

# Algoritmos e Estruturas de Dados I

## Passagem de Parâmetros

Pedro O.S. Vaz de Melo

# Passagem de parâmetros

- Toda função define um processamento a ser realizado.
- Este processamento depende dos valores dos parâmetros da função.
- Assim, para usar uma função, um programa precisa fornecer a ela os parâmetros adequados. Exemplo:
  - Para calcular o seno de  $30^\circ$ , escrevemos: `sin(pi/6);`
  - Para calcular o valor absoluto de  $a-b$ , escrevemos: `abs(a-b);`
  - Para calcular o mdc de 12 e 8, escrevemos: `mdc(12,8);`

# Passagem de parâmetros

- O mecanismo de informar os valores a serem processados pela função chama-se **passagem de parâmetros**.
- A Linguagem C define duas categorias de passagem de parâmetros: **passagem por valor** e **passagem por endereço** (ou **passagem por referência**).
- Normalmente, a passagem de parâmetros a uma função é **por valor**.
- Mas, como os parâmetros de uma função são variáveis **locais**, alguns aspectos devem ser observados.

# Passagem por valor

- Considere o exemplo abaixo:

```
void alterar(int x, int y, int z)
{
    printf("Valores recebidos ... %d, %d e %d\n",x,y,z);
    x++;
    y++;
    z++;
    printf("Valores alterados ... %d, %d e %d\n",x,y,z);
}

void main()
{
    int a = 1, b = 2, c = 3;

    alterar(a,b,c);
    printf("Valores finais ..... %d, %d e %d\n",a,b,c);
}
```

O que este programa irá exibir?

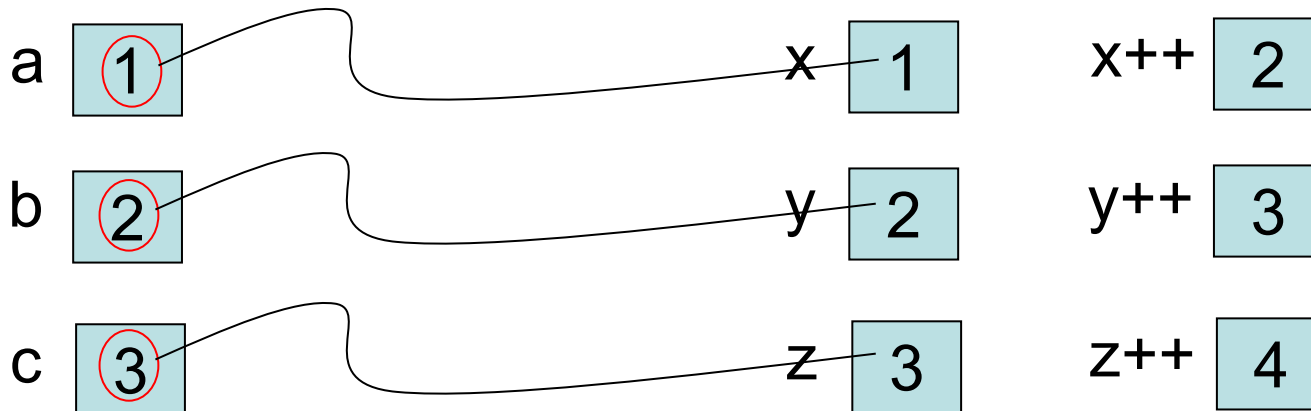
Valores recebidos ...	1, 2 e 3
Valores alterados ...	2, 3 e 4
Valores finais .....	1, 2 e 3

# Passagem por valor

- Observe que os valores das variáveis **a**, **b** e **c** não foram modificados na função **alterar**. Por quê?
- O tipo de passagem de parâmetros utilizado é **por valor**. Ou seja, são feitas apenas cópias dos valores das variáveis **a**, **b**, e **c** nas variáveis **x**, **y** e **z**.

Escopo: função **main**

Escopo: função **alterar**



Apenas os  
conteúdos  
de **x**, **y** e **z**  
são alterados.

# Passagem por referência

- Mas, e se quisermos que a função modifique os valores das variáveis **a**, **b** e **c** passadas a ela como parâmetros?
- Neste caso, em vez de passar para a função os valores destas variáveis, é preciso passar os seus **endereços**.  
Como assim?
- Considere, por exemplo, que as variáveis **a**, **b** e **c** correspondem, respectivamente, aos endereços (hexadecimais) **F000**, **F010** e **F020**.

