Marisangila Alves, MSc
marisangila.alves@udesc.com
marisangila.com.br

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA
JOINVILLE
CENTRO DE CIÊNCIAS TECNOLÓGICAS

UDESC
Universidade do Estado de Santa Catarina

2025/1

Linguagem de Programação

Linguagem de Programação C
Vetor

Sumário

Definição

Definição I

> *É um grupo de locais da memória relacionados pelo fato de que todostêm o mesmo nome e o mesmo tipo. Para fazer referência a um determinado local ouelemento no array, especificamos o nome do array e o número da posiçãodaqueleelemento no array.*
>
> (Deitel; Deitel, 2011)

O que isso quer dizer?

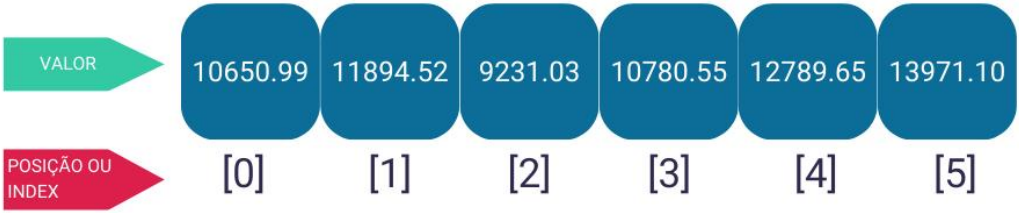
- Até o momento, utilizamos variáveis para armazenar dados, que permitiam armazenar apenas um único valor de um único tipo.
- No entanto, é possível usar uma estrutura chamada vetor, que permiti armazenar múltiplos valores de um mesmo tipo.

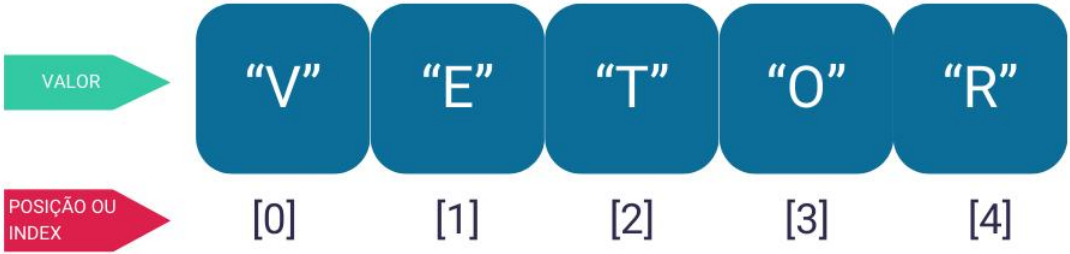
- Sendo assim, um vetor é uma variável composta e homogênea que permite armazenar um **conjunto** de valores relacionados em uma **única variável**.
- O vetor pode também pode ser chamado de *array* ou matriz unidimensional.
- Os valores são organizados em posições sequenciais, cada uma identificada por um índice numérico.
- Podemos chamar cada uma desses valores de elementos;
- Vetores são estruturas homogêneas.

```
1 tipo nome_vetor[quantidade_posicoes];
```

Código 1: Estrutura básica.


VALOR
20
21
18
19
20
POSIÇÃO OU INDEX
[0]
[1]
[2]
[3]
[4]


VALOR
10650.99
11894.52
9231.03
10780.55
12789.65
13971.10
POSIÇÃO OU INDEX
[0]
[1]
[2]
[3]
[4]
[5]


VALOR
“V”
“E”
“T”
“O”
“R”
POSIÇÃO OU INDEX
[0]
[1]
[2]
[3]
[4]

Declaração

```c
int main(){
    int idades[5];
    float lucro_semestral[6];
    char nome_animal[10];
    return 0;
}
```

Código 2: Declaração.

```c
#define TAMANHO 5
int main(){
    int idades[TAMANHO];
}
```

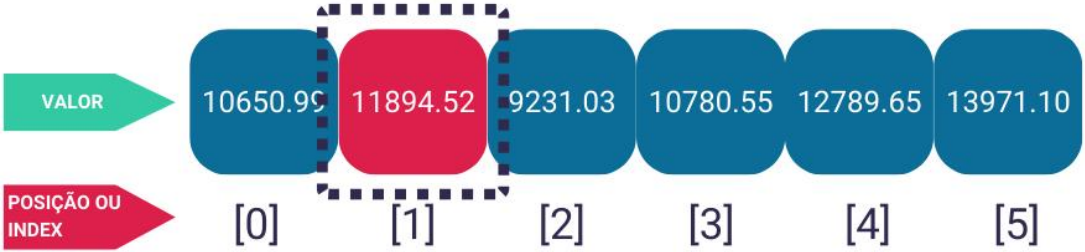
Código 3: Definindo tamanho usando constante.

