

## 7.1 Matriz em algoritmos

### 7.1.1 Definição de matriz

Uma matriz é uma variável composta homogênea multidimensional. Ela é formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome), e alocadas sequencialmente na memória. Uma vez que as variáveis têm o mesmo nome, o que as distingue são índices que referenciam sua localização dentro da estrutura. Uma variável do tipo matriz precisa de um índice para cada uma de suas dimensões.

### 7.1.2 Declaração de matriz

Um algoritmo pode declarar uma matriz, conforme descrito a seguir.

```
DECLARE nome[dimensão1, dimensão2, dimensão3, ..., dimensãoN] tipo
```

onde:

nome: é o nome da variável do tipo matriz;

dimensão1: é a quantidade de elementos da 1ª dimensão (muitas vezes, chamada linha);

dimensão2: é a quantidade de elementos da 2ª dimensão (muitas vezes, denominada coluna);

dimensão3: é a quantidade de elementos da 3ª dimensão (muitas vezes, chamada profundidade);

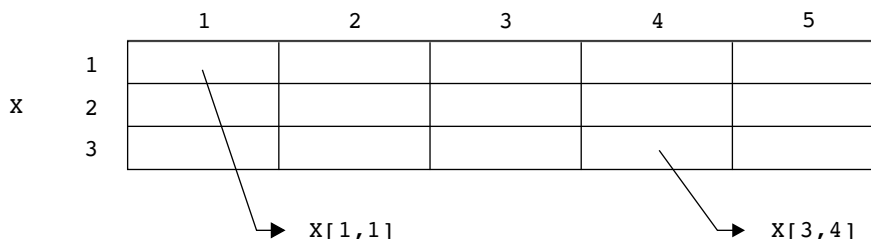
dimensãoN: é a quantidade de elementos da enésima dimensão;

tipo: é o tipo de dados dos elementos da matriz.

### 7.1.3 Exemplos de matriz

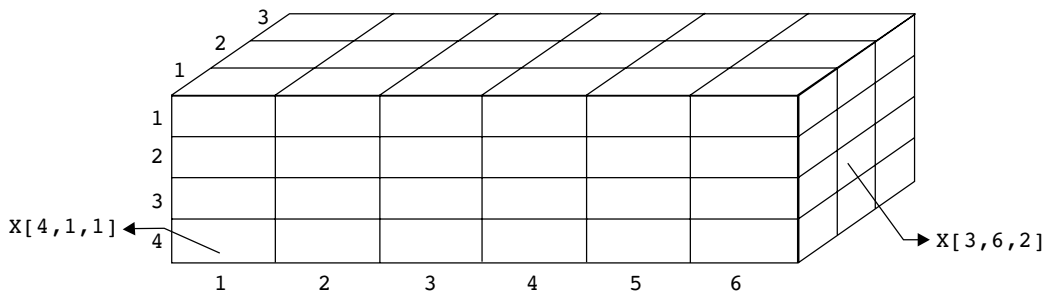
O exemplo a seguir define uma matriz bidimensional, onde o tamanho da 1ª dimensão (linha) é 3 e o da 2ª dimensão (coluna) é 5.

```
DECLARE X[3,5] NUMÉRICO
```



O exemplo que se segue define uma matriz tridimensional, onde o tamanho da 1ª dimensão (linha) é 4; o da 2ª dimensão (coluna), 6; e o da 3ª dimensão (profundidade), 3.

```
DECLARE X[4,6,3] NUMÉRICO
```



#### 7.1.4 Atribuindo valores a uma matriz

Cada elemento de uma matriz pode armazenar um valor. Para fazer esse armazenamento, é necessário executar uma atribuição, informando o número da posição desejada em cada dimensão.

Exemplo 1:

```
declare mat[5,4] numérico
mat[2,4] ← 45
mat[3,1] ← -8
mat[1,3] ← 10
```

No exemplo 1, a declaração da matriz `mat` informa que ela tem 2 dimensões. A primeira dimensão, que representa as linhas, tem tamanho 5; a segunda dimensão de `mat` tem tamanho 4. Ou seja, para cada linha há 4 colunas, permitindo, assim, que a matriz tenha espaço para armazenar 20 valores numéricos. A representação gráfica da matriz `mat` e as três atribuições podem ser vistas a seguir.

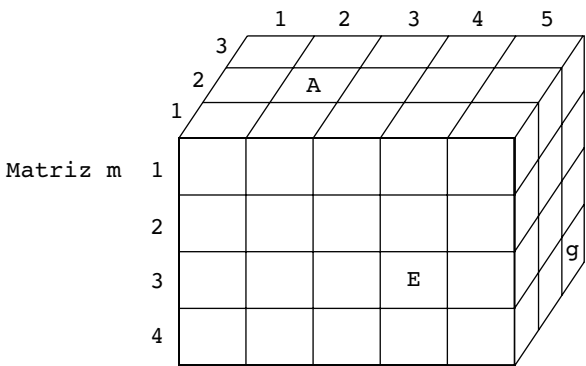
Matriz mat

1			10	
2				45
3	-8			
4				
5				
	1	2	3	4

Exemplo 2:

```
declare m[4,5,3] literal
m[3,4,1] ← "E"
m[4,5,3] ← "g"
m[1,2,2] ← "A"
```

No exemplo 2, a declaração da matriz `m` informa que ela tem 3 dimensões. A primeira dimensão, que representa as linhas, tem tamanho 4; a segunda dimensão, que representa as colunas, tem tamanho 5; e a terceira dimensão, que representa a profundidade, tem tamanho 3. Assim, a matriz `m` tem espaço para armazenar 60 valores literais. A representação gráfica da matriz `m` e as três atribuições podem ser vistas a seguir.



7.1.5 Preenchendo uma matriz

Para preencher uma matriz é necessário identificar todas as suas posições. Isso exige a utilização de um índice para cada uma de suas dimensões.

No exemplo a seguir, é mostrada uma matriz bidimensional com três linhas e cinco colunas. Observe que os valores assumidos pela variável *i* estão dentro do intervalo de 1 a 3, ou seja, exatamente o número das linhas da matriz. Por essa razão, a variável *i* é utilizada para indicar a primeira dimensão, dentro dos colchetes. Para cada valor assumido por *i*, a variável *j* assume os valores no intervalo de 1 a 5, ou seja, exatamente o número das colunas. Portanto, a variável *j* é utilizada para indicar a 2ª dimensão, dentro dos colchetes.

```
PARA i ← 1 ATÉ 3 FAÇA
INÍCIO
  PARA j ← 1 ATÉ 5 FAÇA
  INÍCIO
    ESCREVA "Digite o número da linha ",i, " e coluna: ", j
    LEIA X[i,j]
  FIM
FIM
```

Simulação:

MEMÓRIA		TELA
i	j	
1	1	Digite o número da linha 1 e coluna 1: 12
	2	Digite o número da linha 1 e coluna 2: 9
	3	Digite o número da linha 1 e coluna 3: 3
	4	Digite o número da linha 1 e coluna 4: 7
	5	Digite o número da linha 1 e coluna 5: -23
2	1	Digite o número da linha 2 e coluna 1: 15
	2	Digite o número da linha 2 e coluna 2: 4
	3	Digite o número da linha 2 e coluna 3: 2
	4	Digite o número da linha 2 e coluna 4: 34
	5	Digite o número da linha 2 e coluna 5: -4
3	1	Digite o número da linha 3 e coluna 1: 3
	2	Digite o número da linha 3 e coluna 2: 45
	3	Digite o número da linha 3 e coluna 3: 3
	4	Digite o número da linha 3 e coluna 4: 0
	5	Digite o número da linha 3 e coluna 5: -3

Assim, podemos imaginar os elementos dispostos em uma estrutura bidimensional, como uma tabela.

		1	2	3	4	5
X	1	12	9	3	7	-23
	2	15	4	2	34	-4
	3	3	45	3	0	-3

Já no exemplo que se segue, é preenchida uma matriz tridimensional com quatro linhas, três colunas e profundidade dois. Observe que os valores da variável  $i$  estão dentro do intervalo de 1 a 4, ou seja, exatamente o número das linhas da matriz. Para cada valor assumido por  $i$ , os valores da variável  $j$  se movimentam de 1 a 3, ou seja, as três colunas que cada linha possui. Por fim, os valores da variável  $k$  se alternam entre 1 e 2, exatamente os valores da profundidade.

```

PARA i ← 1 ATÉ 4 FAÇA
INÍCIO
  PARA j ← 1 ATÉ 3 FAÇA
  INÍCIO
    PARA k ← 1 ATÉ 2 FAÇA
    INÍCIO
      ESCREVA "Digite o número da linha ", i, " coluna ", j, " e profundidade
        ↳ ", k, ":"
      LEIA X[i,j,k]
    FIM
  FIM
FIM
FIM

```

**Simulação:**

MEMÓRIA			TELA
i	j	k	
1	1	1	Digite o número da linha 1 coluna 1 e profundidade 1: 2
		2	Digite o número da linha 1 coluna 1 e profundidade 2: 5
	2	1	Digite o número da linha 1 coluna 2 e profundidade 1: -1
		2	Digite o número da linha 1 coluna 2 e profundidade 2: 0
	3	1	Digite o número da linha 1 coluna 3 e profundidade 1: 15
		2	Digite o número da linha 1 coluna 3 e profundidade 2: 8
2	1	1	Digite o número da linha 2 coluna 1 e profundidade 1: -25
		2	Digite o número da linha 2 coluna 1 e profundidade 2: 3
	2	1	Digite o número da linha 2 coluna 2 e profundidade 1: 6
		2	Digite o número da linha 2 coluna 2 e profundidade 2: 9
	3	1	Digite o número da linha 2 coluna 3 e profundidade 1: 7
		2	Digite o número da linha 2 coluna 3 e profundidade 2: 11
3	1	1	Digite o número da linha 3 coluna 1 e profundidade 1: 23
		2	Digite o número da linha 3 coluna 1 e profundidade 2: -2
	2	1	Digite o número da linha 3 coluna 2 e profundidade 1: -5
		2	Digite o número da linha 3 coluna 2 e profundidade 2: 46
	3	1	Digite o número da linha 3 coluna 3 e profundidade 1: 19
		2	Digite o número da linha 3 coluna 3 e profundidade 2: 1
4	1	1	Digite o número da linha 4 coluna 1 e profundidade 1: 14
		2	Digite o número da linha 4 coluna 1 e profundidade 2: 27
	2	1	Digite o número da linha 4 coluna 2 e profundidade 1: 5
		2	Digite o número da linha 4 coluna 2 e profundidade 2: 4
	3	1	Digite o número da linha 4 coluna 3 e profundidade 1: 10
		2	Digite o número da linha 4 coluna 3 e profundidade 2: 65

Assim, podemos imaginar os elementos dispostos em uma estrutura tridimensional, como um cubo.

Matriz x

	1	5	0	8	
	2	-1	15	8	
1	2	-1	15	15	11
2	-25	6	7	7	1
3	23	-5	19	19	65
4	14	5	10	10	
	1	2	3		

### 7.1.6 Mostrando os elementos de uma matriz

Para mostrar os elementos de uma matriz é preciso identificar suas posições. Como acontece com todas as operações realizadas com matrizes, é necessária a utilização de um índice para cada dimensão da matriz.

No exemplo a seguir, uma matriz bidimensional com três linhas e cinco colunas é apresentada. Observe que a variável *i* assume valores sequenciais no intervalo de 1 a 3, ou seja, exatamente as linhas da matriz. Para cada valor assumido por *i*, a variável *j* assume valores sequenciais de 1 a 5, ou seja, as cinco colunas que cada linha possui.

```

PARA i ← 1 ATÉ 3 FAÇA
INÍCIO
    PARA j ← 1 ATÉ 5 FAÇA
    INÍCIO
        ESCRIVA X[i,j]
    FIM
FIM
FIM

```

### 7.1.7 Percorrendo uma matriz

Vimos anteriormente, nos tópicos 7.1.5 e 7.1.6, formas para preencher toda uma matriz e a fim de mostrar todas as posições de uma matriz. Em tais operações, foi preciso passar por todas as posições da matriz, ou seja, foi necessário percorrer a matriz.

Uma das formas mais simples de percorrer uma matriz pode ser por meio do uso de uma estrutura de repetição para cada dimensão da matriz. A disposição de tais estruturas de repetição define a forma como a matriz será percorrida.

A seguir, apresentaremos duas formas para percorrer uma mesma matriz, chamada *x*, contendo 3 linhas e 4 colunas.

Matriz x

1	4	5	1	10
2	16	11	76	8
3	9	54	32	89
	1	2	3	4

**Forma 1:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada linha. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do algoritmo, servirá apenas para facilitar a explicação).

```

1. PARA i ← 1 ATÉ 3 FAÇA
2. INÍCIO
3.     ESCRIVA "Elementos da linha ", i
4.     PARA j ← 1 ATÉ 4 FAÇA
5.     INÍCIO
6.         ESCRIVA x[i,j]
7.     FIM
8. FIM

```

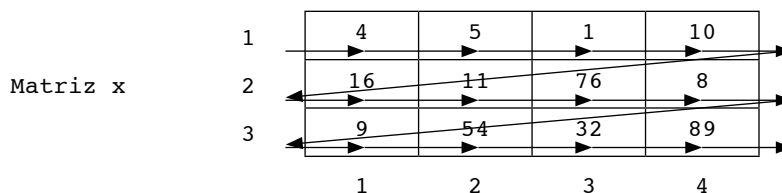
A primeira estrutura de repetição (linha 1) é controlada pela variável  $i$ , que poderá assumir valores dentro do intervalo de 1 a 3. Cada vez que essa estrutura PARA for executada, encontrará a segunda estrutura de repetição (linha 4), controlada pela variável  $j$ , que assumirá os valores dentro do intervalo de 1 a 4. Assim, cada valor assumido pela variável  $i$  estará associado a 4 valores da variável  $j$ .

Esse arranjo resolve o problema de mostrar os elementos, separando-os por linhas, já que a variável  $i$  ficará com valor fixo, enquanto a variável  $j$  assumirá valores de 1 a 4, ou seja, formará todos os pares possíveis de índices. Se  $i$  estiver valendo 1, serão mostrados todos os elementos da linha 1, já que serão formados os seguintes pares:  $x[1,1]$ ,  $x[1,2]$ ,  $x[1,3]$  e  $x[1,4]$ . Depois, a variável  $i$  assume o valor 2 e novamente a  $j$  terá seus valores variando de 1 a 4. Com isso, será possível percorrer toda a linha 2 por meio da formação dos pares  $x[2,1]$ ,  $x[2,2]$ ,  $x[2,3]$  e  $x[2,4]$ . Esse processo se repetirá para os demais valores possíveis de  $i$ . A tabela a seguir mostra uma simulação de execução do algoritmo. Nessa simulação, é importante observar como as variáveis  $i$  e  $j$  têm seus valores alterados.

### Simulação:

MEMÓRIA		TELA
$i$	$j$	
1		Elementos da linha 1
1	1	4
1	2	5
1	3	1
1	4	10
2		Elementos da linha 2
2	1	16
2	2	11
2	3	76
2	4	8
3		Elementos da linha 3
3	1	9
3	2	54
3	3	32
3	4	89

A figura a seguir dá uma outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis  $i$  e  $j$  e o caminho utilizado para percorrer a matriz.



