



Técnicas de Programação 1

8ª parte

Funções – Material Extra

Prof. Jobson Massollar

jobson@uniriotec.br



Quando dizemos que uma função deve retornar um ou mais valores, não estamos afirmando que deve ser usado o comando **return**.

Regra 1: o comando **return** só pode retornar **um valor** (tipo primitivo, ponteiro ou struct)!



Situação 1: se a função deve retornar apenas **um valor**, **primitivo** (int, char, float, double, bool, etc.) **ou struct**, então:

- a) Retorne o valor com o comando **return** (mais indicado)
- b) Retorne o valor em uma **referência**



Exemplo 1: função que retorna a raiz cúbica de um número.

Solução A

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

double raiz_cubica(double n) {
    return pow(n, 1/3.0);
}

int main()
{
    double rc = raiz_cubica(4.5);
}
```

Solução B

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

void raiz_cubica(double n, double &rc) {
    rc = pow(n, 1/3.0);
}

int main()
{
    double rc;

    raiz_cubica(4.5, rc);
}
```



Situação 2: se a função deve retornar **dois valores, primitivos** (int, char, float, double, bool, etc.) **ou struct**, então:

- a) Retorne os dois valores usando **duas referências** (mais indicado)
- b) Retorne um valor em **uma referência** e o outro no comando **return**



Exemplo 2: função que retorna o menor e o maior valor de um vetor.



Solução A

```
#include <stdio.h>

void maior_menor(int v[], int tam, int &menor, int &maior) {
    maior = v[0];
    menor = v[0];
    for (int i = 1; i < tam; i++)
        if (v[i] > maior)
            maior = v[i];
        else if (v[i] < menor)
            menor = v[i];
}

int main()
{
    int vetor[] = { 5, 3, 8, 1, 12, 4, 9, 3, 10, 6 };
    int menor, maior;

    maior_menor(vetor, 10, menor, maior);
}
```



Solução B

```
#include <stdio.h>

int maior_menor(int v[], int tam, int &menor) {
    int maior = v[0];
    menor = v[0];
    for (int i = 1; i < tam; i++)
        if (v[i] > maior)
            maior = v[i];
        else if (v[i] < menor)
            menor = v[i];
    return maior;
}

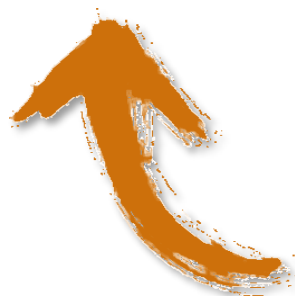
int main()
{
    int vetor[] = { 5, 3, 8, 1, 12, 4, 9, 3, 10, 6 };
    int menor, maior;

    maior = maior_menor(vetor, 10, menor);
}
```




Situação 3: se a função deve retornar **dois ou mais valores**, **primitivos** (int, char, float, double, bool, etc.) ou **struct**, então:

- a) Retorne os todos os valores usando **referências** (mais indicado)
- b) Retorne um valor com o comando **return** e os demais usando **referências**



Generalização da Situação 2



Quando dizemos que uma função deve retornar um ou mais **vetores**, não podemos usar o comando **return**.

Regra 2: funções **não** podem retornar vetores!



```
#include <stdio.h>

const int TAM = 5;

int [] le_vetor(int tam) {
    int v[tam];

    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }

    return v;
}
```

```
int main()
{
    int vetor[] = le_vetor(TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);
}
```



Erros de compilação!



```
#include <stdio.h>

const int TAM = 5;

int * le_vetor(int tam) {
    int v[tam];

    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }

    return v;
}
```

```
int main()
{
    int *vetor = le_vetor(TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);
}
```



Esse programa compila,
mas dá erro de execução!



