Matrizes em C/*C*++

CI208/UFPR: Programação de Computadores

Prof. Luciano Silva

Setembro/2020

Matrizes (aula anterior - vetores)

- Uma matriz é uma coleção de elementos de dados que são do mesmo tipo
  - Exemplos: coleção de inteiros, coleção de caracteres, coleção de floats

- Elementos (ou células) da matriz podem ser indicados pelos índices das células (números indicando posição das células)

- É um conhecido conceito matemático:
  - uma matriz é um arranjo retangular (ou tabela) de números, símbolos ou expressões, organizados em linhas e colunas.
  - Exemplo: a dimensão da matriz M abaixo de 2 × 3, ou seja: existem duas linhas e três colunas

$$\begin{bmatrix} 4.3 & 7.0 & -1.2 \\ -3.0 & 2.2 & 19.1 \end{bmatrix}$$

Declaração de Matrizes em C/C++

- A sintaxe para **declarar** variáveis do tipo matriz **de N dimensões** é a seguinte:

<tipo> nome_da_matriz [<dim0>] [<dim1>] . . . [<dimN-1>];

onde:

  - <tipo> é o **tipo** de cada célula da matriz
  - [<dim>] é o **número de elementos** em uma dimensão da matriz

- Acima as dimensões estão numeradas de 0 a N-1, sendo a dimensão [<dim0>] mais à esquerda.
- No código, cada parte entre colchetes **[ ]** é usada **para indexar** uma das dimensões da matriz.

- Neste curso usaremos matrizes com apenas duas ou três dimensões.

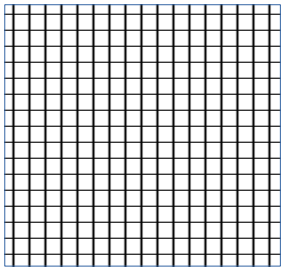
Declaração de Matrizes em C/C++

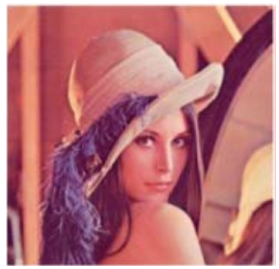
- Exemplo de matriz com 3 dimensões:

  - int imagem_rgb[200][200]**[3]**;

- A matriz imagem_rgb pode ser usada para armazenar uma foto de tamanho 200x200 pixels, **colorida**, onde cada pixel RGB tem 3 componentes inteiras nos índices mais à direita:
  - quantidade de vermelho (R) no índice [0]
  - quantidade de verde (G) no índice [1]
  - quantidade de azul (B) no índice [2]

Matriz (200x200) pixels RGB





Imagem

Declaração de Matrizes em C/C++

- Outra representação possível para a imagem, com "**canais** RGB", nesse caso a matriz com 3 dimensões seria:

int imagem_rgb**[3]**[200][200];

- A matriz imagem_rgb pode ser usada para armazenar uma foto de tamanho 200x200 pixels, **colorida**, onde os **canais** RGB são indexados pela dimensão mais à esquerda.
  - no índice [0] temos o canal R (com a componente vermelha de **200x200** int)
  - no índice [1] temos o canal G (com a componente verde de **200x200** int)
  - no índice [2] temos o canal azul (B (com a componente vermelha de **200x200 int**)

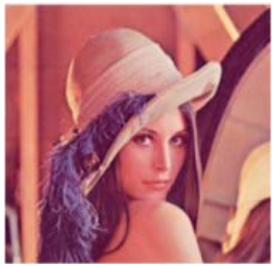
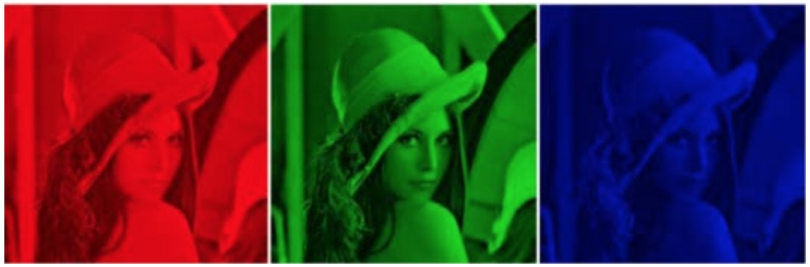


Imagem representada



imagem_rgb: são canais de 200x200 ints

Declaração de Matrizes em C/C++

- A sintaxe para uma matriz com 2 dimensões:

