

Processando a Informação: um livro prático de programação independente de linguagem

Rogério Perino de Oliveira Neves

Francisco de Assis Zampirolli

EDUFABC

editora.ufabc.edu.br

Notas de Aulas inspiradas no livro

Utilizando a(s) Linguagem(ns) de Programação:

C

Exemplos adaptados para Correção Automática no Moodle+VPL

Francisco de Assis Zampirolli

24 de setembro de 2022

Sumário

1	Processando a Informação: Cap. 6: Matrizes	3
1.1	Sumário	3
1.2	Revisão do capítulo anterior (Vetores)	3
1.3	Introdução	4
1.4	Instanciando Matrizes	5
1.5	Acessando elementos de uma matriz	5
1.6	Formas de se percorrer uma matriz	6
1.7	Exemplo 01 - Ler/Escrever matriz	6
1.8	Arquivos	9
1.9	Exercícios	11
1.10	Atividades no Moodle+VPL	11
1.10.1	Entrada de Dados (cada linha contem um texto ou <i>string</i> com elementos da linha da matriz e vários espaços “”):	11
1.11	Revisão deste capítulo de Vetores	12

1 Processando a Informação: Cap. 6: Matrizes



Este caderno (Notebook) é parte complementar *online* do livro [Processando a Informação: um livro prático de programação independente de linguagem](#), que deve ser consultado no caso de dúvidas sobre os temas apresentados.

Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

1.1 Sumário

- Revisão do capítulo anterior
- Introdução
- Instanciando matrizes
- Acessando elementos de uma matriz
- Formas de percorrer uma matriz
- Aplicações usando matrizes
- Revisão deste capítulo
- Exercícios

1.2 Revisão do capítulo anterior (Vetores)

- Introdução > Vetores são estruturas para armazenar vários elementos de um mesmo tipo de dados em uma única variável.
- Trabalhando com vetores > Cada linguagem possui uma sintaxe própria para declarar e alocar vetores.
- Acessando elementos de um vetor > **ATENÇÃO** para não acessar uma posição do vetor não reservada/alocada, geralmente <0 e $\geq n$.
- Formas de percorrer um vetor > É possível varrer um vetor na forma *raster* e *anti-raster*, também usando diferentes passos, mas geralmente é $\text{passo}=1$.
- Modularização e vetores > Muito útil usar principalmente os módulos de `leiaVetor` e `escrevaVetor`, podendo ser reaproveitados em vários códigos.

- Tudo que foi visto no capítulo anterior de **Vetores** (ou estruturas **unidimensional** ou **1D**) se estende a estrutura de **Matrizes** (ou estruturas **bidimensional** ou **2D**)).

1.3 Introdução

- Existem situações onde é necessário estender a definição de vetor para mais de uma dimensão de dados.
- Por exemplo, uma forma muito utilizada de manipulação de dados é em tabelas.
- Como exemplo, para listar todos os alunos de uma turma e suas notas em uma disciplina,
 - as linhas da tabela podem armazenar a identificação dos alunos na primeira coluna,
 - nas colunas seguintes podem armazenar as notas da prova1, prova2, projeto e, numa última coluna, podem armazenar a nota final do aluno.
- Analogamente a um vetor, em uma matriz cada elemento possui apenas um dado.
- Além disso, na maioria das linguagens de programação, todos os elementos de uma matriz são de um mesmo tipo de dado.
- Um outro exemplo de matrizes muito usado, especialmente com a popularização dos dispositivos móveis como celulares e *tablets*, que comumente trazem câmeras digitais acopladas, ocorre no armazenamento e processamento digital de imagens.
- Mas antes das câmeras digitais, já havia *scanners* e digitalizadores, e uma imagem pode ser representada por uma **matriz** em um computador.
- Veja o conteúdo no **QRCode** (imagem=matriz) abaixo usando um aplicativo leitor de QRCode, disponível para *Smartphones* (teste nesta imagem).
- O conteúdo do **QRCode** direciona para uma página *web* mostrando uma imagem colorida.
- As imagens coloridas podem ser armazenada em uma estrutura de matriz **tridimensional**, onde cada dimensão armazena uma matriz para uma cor primária (RGB - *Red-Green-Blue* ou vermelho, verde e azul).



