22 Matrizes

Ronaldo F. Hashimoto e Carlos H. Morimoto

O objetivo desta aula é introduzir o tipo **matriz**. Ao final dessa aula você deverá saber:

- descrever o que são matrizes em C.
- Declarar matrizes.
- Como acessar elementos de uma matriz e percorrer uma matriz.
- Utilizar matrizes para resolver problemas computacionais.

22.1 Matrizes

Matrizes são estruturas indexadas (em forma matricial - como ilustrado na figura abaixo) utilizadas para armazenar dados de um mesmo tipo: **int**, **char**, **float** ou **double**.

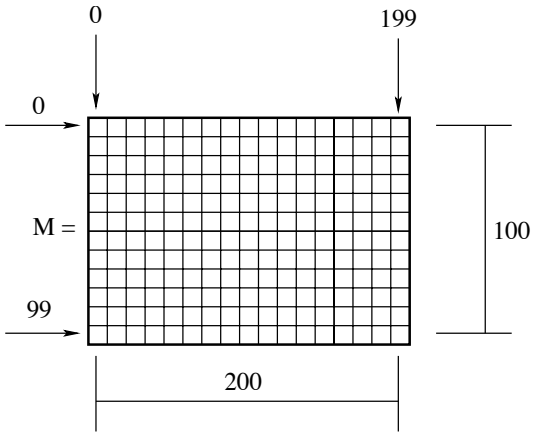


Figura 1: Uma matriz contem elementos de um mesmo tipo, com início em (0,0).

22.1.1 Declaração de Matrizes

A declaração de uma matriz é feita da seguinte forma:

`<tipo_da_matriz> <nome_da_matriz> [<numero_de_linhas>][<numero_de_colunas>];`

Exemplos:

- **int** `M`[100][200];

```
100 é o número de linhas!
200 é o número de colunas!
```

  A declaração acima aloca uma matriz com 100 linhas e 200 colunas na memória. Cada casa da matriz guarda um **int**.

- **float** `x`[20][30];

```
20 é o número de linhas!
30 é o número de colunas!
```

  A declaração acima aloca uma matriz com 20 linhas e 30 colunas na memória. Cada casa da matriz guarda um **float**.

Observação Importante:

1. Na **declaração de matriz**, o que está entre colchetes deve ser um **número constante**.
2. Assim, não é possível fazer algo deste tipo:

```
int nL = 20, nC = 30; /* num Linhas e num Colunas */
float x[nL][nC]; /* não se deve usar variáveis entre colchetes na declaração */
```

ou

```
int nL, nC;

printf ("Entre com nL>0 e nC>0: ");
scanf ("%d %d", &nL, &nC);

float x[nL][nC];
```

O correto seria:

```
int nL, nC;
float x[20][30]; /* o correto é declarar sempre tamanhos fixos */
```

22.1.2 Uso de Matrizes

- São usados índices para acessar uma linha e uma coluna de uma matriz.
- Os índices são números naturais.
- O índice da **primeira** linha é sempre **zero**.
- O índice da **primeira** coluna é sempre **zero**.

22.1.3 Exemplo de Uso de Matrizes

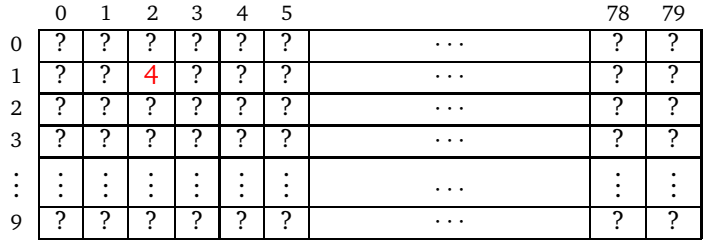
- Exemplo:

```
1       # include <stdio.h>
2
3       int main () {
4         int A[10][80], lin, col;
5
6         A[1][2] = 4; /* casa da linha 1 e coluna 2 recebe o inteiro 4 */
7         lin = 2; col = 3;
8         A[lin][col] = 5; /* casa de índice 2 do vetor v recebe o inteiro 3 */
9         A[A[lin-1][col-1] - 1][A[lin][col]] = 10; /* vc saberia dizer qual casa
10                                                  * da matriz A recebe o inteiro 10?
11                                                  */
12        return 0;
13      }
```