# Passagem por referência

Ou seja:

Endereço	Conteúdo	Variável
F000	1	a
F010	2	b
F020	3	c

Sabemos, portanto, que:

&a = F000 (endereço de **a**);

&b = F010 (endereço de **b**);

&c = F020 (endereço de **c**);

a = 1, b = 2, c = 3 (valores das variáveis).

# Passagem por referência

- Considere uma variável declarada como:

```
int *x;
```

F000	1	a
F010	2	b
F020	3	c

- $x$  é um **ponteiro para int**, ou seja,  $x$  é uma variável que armazena o endereço de uma variável do tipo **int**.
- Considere agora que: 

```
x = &a;
```
- Neste caso,  $x$  armazena o valor **F000**.

Define-se  $*x$ , como sendo o valor contido na posição de memória apontada por  $x$ . Ou seja,  $*x$  vale 1.



# Passagem por referência

- Considere agora o exemplo anterior reescrito como:

```
void alterar(int *x, int *y, int *z)
{
    printf("Valores recebidos ... %d, %d e %d\n", *x, *y, *z);
    *x++;
    *y++;
    *z++;
    printf("Valores alterados ... %d, %d e %d\n", *x, *y, *z);
}

void main()
{
    int a = 1, b = 2, c = 3;

    alterar(&a, &b, &c);
    printf("Valores finais ..... %d, %d e %d\n", a, b, c);
}
```

O que este programa irá exibir?

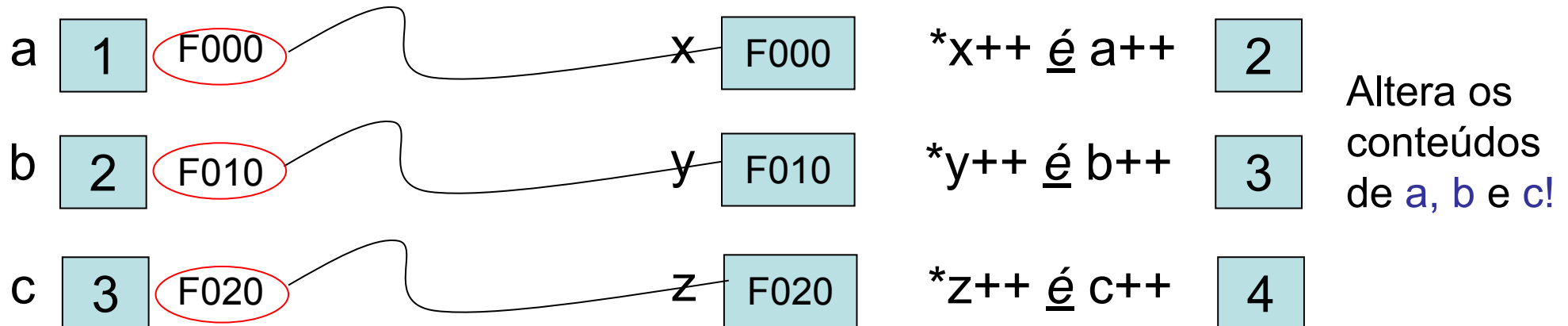
Valores recebidos ...	1, 2 e 3
Valores alterados ...	2, 3 e 4
Valores finais .....	2, 3 e 4

# Passagem por referência

- Observe agora que os valores das variáveis **a**, **b** e **c** foram modificados na função **alterar**. Por quê?
- O tipo de passagem de parâmetros utilizado é **por referência**. Ou seja, são passados os endereços das variáveis **a**, **b**, e **c** para os ponteiros **x**, **y** e **z**.

Escopo: função **main**

Escopo: função **alterar**



# Passagem por referência

- **Atenção!**

Considere que o endereço de **x** é **FFF1**.

```
int x = 1;  
int *a;  
a = &x;
```

Neste caso:

```
a = FFF1 (endereço de x)  
*a = 1   (pois *a = x = *(&x) = 1)
```

Logo:

```
&(*a) = &x = FFF1 = a
```

Portanto:

```
&(*a) ≡ a
```

# Problema 1

- Implemente uma função que classifique os elementos de um vetor em ordem crescente usando o algoritmo de ordenação por seleção.
- Teste a função com dados aleatórios.