Atenção!

Não se deve usar variáveis para definir o tamanho de vetores estáticos porque o tamanho do vetor é alocado em tempo de compilação, e alterar a variável após a declaração não redimensiona o vetor, o que pode levar a acessos fora dos limites da memória (*buffer overflow*).

Atribuição

Com acessar a segunda posição no vetor?



Atenção!

Lembrando que a primeira posição do vetor é acessada pelo índice zero[a].

[a] É importante observar a diferença entre o "segundo elemento do vetor"e o "elemento número dois do vetor". Como os índices dos vetores iniciam em 0, o "segundo elemento do vetor"tem índice 1, ao passo que o "elemento número dois do vetor"tem índice 2 e é, na realidade, o terceiro elemento do vetor. Esse fato é uma fonte de erros de "diferença- um".

```c
int main(){
    int idades[5];
    float lucro_semestral[6];
    char nome_animal[10];

    idades[0] = 10;
    lucro_semestral[2] = 9231.03;
    nome_animal[0] = 'C';
    return 0;
}
```

Código 4: Atribuição.

Definição

```c
int main(){
    int idades[5] = {20, 21, 18, 19, 20};
    float lucro_semestral[6] = {10650.99, 11894.52, 9231.03,
    ↪  10780.55, 12789.65, 13971.10};
    char nome_animal[10] = "Coelho";
    return 0;
}
```

Código 5: Definição.

Atenção!

Se o tamanho do vetor for omitido de uma declaração com uma lista de inicializadores, o número de elementos do vetor será o número de elementos da lista de inicializadores. Por exemplo: `int idades[] = 20,21,18`, o vetor tera três elementos.

Manipulação

```c
#include <stdio.h>
int main(){
    int idades[5];
    idades[0] = 20;
    idades[1] = 19;
    idades[2] = 18;
    idades[3] = 21;
    idades[4] = 18;
    printf("%d\n", idades[0]);
    printf("%d\n", idades[1]);
    printf("%d\n", idades[2]);
    printf("%d\n", idades[3]);
    printf("%d\n", idades[4]);
    return 0;
}
```

Código 6: Exemplo idades.

```
1 idades[1 + 2] = 25;
```

Soma o índice.

```
1 idades[1] + idades[2] = 25;
```

Soma o valor do elemento.

Código 7: Ordem de operação nos colchetes.


VIDA DE PROGRAMADOR
.COM.BR
real historia;
string sender;
sender = "Carlos";
#1211
ENTÃO, ALONSO, É ASSIM QUE FUNCIONA O FOR EM C... AGORA DECLARA AÍ UM VETOR COM 4 POSIÇÕES E INICIALIZA OS VALORES DE 1 A 4...
v[0] = 1;
v[1] = 2;
v[2] = 3;
v[3] = 4;
NO DONUTS FOR YOU
AFF, ALONSO! E SE O VETOR TIVER 100 POSIÇÕES?
COPIA E COLA...
NO DONUTS FOR YOU
E SE TIVER UM MILHÃO DE POSIÇÕES...
HMMMM... AÍ COPIA E COLA BASTANTE...
KABOOM!

```c
#include <stdio.h>
int main(){
    int idades[100];
    for (int i = 0; i < 100; i++)
    {
        printf("Qual idade?\n");
        scanf("%d", &idades[i]);
    }
    return 0;
}
```

Código 8: Exemplo idades entrada.

```c
#include <stdio.h>
int main(){
    int idades[100];
    for (int i = 0; i < 100; i++)
    {
        printf("%d\n", idades[i]);
    }
    return 0;
}
```

Código 9: Exemplo idades saída.

```c
#include <stdio.h>
int main(){
    int idades[100];
    for (int i = 0; i < 100; i++)
    {
        printf("Qual idade?\n");
        scanf("%d", &idades[i]);
    }
    for (int i = 0; i < 100; i++)
    {
        printf("%d\n", idades[i]);
    }
    return 0;
}
```

Código 10: Exemplo idades.

```c
#include <stdio.h>
#include <time.h>
int main(){
    int numero[100];
    srand(time(NULL));
    for (int i = 0; i < 100; i++)
    {
        printf("Qual numero?\n");
        numero[i] = (rand() % 100) + 1;
    }
    for (int i = 0; i < 100; i++)
    {
        printf("%d\n", numero[i]);
    }
    return 0;
}
```

Código 11: Atribuição de valores randômicos.

