

Introdução à Programação

V E T O R E S

Giorgia de Oliveira Mattos

giorgia@ci.ufpb.br

VETORES - Definição

- Uma **variável estruturada** é aquela que contém mais de um componente e permite que eles sejam acessados individualmente
 - São **homogêneas** quando os componentes são de um mesmo tipo
 - São **heterogêneas** quando os componentes são de tipos diferentes
- Vetor é uma variável estruturada **homogênea**
 - Trata-se de um conjunto de **variáveis do mesmo tipo**, que possuem o mesmo identificador (nome) e são alocadas sequencialmente na memória
 - Cada componente/parte do vetor é chamado de **elemento**
 - Como as variáveis tem o mesmo nome, o que as distingue é **um índice** que referencia sua localização dentro do conjunto
 - Também são conhecidos como arranjo, array e outros

Os vetores são essenciais em programas que processam grandes quantidades de dados e seria inviável a definição de uma variável para cada informação.

```
#include <stdio.h>
```

```
int main(){
```

```
    double    notaAluno1, notaAluno2, notaAluno3, notaAluno4, notaAluno5,  
              notaAluno6, notaAluno7, notaAluno8, notaAluno9 , notaAluno10;
```

```
    double    media;
```

```
    printf("Digite a nota do aluno 1: ");
```

```
    scanf("%lf", &notaAluno1);
```

```
    printf("Digite a nota do aluno 2: ");
```

```
    scanf("%lf", &notaAluno2);
```

```
    printf("Digite a nota do aluno 3: ");
```

```
    scanf("%lf", &notaAluno3);
```

```
    /* ... */
```

```
    printf("Digite a nota do aluno 10: ");
```

```
    scanf("%lf", &notaAluno10);
```

```
    media = (notaAluno1+notaAluno2+/*...*/notaAluno10)/10;
```

```
    return 0;
```

```
}
```

- ✓ Programa armazena a nota de 10 alunos para realizar cálculos estatísticos
- ✓ Se fossem 50 alunos?
- ✓ Repetir a leitura 50 vezes
- ✓ Atualizar a expressão da média com 50 notas

VETORES - Declaração

tipo nome [t];

- tipo - Tipo do vetor e, portanto, o tipo de todos os elementos deste. Pode ser qualquer tipo primitivo ou criado pelo programador
- nome - Identificador (ou nome) da variável vetor
- t - Quantidade de elementos do vetor

```
double notas[50];
```

Definição de um **vetor**
chamado **notas** com **50**
elementos do tipo **double**.

```
#include <stdio.h>
```

```
int main(){
```

```
    double    notaAluno1, notaAluno2, notaAluno3, notaAluno4, notaAluno5,  
              notaAluno6, notaAluno7, notaAluno8, notaAluno9 , notaAluno10;
```

```
    double media;
```

```
    return 0;
```

```
}
```

```
#include <stdio.h>
int main(){
    double notaAluno[10];
    double media;
```

```
    return 0;
```

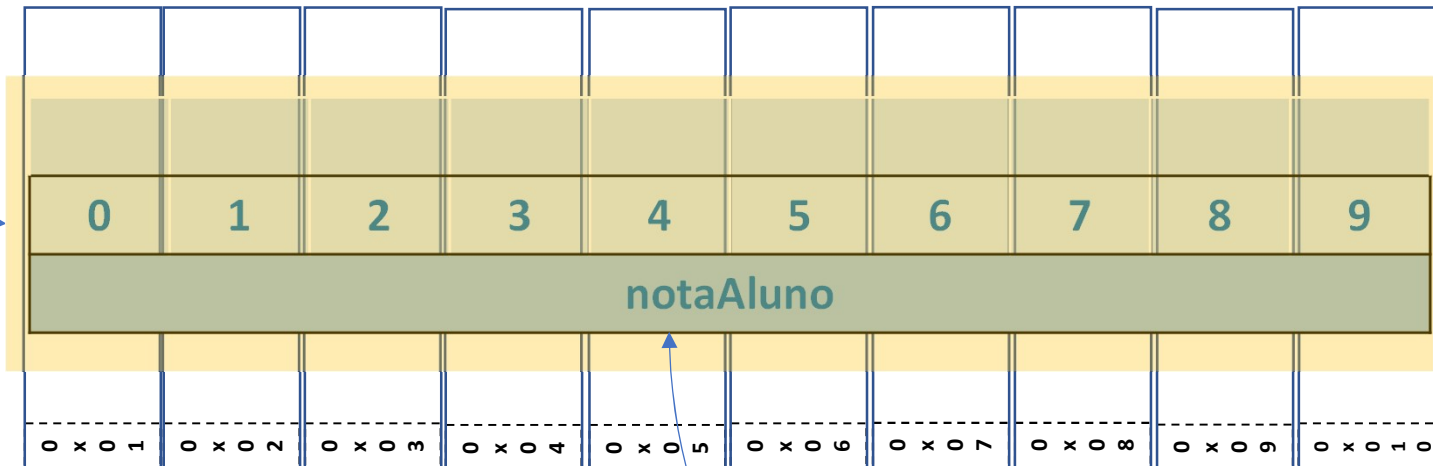
```
}
```

Como fica a memória quando um vetor é alocado?

