

ESCOPO DE VARIÁVEIS E PASSAGEM DE PARÂMETROS

Prof. Muriel Mazzetto
Algoritmos 2

Escopo de Variáveis

2

- ❑ **Escopo da variável:** trecho de código em que a variável existe (visível/alterada diretamente).
- ❑ Local
- ❑ Global
- ❑ Local Estática

Escopo de Variáveis

3

- Variável **Local**:

- Declarada dentro de uma função (nos parâmetros ou no corpo);

Escopo de Variáveis

4

□ Variável **Local**:

- ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);
- ▣ O valor é acessado diretamente apenas dentro da função em que foi declarada.

Escopo de Variáveis

5

□ Variável **Local**:

- ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);
- ▣ O valor é acessado diretamente apenas dentro da função em que foi declarada.
- ▣ A variável deixa de existir quando a função terminar a execução;

Escopo de Variáveis

6

□ Variável **Local**:

- Declarada dentro de uma função (nos parâmetros ou no corpo);
- O valor é acessado diretamente apenas dentro da função em que foi declarada.
- A variável deixa de existir quando a função terminar a execução;
- O valor é perdido quando a função terminar a execução.

Escopo de Variáveis

7

- Variável **Global**:

- ▣ Declarada fora das funções (geralmente no início do arquivo, após os protótipos de função);

Escopo de Variáveis

8

□ Variável **Global**:

- ▣ Declarada fora das funções (geralmente no início do arquivo, após os protótipos de função);
- ▣ O valor é acessado diretamente **por qualquer função do arquivo** em que foi declarada.

Escopo de Variáveis

9

□ Variável **Global**:

- Declarada fora das funções (geralmente no início do arquivo, após os protótipos de função);
- O valor é acessado diretamente **por qualquer função** do arquivo em que foi declarada.
- A variável existe enquanto o **código** estiver sendo executado.

Escopo de Variáveis

10

□ Variável **Global**:

- Declarada fora das funções (geralmente no início do arquivo, após os protótipos de função);
- O valor é acessado diretamente **por qualquer função** do arquivo em que foi declarada.
- A variável existe enquanto o **código** estiver sendo executado.
- O valor nunca é perdido (mas pode ser alterado).

Escopo de Variáveis

11

□ Variável **Global**:

- Declarada fora das funções (geralmente no início do arquivo, após os protótipos de função);
- O valor é acessado diretamente **por qualquer função** do arquivo em que foi declarada.
- A variável existe enquanto o **código** estiver sendo executado.
- O valor nunca é perdido (mas pode ser alterado).
- Cuidado com nomes repetidos de variáveis locais e globais!

Escopo de Variáveis

12

- Variável **Local Estática**:
 - ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);

Escopo de Variáveis

13

- Variável **Local Estática**:
 - ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);
 - ▣ O valor é acessado diretamente apenas dentro da função em que foi declarada.

Escopo de Variáveis

14

□ Variável **Local Estática**:

- ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);
- ▣ O valor é acessado diretamente apenas dentro da função em que foi declarada.
- ▣ A variável não deixa de existir quando a função terminar a execução;

Escopo de Variáveis

15

□ Variável **Local Estática**:

- ▣ Declarada dentro de uma função (nos parâmetros ou no corpo);
- ▣ O valor é acessado diretamente apenas dentro da função em que foi declarada.
- ▣ A variável não deixa de existir quando a função terminar a execução;
- ▣ O valor não é perdido quando a função terminar a execução.

Escopo de Variáveis

16

□ Variável **Local Estática**:

- Declarada dentro de uma função (nos parâmetros ou no corpo);
- O valor é acessado diretamente apenas dentro da função em que foi declarada.
- A variável não deixa de existir quando a função terminar a execução;
- O valor não é perdido quando a função terminar a execução.
- Ao ser acessada, o valor continua a partir da última alteração.

Escopo de Variáveis

17

```
#include <stdio.h>

char nome[26];
int idade;

void le_nome(void)
{
    printf("Insira seu nome: ");
    scanf(" %s", nome);
}

void le_idade(void)
{
    printf("Insira sua idade: ");
    scanf("%d", &idade);
}

void imprime_dados(void)
{
    printf("Nome: %s \n", nome);
    printf("Idade: %d \n", idade);
}
```

```
void engordar(void)
{
    static int peso = 0;
    peso++;
    printf("Peso: %d\n", peso);
}

int main(void)
{
    char opcao;

    do
    {
        le_nome();
        le_idade();
        imprime_dados();
        engordar();

        printf("\nContinuar? s/n: ");
        scanf(" %c", &opcao);
        printf("\n~~~~~\n");

    }while(opcao == 's');

    return 0;
}
```

Escopo de Variáveis

18

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

```
char nome[26];  
int idade;
```

Variáveis Globais

```
void le_nome(void)
```

```
{  
    printf("Insira seu nome: ");  
    scanf(" %s", nome);  
}
```

```
void le_idade(void)
```

```
{  
    printf("Insira sua idade: ");  
    scanf("%d", &idade);  
}
```

```
void imprime_dados(void)
```

```
{  
    printf("Nome: %s \n", nome);  
    printf("Idade: %d \n", idade);  
}
```

```
void engordar(void)
```

```
{  
    static int peso = 0;  
    peso++;  
    printf("Peso: %d\n", peso);  
}
```

```
int main(void)
```

```
{  
    char opcao;  
  
    do  
    {  
        le_nome();  
        le_idade();  
        imprime_dados();  
        engordar();  
  
        printf("\nContinuar? s/n: ");  
        scanf(" %c", &opcao);  
        printf("\n~~~~~\n");  
  
    }while(opcao == 's');  
  
    return 0;  
}
```