1.4 Instanciando Matrizes

- Nas linguagens MatLab e R, uma matriz é definida com elementos da posição (1,1) até a posição (L, C), onde L representa o número de linhas e C representa o número de colunas de uma matriz.
- Na maioria das linguagens de programação, no entanto, o índice começa no zero, sendo os elementos de uma matriz armazenados da posição (0,0) até (L-1,C-1).
- Nos exemplos a seguir são apresentados alguns exemplos de instanciação de matrizes em diferentes linguagens de programação.

1.5 Acessando elementos de uma matriz

- Uma matriz está pronta para se inserir elementos ou se alterar seus dados após instanciada e alocada na memória, em todas as suas dimensões.
- Veja uma forma de visualizar uma matriz 2D na Figura abaixo.

Matriz m[3,2]

		Índice i
Índice j →	0 1	
	5 6	↓ 0
	7 8	1
	9 7	2

- Essa figura ilustra uma matriz com 3 linhas e 2 colunas.
- Para visualizar seus elementos podemos usar os índices i e j,
 - com valores para as linhas $0 \leq i < 3$ e
 - para as colunas $0 \leq j < 2$.
- Analogamente ao vetor, mas com uma dimensão a mais, a matriz recebe valores para os seus elementos, conforme a seguinte estrutura em pseudocódigo:

```

Instanciar uma matriz m com 3 linhas e 2 colunas
m[0,0] = 5 # linha i=0
m[0,1] = 6
m[1,0] = 7 # linha i=1
m[1,1] = 8
m[2,0] = 9 # linha i=2
m[2,1] = 7

```

1.6 Formas de se percorrer uma matriz

- Analogamente ao vetor, uma matriz, após criada, possui tamanho fixo.
- Geralmente, para cada dimensão da matriz, é recomendado usar uma estrutura de repetição **para**, > com o objetivo de tornar o código mais compacto e genérico para matrizes de quaisquer dimensões.
- Além disso, é natural percorrer (ou varrer) a matriz
 - da linha $i=0$ até a linha $i<L$, onde L representa o número de linhas de uma matriz.
 - e da coluna $j=0$ até a coluna $j<C$.
- No exemplo a seguir em pseudocódigo, uma matriz m é criada com 6 elementos, sendo três linhas e 2 colunas, e os dois laços **para** inicializam todos os seus elementos com o valor 0.

Instanciar uma matriz m com 3 linhas e 2 colunas
inteiros $L=3$, $C=2$

Para cada i , de $i=0$; até $i<L$; passo $i=i+1$ faça
 Para cada j , de $j=0$; até $j<C$; passo $j=j+1$ faça
 $m[i,j] = 0$

- Existem várias formas de percorrer (ou varrer) uma matriz usando uma estrutura de repetição.
- Por exemplo, é possível percorrer uma matriz
 - do primeiro elemento $m[0,0]$ > (convencionando como canto superior esquerdo)
 - até o último elemento $m[L-1,C-1]$ > (convencionando como canto inferior direito da matriz),
 - linha por linha, ou coluna por coluna.
- Esse tipo de varredura é chamada **raster**.
- A varredura inversa é chamada **anti-raster**, do último elemento inferior direito, até o primeiro elemento superior esquerdo.

1.7 Exemplo 01 - Ler/Escriver matriz

- Analogamente ao que foi feito no capítulo anterior sobre vetores, onde alocamos os vetores em tempo de execução através de métodos, é possível usar modularização para melhorar a organização, manutenção e reaproveitamento de código.
- Aqui é apresentado um método `leiaMatriz` e `escrevaMatriz` genéricos
- Para entrada de dados, ou seja, inserir valores nos elementos alocados na memória para uma matriz.
- Além de saída de dados, para escrever a matriz, linha por linha.

