# Aula 17 – Variáveis indexadas de vários índices (matrizes)

Variáveis indexadas podem ter mais de um índice e são chamadas genericamente de matrizes.

**Declaração:**

```
int a[10][10];    /*declara 100 variáveis a[0][0], . . ., a[9][9]*/
float b[30][40];  /*declara 1200 variáveis b[0][0], . . ., b[29][39]*/
double c[3][4][5];/*declara 60 variáveis c[0][0][0], . . . ,c[2][3][4]*/
```

Como no caso dos vetores, sempre se declara o número máximo de elementos, mesmo que não sejam todos utilizados.

Também como no caso dos vetores, os elementos de uma matriz estão localizados contiguamente na memória, variando-se sempre os últimos índices em primeiro lugar. Quando a matriz é de 2 índices, os elementos estão dispostos linha a linha na memória.

Assim, para a matriz a[10][10] acima:

```
a[0][0] a[0][1] a[0][2] . . . a[0][9]
a[1][0] a[1][1] a[1][2] . . . a[1][9]
:
:
:
a[9][0] a[9][1] a[9][2] . . . a[9][9]
```

Para a matriz b[30][40]:

```
b[0][0] b[0][1] a[0][2] . . . b[0][39]
b[1][0] b[1][1] b[1][2] . . . b[1][39]
:
:
:
b[29][0] b[29][1] b[29][2] . . . b[29][39]
```

Para a matriz c[3][4][5]:

```
c[0][0][0] c[0][0][1] c[0][0][2] c[0][0][3] c[0][0][4]
c[0][1][0] c[0][1][1] c[0][1][2] c[0][1][3] c[0][1][4]
c[0][2][0] c[0][2][1] c[0][2][2] c[0][2][3] c[0][2][4]
c[0][3][0] c[0][3][1] c[0][3][2] c[0][3][3] c[0][3][4]
c[1][0][0] c[1][0][1] c[1][0][2] c[1][0][3] c[1][0][4]
c[1][1][0] c[1][1][1] c[1][1][2] c[1][1][3] c[1][1][4]
c[1][2][0] c[1][2][1] c[1][2][2] c[1][2][3] c[1][2][4]
c[1][3][0] c[1][3][1] c[1][3][2] c[1][3][3] c[1][3][4]
c[2][0][0] c[2][0][1] c[2][0][2] c[2][0][3] c[2][0][4]
c[2][1][0] c[2][1][1] c[2][1][2] c[2][1][3] c[2][1][4]
c[2][2][0] c[2][2][1] c[2][2][2] c[2][2][3] c[2][2][4]
c[2][3][0] c[2][3][1] c[2][3][2] c[2][3][3] c[2][3][4]
```

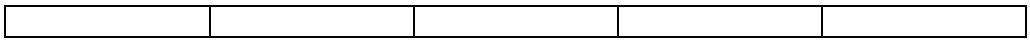
**Uso – Como se fossem variáveis simples**

```
/* atribuição */
a[2][3] = 0;
b[5][20] = b[2][2];
c[i][j][k] = 0;
```

```
/* zerar a matriz a */
for (i = 0; i < 10; i++)
   for (j = 0; j < 10; j++)
       a[i][j] = 0;

/* zerar a matriz b */
for (i = 0; i < 30; i++)
   for (j = 0; i < 40; j++)
        b[i][j] = 0.0;

/* zerar a matriz c */
for (i = 0; i < 3; i++)
   for (j = 0; i < 4; j++)
      for (k = 0; k < 50; k++)
         c[i][j][k] = 0.0;

/* ler e imprimir os 100 elementos de a */
for (i = 0; i < 10; i++)
   for (j = 0; j < 10; i++)
      {scanf("%d", &a[i][j]);
       printf ("%d", a[i][j]);
      }

/* contar quantos nulos tem em c */
cont = 0;
for (k = 0; j < 3; k++)
   for (j = 0; j < 4; j++)
      for (i = 0; i < 5; i++)
        if (c[k][j][i] == 0.0) cont++;

/* Supondo n < 10, calcular o maior, o menor é a média dos elementos da
   matriz a de n linhas por n colunas. Neste caso embora a tenha sido
   declarado com 10 linhas por 10 colunas, só estamos considerando n
   linhas e n colunas */
soma = 0;
max = min =  a[0][0];
for (i = 0; i < n; i++)
  for (i = 0; i < n; i++)
    {if (a[i][j] > max) max = a[i][j];
     if (a[i][j] < min) min = a[i][j];
     soma = soma + a[i][j];
    }
printf ("máximo = %10d – mínimo = %10d – média = %10.5lf",
         max, min, (double) soma / (double) n);

/* Supondo n < 30 e m < 40, calcular o máximo de cada uma das colunas
   da matriz b de n linhas por m colunas */
for (col = 0; col < m; col++)
   {max = b[0][col];
    for (lin = 1; lin < n; lin++)
       if (b[lin][col] > max) max = b[lin][col];
    printf("\no máximo da coluna %2d é = %15.5f", col, max);
   }
```