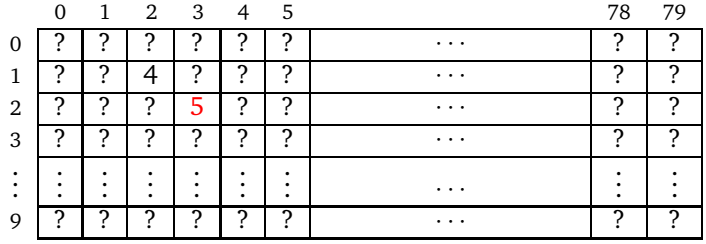
Na Linha 4, a matriz `A` com 10 linhas e 80 colunas é declarada:

	0	1	2	3	4	5		78	79
0	?	?	?	?	?	?	...	?	?
1	?	?	?	?	?	?	...	?	?
2	?	?	?	?	?	?	...	?	?
3	?	?	?	?	?	?	...	?	?
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...	⋮	⋮
9	?	?	?	?	?	?	...	?	?

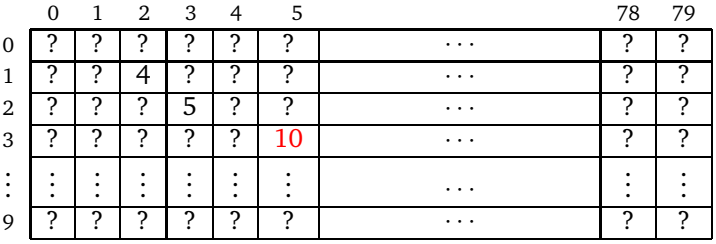
Na Linha 6, casa de linha 1 e coluna 2 da matriz `A` recebe o inteiro 4:



Na Linha 8, casa de linha 2 e coluna 3 da matriz `A` recebe o inteiro 5:



Na Linha 9, como `lin`=2, `col`=3, temos que `A[lin-1][col-1]`=`A[1][2]`=4 e `A[lin][col]`=`A[2][3]`=5. Assim, temos que `A[A[lin-1][col-1] - 1][A[lin][col]]`=`A[4-1][5]`=`A[3][5]`=10. Dessa forma, no comando da Linha 9, a linha 3 e coluna 5 da matriz `A` recebe o inteiro 10:



22.2 Percorrimento de Matrizes

Percorrer uma matriz significa visitar cada elemento da matriz (ou um subconjunto de elementos) de casa em casa em uma determinada ordem. Por exemplo, podemos percorrer apenas os elementos da diagonal principal de uma matriz quadrada, ou percorrer todos os elementos de uma matriz retangular, linha a linha, a partir da linha 0 (zero), e para cada linha, visitar os elementos de cada coluna, a partir da coluna 0 (zero). Nesse último caso é necessário saber o número de linhas e colunas que se deve fazer este percorrimento. Este número normalmente é guardado em duas variáveis inteiras (no nosso exemplo, as variáveis `nL` e `nC`).

Muitos problemas computacionais que envolvem matrizes têm como solução o uso de um padrão para percorrimento de matrizes.

Para os exemplos desta seção, vamos considerar a seguinte declaração de matriz:

```
int A[20][30];
```

e as variáveis inteiras

```
int lin, col, nL, nC, cont;
```

onde `nL` e `nC` são o número de linhas e colunas que devem ser consideradas na matriz `A`. É claro que neste caso, `nL` tem que ser menor que 20 e `nC` menor que 30.