Escopo de Variáveis

19

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

```
char nome[26];  
int idade;
```

```
void le_nome(void)
```

```
{  
    printf("Insira seu nome: ");  
    scanf(" %s", nome);  
}
```

```
void le_idade(void)
```

```
{  
    printf("Insira sua idade: ");  
    scanf("%d", &idade);  
}
```

```
void imprime_dados(void)
```

```
{  
    printf("Nome: %s \n", nome);  
    printf("Idade: %d \n", idade);  
}
```

Variável Local



```
void engordar(void)
```

```
{  
    static int peso = 0;  
    peso++;  
    printf("Peso: %d\n", peso);  
}
```

```
int main(void)
```

```
{  
    char opcao;  
  
    do  
    {  
        le_nome();  
        le_idade();  
        imprime_dados();  
        engordar();  
  
        printf("\nContinuar? s/n: ");  
        scanf(" %c", &opcao);  
        printf("\n~~~~~\n");  
  
    } while (opcao == 's');  
  
    return 0;  
}
```

Escopo de Variáveis

20

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

```
char nome[26];  
int idade;
```

```
void le_nome(void)  
{
```

```
    printf("Insira seu nome: ");  
    scanf(" %s", nome);  
}
```

```
void le_idade(void)  
{
```

```
    printf("Insira sua idade: ");  
    scanf("%d", &idade);  
}
```

```
void imprime_dados(void)  
{
```

```
    printf("Nome: %s \n", nome);  
    printf("Idade: %d \n", idade);  
}
```

**Variável Local
Estática**
Uso do *static* para
declarar

```
void engordar(void)  
{
```

```
    static int peso = 0;  
    peso++;  
    printf("Peso: %d\n", peso);  
}
```

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

```
    char opcao;
```

```
    do  
    {
```

```
        le_nome();  
        le_idade();  
        imprime_dados();  
        engordar();
```

```
        printf("\nContinuar? s/n: ");  
        scanf(" %c", &opcao);  
        printf("\n~~~~~\n");
```

```
    } while (opcao == 's');
```

```
    return 0;
```

```
}
```

Escopo de Variáveis

21

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

```
char nome[26];  
int idade;
```

```
void le_nome(void)
```

```
{  
    printf("Insira seu nome: ");  
    scanf(" %s", nome);  
}
```

```
void le_idade(void)
```

```
{  
    printf("Insira sua idade: ");  
    scanf("%d", &idade);  
}
```

```
void imprime_dados(void)
```

```
{  
    printf("Nome: %s \n", nome);  
    printf("Idade: %d \n", idade);  
}
```

"C:\Users\Muriel\Dropbox\1. UTFPR - DV\Algoritmo"

```
Insira seu nome: muriel  
Insira sua idade: 22  
Nome: muriel  
Idade: 22  
Peso: 1
```

```
Continuar? s/n: s
```

```
~~~~~
```

```
Insira seu nome: MURIEL  
Insira sua idade: 23  
Nome: MURIEL  
Idade: 23  
Peso: 2
```

```
Continuar? s/n: s
```

```
~~~~~
```

```
Insira seu nome: MURIEL_1  
Insira sua idade: 24  
Nome: MURIEL_1  
Idade: 24  
Peso: 3
```

```
Continuar? s/n: s
```

```
rdar(void)
```

```
c int peso = 0;
```

```
+
```

```
f("Peso: %d\n", peso);
```

```
void)
```

```
opcao;
```

```
le_nome();
```

```
le_idade();
```

```
imprime_dados();
```

```
ngordar();
```

```
rintf("\nContinuar? s/n: ");
```

```
canf(" %c", &opcao);
```

```
rintf("\n~~~~~\n");
```

```
e(opcao == 's');
```

```
n 0;
```

Escopo de Variáveis

22

```
#include <stdio.h>

char nome[26];

void le_nome(void)
{
    printf("Insira seu nome: ");
    scanf(" %s", nome);
}

void le_idade(void)
{
    int idade;
    printf("Insira sua idade: ");
    scanf("%d", &idade);
}

void imprime_dados(void)
{
    printf("Nome: %s \n", nome);
    printf("Idade: %d \n", idade);
}
```

```
void engordar(void)
{
    static int peso = 0;
    peso++;
    printf("Peso: %d\n", peso);
}

int main(void)
{
    char opcao;

    do
    {
        le_nome();
        le_idade();
        imprime_dados();
        engordar();

        printf("\nContinuar? s/n: ");
        scanf(" %c", &opcao);
        printf("\n~~~~~\n");

    }while(opcao == 's');

    return 0;
}
```

Escopo de Variáveis

23

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

```
char nome[26];
```

```
void le_nome(void)
```

```
{  
    printf("Insira seu nome: ");  
    scanf(" %s", nome);  
}
```

```
void le_idade(void)
```

```
{  
    int idade;  
    printf("Insira sua idade: ");  
    scanf("%d", &idade);  
}
```

```
void imprime_dados(void)
```

```
{  
    printf("Nome: %s \n", nome);  
    printf("Idade: %d \n", idade);  
}
```

```
void engordar(void)
```

```
{  
    static int peso = 0;  
    peso++;  
    printf("Peso: %d\n", peso);  
}
```

```
int main(void)
```

```
{  
    le_nome();  
    le_idade();  
    engordar();  
  
    printf("\nContinuar? s/n: ");  
    scanf(" %c", &opcao);  
    printf("\n~~~~~\n");  
  
    while(opcao == 's');  
  
    return 0;  
}
```

ERRO!