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

```
double notaAluno[10];
```

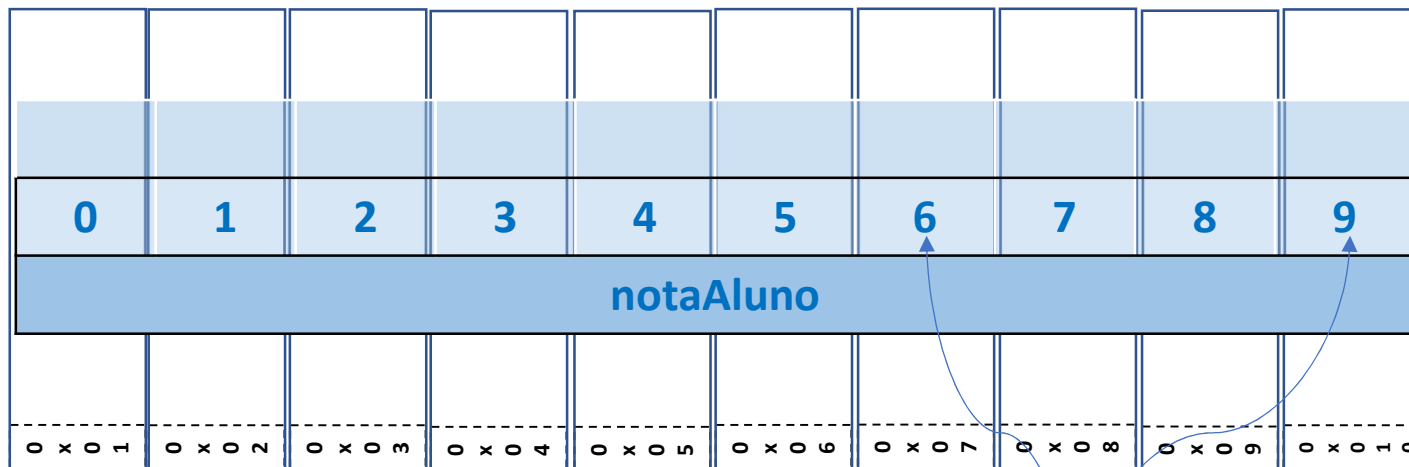
O **vetor** é alocado *contiguamente* na memória, ocupando vários espaços dela



Todo o **vetor** é associado a um único nome, nesse exemplo ***notaAluno***

```
double  notaAluno[10];
```

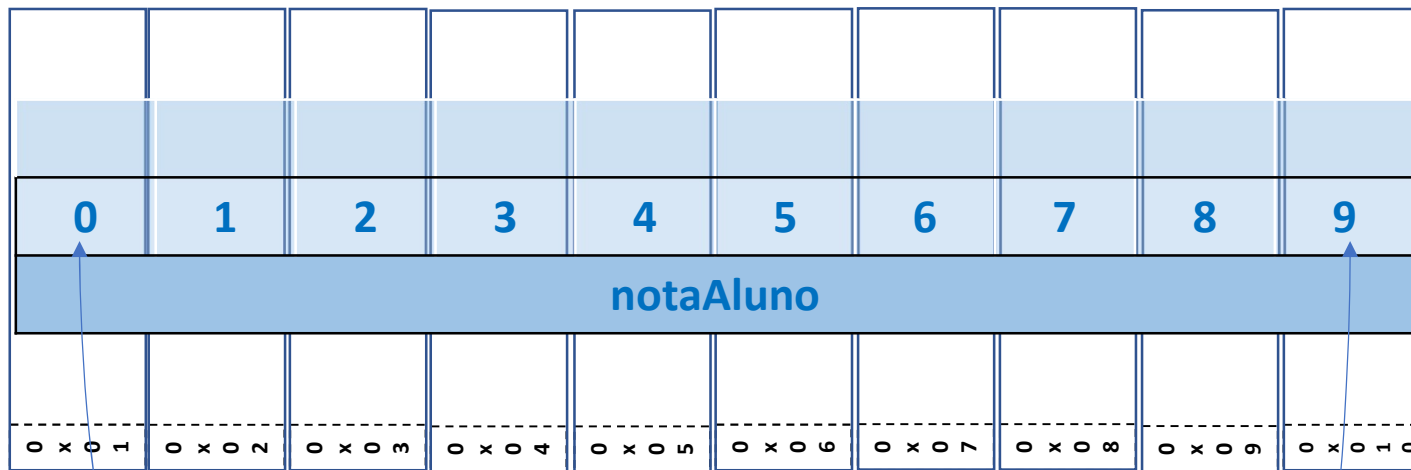
Como o espaço ocupado pelo **vetor** é contíguo, todos os seus elementos são vizinhos na memória



Cada **elemento** ocupa uma **posição** própria na **memória**, inclusive com um **endereço** próprio


```
double notaAluno[10];
```

Para simplificar o acesso, a cada **elemento** é **associado** um *número inteiro* de **índice**.