22.2.1 Percorrimento de uma Linha:

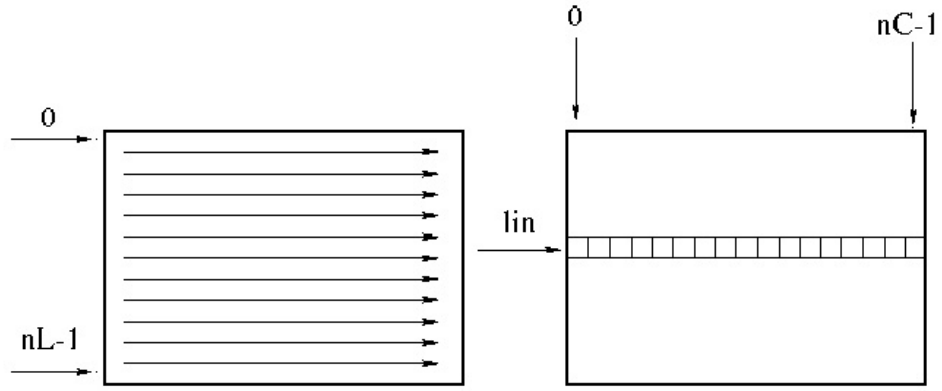


Figura 2: Percorrimento de uma linha de uma matriz.

Um padrão para percorrer uma linha `lin` da matriz `A` é usar um comando de repetição (no caso, vamos usar o comando **for**) com uma variável inteira `col` para o índice das colunas da matriz `A`:

```
/* para uma linha fixa lin */
for (col=0; col < nC; col++) {
  /* comandos usando a matriz A[lin][col] */
}
```

Exemplo:

```
cont = 0;
/* para uma linha fixa lin */
for (col=0; col < nC; col++) {
  A[lin][col] = cont;
  cont++;
}
```

22.2.2 Percorrimento Completo da Matriz:

Um padrão para percorrer completamente a matriz `A` (isto é, as `nL` linhas e as `nC` colunas) por linhas é usar dois comandos de repetição (no caso, vamos usar o comando **for**) com duas variáveis inteiras `lin` e `col`, um para percorrer as linhas e a outra para percorrer as colunas da matriz `A`:

```
for (lin=0; lin < nL; lin++) {
  for (col=0; col < nC; col++) {
    /* comandos usando a matriz A[lin][col] */
  }
}
```

O exemplo abaixo

```
for (lin=0; lin < nL; lin++) {
  for (col=0; col < nC; col++) {
    A[lin][col] = 0;
  }
}
```

inicializa as `nL` linhas e as `nC` colunas da matriz `A` com zero.

22.2.3 Observação sobre Percorrimento

Na declaração de uma matriz é definido um número fixo de linhas e colunas, uma vez que sempre deve-se colocar uma constante na definição do número de linhas e colunas da matriz. Por exemplo:

```
int A[20][30];
```

Mas, como podemos ver nos exemplos de percorrimento para ler e imprimir uma matriz, um usuário não necessariamente irá usar todas as linhas e colunas disponíveis da matriz. Note que no padrão de percorrimento por linhas deve sempre existir duas variáveis indicando quantas linhas e colunas da matriz estão sendo verdadeiramente usadas (variável `nL` e `nC` do padrão).

Assim, normalmente, em problemas computacionais que envolvem matrizes deve-se sempre ter duas variáveis inteiras associadas à matriz que diz quantas linhas e colunas da matriz estão sendo usadas (por exemplo, variável inteira `nL` e `nC` associadas à matriz `A` nos exemplos de leitura e impressão de matrizes).

22.3 Leitura de uma Matriz

Para leitura de uma matriz, devemos ler elemento a elemento usando o padrão de percorrimento por linhas.

```
1   # include <stdio.h>
2   # define MAX_L 100
3   # define MAX_C 200
4
5   int main () {
6     float A[MAX_L][MAX_C];
7     int lin, col, nL, nC;
8
9     printf ("Entre com 0<nL<%d: ", MAX_L);
10    scanf ("%d" &nL);
11
12    printf ("Entre com 0<nC<%d: ", MAX_C);
13    scanf ("%d" &nC);
14
15    /* percorrer a matriz A elemento a elemento
16     * colocando o valor lido pelo teclado */
17    for (lin=0; lin < nL; lin++) {
18      for (col=0; col < nC; col++) {
19        printf ("Entre com A[%d][%d] = ", lin, col);
20        scanf ("%f", &A[lin][col]);
21      }
22    }
23
24    return 0;
25  }
```

Observe com cuidado a linha do programa utilizada para ler o elemento da linha `lin` e coluna `col` da matriz `A`:

```
scanf ("%f", &A[lin][col]);
```

A linha `lin` e a coluna `col` da matriz `A`, ou seja, `A[lin][col]`, é utilizada da mesma forma que utizamos qualquer variável até o momento, ou seja, precedida pelo caractere '&'.