Variável é local na função.
Não está acessível fora da função
em que foi declarada.

Matrizes Globais

24

```
#include <stdio.h>
int linhas = 5;
int colunas = 10;
int matriz[5][10];

void preencher(void)
{
    int i, j;
    static int Valor = 3;

    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
        {
            matriz[i][j] = Valor;
            Valor += 2;
        }
    }
}
```

```
void imprimir(void)
{
    int i, j;
    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
            printf("%d\t", matriz[i][j]);
        printf("\n");
    }
}
```

```
int main(void)
{
    printf("Primeira matriz:\n");
    preencher();
    imprimir();

    printf("Segunda matriz:\n");
    preencher();
    imprimir();

    return 0;
}
```


Matrizes Globais

25

```
#include <stdio.h>
int linhas = 5;
int colunas = 10;
int matriz[5][10];

void preencher(void)
{
    int i, j;
    static int Valor = 3;

    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
        {
            matriz[i][j] = Valor;
            Valor += 2;
        }
    }
}
```

```
void imprimir(void)
{
    int i, j;
    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
        {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }
}
```

**FUNCIONA APENAS SE A
DECLARAÇÃO FOR
NUMÉRICA (sem variáveis).**

```
printf("Primeira matriz:\n");
preencher();
imprimir();

printf("Segunda matriz:\n");
preencher();
imprimir();

return 0;
```

Matrizes Globais

26

```
#include <stdio.h>
int linhas = 5;
int colunas = 10;
int matriz[5][10];

void preencher(void)
{
    int i, j;
    static int Valor = 3;

    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
        {
            matriz[i][j] = Valor;
            Valor += 2;
        }
    }
}
```

```
void imprimir(void)
{
    int i, j;
    for(i = 0; i < linhas; i++)
    {
        for(j = 0; j < colunas; j++)
            printf("%d\t", matriz[i][j]);
        printf("\n");
    }
}
```

```
int main(void)
{
    printf("Primeira matriz:\n");
    preencher();
    imprimir();

    printf("Segunda matriz:\n");
    preencher();
    imprimir();

    return 0;
}
```

**VARIÁVEIS ACESSÍVEIS EM
TODAS AS FUNÇÕES.**

Questionário

27

- 1) Defina os diferentes tipos de escopos e defina suas características.

Passagem de parâmetros

28

- Passagem de parâmetro: mecanismo para informar os valores que serão processados na função.

- A linguagem C permite duas formas:
 - ▣ Passagem por valor.
 - ▣ Passagem por referência.

Passagem de parâmetros

29

- Passagem **por valor**:
 - ▣ **Cópia do valor** para uma variável dos parâmetros;

- Passagem **por referência**:
 - ▣ Passagem do **endereço de memória** do dado original.

Passagem de parâmetros

30

- Passagem **por valor**:
 - ▣ **Cópia do valor** para uma variável dos parâmetros;
 - Alterar a variável na função, não altera o dado original.

- Passagem **por referência**:
 - ▣ Passagem do **endereço de memória** do dado original.
 - Alterar a variável na função, altera o dado original.

Passagem de parâmetros

31

□ Passagem por valor:

```
#include <stdio.h>

void divisores(int num)
{
    int i = 1;
    printf("Divisores de %d: ", num);
    for(i = 1; i <= num; i++)
    {
        if( (num%i) == 0 ) printf("%d ", i);
    }

    printf("\n\n");
}

int main(void)
{
    int valor = 10;
    divisores(valor);
    divisores(50);
    divisores(35);
    valor = 7;
    divisores(valor);

    return 0;
}
```

Passagem de parâmetros

32

□ Passagem **por valor**:

```
#include <stdio.h>

void troca(int A, int B)
{
    int aux;

    aux = A;
    A = B;
    B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(X, Y);

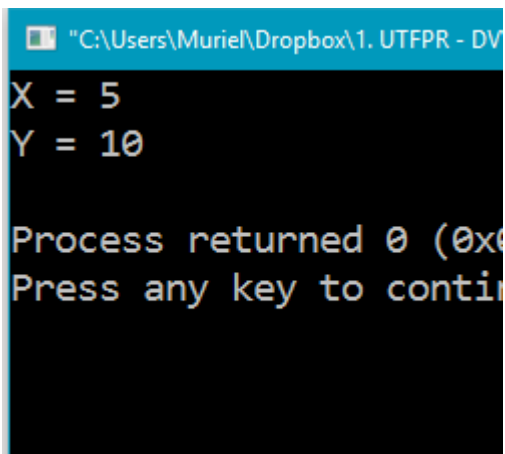
    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```


Passagem de parâmetros

33

□ Passagem **por valor**:



```
"C:\Users\Muriel\Dropbox\1. UTFPR - DV
X = 5
Y = 10

Process returned 0 (0x0)
Press any key to continue
```

```
#include <stdio.h>

void troca(int A, int B)
{
    int aux;

    aux = A;
    A = B;
    B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(X, Y);

