



MCTA028-15: Programação Estruturada

Aula 6: Tipos Definidos pelo Programador

Wagner Tanaka Botelho

wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com

Universidade Federal do ABC (UFABC)

Centro de Matemática, Computação e Cognição (CMCC)

Introdução

- Os **tipos de variáveis** que já estudamos até agora são **classificados** em **DUAS** categorias:
 - **Tipos básicos:**
 - Char, int, float, double, etc;
 - **Tipos compostos homogêneos:**
 - Array.
- Dependendo da situação que desejamos modelar, esses tipos podem **NÃO** ser suficientes:
 - Por esse motivo, a linguagem C permite criar **NOVOS** tipos de dados a partir dos **tipos básicos**.

Introdução

- Para criar um **NOVO** tipo de dado, tem-se os seguintes **comandos**:
 - **Estruturas**: comando `struct`;
 - **Uniões**: comando `union`;
 - **Enumerações**: comando `enum`;
 - **Renomear um tipo existente**: comando `typedef`.

Estruturas: Struct

Estruturas: Struct

- Uma **estrutura** pode ser vista como um **conjunto de variáveis** sob o mesmo **nome**, e cada uma delas pode ter qualquer tipo;
- A **ideia básica** é criar apenas um **tipo de dado** que contenha **VÁRIAS** variáveis, ou seja:
 - Cria-se **UMA** variável que contém dentro de si **OUTRAS** variáveis.
- A principal **VANTAGEM** do seu uso está relacionada com a possibilidade de **agrupar**, de forma **organizada**, diferentes tipos de dados dentro de **UMA** única variável.

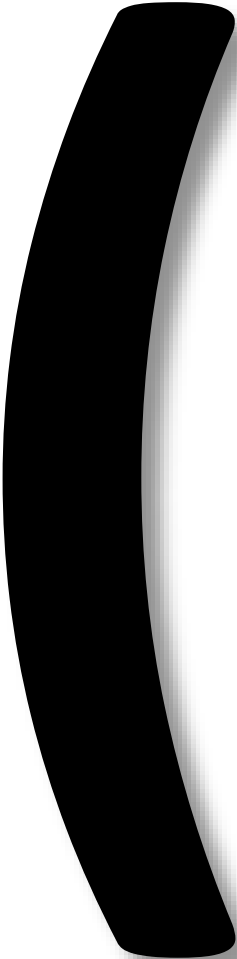
Declaração da Struct

Declaração

➤ A forma geral é:

```
struct nome_struct{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipon campoN;  
};
```

➤ As estruturas podem ser **declaradas** em qualquer escopo do programa (**GLOBAL** ou **LOCAL**).



Escopo Local

*Ex_01.c X

```
1  #include<stdio.h>
2  void main() {
3      int a=0, b=0, c=0;
4
5      a = 10;
6      b = 5;
7
8      c = a + b;
9
10     printf("Soma: %i", c);
11 }
```

Linha	a	b	c
3	0	0	0
5	10		
6		5	
8			15
10			{15}

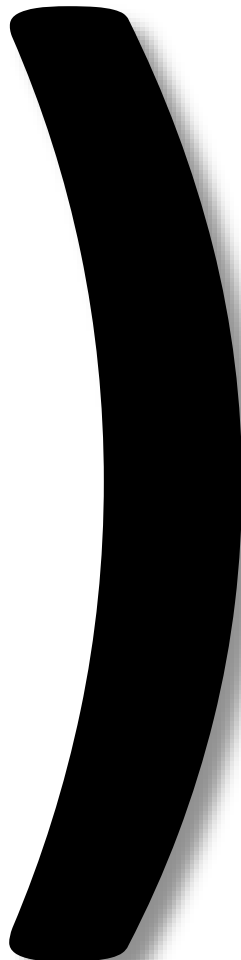
Escopo Global

```

1  #include<stdio.h>
2  int a=0, b=0, c=0, d=0;
3
4  void main(){
5      a = 10;
6      b = 5;
7
8      c = a + b;
9
10     printf("c (1): %i \n", c);
11
12     Funcao();
13
14     printf("c (3): %i \n", c);
15 }
16
17 void Funcao(){
18     c=12;
19     printf("c (2): %i \n", c);
20 }

```

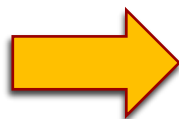
Linha	a	b	c	d
2	0	0	0	0
5	10			
6		5		
8			15	
10			{15}	
18			12	
19			{12}	
14			{12}	