Note que neste exemplo definimos as constantes `MAX_L` e `MAX_C` usando o "comando" `define`. Observe que `MAX_L` e `MAX_C` são constantes e não variáveis. Para saber mais sobre a definição de constantes, veja o material didático **Alguns Detalhes da Linguagem C**.

22.4 Impressão de uma Matriz

Para impressão de uma matriz, devemos imprimir elemento a elemento usando o padrão de percorrimento por linhas.

```c
#include <stdio.h>

#define MAX_L 100
#define MAX_C 200

int main () {
  float A[MAX_L][MAX_C];
  int lin, col, nL, nC;

  printf ("Entre com 0<nL<%d: ", MAX_L);
  scanf ("%d" &nL);

  printf ("Entre com 0<nC<%d: ", MAX_C);
  scanf ("%d" &nC);

  /* percorrer a matriz A elemento a elemento
   * imprimindo o valor de cada casa */
  for (lin=0; lin<nL; lin++) {
    for (col=0; col<nC; col++) {
      printf ("%f ", A[lin][col]);
    }
    printf ("\n");
  }

  return 0;
}
```

22.5 Exercícios Comentados

22.5.1 Exercício 1

Faça um programa que leia um inteiro $n < 100$ e os elementos de uma matriz real quadrada $A_{n \times n}$ e verifica se a matriz A tem uma linha, coluna ou diagonal composta apenas por zeros.

Percorrimento de uma Linha de uma Matriz:

Para verificar se uma matriz `A` tem uma linha com todos elementos nulos, devemos percorrer uma linha `lin` da matriz `A`.

Ora, nós já conhecemos o padrão para percorrer uma linha `lin` da matriz `A`:

```c
/* para uma linha fixa lin */
for (col=0; col<n; col++) {
  /* comandos usando a matriz A[lin][col] */
}
```

Para contar quantos elementos nulos tem uma linha, podemos usar o padrão de percorrimento de uma linha da seguinte maneira:

```
cont = 0;
/* para uma linha fixa lin */
for (col=0; col<n; col++) {
  if (A[lin][col] == 0)
    cont++;
}
```

Neste exemplo, o padrão conta quantos elementos nulos tem a linha `lin`. Se quisermos saber se a linha `lin` tem todos os elementos nulos, basta comparar se `cont` é igual a `n`. Assim:

```
cont = 0;
/* para uma coluna fixa col */
for (lin=0; lin<n; lin++) {
  if (A[lin][col] == 0)
    cont++;
}
if (cont == n)
  printf ("A linha %d tem todos elementos nulos\n", i);
```

Assim, para verificar se uma matriz `A` tem uma linha com todos elementos nulos, devemos verificar cada linha `lin` da matriz:

```
linha_nula = 0;
for (lin=0; lin<n; lin++) {
  cont = 0;
  /* para uma linha lin */
  for (col=0; col<n; col++) {
    if (A[lin][col] == 0)
      cont++;
  }
  if (cont == n)
    linha_nula = 1;
}
if (linha_nula == 1)
  printf ("Matriz tem uma linha com todos elementos nulos\n",);
```

Percorrimento de uma Coluna de uma Matriz:

Para verificar se uma matriz `A` tem uma coluna com todos elementos nulos, devemos saber como percorrer uma coluna `col` da matriz `A`.