**Forma 2:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada coluna. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do algoritmo, servirá apenas para facilitar a explicação).

```

1. PARA i ← 1 ATÉ 4 FAÇA
2. INÍCIO
3.   ESCREVA "Elementos da coluna ", i
4.   PARA j ← 1 ATÉ 3 FAÇA
5.   INÍCIO
6.     ESCREVA x[j, i]
7.   FIM
8. FIM

```

A primeira estrutura de repetição (linha 1) é controlada pela variável  $i$ , que poderá assumir valores dentro do intervalo de 1 a 4. Cada vez que essa estrutura PARA for executada, encontrará a segunda estrutura de repetição (linha 4), controlada pela variável  $j$ , que assumirá os valores dentro do intervalo de 1 a 3. Assim, cada valor assumido pela variável  $i$  estará associado a 3 valores da variável  $j$ . Esse arranjo resolve o problema de mostrar os elementos, separando-os por colunas, já que a variável  $i$  ficará com valor fixo enquanto a variável  $j$  assumirá valores de 1 a 3, ou seja, formará todos os pares possíveis de índices. Se  $i$  estiver valendo 1, serão mostrados todos os elementos da coluna 1, já que serão formados os seguintes pares:  $x[1,1]$ ,  $x[2,1]$  e  $x[3,1]$ . Depois, a variável  $i$  assumirá o valor 2 e novamente  $j$  terá seus valores variando de 1 a 3. Com isso, será possível percorrer toda a coluna 2, por meio da formação dos pares  $x[1,2]$ ,  $x[2,2]$  e  $x[3,2]$ . Esse processo se repetirá para os demais valores possíveis de  $i$ . A tabela a seguir mostra uma simulação de execução do algoritmo. Nessa simulação é importante observar como as variáveis  $i$  e  $j$  têm seus valores alterados.

### Simulação:

MEMÓRIA		TELA
$i$	$j$	
1		Elementos da coluna 1
1	1	4
1	2	16
1	3	9
2		Elementos da coluna 2
2	1	5
2	2	11
2	3	54
3		Elementos da coluna 3
3	1	1
3	2	76
3	3	32
4		Elementos da coluna 4
4	1	10
4	2	8
4	3	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis  $i$  e  $j$  e o caminho utilizado para percorrer a matriz.

Matriz  $x$

1	4	5	1	10
2	16	11	76	8
3	9	54	32	89
	1	2	3	4

Pelas formas de percorrer uma matriz, apresentadas anteriormente, podemos observar alguns pontos que merecem atenção:

- a mudança dos valores das variáveis  $i$  e  $j$ , que controlam as estruturas de repetição, permite a formação de todos os possíveis pares de linha e coluna existentes na matriz.
- a mudança do valor da variável  $i$ , utilizada no PARA externo, acontece mais lentamente que a mudança da variável  $j$ , utilizada no PARA interno. Logo, foi a variável  $i$  quem indicou como seria o percurso: na primeira forma apresentada,  $i$  variou de 1 a 3 e foi usada na primeira posição dentro do colchetes, isso mostrou que o percurso seria horizontal, porque o índice da linha ficava parado enquanto  $j$  assumia todas as colunas possíveis para aquela linha. Já na segunda forma apresentada,  $i$  variou de 1 a 4 e foi usada na segunda posição dentro dos colchetes, indicando que o percurso seria vertical, pois o índice da coluna ficava parado enquanto  $j$  assumia todas as linhas possíveis para aquela coluna.

## 7.2 Matriz em PASCAL

### 7.2.1 Definição de matriz

As variáveis compostas homogêneas multidimensionais (ou, simplesmente, matrizes) são conhecidas na linguagem PASCAL como `ARRAY`. Uma estrutura do tipo `ARRAY` é uma sequência de variáveis com o mesmo identificador (mesmo nome) e alocadas sequencialmente na memória. Todas as variáveis que compõem uma `ARRAY` devem ser do mesmo tipo.

Uma vez que as variáveis recebem o mesmo nome, o que as distingue são os índices que referenciam sua posição em cada dimensão da estrutura. Assim, se a matriz for bidimensional necessitará de dois índices, se for tridimensional necessitará de três índices, e assim por diante.

### 7.2.2 Declaração de matriz

```
VAR nome_da_variável: ARRAY[início1..fim1, início2..fim2,
                             inícioN..fimN] OF tipo_dos_dados;
```

onde:

`nome_da_variável`: é o nome da variável do tipo matriz;

`início1`: é o índice inicial da primeira dimensão da matriz;

`fim1`: é o índice final da primeira dimensão da matriz;

`início2`: é o índice inicial da segunda dimensão da matriz;

`fim2`: é o índice final da segunda dimensão da matriz;

`inícioN`: é o índice inicial da n-ésima dimensão da matriz;

`fimN`: é o índice final da n-ésima dimensão da matriz;

`tipo_dos_dados`: é o tipo básico dos dados que serão armazenados na matriz.

É importante salientar que o valor do `índice_inicial` deve ser maior ou igual ao valor do `índice_final`, para cada dimensão. As posições são identificadas com valores dentro desse intervalo.

Além disso, é importante destacar que os índices também podem ser representados por valores alfabéticos. Com isso, é permitido o uso de caracteres para representar o início e o fim de cada dimensão. O exemplo a seguir ilustra essa possibilidade. Como se pode ver, a matriz terá 5 linhas (de C a G) e 4 colunas (de L a O).

```
VAR matriz: ARRAY ['C'..'G', 'L'..'O'] OF REAL;
```



#### Observação

Os valores que indicam o `índice_inicial` e o `índice_final` devem representar valores fixos (literais<sup>1</sup> ou constantes), não podendo ser substituídos por variáveis.

### 7.2.3 Exemplos de matriz

```
VAR X: ARRAY[1..2, 1..6] OF REAL;
```

		1	2	3	4	5	6
X	1						
	2						

```
VAR MAT: ARRAY[2..5, 3..6] OF CHAR;
```

<sup>1</sup> Literal é um valor fixo, definido quando se escreve o programa. Por exemplo: `x:=10.3`; onde 10.3 é um literal. `vet: array [3..7, 4..9] of integer`; onde 3, 7, 4 e 9, escritos dentro dos colchetes, são literais.



		3	4	5	6
MAT	2				
	3				
	4				
	5				

```
VAR Y: ARRAY[1..2,1..4,1..3] OF REAL;
```

```
VAR MAT: ARRAY['A'..'D', 'F'..'J'] OF INTEGER;
```

MAT	A					
	B					
	C					
	D					
		F	G	H	I	J

```
CONST inicio1 = 3;
CONST fim1 = 6;
CONST inicio2 = 8;
CONST fim2 = 10;
VAR MATRIZ: ARRAY[inicio1..fim1, inicio2..fim2] OF INTEGER;
```

MATRIZ	3			
	4			
	5			
	6			
		8	9	10

### 7.2.4 Atribuindo valores a uma matriz

Atribuir valor à matriz significa armazenar uma informação em um de seus elementos, identificado de forma única por meio de seus índices.

`x[1,4] := 5;` → Atribui o valor 5 à posição identificada pelos índices 1 (linha) e 4 (coluna).

	1	2	3	4	5	6
x	1			5		
	2					

`MAT[4,5] := 'D';` → Atribui a letra D à posição identificada pelos índices 4 (linha) e 5 (coluna).

		3	4	5	6
2					
3					
4			D		
5					

`Y[1,4,2] := 12;` → Atribui o valor 12 à posição identificada pelos índices 1 (linha), 4 (coluna) e 2 (profundidade).

			3						
			2						
			1						
1								12	
2								12	
	1	2	3	4					

`MAT['C','H'] := 20;` → Atribui o valor 20 à posição identificada pelos índices C (linha) e H (coluna).

	A					
	B					
	C			20		
	D					
		F	G	H	I	J

### 7.2.5 Preenchendo uma matriz

Preencher uma matriz significa percorrer todos os seus elementos, atribuindo-lhes um valor. Esse valor pode ser recebido do usuário, pelo teclado, ou pode ser gerado pelo programa.

Exemplo 1:

```
FOR i:= 1 TO 7 DO
  BEGIN
    FOR j:=1 TO 3 DO
      BEGIN
        READLN(X[i,j]);
      END;
    END;
  END;
```

Exemplo 2:

```
FOR i:='C' TO 'G' DO
  BEGIN
    FOR j:= 'L' TO 'O' DO
      BEGIN
        READLN(MAT[i,j]);
      END;
    END;
  END;
```

O exemplo 1 apresentou duas estruturas de repetição FOR para garantir que a variável *i* assumisse todos os valores possíveis para linha (de 1 a 7) e a variável *j* assumisse todos os valores possíveis para coluna (de 1 a 3) da matriz *x*. Já o exemplo 2 utilizou duas estruturas de repetição FOR para garantir que a variável *i* assumisse todos os valores possíveis para linha (de C a G) e a variável *j* assumisse todos os valores possíveis para coluna (de L a O) da matriz *MAT*. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz é preenchida com um valor digitado pelo usuário por meio do comando READLN.

### 7.2.6 Mostrando os elementos de uma matriz

Pode-se também percorrer todos os elementos da matriz, acessando seu conteúdo. Para mostrar os valores armazenados em uma matriz, veremos, a seguir, alguns trechos de programas.

Exemplo 1:

```
...
VAR X:ARRAY[1..10,1..6] OF REAL;
...
FOR i:=1 TO 10 DO
  BEGIN
    FOR j:= 1 TO 6 DO
      BEGIN
        WRITELN(X[i,j]);
      END;
    END;
  END;
```

Exemplo 2:

```
...
VAR MAT:ARRAY['C..'G','L..'O'] OF REAL;
...
FOR i:='C' TO 'G' DO
  BEGIN
    FOR j:= 'L' TO 'O' DO
      BEGIN
        WRITELN(MAT[i, j]);
      END;
    END;
  END;
```

O exemplo 1 usou duas estruturas de repetição FOR para garantir que a variável *i* assumisse todos os valores possíveis para linha (de 1 a 10) e a variável *j* assumisse todos os valores possíveis para coluna (de 1 a 6) da matriz *x*. Já o exemplo 2 utilizou duas estruturas de repetição FOR para garantir que a variável *i* assumisse todos os valores possíveis para linha (de C a G) e a variável *j* assumisse todos os valores possíveis para coluna (de L a O) da matriz *MAT*. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz é acessada e seu valor mostrado por meio do comando WRITELN.

### 7.2.7 Percorrendo uma matriz

Vimos anteriormente, nos tópicos 7.2.5 e 7.2.6, formas para preencher toda uma matriz e para mostrar todas as posições de uma matriz. Em tais operações, foi necessário passar por todas as posições da matriz, ou seja, foi preciso percorrer a matriz.

Uma das formas mais simples de percorrer uma matriz pode ser por meio do uso de uma estrutura de repetição para cada dimensão da matriz. A disposição de tais estruturas de repetição define a forma como a matriz será percorrida.

A seguir, apresentaremos duas formas para percorrer uma mesma matriz, chamada *x*, contendo 3 linhas e 4 colunas.

Matriz x	1	4	5	1	10
	2	16	11	76	8
	3	9	54	32	89
		1	2	3	4

**Forma 1:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada linha. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).

```

1. FOR i := 1 TO 3 DO
2. BEGIN
3.   WRITELN( 'Elementos da linha ', i);
4.   FOR j := 1 TO 4 DO
5.     BEGIN
6.       WRITELN(x[i,j]);
7.     END;
8. END;
```

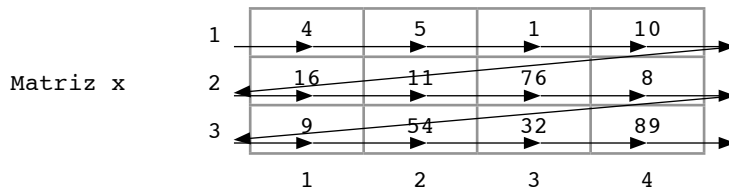
A primeira estrutura de repetição (linha 1) é controlada pela variável *i*, que poderá assumir valores dentro do intervalo de 1 a 3. Cada execução da estrutura FOR encontrará a segunda estrutura de repetição (linha 4), controlada pela variável *j*, que assumirá os valores dentro do intervalo de 1 a 4. Assim, cada valor assumido pela variável *i* estará associado a 4 valores da variável *j*.

Esse arranjo resolve o problema de mostrar os elementos, separando-os por linhas, já que a variável *i* ficará com valor fixo enquanto a variável *j* assumirá valores de 1 a 4, ou seja, formará todos os pares possíveis de índices. Se *i* estiver valendo 1, serão mostrados todos os elementos da primeira linha, já que serão formados os seguintes pares: *x*[1,1], *x*[1,2], *x*[1,3] e *x*[1,4]. Depois, a variável *i* assume o valor 2 e novamente a *j* terá seus valores variando de 1 a 4. Com isso, será possível percorrer toda a segunda linha, por meio da formação dos pares *x*[2,1], *x*[2,2], *x*[2,3] e *x*[2,4]. Esse processo se repetirá para os demais valores possíveis de *i*. A tabela a seguir mostra uma simulação de execução do programa. Nessa simulação é importante observar como as variáveis *i* e *j* têm seus valores alterados.

### Simulação:

MEMÓRIA		TELA
i	j	
1		Elementos da linha 1
1	1	4
1	2	5
1	3	1
1	4	10
2		Elementos da linha 2
2	1	16
2	2	11
2	3	76
2	4	8
3		Elementos da linha 3
3	1	9
3	2	54
3	3	32
3	4	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis  $i$  e  $j$  e o caminho utilizado para percorrer a matriz.



**Forma 2:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada coluna. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).

```

1. FOR i := 1 TO 4 DO
2. BEGIN
3.   WRITELN('Elementos da coluna ', i);
4.   FOR j := 1 TO 3 DO
5.     BEGIN
6.       WRITELN(x[j,i]);
7.     END;
8. END;
```

A primeira estrutura de repetição (linha 1) é controlada pela variável  $i$ , que poderá assumir valores dentro do intervalo de 1 a 4. Cada execução da estrutura **FOR** encontrará a segunda estrutura de repetição (linha 4), controlada pela variável  $j$ , que assumirá os valores dentro do intervalo de 1 a 3. Assim, cada valor assumido pela variável  $i$  estará associado a 3 valores da variável  $j$ . Esse arranjo resolve o problema de mostrar os elementos, separando-os por colunas, já que a variável  $i$  ficará com valor fixo enquanto a variável  $j$  assumirá valores de 1 a 3, ou seja, formará todos os pares possíveis de índices. Se  $i$  estiver valendo 1, serão mostrados todos os elementos da primeira coluna, já que serão formados os seguintes pares:  $x[1,1]$ ,  $x[2,1]$  e  $x[3,1]$ . Depois, a variável  $i$  assumirá o valor 2 e novamente  $j$  terá seus valores variando de 1 a 3. Com isso, será possível percorrer toda a segunda coluna, por meio da formação dos pares  $x[1,2]$ ,  $x[2,2]$  e  $x[3,2]$ . Esse processo se repetirá para os demais valores possíveis de  $i$ . A tabela a seguir mostra uma simulação de execução do programa. Nessa simulação, é importante observar como as variáveis  $i$  e  $j$  têm seus valores alterados.

#### Simulação:

MEMÓRIA		TELA
$i$	$j$	
1		Elementos da coluna 1
1	1	4
1	2	16
1	3	9
2		Elementos da coluna 2
2	1	5
2	2	11
2	3	54
3		Elementos da coluna 3
3	1	1
3	2	76
3	3	32
4		Elementos da coluna 4
4	1	10
4	2	8
4	3	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis *i* e *j* e o caminho utilizado para percorrer a matriz.

