux
27.                   fimse
28.           fimpara
29.         fimpara //fim do método de ordenação
30.
31.  //saída
32.   para i de 1 ate 5 passo 1 faca
33.         escreval("V[", i, "] = ", V[i])
34.     fimpara
35. Fimalgoritmo
```

Figura 182 – Algoritmo para ordenar os dados de uma matriz

O algoritmo declara o vetor e a constante com o tamanho do vetor que será ordenado. A constante N refere-se ao tamanho do vetor.

O bloco entre as linhas 9 e 16 é de comandos para preenchimento automático do vetor com valores aleatórios. Os comandos **aleatório on** e **aleatório off** delimitam, respectivamente, o início e o fim da captura de dados aleatórios para cada execução do comando **leia**.

Entre as linhas 21 e 29, o método de ordenação dos dados é implementado. Na primeira iteração do laço PARA externo, o algoritmo percorrerá todas as **N** posições do vetor, comparando a posição **V[i]** com a posição **V[i+1]**, trocando-as quando **V[i]** for menor do que **V[i+1]** com o auxílio da variável **aux**. Ao término da primeira iteração, o maior valor estará na última posição do vetor.

Quando se encerra o primeiro ciclo do laço interno, o laço fica preso no laço externo e repetirá o processo até que todo o vetor esteja ordenado. As iterações do algoritmo são representadas na figura a seguir:

3	1	1	1	4	4	4	4	4	4
2	2	2	4	1	1	1	3	3	3
1	3	4	2	2	2	3	1	1	2
0	4	3	3	3	3	2	2	2	1

Figura 183

Resumo

Vetores e matrizes são estruturas de dados lineares para armazenamento de dados. Matriz é uma forma de organização, chamada de organização tabular, para facilitar a resolução de alguns tipos de problemas. Na computação, as matrizes, além de terem esse propósito, são utilizadas para armazenar dados.

Os dados armazenados nessas estruturas são indexados e podem ser acessados muito rapidamente quando o índice é conhecido. A figura a seguir representa a notação dos índices da matriz A, onde i e j representam, respectivamente, o número de linhas e colunas.

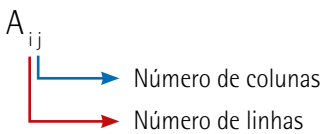

Figura 184 – Representação do índice de uma matriz A

Os índices ajudam a identificar rapidamente a posição de memória na qual o dado está armazenado. O primeiro índice denomina o número de linhas que a matriz possui, já o segundo denomina o número de colunas dessa mesma matriz.

Os fundamentos matemáticos para a compreensão de matrizes ajudam na elaboração de algoritmos mais complexos envolvendo matrizes.

Todo vetor é uma matriz. Vetores são matrizes com uma dimensão e o termo matriz é associado a matrizes com duas ou mais dimensões.

Exercícios

Questão 1. Uma empresa deve investir na criação de um sistema para cadastrar uma série de produtos que ela produz ou comercializa. Cada produto apresenta um código associado, que é um número inteiro maior do que zero. Esse código deve ser checado de acordo com um critério de validade. Se o código for considerado válido, o programa deve informar ao usuário que o código foi aceito. Caso contrário, o código é inválido, e o programa deve informar ao cliente que o código digitado está incorreto. As regras a seguir devem ser utilizadas para a checagem do código do produto.

- Todos os códigos válidos são números inteiros maiores do que zero (mas nunca iguais a zero) e menores que 20000 (podendo ser iguais a 20000).
- Se um código estiver entre 1 e 1000 (inclusive), ele deve ser automaticamente aceito.
- Se um código estiver entre 1001 e 20000 (inclusive), ele deve ser aceito apenas nas seguintes condições: se o código for um número par ou se ele for um múltiplo do número 7.
- Nas condições não contempladas pelas regras anteriores, um código não deve ser aceito.

Para resolver o problema, foi solicitada a um programador a criação de um pequeno programa, também chamado de protótipo, com o intuito de entender a regra de negócio associada à aceitação dos códigos dos produtos. Adicionalmente, foi pedido que esse programa contemplasse mais uma funcionalidade: o usuário deve ter um número máximo de tentativas para entrar com o código de um produto. Dessa forma, ao digitar um código incorreto, o programa solicita ao usuário que digite novamente o código do produto, até o limite máximo de 5 tentativas. A cada tentativa errada, o programa deve informar o número de tentativas restantes. A última tentativa deve conter um alerta especial para o usuário. Por sua vez, se o usuário entrar com o código correto, o programa deve informar imediatamente ao cliente que o código foi aceito e terminar a execução. Observe que, nesse caso, nenhuma nova tentativa deve ser solicitada, já que o código fornecido está correto.