Exemplos

```c
#include <stdio.h>
int main() {
    float notas[5] = {7.5, 8.0, 9.2, 6.5, 7.8};
    float soma = 0.0;
    float media;
    for (int i = 0; i < 5; i++) {
        soma += notas[i];
    }
    media = soma / 5;
    printf("Média das notas: %.2f\n", media);
    return 0;
}
```

Código 12: Exemplo média de notas.

```c
#include <stdio.h>
int main() {
    int numeros[5] = {10, 20, 30, 25, 15};
    int maior = -99;
    for (int i = 1; i < 5; i++) {
        if (numeros[i] > maior) {
            maior = numeros[i];
        }
    }
    printf("O maior valor no vetor é: %d\n", maior);
    return 0;
}
```

Código 13: Exemplo maior valor.

```c
#include <stdio.h>
int main() {
    int a[5] = {1, 2, 3, 4, 5};
    int b[5] = {5, 4, 3, 2, 1};
    int c[5];
    for (int i = 0; i < 5; i++) {
        c[i] = a[i] + b[i];
    }
    printf("Soma dos vetores:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", c[i]);
    }
    return 0;
}
```

Código 14: Exemplo soma de vetores.

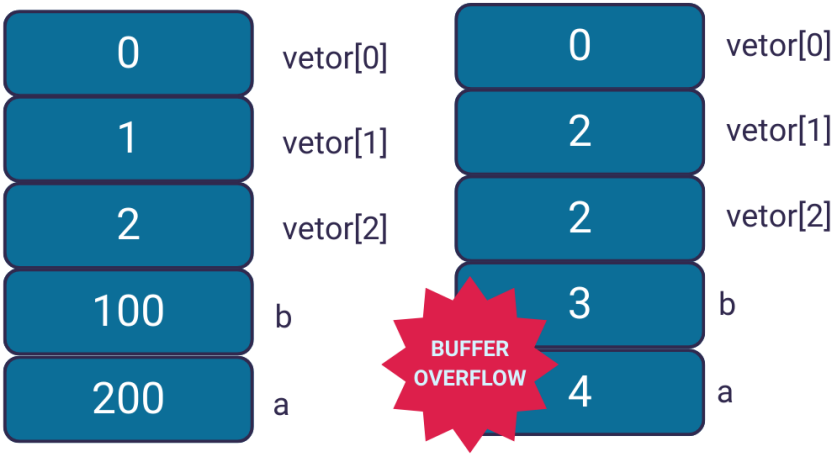
```c
#include <stdio.h>
#define TAMANHO_VETOR 5
int main() {
    int vetor[TAMANHO_VETOR] = {10, 20, 30, 40, 50};
    for (int i = 0; i < TAMANHO_VETOR; i++) {
        printf("%d",vetor[i]);
    }
    return 0;
}
```

Código 15: Exemplo tamanho definido como constante.

Buffer Overflow

```c
#include <stdio.h>
int main()
{
    int a = 100, b = 200;
    int vetor[3];
    for(int i = 0; i < 5; i++)
        vetor[i] = i;
    printf("%d - %d\n", a, b);
    return 0;
}
```

Código 16: Comportamento indefinido.


0
vetor[0]
1
vetor[1]
2
vetor[2]
100
b
200
a
0
vetor[0]
2
vetor[1]
2
vetor[2]
3
b
4
a
BUFFER
OVERFLOW

- **Por que não é possível usar tamanho dinâmico em vetores (sem alocação dinâmica)?**
  - O tamanho de vetores deve ser conhecido em tempo de compilação.
  - Vetores são alocados estaticamente, ou seja, o tamanho é fixo durante a execução.
  - Para vetores de tamanho variável em tempo de execução, é necessário usar alocação dinâmica (como `malloc`).
- **O que é Buffer Overflow?**
  - Ocorre quando dados são gravados além do limite de um buffer (vetor).
  - Sobrescrita de memória pode corromper variáveis ou dados importantes.
  - Pode causar comportamento imprevisível.
  - Pode resultar em Segmentation Fault se ultrapassar o espaço reservado para o processo.