# Análise do programa

```
void troca(int v[], int i, int j)
{
    int aux;
    aux = v[i];
    v[i] = v[j];
    v[j] = aux;
}

void ordenar_por_selecao(int x[], int n)
{
    int menor, pos;
    int i, k = 0;

    while (k < n)
    {
        menor = INFINITO;
        for (i = k; i < n; i++)
            if (x[i] < menor)
            {
                menor = x[i];
                pos = i;
            }
        troca(x, k, pos);
        k++;
    }
}
```

# Análise do programa

```
void troca(int v[], int i, int j)
{
    int aux;
    aux = v[i];
    v[i] = v[j];
    v[j] = aux;
}

void ordenar_por_selecao(int x[], int n)
{
    int menor, pos;
    int i, k = 0;

    while (k < n)
    {
        menor = INFINITO;
        for (i = k; i < n; i++)
            if (x[i] < menor)
            {
                menor = x[i];
                pos = i;
            }
        troca(x, k, pos);
        k++;
    }
}
```

Para haver a troca, os parâmetros não devem ser passados por referência (endereço)?

Observe como um vetor é passado como parâmetro de uma função: o **número de elementos** do vetor não precisa ser declarado.

# Passagem de parâmetros

- Na aula anterior, discutimos a **passagem de parâmetros por referência**. Quando deve ser feita este tipo de passagem? Veja o exemplo:

```
void troca (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void main()
{
    int x = 2, y = 5;
    troca(x, y);
    printf("x = %d y = %d\n", x, y);
}
```

Exibe: x = 2 y = 5

Passagem por Valor

```
void troca (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

void main()
{
    int x = 2, y = 5;
    troca(&x, &y);
    printf("x = %d y = %d\n", x, y);
}
```

Exibe: x = 5 y = 2

Passagem por Referência

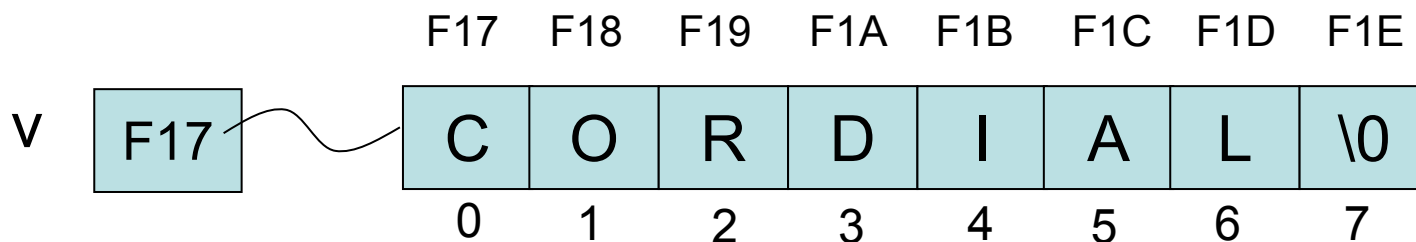
# Vetores como parâmetros

- No caso de uma função ter como parâmetro um vetor, temos um caso particular de grande importância.

**Por quê?** Porque o nome de um vetor nada mais é que um ponteiro para sua primeira posição.

- Exemplo:

```
char v[8];
```





# Vetores como parâmetros

- Mas, então como é possível usarmos a notação:

```
nome_do_vetor[índice]
```

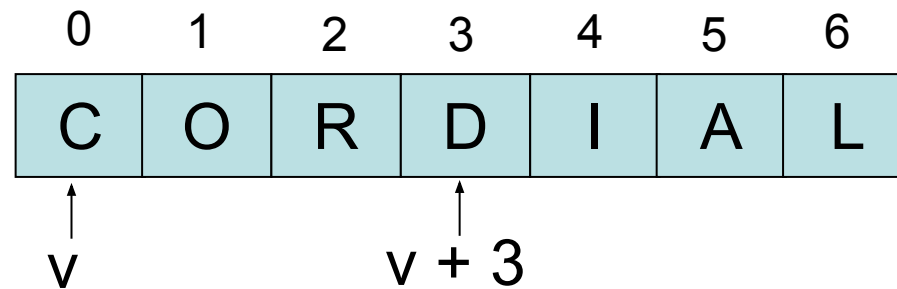
- Isto pode ser facilmente explicado, desde que se entenda que a notação acima é **absolutamente equivalente** a:

```
* (nome_do_vetor + índice)
```

- Mas, o que significa somar (ou subtrair) um valor a um ponteiro?