Após a leitura cuidadosa desses requisitos, foi escrito o programa apresentado a seguir (utilizando a sintaxe do programa VisuAlg).

```
Algoritmo "PerguntaFuncao"
var
   codigo: inteiro
   checagem: logico
   tentativa: inteiro
   continua: logico
funcao checaValores(val: inteiro): logico
var resposta: logico
inicio
     resposta<-Falso
     se (val>0) e (val<=1000) entao
       resposta<-Verdadeiro
     fimse
     se (val > 1000) e (val <= 20000) entao
       se (val % 2 = 0) ou (val % 7 = 0) entao
         resposta<-Verdadeiro
       fimse
     fimse
     retorne resposta
fimfuncao
inicio
     tentativa<-5
     continua<-Verdadeiro
     repita
       escreval("Digite o codigo do produto:")
       se tentativa=1 entao
         escreval("CUIDADO: Ultima tentativa!")
       fimse
       leia(codigo)
       checagem<-checaValores(codigo)
       se checagem = Falso entao
         se tentativa > 2 entao
           escreval("Codigo incorreto. Digite novamente. Numero de tentativas restantes:", tentativa-1)
         fimse
         se tentativa = 2 entao
           escreval("Codigo incorreto. Digite novamente. Resta apenas uma tentativa.")
         fimse
       senao
           continua<-Falso
       fimse
       tentativa<-tentativa-1
     ate (continua=Falso) ou (tentativa=0)
     se (checagem=Verdadeiro) entao
       escreval("O codigo digitado foi aceito!")
     senao
       escreval("Codigo incorreto: fim das tentativas.")
     fimse
FimAlgoritmo
```

Figura 185

Com base no programa exibido e nos seus conhecimentos, avalie as afirmativas.

I – A função **checaValores** é chamada apenas uma vez no código, o que é incorreto, pois o programa deve apresentar cinco tentativas para o usuário, e não apenas uma. Por esse motivo, a função **checaValores** deveria aparecer 5 vezes no código.

II – A linha com o comando **se tentativa = 2 entao** está incorreta, pois ela deveria checar se essa é a última tentativa de entrada de código pelo usuário. A linha correta deveria ser: **se tentativa = 1 entao**, pois a última tentativa é sinalizada pelo número 1.

III – A estrutura de repetição REPITA... ATÉ é controlada por duas variáveis: a variável **continua** e a variável **tentativa**. A variável **continua** serve para controlar a condição na qual o usuário digitou um código inválido, tendo que entrar com o valor novamente (caso ainda existam tentativas disponíveis). A variável **tentativa** identifica qual é o número da tentativa atual, começando pelo valor 5, sendo decrementada até atingir o valor 0, indicando o fim da execução do laço.

IV – O uso do laço REPITA... ATÉ garante que os comandos a ele pertencentes sejam executados ao menos uma vez. Isso foi feito pois, dessa forma, o programa sempre possibilita a leitura dos dados (os códigos) ao menos uma vez.

É correto o que se afirma apenas em:

A) I.

B) II.

C) III.

D) IV.

E) III e IV.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: não é necessário que uma função seja chamada várias vezes no código para que ela tenha o efeito desejado: a utilização de uma estrutura de repetição faz com que a função seja chamada várias vezes, mesmo que, no código, ela apareça uma única vez.

II – Afirmativa incorreta.

Justificativa: devemos observar cuidadosamente a lógica do programa e perceber que a variável **tentativa** faz uma contagem regressiva. Assim, a primeira tentativa é armazenada no código com o valor 5, a segunda, com o valor 4, e assim por diante. Quando o valor dessa variável é igual a 2, o código está na penúltima tentativa. Porém, no momento em que a linha **se tentativa = 2 entao** é executada, o usuário já digitou o valor do código referente a essa tentativa. Se esse código estiver incorreto, o usuário terá apenas mais uma tentativa restante. Observe também que a linha que decrementa o valor da variável **tentativa** (no código, essa linha é a seguinte: **tentativa<-tentativa-1**) ocorre logo antes da linha com o comando ATÉ... e depois da verificação **se tentativa = 2 entao**.

III – Afirmativa correta.

Justificativa: o objetivo do laço REPITA...ATÉ utilizado no código é possibilitar as 5 tentativas de entrada de código pelo usuário. O laço deve ser interrompido em duas condições: caso o usuário digite o código correto ou caso o número de tentativas seja esgotado. Na solução apresentada, foram usadas duas variáveis para controlar cada uma dessas situações: a variável **continua** e a variável **tentativa**. Outras soluções seriam possíveis, por exemplo, utilizando o comando **interrompa** do VisuAlg.