Ordenação

```c
#include <stdio.h>
int main() {
    int vetor[] = {5, 2, 1, 9, 6};
    int aux, trocou;
    do {
        trocou = 0;
        for (int i = 0; i < 4; i++)
            if (vetor[i] > vetor[i + 1]) {
                aux = vetor[i];
                vetor[i] = vetor[i + 1];
                vetor[i + 1] = aux;
                trocou = 1;
            }
    } while (trocou != 0);
    return 0;
}
```

Código 17: Ordenação de um vetor.

- **Bubble Sort** é um algoritmo de ordenação simples.
- Ele percorre repetidamente um vetor, comparando elementos adjacentes.
- Quando encontra elementos na ordem errada, ele os **troca**.
- Esse processo continua até que nenhuma troca seja necessária, resultando na ordenação dos elementos.
- Embora simples, o Bubble Sort possui um desempenho ineficiente, especialmente em vetores grandes.
- Atualmente, existem algoritmos mais **sofisticados** e com melhor desempenho de tempo.

Busca

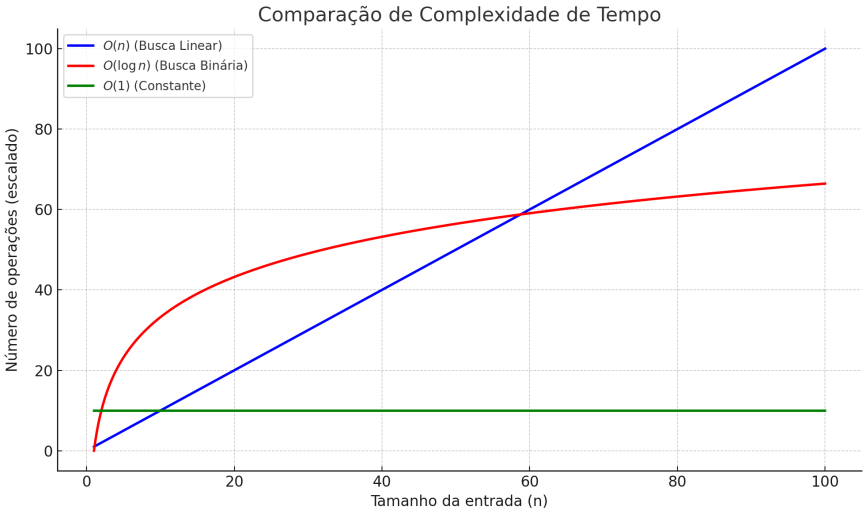
```c
#include <stdio.h>
int main()
{
    int vetor[] = {0,1,4,6,8,10,12,13,15,18};
    int encontra = 15;

    for(int i= 0; i< 10;i++)
        if (vetor[i] == encontra){
            printf("Encontrou na posicao %d.", i);
            break;
        }
    return 0;
}
```

Código 18: Busca Sequencial.

```c
#include <stdio.h>
int main()
{
    int vetor[] = {0, 1, 4, 6, 8, 10, 12, 13, 15, 18};
    int encontra = 15;
    int inicio = 0, fim = 9, meio;
    while (inicio <= fim) {
        meio = (inicio + fim) / 2;
        if (vetor[meio] == encontra) {
            printf("Encontrou na posição %d.\n", meio);
            break;
        } else if (vetor[meio] < encontra) {
            inicio = meio + 1;
        } else {
            fim = meio - 1;
        }
    }
    return 0;
}
```

Código 19: Busca Sequencial.

Sequencial vs Binária I



DE OLIVEIRA, J.F.; MANZANO, J.A.N.G. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores**. 16. ed. São Paulo: Editora Érica, 2004.

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

SCHILDT, Herbert. **C Completo e Total**. 4ª. São Paulo: McGraw-Hill Brasil, 2015.

Estes slides estão protegidos por uma licença Creative Commons

Este modelo foi adaptado de Maxime Chupin.

Marisangila Alves, MSc
marisangila.alves@udesc.com
marisangila.com.br

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA
JOINVILLE
CENTRO DE CIÊNCIAS TECNOLÓGICAS

UDESC
Universidade do Estado de Santa Catarina

2025/1

Linguagem de Programação

Linguagem de Programação C
Vetor