

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

3 de setembro de 2022

Sumário

1	Processando a Informação: Cap. 5: Vetores	3
1.1	Sumário	3
1.2	Revisão do capítulo anterior (Estruturas de Repetição - Laços)	3
1.3	Introdução	4
1.4	Trabalhando com vetores	4
1.5	Exemplo 01 - Ler/Escriver vetor	5
1.6	Formas de Percorrer um Vetor	7
1.6.1	Percorrer um vetor com o laço para	7
1.6.2	Percorrer um vetor com o laço enquanto	7
1.6.3	Outras formas de percorrer um vetor	8
1.7	Modularização e Vetores	8
1.8	Exemplo 02 - Aplicação simples 1 usando vetor	9
1.9	Exemplo 03 - Aplicação simples 2 usando vetor	11
1.10	Exemplo 04 - Aplicação “dilata” vetor	13
1.11	Exercícios	16
1.12	Atividades no Moodle+VPL	16
1.12.1	Entrada de Dados (cada linha contem um texto ou <i>string</i> incluindo os elementos do vetor e vários espaços “ ”):	16
1.12.2	Entrada de Dados (a linha contem um texto ou <i>string</i>):	16
1.13	Revisão deste capítulo de Vetores	17

1 Processando a Informação: Cap. 5: Vetores



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
- Trabalhando com vetores
- Acessando elementos de um vetor
- Formas de percorrer um vetor
- Modularização e vetores
- Revisão deste capítulo
- Exercícios

1.2 Revisão do capítulo anterior (Estruturas de Repetição - Laços)

- Estruturas de repetição (laços) devem ser utilizadas quando existem instruções que se repetem e podem ser de vários tipos, dependendo da linguagem utilizada, por exemplo:
 - `do-while` (não aceita em Python)
 - `while` (a maioria)
 - `for` (a maioria)
 - `repeat` (R)
- O uso de laços depende da lógica a ser implementada e da linguagem utilizada. Mas, se tiver um número fixo de iterações, geralmente se usa `for`.
- Validação de dados utilizando laços:
 - Incluir um laço para verificar o valor lido.

- Interrupção da execução em laços:
 - Depende da linguagem, algumas possibilidades:
 - * **break** - interrompe o laço
 - * **continue** - não executa o final do laço
 - * **exit** - aborta o laço e o programa!
- Neste capítulo iremos abordar a estrutura de **Vetor**, incluindo os conceitos apresentados nos capítulos anteriores.

1.3 Introdução

- Um **vetor** em programação é formado por um **conjunto de n variáveis de um mesmo tipo** de dado, sendo n obrigatoriamente maior que zero.
- Essas variáveis (ou elementos) são identificadas e acessadas por um **nome** e um **índice**.
- Na maioria das linguagens de programação, o **índice** recebe valores de 0 (**primeiro elemento**) até $n - 1$ (**último elemento**).
- As variáveis de um vetor são armazenadas em posições consecutivas de memória.

1.4 Trabalhando com vetores

- Quando um vetor é criado, é instanciada uma variável do tipo vetor (Em algumas linguagens de programação, sendo necessário se definir um nome, o tipo de dado e o tamanho do vetor).
- Na linguagem C, por exemplo, a definição do tamanho de um vetor ocorre antes de compilar o programa.
 - Ou seja, o programador deverá informar no código a quantidade de memória a ser reservada para o vetor, através de um número inteiro positivo ou constante que o contenha, especificando o número de elementos do mesmo tipo que a memória reservada irá comportar.
 - Neste caso, uma variável não pode ser usada.
- Já na linguagem Java, é possível alocar memória em tempo de execução do programa.
 - Por exemplo, é possível criar um vetor para armazenar as notas de uma turma de alunos, onde o número de alunos é uma variável a ser lida durante a execução do programa.
- Em muitas linguagens, como Java e C, o processo de criar um vetor ocorre em dois passos distintos:
 1. primeiro se cria uma variável de referência para o vetor;
 2. em seguida se reserva a memória para um dado número de elementos do mesmo tipo.
- No exemplo da Figura abaixo,
 - A criação da variável de referência de um vetor **v** define apenas a posição de memória em hexadecimal **0A**, onde será armazenado o seu primeiro elemento.
 - * Neste exemplo, **V[0] = -128**
 - * A quantidade de **bytes** por elemento vai depender do tipo de dado armazenado. Neste exemplo cada elemento ocupa um **byte**.