# Vetores como parâmetros

- Quando incrementamos um ponteiro, ele passa a apontar para o próximo valor do mesmo tipo.
- Exemplo:** ao incrementar um ponteiro para **char**, ele anda 1 byte na memória e ao incrementar um ponteiro para **double** ele anda 8 bytes.



```
 $v \equiv \&v[0]$   
 $*v \equiv v[0] = 'C'$   
 $*(v + 3) \equiv v[3] = 'D'$ 
```

➡ São equivalentes!!!

# Vetores como parâmetros

- Logo, como o nome de um **vetor** é também um ponteiro, a **passagem de parâmetro** para vetores é sempre **por referência**.

Assim, qualquer modificação ocorrida no vetor dentro da função será, na realidade, feita sobre o parâmetro usado na chamada.

```
void ordenar_por_selecao(int x[], int n)
{
    int menor, pos;
    int i, k = 0;

    // Classificar vetor
    ordenar_por_selecao(a, n);
}
```

Alterações no vetor **x** dentro da função **ordenar\_por\_selecao** serão também realizadas no vetor **a**.

Na definição da função, podemos substituir **int x[]** por **int x[TAM\_MAX]** ou ainda **int \*x**.

# Vetores como parâmetros

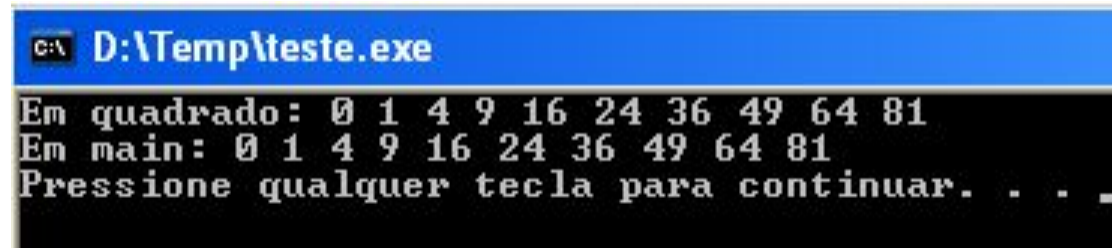
- O que devemos fazer se desejarmos que os elementos de um vetor, passado como parâmetro para uma função, não sejam alterados?
- **Resposta:** dentro da função é preciso atribuir os elementos do vetor a uma variável local.
- Considere o exemplo mostrado a seguir.

# Vetores como parâmetros

```
#define TAM_MAX 10

void quadrado(int v[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        v[i] = pow(v[i],2);
    printf("Em quadrado: ");
    for (i = 0; i < n; i++)
        printf("%d ",v[i]);
    printf("\n");
}

void main()
{
    int i;
    int a[TAM_MAX] = { 0,1,2,3,4,5,6,7,8,9 };
    quadrado(a,10);
    printf("Em main: ");
    for (i = 0; i < 10; i++)
        printf("%d ",a[i]);
    printf("\n");
    system("pause");
}
```



```
C:\ D:\Temp\teste.exe
Em quadrado: 0 1 4 9 16 24 36 49 64 81
Em main: 0 1 4 9 16 24 36 49 64 81
Pressione qualquer tecla para continuar. . . .
```

**Sim!** A passagem é por referência.

Os elementos do vetor **a** terão seus valores modificados pela função **quadrado**?

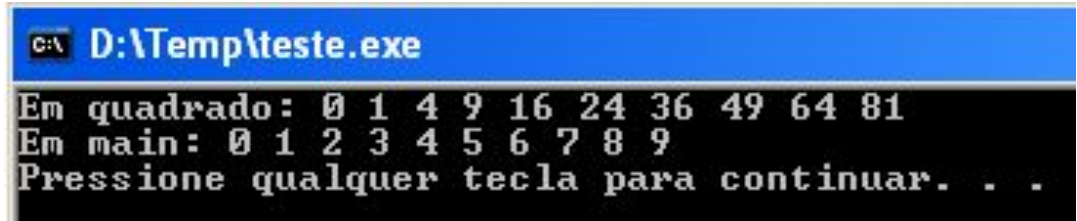
# Vetores como parâmetros

```
#define TAM_MAX 10

void quadrado(int v[], int n)
{
    int i, x[TAM_MAX];
    for (i = 0; i < n; i++)
        x[i] = v[i];
    for (i = 0; i < n; i++)
        x[i] = pow(x[i],2);
    printf("Em quadrado: ");
    for (i = 0; i < n; i++)
        printf("%d ", x[i]);
    printf("\n");
}

void main()
{
    int i;
    int a[TAM_MAX] = { 0,1,2,3,4,5,6,7,8,9 };
    quadrado(a,10);
    printf("Em main: ");
    for (i = 0; i < 10; i++)
        printf("%d ", a[i]);
    printf("\n");
    system("pause");
}
```