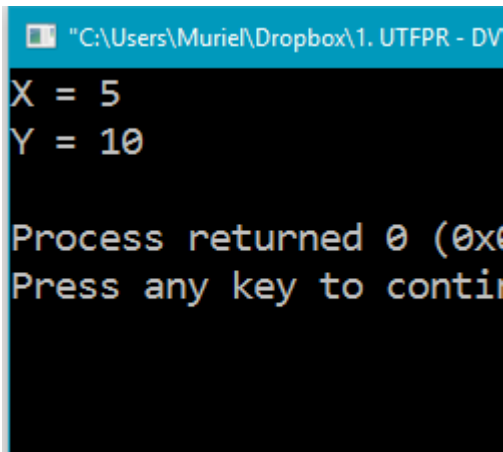
    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

34

- ❑ Passagem **por valor**:
 - ▣ Copiou valores de X e Y para A e B.
 - ▣ Realizou a troca apenas dentro da função.
 - ▣ Não alterou os originais.



```
"C:\Users\Muriel\Dropbox\1. UTFPR - DV"
X = 5
Y = 10

Process returned 0 (0x0)
Press any key to continue
```

```
#include <stdio.h>

void troca(int A, int B)
{
    int aux;

    aux = A;
    A = B;
    B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(X, Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

35

- Passagem **por referência**:
 - ▣ Utiliza operadores para trabalhar com endereço da variável.

Passagem de parâmetros

36

□ Passagem **por referência**:

- ▣ Utiliza operadores para trabalhar com endereço da variável.
- ▣ & - “endereço de” é utilizado para fornecer o endereço de memória de uma variável.
- ▣ * - “valor de” é utilizado para declarar uma variável de endereço e para acessar o seu valor.

Passagem de parâmetros

37

□ Passagem **por referência**:

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

38

□ Passagem **por referência**:

Passando o **ENDEREÇO** de cada variável.



```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

39

□ Passagem **por referência**:

Variáveis especiais para receber **endereço de memória**.

Declarar usando o tipo do dado e * para indicar que armazenará um endereço.

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

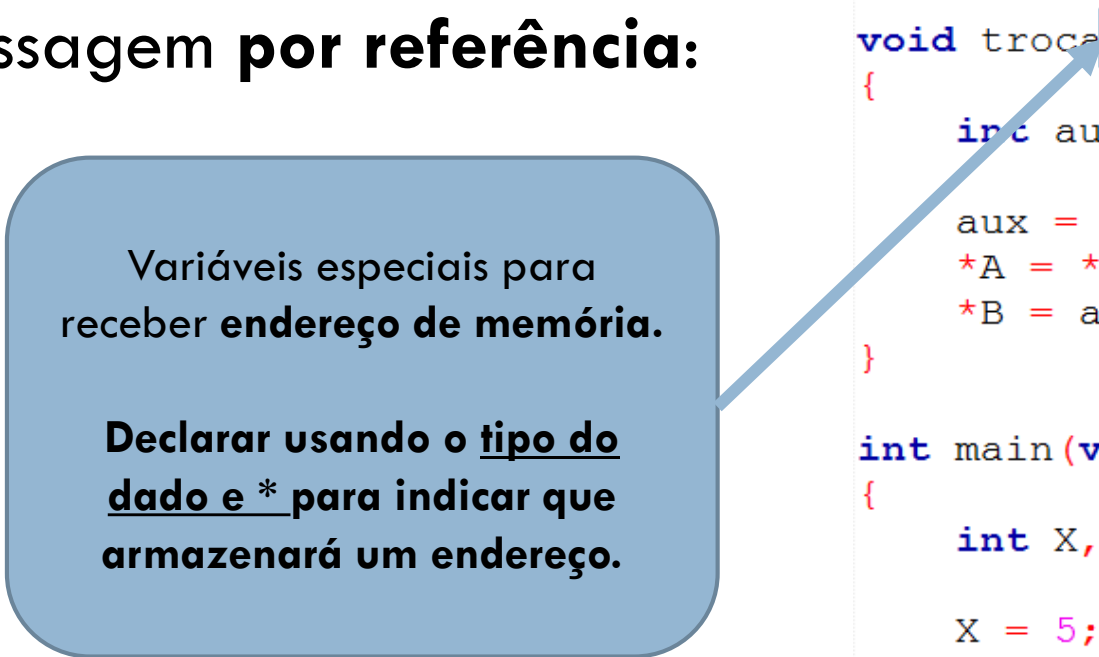
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```



Passagem de parâmetros

40

□ Passagem **por referência**:

Acessando o **VALOR** de cada variável.

Sempre usar ***** para acessar o valor nessas variáveis.