Matriz x

1	4	5	1	10
2	16	11	76	8
3	9	54	32	89
	1	2	3	4

Pelas formas de percorrer uma matriz apresentadas anteriormente, podemos observar alguns pontos que merecem atenção:

- a mudança dos valores das variáveis *i* e *j*, que controlam as estruturas de repetição, permite a formação de todos os possíveis pares de linha e coluna existentes na matriz.
- a mudança do valor da variável *i*, utilizada no **FOR** externo, acontece mais lentamente que a mudança da variável *j*, utilizada no **FOR** interno. Logo, foi a variável *i* que indicou como seria o percurso: na primeira forma apresentada, *i* variou de 1 a 3 e foi usada na primeira posição dentro do colchete, isso mostrou que o percurso seria horizontal, porque o índice da linha ficava parado enquanto *j* assumia todas as colunas possíveis para aquela linha. Já na segunda forma apresentada, *i* variou de 1 a 4 e foi usada na segunda posição dentro dos colchetes, indicando que o percurso seria vertical, pois o índice da coluna ficava parado enquanto *j* assumia todas as linhas possíveis para aquela coluna.

## 7.3 Matriz em C/C++

### 7.3.1 Definição de matriz

Uma matriz pode ser definida como um conjunto de variáveis de mesmo tipo e identificadas pelo mesmo nome. Essas variáveis são diferenciadas por meio da especificação de suas posições dentro dessa estrutura.

A linguagem C/C++ permite a declaração de matrizes unidimensionais (mais conhecidas como *vetores* — descritos no capítulo anterior), bidimensionais e multidimensionais. O padrão ANSI prevê até 12 dimensões. Entretanto, o limite de dimensões fica por conta da quantidade de recursos computacionais disponíveis. Apesar disso, as matrizes mais utilizadas possuem duas dimensões. Para cada dimensão deve ser utilizado um índice.

Os índices usados na linguagem C/C++, para identificar as posições de uma matriz, começam sempre em 0 (zero) e vão até o tamanho da dimensão menos uma unidade. Os índices de uma matriz em C/C++ devem sempre ser representados por um dos tipos inteiros disponíveis na linguagem.

### 7.3.2 Declaração de matriz

```
tipo_dos_dados nome_variável [dimensão1] [dimensão2] [...] [dimensãoN];
```

onde:

`tipo_dos_dados`: é o tipo dos dados que serão armazenados na matriz;

`nome_variável`: é o nome dado à variável do tipo matriz;

`[dimensão1]`: representa o tamanho da 1ª dimensão da matriz;

`[dimensão2]`: representa o tamanho da 2ª dimensão da matriz;

`[dimensãoN]`: representa o tamanho da n-ésima dimensão da matriz.

Em C/C++, a indicação do tamanho das dimensões de uma matriz deve ser feita por um valor inteiro fixo (representado por um literal<sup>2</sup> ou uma constante). Se houver necessidade de definir o tamanho do vetor em tempo de execução, deve-se fazê-lo por meio de ponteiros (o Capítulo 8 apresentará o conceito de ponteiro).

<sup>2</sup> Literal é um valor fixo, definido quando se escreve o programa. Por exemplo: `double x=10.3`; onde 10.3 é um literal. `char mat [2][5]`; onde 2 e 5, escritos dentro dos colchetes, são literais.

### 7.3.3 Exemplos de matriz

Da mesma maneira como ocorre com os vetores, os índices das dimensões das matrizes começam sempre em 0 (zero).

A seguir, são apresentadas algumas formas de criação de matrizes.

Exemplo 1:

```
float X[2][6];
```

Na declaração do exemplo 1, criou-se uma variável chamada `x` contendo duas linhas (de 0 a 1) com seis colunas cada (de 0 a 5), capaz de armazenar números reais, como pode ser observado a seguir.

		0	1	2	3	4	5
x	0						
	1						

Exemplo 2:

```
char MAT [4][3];
```

A declaração do exemplo 2 criou uma variável chamada `MAT` contendo quatro linhas (de 0 a 3) com três colunas cada (de 0 a 2), capaz de armazenar caracteres, como pode ser observado a seguir.

		0	1	2
MAT	0			
	1			
	2			
	3			

Exemplo 3:

```
float Y[2][4][3];
```

A declaração do exemplo 3 criou uma variável chamada `y` contendo duas linhas (de 0 a 1) com quatro colunas cada (de 0 a 3) e profundidade três (de 0 a 2), capaz de armazenar números reais, como pode ser observado a seguir.

A 3D perspective diagram of a 2x4x3 matrix. The front face is a 2x4 grid with rows labeled 0 and 1, and columns labeled 0, 1, 2, and 3. The depth axis is labeled with 0, 1, and 2, representing the third dimension of the array.

### 7.3.4 Atribuindo valores a uma matriz

Atribuir valor a uma matriz significa armazenar informação em seus elementos, identificados de forma única por meio de seus índices.

`x[1][4] = 5;` → Atribui o valor 5 à posição identificada pelos índices 1 (2ª linha) e 4 (5ª coluna).

		0	1	2	3	4	5
x	0						
	1					5	

`MAT[3][2] = 'D';` → Atribui a letra D à posição identificada pelos índices 3 (4ª linha) e 2 (3ª coluna).

		0	1	2
MAT	0			
	1			
	2			
	3			D

`Y[0][3][1] = 12;` → Atribui o valor 12 à posição identificada pelos índices 0 (1ª linha), 3 (4ª coluna) e 1 (2ª profundidade).

			2						
			1						
			0						
0								12	
1									12
	0	1	2	3					

### 7.3.5 Preenchendo uma matriz

Preencher uma matriz significa percorrer todos os seus elementos, atribuindo-lhes um valor. Esse valor pode ser recebido do usuário, por meio do teclado, ou pode ser gerado pelo programa.

No exemplo que se segue, todos os elementos de uma matriz bidimensional são percorridos, atribuindo-lhes valores digitados pelo usuário e capturados pelo comando `scanf`.

```
for (i=0;i<7;i++)
{
    for (j=0;j<3;j++)
        scanf("%d%c", &MAT[i][j]);
}
```

Como a matriz possui 7 linhas e 3 colunas, o exemplo apresentou duas estruturas de repetição `for` para garantir que a variável `i` assumisse todos os valores possíveis para linha (de 0 a 6) e a variável `j` assumisse todos os valores possíveis para coluna (de 0 a 2) da matriz `MAT`. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz foi preenchida por um valor digitado pelo usuário por meio do comando `scanf`.

### 7.3.6 Mostrando os elementos de uma matriz

Pode-se também percorrer todos os elementos de uma matriz acessando seu conteúdo. Para mostrar os valores armazenados dentro de uma matriz, supondo que ela tenha sido declarada como `float x[10][6]`, podem-se executar os comandos a seguir.

```
for (i=0;i<10;i++)
{ for (j=0;j<6;j++)
    printf("%f", x[i][j]);
}
```

Como a matriz possui dez linhas e seis colunas, o exemplo usou duas estruturas de repetição `for` para garantir que a variável `i` assumisse todos os valores possíveis para linha (de 0 a 9) e a variável `j` assumisse todos os valores possíveis para coluna (de 0 a 5) da matriz `x`. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz foi acessada e seu conteúdo mostrado por meio do comando `printf`.



### 7.3.7 Percorrendo uma matriz

Vimos anteriormente, nos tópicos 7.3.5 e 7.3.6, formas para preencher toda uma matriz e para mostrar todas as posições de uma matriz. Em tais operações, foi necessário passar por todas as posições, ou seja, foi preciso percorrer a matriz.

Uma das formas mais simples de percorrer uma matriz pode ser por meio do uso de uma estrutura de repetição para cada dimensão da matriz. A disposição de tais estruturas de repetição define a forma como a matriz será percorrida.

A seguir, apresentaremos duas formas para percorrer uma mesma matriz, chamada *x*, contendo 3 linhas e 4 colunas.

Matriz x	0	4	5	1	10
	1	16	11	76	8
	2	9	54	32	89
		0	1	2	3

**Forma 1:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada linha. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).

```

1. for(i=0;i<3;i++)
2. {
3.     printf("Elementos da linha %d", i);
4.     for(j=0;j<4;j++)
5.     {
6.         printf("%d", x[i,j]);
7.     }
8. }
```

A primeira estrutura de repetição (linha 1) é controlada pela variável *i*, que poderá assumir valores dentro do intervalo de 0 a 2. Cada execução da estrutura *for* encontrará a segunda estrutura de repetição (linha 4), controlada pela variável *j*, que assumirá os valores dentro do intervalo de 0 a 3. Assim, cada valor assumido pela variável *i* estará associado a 4 valores da variável *j*.

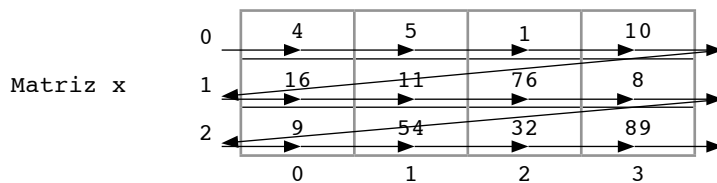
Esse arranjo resolve o problema de mostrar os elementos, separando-os por linhas, já que a variável *i* ficará com valor fixo enquanto a variável *j* assumirá valores de 0 a 3, ou seja, formará todos os pares possíveis de índices. Se *i* estiver valendo 0, serão mostrados todos os elementos da primeira linha, já que serão formados os seguintes pares: *x*[0,0], *x*[0,1], *x*[0,2] e *x*[0,3]. Depois, a variável *i* assume o valor 1 e novamente a *j* terá seus valores variando de 0 a 3. Com isso, será possível percorrer toda a segunda linha, por meio da formação dos pares *x*[1,0], *x*[1,1], *x*[1,2] e *x*[1,3]. Esse processo se repetirá para os demais valores possíveis de *i*. A tabela a seguir mostra uma simulação de execução do programa. Nessa simulação é importante observar como as variáveis *i* e *j* têm seus valores alterados.

#### Simulação:

MEMÓRIA		TELA
i	j	
0		Elementos da linha 0
0	0	4
0	1	5
0	2	1
0	3	10
1		Elementos da linha 1
1	0	16

MEMÓRIA		TELA
1	1	11
1	2	76
1	3	8
2		Elementos da linha 2
2	0	9
2	1	54
2	2	32
2	3	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis  $i$  e  $j$  e o caminho utilizado para percorrer a matriz.



**Forma 2:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada coluna. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).

```

1. for (i=0;i<4;i++)
2. {
3.     printf("Elementos da coluna %d", i);
4.     for (j=0;j<3;j++)
5.     {
6.         printf("%d", x[j, i]);
7.     }
8. }
```

A primeira estrutura de repetição (linha 1) é controlada pela variável  $i$ , que poderá assumir valores dentro do intervalo de 0 a 3. Cada execução dessa estrutura `for` encontrará a segunda estrutura de repetição (linha 4), controlada pela variável  $j$ , que assumirá os valores dentro do intervalo de 0 a 2. Assim, cada valor assumido pela variável  $i$  estará associado a 3 valores da variável  $j$ . Esse arranjo resolve o problema de mostrar os elementos, separando-os por colunas, já que a variável  $i$  ficará com valor fixo enquanto a variável  $j$  assumirá valores de 0 a 2, ou seja, formará todos os pares possíveis de índices. Se  $i$  estiver valendo 0, serão mostrados todos os elementos da primeira coluna, já que serão formados os seguintes pares:  $x[0,0]$ ,  $x[1,0]$  e  $x[2,0]$ . Depois, a variável  $i$  assumirá o valor 1 e novamente  $j$  terá seus valores variando de 0 a 2. Com isso, será possível percorrer toda a segunda coluna, por meio da formação dos pares  $x[0,1]$ ,  $x[1,1]$  e  $x[2,1]$ . Esse processo se repetirá para os demais valores possíveis de  $i$ . A tabela a seguir mostra uma simulação de execução do programa. Nessa simulação, é importante observar como as variáveis  $i$  e  $j$  têm seus valores alterados.

**Simulação:**

MEMÓRIA		TELA
$i$	$j$	
0		Elementos da coluna 0
0	0	4
0	1	16
0	2	9
1		Elementos da coluna 1
1	0	5

MEMÓRIA		TELA
1	1	11
1	2	54
2		Elementos da coluna 2
2	0	1
2	1	76
2	2	32
3		Elementos da coluna 3
3	0	10
3	1	8
3	2	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis *i* e *j* e o caminho utilizado para percorrer a matriz.

Matriz x

0	4	5	1	10
1	16	11	76	8
2	9	54	32	89
	0	1	2	3

Pelas formas de percorrer uma matriz apresentadas anteriormente, podemos observar alguns pontos que merecem atenção:

- a mudança dos valores das variáveis *i* e *j*, que controlam as estruturas de repetição, permite a formação de todos os possíveis pares de linha e coluna existentes na matriz.
- a mudança do valor da variável *i*, utilizada no `for` externo, acontece mais lentamente que a mudança da variável *j*, utilizada no `for` interno. Logo, foi a variável *i* que indicou como seria o percurso: na primeira forma apresentada, *i* variou de 0 a 2 e foi usada na primeira posição dentro dos colchetes, isso mostrou que o percurso seria horizontal, porque o índice da linha ficava parado enquanto *j* assumia todas as colunas possíveis para aquela linha. Já na segunda forma apresentada, *i* variou de 0 a 3 e foi usada na segunda posição dentro dos colchetes, indicando que o percurso seria vertical, pois o índice da coluna ficava parado enquanto *j* assumia todas as linhas possíveis para aquela coluna.

## 7.4 Matriz em JAVA

### 7.4.1 Definição de matriz

Uma matriz pode ser definida como um conjunto de variáveis de mesmo tipo e identificadas pelo mesmo nome. Essas variáveis são diferenciadas por meio da especificação de suas posições dentro dessa estrutura.

A linguagem JAVA permite a declaração de matrizes unidimensionais (mais conhecidas como *vetores* — descritos no capítulo anterior), bidimensionais e multidimensionais. As matrizes mais utilizadas possuem duas dimensões. Para cada dimensão deve ser adotado um índice.

Os índices utilizados na linguagem JAVA para identificar as posições de uma matriz começam sempre em 0 (zero) e vão até o tamanho da dimensão menos uma unidade. Os índices de uma matriz em JAVA devem sempre ser representados por um dos tipos inteiros disponíveis na linguagem.

### 7.4.2 Declaração de matriz

Matrizes em JAVA são definidas pela existência de colchetes vazios antes ou depois do nome da variável, no momento da declaração. Logo depois, deve ser feita a definição do tamanho de cada dimensão da matriz.

Em JAVA, a indicação do tamanho das dimensões da matriz pode ser feita por um valor inteiro fixo (representado por um literal<sup>3</sup> ou uma constante) ou por uma variável, cujo valor é definido em tempo de execução.

3 Literal é um valor fixo, definido quando se escreve o programa. Por exemplo: `double x=10.3`; onde 10.3 é um literal. `char mat[][]= new char[3][7]`; onde 3 e 7, escritos dentro dos colchetes, são literais.

Para utilizar uma matriz em JAVA, é preciso seguir dois passos:

1º PASSO: DECLARAR UMA VARIÁVEL QUE FARÁ REFERÊNCIA AOS ELEMENTOS

```
tipo_dos_dados nome_variável[][][]...[];
```

Os pares de colchetes vazios após o nome da variável definem que a variável será uma estrutura multidimensional.

2º PASSO: DEFINIR O TAMANHO DAS DIMENSÕES DA MATRIZ

```
nome_variável = new tipo_dos_dados [dimensão1][dimensão2][dimensão3]...[dimensãoN];
```

onde:

tipo\_dos\_dados: é o tipo de dados que poderá ser armazenado na sequência de variáveis que formam a matriz;

nome\_variável: é o nome dado à variável do tipo matriz;

[dimensão1]: representa o tamanho da primeira dimensão da matriz;

[dimensão2]: representa o tamanho da segunda dimensão da matriz;

[dimensãoN]: representa o tamanho da n-ésima dimensão da matriz.