Atenção! Observe que a cópia dos elementos foi feita **um a um**. O que aconteceria se fizéssemos:  **$x = v$** ?



```
C:\ D:\Temp\teste.exe
Em quadrado: 0 1 4 9 16 24 36 49 64 81
Em main: 0 1 2 3 4 5 6 7 8 9
Pressione qualquer tecla para continuar. . .
```

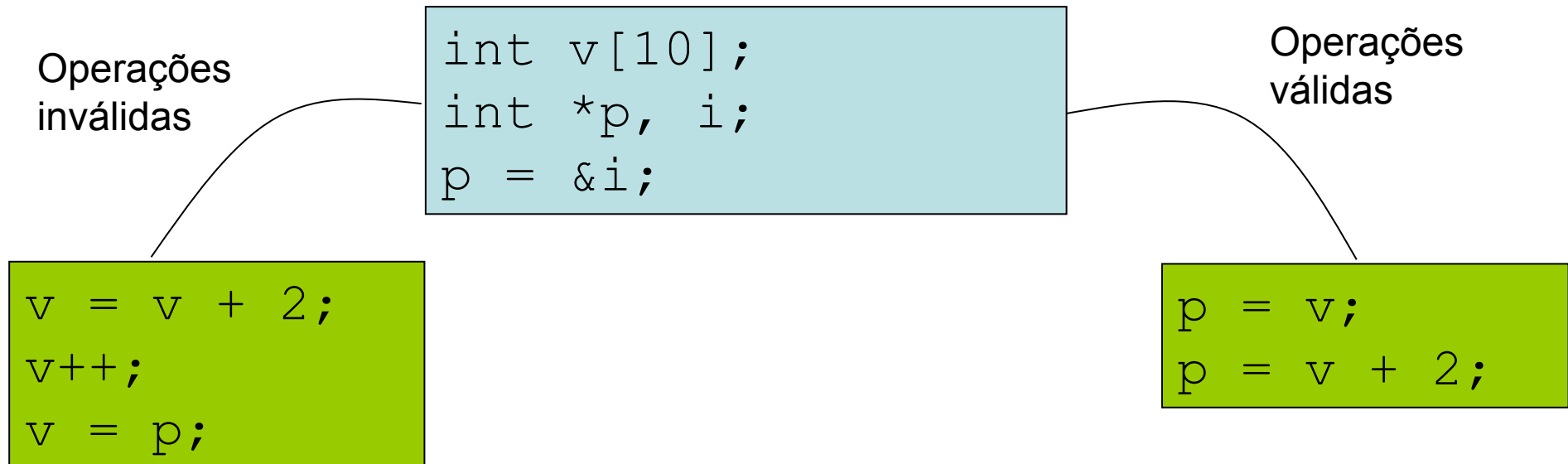
**Não!** Foi feita uma cópia de **v** em **x**.

E agora, os elementos do vetor **a** terão seus valores modificados pela função **quadrado**?

# Diferença entre vetores e ponteiros

## Atenção!!

- Há uma **diferença importante** entre o nome de um vetor e um ponteiro: um ponteiro é uma variável, mas o **nome de um vetor não é uma variável**.
- Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor". Exemplo:



# Qualificador **const**

- Para indicar que um parâmetro de função não deve ser alterado, usa-se o qualificador **const**.
- Exemplo:

```
void copiar(const int *a, int *b)
{
    *a = *b;
}

int main()
{
    int a = 2, b = 3;

    printf("a = %d    b = %d\n", a, b);
    copiar(&a, &b);
    printf("a = %d    b = %d\n", a, b);
    system("pause");
    return 0;
}
```

Se existir na função uma atribuição de valor a um parâmetro declarado como constante, a compilação indicará um erro.