O **índice** do **primeiro** elemento é sempre **0** e do **último** é o **tamanho do vetor - 1**. Ou seja, o **índice** indica a **posição** do **elemento** dentro do **vetor**, começando do **zero**.

VETORES – Acesso aos elementos

nome [índice];

```
double notas[50];
```

```
notas[0] = 5.0; /* Atribui 5.0 ao elemento inicial do vetor */
```

```
notas[49] = 7.5; /* Atribui 7.5 ao último elemento do vetor */
```

```
notas[50] = 9.0; /* Esta referência é problemática */
```

```
double notas[50];
```

```
notas[0] = 5.0; /* Atribui 5.0 ao elemento inicial do vetor */
```

```
notas[49] = 7.5; /* Atribui 7.5 ao último elemento do vetor */
```

```
notas[50] = 9.0; /* Esta referência é problemática */
```

- Cada elemento do vetor pode ser considerado uma variável do tipo usado na definição do vetor, de modo que podemos utilizar qualquer elemento em qualquer operação permitida em variáveis desse tipo.
- A última referência é problemática porque o último índice válido é o **49**. Ou seja, o vetor não possui um elemento de índice **50**.
- ATENÇÃO! Fique atento para nunca acessar índices inválidos em um vetor. Tal operação não é detectada pelo compilador e costuma causar problemas na execução do programa.

```
/*
```

Esse código atribui o valor **0** a todos os elementos do vetor **ar**.

```
*/
```

A variável inteira **i** como *índice*, dá acesso aos elementos do vetor **ar**

```
#include <stdio.h>
```

```
int main(void) {
```

```
    int ar[9], i;
```

```
    for (i = 0; i < 9; i++) {
```

```
        printf("Iniciando o elemento #%d\n", i);
```

```
        ar[i] = 0;
```

```
    }
```

```
    return 0;
```

```
}
```

Devido às instruções do **for**, **i** assumirá os valores de **0** à **8**, justamente os índices válidos do vetor **ar**, que tem **9** elementos

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

```
/*
```

Esse código atribui o valor **0** a todos os elementos do vetor **ar**.

```
*/
```

A variável inteira **i** como *índice*, dá acesso aos elementos do vetor **ar**.

```
#include <stdio.h>
```

```
int main(void) {
```

```
    int ar[9], i;
```

```
    for (i = 0; i < 9; i++) {
```

```
        printf("Iniciando o elemento #%d\n", i);
```

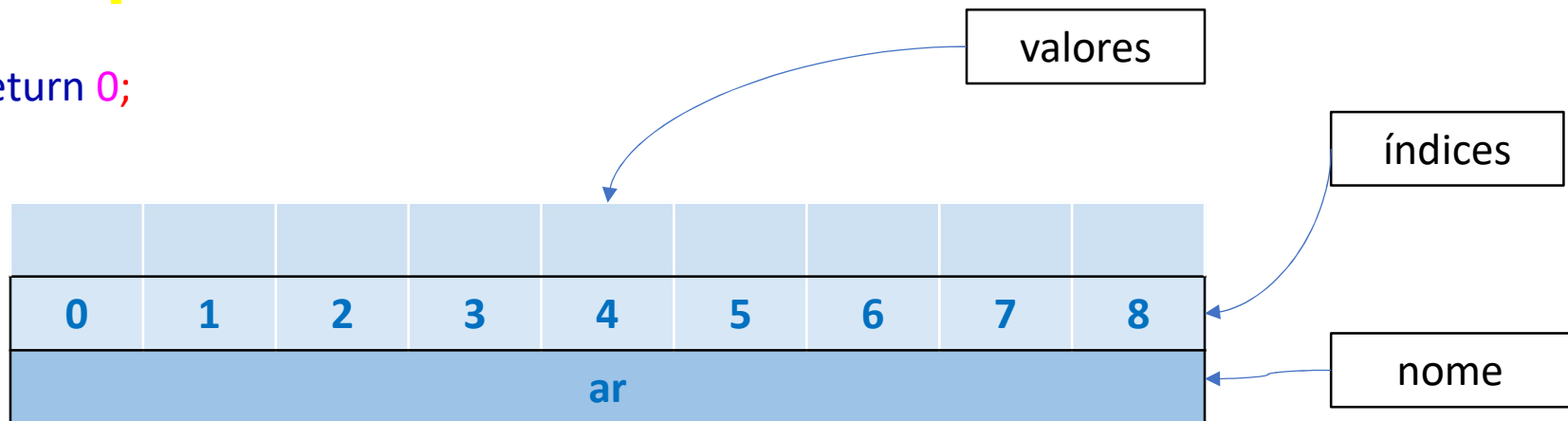
```
        ar[i] = 0;
```

```
    }
```

```
    return 0;
```

```
}
```