**Matrizes e Funções**

Matrizes podem ser usadas como parâmetros de funções.
Conforme vimos acima, os elementos das matrizes estão dispostos contiguamente na memória, linha a linha, ou seja, variando-se em primeiro lugar os últimos índices.
A memória é linear, assim, do ponto de vista de armazenamento na memória, não há diferença entre uma matriz e um vetor, pois os elementos estão contíguos na memória. No caso do vetor, o índice, determina o deslocamento do elemento a partir do início do vetor. Por exemplo, considere o vetor `int x[5]` que estará disposto na memória assim:




Observe que `x[i]` estará na posição (início de `x`) `+i`.

No caso de matrizes, o deslocamento em relação ao início, é uma função dos índices. Esta função, nada mais é que a transformação dos vários índices num só índice, ou seja, a linearização dos vários índices. Vejamos alguns exemplos nas matrizes declaradas acima.

1) a[1][2] é o 12° elemento (o primeiro é o 0°), ou (10.1+2)º. Ou seja, o elemento 2 da linha 1.
2) a[2][3] é o 23° elemento, ou (2.10+3)°
3) a[9][5] é o 95° elemento, ou (9.10+5)°
4) b[29][2] é o 1162° = (29.40+2)º
5) c[2][1][3] é o 48° = (2.4.5 + 1.5 + 3)º
6) c[1][3][1] é o 36° = (1.4.5+3.5+1)°

Portanto, para 2 índices, se temos a matriz `mat` com o primeiro e segundo índices de valor máximo `d1` e `d2` (`int mat[d1][d2]`) o elemento `mat[i][j]` estará na posição (início de `mat`) `+ i.d2 + j`.

Para 3 índices, a matriz mat com o primeiro, segundo e terceiro índices de valores máximos `d1`, `d2` e `d3` (`int mat[d1][d2][d3]`) o elemento `mat[i][j][k]` estará na posição (início de `mat`) `+ i.d2.d3 + j.d3 + k`.

Para `n` índices, a matriz `mat` com índices de valores máximos `d1`, `d2`, ..., `dn` (`int mat[d1][d2]...[dn]`) o elemento `mat[i1][i2]...[in]` estará na posição: (início de `mat`) +

```
i1.d2.d3. ... .dn +
12.d3.d4. ... .dn +
i3.d4.d5. ... .dn +
...
i(n-1).dn +
in
```

Então, para acessar um elemento, é necessário conhecer-se as dimensões de todos os índices, com exceção do primeiro. Por isso, na declaração de uma matriz como parâmetro formal, é necessário declarar-se todos os índices máximos a partir do segundo.

Exemplos:

```
int funcx(double a[][100], int n);
double funcy (int x[][200][10], y[], z[][23][30]);
```

**Outra maneira de entender a necessidade de declarar os índices de uma matriz quando esta matriz é parâmetro de função**

Outra forma de entender melhor a necessidade de declarar na função os índices a partir do segundo através de um exemplo com duas dimensões:
Suponha uma função F que recebe com parâmetro uma matriz de no máximo 10 linhas por 20 colunas e também 2 outros parâmetros inteiros n e m que representam as linhas e colunas que serão realmente usadas.