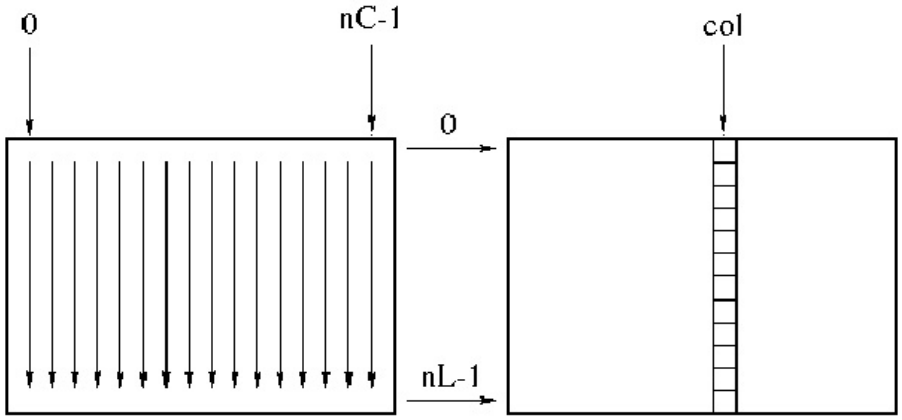


Figura 3: Percorrimento de uma coluna de uma matriz.

Um padrão para percorrer uma coluna `col` de uma matriz `A` é usar um comando de repetição (no caso, vamos

usar o comando **for**) com uma variável inteira `lin` para o índice das linhas da matriz `A`:

```
/* para uma coluna fixa col */
for (lin=0; lin<n; lin++) {
  /* comandos usando a matriz A[lin][col] */
}
```

Exemplos:

```
cont = 0;
/* para uma coluna fixa col */
for (lin=0; lin<n; lin++) {
  A[lin][col] = cont;
  cont++;
}

cont = 0;
/* para uma coluna fixa col */
for (lin=0; lin<n; lin++) {
  if (A[lin][col] == 0)
    cont++;
}
```

No último exemplo, o padrão conta quantos elementos nulos tem a coluna `col`. Se quisermos saber se a coluna `col` de uma matriz $A_{n\times n}$ tem todos os elementos nulos, basta comparar se `cont` é igual a `n`. Assim:

```
cont = 0;
/* para uma coluna fixa col */
for (lin=0; lin<n; lin++) {
  if (A[lin][col] == 0)
    cont++;
}
if (cont == n)
  printf ("A coluna %d tem todos elementos nulos\n", j);
```

Assim, para verificar se uma matriz quadrada `A` tem uma coluna com todos elementos nulos, devemos verificar cada coluna `col` da matriz:

```
coluna_nula = 0;
for (col=0; col<n; col++) {
  cont = 0;
  /* para uma coluna col */
  for (lin=0; lin<n; lin++) {
    if (A[lin][col] == 0)
      cont++;
  }
  if (cont == n)
    coluna_nula = 1;
}
if (coluna_nula == 1)
  printf ("Matriz tem uma coluna com todos elementos nulos\n",);
```

Percorrimento da Diagonal Principal de uma Matriz:

Para verificar se uma matriz quadrada $A_{n\times n}$ tem a diagonal principal com todos elementos nulos, devemos saber como percorrer esta diagonal da matriz `A`.

Como na diagonal principal temos que a linha é igual a coluna, um padrão para percorrer a diagonal principal de `A` é usar um comando de repetição (no caso, vamos usar o comando **for**) com uma variável inteira `lin` para o índice das linhas e colunas da matriz `A`:

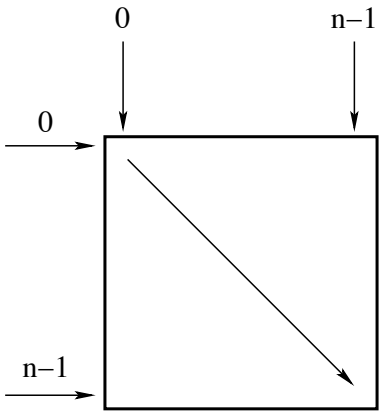


Figura 4: Percorrimento da diagonal.

```
for (lin=0; lin<n; lin++) {
  /* comandos usando a matriz A[lin][lin] */
}
```

Exemplos:

```
cont = 0;
for (lin=0; lin<n; lin++) {
  A[lin][lin] = cont;
  cont++;
}
```

```
cont = 0;
for (lin=0; lin<n; lin++) {
  if (A[lin][lin] == 0)
    cont++;
}
```

No último exemplo, o padrão conta quantos elementos nulos tem a diagonal principal. Se quisermos saber se a diagonal principal de uma matriz $A_{n\times n}$ tem todos os elementos nulos, basta comparar se `cont` é igual a `n`. Assim:

```
cont = 0;
for (lin=0; lin<n; lin++) {
  if (A[lin][lin] == 0)
    cont++;
}
if (cont == n)
  printf ("A diagonal principal tem todos elementos nulos\n");
```

Percorrimento da Diagonal Secundária de uma Matriz:

Para verificar se uma matriz quadrada $A_{n\times n}$ tem a diagonal secundária com todos elementos nulos, devemos saber como percorrer esta diagonal da matriz `A`.

