

# Estruturas de Dados

## Introdução e Revisão

Universidade Estadual Vale do Acaraú – UVA

---

Paulo Regis Menezes Sousa

paulo\_regis@uvanet.br

# Introdução

## Revisão

Arrays

Ponteiros

## Funções

Passagem de Parâmetros por Valor e por Referência

Módulos de Programas em C

Arquivos de Cabeçalho

Compilação de módulos

- Assuntos das unidades:
  - Unidade 1 Revisão
  - Unidade 2 Estruturas Lineares
  - Unidade 3 Estruturas Hierárquicas
  - Unidade 4 Grafos

- Para **declarar um array** se especifica um tipo, um nome e um número de elementos.

```
int v[12];
```

- Pode-se declarar vários arrays com uma única linha.

```
int a[10], b[15];
```

- Pode-se declarar arrays com  $n$  dimensões através da seguinte sintaxe:

```
tipo nome[dim1][dim2] ... [dim $n$ ];
```

- Os **ponteiros** são variáveis que contêm endereços de memória como valores.
- Os ponteiros, como quaisquer outras variáveis, devem ser declarados antes de serem usados.

```
int *contPtr, cont;
```

- Quando o \* é usado dessa forma em uma declaração, ele indica que a variável que está sendo declarada é um ponteiro.
- Os ponteiros podem ser declarados para apontar objetos de qualquer tipo de dados.

```
int *p1;
```

```
char *p2;
```

```
float *p3;
```

- Os ponteiros devem ser inicializados ao serem declarados ou em uma instrução de atribuição.
- Um ponteiro pode ser inicializado com 0, NULL ou um endereço.
- Um ponteiro com o valor NULL não aponta para lugar algum.
- NULL é uma constante simbólica definida no arquivo de cabeçalho `<stdio.h>`.
- O valor 0 é o único valor inteiro que pode ser atribuído diretamente a uma variável de ponteiro.

- O `&` ou operador de ponteiro é um operador unário que retorna o endereço de seu operando. Por exemplo, admitindo as declarações

```
1 int y = 5;  
2 int *yPtr;
```

a instrução

```
1 yPtr = &y;
```

atribui o endereço da variável `y` à variável de ponteiro `yPtr`. Diz-se que a variável `yPtr` “aponta para” `y`.

- O operador `*`, chamado frequentemente **operador de referência indireta** ou **operador de desreferenciamento**, retorna o valor do objeto ao qual seu operando (i.e., um ponteiro) aponta. Por exemplo, a instrução

```
printf("%d", *yPtr);
```

imprime o valor da variável `y`, ou seja, 5. Usar `*` dessa maneira é chamado *desreferenciar* um poteiro.

### Warning!

Desreferenciar um ponteiro que não foi devidamente inicializado ou que não foi atribuído para apontar para um local específico da memória. Isso poderia causar um erro fatal de tempo de execução, ou poderia modificar acidentalmente dados importantes e permitir que o programa seja executado até o final fornecendo resultados incorretos.



- Considere um programa que use a função **square** para calcular os quadrados dos números inteiros de 1 a 10.

```
1  #include <stdio.h>
2
3  int square(int); /* protótipo da funcao */
4
5  int main(){
6      int x;
7      for (x = 1; x <= 10; x++)
8          printf("%d ", square(x));
9      return 0;
10 }
11
12 /* Definição da funcao */
13 int square(int y){
14     return y * y;
15 }
```

- Uma função definida pelo programador denominada **maximum** para determinar e retornar o maior entre três inteiros.

```
1  #include <stdio.h>
2  /* Encontrar o maior de tres inteiros */
3  int maximum(int, int, int);
4
5  int main(){
6      int a, b, c;
7      printf("Entre com tres inteiros: ");
8      scanf("%d%d%d", &a, &b, &c);
9      printf("O maior e: %d\n", maximum(a, b, c));
10     return 0;
11 }
12
13 int maximum(int a, int b, int c) {
14     if (a > b && a > c) return a;
15     if (b > a && b > c) return b;
16     if (c > a && c > b) return c;
17 }
```

### Exercício 1

Faça um programa contendo uma função que receba dois números positivos e retorne a soma dos  $N$  números pares existentes entre eles (inclusive). Exemplo:  $(2,7) \rightarrow 2 + 4 + 6 = 12$ .