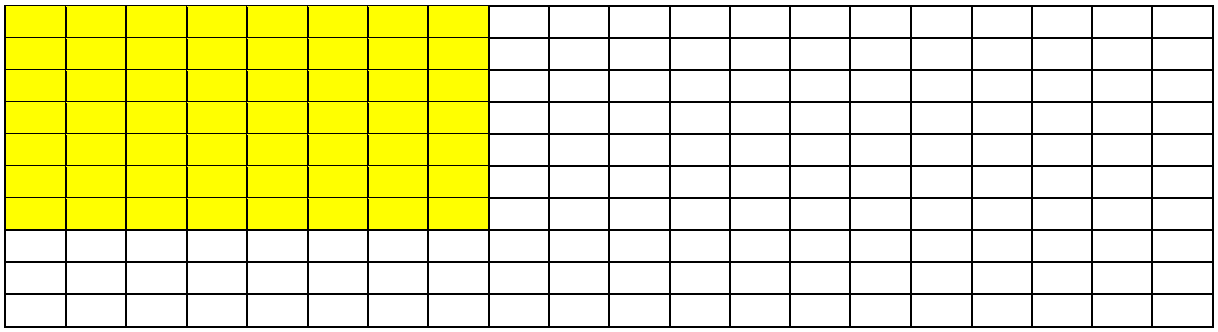
```
void F(int A[][20], int n, int m) {
…
}
```

Os elementos da matriz estão dispostos na memória linha a linha.

Suponha a chamada abaixo da função F:

```
int x[10][20];
...
...
...
F(X,8,7); /* usando apenas 7 linhas e 8 colunas de X */
...
```

Será usado apenas o trecho indicado de **X**.
**F** precisa saber quantos elementos tem que pular por linha da matriz para acessar os elementos corretos.



P83) Escreva uma função `int simetrica(double a[][maxcol],int n)` que verifica se a matriz a `nxn` é uma matriz simétrica, isto é, se `a[i][j]` é igual a `a[j][i]`, devolvendo 1 (sim) ou 0 (não).

```
#define maxlin 10
#define maxcol 10

int simetrica (double a[][maxcol], int n) {
   int i, j;

   for (i = 0; i < n; i++)
       for (j = 0; j < n; j++)
          if (a[i][j] != a[j][i]) return 0; /* não é */

   return 1; /* é simétrica */
}
```

Veja abaixo exemplos de chamada e o que será impresso:

```
/* Exemplo de chamada */
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
 double a[maxlin][maxcol],
        b[maxlin][maxcol];

 int n = 2,
     m = 3;

 /* preenche a (simetrica) */
 a[0][0] = 1; a[1][1] = 2;
 a[0][1] = a[1][0] = 3;

 if (simetrica(a,n)) printf ("\n a e simetrica");
 else printf ("\n a nao e simetrica");

 /* preenche b (não simetrica) */
 b[0][0] = 1; b[1][1] = 2; b[2][2] = 3;
 b[0][1] = b[1][0] = 3;
 b[0][2] = b[2][0] = 4;
 b[1][2] = 5; b[2][1] = 6;

 if (simetrica(b, m)) printf ("\n b e simetrica");
 else printf ("\n b nao e simetrica");
 system("pause");return 0;
}
```

```
 a e simetrica
 b nao e simetrica
```

Observe que na função `simetrica`, fazemos comparações desnecessárias, pois comparamos `a[i][j]` com `a[j][i]` e depois `a[j][i]` com `a[i][j]`, pois `i` e `j` variam de `0` a `n-1`. Bastaria comparar apenas quando `i>j` ou quando `i<j`, ou seja, percorrendo o triângulo inferior ou o triângulo superior.

P84) Idem, percorrendo o triângulo inferior

```
#define maxlin 10
#define maxcol 10

int simetrica (double a[][maxcol], int n) {
   int i, j;

   for (i = 0; i < n; i++)
       for (j = 0; j < i; j++)
          if (a[i][j] != a[j][i]) return 0; / não é */

   return 1; /* é simétrica */
}
```

P85) Idem, percorrendo o triângulo superior.

P86) Escreva uma função `void somamat(double a[][mc],b[][mc],c[][mc],int n)` que recebe as matrizes `a` e `b` e devolve a matriz `c`, soma de `a` com `b`. `a` , `b` e `c` tem dimensão `n x n`.