Devido às instruções do **for**, **i** assumirá os valores de **0** à **8**, justamente os índices válidos do vetor **ar**, que tem **9** elementos.



```
#include <stdio.h>
```

```
int main(){
```

```
    double notaAluno[10];
```

```
    double media;
```

```
    printf("Digite a nota do aluno 1: ");
```

```
    scanf("%lf", &notaAluno[0]);
```

```
    printf("Digite a nota do aluno 2: ");
```

```
    scanf("%lf", &notaAluno[1]);
```

```
    printf("Digite a nota do aluno 3: ");
```

```
    scanf("%lf", &notaAluno[2]);
```

```
    /* ... */
```

```
    printf("Digite a nota do aluno 10: ");
```

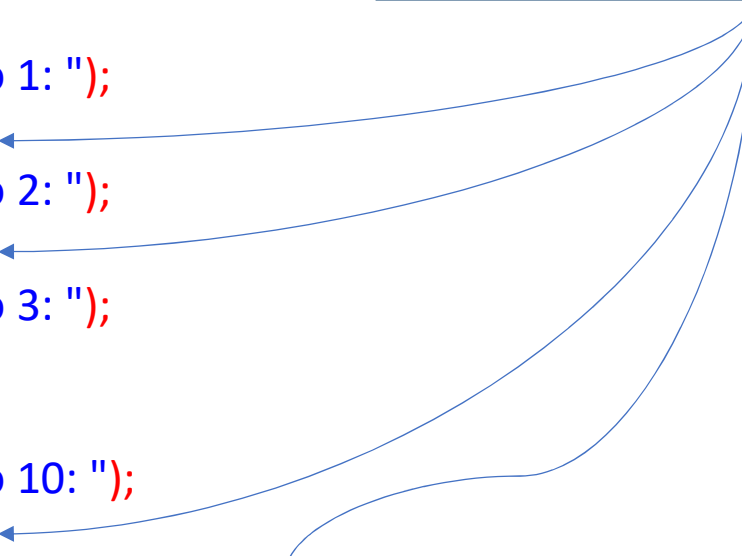
```
    scanf("%lf", &notaAluno[9]);
```

```
    media = (notaAluno[0] + /*...*/ + notaAluno[9])/10;
```

```
    return 0;
```

```
}
```

É possível acessar cada elemento do vetor como se fossem variáveis do tipo **double**.

The diagram consists of four blue curved arrows originating from the text box. The first arrow points to the `notaAluno[0]` in the first `scanf` call. The second arrow points to the `notaAluno[1]` in the second `scanf` call. The third arrow points to the `notaAluno[9]` in the final `scanf` call. The fourth arrow points to the `notaAluno[9]` in the `media` calculation expression.

Está melhor mas as instruções ainda estão muito repetidas e as expressões muito grandes. A única diferença é que, ao invés de 10 variáveis, foi usado um vetor de 10 elementos.

```
#include <stdio.h>
int main(){
    double notaAluno[10], media, soma;
    int i;

    for(i = 0; i < 10; i++){
        printf("Digite a nota do aluno %d: ", i+1);
        scanf("%d", &notaAluno[i]);
    }
    soma = 0;
    for(i = 0; i < 10; i++){
        soma = soma + notaAluno[i];
    }
    media = soma/10;
    return 0;
}
```

O ideal é usar um **laço de repetição** e uma **variável inteira** como **índice...**

```
#include <stdio.h>
int main(){
    double notaAluno[10], media, soma;
    int i;

    for(i = 0; i < 10; i++){
        printf("Digite a nota do aluno %d: ", i+1);
        scanf("%d", &notaAluno[i]);
    }
    soma = 0;
    for(i = 0; i < 10; i++){
        soma = soma + notaAluno[i];
    }
    media = soma/10;
    return 0;
}
```

O ideal é usar um **laço de repetição** e uma **variável inteira como índice...**

Observe o uso do **i** como **índice**

E como os **fors** foram implementados para que **i** assumisse apenas valores de **índices válidos**


```

#include <stdio.h>
int main(){
    double notaAluno[10], media, soma;
    int i;

    for(i = 0; i < 10; i++){
        printf("Digite a nota do aluno %d: ", i+1);
        scanf("%d", &notaAluno[i]);
    }
    soma = 0;
    for(i = 0; i < 10; i++){
        soma = soma + notaAluno[i];
    }
    media = soma/10;
    return 0;
}

```

É possível passar normalmente o endereço de um elemento com o operador **&** em um **scanf** (ou qualquer outra situação que precise do endereço de um elemento).

Inclusive, um ponteiro para **double** poderia apontar para um único elemento sem problemas.

```
double *p = &notaAluno[3];
```

Ou seja, cada elemento do vetor **notaAluno** pode ser utilizado exatamente como uma variável **double** qualquer.

```
#include <stdio.h>
int main(){
    double notaAluno[50], media, soma;
    int i;

    for(i = 0; i < 50; i++){
        printf("Digite a nota do aluno %d: ", i+1);
        scanf("%d", &notaAluno[i]);
    }
    soma = 0;
    for(i = 0; i < 50; i++){
        soma = soma + notaAluno[i];
    }
    media = soma/50;
    return 0;
}
```

Para manipular a nota de **50 alunos**, basta atualizar esses valores.

```
#include <stdio.h>
```

```
#define N_ALUNOS 50
```

```
int main(){  
    double notaAluno[N_ALUNOS], media, soma;  
    int i;  
  
    for(i = 0; i < N_ALUNOS; i++){  
        printf("Digite a nota do aluno %d: ", i+1);  
        scanf("%d", &notaAluno[i]);  
    }  
    soma = 0;  
    for(i = 0; i < N_ALUNOS; i++){  
        soma = soma + notaAluno[i];  
    }  
    media = soma/N_ALUNOS;  
    return 0;  
}
```

Usar uma constante simbólica!