Como na diagonal secundária temos que a soma da linha com a coluna é igual a `n`-1 (ou seja, para uma linha `lin`, a coluna deve ser `n`$-1-$`i`), um padrão para percorrer a diagonal secundária de `A` é usar um comando de repetição (no caso, vamos usar o comando **for**) com uma variável inteira `lin` para o índice das linhas e colunas da matriz `A`:

```
for (lin=0; lin<n; lin++) {
  /* comandos usando a matriz A[lin][n-1-lin] */
}
```

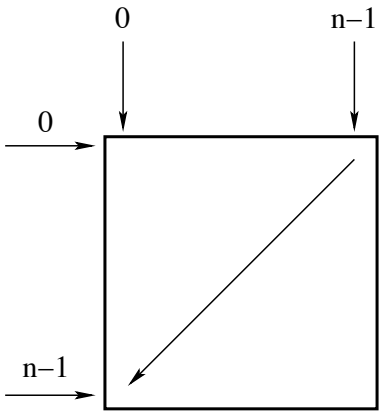


Figura 5: Percorrimento da diagonal secundária.

Exemplos:

```
cont = 0;
for (lin=0; lin<n; lin++) {
  A[lin][n-1-lin] = cont;
  cont++;
}

cont = 0;
for (lin=0; lin<n; lin++) {
  if (A[lin][n-1-lin] == 0)
    cont++;
}
```

No último exemplo, o padrão conta quantos elementos nulos tem a diagonal secundária. Se quisermos saber se a diagonal secundária de uma matriz $A_{n\times n}$ tem todos os elementos nulos, basta comparar se `cont` é igual a `n`. Assim:

```
cont = 0;
for (lin=0; lin<n; lin++) {
  if (A[lin][n-1-lin] == 0)
    cont++;
}
if (cont == n)
  printf ("A diagonal secundária tem todos elementos nulos\n");
```

Juntando Tudo

Fica como exercício você fazer um programa que resolva o Exercício 1, ou seja, fazer um programa que leia um inteiro $n < 100$ e os elementos de uma matriz real $A_{n\times n}$ e verifica se a matriz A tem uma linha, coluna ou diagonal composta apenas por zeros.

22.5.2 Exercício 2

Dado $0 < n < 200$ e uma matriz real $A_{n\times n}$, verificar se A é simétrica.

Uma matriz $A_{n\times n}$ é simétrica se, e somente se, A é igual a sua transposta, ou seja, $A = A^t$.

Neste caso, temos que verificar se cada $A[lin][col]$ é igual a $A[col][lin]$ como indicado na figura. Note que devemos percorrer somente uma parte da matriz, no caso, a parte superior da matriz.

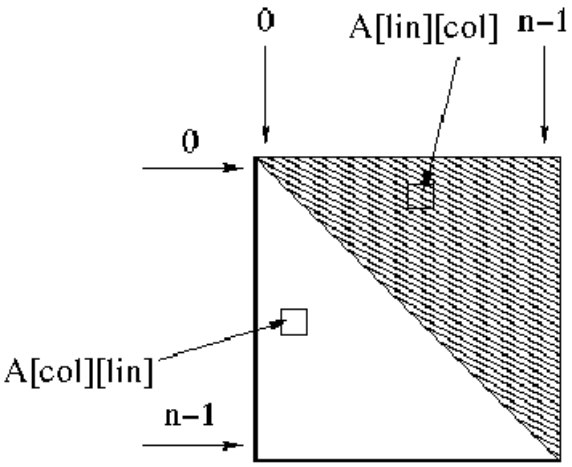


Figura 6: Matriz simétrica.