IV – Afirmativa correta.

Justificativa: como o propósito do programa é checar a validade dos códigos, é natural que o usuário tenha ao menos uma tentativa para entrar com o código no programa e, por isso, foi utilizado o laço REPITA... ATÉ. Observe também que o número de tentativas é facilmente configurável por meio da variável **tentativa** no início do programa. No programa do enunciado, o valor inicialmente associado é igual a 5, mas, caso seja necessário mudar esse valor, isso deve ser feito apenas nesse ponto. Todo o restante do programa continua funcionando da mesma forma. Contudo, outras soluções também seriam possíveis, com o uso de outros tipos de laços. De qualquer forma, a solução apresentada está correta.

Questão 2. Uma empresa quer desenvolver um programa que vai ser utilizado por pessoas que falam diferentes línguas. O objetivo é que o usuário, ao iniciar o programa, possa escolher a língua das mensagens emitidas pelo programa. O programador vai receber um arquivo de texto que contém as mensagens em três línguas diferentes: português, inglês e italiano. No arquivo, além do texto das mensagens nas diversas línguas, existe um código associado a cada mensagem. Esse arquivo vai ser produzido por um tradutor e enviado ao programador, que deve carregar as mensagens no programa. Dessa forma, o programador não precisa saber falar as três línguas: quando ele quiser exibir determinada mensagem ao usuário, deve apenas utilizar o código associado. Durante a execução, o programa precisa ser capaz de selecionar a mensagem correta, utilizando a língua especificada inicialmente pelo usuário e o código fornecido pelo programador. Um dos programadores da equipe de desenvolvimento resolveu fazer um pequeno protótipo desse programa, utilizando o programa VisuAlg, para investigar a melhor maneira de resolver o problema. O resultado é mostrado a seguir.

```
Algoritmo "Programa"
var
   ret: logico
   ling: inteiro
   nome: caractere
   mensagens:vetor[1..3,1..9] de caractere
procedimento carregaValores
inicio
    mensagens[1,1]<-"Bem vindo!"
    mensagens[1,2]<-"Por favor, entre com o seu nome:"
    mensagens[1,3]<-"Olá "
    mensagens[1,4]<-"Existem novas mensagens para você."
    mensagens[1,5]<-"Não há novas mensagens para você."
    mensagens[1,6]<-"Até logo."
    mensagens[1,7]<-"Escolha a linguagem:"
    mensagens[1,8]<-"1)Português 2)Inglês 3)Italiano"
    mensagens[1,9]<-"Erro. Escolha inválida."
    mensagens[2,1]<-"Welcome!"
    mensagens[2,2]<-"Please enter your first name:"
    mensagens[2,3]<-"Hello "
    mensagens[2,4]<-"There are new messages for you."
    mensagens[2,5]<-"There are no new messages for you."
    mensagens[2,6]<-"Goodbye."
    mensagens[2,7]<-"Choose your language:"
    mensagens[2,8]<-"1)Portuguese 2)English 3)Italian"
    mensagens[2,9]<-"Error. Invalid choice."
    mensagens[3,1]<-"Benvenuto!"
    mensagens[3,2]<-"Per favore inserisci il tuo nome:"
    mensagens[3,3]<-"Ciao "
    mensagens[3,4]<-"Ci sono messaggi per te."
    mensagens[3,5]<-"Non ci sono nuovi messaggi per te."
    mensagens[3,6]<-"Arrivederci."
    mensagens[3,7]<-"Scegli la tua lingua:"
    mensagens[3,8]<-"1)Portoghese 2)Inglese 3)Italiano"
    mensagens[3,9]<-"Errore. Scelta non valida."
fimprocedimento
funcao mostraMensagem(codLing, codMsg: inteiro; arg: caractere): logico
inicio
    se (codLing>3) ou (codMsg>9) ou (codLing<0) ou (codMsg<0) entao
      retorne Falso
    fimse
    se codMsg = 3 entao
      escreval(mensagens[codLing,codMsg],arg,".")
    senao
      escreval(mensagens[codLing,codMsg])
    fimse
    retorne Verdadeiro
fimfuncao
inicio
    carregaValores
    ret<-mostraMensagem(1,7,"")
    ret<-mostraMensagem(1,8,"")
    leia(ling)
    se (ling<0) ou (ling>3) entao
      ret<-mostraMensagem(1,9,"") // erro na escolha da linguagem
```

```
    ling<-1 // carrega valor padrao
  fimse
  ret<-mostraMensagem(ling,1,"")
  ret<-mostraMensagem(ling,2,"")
  leia(nome)
  ret<-mostraMensagem(ling,3,nome)
  ret<-mostraMensagem(ling,5,"")
  ret<-mostraMensagem(ling,6,"")
FimAlgoritmo
```

Figura 186

Com base no programa e nos seus conhecimentos, avalie as afirmativas:

I – O procedimento **carregaValores** é desnecessário e pode ser removido sem afetar o funcionamento do programa.