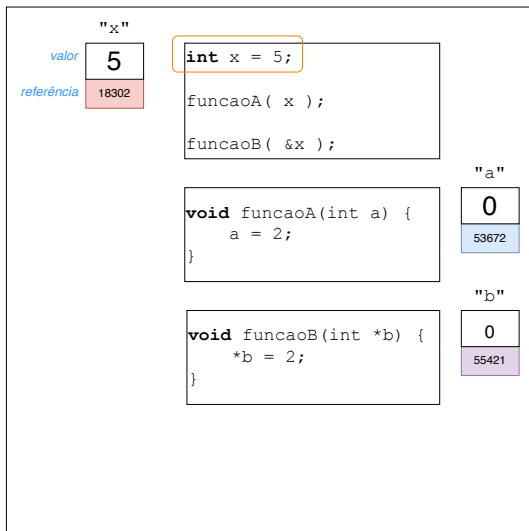
### Exercício 2

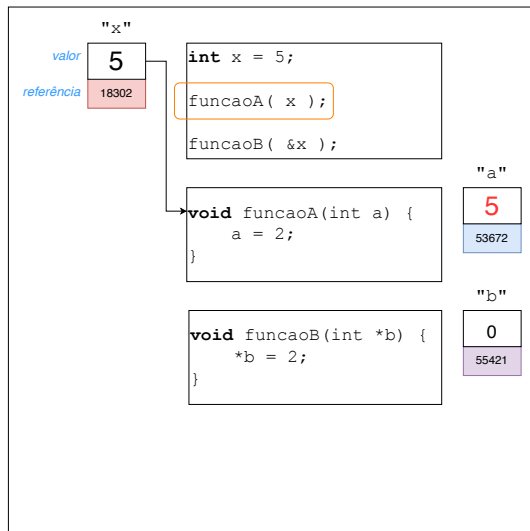
Crie uma função que receba como parâmetro dois valores inteiros  $b$  e  $e$ , calcule e retorne  $b^e$ .

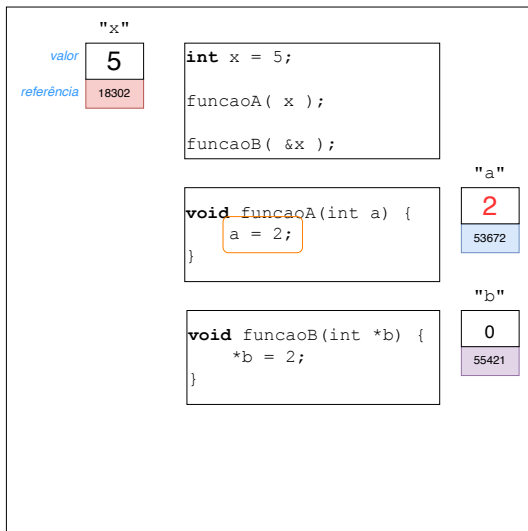
- As duas maneiras de ativar as funções em muitas linguagens de programação são chamadas por **valor** e chamadas por **referência**.

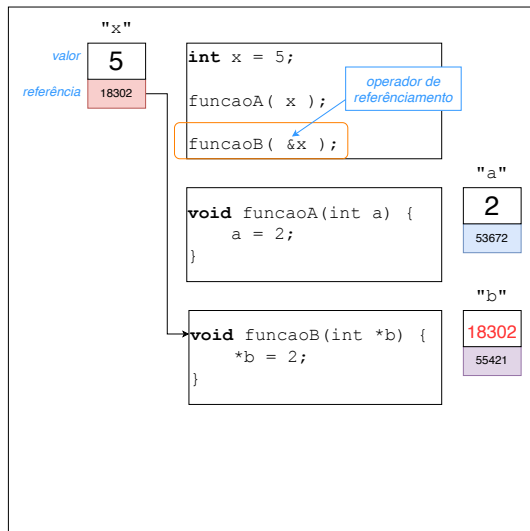
```
1 int parametroPorValor(int x)
```

```
1 int parametroPorReferencia(int *x)
```