  <tipo> nome_da_matriz [<nLinhas>] [<nColunas>]

- Exemplo:

  float **notas**[4][3];

- A matriz **notas** possui 4*3 = 12 variáveis do tipo float, organizadas em:
  - 4 linhas, indexadas de 0 a 3 (0 a nLinhas-1)
  - 3 colunas, indexadas de 0 a 2 (0 a nColunas-1)

Declaração de Matrizes em C/C++

- Na declaração de uma matriz geralmente são usadas constantes para definir as dimensões máximas da matriz #define

```
#define NLIN   4
#define NCOL  3
...
float notas[ NLIN ][ NCOL ];
```

- Dependendo do problema, você pode caracterizar as dimensões usando nomes mais apropriados para estas constantes

```
#define NALUNOS   30      // quantidade de alunos na turma
#define NPROVAS  3       // quantidade de provas realizadas
float notas[ NALUNOS ][ NPROVAS ];
```

```
// matriz com a nota das 3 provas de cada aluno da turma
// alunos são numerados de 0 a 29
```

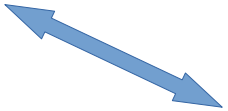
Inicializando valores na Matriz

- Considere a declaração da matriz M de inteiros:

```
int M[3][3]={1,2,3,4,5,6,7,8,9};
int M[3][3]={{1,2,3},{4,5,6},{7,8,9}};
```

- Em ambos os casos é criada a matriz:

1 2 3
4 5 6
7 8 9



Os índices de M são:

$$\begin{bmatrix} M[0][0] & M[0][1] & M[0][2] \\ M[1][0] & M[1][1] & M[1][2] \\ M[2][0] & M[2][1] & M[2][2] \end{bmatrix}$$

Acessando valores da Matriz

- Considere a declaração da matriz M de inteiros:

```
#define NLIN    4
#define NCOL   3
...
int M[ NLIN ][ NCOL ];          // declara
```

- Usualmente a leitura de valores para a matriz se faz assim:

```
for (int l=0; l < NLIN; l++)
    for (int c=0; c < NCOL; c++)
      cin >> M[l][c];                  // usa no código
```

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

Receita!

- Considere a seguinte entrada com números esta sequência:

  30 31 32

  40 41 42

  50 51 52

  60 61 62

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30		

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30	31	

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30	31	32

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30	31	32
40		

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30	31	32
40	41	

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32

40 41 42

50 51 52

60 61 62

Matriz M -->

30	31	32
40	41	42

Acessando valores da Matriz

- Neste caso os valores de entrada são armazenados da 1a. Linha até a última, preenchendo em cada linha da 1a. Coluna até a última.

```
for (int l=0; l < NLIN; l++)
   for (int c=0; c < NCOL; c++)
      cin >> M[l][c];
```

- Considere a seguinte entrada com números esta sequência:

30 31 32
40 41 42
50 51 52
60 61 62

Matriz M -->

30	31	32
40	41	42
50	51	52
60	61	62

Acessando valores da Matriz

- Podem existir situações onde é determinado outra sequência na ordem de leitura dos dados para a matriz. Qual seria a receita para este forma de leitura?