O código-fonte fica com melhor legibilidade e manutenibilidade

Alterar o valor da constante simbólica fará com que o programa trate uma nova quantidade de alunos.

```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```

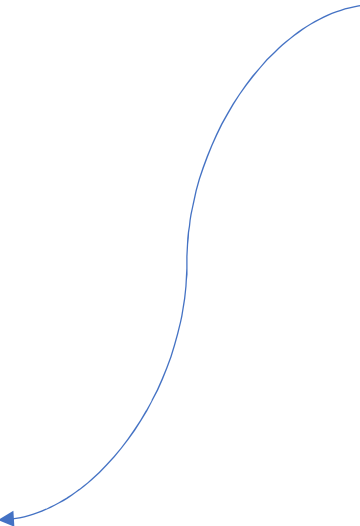
```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }
```

```
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

Esse programa faz a leitura de 5 valores e os armazena em um vetor.



```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```


```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }
```

```
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

Os elementos armazenados
são percorridos um a um.
Eles são somados e
checados se são pares.



```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;  
  
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }  
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }  
}
```

Ao final são exibidas a média e a quantidade de números pares armazenados no vetor.

```
printf("Media dos valores: %.1f\n", soma/5.0);  
printf("Quantidade de numeros pares: %d\n", qPares);  
return 0;
```

```
}
```

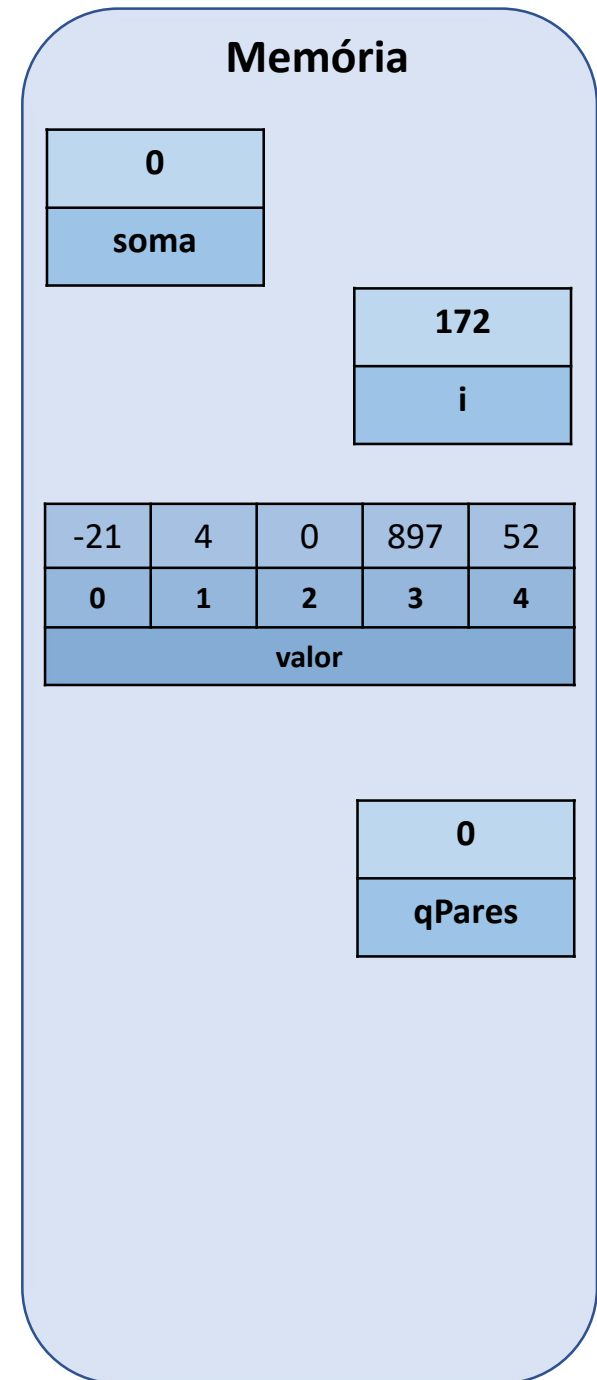
```

#include <stdio.h>
int EhPar(int n){
    return n % 2 == 0;
}

int main(){
    int valor[5];
    int i, soma = 0, qPares = 0;

    for(i = 0 ; i < 5; i++){
        scanf("%d", &valor[i]);
    }
    for(i = 0; i < 5; i++){
        soma = soma + valor[i];
        if (EhPar(valor[i])){
            qPares++;
        }
    }
    printf("Media dos valores: %.1f\n", soma/5.0);
    printf("Quantidade de numeros pares: %d\n", qPares);
    return 0;
}

```



```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```

```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }
```

```
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

O laço **for** fará com que **i** assuma os valores de **0** a **4**.