II – No programa do enunciado, as mensagens nas diferentes línguas estão armazenadas na matriz Mensagens. O primeiro índice seleciona a língua desejada, enquanto o segundo seleciona a mensagem específica. Dessa forma, todo elemento da matriz Mensagens que possua o segundo índice igual (e que visualmente estaria em uma mesma coluna) deve comunicar uma mesma mensagem, e cada linha especifica uma língua diferente.

III – Inicialmente, como não sabe qual é a língua desejada pelo usuário, o programa repete a mensagem **Escolha a linguagem:** em todas as línguas disponíveis.

IV – Se, ao escolher a língua desejada, o usuário digitar um número inválido (no caso, um número diferente de 1, 2 ou 3) o programa aborta a sua execução sem fazer nenhuma interação posterior com o usuário.

É correto o que se afirma apenas em:

A) I.

B) II.

C) III.

D) IV.

E) I e IV.

Reposta correta: alternativa B.

Análise das afirmativas

I – Afirmativa incorreta.

Justificativa: o procedimento **carregaValores** é responsável por inicializar a variável global Mensagens (uma matriz) com as mensagens desejadas. Sem a inicialização, o programa não mostrará nenhuma mensagem para o usuário.

II – Afirmativa correta.

Justificativa: a matriz Mensagens associa três elementos fundamentais do problema: o código da língua (responsável por identificar a língua desejada), o código da mensagem (responsável por identificar o tipo da mensagem) e o texto da mensagem propriamente dito. Cada elemento da matriz é uma mensagem específica. Ao selecionar uma "linha" e uma "coluna" da matriz, o programa pode mostrar o texto na língua que o usuário desejar. O programador não precisa conhecer todas as línguas do programa, é necessário apenas saber qual é o código da mensagem.

III – Afirmativa incorreta.

Justificativa: no programa do enunciado, as mensagens são inicialmente mostradas apenas em uma língua, o português. Podemos perceber isso ao observarmos as linhas **ret<-mostraMensagem(1,7,"")** e **ret<-mostraMensagem(1,8,"")**, logo após a chamada do procedimento **carregaValores** e antes da linha **leia(ling)**. Note que a função **mostraMensagem** é chamada com o primeiro argumento igual a 1, que corresponde à língua portuguesa. A mudança de língua ocorre apenas após a linha **leia(ling)**, e caso o usuário tenha selecionado uma opção válida.

IV – Afirmativa incorreta.

Justificativa: se o usuário não digitar uma opção válida (1, 2 ou 3), o programa emite uma mensagem de erro em português e continua a execução com todas as mensagens em português. Isso acontece porque a variável **ling** é associada ao valor 1 no caso de erro e o programa continua a execução normalmente, como se o usuário tivesse selecionado o português como língua desejada.

REFERÊNCIAS

Textuais

ALENCAR FILHO, E. *Iniciação à lógica matemática*. 21. ed. São Paulo: Nobel, 2017.

ASCENCIO, A. F. G.; CAMPOS, E. A. V. *Fundamentos da programação de computadores*. São Paulo: Pearson, 2012.

BRASIL. Obesidade. *Biblioteca Virtual em Saúde*, dez. 2009. Disponível em: https://bit.ly/39xBcJL. Acesso em: 21 set. 2021.

CORMEN, T. H. *et al. Algoritmos*: teoria e prática. 2. ed. Rio de Janeiro: Elsevier, 2002.

FORBELLONE, A.; EBERSPACHER, H. *Lógica de programação*: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Makron Books, 2005.

IEC TECHNICAL COMMITTEE. *IEC 80000-13:2008*: quantities and units. 2008.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA. *Manual do VisualG*. [s.d.]. Disponível em: https://bit.ly/3jyuQQk. Acesso em: 2 set. 2021.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. de. *Algoritmos*: lógica para desenvolvimento de programação de computadores. 28. ed. São Paulo: Érica, 2016.

SCOTT, T. C.; MARKETOS, P. *On the origin of the Fibonacci Sequence*. St. 2014. Disponível em: https://bit.ly/2Ym4fxB. Acesso em: 5 out. 2021.

Interativa

Informações:
www.sepi.unip.br ou 0800 010 9000