### 7.4.3 Exemplos de matriz

Nos exemplos a seguir, são utilizadas duas linhas de comando: a primeira declara uma matriz, e a segunda define o tamanho das dimensões.

É importante ressaltar que, em JAVA, os pares de colchetes podem aparecer todos antes do nome da variável ou depois do nome da variável ou, ainda, alguns antes e outros posteriormente. Assim, todos os exemplos a seguir são válidos.

Exemplo 1:

```
float X[][];  
X = new float[2][6];
```

A declaração anterior criou uma variável chamada `x` contendo duas linhas (0 a 1) com seis colunas cada (0 a 5), capazes de armazenar números reais, como pode ser observado a seguir.

		0	1	2	3	4	5
X	0						
	1						

Exemplo 2:

```
char [][]MAT;  
MAT = new char[4][3];
```

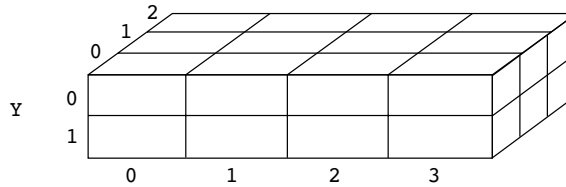
A declaração anterior criou uma variável chamada `MAT` contendo quatro linhas (0 a 3) com três colunas cada (0 a 2), capazes de armazenar caracteres, como pode ser observado a seguir.

		0	1	2
MAT	0			
	1			
	2			
	3			

Exemplo 3:

```
int [][]Y[];  
Y = new int[2][4][3];
```

A declaração anterior criou uma variável chamada `y` contendo duas linhas (0 a 1) com quatro colunas (0 a 3) e três profundidades (0 a 2), capazes de armazenar números inteiros, como pode ser observado a seguir.



Além das formas descritas nos exemplos anteriores, a linguagem JAVA também permite que os dois passos necessários para utilização de uma matriz (declaração e dimensionamento) sejam realizados em apenas uma linha de comando. Assim, nos exemplos apresentados a seguir, utilizou-se uma forma condensada de criação de matriz, onde a declaração e o dimensionamento são feitos utilizando-se uma única linha.

Exemplo 4:

```
float x[][] = new float[2][6];
```

A declaração anterior criou uma variável chamada `x` contendo duas linhas (0 a 1) com seis colunas cada (0 a 5), capazes de armazenar números reais, como pode ser observado a seguir.

		0	1	2	3	4	5
x	0						
	1						

Exemplo 5:

```
char [][]MAT = new char[4][3];
```

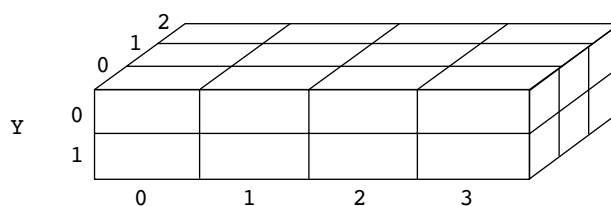
A declaração anterior criou uma variável chamada `MAT` contendo quatro linhas (0 a 3) com três colunas cada (0 a 2), capazes de armazenar caracteres, como pode ser observado a seguir.

		0	1	2
MAT	0			
	1			
	2			
	3			

Exemplo 6:

```
int [][]Y[] = new int[2][4][3];
```

A declaração anterior criou uma variável chamada `y` contendo duas linhas (0 a 1) com quatro colunas (0 a 3) e três profundidades (0 a 2), capazes de armazenar números inteiros, como pode ser observado a seguir.



É importante lembrar, também, que o tamanho das dimensões não precisa ser feito em um mesmo momento. Os exemplos que seguem mostram essa flexibilidade.

Exemplo 7:

Nesse exemplo (a numeração das linhas foi utilizada apenas para facilitar a explicação), foi definido pela linha 1 que a variável Y é uma matriz bidimensional. A linha 2 estabelece que o tamanho da primeira dimensão é 2, ou seja, essa matriz tem duas linhas, identificadas pelos índices 0 e 1. Na linha 3, foi definido o tamanho da linha 0, que passou a ter 5 colunas. Já a linha 4 define o tamanho da linha 1, que passou a ter 2 colunas.

```
1. int Y[][];  
2. Y = new int[2][];  
3. Y[0] = new int[5];  
4. Y[1] = new int[2];
```

Uma representação dessa matriz poderia ser:

Matriz Y	0					
	1					
		0	1	2	3	4

#### 7.4.4 Atribuindo valores a uma matriz

Atribuir valor a uma matriz significa armazenar uma informação em um de seus elementos, identificando de forma única por meio de seus índices.

`x[1][4]=5;` → Atribui o valor 5 à posição identificada pelos índices 1 (2ª linha) e 4 (5ª coluna).

		0	1	2	3	4	5
x	0						
	1					5	

`MAT[3][2] = 'D';` → Atribui a letra D à posição identificada pelos índices 3 (4ª linha) e 2 (3ª coluna).

		0	1	2
MAT	0			
	1			
	2			
	3			D

`Y[0][3][1] = 12;` → Atribui o valor 12 à posição identificada pelos índices 0 (1ª linha), 3 (4ª coluna) e 1 (2ª profundidade).

A 3D perspective diagram of a 4x2x2 grid of cubes. The front face is labeled with 'Y' on the left, '0' and '1' on the vertical axis, and '0', '1', '2', '3' on the horizontal axis. The top face is labeled with '0' and '1' on the left edge, and '2' on the top edge. The right face is labeled with '12' on the vertical edge and '12' on the horizontal edge.

### 7.4.5 Preenchendo uma matriz

Preencher uma matriz significa percorrer todos os seus elementos, atribuindo-lhes um valor. Esse valor pode ser recebido do usuário, por meio do teclado, ou pode ser gerado pelo programa.

Nos exemplos a seguir, todos os elementos de uma matriz bidimensional são acessados e a eles são atribuídos valores inteiros, digitados pelo usuário.

Exemplo 1:

```
int X[][] = new int[7][3];
Scanner e = new Scanner(System.in);
for (i=0;i<7;i++)
{
    for (j=0;j<3;j++)
        X[i][j] = e.nextInt();
}
```

Como a matriz possui 7 linhas e 3 colunas, o exemplo apresentou duas estruturas de repetição `for` para garantir que a variável `i` assumisse todos os valores possíveis para linha (de 0 a 6) e a variável `j` assumisse todos os valores possíveis para coluna (de 0 a 2) da matriz `x`. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz foi preenchida por um valor digitado pelo usuário por meio do método `nextInt()` da classe `Scanner`.

Exemplo 2:

```
1. int MAT[][];
2. MAT = new int[3][];
3. MAT[0]= new int[2];
4. MAT[1]= new int[5];
5. MAT[2]= new int[3];
6. Scanner e = new Scanner(System.in);
7. for (i=0;i<MAT.length;i++)
8. {
9.     for (j=0;j<MAT[i].length;j++)
10.         X[i][j] = e.nextInt();
11. }
```

No exemplo 2, onde a numeração não faz parte do código e é utilizada apenas para explicação, usamos uma matriz cujas linhas têm tamanhos diferentes.

A linha 1 indicou que `MAT` é uma matriz bidimensional. A linha 2 definiu que a quantidade de linhas de `MAT` é 3, mas não estabeleceu a quantidade de colunas. A linha 3 determinou que o tamanho da linha 0 de `MAT` é 2. A linha 4 definiu que o tamanho da linha 1 de `MAT` é 5. A linha 5 estipulou que o tamanho da linha 2 de `MAT` é 3.

Dessa variação de tamanho de linhas surge a necessidade de ter algum recurso para recuperar, quando for preciso, o tamanho de cada dimensão de uma `Array` (vetor ou matriz). Esse recurso é o atributo `length`.

Na linha 7, podemos ver o uso de `MAT.length`, que significa o tamanho da primeira dimensão de `MAT`, ou seja, a quantidade de linhas, que é 3.

Na linha 9, podemos ver `MAT[i].length`, que significa o tamanho da linha `i` de `MAT`. Quando `i` estiver valendo 0, esse tamanho será 2; quando `i` estiver valendo 1, esse tamanho será 5; e quando `i` estiver valendo 2, esse tamanho será 3.

Com as duas estruturas de repetição `for`, pode-se garantir que a variável `i` assuma todos os valores possíveis para cada linha (de 0 a 2) e a variável `j` assuma todos os valores possíveis para coluna, respeitando os dimensionamentos feitos. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz foi preenchida por um valor digitado pelo usuário, por meio do método `nextInt()` da classe `Scanner`.

### 7.4.6 Mostrando os elementos de uma matriz

Pode-se, também, percorrer todos os elementos da matriz, acessando seu conteúdo.

Exemplo 1:

```
for (i=0; i<10; i++)
{
    for (j=0; j<6; j++)
        System.out.println(X[i][j]);
}
```

No exemplo 1 foram mostrados todos os elementos de uma matriz contendo dez linhas e seis colunas. Observe que são usados dois índices, *i* e *j*. Esses índices estão atrelados a estruturas de repetição que mantêm a variação de ambos dentro de intervalos permitidos. Ou seja, o índice *i*, que representa as linhas, varia de 0 a 9 e o índice *j*, que representa as colunas, varia de 0 a 5.

Exemplo 2:

```
1. for (i=0; i<MAT.length; i++)
2. {
3.     for (j=0; j<MAT[i].length; j++)
4.         System.out.println(MAT[i][j]);
5. }
```

No exemplo 2, também foram mostrados todos os elementos de uma matriz. Contudo, em virtude do uso do atributo `length` (apresentado em detalhes na Seção 7.4.5), as linhas dessa matriz podem ter tamanhos diferentes.

Na linha 1, podemos ver o uso de `MAT.length`, que significa o tamanho da primeira dimensão de `MAT`, ou seja, a quantidade de linhas. Na linha 3, podemos ver `MAT[i].length`, que significa o tamanho da linha *i* de `MAT`.

Com as duas estruturas de repetição `for`, pode-se garantir que a variável *i* assuma todos os valores possíveis para linha e a variável *j* assuma todos os valores possíveis para coluna, respeitando os dimensionamentos feitos. Assim, para cada execução das estruturas de repetição, uma posição diferente da matriz foi acessada e seu valor mostrado por meio do método `println()` da classe `System`.

### 7.4.7 Percorrendo uma matriz

Vimos anteriormente, nos tópicos 7.4.5 e 7.4.6, formas para preencher toda uma matriz e para mostrar todas as posições de uma matriz. Em tais operações, foi necessário passar por todas as posições, ou seja, foi preciso percorrer a matriz.

Uma das formas mais simples de percorrer uma matriz pode ser por meio do uso de uma estrutura de repetição para cada dimensão da matriz. A ordem de disposição de tais estruturas de repetição define a forma como a matriz será percorrida.

A seguir, apresentaremos duas formas para percorrer uma mesma matriz, chamada *x*, contendo 3 linhas e 4 colunas.

Matriz x	0	4	5	1	10
	1	16	11	76	8
	2	9	54	32	89
		0	1	2	3

**Forma 1:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada linha. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).



```

1. for(i=0;i<3;i++)
2. {
3.     System.out.println("Elementos da linha " + i);
4.     for(j=0;j<4;j++)
5.     {
6.         System.out.println(x[i,j]);
7.     }
8. }

```

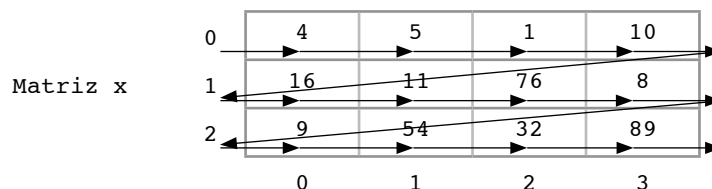
A primeira estrutura de repetição (linha 1) é controlada pela variável *i*, que poderá assumir valores dentro do intervalo de 0 a 2. Cada execução da estrutura *for* encontrará a segunda estrutura de repetição (linha 4), controlada pela variável *j*, que assumirá os valores dentro do intervalo de 0 a 3. Assim, cada valor assumido pela variável *i* estará associado a 4 valores da variável *j*.

Esse arranjo resolve o problema de mostrar os elementos, separando-os por linhas, já que a variável *i* ficará com valor fixo enquanto a variável *j* assumirá valores de 0 a 3, ou seja, formará todos os pares possíveis de índices. Se *i* estiver valendo 0, serão mostrados todos os elementos da primeira linha, já que serão formados os seguintes pares: *x*[0,0], *x*[0,1], *x*[0,2] e *x*[0,3]. Depois, a variável *i* assume o valor 1 e novamente a *j* terá seus valores variando de 0 a 3. Com isso, será possível percorrer toda a segunda linha, por meio da formação dos pares *x*[1,0], *x*[1,1], *x*[1,2] e *x*[1,3]. Esse processo se repetirá para os demais valores possíveis de *i*. A tabela a seguir mostra uma simulação de execução do programa. Nessa simulação, é importante observar como as variáveis *i* e *j* têm seus valores alterados.

#### Simulação:

MEMÓRIA		TELA
<i>i</i>	<i>j</i>	
0		Elementos da linha 0
0	0	4
0	1	5
0	2	1
0	3	10
1		Elementos da linha 1
1	0	16
1	1	11
1	2	76
1	3	8
2		Elementos da linha 2
2	0	9
2	1	54
2	2	32
2	3	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis *i* e *j* e o caminho utilizado para percorrer a matriz.



**Forma 2:** precisamos percorrer a matriz, de tal forma que seja possível mostrar todos os elementos gravados em cada coluna. Para isso, utilizaremos duas estruturas de repetição, conforme mostrado a seguir (a numeração à esquerda não faz parte do programa, servirá apenas para facilitar a explicação).

```

1. for (i=0;i<4;i++)
2. {
3.     System.out.println("Elementos da coluna " + i);
4.     for (j=0;j<3;j++)
5.     {
6.         System.out.println(x[j,i]);
7.     }
8. }

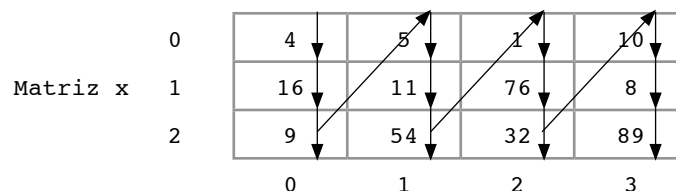
```

A primeira estrutura de repetição (linha 1) é controlada pela variável *i*, que poderá assumir valores dentro do intervalo de 0 a 3. Cada execução da estrutura encontrará a segunda estrutura de repetição (linha 4), controlada pela variável *j*, que assumirá os valores dentro do intervalo de 0 a 2. Assim, cada valor assumido pela variável *i* estará associado a 3 valores da variável *j*. Esse arranjo resolve o problema de mostrar os elementos, separando-os por colunas, já que a variável *i* ficará com valor fixo enquanto a variável *j* assumirá valores de 0 a 2, ou seja, formará todos os pares possíveis de índices. Se *i* estiver valendo 0, serão mostrados todos os elementos da primeira coluna, já que serão formados os seguintes pares: *x*[0,0], *x*[1,0] e *x*[2,0]. Depois, a variável *i* assumirá o valor 1 e novamente *j* terá seus valores variando de 0 a 2. Com isso, será possível percorrer toda a segunda coluna, por meio da formação dos pares *x*[0,1], *x*[1,1] e *x*[2,1]. Esse processo se repetirá para os demais valores possíveis de *i*. A tabela a seguir mostra uma simulação de execução do algoritmo. Nessa simulação é importante observar como as variáveis *i* e *j* têm seus valores alterados.