Memória

0
soma

172
i

-21	4	0	897	52
0	1	2	3	4
valor				

0
qPares


```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```

```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }
```

```
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

O laço **for** fará com que **i** assuma os valores de **0** a **4**.

Memória

0
soma

5
i

2	5	1	6	2
0	1	2	3	4
valor				

0
qPares

```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```

```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);  
    }
```

```
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

No segundo for o *i* também assumirá os valores de **0** a **4** (os índices válidos do vetor **valor**)

A variável soma acumulará a soma de todos os elementos de valor

A função EhPar avalia cada elemento do vetor. E qPares é incrementada

Memória

16
soma

5
i

2	5	1	6	2
0	1	2	3	4
valor				

3
qPares

```
#include <stdio.h>
```

```
int EhPar(int n){  
    return n % 2 == 0;  
}
```

```
int main(){  
    int valor[5];  
    int i, soma = 0, qPares = 0;
```

```
    for(i = 0 ; i < 5; i++){  
        scanf("%d", &valor[i]);
```

```
    }  
    for(i = 0; i < 5; i++){  
        soma = soma + valor[i];  
        if (EhPar(valor[i])){  
            qPares++;  
        }  
    }
```

```
    printf("Media dos valores: %.1f\n", soma/5.0);  
    printf("Quantidade de numeros pares: %d\n", qPares);  
    return 0;
```

```
}
```

Os **printfs** imprimem o valor da **soma** dividido por 5 (3.2) e o valor de **qPares** (3) .

Memória

16
soma

5
i

2	5	1	6	2
0	1	2	3	4
valor				

3
qPares

VETORES – Inicialização

- Vetores de **duração fixa** são iniciados com **zero** por padrão
 - Ao menos que sejam iniciados explicitamente com outros valores.
- Vetores de **duração automática** só são iniciados explicitamente.
 - Ou seja, são alocados com *lixos de memória* se não forem iniciados.

Sendo variáveis, as mesmas regras de **escopo** e **duração** são válidas para os vetores.

```
static int meuVetor1[5];  
static int meuVetor2[5] = {1, 2, 3.14, 4, 5};  
  
int vetorA[3] = {1, 2, 3, 4, 5};  
int vetorB[5] = {2, -5};
```

```
static int meuVetor1[5];  
static int meuVetor2[5] = {1, 2, 3.14, 4, 5};  
static int meuArray2[] = {1, 2, 3.14, 4, 5};  
  
int vetorA[3] = {1, 2, 3, 4, 5};  
int vetorB[5] = {2, -5};
```

Por ter *duração fixa* (pelo uso do ***static***), meuVetor1 é *implicitamente* iniciado com **0** (todos os elementos são **0**).

meuVetor2 é iniciado *explicitamente* com os valores entre as chaves, separados por vírgula. Serão atribuídos aos elementos do vetor, do primeiro ao último, na ordem em que aparecem. Podem ocorrer conversões automáticas de tipo.

Se o vetor **vetorA** não tivesse sido iniciado, os valores iniciais dos elementos seriam *lixos de memória*.

Ao tentar iniciar um vetor com mais valores que a sua capacidade, os elementos a mais são ignorados.

Ao tentar iniciar um vetor com menos valores que a sua capacidade, os elementos restantes são *implicitamente* iniciados com **0**.

```
#include <stdio.h>
```

```
int main(void) {
```

```
    int i, ar[5] = { 1, -1, 2, 0, 4 };
```



1	-1	2	0	4
0	1	2	3	4
ar				

```
    /* Exibe o array do primeiro ao último elemento */
```

```
    for (i = 0; i < 5; ++i) {
```

```
        printf("ar[%d] = %2d\n", i, ar[i]);
    }
```

Acessa os elementos na ordem crescente dos índices.

```
    printf("\n\n"); /* Pula duas linhas */
```

```
    /* Exibe o array do último ao primeiro elemento */
```

```
    for (i = 4; i >= 0; --i) {
```

```
        printf("ar[%d] = %2d\n", i, ar[i]);
    }
```

Acessa os elementos na ordem decrescente dos índices.

```
    return 0;
```

```
}
```

```

#include <stdio.h>
int main(void) {

    int i, ar[5] = { 1, -1, 2, 0, 4 };
    /* Exibe o array do primeiro ao último elemento */

    for (i = 0; i < 5; ++i) {
        printf("ar[%d] = %2d\n", i, ar[i]);
    }

    printf("\n\n"); /* Pula duas linhas */

    /* Exibe o array do último ao primeiro elemento */
    for (i = 4; i >= 0; --i) {
        printf("ar[%d] = %2d\n", i, ar[i]);
    }
    return 0;
}

```

```

ar[0] = 1
ar[1] = -1
ar[2] = 2
ar[3] = 0
ar[4] = 4

```

```

ar[4] = 4
ar[3] = 0
ar[2] = 2
ar[1] = -1
ar[0] = 1

```

VETORES – O operador sizeof

- Operador unário onde o operando pode ser:
 - Um tipo de dado
 - Uma constante
 - Uma variável ou
 - Uma expressão.