- O segundo passo da criação de um vetor é reservar (ou alocar) memória para todos os seus elementos (dependendo da linguagem). > essa alocação de memória pode ocorrer em tempo de execução, com ocorre em Python!

	Conteúdo do vetor	Posição na RAM
	v	
		09
v[0]	-128	0A
v[1]	0	0B
v[2]	6	0C
v[3]	98	0D
v[4]	127	0E
v[5]	4	0F
		10

- Note que a variável **v** acima agora se refere a uma posição de memória e não a um elemento de dado.
- Os elementos devem ser acessados através do índice, um por vez.
- Isso pode requerer uso de laços para leitura e impressão dos seus elementos, dependendo da linguagem de programação utilizada. > O índice de vetor é sempre um número inteiro.

1.5 Exemplo 01 - Ler/Escriver vetor

Pseudocódigo

```

Instanciar o vetor v1 de inteiro com tamanho 5 // ou
vetor v2 = vetor de inteiros com 100 elementos // ou ainda
vetor v3 = inteiro(10)
v1 = {4,1,10,2,3} // atribuindo valores a v1

```

Exemplo 01: Ler uma lista com **n** alunos com **RA** e **Nome** e escrever formatando a saída como no exemplo:

LISTA DE ALUNOS

Número	RA	Nome
1	2134	Maria Campos
2	346	João Silva

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
2345
9870
Maria Campos
João Silva
output=
LISTA DE ALUNOS
Número RA      Nota
1      2345    Maria Campos
2      9870    João Silva
```

```
[ ]: %%writefile cap05exem01.c
#include <stdio.h>

int main(void) {

    // ENTRADA DE DADOS
    int n, ra[100];      // aloca 100 ra's
    char nome[100][40]; // aloca 100 nomes de até 40 caracteres cada

    printf("Digite o numero de alunos: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("RA: ");
        scanf("%d", &ra[i]);
        printf("Nome: ");
        scanf("%s", nome[i]); // ATENÇÃO: NÃO ACEITA NOME COM ESPAÇOS
    }
    // PROCESSAMENTO ?
    // SAÍDA
    printf("LISTA DE ALUNOS\nNúmero\t RA\t Nome\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t %d\t %s\n", i+1, ra[i], nome[i]);
    }
    return 0;
}
```

```
[ ]: %%shell
gcc cap05exem01.c -o output2
./output2
```

1.6 Formas de Percorrer um Vetor

1.6.1 Percorrer um vetor com o laço para

- Como um vetor, após criado (alocado e com elementos), tem tamanho fixo (sempre com número de elementos $n > 0$), geralmente é indicado usar estruturas de repetição do tipo **para** (**for**), de forma a percorrer todos os elementos do vetor
 - assim tornando o código genérico para um vetor de qualquer tamanho n .
- Além disso, é natural percorrer (ou varrer) o vetor da posição $i=0$ até a posição $i < n$
 - O índice i assume automaticamente os valores 0, 1, 2, até $n-1$, em cada iteração do laço.
 - Em algumas linguagens, como Matlab e R, o índice assume valores entre 1 até n .
- No exemplo a seguir em pseudocódigo, um vetor v é criado com 6 posições e o laço **para** inicializa todos os seus elementos com o valor 0.
- A forma natural de percorrer os elementos de um vetor (ou **varrendo** o vetor) é na ordem **raster**, do primeiro até o último elemento.
- A vantagem de se usar a estrutura **para** para varrer um vetor é permitir embutir na própria sintaxe (linha) da instrução:
 - declaração e inicialização do índice,
 - incremento e
 - condição de saída do laço.

Pseudocódigo

Instanciar um vetor v de inteiro com 6 elementos ($n=6$)
 para cada índice i , de $i=0$; até $i < n$; passo $i=i+1$ faça
 $v[i] = 0$

1.6.2 Percorrer um vetor com o laço enquanto

- Se o programador preferir, ou o problema a ser resolvido exigir, o vetor pode alternativamente ser percorrido utilizando-se uma estrutura de repetição do tipo **enquanto**:

Pseudocódigo

```
n=6
vetor v de inteiros com n elementos
inteiro i=0
enquanto i<n faça {
    v[i] = 0
    i=i+1
}
```

- O código usando a estrutura de repetição **enquanto** produz o mesmo resultado que usando com **para**, porém, usando mais instruções:
 - a criação do contador i
 - o incremento $i=i+1$.
- Essas duas instruções ficam encapsuladas na própria instrução **para**.

1.6.3 Outras formas de percorrer um vetor

- Dependendo do problema a ser tratado, existem várias formas de se varrer um vetor usando estruturas de repetição para acessar cada elemento.
- Por exemplo, é possível percorrer o vetor do último elemento para o primeiro (essa varredura é chamada de **anti-raster**):

Pseudocódigo: percorrer um vetor usando para na ordem inversa (anti-raster).