#### Simulação:

MEMÓRIA		TELA
<i>i</i>	<i>j</i>	
0		Elementos da coluna 0
0	0	4
0	1	16
0	2	9
1		Elementos da coluna 1
1	0	5
1	1	11
1	2	54
2		Elementos da coluna 2
2	0	1
2	1	76
2	2	32
3		Elementos da coluna 3
3	0	10
3	1	8
3	2	89

A figura a seguir dá outra visão da forma utilizada para percorrer a matriz. A direção das setas indica a mudança no valor das variáveis *i* e *j* e o caminho utilizado para percorrer a matriz.



Pelas formas de percorrer uma matriz apresentadas anteriormente, podemos observar alguns pontos que merecem atenção:

- a mudança dos valores das variáveis *i* e *j*, que controlam as estruturas de repetição, permite a formação de todos os possíveis pares de linha e coluna existentes na matriz.

- a mudança do valor da variável *i*, utilizada no *for* externo, acontece mais lentamente que a mudança da variável *j*, utilizada no *for* interno. Logo, foi a variável *i* que indicou como seria o percurso: na primeira forma apresentada, *i* variou de 0 a 2 e foi usada na primeira posição dentro do colchetes, isso indicou que o percurso seria horizontal, pois o índice da linha ficava parado, enquanto *j* assumia todas as colunas possíveis para aquela linha. Já na segunda forma apresentada, *i* variou de 0 a 3 e foi usada na segunda posição dentro dos colchetes, indicando que o percurso seria vertical, porque o índice da coluna ficava parado, enquanto *j* assumia todas as linhas possíveis para aquela coluna.

## EXERCÍCIOS RESOLVIDOS

1. Faça um programa que preencha uma matriz *M* ( $2 \times 2$ ), calcule e mostre a matriz *R*, resultante da multiplicação dos elementos de *M* pelo seu maior elemento.

**ALGORITMO** SOLUÇÃO:

```
ALGORITMO
  DECLARE mat[2,2], resultado[2,2], i, j, maior NUMÉRICO
  PARA i ← 1 ATÉ 2 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 2 FAÇA
        INÍCIO
          LEIA mat[i,j]
        FIM
      FIM
    maior ← mat[1,1]
    PARA i ← 1 ATÉ 2 FAÇA
      INÍCIO
        PARA j ← 1 ATÉ 2 FAÇA
          INÍCIO
            SE mat[i,j] > maior
              ENTÃO maior ← mat[i,j]
          FIM
        FIM
      PARA i ← 1 ATÉ 2 FAÇA
        INÍCIO
          PARA j ← 1 ATÉ 2 FAÇA
            INÍCIO
              resultado[i,j] ← maior * mat[i,j]
            FIM
          FIM
        FIM
      PARA i ← 1 ATÉ 2 FAÇA
        INÍCIO
          PARA j ← 1 ATÉ 2 FAÇA
            INÍCIO
              ESCREVA resultado[i,j]
            FIM
          FIM
        FIM
      FIM
    FIM
  FIM
FIM_ALGORITMO.
```

**PASCAL**

SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX1.PAS e \EXERC\CAP7\PASCAL\EX1.EXE

**C/C++**

SOLUÇÃO:

\EXERC\CAP7\C++\EX1.CPP e \EXERC\CAP7\C++\EX1.EXE

**JAVA**

SOLUÇÃO:

\EXERC\CAP7\JAVA\EX1.java e \EXERC\CAP7\JAVA\EX1.class

- 2.** Faça um programa que preencha uma matriz  $10 \times 3$  com as notas de dez alunos em três provas. O programa deverá mostrar um relatório com o número dos alunos (número da linha) e a prova em que cada aluno obteve menor nota. Ao final do relatório, deverá mostrar quantos alunos tiveram menor nota em cada uma das provas: na prova 1, na prova 2 e na prova 3.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE notas[10,3], q1, q2, q3, menor, p_menor, i, j NUMÉRICO
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                LEIA notas[i,j]
            FIM
        FIM
    FIM
q1 ← 0
q2 ← 0
q3 ← 0
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        ESCRIVA i
        menor ← notas[i,1]
        p_menor ← 1
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                SE notas[i, j] < menor
                    ENTÃO INÍCIO
                        menor ← notas[i, j]
                        p_menor ← j
                    FIM
            FIM
        ESCRIVA p_menor
        SE p_menor = 1
            ENTÃO q1 ← q1 + 1
        SE p_menor = 2
            ENTÃO q2 ← q2 + 1
        SE p_menor = 3
            ENTÃO q3 ← q3 + 1
    FIM
ESCREVA q1, q2, q3
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX2.PAS e \EXERC\CAP7\PASCAL\EX2.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX2.CPP e \EXERC\CAP7\C++\EX2.EXE

**JAVA** SOLUÇÃO:

\EXERC\CAP7\JAVA\EX2.java e \EXERC\CAP7\JAVA\EX2.class

- 3.** Faça um programa que preencha:

- um vetor com oito posições, contendo nomes de lojas;
- outro vetor com quatro posições, com nomes de produtos;
- uma matriz com os preços de todos os produtos em cada loja.

O programa deverá mostrar todas as relações (nome do produto — nome da loja) em que o preço não ultrapasse R\$ 120,00.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
  DECLARE lojas[8], produtos[4] LITERAL
    preços[4,8], i, j NUMÉRICO
  PARA j ← 1 ATÉ 8 FAÇA
    INÍCIO
      LEIA lojas[j]
    FIM
  PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
      LEIA produtos[i]
    FIM
  PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 8 FAÇA
        INÍCIO
          LEIA preços[i, j]
        FIM
      FIM
    PARA i ← 1 ATÉ 4 FAÇA
      INÍCIO
        PARA j ← 1 ATÉ 8 FAÇA
          INÍCIO
            SE preços[i, j] < 120
              ENTÃO ESCREVA produtos[i], lojas[j]
          FIM
        FIM
      FIM
    FIM
  FIM
FIM_ALGORITMO.

```

**PASCAL**

SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX3.PAS e \EXERC\CAP7\PASCAL\EX3.EXE

**C/C++**

SOLUÇÃO:

\EXERC\CAP7\C++\EX3.CPP e \EXERC\CAP7\C++\EX3.EXE

**JAVA**

SOLUÇÃO:

\EXERC\CAP7\JAVA\EX3.java e \EXERC\CAP7\JAVA\EX3.class

- 4.** Crie um programa que preencha uma matriz  $10 \times 20$  com números inteiros e some cada uma das linhas, armazenando o resultado das somas em um vetor. A seguir, o programa deverá multiplicar cada elemento da matriz pela soma da linha correspondente e mostrar a matriz resultante.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
  DECLARE mat[10,20], soma[10], i, j NUMÉRICO
  PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 20 FAÇA
        INÍCIO
          LEIA mat[i, j]
        FIM
      FIM
    FIM
  FIM


```

```


PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        soma[i] ← 0
        PARA j ← 1 ATÉ 20 FAÇA
            INÍCIO
                soma[i] ← soma[i] + mat[i, j]
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 20 FAÇA
            INÍCIO
                mat[i, j] ← mat[i, j] * soma[i]
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 20 FAÇA
            INÍCIO
                ESCREVA mat[i, j]
            FIM
        FIM
    FIM
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

 \EXERC\CAP7\PASCAL\EX4.PAS e \EXERC\CAP7\PASCAL\EX4.EXE

**C/C++** SOLUÇÃO:

 \EXERC\CAP7\C++\EX4.CPP e \EXERC\CAP7\C++\EX4.EXE

**JAVA** SOLUÇÃO:

 \EXERC\CAP7\JAVA\EX4.java e \EXERC\CAP7\JAVA\EX4.class

- 5.** Na teoria dos sistemas, define-se o elemento MINMAX de uma matriz como o maior elemento da linha em que se encontra o menor elemento da matriz. Elabore um programa que carregue uma matriz  $4 \times 7$  com números reais, calcule e mostre seu MINMAX e sua posição (linha e coluna).

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
    DECLARE mat[4,7], menor, maior, i, j, l_menor, col NUMÉRICO
    PARA i ← 1 ATÉ 4 FAÇA
        INÍCIO
            PARA j ← 1 ATÉ 7 FAÇA
                INÍCIO
                    LEIA mat[i, j]
                FIM
            FIM
        FIM
    menor ← mat[1, 1]
    l_menor ← 1
    PARA i ← 1 ATÉ 4 FAÇA
        INÍCIO
            PARA j ← 1 ATÉ 7 FAÇA
                INÍCIO
                    SE mat[i, j] < menor

```

```

        ENTÃO INÍCIO
            menor ← mat[i, j]
            l_menor ← i
        FIM
    FIM
FIM
maior ← mat[l_menor, 1]
col ← 1
PARA j ← 1 ATÉ 7 FAÇA
    INÍCIO
        SE mat[l_menor, j] > maior
            ENTÃO INÍCIO
                maior ← mat[l_menor, j]
                col ← j
            FIM
    FIM
FIM
ESCREVA maior, l_menor, col
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX5.PAS e \EXERC\CAP7\PASCAL\EX5.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX5.CPP e \EXERC\CAP7\C++\EX5.EXE

**JAVA**SOLUÇÃO:

EXERC\CAP7\JAVA\EX5.java e \EXERC\CAP7\JAVA\EX5.class

- 6.** Crie um programa que preencha uma primeira matriz de ordem  $4 \times 5$  e uma segunda matriz  $5 \times 2$ . O programa deverá, também, calcular e mostrar a matriz resultante do produto matricial das duas matrizes anteriores, armazenando-o em uma terceira matriz de ordem  $4 \times 2$ .

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
    DECLARE a[4, 5], b[5, 2], c[4, 2] NUMÉRICO
        i, j, k, soma, mult NUMÉRICO
    PARA i ← 1 ATÉ 4 FAÇA
        INÍCIO
            PARA j ← 1 ATÉ 5 FAÇA
                INÍCIO
                    LEIA a[i, j]
                FIM
            FIM
        PARA i ← 1 ATÉ 5 FAÇA
            INÍCIO
                PARA j ← 1 ATÉ 2 FAÇA
                    INÍCIO
                        LEIA b[i, j]
                    FIM
                FIM
            FIM
        soma ← 0
        PARA i ← 1 ATÉ 4 FAÇA
            INÍCIO
                PARA k ← 1 ATÉ 2 FAÇA

```

```

        INÍCIO
        PARA j ← 1 ATÉ 5 FAÇA
            INÍCIO
                mult ← a[i, j] * b[j, k]
                soma ← soma + mult
            FIM
        c[i, k] ← soma
        soma ← 0
    FIM
FIM
FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 2 FAÇA
            INÍCIO
                ESCREVA c[i, j]
            FIM
        FIM
    FIM
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX6.PAS e \EXERC\CAP7\PASCAL\EX6.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX6.CPP e \EXERC\CAP7\C++\EX6.EXE

**JAVA** SOLUÇÃO:

\EXERC\CAP7\JAVA\EX6.java e \EXERC\CAP7\JAVA\EX6.class

- 7.** Um elemento  $a_{ij}$  de uma matriz é dito ponto de sela da matriz  $A$  se, e somente se,  $a_{ij}$  for, ao mesmo tempo, o menor elemento da linha  $i$  e o maior elemento da coluna  $j$ . Faça um programa que carregue uma matriz de ordem  $5 \times 7$ , verifique se a matriz possui ponto de sela e, se possuir, mostre seu valor e sua localização.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE mat[5,7] NUMÉRICO
        i, j, maior, menor, linha, coluna, cont NUMÉRICO
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 7 FAÇA
            INÍCIO
                LEIA mat[i, j]
            FIM
        FIM
    cont ← 0
    PARA i ← 1 ATÉ 5 FAÇA
        INÍCIO
            menor ← mat[i, 1]
            linha ← i
            coluna ← 0
            PARA j ← 1 ATÉ 7 FAÇA
                INÍCIO
                    SE mat[i,j] < menor
                        ENTÃO INÍCIO
                            menor ← mat[i, j]
                            linha ← i

```



```

                                coluna ← j
                                FIM
                            FIM
                    maior ← mat[1, coluna]
                    PARA j ← 1 ATÉ 5 FAÇA
                        INÍCIO
                            SE mat[j,coluna] > maior
                                ENTÃO INÍCIO
                                    maior ← mat[j, coluna]
                                FIM
                        FIM
                    FIM
                    SE menor = maior
                        ENTÃO INÍCIO
                            ESCREVA "Ponto de sela = ", maior, " na posição", linha, coluna
                            cont ← cont + 1
                        FIM
                    FIM
                FIM
            SE cont = 0
                ENTÃO ESCREVA "Não existe ponto de sela nesta matriz"
            FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

```
\EXERC\CAP7\PASCAL\EX7.PAS e \EXERC\CAP7\PASCAL\EX7.EXE
```

**C/C++**SOLUÇÃO:

```
\EXERC\CAP7\C++\EX7.CPP e \EXERC\CAP7\C++\EX7.EXE
```

**JAVA**SOLUÇÃO:

```
\EXERC\CAP7\JAVA\EX7.java e \EXERC\CAP7\JAVA\EX7.class
```

- 8.** Elabore um programa que preencha uma matriz  $6 \times 4$  com números inteiros, calcule e mostre quantos elementos dessa matriz são maiores que 30 e, em seguida, monte uma segunda matriz com os elementos diferentes de 30. No lugar do número 30, da segunda matriz, coloque o número zero.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
    DECLARE mat1[6,4], mat2[6,4], i, j, qtde NUMÉRICO
    PARA i ← 1 ATÉ 6 FAÇA
        INÍCIO
            PARA j ← 1 ATÉ 4 FAÇA
                INÍCIO
                    LEIA mat1[i, j]
                FIM
            FIM
        qtde ← 0
        PARA i ← 1 ATÉ 6 FAÇA
            INÍCIO
                PARA j ← 1 ATÉ 4 FAÇA
                    INÍCIO
                        SE mat1[i, j] > 30
                            ENTÃO qtde ← qtde + 1
                    FIM
                FIM
            FIM
        PARA i ← 1 ATÉ 6 FAÇA

```

```

INÍCIO
PARA j ← 1 ATÉ 4 FAÇA
    INÍCIO
        SE mat1[i, j] = 30
            ENTÃO mat2[i, j] ← 0
            SENÃO mat2[i, j] ← mat1[i, j]
        FIM
    FIM
FIM

```

ESCREVA qtde

```

PARA i ← 1 ATÉ 6 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                ESCRIBA mat2[i, j]
            FIM
        FIM
    FIM
FIM ALGORITMO.

```

**PASCAL**      SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX8.PAS e \EXERC\CAP7\PASCAL\EX8.EXE

C/C++

\EXERC\CAP7\C++\EX8.CPP e \EXERC\CAP7\C++\EX8.EXE

**JAVA** SOLUÇÃO:

```
\EXERC\CAP7\JAVA\EX8.java e \EXERC\CAP7\JAVA\EX8.class
```

- 9.** Crie um programa que preencha uma matriz  $15 \times 5$  com números inteiros, calcule e mostre quais elementos da matriz se repetem e quantas vezes cada um se repete.