P87) Idem, uma função `void multmatvet(double a[][mc],double b[],double c[],int n,int m)` que recebe a matriz a de `n x m` elementos, o vetor `b` de `m` e devolve o vetor `c` de `n` elementos que é a multiplicação de `a` por `b`.

```
#define maxlin 100
#define maxcol 100

int multmatvet (double a[][maxcol], double b[], double c[],
                int n, int m) {
   int i, j;

   for (i = 0; i < n; i++) {
         /* multiplica a linha i da matriz pelo vetor */
         c[i] = 0;
         for (j = 0; j < m; j++)
           c[i] = c[i] + a[i][j] * b[j];
        }

   return 0; /* não precisa retornar nada */
}
```

P88) Idem, uma função `multmatmat (double a[][mc], double b[][mc], double c[][mc], int n, int m, int p)` que recebe a matriz a de `n x m` elementos, a matriz `b` de `m x p` elementos e devolve a matriz `c` de `n x p` elementos, que é a multiplicação de `a` por `b`.

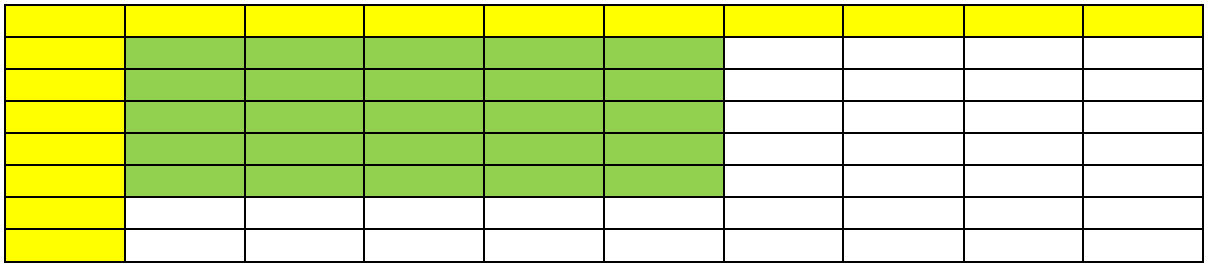
**Uso de matrizes para resolver problemas de álgebra linear**

É usual quando se resolve um problema de álgebra com matrizes como os problemas acima (P83 a P88), as matrizes e vetores terem os índices começando de 1 e não de zero.
Como em C, os índices sempre começam de zero temos que fazer a correspondência.
Uma maneira simples de resolver isso é não usar o elemento de índice zero de um vetor e não usar os elementos da linha zero e coluna zero de uma matriz.

Isso faz com que gastemos mais memória que o necessário, mas às vezes é melhor gastar mais memória e deixar o programa mais claro.
Se usar este artifício, não se esquecer de declarar o vetor com 1 elemento a mais. Se for uma matriz, com uma linha e uma coluna a mais.

Exemplo:
A matriz abaixo foi declarada **`int Mat[8][10]`**.
Vamos usar apenas o trecho:

```
M[1][1]...M[1][5]
...
M[5][1]...M[5][5]
```



Exemplo – função Zera(A,N) que zera a matriz A de N linhas por N colunas. Supor N no máximo 10.

```
#define Max 11
void Zera(int A[][Max],int N) {
```

```
    int i,j;
    for(i=1;i<=N;i++)
       for(j=1;j<=N;j++) A[i][j]=0;
}

int main() {
    int X[Max][Max], Y[Max][Max],M,K;
    ...
    Zera(X,M);
    ...
    Zera(Y,K);
    ...
}
```

**Mais exemplos usando matrizes e vetores começando com índice 1 e não 0**

Refaça todos os exercícios P83 a P88 usando vetores a partir do índice [1] e matrizes a partir do índice [1][1].

