Capítulo 6
Matrizes

" Vamos mudar a nossa atitude tradicional de construir programas. Em vez de imaginar que a nossa tarefa principal é instruir um computador a fazer o que queremos, deveremos começar a pensar que a nossa tarefa é explicar a seres humanos o que nós queremos que o computador faça "

- Donald E. Knuth -

Sumário:

6.1- Conceitos

Vimos que um vector era um conjunto finito de elementos do mesmo tipo. Mas se cada elemento desse vector fosse um outro vector, teríamos necessariamente um vector de vectores. Uma matriz é um vector de vectores, ou seja, um **vector bidimensional.**

Como uma matriz é representada por um conjunto bidimensional, então devemos especificar as duas dimensões da sua declaração, ou seja, o número de linhas e o número de colunas. Por exemplo, uma matriz com elementos do tipo inteiro com 4 linhas e 6 colunas é declarada como:

```
int a[M][N];
```

onde

```
#define M  4
#define N  6
```

Para aceder a qualquer elemento de uma matriz, utilizamos uma variável com indexação dupla, denotada por a[i][j], para $0 \leq i < M$ e $0 \leq j < N$.

Mas, para acedermos a todos os elementos de uma matriz, devemos utilizar laços aninhados, ou seja, um laço dentro de outro laço. Se quisermos percorrer por linhas, o laço externo percorre a matriz por linhas e, para cada linha, o laço interno percorre a matriz por colunas. Um segmento de código para representar esse percurso é descrito a seguir:

```
int i, j;
for ( i = 0 ; i < M ; i++)          /* percorre por linhas  */
  for ( j = 0 ; j < N ; j++)        /* percorre por colunas */
```

6.2- Passagem de Parâmetros

Como a memória do computador é linear, as matrizes são armazenadas na memória do computador em linhas. Os primeiros espaços de memória, estão associados a primeira linha, os espaços seguintes a segunda linha e por aí em diante. Por esse facto, quando declaramos uma matriz como parâmetro de uma função, apenas devemos especificar o número máximo de colunas.

6.3- Operações Iterativas com Matrizes

6.3.1- Imprimir os Elementos de uma Matriz

A **operação para imprimir todos os elementos de uma matriz**, é muito simples. Ela consiste em percorrer a matriz por linhas e para cada elemento percorrido, mostrar na tela o seu conteúdo.

```
void  imprimirMatriz ( int a[ ][M] )
  {
     int i, j ;
     for ( i = 0 ; i < M ; i++ )
         for ( j = 0 ; j < N ; j++ )
              printf ( "%d \n", a[i][j]);
   }
```

6.3.2- Somar os Elemento de uma Matriz

A operação para somar o conteúdo de todos os elementos de uma matriz, também é muito simples. Ela consiste em percorrer a matriz por linhas, e por cada elemento percorrido, acumular numa variável auxiliar o seu conteúdo. Essa variável auxiliar deverá ser inicializada com zeros antes do iniciarmos esse percurso.

```
int  somaElementosMatriz( int a[ ][M] )
 {
    int i, j ; soma = 0;
    for ( i = 0 ; i < M ; i++ )
        for ( j = 0 ; j < N ; j++ )
            soma += a[i][j];
 }
```

6.4- Operações Recursivas com Vectores

De forma análoga aos vectores, as matrizes são estruturas de dados iminentemente recursivas, pois vejamos: seja a uma matriz retangular com n linhas e m colunas. Essa matriz é igual a união do elemento qualquer a(i,j) com todos os elementos de A menos esse elemento.

Em função dessa definição recursiva, vamos discutir a seguir, a implementação de algumas operações com a estratégia de decrementar para conquistar.

6.4.1- Imprimir os Elementos de uma Matriz

A **operação para imprimir todos os elementos de uma matriz**, consiste em fragmentar a matriz passada como parâmetro em duas submatrizes. Uma unitária, cujo conteúdo será mostrado na tela, e outra de menor dimensão, que sofrerá novamente um processo de fragmentação, até tornar-se unitária. A solução do problema de imprimir todos os elementos de uma matriz, consiste em imprimir o conteúdo das matrizes unitárias geradas pelo processo de fragmentação.

```
void  imprimirMatrizRec ( int a[ ][M], int i, int j )
 {
    if ( i < M )
     {
        if ( j < N )
         {
            printf (“ %d \n “, a[i][j] );
            imprimirMatrizRec ( a , i , j + 1);
         }
        else
           imprimirMatrizRec ( a , i + 1 , 0);
     }
  }
```