```
Instanciar um vetor v de inteiro com 6 elementos (n=6)
para cada indice i, de i=n-1; até i<=0; passo i=i-1 faça
    v[i] = 0
```

- Também, é possível percorrer somente alguns elementos do vetor, como para acessar os elementos que estão nos índices pares ou ímpares, diferenciadamente:

Pseudocódigo: percorrer um vetor usando para, com passo 2.

```
vetor v de inteiro com 6 elementos
n=6
para cada indice i, de i=0; até i<n-1; passo i=i+2 faça {
    v[i] = 0
    v[i+1] = 1
}
```

1.7 Modularização e Vetores

- Como uma boa prática de programação, além de comentar os códigos e organizar com tabulação, como apresentado nos exemplos deste livro,
 - também é recomendável modularizar o código usando métodos, como apresentado no Capítulo 2.
- Todo sistema computadorizado de informação possui três partes bem definidas, agrupadas em
 - módulo(s) de entrada,
 - módulo(s) de processamento e
 - módulo(s) de saída.
- Ao manipular informações armazenadas em vetores, é natural criar também pelo menos três módulos ou métodos:
 - `leiaVetor`,
 - `processaVetor` e
 - `escrevaVetor`, > satisfazendo essa definição de sistema de informação.

Para a reutilização de código, os módulos `leiaVetor` e `escrevaVetor` poderão ser muito reutilizados para resolver outros problemas de manipulação de vetores.

Pseudocódigo: método para ler um vetor de inteiro tamanho n.

```
método leiaVetor(inteiro n): retorna vetor de inteiro v[]
    vetor v de inteiros com n elementos
```



```

para cada índice i, de i=0; até i<n; passo i=i+1 faça
    v[i] = leia("Entre com o elemento " + i + ":");
retorne v

```

Pseudocódigo: método para escrever um vetor de inteiro tamanho n.

```

método escrevaVetor(inteiro v[], inteiro n):
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        escreva(" " + v[i]);

```

Pseudocódigo: aplicação de vetor usando módulos.

```

// Instâncias e Atribuições
inteiro n = leia("Digite o tamanho do vetor:");

// ENTRADA
inteiro v1[] = leiaVetor(n)

// PROCESSAMENTO
// ?

// SAÍDA
escrevaVetor(v2, n)

```

1.8 Exemplo 02 - Aplicação simples 1 usando vetor

Considere um algoritmo para ler a quantidade n de alunos de uma turma. Ler uma lista com n RA's de alunos. Em seguida, ler também uma lista com n notas de alunos. Como saída do algoritmo, escrever a seguinte saída.

LISTA DE ALUNOS

Número	RA	Nota
1	2134	9
2	346	7

Utilizar os métodos `leiaVetor` e `escrevaVetor`, se necessários.

Pseudocódigo

```

// ENTRADAS:
// instâncias e atribuições
real media, somador = 0
inteiro contador = 0

// leitura do número de alunos = tamanho do vetor
inteiro n = leia("Digite o número de alunos:");

inteiro ras[] = leiaVetor(n)
inteiro notas[] = leiaVetor(n)