**Outros exemplos de funções usando matrizes**

P89) Escreva uma função `ind_min_col (double a[][maxc], int n, int col)` que receba a matriz a de `n x n` elementos e devolva como resultado o índice do menor elemento da coluna `col` desta matriz.

```
#define maxlin 100
#define maxcol 100

int ind_min_col (double a[][maxcol], int n, int col) {

   int lin; /* índice para as linhas */
   int indmin = 0; /* índice do mínimo na coluna. */
                   /* começa supondo que é o primeiro da coluna */

   /* Varre todos os elementos desta linha a partir do segundo,
      verificando se é menor que o mínimo corrente.
      Se for troca o índice do mínimo corrente */
   for (lin = 1; lin < n; lin++)
        if (a[lin][col] < a[indmin][col]) indmin = lin;
   return indmin;
}
```

P90) Faça um programa que dada uma matriz de preços de `n x m` elementos, que são o preço de `m` produtos em `n` lojas, e também um vetor de `m` elementos que são as quantidades de cada um dos `m` produtos que serão compradas e determine:

- O valor total da compra em cada loja (use a função `multmatvet`)
- Quais a loja que têm o valor mínimo para a compra e qual esse valor.
- O preço mínimo de cada produto e qual das `n` lojas tem esse preço mínimo (use a função `ind_min_col`)
- O valor da compra, se usado o preço mínimo de cada produto (use a função `ind_min_col`)

```
#define maxlin 100
#define maxcol 100

#include <stdio.h>
```

```
int main()
{double p[maxlin][maxcol], /* preços */
       q[maxcol],          /* quantidades */
       totais[maxlin],     /* totais da compra em cada loja */
       pmin[maxcol];       /* preços mínimos de cada produto */

 int np, nl,       /* número de produtos e de lojas */
     imin, i, j;   /* índices */

 double totalmin; /* preço total da compra usando os preços mínimos */

 /* ler nl e np sem consistência */
 printf("\n entre com nl e np separados por branco:");
 scanf("%d%d", &nl, &np);

 /* ler a matriz p */
 printf("\n\n ********** matriz de precos");
 for (i = 0; i < nl; i++)
      /* ler a linha i */
      for (j = 0; j < np; j++)
        {printf("\n entre com o elemento p[%2d][%2d] = ", i, j);
         scanf("%lf", &p[i][j]);
        }

 /* ler o vetor q */
 printf("\n\n ********** vetor de quantidades");
 for (i = 0; i < np; i++)
       {printf("\n entre com o elemento q[%2d] = ", i);
        scanf("%lf", &q[i]);
       }

 /* calcula o valor total da compra em cada loja usando multmatvet */
 multmatvet (p, q, totais, nl, np);
 printf("\n\n ********** valor total da compra em cada loja");
 for (i = 0; i < nl; i++)
      printf("\n valor da compra na loja %2d = %10.2lf", i, totais[i]);

 /* Determina qual a loja que valor é o valor mínimo da compra */
 /* Pode haver mais de uma loja. Vamos determinar apenas a primeira */
 printf("\n\n ********** loja de valor minimo");
 imin = 0;
 for (i = 1; i < nl; i++)
       if (totais[i] < totais[imin]) imin = i;
       printf("\n na loja %2d o valor da compra minimo = %15.2lf",
               imin, totais[imin]);

 /* determina o preço mínimo de cada produto e qual das nl lojas
    tem esse preço mínimo usando ind_min_col */
 /* Pode haver mais de uma loja cujo preço de um produto é mínimo.
    Vamos determinar apenas a primeira delas */
 printf("\n\n ********** preco minimo de cada produto");
 for (i = 0; i < np; i++)
    {imin = ind_min_col (p, nl, i);
     printf("\n produto %d na loja %2d tem valor minimo = %15.2lf",
         i, imin, p[imin][i] );
     pmin[i] = p[imin][i];
    }
```

```
 /*determina o valor da compra, se usado o preço mínimo de cada produto*/
 printf("\n\n ********** preco minimo possivel da compra");
 totalmin = 0;
 for (i = 0; i < np; i++)
     totalmin = totalmin + q[i] * pmin[i];
 printf("\n preco minimo possivel da compra = %15.2lf", totalmin);
}
```

Abaixo um exemplo do que será impresso.

```
entre com nl e np separados por branco:2 2


********** matriz de precos
entre com o elemento p[ 0][ 0] = 1.0

entre com o elemento p[ 0][ 1] = 2.0

entre com o elemento p[ 1][ 0] = 2.0

entre com o elemento p[ 1][ 1] = 1.0


********** vetor de quantidades
entre com o elemento q[ 0] = 10

entre com o elemento q[ 1] = 20


********** valor total da compra em cada loja
valor da compra na loja  0 =      50.00
valor da compra na loja  1 =      40.00

********** loja de valor minimo
na loja  1 o valor da compra minimo =           40.00

********** preco minimo de cada produto
produto 0 na loja  0 tem valor minimo =            1.00
produto 1 na loja  1 tem valor minimo =            1.00

********** preco minimo possivel da compra
preco minimo possivel da compra =          30.00
```

P90a) Modifique o programa acima, mostrando todas as lojas que tem preço mínimo (pode haver mais que uma). Modifique também para mostrar todas as lojas cujo preço de um produto é mínimo (pode haver mais que uma).