VETORES – O operador sizeof

- Quando aplicado a um tipo de dado, resulta no número de *bytes* necessários para alocar uma variável desse tipo.

```
int main(void)
```

```
{
```

```
    size_t tamanhoTipoDouble;
```

```
    tamanhoTipoDouble = sizeof(double);
```

```
    printf("\n>>> Bytes ocupados por uma variavel do"
```

```
           "\n>>> tipo double: %d\n", tamanhoTipoDouble);
```

```
    return 0;
```

```
}
```

Inteiro sem sinal.

Parece uma função mas é um operador e seu operando.

VETORES – O operador sizeof

- Quando aplicado a uma constante ou variável, o resultado é o numero de *bytes* necessários para armazenar a constante ou variável.
- Quando aplicado a uma expressão, resulta no número de *bytes* necessários para guardar o resultado da expressão, caso ela seja avaliada.
 - A expressão não é de fato avaliada.

VETORES – O operador sizeof

```
#include <stdio.h>
int main(void) {
    int i = 0;
    size_t tamanhoDaExpressao;

    tamanhoDaExpressao = sizeof(++i);
    printf("\n>>> Bytes ocupados pela expressao ++i: %d", tamanhoDaExpressao);
    printf("\n>>> Valor de i: %d\n", i);
    return 0;
}
```

A expressão **não** é
avaliada! **i** não é
incrementado

>>> Bytes ocupados pela expresao ++i: 4
>>> Valor de i: 0

VETORES – O operador sizeof

```
int main(void) {  
    int ar[] = { -1, 2, -2, 7, 3 };  
    size_t tamanhoDoArray, tamanhoDeElemento, nElementos;  
  
    tamanhoDoArray = sizeof(ar);  
    tamanhoDeElemento = sizeof(ar[0]);  
    nElementos = tamanhoDoArray / tamanhoDeElemento;  
  
    printf("\n>>> Bytes ocupados pelo array: %d\n", tamanhoDoArray);  
    printf("\n>>> Bytes ocupados por um elemento "  
        "do array: %d\n", tamanhoDeElemento);  
    printf("\n>>> Numero de elementos do array: %d\n", nElementos);  
    return 0;  
}
```

Tamanho em *bytes* do vetor **inteiro**.

Tamanho em *bytes* de **um elemento** do vetor.

```
>>> Bytes ocupados pelo array: 20  
>>> Bytes ocupados por um elemento do array: 4  
>>> Numero de elementos do array: 5
```

VETORES – uso com funções

- *Vetor* declarado como parâmetro formal:

```
void MinhaFuncao(double ar[])
```



Sintaxe para indicar que o parâmetro **ar** é um **vetor** de **double**.

O tamanho não é indicado.

VETORES – uso com funções

- *Vetor* declarado como parâmetro formal:

```
void MinhaFuncao(double ar[]) {  
    printf("O tamanho do array e': %d\n", sizeof(ar));  
}
```

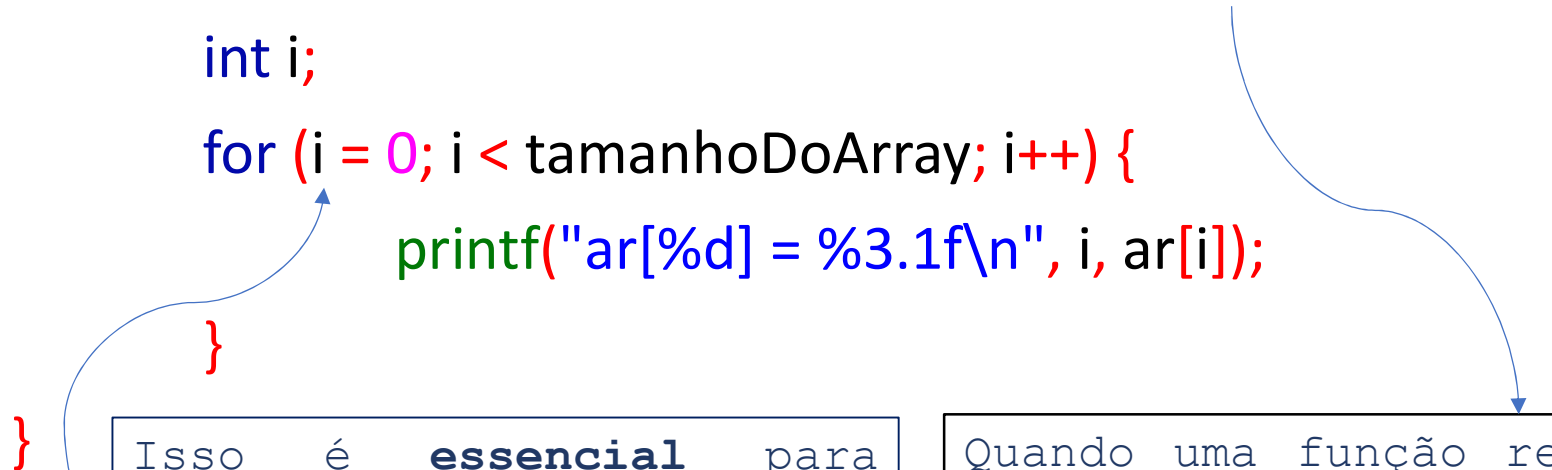
Dentro de uma função, o **sizeof** não irá resultar no tamanho do *array*!

