CAPÍTULO

7 Matriz

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.

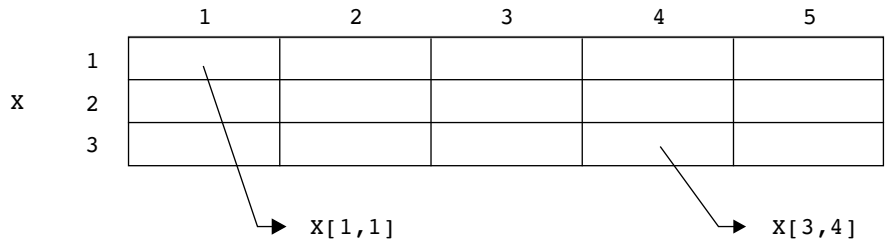
```
DECLARE nome[dimensão1, dimensão2, dimensão3, ..., dimensãoN] tipo
```

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

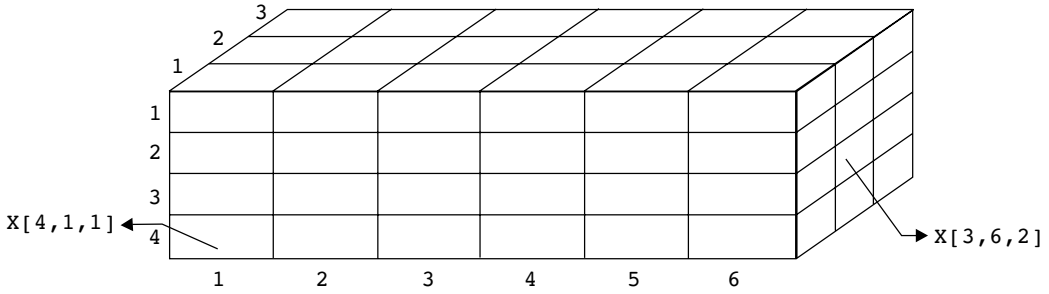
O exemplo a seguir define uma matriz bidimensional, onde o tamanho da 1ª dimensão (linha) é 3 e o da 2ª dimensão (coluna) é 5.

```
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.

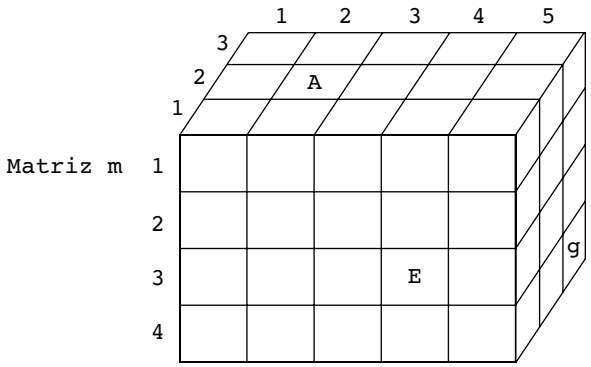
Matriz mat

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 primera 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
	1	12	9	3	7	-23
X	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.

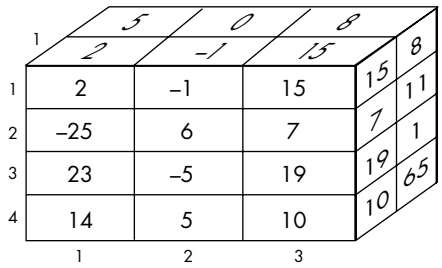
```
PARA i ← 1 ATÉ 4 FAÇA
INÍCIO
   PARA j ← 1 ATÉ 3 FAÇA
   INÍCIO
       PARA k ← 1 ATÉ 2 FAÇA
          INÍCIO
              ESCREVA "Digite o número da linha ",i, " coluna ", j, " e profundidade
                      ➥ ", k, ":"
              LEIA X[i,j,k]
          FIM
   FIM
FIM
```

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



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
      ESCREVA X[i,j]
   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.    ESCREVA "Elementos da linha ", i
4.    PARA j ← 1 ATÉ 4 FAÇA
5.    INÍCIO
6.       ESCREVA x[i,j]
7.    FIM
8. FIM
```

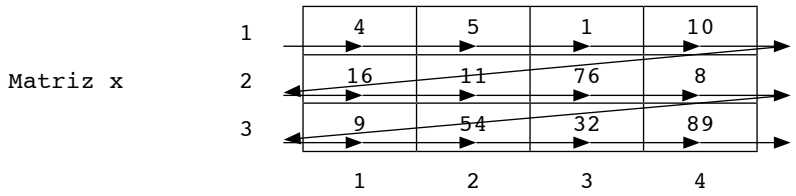
A primeira estrutura de repetição (linha 1) é controlada pela variável `i`, que poderá assumir valores dentro do intervalo de 1 a 3. Cada vez que essa estrutura PARA for executada, encontrará a segunda estrutura de repetição (linha 4), controlada pela variável `j`, que assumirá os valores dentro do intervalo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

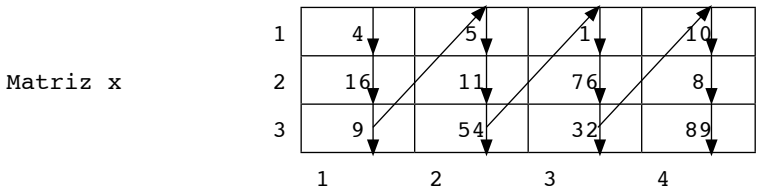
```
1. PARA i ←1 ATÉ 4 FAÇA
2. INÍCIO
3.    ESCREVA "Elementos da coluna ", i
4.    PARA j ← 1 ATÉ 3 FAÇA
5.    INÍCIO
6.       ESCREVA x[j, i]
7.    FIM
8. FIM
```

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.



