

# Funções -- declaração

## ■ Forma Geral:

*tipo nome\_da\_função (lista de parâmetros)*  
*{declarações sentenças}*

- Tudo antes do “abre-chaves” compreende o **cabeçalho** da **definição** da função.
- Tudo entre as chaves compreende o **corpo** da **definição** da função.

# Exemplo- função

The diagram illustrates the components of a C function definition. The function signature `char func (int x, char y)` is enclosed in a red box, with an arrow pointing to the label "cabeçalho" (header). The function body, enclosed in a blue box, contains the following code:  
`{  
 char c;  
 c = y+ x;  
 return (c);  
}`  
An arrow points from the blue box to the label "corpo" (body).

```
char func (int x, char y)
{
    char c;
    c = y+ x;
    return (c);
}
```

cabeçalho

corpo

# Funções

- *tipo nome\_da\_função (lista de parâmetros)*  
*{declarações sentenças}*
- **tipo:** é o tipo da função, i.e., especifica o tipo do valor que a função deve retornar (*return*).
  - Pode ser qualquer tipo válido.
  - Se a função não retorna valores, seu tipo é **void**.
  - Se o tipo não é especificado, tipo *default* é **int**.
  - Se necessário, o valor retornado será convertido para o tipo da função.



# Exemplo

```
int soma(int x,int y);
```

```
void main() {  
    int x,y, result;  
    result= soma(x,y);  
}
```

```
int soma (int x, int y) {  
    int c;  
    c = y + x;  
    return (c);  
}
```



# Parâmetro das funções

Podem variar por **tipo**, **ordem** e **quantidade** de parâmetros

double calculaArea(float base, int altura, char k);

**tipo**

double calculaArea(int altura , float base, char k);

**ordem**

double calculaArea(float base, int altura);

**quantidade**

# Passagem de parâmetro por valor

- Modo **default** de passagem em C e C++
- Na chamada da função, os parâmetros são **copiados localmente**
- Uma **mudança interna não** acarreta uma **mudança externa**

```
void potencia2_valor (int n) { n = n * n; }
```

# Passagem de parâmetro por valor

```
void potencia2_valor (int n) { n = n * n; }
```

Chamada:

```
int x = 3;  
potencia2_valor ( x );  
cout << x ;           //Imprime 3
```



# Exercício: Passagem por valor

Faça um programa que leia a base e a altura de um retângulo e imprima o perímetro, a área e a diagonal. Para fazer os cálculos, implemente três funções, cada uma deve realizar um cálculo específico conforme solicitado. Utilize as fórmulas a seguir.

$$\textit{perimetro} = 2 \times (\textit{base} + \textit{altura})$$

$$\textit{area} = \textit{base} \times \textit{altura}$$

$$\textit{diagonal} = \sqrt{\textit{base}^2 + \textit{altura}^2}$$





# Tipos de funções

- As funções são geralmente de três tipos
  1. Executam um cálculo: sqrt(), sin(),
  2. Manipula informações: retorna sucesso/falha
  3. Procedimentos: exit(), retorna void

Não é necessário utilizar os valores retornados



# Funções de tipo não inteiro

- Antes de ser usada, seu tipo deve ser declarado de duas formas
  - Método tradicional
  - Protótipos

# Funções de tipo não inteiro

- Método tradicional

- Declaração do tipo e nome antes do uso
- Os argumentos não são indicados, mesmo que existam

```
float sum();
```

```
void main() { .... }
```

```
float sum( float a, float b ) {...}
```

# Funções de tipo não inteiro

## ■ Protótipo

- Inclui também a quantidade e tipos dos parâmetros

`float sum(int , int);`

`void main() { .... }`

`float sum( float a, float b ) {...}`

- E no caso de funções sem argumento?
  - Declarar lista como `void`



# Estruturas: struct

- **Coleção de variáveis** organizadas em um único conjunto.
  - Possivelmente coleção de tipos distintos
- As variáveis que compreendem uma estrutura são comumente chamadas de **elementos** ou **campos**.