Regra 3: se um vetor é definido em uma função, então ele é uma **variável local** dessa função. Dessa forma, quando a função terminar o vetor vai **deixar de existir!**



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

```
const int TAM = 5;
```

```
int * le_vetor(int tam) {  
    int v[tam];
```

```
    for(int i = 0; i < tam; i++) {  
        printf("Digite um no.: ");  
        scanf("%d", &v[i]);  
    }
```

```
    return v;
```

```
}
```

```
int main()
```

```
{
```

```
    int *vetor = le_vetor(TAM);
```

```
    for(int i = 0; i < TAM; i++)  
        printf("%d ", vetor[i]);
```

```
}
```

Vetor v ainda não existe na memória!



```
#include <stdio.h>

const int TAM = 5;

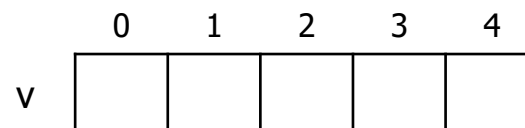
int * le_vetor(int tam) {
    int v[tam];

    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }

    return v;
}
```

```
int main()
{
    int *vetor = le_vetor(TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);
}
```





```
#include <stdio.h>

const int TAM = 5;

int * le_vetor(int tam) {
    int v[tam];

    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }

    return v;
}
```

```
int main()
{
    int *vetor = le_vetor(TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);
}
```

	0	1	2	3	4
v	2	5	9	4	3



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

```
const int TAM = 5;
```

```
int * le_vetor(int tam) {  
    int v[tam];
```

```
    for(int i = 0; i < tam; i++) {  
        printf("Digite um no.: ");  
        scanf("%d", &v[i]);  
    }
```

```
    return v;  
}
```

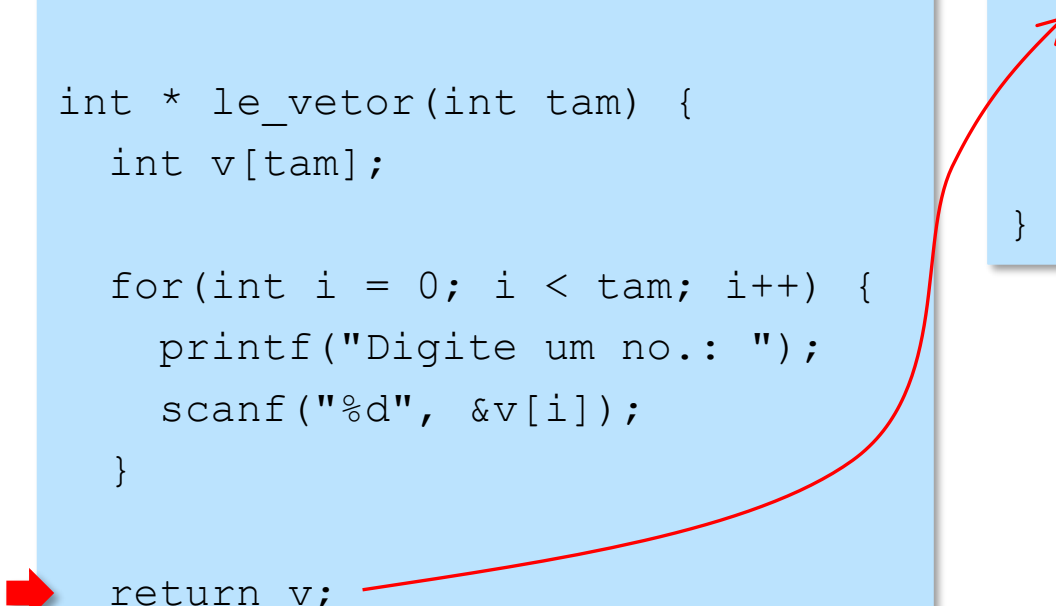
```
int main()
```

```
{
```

```
    int *vetor = le_vetor(TAM);
```

```
    for(int i = 0; i < TAM; i++)  
        printf("%d ", vetor[i]);
```

```
}
```



	0	1	2	3	4
v	2	5	9	4	3



```
#include <stdio.h>

const int TAM = 5;

int * le_vetor(int tam) {
    int v[tam];

    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }

    return v;
}
```

```
int main()
{
    int *vetor = le_vetor(TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);
}
```



Vetor v deixa de existir!

vetor **????**



Importante!

Existe a possibilidade de uma função criar e retornar um vetor: usando **ponteiros e alocação dinâmica de memória**, mas esse é um tópico avançado.





Então, como trabalhar de forma correta com funções que recebem vetores?