Sendo assim, a função não tem como descobrir quantos elementos possui o *vetor*.

VETORES – uso com funções

- *Vetor* declarado como parâmetro formal:

```
void ExibeArrayDoubles(double ar[], int tamanhoDoArray) {  
    int i;  
    for (i = 0; i < tamanhoDoArray; i++) {  
        printf("ar[%d] = %3.1f\n", i, ar[i]);  
    }  
}
```



Isso é **essencial** para evitar que a função acesse elementos não pertencentes ao vetor.

Quando uma função recebe um vetor como um dos seus parâmetros, normalmente um *parâmetro adicional* é necessário para indicar o tamanho do vetor.

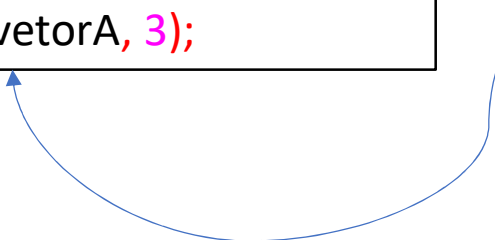
VETORES – uso com funções

- *Vetor* declarado como parâmetro formal:

```
void ExibeArrayDoubles(double ar[], int tamanhoDoArray) {  
    int i;  
    for (i = 0; i < tamanhoDoArray; i++) {  
        printf("ar[%d] = %3.1f\n", i, ar[i]);  
    }  
}
```

```
double vetorA[3] = {2, 4, 6};  
ExibeArrayDoubles(vetorA, 3);
```

No parâmetro real da chamada da função, basta colocar o nome de um *vetor* do mesmo tipo do parâmetro formal.



VETORES – passagem de parâmetros

- **Passagem de Vetores como parâmetros**

- A linguagem C não permite que vetores sejam passados na íntegra como parâmetros (passagem por valor) para uma função
- Deve-se passar apenas o endereço da posição inicial do vetor. Este endereço é obtido utilizando-se o nome do vetor sem o índice entre colchetes
- Isso quer dizer que é possível passar um vetor para uma função somente se a passagem for **por referência**

VETORES – Exercícios

1. Calcular a soma dos elementos de um vetor de 10 posições.
2. Calcular a média dos elementos do vetor do exercício 1.
3. Calcular quantos elementos do vetor, do exercício 2, estão acima da média.
4. Multiplicar os elementos que estão nos índices ímpares do vetor pela constante 15.
5. Preencher um vetor X de 10 elementos com o número 1 se o índice do elemento for ímpar e com o número 0 se o índice for par. Mostrar o vetor X.
6. Dados dois vetores A e B com 10 elementos cada. Armazenar no vetor C a soma do elemento em A com o elemento em B em cada uma das posições.

VETORES – Exercícios

7. Ler dois vetores A e B de 10 elementos cada. Intercalar os elementos de A com os elementos de B de maneira a formar um terceiro vetor, C. Escrever o vetor C.
8. Ler um vetor A com 20 elementos. Gerar e mostrar o vetor B obtido pela inversão da ordem do vetor A.
9. Obter um vetor V de 20 posições. Mostrar o maior elemento do vetor V e a sua posição.
10. Ler um vetor de 16 posições. Troque os 8 primeiros valores pelos 8 últimos. Escreva ao final o vetor obtido.
11. **TAREFA DE CASA**: Pesquisar as funções srand() e rand ().

VETORES – Exercícios

12. Escreva uma função que concatena/junta dois vetores. Por exemplo, $V1 = 0, 1, 2, 3, 4$ e $V2 = 4, 3, 2, 1, 0$, o vetor resultante será $R = 0, 1, 2, 3, 4, 4, 3, 2, 1, 0$.
13. Fazer uma função que procura por um número em um vetor e retorna o seu endereço (índice) caso o encontre. Se não encontrar mostrar uma mensagem de que não achou.
14. Faça uma função que recebe, por parâmetro, 2 vetores A e B de tamanho 10 de inteiros. Ao final da função, B deve conter o fatorial de cada elemento de A. O vetor B retorna alterado.
$$\begin{array}{rcccc} A = & 4 & 1 & 0 & 3 \dots \\ B = & 24 & 1 & 1 & 6 \dots \end{array}$$
15. Fazer um programa que leia dois conjuntos, A e B, de 10 inteiros cada. Escrever uma função que determine o conjunto interseção entre A e B (elementos comuns aos dois conjuntos).