**ALGORITMO** SOLUÇÃO:

```
ALGORITMO  
DECLARE mat[15,5], rep[15,5], vezes[15,5] NUMÉRICO  
      i, j, k, l, lin, lin2, col, x, num, qtde, achou NUMÉRICO  
PARA i ← 1 ATÉ 15 FAÇA  
    INÍCIO  
      PARA j ← 1 ATÉ 5 FAÇA  
        INÍCIO  
          LEIA mat[i,j]  
        FIM  
      FIM  
    FIM  
lin ← 1  
col ← 1  
PARA i ← 1 ATÉ 15 FAÇA  
  INÍCIO  
    PARA j ← 1 ATÉ 5 FAÇA  
      INÍCIO  
        qtde ← 1  
        num ← mat[i,j]  
        PARA k ← 1 ATÉ 15 FAÇA  
          INÍCIO  
            PARA l ← 1 ATÉ 5 FAÇA  
              INÍCIO  
                SE i <> k OU j <> l
```

```

        ENTÃO SE mat[k,l] = num
            ENTÃO qtde ← qtde + 1
        FIM
    FIM
SE qtde > 1
    ENTÃO INÍCIO
        achou ← 0
        SE col = 1
            ENTÃO lin2 ← lin-1
        SENÃO lin2 ← lin
        PARA k ← 1 ATÉ lin2 FAÇA
            INÍCIO
                SE (k < lin2)
                    ENTÃO x ← 5
                SENÃO x ← col-1
                PARA l ← 1 ATÉ x FAÇA
                    INÍCIO
                        SE num = rep[k,l]
                            ENTÃO achou ← 1
                    FIM
            FIM
        SE achou = 0
            ENTÃO INÍCIO
                rep[lin,col] ← num
                vezes[lin,col] ← qtde
                col ← col + 1
                SE col > 5
                    ENTÃO INÍCIO
                        lin ← lin + 1
                        col ← 1
                    FIM
            FIM
        FIM
    FIM
FIM
PARA i ← 1 ATÉ lin FAÇA
    INÍCIO
        SE i < lin
            ENTÃO x ← 5
        SENÃO x ← col-1
        PARA j ← 1 ATÉ x FAÇA
            INÍCIO
                ESCREVA "O número ",rep[i,j], " repetiu ",vezes[i,j],"
                    vezes"
            FIM
        FIM
FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX9.PAS e \EXERC\CAP7\PASCAL\EX9.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX9.CPP e \EXERC\CAP7\C++\EX9.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX9.java e \EXERC\CAP7\JAVA\EX9.class

**10.** Elabore um programa que preencha uma matriz  $10 \times 10$  com números inteiros, execute as trocas especificadas a seguir e mostre a matriz resultante:

- a linha 2 com a linha 8;
- a coluna 4 com a coluna 10;
- a diagonal principal com a diagonal secundária;
- a linha 5 com a coluna 10.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
  DECLARE mat[10,10], aux, i, j NUMÉRICO
  PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 10 FAÇA
        INÍCIO
          LEIA mat[i, j]
        FIM
      FIM
    PARA i ← 1 ATÉ 10 FAÇA
      INÍCIO
        aux ← mat[2, i]
        mat[2,i] ← mat[8,i]
        mat[8, i] ← aux
      FIM
    PARA i ← 1 ATÉ 10 FAÇA
      INÍCIO
        aux ← mat[i, 4]
        mat[i, 4] ← mat[i,10]
        mat[i, 10] ← aux
      FIM
    j ← 10
    PARA i ← 1 ATÉ 10 FAÇA
      INÍCIO
        aux ← mat[i, i]
        mat[i, i] ← mat[i, j]
        mat[i, j] ← aux
        j ← j - 1
      FIM
    PARA j ← 1 ATÉ 10 FAÇA
      INÍCIO
        aux ← mat[5, j]
        mat[5, j] ← mat[j,10]
        mat[j, 10] ← aux
      FIM
    PARA i ← 1 ATÉ 10 FAÇA
      INÍCIO
        PARA j ← 1 ATÉ 10 FAÇA
          INÍCIO
            ESCREVA mat[i, j]
          FIM
        FIM
      FIM
    FIM
  FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX10.PAS e \EXERC\CAP7\PASCAL\EX10.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX10.CPP e \EXERC\CAP7\C++\EX10.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX10.java e \EXERC\CAP7\JAVA\EX10.class

- 11.** Crie um programa que preencha uma matriz  $8 \times 8$  com números inteiros e mostre uma mensagem dizendo se a matriz digitada é simétrica. Uma matriz só pode ser considerada simétrica se  $A[i, j] = A[j, i]$ .

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
  DECLARE mat[8,8], i, j NUMÉRICO
    simetria LÓGICO
  PARA i ← 1 ATÉ 8 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 8 FAÇA
        INÍCIO
          LEIA mat[i, j]
        FIM
      FIM
    FIM
  simetria ← verdadeiro
  PARA i ← 1 ATÉ 8 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 8 FAÇA
        INÍCIO
          SE mat[i, j] ≠ mat[j, i]
            ENTÃO simetria ← falso
          FIM
        FIM
      FIM
  SE simetria = verdadeiro
    ENTÃO ESCREVA "Matriz Simétrica"
  SENÃO ESCREVA "Matriz Assimétrica"
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX11.PAS e \EXERC\CAP7\PASCAL\EX11.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX11.CPP e \EXERC\CAP7\C++\EX11.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX11.java e \EXERC\CAP7\JAVA\EX11.class

- 12.** Elabore um programa que preencha uma matriz  $4 \times 4$  com números inteiros e verifique se essa matriz forma o chamado quadrado mágico. Um quadrado mágico é formado quando a soma dos elementos de cada linha é igual à soma dos elementos de cada coluna dessa linha, é igual à soma dos elementos da diagonal principal e, também, é igual à soma dos elementos da diagonal secundária.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
  DECLARE mat[4,4] NUMÉRICO
    soma_linha[4] NUMÉRICO
    soma_coluna[4] NUMÉRICO
    soma_diagp, soma_diags, i, j, compara NUMÉRICO
    q_magico LÓGICO
  PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 4 FAÇA

```

```

        INÍCIO
            LEIA mat[i,j]
        FIM
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        soma_linha[i] ← 0
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                soma_linha[i] ← soma_linha[i] + mat[i,j]
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        soma_coluna[i] ← 0
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                soma_coluna[i] ← soma_coluna[i] + mat[j,i]
            FIM
        FIM
    FIM
soma_diagp ← 0
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        soma_diagp ← soma_diagp + mat[i,i]
    FIM
soma_diags ← 0
j ← 4
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        soma_diags ← soma_diags + mat[i,j]
        j ← j - 1
    FIM
q_magico ← verdadeiro
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                SE soma_linha[i] ≠ soma_coluna[j]
                    ENTÃO q_magico ← falso
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        SE soma_linha[i] ≠ soma_diagp
            ENTÃO q_magico ← falso
        FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        SE soma_linha[i] ≠ soma_diags
            ENTÃO q_magico ← falso
        FIM
SE q_magico = verdadeiro
    ENTÃO ESCREVA "Forma quadrado mágico"
    SENÃO ESCREVA "Não forma quadrado mágico"
FIM_ALGORITMO.

```



SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX12.PAS e \EXERC\CAP7\PASCAL\EX12.EXE



SOLUÇÃO:

\EXERC\CAP7\C++\EX12.CPP e \EXERC\CAP7\C++\EX12.EXE



SOLUÇÃO:

\EXERC\CAP7\JAVA\EX12.java e \EXERC\CAP7\JAVA\EX12.class

**13.** Faça um programa que preencha:

- um vetor com os nomes de cinco produtos;
- uma matriz  $5 \times 4$  com os preços dos cinco produtos em quatro lojas diferentes;
- outro vetor com o custo do transporte dos cinco produtos.

O programa deverá preencher uma segunda matriz  $5 \times 4$  com os valores dos impostos de cada produto, de acordo com a tabela a seguir.

PREÇO	% DE IMPOSTO
Até R\$ 50,00	5
Entre R\$ 50,00 e R\$ 100,00 (inclusive)	10
Acima de R\$ 100,00	20

O programa deverá mostrar, ainda, um relatório com o nome do produto, o número da loja onde o produto é encontrado, o valor do imposto a pagar, o custo de transporte, o preço e o preço final (preço acrescido do valor do imposto e do custo do transporte).

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE nome[5] LITERAL
      preco, imp[5,4], custo[5], i, j, final NUMÉRICO
NUMÉRICO PARA i ← 1 ATÉ 5 FAÇA
  INÍCIO
    LEIA nome[i]
  FIM
PARA i ← 1 ATÉ 5 FAÇA
  INÍCIO
    PARA j ← 1 ATÉ 4 FAÇA
      INÍCIO
        LEIA preco[i,j]
      FIM
    FIM
  PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
      LEIA custo[i]
    FIM
  PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 4 FAÇA
        INÍCIO
          SE preco[i,j] ≤ 50
            ENTÃO imp[i,j] ← preco[i,j] * 5 / 100

```

```

        SENÃO SE preco[i,j] > 50 E preco[i,j] <= 100
            ENTÃO imp[i,j] ← preco[i,j]*10/100
            SENÃO imp[i,j] ← preco[i,j]*15/100

    FIM
FIM
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        ESCREVA "Nome do produto ", nome[i]
        ESCREVA "Custo = ", custo[i]
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                final ← preco[i,j] + imp[i,j] + custo[i]
                ESCREVA "Imposto na loja ", j, " = ", imp[i,j]
                ESCREVA "Preço na loja ", j, " = ", preco[i,j]
                ESCREVA "Preço final na loja ", j, " = ", final
            FIM
        FIM
    FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX13.PAS e \EXERC\CAP7\PASCAL\EX13.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX13.CPP e \EXERC\CAP7\C++\EX13.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX13.java e \EXERC\CAP7\JAVA\EX13.class

**14.** Faça um programa que receba:

- um vetor com o nome de cinco cidades diferentes;
- uma matriz  $5 \times 5$  com a distância entre as cidades, e na diagonal principal deve ser colocada automaticamente a distância zero, ou seja, não deve ser permitida a digitação;
- o consumo de combustível de um veículo, ou seja, quantos quilômetros esse veículo percorre com um litro de combustível.

O programa deverá calcular e mostrar:

- os percursos que não ultrapassem 250 quilômetros (os percursos são compostos pelos nomes das cidades de origem e pelos nomes das cidades de destino);
- todos os percursos (nome da cidade de origem e nome da cidade de destino), junto com a quantidade de combustível necessária para o veículo percorrê-los.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE cidade[5] LITERAL
        distancia[5,5], i, j, consumo, qtde NUMÉRICO
NUMÉRICO PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        LEIA cidade[i]
    FIM
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 5 FAÇA
            INÍCIO
                SE i = j

```



```

                                ENTÃO distancia[i, j] ← 0
                                SENÃO LEIA distancia[i, j]
                                FIM
                                FIM
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 5 FAÇA
            INÍCIO
                SE distancia[i, j] ≤ 250 E distancia[i, j] > 0
                    ENTÃO ESCRIVA "Distancia: ", distancia[i, j], " entre ", cidade[i],
                        " e ", cidade[j]
            FIM
        FIM
    FIM
LEIA consumo
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 5 FAÇA
            INÍCIO
                SE i ≠ j
                    ENTÃO INÍCIO
                        qtde ← distancia[i, j] / consumo
                        ESCRIVA "Consumo entre ", cidade[i], " e ", cidade[j], " =
                            ", qtde
                    FIM
            FIM
        FIM
    FIM
FIM
FIM
FIM_ALGORITMO.

```

**PASCAL**

SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX14.PAS e \EXERC\CAP7\PASCAL\EX14.EXE

**C/C++**

SOLUÇÃO:

\EXERC\CAP7\C++\EX14.CPP e \EXERC\CAP7\C++\EX14.EXE

**JAVA**

SOLUÇÃO:

\EXERC\CAP7\JAVA\EX14.java e \EXERC\CAP7\JAVA\EX14.class

**15.** Elabore um programa que preencha:

- um vetor com cinco números inteiros;
- outro vetor com dez números inteiros;
- uma matriz  $4 \times 3$ , também com números inteiros.

O programa deverá calcular e mostrar:

- o maior elemento do primeiro vetor multiplicado pelo menor elemento do segundo vetor. O resultado dessa multiplicação, adicionado aos elementos digitados na matriz, dará origem a uma segunda matriz (resultante);
- a soma dos elementos pares de cada linha da matriz resultante;
- a quantidade de elementos entre 1 e 5 em cada coluna da matriz resultante.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE vet1[5], vet2[10], mat[4,3], mat_result[4,3] NUMÉRICO
i, j, maior, menor, mult, soma, qtde NUMÉRICO
PARA i ← 1 ATÉ 5 FAÇA

```

```


    INÍCIO
        LEIA vet1[i]
    FIM
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        LEIA vet2[i]
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                LEIA mat[i,j]
            FIM
        FIM
maior ← vet1[1]
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        SE vet1[i] > maior
            ENTÃO maior ← vet1[i]
    FIM
menor ← vet2[1]
PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
        SE vet2[i] < menor
            ENTÃO menor ← vet2[i]
    FIM
mult ← maior * menor
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                mat_result[i, j] ← mat[i, j] + mult
            FIM
        FIM
    FIM
ESCREVA "Matriz resultante"
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                ESCREVA mat_result[i,j]
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
        soma ← 0
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                SE RESTO(mat_result[i,j]/2) = 0
                    ENTÃO soma ← soma + mat_result[i,j]
            FIM
        ESCREVA "Soma dos elementos pares da linha ", i, " da matriz resultante = ", soma
    FIM
PARA j ← 1 ATÉ 3 FAÇA
    INÍCIO
        qtde ← 0
        PARA i ← 1 ATÉ 4 FAÇA
            INÍCIO

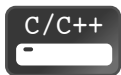
```


```

        SE mat_result[i,j] > 1 E mat_result[i,j] < 5
            ENTÃO qtde ← qtde + 1
        FIM
    ESCRIVA "Total de números entre 1 e 5 na coluna ", j, " da matriz resultante
    = ",qtde
FIM
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:  
 \EXERC\CAP7\PASCAL\EX15.PAS e \EXERC\CAP7\PASCAL\EX15.EXE

**C/C++** SOLUÇÃO:  
 \EXERC\CAP7\C++\EX15.CPP e \EXERC\CAP7\C++\EX15.EXE

**JAVA** SOLUÇÃO:  
 \EXERC\CAP7\JAVA\EX15.java e \EXERC\CAP7\JAVA\EX15.class

- 16.** Faça um programa que preencha uma matriz  $7 \times 7$  de números inteiros e crie dois vetores com sete posições cada um que contenham, respectivamente, o maior elemento de cada uma das linhas e o menor elemento de cada uma das colunas. Escreva a matriz e os dois vetores gerados.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE mat[7,7], vet1[7], vet2[7] NUMÉRICO
    i, j, maior, menor NUMÉRICO
PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 7 FAÇA
            INÍCIO
                LEIA mat[i, j]
            FIM
        FIM
    FIM
PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        maior ← mat[i,1]
        PARA j ← 2 ATÉ 7 FAÇA
            INÍCIO
                SE (mat[i, j] > maior)
                    ENTÃO maior ← mat[i, j]
            FIM
        vet1[i] ← maior
    FIM
PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        menor ← mat[1,i]
        PARA j ← 2 ATÉ 7 FAÇA
            INÍCIO
                SE (mat[j, i] < menor)
                    ENTÃO menor ← mat[j, i]
            FIM
        vet2[i] ← menor
    FIM
PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 7 FAÇA
            INÍCIO
                ESCRIVA mat[i, j]
            FIM
        FIM
FIM

```

```

PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        ESCRIVA vet1[i]
    FIM
PARA i ← 1 ATÉ 7 FAÇA
    INÍCIO
        ESCRIVA vet2[i]
    FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