// PROCESSAMENTO

```

```
// ?
```

```
// SAÍDAS:
```

```
escreva("LISTA DE ALUNOS")
```

```
escreva("Número RA      Nota")
```

```
para cada indice i, de i=0; até i<n; passo i=i+1 faça
```

```
    escreva(i+1, "\t", ras[i], "\t", notas[i])
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em **Casos para teste**, o seguinte texto (pode incluir mais casos):

```
case=caso1
```

```
input=2
```

```
3456
```

```
2345
```

```
5
```

```
2
```

```
output=
```

```
LISTA DE ALUNOS
```

```
Número   RA  Nota
```

```
1       3456   5
```

```
2       2345   2
```

```
[ ]: %%writefile cap05exem02.c
#include <stdio.h>
#include<malloc.h>

int * leiaVetor(int n) {
    int *v= malloc(sizeof(n));
    for (int i = 0; i < n; i++) {
        scanf("%d", &v[i]);
    }
    return v;
}

int main(void) {

    // ENTRADA DE DADOS
    int n, *ras, *notas;    // variaveis de referência ras e notas

    printf("Digite o numero de alunos: ");
    scanf("%d", &n);

    printf("RAs: ");
    ras = leiaVetor(n);

    printf("Notas: ");
```

```

notas = leiaVetor(n);

// PROCESSAMENTO ?
// SAÍDA
printf("LISTA DE ALUNOS\nNúmero\t RA\t Nota\n");
for (int i = 0; i < n; i++) {
    printf("%d\t %d\t %d\n", i+1, ras[i], notas[i]);
}
free(ras); // liberar memória alocado com malloc
free(notas);
return 0;
}

```

```

[ ]: %%shell
gcc cap05exem02.c -o output2
./output2

```

1.9 Exemplo 03 - Aplicação simples 2 usando vetor

Considere um algoritmo para ler a quantidade n de alunos de uma turma. Ler também uma lista com n notas de alunos. Como saída do algoritmo, escrever a média da turma e quantos alunos ficaram acima da média.

Pseudocódigo

```

// ENTRADAS:
// instâncias e atribuições
real media, somador = 0
inteiro contador = 0

// leitura do número de alunos = tamanho do vetor
inteiro n = leia("Digite o número de alunos:");

inteiro ras[] = leiaVetor(n)
real notas[] = leiaVetor(n)

// PROCESSAMENTO: soma, média e contador
para cada índice i, de i=0; até i<n; passo i=i+1 faça
    somador = somador + notas[i] // soma
media = somador / n // média
para cada índice i, de i=0; até i<n; passo i=i+1 faça
    se ( notas[i] >= media ) // conta alunos>=media
        contador = contador + 1

// SAÍDAS:
escreva("Média da turma=" + media) // Saída
escreva("Alunos acima da média: " + contador)
escreva("LISTA DE ALUNOS ACIMA DA MÉDIA")

```

```

escreva("RA\t Nota")
para cada indice i, de i=0; até i<n; passo i=i+1 faça
    se (nota[i] >= media)
        escreva(ras[i], "\t", notas[i])

```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 03, basta incluir em **Casos para teste**, o seguinte texto (pode incluir mais casos):

```

case=caso1
input=2
233245
234534
9
4
output=
Média da turma = 6.5
Número de alunos acima da média = 1
LISTA DE ALUNOS ACIMA DA MÉDIA
RA      Nota
233245  9

```

```

[ ]: %%writefile cap05exem03.c
#include <stdio.h>
#include <malloc.h>

int * leiaVetor(int n) {
    int *v= malloc(sizeof(n));
    for (int i = 0; i < n; i++) {
        scanf("%d", &v[i]);
    }
    return v;
}

int main(void) {

    // ENTRADA DE DADOS
    int n, *ras, *notas;    // variaveis de referência ras e notas
    float media,somador = 0.0;
    int contador = 0;

    printf("Digite o numero de alunos: ");
    scanf("%d", &n);

    printf("RAs: ");
    ras = leiaVetor(n);

    printf("Notas: ");

```

```
notas = leiaVetor(n);

// PROCESSAMENTO: soma, média e contador
for (int i = 0; i < n; i++) {
    somador = somador + notas[i];    // soma
}
media = (float) somador / n;        // média

for (int i = 0; i < n; i++) {
    if (notas[i] >= media) {        // conta alunos>=media
        contador = contador + 1;
    }
}

// SAÍDA DE DADOS
printf("Média da turma = %.1f\n",media);
printf("Número de alunos acima da média = %d\n",contador);
printf("LISTA DE ALUNOS ACIMA DA MÉDIA\nRA\t Nota\n");
for (int i = 0; i < n; i++) {
    if (notas[i] >= media) {        // conta alunos>=media
        printf("%d\t %d\n",ras[i],notas[i]);
    }
}
free(ras); // liberar memória alocado com malloc
free(notas);
return 0;
}
```

```
[ ]: %%shell
gcc cap05exem03.c -o output2
./output2
```

1.10 Exemplo 04 - Aplicação “dilata” vetor

- O programador deve tomar cuidado ao acessar os elementos de um vetor para **não ultrapassar os seus limites alocados previamente**, ou seja,
 - para um vetor v de tamanho n , índices i com $i < 0$ ou $i \geq n$ não existem.
- O exemplo a seguir ilustra este problema.
- Considere o problema de “**dilatar**” um vetor.
 - O objetivo é criar um vetor $v1$ de inteiros com n posições e um outro vetor $v2$ de inteiros também com n posições.
 - Cada posição i de $v2$ armazena o cálculo do **máximo** entre cada elemento i em $v1$ e seus vizinhos:
 1. à esquerda $v1[i-1]$,
 2. do próprio elemento $v1[i]$ e
 3. do seu vizinho à direita $v1[i+1]$.