Pseudocódigo Exemplo 01: Considere um algoritmo para: * Ler um inteiro L (linhas) representando o número de alunos, * Ler um inteiro C representando o número de avaliações. * Considere a primeira coluna o RA do aluno, assim $C=C+1$. * Criar uma matriz m com dimensões $L \times C$. * Ler todos os elementos da matriz. * Escrever todos os

elementos da matriz, formando a saída, linha por linha, por exemplo, para uma matriz com 2 alunos e 3 avaliações, escreva:

LISTA DE ALUNOS vs Avaliações:

1234 4 3 9

3456 6 4 8

```
Função inteiro m[][] leiaMatriz(inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            m[i,j] = leia("Digite um número inteiro:");
```

```
Função escrevaMatriz(inteiro m[][], inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            escreva(" ", m[i,j]);
        escreva("\n"); // pula linha
```

```
// PROGRAMA PRINCIPAL
```

```
// ENTRADAS
```

```
inteiro L = leia("Digite o numero de alunos:")
```

```
inteiro C = leia("Digite o numero de avaliações:")
```

```
C = C + 1 // a primeira coluna é o RA
```

```
Instanciar uma matriz m com L linhas e C colunas
```

```
m = leiaMatriz(L,C)
```

```
// PROCESSAMENTO: ?
```

```
// SAÍDA
```

```
escrevaMatriz(m)
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em **Casos para teste**, o seguinte texto (pode incluir mais casos):

case=caso1

input=2

3

1234

4

3

9

3456

6

4

8

output=

LISTA DE ALUNOS vs Avaliações:

1234 4 3 9

3456 6 4 8

```
[ ]: %%writefile cap6ex01.c
#include <stdio.h>
#include <malloc.h>

int ** leiaMatriz(int L, int C) {
    int **m = (int **)malloc(L*sizeof(int*));
    for (int i = 0; i < L; i++) {
        m[i] = (int *)malloc(C * sizeof(int)); // for each row allocate C
        ↪ints
        for (int j = 0; j < C; j++) {
            scanf("%d", &m[i][j]);
        }
    }
    return m;
}

// you must supply the number of rows
void free_matrix(int **m, int L)
{
    // first free each row
    for (int i = 0; i < L; i++) {
        free(m[i]);
    }

    // Eventually free the memory of the pointers to the rows
    free(m);
}

void escrevaMatriz(int **m, int L, int C) {
    for (int i = 0; i < L; i++) {
        for (int j = 0; j < C; j++) {
            printf("%d\t", m[i][j]);
        }
        printf("\n");
    }
}

int main(void) {
    // ENTRADA DE DADOS
    int L, C, **m; // variaveis de referência m
    printf("Digite o número de alunos: ");
    scanf("%d", &L);

    printf("Digite o número de avaliações: ");
    scanf("%d", &C);
```



```

C = C + 1; // a primeira coluna é o RA do aluno

printf("Digite os elementos da matriz");
m = leiaMatriz(L,C);

// PROCESSAMENTO ?

// SAÍDA DE DADOS
printf("\nLISTA DE ALUNOS vs Avaliações:\n");
printf("RA ");
for (int i = 0; i < C-1; i++) {
    printf("\t%d", (i+1));
}
printf("\n");
escrevaMatriz(m,L,C);
free_matrix(m,L); // liberar memória alocado com malloc
return 0;
}

```

```

[ ]: %%shell
gcc -Wall -std=c99 cap6ex01.c -o output2
./output2

```

1.8 Arquivos

Os arquivos armazenados em disco são considerados como uma sequência de caracteres e podem ser acessados em memória através de um ponteiro para o primeiro caracter do arquivo (texto ou binário), como segue:

```

FILE *fp;
fp = fopen("nomeArquivo.txt", "modo"); // modo é definido a seguir

```

Seguem as principais funções para manipular arquivos da biblioteca `stdio.h`:

Função	Descrição
<code>fopen()</code>	Abre arquivo
<code>fclose()</code>	Fecha arquivo
<code>putc()</code>	Escreve um caracter no arquivo
<code>fputc()</code>	Igual <code>putc()</code>
<code>getc()</code>	Lê um caracter do arquivo
<code>fgetc()</code>	Igual <code>getc()</code>
<code>fseek()</code>	Posiciona o arquivo em um determinado byte
<code>fprintf()</code>	Semelhante ao <code>printf()</code> , mas para arquivo
<code>fscanf()</code>	Semelhante ao <code>scanf()</code> , mas para arquivo
<code>feof()</code>	Verifica final de arquivo
<code>ferror()</code>	Verifica se ocorreu erro
<code>rewind()</code>	Posiciona o ponteiro para o início do arquivo

Função	Descrição
<code>remove()</code>	Apaga arquivo
<code>fflush()</code>	Descarrega o <i>buffer</i> do arquivo
<code>fgets()</code>	Obtém uma string do arquivo
<code>fread()</code>	Lê um bloco de dados do arquivo
<code>fwrite()</code>	Escreve um bloco de dados no arquivo
<code>ftell()</code>	Retorna a posição do ponteiro

Modo	Arquivo	Função
<code>r</code>	Texto	Leitura
<code>w</code>	Texto	Escrita em arquivo novo
<code>a</code>	Texto	Escrita em arquivo existente
<code>r+</code>	Texto	Leitura/Escrita
<code>w+</code>	Texto	Leitura/Escrita
<code>a+</code>	Texto	Leitura/Escrita
<code>rb</code>	Binário	Leitura
<code>wb</code>	Binário	Escrita
<code>ab</code>	Binário	Escrita
<code>r+b</code>	Binário	Leitura/Escrita
<code>w+b</code>	Binário	Leitura/Escrita
<code>a+b</code>	Binário	Leitura/Escrita

Exemplo para criar um arquivo texto.

```
[ ]: %%writefile cap6ex02.c
#include <stdio.h>

int main() {
    char* filename = "teste.txt";

    FILE* fp = fopen(filename, "w"); // Cria arquivo para escrita
    if (fp == NULL) {
        printf("Erro ao abrir o arquivo: %s\n", filename);
        return -1;
    }

    for (int i = 0; i < 10; i++)
        fprintf(fp, "Linha %02d\n", i + 1); // Escrever algo no arquivo

    fclose(fp); // fecha arquivo

    return 0;
}
```

```
[ ]: %%shell
gcc -Wall -std=c99 cap6ex02.c -o output2
```

```
./output2
cat teste.txt
```

Exemplo para ler um arquivo texto.

```
[ ]: %%writefile cap6ex03.c
#include <stdio.h>

int main() {
    char* filename = "teste.txt", ch;

    FILE* fp;
    fp = fopen(filename, "r"); // Cria arquivo para escrita
    if (fp == NULL) {
        printf("Erro ao abrir o arquivo: %s\n", filename);
        return -1;
    }

    while ((ch = fgetc(fp)) != EOF) { // Lê arquivo
        printf("%c", ch); // caracter por caracter
    }

    fclose(fp); // fecha arquivo

    return 0;
}
```

```
[ ]: %%shell
gcc -Wall -std=c99 cap6ex03.c -o output3
./output3
```

1.9 Exercícios

Ver notebook Colab nos arquivos `cap6.partX.lab.*.ipynb` ($X \in [2,3,4,5]$ e $*$ é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, organizadas em subpastas contidas em "gen", na pasta do Google Drive [colabs](#).

1.10 Atividades no Moodle+VPL

Algumas atividades no Moodle+VPL pedem como entradas matrizes de inteiros (ou reais), **armazenados em várias linhas**. Exemplo de entrada a ser lida:

1.10.1 Entrada de Dados (cada linha contem um texto ou *string* com elementos da linha da matriz e vários espaços “”):

```
0 9 3 6 9 8 4 5 4
8 1 2 3 5 2 9 9 6
4 1 1 0 9 9 8 2 7
```

```
5 2 8 4 6 6 0 8 0
4 7 6 4 3 9 3 3 5
2 6 0 4 0 7 5 5 2
9 8 4 8 4 7 1 4 3
```

Para não ter que incluir várias entradas inteiras, a melhor solução é fazer um método de leitura, passando como argumento um texto (*string*) com várias linhas. O final de cada linha é definido por `\n` e o final da matriz deve ser uma linha em branco com apenas `\n`. Esse método deve retornar a matriz.

1.11 Revisão deste capítulo de Vetores

- Introdução
- Instanciando matrizes
- Acessando elementos de uma matriz
- Formas de percorrer uma matriz
- Aplicações usando matrizes
- Exercícios
- Revisão deste capítulo de Vetores