```
\EXERC\CAP7\PASCAL\EX16.PAS e \EXERC\CAP7\PASCAL\EX16.EXE
```

**C/C++**SOLUÇÃO:

```
\EXERC\CAP7\C++\EX16.CPP e \EXERC\CAP7\C++\EX16.EXE
```

**JAVA**SOLUÇÃO:

```
\EXERC\CAP7\JAVA\EX16.java e \EXERC\CAP7\JAVA\EX16.class
```

**17.** Faça um programa que utilize uma matriz  $5 \times 5$  a qual aceite três tipos de valores: múltiplos de 5, múltiplos de 11 e múltiplos de 13. Devem ser lidos apenas valores maiores que zero. Após a leitura, os números devem ser distribuídos da seguinte maneira:

- os múltiplos de 5 devem ocupar a diagonal principal;
- os múltiplos de 11 devem ficar acima da diagonal principal;
- os múltiplos de 13 devem ficar abaixo da diagonal principal.

Como alguns números podem ser múltiplos de 5, de 11 e também de 13 (por exemplo, 55 é múltiplo de 5 e de 11; 65 é múltiplo de 5 e de 13), deve-se, primeiro, verificar se o número digitado é múltiplo de 5. Caso não seja, deve-se verificar se é múltiplo de 11. Caso não seja, deve-se verificar se é múltiplo de 13. Caso não seja, o programa deverá mostrar a mensagem Número inválido (por exemplo, o número 55 deverá ser considerado múltiplo de 5, pois esta é a comparação que será feita primeiro).

Segue-se um exemplo.

5	44	11	33	55
26	15	77	99	88
39	13	10	121	22
52	78	65	40	132
91	117	104	143	25

Esse programa deverá observar as seguintes situações:

- quando o usuário digitar um múltiplo de 5 e não houver mais espaço na diagonal principal, deverá mostrar a mensagem *Diagonal totalmente preenchida*;
- quando o usuário digitar um múltiplo de 11 e não houver mais espaço disponível na matriz, deverá mostrar a mensagem *Não existe espaço acima da diagonal principal*;
- quando o usuário digitar um múltiplo de 13 e não houver mais espaço disponível na matriz, deverá mostrar a mensagem *Não existe espaço abaixo da diagonal principal*;
- quando a matriz estiver totalmente preenchida, deverá mostrar todos os elementos da matriz, junto com suas posições (linha e coluna).

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE mat[5, 5] NUMÉRICO
    i, j, dp, lin_ac, col_ac, lin_ab NUMÉRICO
    col_ab, num, r, cont_dp, cont_ac, cont_ab NUMÉRICO
dp ← 1
lin_ac ← 1
col_ac ← 2
lin_ab ← 2
col_ab ← 1
cont_dp ← 0
cont_ac ← 0
cont_ab ← 0
ENQUANTO cont_ac < 10 OU cont_ab < 10 OU cont_dp < 5 FAÇA
INÍCIO
    LEIA num
    r ← RESTO (num/5)
    SE r = 0
    ENTÃO INÍCIO
        SE cont_dp < 5
        ENTÃO INÍCIO
            mat[dp,dp] ← num dp ← dp + 1
            cont_dp ← cont_dp + 1
        FIM
        SENÃO ESCREVA "Não existe mais espaço para múltiplos de 5"
    FIM
    SENÃO INÍCIO
        r ← RESTO (num/11)
        SE r = 0
        ENTÃO INÍCIO
            SE cont_ac < 10
            ENTÃO INÍCIO
                mat[lin_ac,col_ac] ← num
                col_ac ← col_ac + 1
                SE col_ac > 5
                ENTÃO INÍCIO
                    lin_ac ← lin_ac + 1
                    col_ac ← lin_ac + 1
                FIM
                cont_ac ← cont_ac + 1
            FIM
            SENÃO ESCREVA "Não existe mais espaço para múltiplos de 11"
        FIM
    SENÃO INÍCIO
        r ← RESTO (num/13)
        SE r = 0
        ENTÃO INÍCIO
            SE cont_ab < 10
            ENTÃO INÍCIO
                mat[lin_ab,col_ab] ← num
                col_ab ← col_ab + 1
                SE col_ab >= lin_ab
                ENTÃO INÍCIO
                    lin_ab ← lin_ab + 1
                    col_ab ← 1
                FIM
            FIM
        FIM
    FIM

```

```

        cont_ab ← cont_ab + 1
    FIM
    SENÃO ESCREVA "Não existe mais espaço para
        múltiplos de 11"

    FIM
SENÃO INÍCIO
    r ← RESTO (num/13)
    SE r = 0
    ENTÃO INÍCIO
        SE cont_ab < 10
        ENTÃO INÍCIO
            mat[lin_ab,col_ab] ← num
            col_ab ← col_ab + 1
            SE col_ab >= lin_ab
            ENTÃO INÍCIO
                lin_ab ← lin_ab + 1
                col_ab ← 1
            FIM
            cont_ab ← cont_ab + 1
        FIM
    SENÃO ESCREVA "Não existe mais espaço para
        múltiplos de 13"

    FIM
    SENÃO ESCREVA "Digite um número múltiplo de 5 ou
        de 11 ou de 13"

    FIM
FIM
FIM
PARA i ← 1 ATÉ 5 FAÇA
INÍCIO
    PARA j ← 1 ATÉ 5 FAÇA
        INÍCIO
            ESCREVA mat[i,j]
        FIM
    FIM
FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX17.PAS e \EXERC\CAP7\PASCAL\EX17.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX17.CPP e \EXERC\CAP7\C++\EX17.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX17.java e \EXERC\CAP7\JAVA\EX17.class

- 18.** Crie um programa que leia um vetor vet contendo 18 elementos. A seguir, o programa deverá distribuir esses elementos em uma matriz  $3 \times 6$  e, no final, mostrar a matriz gerada.

Veja a seguir um exemplo do que seu programa deverá fazer.

vet	3	25	1	58	97	43	65	32	27	19	10	6	88	13	34	57	89	87
-----	---	----	---	----	----	----	----	----	----	----	----	---	----	----	----	----	----	----

mat	3	25	1	58	97	43
	65	32	27	19	10	6
	88	13	34	57	89	87

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE vet [18], mat [3,6], i, j, lin, col NUMÉRICO
PARA i ← 1 ATÉ 18 FAÇA
    INÍCIO
        LEIA vet[i]
    FIM
lin ← 1
col ← 1
PARA i ← 1 ATÉ 18 FAÇA
    INÍCIO
        mat[lin,col] ← vet[i]
        col ← col + 1
        SE col > 6
            ENTÃO INÍCIO
                lin ← lin + 1
                col ← 1
            FIM
    FIM
PARA i ← 1 ATÉ 3 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 6 FAÇA
            INÍCIO
                ESCREVA "Elemento ", i , " - ", j , mat[i,j]
            FIM
        FIM
    FIM
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX18.PAS e \EXERC\CAP7\PASCAL\EX18.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX18.CPP e \EXERC\CAP7\C++\EX18.EXE

**JAVA** SOLUÇÃO:

\EXERC\CAP7\JAVA\EX18.java e \EXERC\CAP7\JAVA\EX18.class

**19.** Faça um programa que utilize uma matriz com dimensões máximas de cinco linhas e quatro colunas. Solicite que sejam digitados os números que serão armazenados na matriz da seguinte maneira:

- se o número digitado for par, deve ser armazenado em uma linha de índice par;
- se o número digitado for ímpar, deve ser armazenado em uma linha de índice ímpar;
- as linhas devem ser preenchidas de cima para baixo (por exemplo, os números pares digitados devem ser armazenados inicialmente na primeira linha par; quando essa linha estiver totalmente preenchida, deve ser utilizada a segunda linha par e assim sucessivamente; o mesmo procedimento deve ser adotado para os números ímpares);
- quando não couberem mais números pares ou ímpares, o programa deverá mostrar uma mensagem ao usuário;
- quando a matriz estiver totalmente preenchida, o programa deverá encerrar a leitura dos números e mostrar todos os elementos armazenados na matriz.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE mat[5,4], i, j, num, r NUMÉRICO
lin_p, col_p, lin_i, col_i, tot NUMÉRICO
lin_p ← 2
col_p ← 1
lin_i ← 1
col_i ← 1
tot ← 0
REPITA
SE tot ≠ 20
    ENTÃO INÍCIO
        LEIA num
        r ← RESTO (num/2)
        SE r = 0
            ENTÃO INÍCIO
                SE lin_p > 4
                    ENTÃO ESCRIVA "Sem espaço para números pares"
                SENÃO INÍCIO
                    mat[lin_p,col_p] ← num
                    col_p ← col_p + 1
                    SE col_p > 4
                        ENTÃO INÍCIO
                            lin_p ← lin_p + 2 col_p ← 1
                        FIM
                FIM
            FIM
        SENÃO INÍCIO
            SE lin_i > 5
                ENTÃO ESCRIVA "Sem espaço para números ímpares"
            SENÃO INÍCIO
                mat[lin_i,col_i] ← num
                col_i ← col_i + 1
                SE col_i > 4
                    ENTÃO INÍCIO
                        lin_i ← lin_i + 2
                        col_i ← 1
                    FIM
            FIM
        FIM
    tot ← tot + 1
FIM
ATÉ tot = 20
ESCREVA "Matriz totalmente preenchida"
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 4 FAÇA
            INÍCIO
                ESCRIVA mat[i,j]
            FIM
        FIM
    FIM
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX19.PAS e \EXERC\CAP7\PASCAL\EX19.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX19.CPP e \EXERC\CAP7\C++\EX19.EXE

**JAVA** SOLUÇÃO:

\EXERC\CAP7\JAVA\EX19.java e \EXERC\CAP7\JAVA\EX19.class



- 20.** Crie um programa que utilize uma matriz com dimensões máximas de cinco linhas e quatro colunas e solicite que sejam digitados os números (desordenadamente), armazenando-os, ordenadamente, na matriz.

Observe o exemplo que se segue.

Supondo que sejam digitados os seguintes números: 10 – 1 – 2 – 20 – 30 – 17 – 98 – 65 – 24 – 12 – 5 – 8 – 73 – 55 – 31 – 100 – 120 – 110 – 114 – 130, estes deverão ser armazenados na matriz da seguinte maneira:

1	2	5	8
10	12	17	20
24	30	31	55
65	73	98	100
110	114	120	130

**ALGORITMO** Solução:

```

ALGORITMO
DECLARE num[5,4] NUMÉRICO
      num_aux, i, j, k, l, m, n, lin, col NUMÉRICO
PARA i ← 1 ATÉ 5 FAÇA
INÍCIO
PARA j ← 1 ATÉ 4 FAÇA
INÍCIO
LEIA num_aux
IF i = 1 E j = 1
    ENTÃO num[i, j] ← num_aux
    SENÃO INÍCIO
        k ← 1
        l ← 1
        ENQUANTO num[k, l] < num_aux E (k ≠ i OU l ≠ j) FAÇA
        INÍCIO
            l ← l + 1
        SE l > 4
        ENTÃO INÍCIO
            k ← k + 1
            l ← 1
        FIM
    FIM
    m ← i;
    n ← j;
    ENQUANTO m ≠ k OU n ≠ 1 FAÇA
    INÍCIO
    SE n-1 < 1 ENTÃO
    INÍCIO
        lin ← m-1
        col ← 4
    FIM
    SENÃO INÍCIO
        lin ← m
        col ← n-1
    FIM
    num[m][n] ← num[lin][col]
    n ← n-1
    SE n < 1
    ENTÃO INÍCIO
        n ← 4
        m ← m-1
    FIM
    FIM
    num[k][l] ← num_aux
FIM
FIM

```

FIM

```

FIM
PARA i ← 1 ATÉ 5 FAÇA
INÍCIO
PARA j ← 1 ATÉ 4 FAÇA
INÍCIO
ESCREVA "Elemento da posição ", i, "-", j, " = ", num[i][j]
FIM
FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

```
\EXERC\CAP7\PASCAL\EX20.PAS e \EXERC\CAP7\PASCAL\EX20.EXE
```

**C/C++**SOLUÇÃO:

```
\EXERC\CAP7\C++\EX20.CPP e \EXERC\CAP7\C++\EX20.EXE
```

**JAVA**SOLUÇÃO:

```
\EXERC\CAP7\JAVA\EX20.java e \EXERC\CAP7\JAVA\EX20.class
```

- 21.** Crie um programa que utilize uma matriz com as dimensões fornecidas pelo usuário e execute as solicitações a seguir.

Para realizar essa tarefa, a matriz deverá ser obrigatoriamente quadrada (número igual de linhas e colunas). Sendo assim, solicite que seja informada a dimensão da matriz.

Posteriormente, o programa deverá realizar a leitura dos elementos que vão compor a matriz. Por fim, deverá somar e mostrar os elementos que estão abaixo da diagonal secundária.

Veja o exemplo.

Imagine que sejam informados números, conforme apresentado nesta matriz.

20	10	1	8
17	42	11	98
19	45	32	87
12	36	65	25

O resultado do problema seria:  $98 + 32 + 87 + 36 + 65 + 25 = 343$

**ALGORITMO** SOLUÇÃO:

ALGORITMO

```
DECLARE num[100,100],dim, l, c, soma, cont NUMÉRICO
```

```
REPITA
```

```
    ESCREVA "Digite a dimensão da matriz (quadrada) - no máximo 100"
```

```
    LEIA dim
```

```
ATÉ dim >= 1 E dim <= 100
```

```
l ← 1
```

```
c ← 1
```

```
ENQUANTO l ≤ dim FAÇA
```

```
INÍCIO
```

```
    ENQUANTO c ≤ dim FAÇA
```

```
    INÍCIO
```

```
        LEIA num[l,c]
```

```
        c ← c + 1
```

```
    FIM
```

```

    l ← l + 1
    c ← 1
FIM
soma ← 0
cont ← 0
l ← 2
c ← dim
ENQUANTO l ≤ dim FAÇA
INÍCIO
    ENQUANTO c ≥ dim-cont FAÇA
    INÍCIO
        ESCRIVA num[l,c]
        soma ← soma + num[l,c]
        c ← c - 1
    FIM
    cont ← cont + 1
    l ← l + 1
    c ← dim
FIM
ESCREVA soma
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

 \EXERC\CAP7\PASCAL\EX21.PAS e \EXERC\CAP7\PASCAL\EX21.EXE

**C/C++**SOLUÇÃO:

 \EXERC\CAP7\C++\EX21.CPP e \EXERC\CAP7\C++\EX21.EXE

**JAVA**SOLUÇÃO:

 \EXERC\CAP7\JAVA\EX21.java e \EXERC\CAP7\JAVA\EX21.class

**22.** Faça um programa que receba o estoque atual de três produtos, armazenados em quatro armazéns, e coloque esses dados em uma matriz  $5 \times 3$ . Considerando que a última linha dessa matriz contenha o custo de cada produto, o programa deverá calcular e mostrar:

- a quantidade de itens quadrados em cada armazém;
- qual armazém possui maior estoque do produto 2;
- qual armazém possui menor estoque;
- qual o custo total de cada produto;
- qual o custo total de cada armazém.

Devem ser desconsiderados empates.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE prod[5,3], tot_arm, maior_e NUMÉRICO
        menor_e, custo_p, custo_a, i, j, ind_a NUMÉRICO
PARA i ← 1 ATÉ 4 FAÇA
INÍCIO
    PARA j ← 1 ATÉ 3 FAÇA
    INÍCIO
        LEIA prod[i,j]
    FIM
FIM
PARA i ← 1 ATÉ 3 FAÇA

```

```

    INÍCIO
    LEIA prod[5,i]
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
    tot_arm ← 0
    PARA j ← 1 ATÉ 3 FAÇA
        INÍCIO
        tot_arm ← tot_arm + prod[i,j]
        FIM
    ESCREVA "O total de itens no armazém ",i, " = ",tot_arm
    SE i=1
        ENTÃO INÍCIO
            menor_e ← tot_arm
            ind_a ← i
            ind_a ← i
            FIM
            SENÃO INÍCIO
                SE tot_arm < menor_e
                    ENTÃO INÍCIO
                        menor_e ← tot_arm
                        ind_a ← i
                        FIM
            FIM
        FIM
    FIM
    ESCREVA "Armazém com menor estoque", ind_a
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
    SE i = 1
        ENTÃO INÍCIO
            maior_e ← prod[i,2]
            ind_a ← i
            FIM
        SENÃO INÍCIO
            SE prod[i,2] > maior_e
                ENTÃO INÍCIO
                    maior_e ← prod[i,2]
                    ind_a ← i
                FIM
            FIM
        FIM
    FIM
    ESCREVA "O maior estoque do produto 2 está no armazém" ,ind_a
PARA j ← 1 ATÉ 3 FAÇA
    INÍCIO
    custo_p ← 0
    PARA i ← 1 ATÉ 4 FAÇA
        INÍCIO
        custo_p ← custo_p + prod[i,j]
        FIM
    custo_p ← custo_p * prod[5,j]
    ESCREVA "O custo total do produto ", j , " = ",custo_p
    FIM
PARA i ← 1 ATÉ 4 FAÇA
    INÍCIO
    custo_a ← 0
    PARA j ← 1 ATÉ 3 FAÇA
        INÍCIO
        custo_a ← custo_a + (prod[i,j] * prod[5,j])
        FIM
    ESCREVA "O custo total do armazém ", i , " = ", custo_a
    FIM
FIM_ALGORITMO.