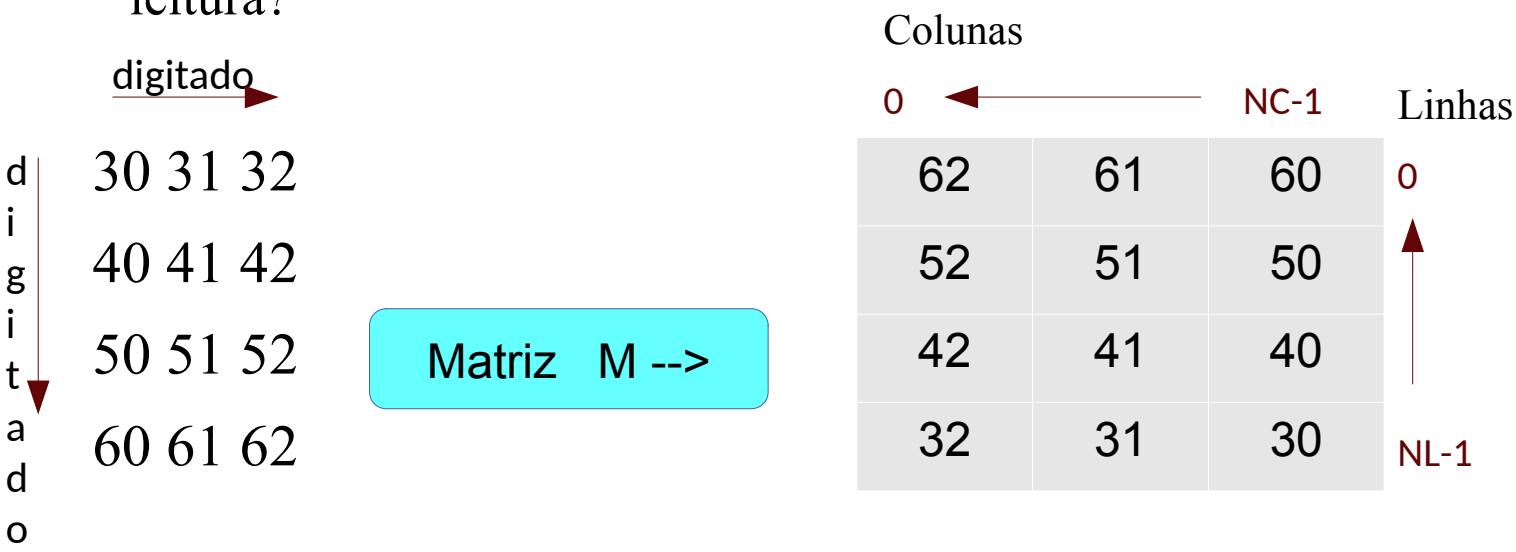
30 31 32
40 41 42
50 51 52
60 61 62

Matriz M -->

62	61	60
52	51	50
42	41	40
32	31	30

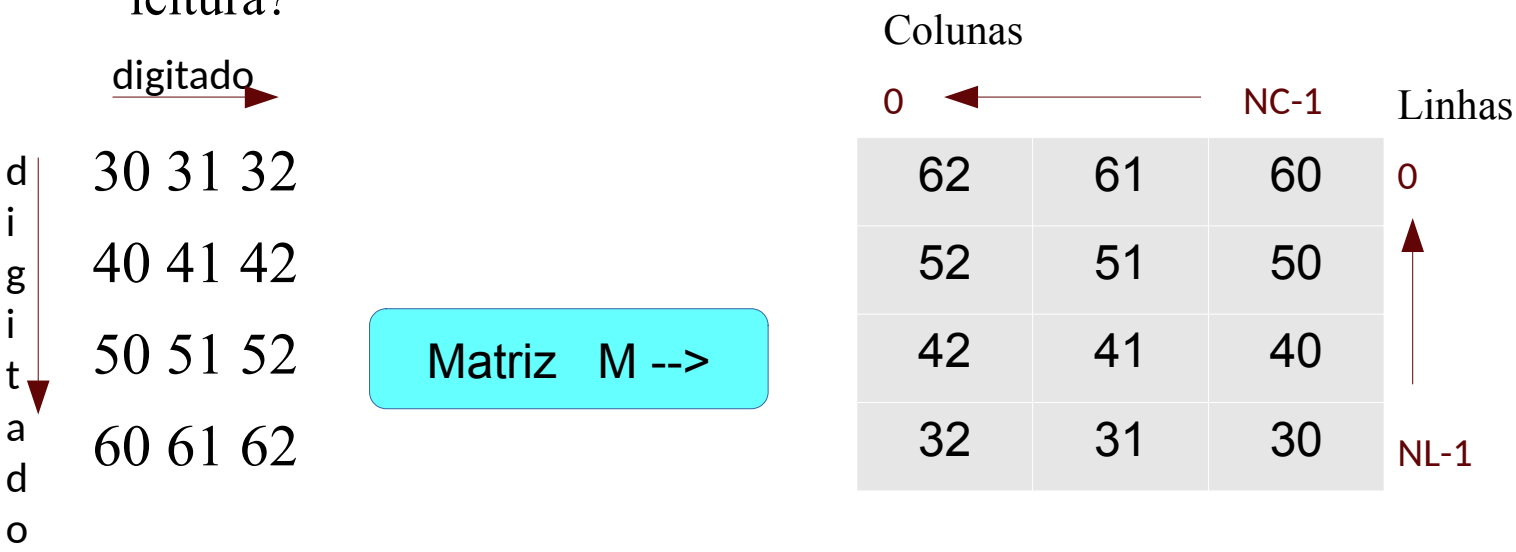
Acessando valores da Matriz

- Podem existir situações onde é determinado outra sequência na ordem de leitura dos dados para a matriz. Qual seria a receita para este forma de leitura?