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

- a mudança dos valores das varáveis `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.

7.3 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[2] 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).

2 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

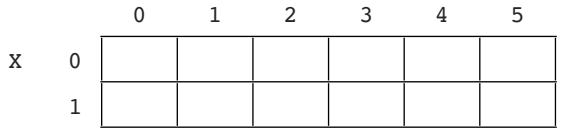
Da mesma maneira como ocorre com os vetores, os índices das dimensões das matrizes começam sempre em 0 (zero).

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

Exemplo 1:

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

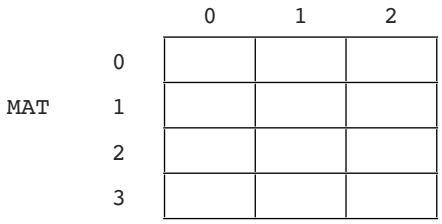
Na declaração do exemplo 1, criou-se uma variável chamada `X` contendo duas linhas (de 0 a 1) com seis colunas cada (de 0 a 5), capaz de armazenar números reais, como pode ser observado a seguir.



Exemplo 2:

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

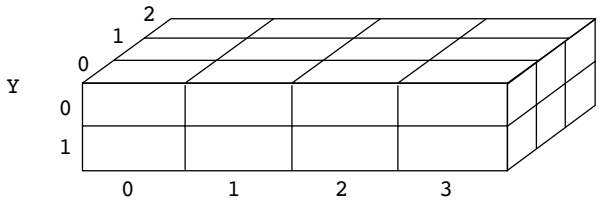
A declaração do exemplo 2 criou uma variável chamada `MAT` contendo quatro linhas (de 0 a 3) com três colunas cada (de 0 a 2), capaz de armazenar caracteres, como pode ser observado a seguir.



Exemplo 3:

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

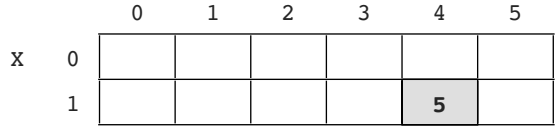
A declaração do exemplo 3 criou uma variável chamada `Y` contendo duas linhas (de 0 a 1) com quatro colunas cada (de 0 a 3) e profundidade três (de 0 a 2), capaz de armazenar números reais, como pode ser observado a seguir.



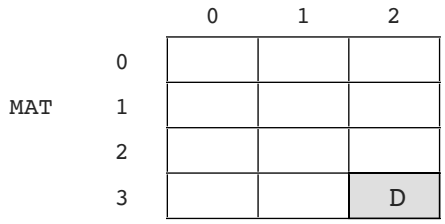
7.3.4 Atribuindo valores a uma matriz

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

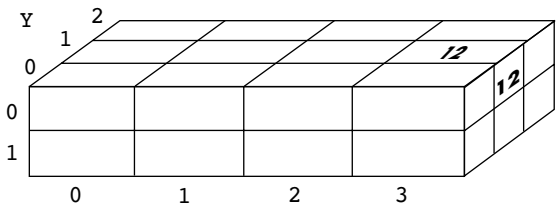
`X[1][4] = 5;` → Atribui o valor 5 à posição identificada pelos índices 1 (2ª linha) e 4 (5ª coluna).



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




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



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`.

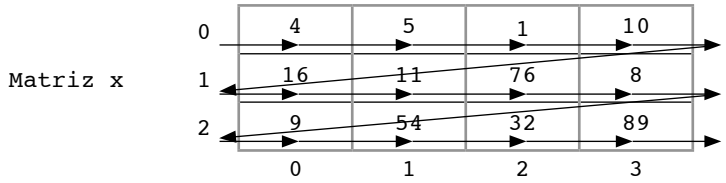
Esse arranjo resolve o problema de mostrar os elementos, separando-os por linhas, já que a variável `i` ficará com valor fixo enquanto a variável `j` assumirá valores de 0 a 3, ou seja, formará todos os pares possíveis de índices. Se `i` estiver valendo 0, serão mostrados todos os elementos da primeira 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. }
```

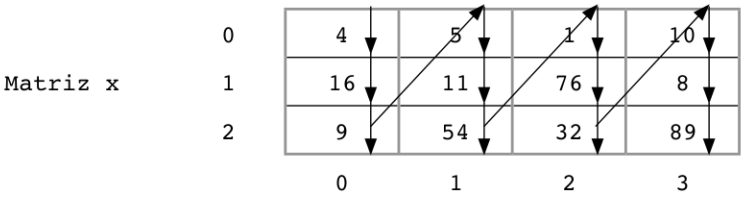
A primeira estrutura de repetição (linha 1) é controlada pela variável `i`, que poderá assumir valores dentro do intervalo de 0 a 3. Cada execução dessa estrutura `for` encontrará a segunda estrutura de repetição (linha 4), controlada pela variável `j`, que assumirá os valores dentro do intervalo de 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.



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

- a mudança dos valores das varáveis `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 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.