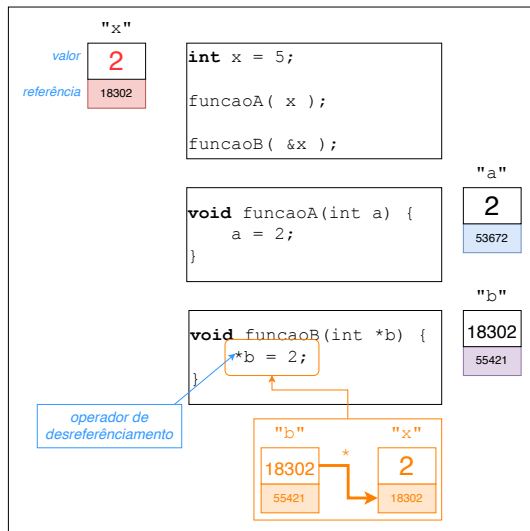












```
1  #include <stdio.h>
2
3  int soma(int a, int b);
4
5  int main(){
6      int x = 5, y = 4, z;
7
8      z = soma(x, y);
9      printf("%d + %d = %d\n", x, y, z);
10
11     return 0;
12 }
13
14 int soma(int a, int b){
15     return a + b;
16 }
```

```
1  #include <stdio.h>
2
3  void soma(int a, int b, int *z);
4
5  int main(){
6      int x = 4, y = 5, z;
7
8      soma(x, y, &z);
9      printf("%d + %d = %d\n", x, y, z);
10
11     return 0;
12 }
13
14 void soma(int a, int b, int *Z){
15     *z = a + b;
16 }
```

```
1  #include <stdio.h>
2
3  void anteSuce(int n, int *a, int *s);
4
5  int main(){
6      int numero, antecessor, sucessor;
7
8      printf("Digite um número inteiro: ");
9      scanf("%d", &numero);
10
11     anteSuce(numero, &antecessor, &sucessor);
12     printf("%d <- %d -> %d\n", antecessor, numero, sucessor);
13     return 0;
14 }
15 void anteSuce(int n, int *a, int *s){
16     *a = n - 1;
17     *s = n + 1;
18 }
```

## Exercício 3

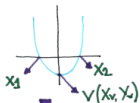
Crie uma função que receba como parâmetro:  $a$ ,  $b$  e  $c$  por valor e  $x_1$  e  $x_2$  por referência, todos os parâmetros reais.

A função deve calcular as raízes de uma equação do segundo grau com coeficientes  $a$ ,  $b$  e  $c$  e calcular as raízes  $x_1$  e  $x_2$ .

O retorno da função deverá ser um número inteiro indicando a quantidade de raízes da função.

Válida para:  
EQUAÇÕES DO 2º GRAU

$$ax^2 + bx + c = 0$$



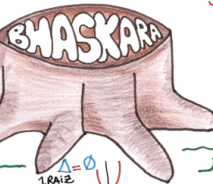
COORDENADAS DO VÉRTICE

$$X_v = \frac{-b}{2a}$$

$$Y_v = \frac{-\Delta}{4a}$$

# Fórmula

de



$\Delta = 0$   
1 RAÍZ

$\Delta > 0$   
2 RAÍZES

Estudo das raízes

Como resolver  
1 → DISCRIMINANTE (DELTA)

$$\Delta = b^2 - 4ac$$

2 → BHASKARA

$$X = \frac{-b \pm \sqrt{\Delta}}{2a}$$

3 → RESULTADO (RAÍZES)

$$X_1 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$X_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

$\Delta < 0$   
0 RAÍZES



- Programas em C, quase sempre, são escritos usando funções disponíveis na biblioteca padrão do C (*C standard library*).
- A C standard library é formada por diversas funções de uso comum, tais como:

Cabeçalho	Descrição
<math.h>	Contém protótipos de funções matemáticas
<stdlib.h>	Contém protótipos de funções para conversão de números em texto e texto em números, alocação de memória, números aleatórios entre outras
<string.h>	Contém protótipos de funções para processamento de strings.
<time.h>	Contém protótipos de funções para manipular horários e datas.

Tabela 1: Exemplos de bibliotecas da linguagem C

- Os arquivos de extensão `.h` são chamados de **arquivos de cabeçalho** (ou *headers*).

Função	Descrição
<code>sqrt(x)</code>	Raiz quadrada de $\sqrt{x}$
<code>exp(x)</code>	Exponencial ( $e^x$ )
<code>log(x)</code>	Logaritmo natural ( $\log_e(x)$ )
<code>log10(x)</code>	Logaritmo na base 10 ( $\log_{10}(x)$ )
<code>fabs(x)</code>	Valor absoluto $  x  $
<code>ceil(x)</code>	Teto ( $\lceil x \rceil$ )
<code>floor(x)</code>	Piso ( $\lfloor x \rfloor$ )
<code>pow(x,y)</code>	Potência ( $x^y$ )
<code>sin(x)</code>	Seno ( $\text{sen}(x)$ )
<code>cos(x)</code>	Cosseno ( $\cos(x)$ )
<code>tan(x)</code>	Tangente ( $\tan(x)$ )