# Exemplo-estruturas

- Definição x Declaração

```
struct pessoa  
{  
    char nome[30];  
    int idade;  
};
```

- Permite declarar variáveis cujo tipo seja **pessoa**.



# Usando typedef nas estruturas

O typedef define um novo tipo:

```
typedef int inteiro;  
typedef float real;  
typedef char caractere;
```

Cria novos tipos a partir de tipos definidos previamente



# Usando typedef nas estruturas

O typedef define um novo tipo

```
struct a{  
    int x; char y;  
};
```

```
typedef struct a NewStruct; int
```

```
main(){  
    MyStruct b; /*declaração da var b, cujo tipo é NewStruct*/  
}
```



# Acesso aos dados da Estrutura

- É feito via o ponto (.)

```
int main (void){  
    MyStruct obj;  
    obj.x = 10;  
    obj.y = 'a';  
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct pessoa {
    char nome[50], rua[50];
    int idade, numero;
};

int main() {

    struct pessoa p;
    strcpy(p.nome, "Nome" );
    strcpy(p.rua, "Street 4" );
    p.idade = 27;
    p.numero = 1874;

    return 0;

}

```

INICIALIZAÇÃO CAMPO A CAMPO

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct pessoa {
    char nome[50], rua[50];
    int idade, numero;
};

int main() {

    struct pessoa p = { "Nome",
        "Street 4", 27, 1874 };

    //Campos não inicializados
    //explicitamente são inicializados
    //com zero
    struct pessoa p2 = { "Nome2",
        "Street 4", 27 };

    return 0;

}

```

INICIALIZAÇÃO COMO VETOR

**ATRIBUIÇÃO COMO  
VARIÁVEL NORMAL**



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

struct ponto {
    int x,y;
};

struct ponto_B {
    int x,y;
};

int main() {

    struct ponto p1, p2 = {1,2};
    struct ponto_B p3 = {3,4};

    p1 = p2; //OK
    p1 = p3; //Erro ! Tipos diferentes

    return 0;
}
```