Acessando valores da Matriz

- Podem existir situações onde é determinado outra sequência na ordem de leitura dos dados para a matriz. Qual seria a receita para este forma de leitura?




```
// supondo M declarado como int M[ NL ][ NC ];
for (int l=NL-1; l>=0; l--)
  for (int c=NC-1; c>=0; c--)
      cin >> M[l][c];
```

Receita!

- preencher de baixo para cima (linhas)
- da direita para a esquerda

Matrizes e funções

- As matrizes podem ser passadas como parâmetros para funções
- Todas as dimensões da matriz, exceto a primeira, precisam ser explicitamente especificadas no cabeçalho e protótipo da função.

- Por exemplo uma função que apenas imprime uma matriz de inteiros X de tamanho MxN seria:

```
void imprime_matriz( int X[][ N ], int nl, int nc )
{
      for (int l=0; l < nl; l++) {
           for (int c=0; c < nc; c++)
                cout << X[l][c] << "  ";
            cout << endl;
      }
 }
```

Note que:

* podemos imprimir (usar) menos linhas e menos colunas
* **nl** indica número de linhas a imprimir
* **nc** indica número de colunas a imprimir

Matrizes e funções: mostrando o uso da funçao anterior

```
#define M  20          // 20 linhas
#define N  10          // 10 colunas
...
void imprime_matriz( int X[][ N ], int nl, int nc )
{
    for (int l=0; l < nl; l++) {
        for (int c=0; c < nc; c++)
            cout << X[l][c] << "   ";
        cout << endl;
    }
}
...
int main() {
  int A[ M ][ N ];  // declara matriz A de MxN
  ... // faz leitura e processamente de A

  imprime_matriz( A, M, N );       // imprime TODA a matriz
  ...
```

Matrizes e funções: mostrando o uso da funçao anterior

```
#define M  20          // 20 linhas
#define N  10          // 10 colunas
...
void imprime_matriz( int X[][ N ], int nl, int nc )
{
    for (int l=0; l < nl; l++) {
        for (int c=0; c < nc; c++)
            cout << X[l][c] << "  ";
        cout << endl;
    }
}
...
int main() {
  int A[ M ][ N ];  // declara matriz A de MxN
  ... // faz leitura e processamente de A

  imprime_matriz( A, M, N/2 );      // imprime somente metade das colunas
  ...
```

Note que:

* podemos imprimir (usar) menos linha e menos colunas
* **nl** indica número de linhas a imprimir
* **nc** indica número de colunas a imprimir

Matrizes e funções: mostrando o uso da funçao anterior

```
#define M  20          // 20 linhas
#define N  10          // 10 colunas
...
void imprime_matriz( int X[][ N ], int nl, int nc )
{
    for (int l=0; l < nl; l++) {
        for (int c=0; c < nc; j++)
            cout << X[l][c] << "  ";
        cout << endl;
    }
}
...
int main() {
  int A[ M ][ N ];  // declara matriz A de MxN
  ... // faz leitura e processamente de A

  imprime_matriz( A, M/2, N );       // imprime somente metade das linhas
  ...
```

Note que:

* podemos imprimir (usar) menos linha e menos colunas
* **nl** indica número de linhas a imprimir
* **nc** indica número de colunas a imprimir

Matrizes e funções

- Outro exemplo clássico com matrizes é quando a função recebe uma matriz e retorna um valor com base na análise dos valores da matriz. Por exemplo, o maior valor, a média dos valores, etc

```
// protótipos
float maiorValor_matriz( float X[][N] , int nl, int nc );
float media_matriz(float X[][N], int nl, int nc );
```

- OU SEJA:

  Assim como em vetores, as matrizes podem ter um tamanho máximo de linhas e colunas, mas o programador/usuário pode, em tempo de compilação/execução, usar um número menor caso necessário

- deve-se cuidar para evitar índices inválidos

Matrizes e funções

- Por exemplo, lin e col, são fornecidos em tempo de execução:

```
#define MAX_M   100
#define MAX_N   100
int main(){
    int matrix[MAX_M][MAX_N];
    int lin,col;
    cout << "Digite o tamanho da matriz: ";
    cin >> lin >> col;        // note que nesse caso se usuário
                              // digita mais de 100, teremos índices inválidos

    for (int l=0; l < lin; l++)
        for (int c=0; c < col; c++)
            cin >> matriz[l][c];
...
```