Tabela 2: Exemplos de funções matemáticas da biblioteca `math.h`

- O programador pode criar arquivos de cabeçalho personalizados.
- Um header definido pelo programador pode ser incluído usando a diretiva de pré-processador `#include`.
- **Por exemplo**, o arquivo de cabeçalho `FuncoesUteis.h` pode ser incluído em nosso programa da seguinte forma

```
#include "FuncoesUteis.h"
```



- Os headers contém definições de protótipos de função que você pode adicionar à sua implementação.
- Mas eles precisam acompanhados de um arquivo de implementação que (com extensão .c):

`FuncoesUteis.c`

### Código 1: FuncoesUteis.h

```
1 void Int_printArray(int v[], int length);
2 void Float_printArray(float v[], int length);
```

### Código 2: FuncoesUteis.c

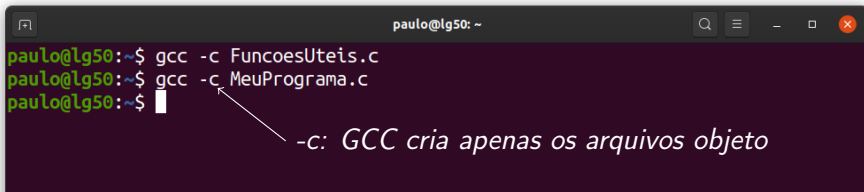
```
1 #include <stdio.h>
2 #include "FuncoesUteis.h"
3
4 void Int_printArray(int v[], int length) {
5     int i;
6     for (i=0; i < length; i++)
7         printf("%d ", v[i]);
8     printf("\n");
9 }
10 void Float_printArray(float v[], int length) {
11     int i;
12     for (i=0; i < length; i++)
13         printf("%f ", v[i]);
14     printf("\n");
15 }
```

- A união de arquivos header e suas implementações formam o que chamamos de **Módulos** em C.
- **Por exemplo**, o programa abaixo faz uso do módulo `FuncoesUteis` para realizar a impressão de dois vetores.

Código 3: `FuncoesUteis-main.c`

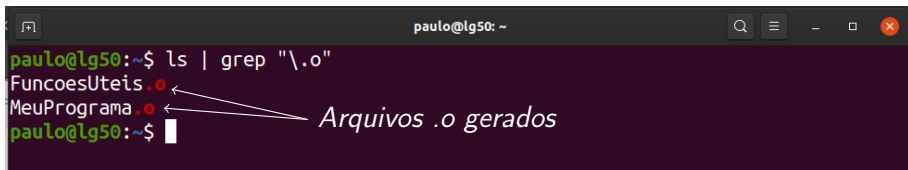
```
1  #include "FuncoesUteis.h"
2
3  int main() {
4      int n[] = {1,3,5,7,9};
5      float N[] = {2.0, 4.1, 6.001, 8.0, 10.9};
6
7      Int_printArray(n, 5);
8      Float_printArray(N, 5);
9
10     return 0;
11 }
```

- O header e uma implementação se conectam na etapa de **ligação** (*linking*) do processo de compilação.
- Para que possamos realizar essa ligação precisamos dos **arquivos objeto** do nosso programa e da implementação.



```
paulo@lg50: ~  
paulo@lg50:~$ gcc -c FuncoesUteis.c  
paulo@lg50:~$ gcc -c MeuPrograma.c  
paulo@lg50:~$
```

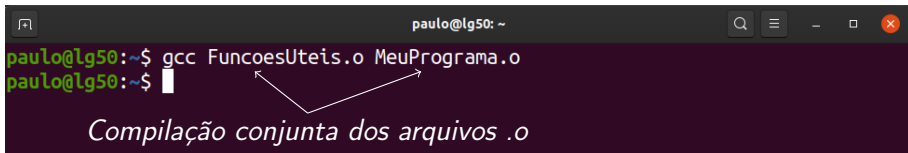
← -c: GCC cria apenas os arquivos objeto



```
paulo@lg50: ~  
paulo@lg50:~$ ls | grep "\.o"  
FuncoesUteis.o  
MeuPrograma.o  
paulo@lg50:~$
```

*Arquivos .o gerados*

Após sua geração, compilamos os arquivos objeto juntos para formar o executável.



```
paulo@lg50: ~  
paulo@lg50:~$ gcc FuncoesUteis.o MeuPrograma.o  
paulo@lg50:~$
```

*Compilação conjunta dos arquivos .o*

- Etapas para a compilação e execução do programa que usando o módulo externo FuncoesUteis.

1. `gcc -c FuncoesUteis.c FuncoesUteis-main.c`

2. `gcc FuncoesUteis.o FuncoesUteis-main.o`

3. `./a.out`