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```


Passagem de parâmetros

41

□ Passagem **por referência**:

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

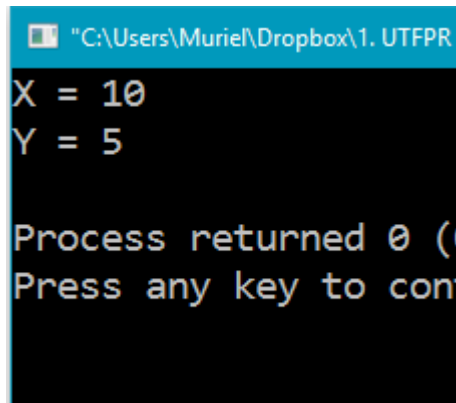
    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

42

□ Passagem por referência:



```
"C:\Users\Muriel\Dropbox\1. UTFPR"
X = 10
Y = 5

Process returned 0 (0x0)
Press any key to continue...
```

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

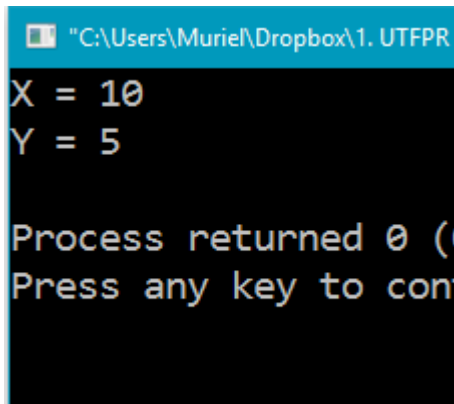
    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

43

- Passagem **por referência**:
 - ▣ Passou o endereço de X e Y para A e B.
 - ▣ Alterou o valor dentro dos endereços (*A e *B).
 - ▣ Alterou o conteúdo dos originais (X e Y).



```
"C:\Users\Muriel\Dropbox\1. UTFPR"
X = 10
Y = 5

Process returned 0 (0x0)
Press any key to continue...
```

```
#include <stdio.h>

void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Passagem de parâmetros

44

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

Passagem de parâmetros

45

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    → int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|--|
| 0x002 | X | |
| 0x005 | Y | |

Passagem de parâmetros

46

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    → X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|---|
| 0x002 | X | 5 |
| 0x005 | Y | |

Passagem de parâmetros

47

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    → Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

Passagem de parâmetros

48

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    → troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

Passagem de parâmetros

49

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

Função
chamadora entra
em espera

Passagem de parâmetros

```
50 #include <stdio.h>

→ void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}

int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    II → troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Inicia função
declarando
parâmetros...

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|---|--|
| 0x029 | A | |
| 0x012 | B | |

Passagem de parâmetros

51 `#include <stdio.h>`

→ `void troca(int* A, int* B)`

```
{  
    int aux;  
  
    aux = *A;  
    *A = *B;  
    *B = aux;  
}
```

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

```
    int X, Y;
```

```
    X = 5;  
    Y = 10;
```

... e inicializa com valores recebidos

II → `troca(&X, &Y);`

```
    printf("X = %d\n", X);  
    printf("Y = %d\n", Y);
```

```
    return 0;  
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

52

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

```
void troca(int* A, int* B)
{
    → int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|--|
| 0x035 | aux | |
|-------|-----|--|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

53

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

```
void troca(int* A, int* B)
{
    int aux;

    → aux = *A;
      *A = *B;
      *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|--|
| 0x035 | aux | |
|-------|-----|--|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

54

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

```
void troca(int* A, int* B)
{
    int aux;

    → aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

```
II → troca(&X, &Y);
```

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Acessado o **valor**
dentro do
endereço que A
armazena

Passagem de parâmetros

55

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    → *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

56

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    → *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    II → troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

Acessando os
valores dentro
do endereço que
A e B armazenam

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 5 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

57

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    → *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 10 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

58

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    → *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;
```

II → `troca(&X, &Y);`

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

59

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```



```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

```

II

```
→ troca(&X, &Y);
```

```
printf("X = %d\n", X);
printf("Y = %d\n", Y);

return 0;
}
```

Fim da execução
da função.
**Variáveis locais
são desalocadas.**

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |

| | | |
|-------|-----|---|
| 0x035 | aux | 5 |
|-------|-----|---|

| | | |
|-------|---|-------|
| 0x029 | A | 0x002 |
| 0x012 | B | 0x005 |

Passagem de parâmetros

60

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;
```

```
    X = 5;
    Y = 10;
```

```
    → troca(&X, &Y);
```

```
    printf("X = %d\n", X);
    printf("Y = %d\n", Y);
```

```
    return 0;
```

```
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |

Função
chamadora volta
à execução

Passagem de parâmetros

61

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

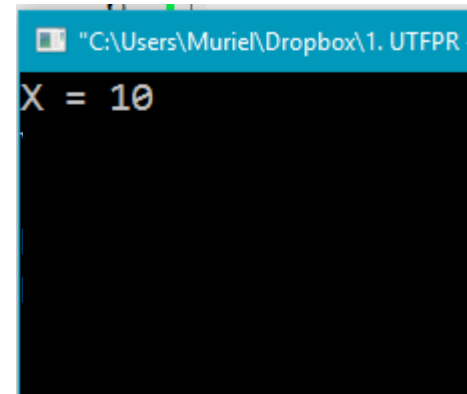
    troca(&X, &Y);

    → printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |



```
"C:\Users\MurIEL\Dropbox\1. UTFPR -
X = 10
```

Passagem de parâmetros

62

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    → printf("Y = %d\n", Y);

    return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |

```
"C:\Users\MurIEL\Dropbox\1. UTFPR-
X = 10
Y = 5

Process returned 0 (0
Press any key to cont
```

Passagem de parâmetros

63

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

```
void troca(int* A, int* B)
{
    int aux;

    aux = *A;
    *A = *B;
    *B = aux;
}
```

```
int main(void)
{
    int X, Y;

    X = 5;
    Y = 10;

    troca(&X, &Y);

    printf("X = %d\n", X);
    printf("Y = %d\n", Y);