6.4.2- Somar os Elementos de uma Matriz

A **operação para somar todos os elementos de uma matriz**, consiste em fragmentar a matriz passada como parâmetro em duas submatrizes. Uma unitária, cujo conteúdo será acedido e outra de menor dimensão que sofrerá novamente um processo de fragmentação, até tornar-se unitária. A solução do problema original consiste na soma das soluções das submatrizes unitárias geradas pelo processo de fragmentação.

```
int  somaMatrizRec ( int a[ ][M], int i, int j )
  {
     if ( i < M )
        if ( j < N )
           return a[i] + somaMatrizRec( a , i , j + 1);
        else
           return somaMatrizRec ( a , i+1, 0);
  }
```

Para terminar é oportuno salientar que raramente se utiliza algoritmos recursivos para processarem matrizes. O elevadíssimo número de chamadas recursivas, é responsável pela pouca eficiência dos mesmos. Por esse facto, existem muito poucos livros essa estratégia para as matrizes.

6.5- Recomendações Bibliográficas

Para o leitor aprofundar os seus conhecimentos sobre matrizes, recomendamos os seguintes livros:

Deitel H. M., Deitel, P. J.;- *C How to Program,* 8 th Edition, Pearson Education,

K. N. King. - *C Programming – A Modern Approach*, W. W. Norton & Company, Inc., 2nd edition, 2008.

6.6- Exercícios Propostos

6.6.1-Desenvolva uma função iterativa e uma função recursiva que recebe uma matriz rectangular com n linhas e m colunas, cujos elementos são do tipo inteiro. Verifique se essa matriz é simétrica.

6.6.2-Desenvolva uma função iterativa e uma função recursiva que recebe duas matrizes rectangulares com n linhas e m colunas, cujos elementos são do tipo inteiro. Verifica se essas matrizes são iguais.

6.6.3- Desenvolva uma função iterativa que recebe uma matriz quadrada, com n linhas e n colunas, cujos elementos são do tipo inteiro. Verifique se essa matriz é um quadrado latino de tamanho n. Por definição, num quadrado latino de tamanho n, em cada linha e em cada coluna aparecem todos os inteiros 1,2,3, …, n, ou seja, cada linha ou coluna é uma permutação dos n primeiros inteiros.

Por exemplo: n = 3.

$$\begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{vmatrix}$$

6.6.4- Desenvolva uma função iterativa e uma função recursiva que recebe uma matriz retangular com n linhas e m colunas, cujos elementos são do tipo inteiro. Devolva o valor mínimo e o valor mínimo dessa matriz.

6.6.5-Desenvolva uma função iterativa e uma função recursiva que recebe uma matriz quadrada com n linhas e n colunas, cujos elementos são do tipo inteiro, e um número inteiro qualquer. Substituir todas as ocorrências desse número por zeros.

6.6.6-Desenvolva uma função iterativa que recebe uma matriz quadrada com n linha e n colunas, cujos elementos são do tipo inteiro. Devolva essa matriz com os elementos ordenados em ordem crescente por colunas. Por exemplo:

3	1	9
7	6	2
5	8	4

← Antes

Depois →

1	4	7
2	5	8
3	6	9

6.6.7-Desenvolva uma função iterativa e uma função recursiva que recebe uma matriz quadrada com n linhas e n colunas, cujos elementos são do tipo real, e verifique se é uma matriz é identidade. Por definição, uma matriz identidade é aqueles cujos elementos da diagonal principal são iguais a um e os restantes iguais a zero.

6.6.8-Suponha que uma matriz binária quadrada M represente a ligação entre um conjunto de n cidades, de tal forma que M[i , j] = 1 representa que existe uma estrada da cidade i para a cidade j, e M[i , j] = 0 no caso contrário. Por exemplo, na matriz a seguir, temos que a cidade 0 possui estradas para 1 e 2, já a cidade 1 possui apenas uma estrada para a cidade 2. Observe que existe uma estrada da cidade 0 para a cidade 1, mas não existe uma estrada da cidade 1 para a cidade 0, só existem estradas das cidades 2 e 3 para a cidade 0.

0 1 1 0
0 0 1 0
1 1 0 1
1 0 1 0

Desenvolva uma função recursiva que dada uma matriz M e uma cidade i, determine todas as cidades que podem ser alcançadas a partir dessa cidade.