Matrizes e funções: um exemplo

- Quando for usar uma função para alterar valores em uma matriz, **verifique** se você não vai precisar dos valores iniciais em próximas etapas do programa. Neste caso, as funções devem ter uma matriz de entrada e uma de saída. Exemplos, eliminar valores inferiores a um limiar

```
void zeraLimiar_valores( float Y[][N], float X[][N], int nl, int nc, float limiar);
```

- A função produz a matrix Y copiando valores de X, mas zerando as células que tem valores menores que o limiar

```
// Obs.: o código da função fica como exercício
```

Matrizes e funções: um pouco mais de detalhes...

- Pode-se reutilizar funções de manipulação de vetores em programas com matrizes. Por exemplo, fazer a leitura da matriz usando a função a **ler_vetor**

```
void ler_vetor(int v[], int n) {
    for (int i=0; i<n ; i++)
        cin >> v[i];
}
int main(){
    int matriz[4][3];
    for (int l=0; l < 4; l++)          // são 4 linhas
        ler_vetor(matriz[l], 3);       // leitura por linha
    …                                  // cada linha é um vetor de 3 colunas
```

Matrizes e funções: um pouco mais de detalhes...

- Pode-se reutilizar funções de manipulação de vetores em programas com matrizes. Por exemplo, fazer a leitura da matriz usando a função a **ler_vetor**

```
void ler_vetor(int v[], int n) {
    for (int i=0; i<n ; i++)
        cin >> v[i];
}
int main(){
    int matriz[4][3];
    for (int l=0; l < 4; l++)          // são 4 linhas
        ler_vetor(matriz[l], 3);       // leitura por linha
    …                                  // cada linha é um vetor de 3 colunas
```

PORÉM:

- não poderíamos fazer isso com colunas **pois:**

  * **cada coluna NÃO é um vetor**

- **para ler e preencher uma coluna apenas teríamos de fazer uma nova função**

Operações com matrizes

- Escreva uma função que recebe duas matrizes quadradas de inteiros A e B, e **apenas imprime** a matriz soma S (onde cada elemento é a soma dos elementos de mesma posição na matriz A e B)

Operações com matrizes

- Escreva uma função que recebe duas **matrizes quadradas** de inteiros A e B, e **apenas imprime** a matriz soma S (onde cada elemento é a soma dos elementos de mesma posição na matriz A e B)

```
#define N  4
 void imprime_soma (int A[][N], int B[][N], int n )
 {
        for (int i=0; i<n; i++){          // Obs.: matriz quadrada nl == nc == n
            for (int j=0; j<n; j++)
                cout << A[i][j]+B[i][j] <<  “  “ ;
             cout  << endl;
}
```

Exercícios com matrizes

1) Escreva um programa que leia uma matriz de inteiros 10 x 10 e um valor X. O programa deverá fazer uma busca na matriz por este valor e informar a localização (linha x coluna) quando encontrar ou a mensagem de "valor não encontrado", caso contrário.

2) Escreva um programa para ler uma matriz A de inteiros de tamanho MxN (via #define) e gerar uma nova matriz B, onde cada elemento é soma dos valores do vizinho acima e abaixo na matriz A. Ou seja, a média do elemento(i,j) = elemento(i-1,j) + elemento (i+1,j). Cuidado com os limites da matriz, na primeira e última linhas (nestas posições haverá apenas 1 vizinho para cada elemento)

3) Escreva um programa que leia uma matriz A de inteiros e aplica uma função para calcular uma matriz B com valores reais, com a média em cada elemento (i,j) da matriz A e seus 4 vizinhos. Cuidado com os extremos da matriz

(elemento(i-1,j)+elemento(i+1,j)+elemento(i,j-1)+elemento(i,j+1))/4.

Bibliografia adicional:
favor consultar também

"Linguagem C++ - Notas de Aula". Revisão para C++ por Armando Luiz N. Delgado, a partir de material de Carmem S. Hara e Wagner N. Zola. 2018.
http://www.inf.ufpr.br/ci208/NotasAula/notas-14_Matrizes_ou_Arrays_Multi.html

Créditos: O conteúdo original deste documento é de autoria dos professores Luciano Silva e Wagner M. Nunan Zola, para uso na disciplina *Programação de Computadores* (CI208, CI180, CI183)

Compartilhe este documento de acordo com a licença abaixo