# Vetor de struct: definição da estrutura

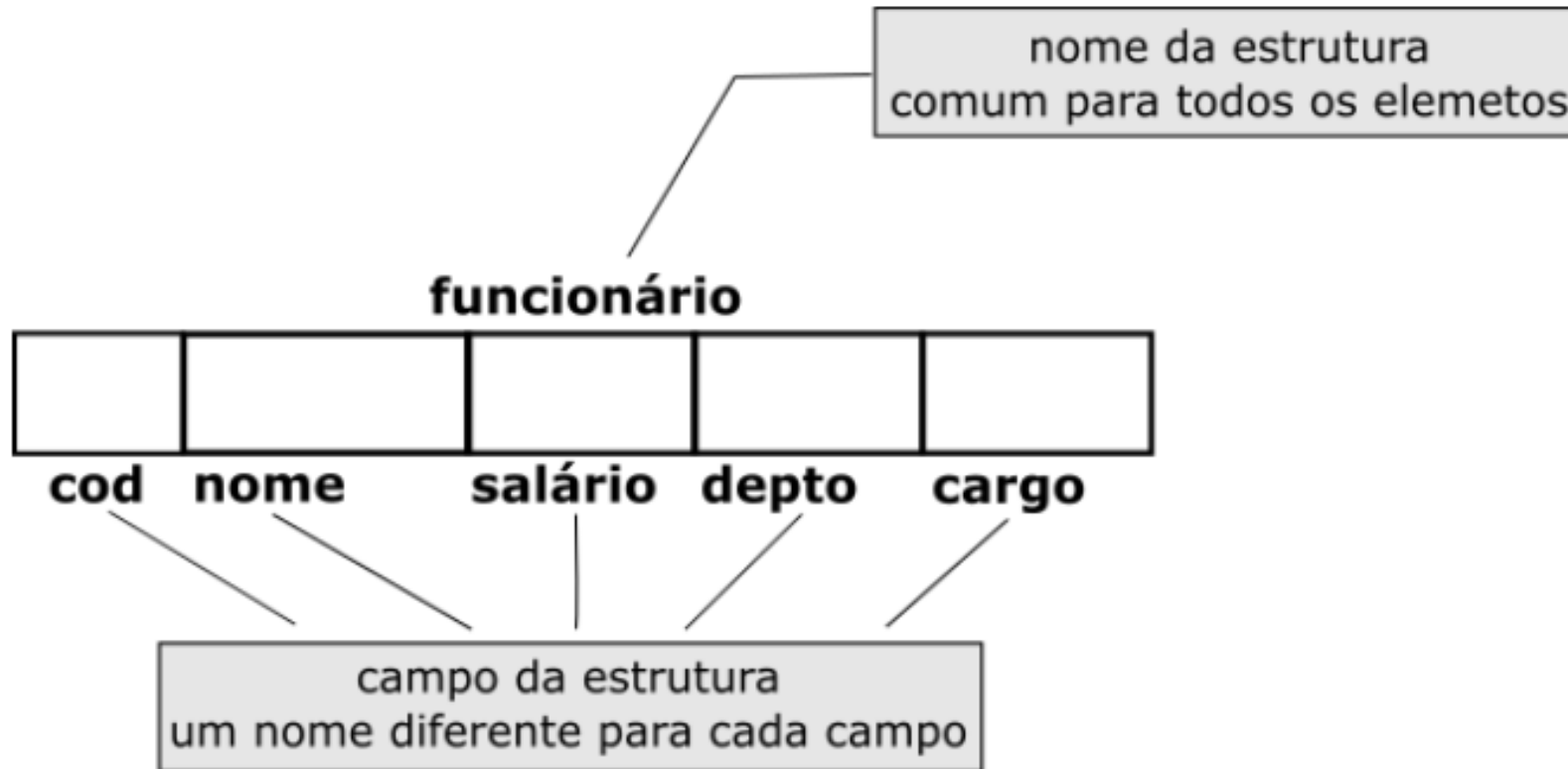
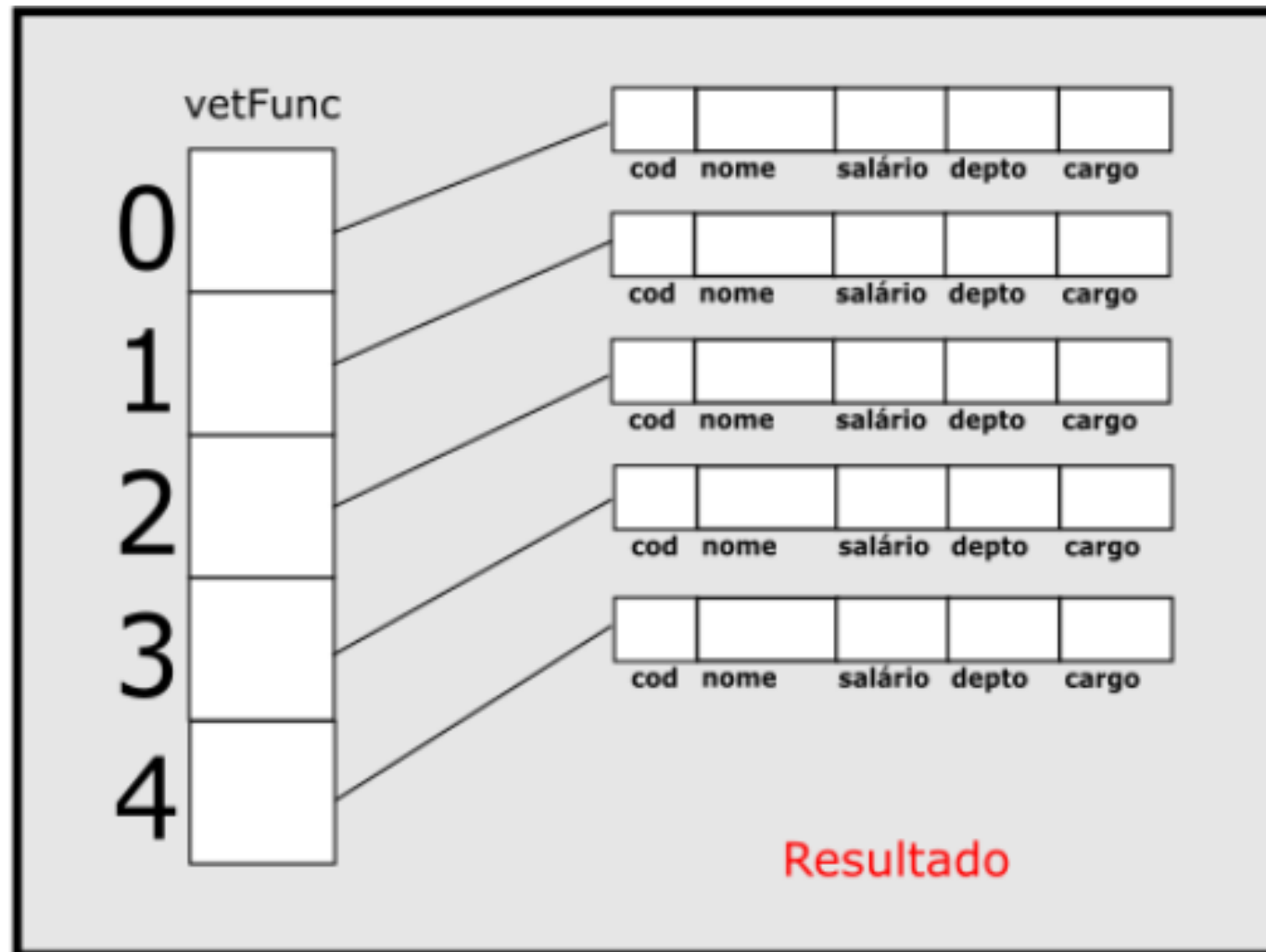