    → return 0;
}
```

MEMÓRIA

| | | |
|-------|---|----|
| 0x002 | X | 10 |
| 0x005 | Y | 5 |

Fim da execução
do código

```
"C:\Users\Muriele\Dropbox\1. UTFPR-
X = 10
Y = 5

Process returned 0 (0
Press any key to cont
```

Questionário

64

- 2) Defina os diferentes tipos de passagem de parâmetros e descreva suas características.
- 3) Que tipo de vantagem o uso de parâmetros por referência traz ao uso das funções em C?

Vetores como parâmetros

65

- Os vetores são **sempre passados por referência**.
 - ▣ Alterar o valor de uma posição dentro da função irá alterar o valor do dado original.
 - ▣ Evita cópias de grandes quantidades de dados para outra posição da memória.

Vetores como parâmetros

66

- Os vetores são **sempre passados por referência**.
 - ▣ Alterar o valor de uma posição dentro da função irá alterar o valor do dado original.
 - ▣ Evita cópias de grandes quantidades de dados para outra posição da memória.
- As funções precisam receber apenas a **primeira posição do vetor e o seu tamanho**.
- A manipulação dos índices continua igual.

Vetores como parâmetros

67

- Regras para passagem de vetores:
 - ▣ O **nome** do vetor referencia o seu **primeiro endereço**.
 - ▣ Sempre receber em uma **variável de endereço do mesmo tipo**.
 - ▣ Sempre passar o **tamanho** do vetor.
 - ▣ Exemplo de como receber vetores nos parâmetros:

```
void exemplo_1(int* vetor, int tamanho)
void exemplo_2(int vetor[], int tamanho)
void exemplo_3(int vetor[10], int tamanho)
```

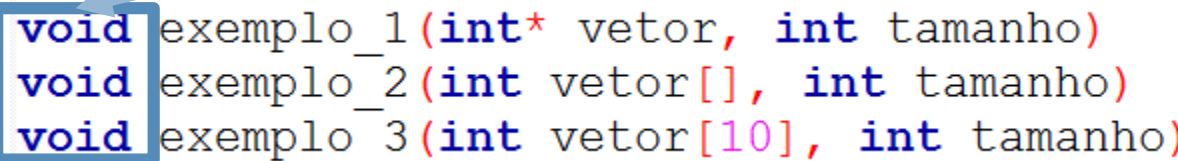
Vetores como parâmetros

68

□ Regras para passagem de vetores:

- O nome do vetor é o primeiro endereço.
- Sempre r... de endereço do **mesmo** tipo.
- Sempre p... r.
- Exemplo c... nos parâmetros:

Independente do tipo do retorno, pode ser o tipo que o problema necessitar.



```
void exemplo_1(int* vetor, int tamanho)
void exemplo_2(int vetor[], int tamanho)
void exemplo_3(int vetor[10], int tamanho)
```

Vetores como parâmetros

69

□ Regras para passagem de vetores:

- O nome do vetor é o primeiro endereço.
- Sempre passar o endereço de início de endereço do vetor.
- Sempre passar o tamanho do vetor.
- Exemplo de como declarar os parâmetros:

Após declarar os vetores, passar o restante dos parâmetros que o problema necessita, independente da ordem.

```
void exemplo_1(int* vetor, int tamanho)
void exemplo_2(int vetor[], int tamanho)
void exemplo_3(int vetor[10], int tamanho)
```

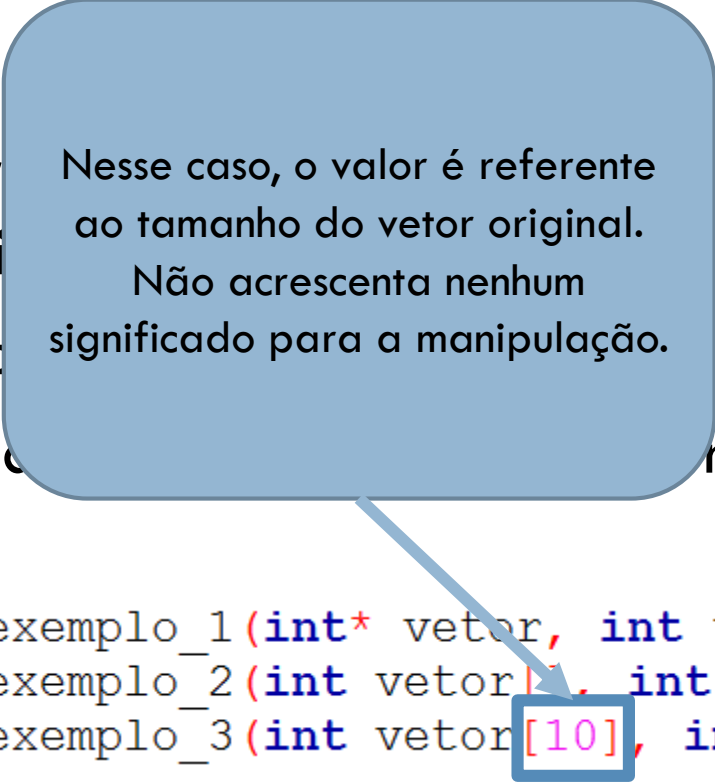
Vetores como parâmetros

70

□ Regras para passagem de vetores:

- O nome do vetor é o primeiro endereço.
- Sempre passar o tamanho do vetor. Nesse caso, o valor é referente ao tamanho do vetor original. Não acrescenta nenhum significado para a manipulação.
- Sempre passar o mesmo tipo de dados.
- Exemplo de como declarar os parâmetros:

```
void exemplo_1(int* vetor, int tamanho)  
void exemplo_2(int vetor[], int tamanho)  
void exemplo_3(int vetor[10], int tamanho)
```



Vetores como parâmetros

71

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

```
void preencher(int* vetor, int tamanho)
```

```
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

```
void imprimir(int* vet, int tamanho)
```

```
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
}
```

```
int main(void)
```

```
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

Vetores como parâmetros

72

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA



Vetores como parâmetros

73

```
int main(void)
{
    → int tam = 10;
      int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

Vetores como parâmetros

74

```
int main(void)
{
    int tam = 10;
    → int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA

| 0x002 | |
|-------|----|
| tam | 10 |

| 0x543 | |
|-------|--|
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

75

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    → preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA

| 0x002 | |
|-------|----|
| tam | 10 |

| 0x543 | |
|-------|--|
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

76

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    II → preencher(VET, tam);
        imprimir(VET, tam);

    return 0;
}
```

Função
chamadora entra
em espera

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-----|
| 0x543 | VET |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

77

→

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

Inicia função
declarando
parâmetros...

MEMÓRIA

| | |
|-------|----|
| 0x002 | |
| tam | 10 |

| | |
|-------|--|
| 0x038 | |
| vetor | |

| | |
|---------|--|
| 0x042 | |
| tamanho | |

| | |
|-------|--|
| 0x543 | |
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

78

→

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

... e inicializar
com valores
recebidos

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|--|
| 0x543 | |
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

79

```
void preencher(int* vetor, int tamanho)
{
    → int i;
    for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|--|
| 0x043 | |
| i | |

| | |
|-------|--|
| 0x543 | |
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

80

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 0 |
| i | |

| | |
|-------|--|
| 0x543 | |
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

81

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 0 |
| i | |

| | |
|-------|--|
| 0x543 | |
| VET | |
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

82

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        → vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 0 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

83

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 1 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

84

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 1 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

85

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        → vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 1 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

86

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 2 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

87

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 2 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

88

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        → vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 2 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

89

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 3 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

90

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 3 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

91

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        → vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 3 |
| i | |

| | |
|-------|---|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Vetores como parâmetros

92

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

APÓS ALGUMAS
ITERAÇÕES

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 9 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | |

Vetores como parâmetros

93

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 9 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | |

Vetores como parâmetros

94

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        → vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|---|
| 0x043 | 9 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

95

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

APÓS ALGUMAS
ITERAÇÕES

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|----|
| 0x043 | 10 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

96

```
void preencher(int* vetor, int tamanho)
{
    int i;
    → for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|----|
| 0x043 | 10 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

97

```
void preencher(int* vetor, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        vetor[i] = i*2;
    }
}
```

Fim da execução.
**Variáveis locais
são desalocadas.**

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vetor | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|----|
| 0x043 | 10 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

98

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    → preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

Função
chamadora volta
à execução

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

99

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    → imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA

| 0x002 | |
|-------|----|
| tam | 10 |

| 0x543 | |
|-------|----|
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

100

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    return 0;
}
```

||

→

Função
chamadora entra
em espera

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

101

→

```
void imprimir(int* vet, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
}
```

Inicia função
declarando
parâmetros...

MEMÓRIA

| | |
|-------|----|
| 0x002 | |
| tam | 10 |

| | |
|-------|--|
| 0x038 | |
| vet | |

| | |
|---------|--|
| 0x042 | |
| tamanho | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

102

→

```
void imprimir(int* vet, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
}
```

... e inicializar
com valores
recebidos

MEMÓRIA

| | |
|-------|----|
| 0x002 | |
| tam | 10 |

| | |
|-------|-------|
| 0x038 | |
| vet | 0x543 |

| | |
|---------|----|
| 0x042 | |
| tamanho | 10 |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

103

```
void imprimir(int* vet, int tamanho)
{
    → int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
}
```

MEMÓRIA

| | |
|-------|----|
| 0x002 | |
| tam | 10 |

| | |
|-------|-------|
| 0x038 | |
| vet | 0x543 |

| | |
|---------|----|
| 0x042 | |
| tamanho | 10 |

| | |
|-------|--|
| 0x043 | |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

104

```
void imprimir(int* vet, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
}
```

MEMÓRIA

| | | | |
|-------|----|-------|-----|
| 0x002 | 10 | 0x543 | VET |
| tam | | | |
| | | | 0 |
| | | | 2 |
| | | | 4 |
| 0x | | | 6 |
| | | | 8 |
| 0x | | | 10 |
| tam | | | 12 |
| | | | 14 |
| 0x | | | 16 |
| | | | 18 |
| | | 9 | 18 |

Vetores como parâmetros

105

```
void imprimir(int* vet, int tamanho)
{
    int i;
    for(i = 0; i < tamanho; i++)
    {
        printf("VETOR[%d] = %d\n", i, vet[i]);
    }
} ←
```

Fim da execução.
Variáveis locais
são desalocadas.

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|-------|
| 0x038 | 0x543 |
| vet | |

| | |
|---------|----|
| 0x042 | 10 |
| tamanho | |

| | |
|-------|----|
| 0x043 | 10 |
| i | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

106

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    → imprimir(VET, tam);

    return 0;
}
```