Percorrimento da Parte Superior de Matrizes por Linha

Para uma linha `lin`, temos que começar a percorrer as colunas a partir da coluna `lin+1` até a última coluna `n−1`, como mostra a figura 7.

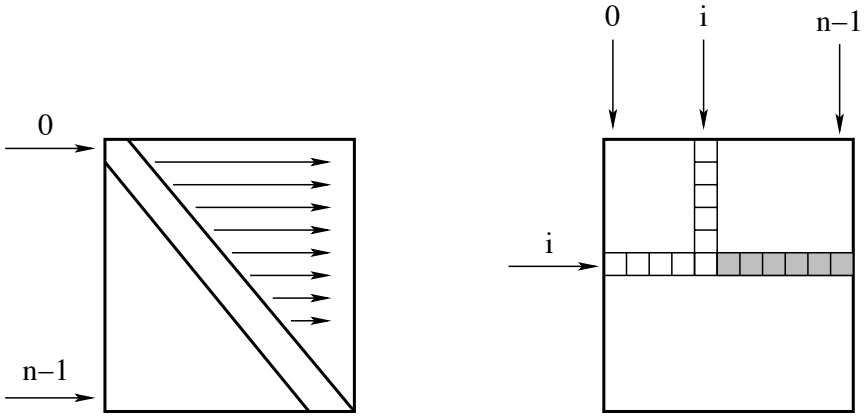


Figura 7: Percorrimento da parte superior da matriz por linha.

Assim, um padrão para percorrer uma linha `lin` da parte superior de uma matriz `A` é usar um comando de repetição (no caso, vamos usar o comando **for**) com uma variável inteira `col` para o índice das colunas da matriz `A`:

```
/* para uma linha fixa lin */
for (col=lin+1; col<n; col++) {
  /* comandos que fazem algo com A[lin][col] */
}
```

Exemplo:

```
cont = 0;
/* para uma linha fixa lin */
for (col=lin+1; col<n; col++) {
  A[lin][col] = cont;
  cont++;
}
```

Um padrão para percorrer completamente a parte superior da matriz `A` por linhas é usar dois comandos de repetição (no caso, vamos usar o comando **for**) com duas variáveis inteiras `lin` e `col`, um para percorrer as linhas e a outra para percorrer as colunas da matriz `A`:

```
for (lin=0; lin<n; lin++) {
  /* para uma linha fixa lin */
  for (col=lin+1; col<n; col++) {
    /* comandos que fazem algo com A[lin][col] */
  }
}
```

Exemplo:

```
cont = 0;
for (lin=0; lin<n; lin++) {
  /* para uma linha fixa lin */
  for (col=lin+1; col<n; col++) {
    A[lin][col] = cont;
    cont++;
  }
}
```

Assim, para verificar se uma matriz real $A_{n \times n}$, verificar se A é simétrica, podemos fazer:

```
simetrica = 1;
for (lin=0; lin<n; lin++) {
  /* para uma linha fixa lin */
  for (col=lin+1; col<n; col++) {
    if (A[lin][col] != A[col][lin])
      simetrica = 0;
  }
}
printf ("Matriz ");
if (simetrica == 0) {
  printf ("nao ");
}
printf ("eh Simetrica\n");
```

Solução Completa:

```c
# include <stdio.h>

# define MAX 100

int main () {
  int lin, col, n, simetrica = 1;
  float A[MAX][MAX];

  printf ("Entre com 0<n<100: ");
  scanf ("%d" &n);

  /* percorrer a matriz A elemento a elemento
  * colocando o valor lido pelo teclado */
  for (lin=0; lin<n; lin++) {
    /* para uma linha fixa lin */
    for (col=0; col<n; col++) {
      printf ("Entre com A[%d][%d] = ", lin, col);
      scanf ("%f", &A[lin][col]);
    }
  }

  /* verificando se eh simetrica */
  for (lin=0; lin<n; lin++) {
    /* para uma linha fixa i */
    for (col=lin+1; col<n; col++) {
      if (A[lin][col] != A[col][lin])
      simetrica = 0;
    }
  }

  /* Impressao da Resposta Final */
  printf ("Matriz ");
  if (simetrica == 0) {
    printf ("nao ");
  }
  printf ("eh Simetrica\n");
  return 0;
}
```