```

**PASCAL**SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX22.PAS e \EXERC\CAP7\PASCAL\EX22.EXE

**C/C++**SOLUÇÃO:

\EXERC\CAP7\C++\EX22.CPP e \EXERC\CAP7\C++\EX22.EXE

**JAVA**SOLUÇÃO:

\EXERC\CAP7\JAVA\EX22.java e \EXERC\CAP7\JAVA\EX22.class

**23.** Crie um programa que receba as vendas semanais (de um mês) de cinco vendedores de uma loja e armazene essas vendas em uma matriz. O programa deverá calcular e mostrar:

- o total de vendas do mês de cada vendedor;
- o total de vendas de cada semana (todos os vendedores juntos);
- o total de vendas do mês.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE vendas[4,5], tot_ven, tot_sem, tot_geral, i, j NUMÉRICO
PARA i ← 1 ATÉ 4 FAÇA
  INÍCIO
    PARA j ← 1 ATÉ 5 FAÇA
      INÍCIO
        LEIA vendas[i, j]
      FIM
    FIM
  FIM
PARA i ← 1 ATÉ 5 FAÇA
  INÍCIO
    tot_ven ← 0
    PARA j ← 1 ATÉ 4 FAÇA
      INÍCIO
        tot_ven ← tot_ven + vendas[j, i]
      FIM
    ESCRVA tot_ven
  FIM
PARA i ← 1 ATÉ 4 FAÇA
  INÍCIO
    tot_sem ← 0
    PARA j ← 1 ATÉ 5 FAÇA
      INÍCIO
        tot_sem ← tot_sem + vendas[i, j]
      FIM
    ESCRVA tot_sem
  FIM
tot_geral ← 0
PARA i ← 1 ATÉ 4 FAÇA
  INÍCIO
    PARA j ← 1 ATÉ 5 FAÇA
      INÍCIO
        tot_geral ← tot_geral + vendas[i, j]
      FIM
    FIM
  FIM
ESCREVA tot_geral
FIM_ALGORITMO.

```

**PASCAL** SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX23.PAS e \EXERC\CAP7\PASCAL\EX23.EXE

**C/C++** SOLUÇÃO:

\EXERC\CAP7\C++\EX23.CPP e \EXERC\CAP7\C++\EX23.EXE

**JAVA** SOLUÇÃO:

\EXERC\CAP7\JAVA\EX23.java e \EXERC\CAP7\JAVA\EX23.class

**24.** Uma escola deseja controlar as médias das disciplinas que seus alunos cursam. Sabe-se que nessa escola existem três turmas, com oito alunos cada uma, e cada aluno cursa quatro disciplinas. Crie um programa que armazene essas médias em uma matriz  $3 \times 8 \times 4$ . Depois da leitura, ele deverá calcular e mostrar:

- a média geral de cada aluno;
- a média de cada turma.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE medias[3][8][4], i, j, k, media_aluno, media_turma NUMÉRICO
PARA i ← 1 ATÉ 3 FAÇA
  INÍCIO
    PARA j ← 1 ATÉ 8 FAÇA
      INÍCIO
        PARA k ← 1 ATÉ 4 FAÇA
          INÍCIO
            LEIA medias[i][j][k]
          FIM
        FIM
      FIM
    FIM
  FIM
PARA i ← 1 ATÉ 3 FAÇA
  INÍCIO
    PARA j ← 1 ATÉ 8 FAÇA
      INÍCIO
        media_aluno ← 0
        PARA k ← 1 ATÉ 4 FAÇA
          INÍCIO
            media_aluno ← media_aluno + medias[i][j][k]
          FIM
        media_aluno ← media_aluno / 4
        ESCREVA "A média do aluno ", j, " na turma ", i, " = ", media_aluno
      FIM
    FIM
  FIM
PARA i ← 1 ATÉ 3 FAÇA
  INÍCIO
    media_turma ← 0
    PARA j ← 1 ATÉ 8 FAÇA
      INÍCIO
        PARA k ← 1 ATÉ 4 FAÇA
          INÍCIO
            media_turma ← media_turma + medias[i][j][k]
          FIM
        FIM
      FIM
    media_turma ← media_turma / (8 * 4)
    ESCREVA "A média da turma ", i, " = ", media_turma
  FIM
FIM_ALGORITMO.

```



SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX24.PAS e \EXERC\CAP7\PASCAL\EX24.EXE



SOLUÇÃO:

\EXERC\CAP7\C++\EX24.CPP e \EXERC\CAP7\C++\EX24.EXE



SOLUÇÃO:

\EXERC\CAP7\JAVA\EX24.java e \EXERC\CAP7\JAVA\EX24.class

**25.** Elabore um programa que receba as vendas de cinco produtos em três lojas diferentes, e em dois meses consecutivos. O programa deverá armazenar essas vendas em duas matrizes  $5 \times 3$ . O bimestre é uma matriz  $5 \times 3$ , resultado da soma das duas matrizes anteriores. Deverá ainda calcular e mostrar:

- as vendas de cada produto, em cada loja, no bimestre;
- a maior venda do bimestre;
- o total vendido, por loja, no bimestre;
- o total vendido de cada produto no bimestre.

**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE mes1[5,3], mes2[5,3], bim[5,3] NUMÉRICO
        i, j, tot_prod, tot_loja, maior NUMÉRICO
PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        PARA j ← 1 ATÉ 3 FAÇA
            INÍCIO
                LEIA mes1[i,j]
            FIM
        FIM
    PARA i ← 1 ATÉ 5 FAÇA
        INÍCIO
            PARA j ← 1 ATÉ 3 FAÇA
                INÍCIO
                    LEIA mes2[i,j]
                FIM
            FIM
        PARA i ← 1 ATÉ 5 FAÇA
            INÍCIO
                PARA j ← 1 ATÉ 3 FAÇA
                    INÍCIO
                        bim[i,i] ← mes1[i,j] + mes2[i,j]
                        ECREVA bim[i,j]
                        SE i=1 E j=1
                            ENTÃO maior ← bim[i,j]
                        SENÃO SE bim[i,j] > maior
                            ENTÃO maior ← bim[i,j]
                    FIM
                FIM
            FIM
        ECREVA maior
        PARA i ← 1 ATÉ 3 FAÇA
            INÍCIO
                tot_loja ← 0
                PARA j ← 1 ATÉ 5 FAÇA
                    INÍCIO
                        tot_loja ← tot_loja + bim[j,i]
                    FIM
                ECREVA tot_loja
            FIM

```

```

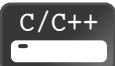
PARA i ← 1 ATÉ 5 FAÇA
INÍCIO
    tot_prod ← 0
    PARA j ← 1 ATÉ 3 FAÇA
    INÍCIO
        tot_prod ← tot_prod + bim[i,j]
    FIM
FIM
ESCREVA tot_prod
FIM
FIM_ALGORITMO.

```



SOLUÇÃO:

\EXERC\CAP7\PASCAL\EX25.PAS e \EXERC\CAP7\PASCAL\EX25.EXE



SOLUÇÃO:

\EXERC\CAP7\C++\EX25.CPP e \EXERC\CAP7\C++\EX25.EXE



SOLUÇÃO:

\EXERC\CAP7\JAVA\EX25.JAVA e \EXERC\CAP7\JAVA\EX25.class

## EXERCÍCIOS PROPOSTOS

1. Faça um programa que preencha uma matriz  $3 \times 5$  com números inteiros, calcule e mostre a quantidade de elementos entre 15 e 20.
2. Crie um programa que preencha uma matriz  $2 \times 4$  com números inteiros, calcule e mostre:
  - a quantidade de elementos entre 12 e 20 em cada linha;
  - a média dos elementos pares da matriz.
3. Elabore um programa que preencha uma matriz  $6 \times 3$ , calcule e mostre:
  - o maior elemento da matriz e sua respectiva posição, ou seja, linha e coluna;
  - o menor elemento da matriz e sua respectiva posição, ou seja, linha e coluna.
4. Faça um programa que receba:
  - as notas de 15 alunos em cinco provas diferentes e armazene-as em uma matriz  $15 \times 5$ ;
  - os nomes dos 15 alunos e armazene-os em um vetor de 15 posições.
 O programa deverá calcular e mostrar:
  - para cada aluno, o nome, a média aritmética das cinco provas e a situação (aprovado, reprovado ou exame);
  - a média da classe.
5. Elabore um programa que preencha uma matriz  $12 \times 4$  com os valores das vendas de uma loja, em que cada linha representa um mês do ano e cada coluna representa uma semana do mês. O programa deverá calcular e mostrar:
  - o total vendido em cada mês do ano, mostrando o nome do mês por extenso;
  - o total vendido em cada semana durante todo o ano;
  - o total vendido pela loja no ano.
6. Faça um programa que preencha uma matriz  $20 \times 10$  com números inteiros, e some cada uma das colunas, armazenando o resultado da soma em um vetor. A seguir, o programa deverá multiplicar cada elemento da matriz pela soma da coluna e mostrar a matriz resultante.
7. Elabore um programa que preencha uma matriz M de ordem  $4 \times 6$  e uma segunda matriz N de ordem  $6 \times 4$ , calcule e imprima a soma das linhas de M com as colunas de N.



- 8.** Crie um programa que preencha duas matrizes  $3 \times 8$  com números inteiros, calcule e mostre:
  - a soma das duas matrizes, resultando em uma terceira matriz também de ordem  $3 \times 8$ ;
  - a diferença das duas matrizes, resultando em uma quarta matriz também de ordem  $3 \times 8$ .
- 9.** Faça um programa que preencha uma matriz  $3 \times 3$  com números reais e outro valor numérico digitado pelo usuário. O programa deverá calcular e mostrar a matriz resultante da multiplicação do número digitado por cada elemento da matriz.
- 10.** Crie um programa que preencha uma matriz  $5 \times 5$  com números inteiros, calcule e mostre a soma:
  - dos elementos da linha 4;
  - dos elementos da coluna 2;
  - dos elementos da diagonal principal;
  - dos elementos da diagonal secundária;
  - de todos os elementos da matriz.
- 11.** Elabore um programa que: receba a idade de oito alunos e armazene-as em um vetor; armazene o código de cinco disciplinas em outro vetor; armazene em uma matriz a quantidade de provas que cada aluno fez em cada disciplina.
- 12.** O programa deverá calcular e mostrar:
  - a quantidade de alunos com idade entre 18 e 25 anos que fizeram mais de duas provas em determinada disciplina cujo código é digitado pelo usuário. O usuário poderá digitar um código não cadastrado; nesse caso, o programa deverá mostrar uma mensagem de erro;
  - uma listagem contendo o código dos alunos que fizeram menos que três provas em determinada disciplina, seguido do código da disciplina;
  - a média de idade dos alunos que não fizeram nenhuma prova em alguma disciplina. Cuidado para não contar duas vezes o mesmo aluno.
- 13.** Elabore um programa que: preencha uma matriz  $6 \times 4$ ; recalcule a matriz digitada, onde cada linha deverá ser multiplicada pelo maior elemento da linha em questão; mostre a matriz resultante.
- 14.** Faça um programa que preencha uma matriz  $2 \times 3$ , calcule e mostre a quantidade de elementos da matriz que não pertencem ao intervalo  $[5,15]$ .
- 15.** Crie um programa que preencha uma matriz  $12 \times 13$  e divida todos os elementos de cada linha pelo maior elemento em módulo daquela linha. O programa deverá escrever a matriz lida e a modificada.
- 16.** Elabore um programa que preencha uma matriz  $5 \times 5$  e crie dois vetores de cinco posições cada um, que contenham, respectivamente, as somas das linhas e das colunas da matriz. O programa deverá escrever a matriz e os vetores criados.
- 17.** Faça um programa que preencha e mostre a média dos elementos da diagonal principal de uma matriz  $10 \times 10$ .
- 18.** Crie um programa que preencha uma matriz  $5 \times 5$  de números reais, calcule e mostre a soma dos elementos da diagonal secundária.
- 19.** Faça um programa que preencha uma matriz  $8 \times 6$  de inteiros, calcule e mostre a média dos elementos das linhas pares da matriz.
- 20.** Elabore um programa que preencha uma matriz  $5 \times 5$  com números reais e encontre o maior valor da matriz. A seguir, o programa deverá multiplicar cada elemento da diagonal principal pelo maior valor encontrado e mostrar a matriz resultante após as multiplicações.
- 21.** Faça um programa que preencha uma matriz  $5 \times 5$  de números reais. A seguir, o programa deverá multiplicar cada linha pelo elemento da diagonal principal daquela linha e mostrar a matriz após as multiplicações.
- 22.** Crie um programa que preencha uma matriz  $6 \times 10$ , some as colunas individualmente e acumule as somas na 7ª linha da matriz. O programa deverá mostrar o resultado de cada coluna.

**23.** Faça um programa que preencha uma matriz  $3 \times 4$ , calcule e mostre:

- a quantidade de elementos pares;
- a soma dos elementos ímpares;
- a média de todos os elementos.

**24.** Elabore um programa que preencha uma matriz  $4 \times 5$ , calcule e mostre um vetor com cinco posições, onde cada posição contém a soma dos elementos de cada coluna da matriz. O programa deverá mostrar apenas os elementos do vetor maiores que dez. Se não existir nenhum elemento maior que dez, deverá mostrar uma mensagem.

**25.** Crie um programa que:

- receba o preço de dez produtos e armazene-os em um vetor;
- receba a quantidade estocada de cada um desses produtos, em cinco armazéns diferentes, utilizando uma matriz  $5 \times 10$ .

O programa deverá calcular e mostrar:

- a quantidade de produtos estocados em cada um dos armazéns;
- a quantidade de cada um dos produtos estocados, em todos os armazéns juntos;
- o preço do produto que possui maior estoque em um único armazém;
- o menor estoque armazenado;
- o custo de cada armazém.