**Exemplos de programas usando matrizes de mais de 2 índices**

P91) Dada uma seqüência de valores que são idades das pessoas de uma população, determinar quantas pessoas de cada idade existem. Supor idade entre 0 e 100 anos, inteiro e ler idades até ser digitado uma idade inválida (< 0 ou > 100). Não usaremos matriz, mas é só para repetir a idéia, já usada anteriormente, de usar o valor como índice e introduzir os próximos exercícios.

```
#define max 101
#include <stdio.h>
```

```
int main()
{int idades[max];

 int id, i;

 /* zerar os contadores */
 for (i = 0; i < max; i++) idades[i] = 0;

 /* ler a primeira id para começar */
 printf("\n entre com a idade :");
 scanf("%d", &id);
 /* repita enquanto id digitada for válida */
 while (id >= 0 && id <= 100)
       {/* incrementa o contador correspondente */
        idades[id]++;
        /* le o próximo */
        printf("\n entre com a idade :");
        scanf("%d", &id);
       }
 /* imprime quantos de cada idade há */
 for (i = 0; i < max; i++)
      printf("\n tem %5d pessoas com %3d anos ", idades[i], i);
}
```

P92) Idem, dados uma seqüência de pares de valores que são idades e alturas das pessoas de uma população, determinar quantas pessoas de cada idade e de cada altura existem. Supor a idade entre 0 e 100 anos e altura entre 0 e 250 cm, inteiros. Ler idades e alturas até ser digitado uma idade inválida(< 0 ou > 100) ou uma altura inválida (< 0 ou > 250).

```
#define maxidade 101
#define maxaltura 251
#include <stdio.h>
int main()
{int freq[maxidade][maxaltura];

 int id, alt, i, j;

 /* zerar os contadores */
 for (i = 0; i < maxidade; i++)
       for (j = 0; j < maxaltura; j++) freq[i][j] = 0;

 /* ler o primeiro par para começar */
 printf("\n entre com a idade e altura :");
 scanf("%d%d", &id, &alt);
 /* repita enquanto id e alt digitadas forem válidas */
 while (id >= 0 && id <= 100 && alt >= 0 && alt <= 250)
       {/* incrementa o contador correspondente */
        freq[id][alt]++;
        /* lê o próximo */
        printf("\n entre com a idade e altura :");
        scanf("%d%d", &id, &alt);
       }
 /* imprime quantos de cada idade e cada altura há */
 for (i = 0; i < maxidade; i++)
  for (j = 0; j < maxaltura; j++)
   /* imprime só os  diferentes de zero */
   if (freq[i][j] != 0)
```

```
    printf("\n tem %5d pessoas com %3d anos e %3d cm", freq[i][j], i, j);
}
```

P93) Idem, dados uma seqüência de triplas de valores que são idades, alturas e pesos das pessoas de uma população, determinar quantas pessoas de cada idade, altura e peso existem. Supor idade entre 0 e 100 anos e altura entre 0 e 250 cm e peso entre 0 e 200 kg, inteiros.

P94) Idem, dados uma seqüência de quádruplas, incluindo o sexo. Sexo pode ser 0 ou 1.

P95) Idem, dados uma seqüência de quíntuplas, incluindo a cor. A cor pode ser 0, 1, 2 ou 3.

## Mais exemplos de funções usando matrizes

P96) Faça uma função `verifica_linhas(double a[][tammax], int n)` que verifica se a matriz a de `n x n`, possui 2 linhas iguais, devolvendo 0 se sim ou –1 se não.

P96a) idem para 2 colunas iguais.

P97) Faça a função ind_`min_max (double a[][tammax],int n,*imin,*jmin,*imax,jmax)` que devolve o índices do menor elemento da matriz a devolvendo em `imin` e `jmin` e do maior elemento devolvendo em `imax` e `jmax`).