22.6 Erros Comuns

Ao desenvolver seus programas com matrizes, preste atenção com relação aos seguintes detalhes:

- **índices inválidos:** tome muito cuidado, especialmente dentro de um **while** ou **for**, de não utilizar índices negativos ou maiores que o tamanho máximo designado para as linhas e colunas da matriz.
- A **definição do tamanho das linhas e colunas da matriz** se faz na declaração da matriz. Os tamanhos das linhas e colunas são constantes; só mudando a sua declaração é que podemos alterar estes tamanhos. Isso significa que podemos estar "desperdiçando" algum espaço da memória por não estar usando todas as casas da matriz. Não cometa o erro de ler `nL` e `nC`, onde `nL` e `nC` seriam os tamanhos das linhas e colunas da matriz, e tentar "declarar" a matriz em seguida.

22.7 Percorrimento de Matrizes

Muitos problemas computacionais que envolvem matrizes têm como soluções o uso de um padrão para percorrimento de matrizes. Nesta aula aprendemos:

- Percorrimento de uma Linha de uma Matriz.
- Percorrimento Completo da Matriz por Linhas.
- Percorrimento de uma Coluna de uma Matriz.
- Percorrimento Completo da Matriz por Colunas.
- Percorrimento da Diagonal Principal de uma Matriz.
- Percorrimento da Diagonal Secundária de uma Matriz.
- Percorrimento da Parte Superior de Matrizes por Linha.

Há muitas outras forma de percorrer matrizes, como mostra a figura 8.

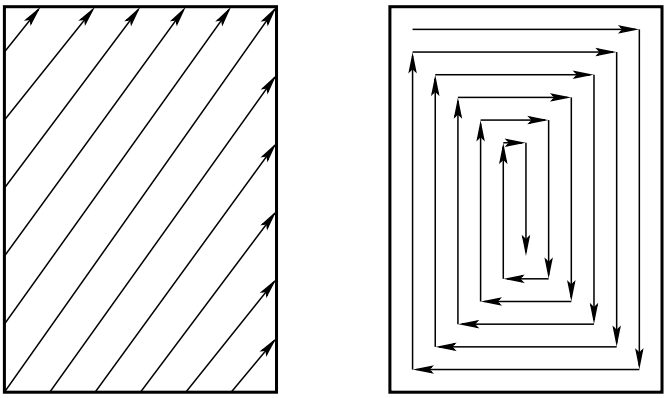


Figura 8: Outras formas de percorrer uma matriz.

22.8 Exercícios Recomendados

1. Escreva um programa que, dadas duas matrizes $A_{m\times n}$ e $B_{n\times p}$, calcula a matriz $C_{m\times p}$ que é o produto de A por B. Note que, para ler as matrizes, é necessário primeiro ler os seus tamanhos $m, n,$ e p.
2. Imprimir as n primeiras linhas do triângulo de Pascal.
3. Um jogo de palavras cruzadas pode ser representado por uma matriz $A_{m\times n}$ onde cada posição da matriz corresponde a um quadrado do jogo, sendo que 0 (zero) indica um quadrado branco e -1 indica um quadrado preto. Indicar na matriz as posições que são início de palavras horizontais e/ou verticais nos quadrados correspondentes (substituindo os zeros), considerando que uma palavra deve ter pelo menos duas letras. Para isso, numere consecutivamente tais posições.

   Exemplo: Dada a matriz:

$$\begin{pmatrix} 0 & -1 & 0 & -1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

   A saída deverá ser:

$$\begin{pmatrix} 1 & -1 & 2 & -1 & -1 & 3 & -1 & 4 \\ 5 & 6 & 0 & 0 & -1 & 7 & 0 & 0 \\ 8 & 0 & -1 & -1 & 9 & 0 & -1 & 0 \\ -1 & 10 & 0 & 11 & 0 & -1 & 12 & 0 \\ 13 & 0 & -1 & 14 & 0 & 0 & -1 & -1 \end{pmatrix}$$