Função
chamadora volta
à execução

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

107

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    → imprimir(VET, tam);

    return 0;
}
```

MEMÓRIA

| 0x002 | |
|-------|----|
| tam | 10 |

| 0x543 | |
|-------|----|
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Vetores como parâmetros

108

```
int main(void)
{
    int tam = 10;
    int VET[tam];

    preencher(VET, tam);
    imprimir(VET, tam);

    → return 0;
}
```

Fim da execução
do código

MEMÓRIA

| | |
|-------|----|
| 0x002 | 10 |
| tam | |

| | |
|-------|----|
| 0x543 | |
| VET | |
| 0 | 0 |
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |
| 6 | 12 |
| 7 | 14 |
| 8 | 16 |
| 9 | 18 |

Questionário

109

- 4) Quais as regras para passagem de vetores como parâmetros?
- 5) É necessário enviar o tamanho em um vetor de char?
- 6) Escreva uma função que receba um vetor de char e desloque -3 em cada caractere da tabela ASCII.

Matrizes como parâmetros

110

- As matrizes também são passadas por referências.
 - ▣ Alterar o valor de uma posição dentro da função irá alterar o valor do dado original.
- As funções precisam receber apenas a **primeira posição da matriz e a quantidade de linhas e de colunas**.
- A manipulação dos índices continua igual.

Matrizes como parâmetros

111

- Regras para passagem de matrizes:
 - ▣ O **nome** da matriz referencia o seu **primeiro endereço**.
 - ▣ Sempre receber em uma **variável de endereço do mesmo tipo**.
 - ▣ Sempre passar a **quantidade de linhas e de colunas**.
 - ▣ Exemplo de como receber matrizes nos parâmetros:

```
void exemplo_1(int matriz[][10], int L, int C)  
void exemplo_2(int **matriz, int L, int C)
```

Matrizes como parâmetros

112

- ❑ Regras para passagem de matrizes:
 - ❑ O nome da matriz é o primeiro endereço.
 - ❑ Sempre passar o mesmo tamanho de endereço do mesmo tipo.
 - ❑ Sempre passar o número de linhas e de colunas.
 - ❑ Exemplo de passagem nos parâmetros:

Independente do tipo do retorno, pode ser o tipo que o problema necessitar.

```
void exemplo_1(int matriz[][10], int L, int C)
void exemplo_2(int **matriz, int L, int C)
```


Matrizes como parâmetros

113

□ Regras para passagem de matrizes:

- O nome da matriz é o primeiro endereço.
- Sempre passar o endereço de início de endereço do mesmo tipo.
- Sempre passar o número de linhas e de colunas.
- Exemplo de passagem nos parâmetros:

Após declarar as matrizes, passar o restante dos parâmetros que o problema necessita, independente da ordem.

```
void exemplo_1(int matriz[][10], int L, int C)  
void exemplo_2(int **matriz, int L, int C)
```

Matrizes como parâmetros

114

- ❑ Regras para passagem de matrizes:
 - ❑ O nome da matriz deve ser o primeiro endereço.
 - ❑ Sempre colocar a quantidade de endereços de endereço do mesmo tipo.
 - ❑ Sempre colocar o tamanho das linhas e de colunas.
 - ❑ Exemplo de como colocar nos parâmetros:

OBRIGATÓRIO colocar a quantidade de colunas.
Impossibilita generalizar para tamanhos diferentes.

```
void exemplo_1(int matriz[][10], int L, int C)  
void exemplo_2(int **matriz, int L, int C)
```

Matrizes como parâmetros

115

- ❑ Regras para passagem de matrizes:
 - ❑ O nome da matriz é o primeiro endereço.
 - ❑ Sempre passar o endereço de endereço do mesmo tipo.
 - ❑ Sempre passar os tamanhos e de colunas.
 - ❑ Exemplo de como usar nos parâmetros:

**APENAS COM ALOCAÇÃO
DINÂMICA.**

Pode generalizar para
tamanhos diferentes.

```
void exemplo_1(int matriz[][10], int L, int C)  
void exemplo_2(int **matriz, int L, int C)
```

Matrizes como parâmetros

116

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

```
void preencher(int matriz[][10], int L, int C)
{
```

```
    int i, j;
```

```
    static int Valor = 3;
```

```
    for(i = 0; i < L; i++)
```

```
    {
```

```
        for(j = 0; j < C; j++)
```

```
        {
```

```
            matriz[i][j] = Valor;
```

```
            Valor += 2;
```

```
        }
```

```
    }
```

```
}
```

```
void imprimir(int m[][10], int L, int C)
```

```
{
```

```
    int i, j;
```

```
    for(i = 0; i < L; i++)
```

```
    {
```

```
        for(j = 0; j < C; j++)
```

```
            printf("%d\t", m[i][j]);
```

```
        printf("\n");
```

```
    }
```

```
}
```

```
int main(void)
```

```
{
```

```
    int linhas = 5;
```

```
    int colunas = 10;
```

```
    int MAT[linhas][colunas];
```

```
    printf("Primeira matriz:\n");
```

```
    preencher(MAT, linhas, colunas);
```

```
    imprimir(MAT, linhas, colunas);
```

```
    printf("Segunda matriz:\n");
```

```
    preencher(MAT, linhas, colunas);
```

```
    imprimir(MAT, linhas, colunas);
```

```
    return 0;
```

```
}
```