## Matrizes Vetores e Funções

Seja uma matriz de 10 linhas por 20 colunas **`Mat[10][20]`**.
Podemos considerá-la como 10 vetores de 20 elementos cada. Cada linha da matriz pode ser vista como um vetor. A maneira de fazer referência a cada um dos vetores individualmente é usar apenas um índice. Por exemplo, **`Mat[2]`** refere-se ao vetor **`Mat[2][0] ...Mat[2][19]`**.

Veja o programa abaixo. A função ImprimeVetor é usada para imprimir cada uma das linhas de uma matriz com 2 índices.

```
#include <stdio.h>
#include <stdlib.h>

void ImprimeVetor(int X[], int N) {
    /* imprime vetor N elementos do vetor X */
    int k;
    for(k = 0; k < N; k++) printf("%5d", X[k]);
    printf("\n");
}

int main() {

   int k, j, n, mat[10][10];
   printf("Entre com n:");
   scanf("%d", &n);
   printf("Entre com n x n elementos da matriz:");
   for (k = 0; k < n; k++)
      for (j = 0;j < n; j++) scanf("%d", &mat[k][j]);
   /* imprima a matriz usando a ImprimeVetor acima */
   for (k = 0; k < n; k++) ImprimeVetor(mat[k], n);
   system("PAUSE");
   return 0;
}
```

O mesmo ocorre se a matriz tiver mais de 2 dimensões. Por exemplo, a matriz **`Mat3[10][20][5]`** pode ser vista como 10x20 = 200 vetores de 5 elementos cada. Para referenciar cada um dos vetores individualmente, basta usar apenas 2 índices. Por exemplo, **`Mat3[2][5]`** refere-se ao vetor **`Mat3[2][5][0] ... Mat3[2][5][4]`**.

Veja o programa abaixo. A função **`ImprimeVetor`** é usada para imprimir cada uma das linhas de uma matriz com 3 índices.

```
#include <stdio.h>
#include <stdlib.h>

void ImprimeVetor(int X[], int N) {
    /* imprime vetor N elementos do vetor X */
    int k;
    for(k = 0; k < N; k++) printf("%5d", X[k]);
    printf("\n");
}

int main() {

   int k, j, l, n, m, p, mat[10][10][10];
   printf("Entre com n, m e p:");
   scanf("%d%d%d", &n, &m, &p);
   printf("Entre com n x m x p elementos da matriz:");
   for (k = 0; k < n; k++)
      for (j = 0; j < m; j++)
         for (l = 0; l < p; l++) scanf("%d", &mat[k][j][l]);
   /* imprima a matriz usando a ImprimeVetor acima */
   for (k = 0; k < n; k++)
      for (j = 0; j < m; j++) ImprimeVetor(mat[k][j], p);
   system("PAUSE");
   return 0;
}
```

A mesma matriz **`Mat3[10][20][5]`** acima pode ser vista como 10 matrizes de 20 linhas por 5 colunas cada uma. Por exemplo, **`Mat3[2]`** refere-se a matriz **`Mat3[2][0][0] ... Mat3[2][19][4]`**.

Veja o programa abaixo. A função **`ImprimeMatriz`** é usada para imprimir cada uma das matrizes de uma matriz de 3 índices.

```
#include <stdio.h>
#include <stdlib.h>

void ImprimeMatriz(int X[][10], int N, int M) {
    /* imprime matriz de N linhas por N colunas */
    int i, j;
    for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) printf("%5d", X[i][j]);
      printf("\n");
    }
    printf("\n");
}

int main() {

   int k, j, l, n, m, p, mat[10][10][10];
```

```
   printf("Entre com n, m e p:");
   scanf("%d%d%d", &n, &m, &p);
   printf("Entre com n x m x p elementos da matriz:");
   for (k = 0; k < n; k++)
      for (j = 0; j < m; j++)
         for (l = 0; l < p; l++) scanf("%d", &mat[k][j][l]);
   /* imprima a matriz usando a ImprimeMatriz acima */
   for (k = 0; k < n; k++) ImprimeMatriz(mat[k], m, p);
   system("PAUSE");
   return 0;
}
```