Regra 4: vetores são sempre passados por **referência**, ou seja, a função manipula o vetor real!



Então, como trabalhar de forma correta com funções que recebem vetores?

Regra 5: os vetores devem ser criados **antes** de serem passados para as funções. As funções vão apenas **ler** ou **alterar** os valores do vetor, mas **nunca** cria-los!



```
#include <stdio.h>

const int TAM = 5;

void le_vetor(int v[], int tam) {
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }
}

int main()
{
    int numeros[TAM];

    le_vetor(numeros, TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```

O vetor **numeros** é criado na função **main** e passado para a função **le_vetor**.



```
#include <stdio.h>

const int TAM = 5;

void le_vetor(int v[], int tam) {
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }
}

int main()
{
    int numeros[TAM];

    le_vetor(numeros, TAM);

    for(int i = 0; i < numeros; i++)
        printf("%d ", vetor[i]);
}
```

O parâmetro **v** referencia o vetor **numeros** passado para a função.



```
#include <stdio.h>

const int TAM = 5;

void le_vetor(int v[], int tam) {
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
    }
}

int main()
{
    int numeros[TAM];

    le_vetor(numeros, TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```

A função **le_vetor** apenas preenche o vetor **v** com os dados informados pelo usuário, ou seja, preenche o vetor **numeros**.



Situação 4: se a função deve retornar **um valor, primitivo** (int, char, float, double, bool, etc.) **ou struct**, mais um ou mais vetores:

- a) Use parâmetros para os vetores e retorne o valor com o comando **return** (mais indicado)
- b) Use parâmetros para os vetores e retorne o valor em uma **referencia**



Exemplo 4: função que lê um vetor e retorna o vetor preenchido e a sua média.



Solução A

```
#include <stdio.h>

const int TAM = 5;

float le_vetor(int v[], int tam) {
    float soma = 0;
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
        soma += v[i];
    }
    return soma / tam;
}
```

```
int main() {
    int numeros[TAM];
    float media;

    media = le_vetor(numeros, TAM);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```



Solução B

```
#include <stdio.h>

const int TAM = 5;

void le_vetor(int v[], int tam, float &media) {
    float soma = 0;
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
        soma += v[i];
    }
    media = soma / tam;
}
```

```
int main() {
    int numeros[TAM];
    float media;

    le_vetor(numeros, TAM, media);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```



Situação 5: se a função deve retornar **dois ou mais valores**, **primitivos** (int, char, float, double, bool, etc.) **ou struct**, mais um ou mais vetores:

- a) Use parâmetros para os vetores e retorne os valores em **referências**
- b) Use parâmetros para os vetores e retorne um dos valores com **return** e os demais em **referências**



Exemplo 5: função que lê um vetor e retorna o vetor preenchido, sua média, o menor e o maior valor.



Solução A

```
#include <stdio.h>

const int TAM = 5;

void le_vetor(int v[], int tam, float &media, int &menor, int &maior) {
    float soma = 0;
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
        soma += v[i];
        if (i == 0) {
            maior = v[0];
            menor = v[0];
        } else if (v[i] > maior)
            maior = v[i];
        else if (v[i] < menor)
            menor = v[i];
    }
    media = soma / tam;
}
```

```
int main() {
    int numeros[TAM];
    int menor, maior;
    float media;

    le_vetor(numeros, TAM, media, menor, maior);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```



Solução B

```
#include <stdio.h>

const int TAM = 5;

float le_vetor(int v[], int tam, int &menor, int &maior) {
    float soma = 0;
    for(int i = 0; i < tam; i++) {
        printf("Digite um no.: ");
        scanf("%d", &v[i]);
        soma += v[i];
        if (i == 0) {
            maior = v[0];
            menor = v[0];
        } else if (v[i] > maior)
            maior = v[i];
        else if (v[i] < menor)
            menor = v[i];
    }
    return soma / tam;
}
```

```
int main() {
    int numeros[TAM];
    int menor, maior;
    float media;

    media = le_vetor(numeros, TAM, menor, maior);

    for(int i = 0; i < TAM; i++)
        printf("%d ", numeros[i]);
}
```