- Veja uma ilustração da operação de dilatação na Figura abaixo, onde **v1** é o vetor de entrada e **v2** é o vetor de saída, contendo a “dilatação” de **v1**, seguido do código para resolver este problema proposto.

Conteúdo do vetor v1	Conteúdo do vetor v2
-128	0
0	6
6	98
98	127
127	127
4	127

Pseudocódigo

```

método escrevaVetor(inteiro v[], inteiro n):
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        escreva(" " + v[i]);

método leiaVetor(inteiro n): retorna vetor de inteiro v[]
    vetor v de inteiros com n elementos
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        v[i] = leia("Entre com o elemento " + i + ":");
    retorne v

// Inicializações
inteiro max=0
// ENTRADA
inteiro n = leia("Digite o tamanho do vetor:");
vetores v1 e v2 de inteiros com n elementos

inteiro v1[] = leiaVetor(n)

// PROCESSAMENTO
para cada índice i, de i=0; até i<n; passo i=i+1 faça
    max = v[i]
    se i-1 >= 0 e max < v1[i-1] faça

```

```

        max = v1[i-1]
    se i+1 < n e max < v1[i+1] faça
        max = v1[i+1]
    v2[i] = max

```

```

// SAÍDA
escrevaVetor(v2)

```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 04, basta incluir em **Casos para teste**, o seguinte texto (pode incluir mais casos):

```

case=caso1
input=6
-128
0
6
98
127
4
output=v2:
0
6
98
127
127
127

```

```

[ ]: %%writefile cap05exem04.c
#include <stdio.h>
#include <malloc.h>

int * leiaVetor(int n) {
    int *v = malloc(n*sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &v[i]);
    }
    return v;
}

void escrevaVetor(int *v, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d\n", v[i]);
    }
}

int main(void) {
    // ENTRADA DE DADOS
    int n, *v1; // variaveis de referência v1
    printf("Digite o tamanho do vetor: ");

```

```

scanf("%d", &n);
int *v2 = malloc(100*sizeof(int));
printf("Digite os elementos: ");
v1 = leiaVetor(n);
// PROCESSAMENTO
for (int i = 0; i < n; i++) {
    int max = v1[i];
    if (i-1 >= 0 && max < v1[i-1])
        max = v1[i-1];
    if (i+1 < n && max < v1[i+1])
        max = v1[i+1];
    v2[i] = max;
}
// SAÍDA:
printf("\nv2:\n");
escrevaVetor(v2,n);
free(v1); // liberar memória alocado com malloc
free(v2);
return 0;
}

```

```

[ ]: %%shell
gcc cap05exem04.c -o output2
./output2

```

1.11 Exercícios

Ver notebook Colab nos arquivos `cap05.partX.lab.*.ipynb` ($X \in [2,3]$ e $*$ é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, organizadas em subpastas contidas de "gen", na pasta do Google Drive [colabs](#).

1.12 Atividades no Moodle+VPL

Algumas atividades no Moodle+VPL pedem como entradas vetores de inteiros (ou reais), armazenados em uma única linha. Exemplo de entrada a ser lida:

1.12.1 Entrada de Dados (cada linha contem um texto ou *string* incluindo os elementos do vetor e vários espaços “”):

```

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

```

Para não ter que incluir várias entradas inteiras, a melhor solução é fazer um método de leitura, passando como argumento um texto (*string*) referente a cada linha. Esse método deve retornar o vetor.

1.12.2 Entrada de Dados (a linha contem um texto ou *string*):

A UFABC completou 15 anos!

Como contar as vogais de um texto?

1.13 Revisão deste capítulo de 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 $>=n$.
- Formas de percorrer um vetor > É possível varrer um vetor na forma *raster* e *anti-raster*, também usando diferentes passos, mas geralmente é passo=1.
- Modularização e vetores > Muito útil usar principalmente os módulos de `leiaVetor` e `escrevaVetor`, podendo ser reaproveitados em vários códigos.
- Exercícios
- Revisão deste capítulo de Vetores