Figura 20 – Exemplo de estrutura

Fonte: Adaptado de (EDELWEISS; LIVI, 2014, p. 294)


## Vetor de struct: vetor em si





## Vetor de struct: exercício

A delegação francesa de Futsal deseja criar um programa que deva conter o nome do atleta, sua posição, idade e altura. Crie uma estrutura representando um atleta. Agora, escreva um programa que leia os dados de cinco atletas. Calcule e exiba os nomes do atleta mais alto e do mais velho.



# Aninhamento de estruturas

```
struct tipo_struct1 {...};
```

```
struct tipo_struct2 {  
    ...  
    struct tipo_struct1 nome;  
}
```

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

```
struct endereco {
    char rua[80];
    int numero;
};

struct pessoa {
    char nome[50];
    int idade;
    struct endereco ender;
};
```

```
int main() {

    struct pessoa p;
    p.idade = 31;
    p.ender.numero = 103;

    return 0;
}
```



# Union

- As estruturas de dados do tipo **Union** permite armazenar diferentes tipos de dados no mesmo local de memória.
- A grande vantagem dessa estrutura está na organização da memória, e no seu reaproveitamento, isto é, as unions fornecem uma maneira eficiente de usar o mesmo local de memória para vários propósitos.

## Definindo uma Union

- Definição de **union** é igual à de typedef struct
- A instrução **union** cria um novo tipo de dado
- O formato da declaração da **union** é o seguinte

```
union [Nome_do_Tipo_Union] {  
    Tipo_de_dado1 variavel1;  
    Tipo_de_dado2 variavel2;  
    ...  
    Tipo_de_dadoN variavelN;  
} [uma ou mais variaveis Union];
```

\*Nome\_do\_Tipo\_Union é opcional. Não é necessário definir.

# Tipo Union

```
#include <stdio.h>
#include <string.h>
int main( ) {
    union {
        int i;
        float f;
        char str[20];
    } dado;

    dado.i = 10; /* union sera do tipo inteiro */
    printf( "Sou inteiro : %d\n", dado.i);
    dado.f = 34.5; /* union sera do tipo float */
    printf( "Sou real : %f\n", dado.f);
    strcpy(dado.str,"Sou String"); /* union sera do tipo String */
    printf( "Sou string : %s\n", dado.str);

    return 0;
}
```