Declaração

- A maioria das estruturas é declarada no **ESCOPO GLOBAL**:
 - Por se tratar de um **NOVO** tipo de dado, muitas vezes é interessante que **todo** o programa tenha **acesso** à estrutura.
- Exemplo de uma **estrutura** declarada para representar o **CADASTRO** de uma pessoa:

```
struct cadastro{  
    char nome[50];  
    int idade;  
    char rua[50];  
    int numero;  
};
```



cadastro

char nome[50];

int idade;

char rua[50];

int numero;

Os **nomes** dos **MEMBROS** (variáveis) devem ser **diferentes**. Entretanto, **ESTRUTURAS DIFERENTES** podem ter **MEMBROS** com **NOMES IGUAIS**

Declarando uma Variável do Tipo da Estrutura

- Após definir a estrutura (**struct**), uma **variável** pode ser declarada de modo similar aos tipos existentes:

```
struct cadastro c;
```

- O uso de **estruturas** facilita a vida do programador na **manipulação dos dados** do programa. Imagine, por exemplo, ter que declarar **QUATRO** cadastros para **QUATRO** pessoas diferentes:

```
char nome1[50], nome2[50], nome3[50], nome4[50];  
int idade1, idade2, idade3, idade4;  
char rua1[50], rua2[50], rua3[50], rua4[50];  
int numero1, numero2, numero3, numero4;
```

- Usando estruturas, a declaração é:

```
struct cadastro c1, c2, c3, c4;
```

Acessando os Campos (Variáveis) de uma Estrutura

- Após definir as **variáveis** do tipo da estrutura (**struct**), é preciso acessar os campos (variáveis):

Cada **VARIÁVEL** pode ser **ACESSADA** usando o **OPERADOR “.” (PONTO)**.

Ex_04.c x

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct cadastro{
5      char nome[50];
6      int idade;
7      char rua[50];
8      int numero;
9  };
10
11 int main(){
12     struct cadastro c;
13
14     strcpy(c.nome, "Carlos");
15
16     c.idade = 18;
17
18     strcpy(c.rua, "Avenida Brasil");
19
20     c.numero = 1082;
21 }
```

Declaração da estrutura

Declarando uma variável da estrutura

Atribui “Carlos” para a variável nome

Atribui 18 para a variável idade

Atribui “Avenida Brasil” para rua

Atribui 1082 para numero

➤ Ler os **VALORES** das **VARIÁVEIS** pelo **TECLADO**:

Ex_11.c X

```
1  #include <stdio.h>
2
3  struct cadastro{
4      char nome[50];
5      int idade;
6      char rua[50];
7      int numero;
8  };
9
10 void main(){
11     struct cadastro c;
12
13     fflush(stdin);
14     fgets(c.nome, 50, stdin);
15     scanf("%d", &c.idade);
16     fflush(stdin);
17     fgets(c.rua, 50, stdin);
18     scanf(" %d", &c.numero);
19 }
20
21
22
```

Declaração da estrutura

Declarando uma variável da estrutura

Limpar o buffer do teclado!

Lê do teclado uma string e armazena em nome
stdin: leitura é feita pelo teclado.

Lê do teclado um inteiro e armazena em idade

Lê do teclado uma string e armazena em para rua

Lê do teclado um inteiro e armazena em numero

Inicialização de Estruturas

➤ Assim como nos arrays, uma estrutura também pode ser inicializada:

```
struct cadastro c = {"Carlos", 18, "Avenida Brasil", 1082};
```

Watches	
Function arguments	
Locals	
c	
nome	"Carlos", '\000' <re
idade	18
rua	"Avenida Brasil", '
numero	1082

Atribuído em
nome

Atribuído em
idade

Atribuído em
rua

Atribuído em
numero

```
struct cadastro c = {"Carlos", 18};
```

Elementos **OMITIDOS** são **inicializados** com **0**.
Se for uma **string**, será **inicializada** com
uma **string VAZIA** ("").

Watches	
Function arguments	
Locals	
c	
nome	"Carlos", '\000' <re
idade	18
rua	'\000' <repeats 49 t
numero	0

Arrays de Estruturas

- Como vimos, imagine ter que declarar **QUATRO** cadastros para **QUATRO** pessoas diferentes:

```
char nome1[50], nome2[50], nome3[50], nome4[50];  
int idade1, idade2, idade3, idade4;  
char rua1[50], rua2[50], rua3[50], rua4[50];  
int numero1, numero2, numero3, numero4;
```

- Usando estruturas, a declaração é:

```
struct cadastro c1, c2, c3, c4;
```

- A representação desses **QUATRO** cadastros pode ser ainda mais simplificada se utilizarmos o conceito de **arrays**:

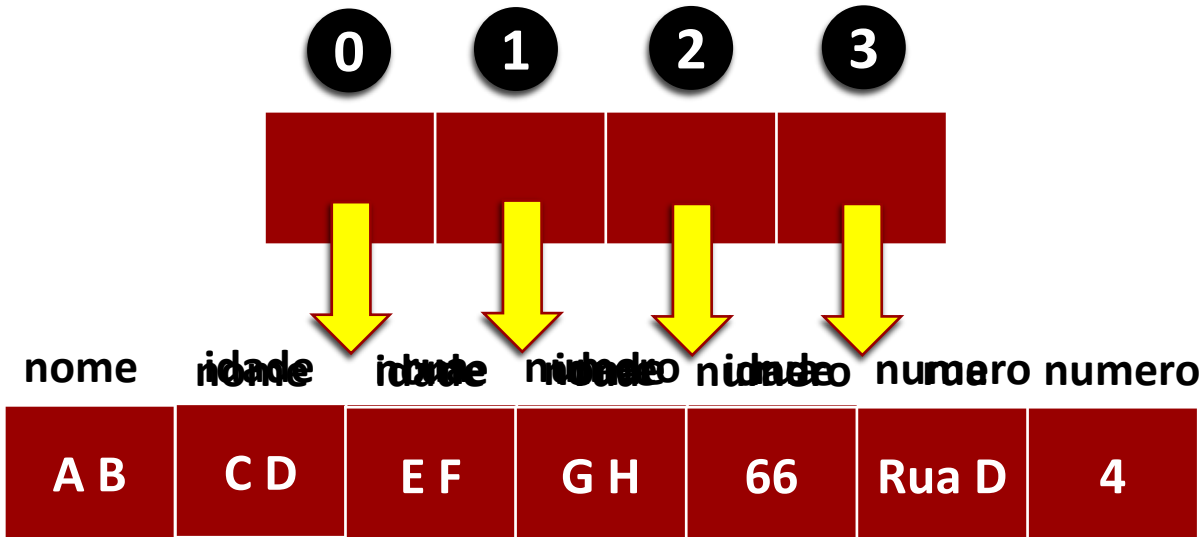
```
struct cadastro c[4];
```

Um **array** de estruturas é criado, em que cada posição do **array** é uma estrutura do tipo cadastro.

```
1  #include <stdio.h>
2
3  struct cadastro{
4      char nome[50];
5      int idade;
6      char rua[50];
7      int numero;
8  };
9
10 void main() {
11     struct cadastro c[4];
12
13     for(int i=0; i<4; i++){
14         fflush(stdin);
15         fgets(c[i].nome, 50, stdin);
16
17         scanf("%d", &c[i].idade);
18
19         fflush(stdin);
20         fgets(c[i].rua, 50, stdin);
21
22         scanf("%d", &c[i].numero);
23     }
24 }
```

Limpar o buffer do teclado!

stdin: leitura é
feita pelo Teclado.



Cada posição do array
é uma estrutura do tipo
cadastro.

Function arguments

Locals

c

[0]

nome	"A B\n\000\000\000\000Egø@û\177\000"
idade	33
rua	"Rua A\n\000\000\000\000\001\000\000\000"
numero	1

[1]

nome	"C D\n\000yyy\023\000\024\000\000\000"
idade	44
rua	"Rua B\n\000\000\000\000\000\023\000\000\000"
numero	2

[2]

nome	"E F\n\000\000\000\000a\000\000P\000"
idade	55
rua	"Rua C\n\000\000\000\000\000\262\000\000\000"
numero	3

[3]

nome	"G H\n", '\000' <repeats 12 times>, "D\0"
idade	66
rua	"Rua D\n", '\000' <repeats 18 times>, "\0"
numero	4

Atribuição Entre Estruturas

- As **atribuições** entre estruturas **só** podem ser feitas quando as estruturas são **IGUAIS**, ou seja, quando possuem o **mesmo NOME**!

Ex_06.c X

```
1  #include <stdio.h>
2
3  struct ponto {
4      int x;
5      int y;
6  };
7
8  struct novo_ponto {
9      int x;
10     int y;
11 };
12
13 void main() {
14     struct ponto p1, p2 = {1, 2};
15     struct novo_ponto p3 = {3, 4};
16     p1 = p2;
17
18     printf("p1 = %d e %d", p1.x, p1.y);
19 }
```

 D:\UFABC\Disciplinas\2021-2025\c

```
p1 = 1 e 2
Process returned 10 (0xA)   e
Press any key to continue.
```

 Estruturas IGUAIS!

Ex_07.c X

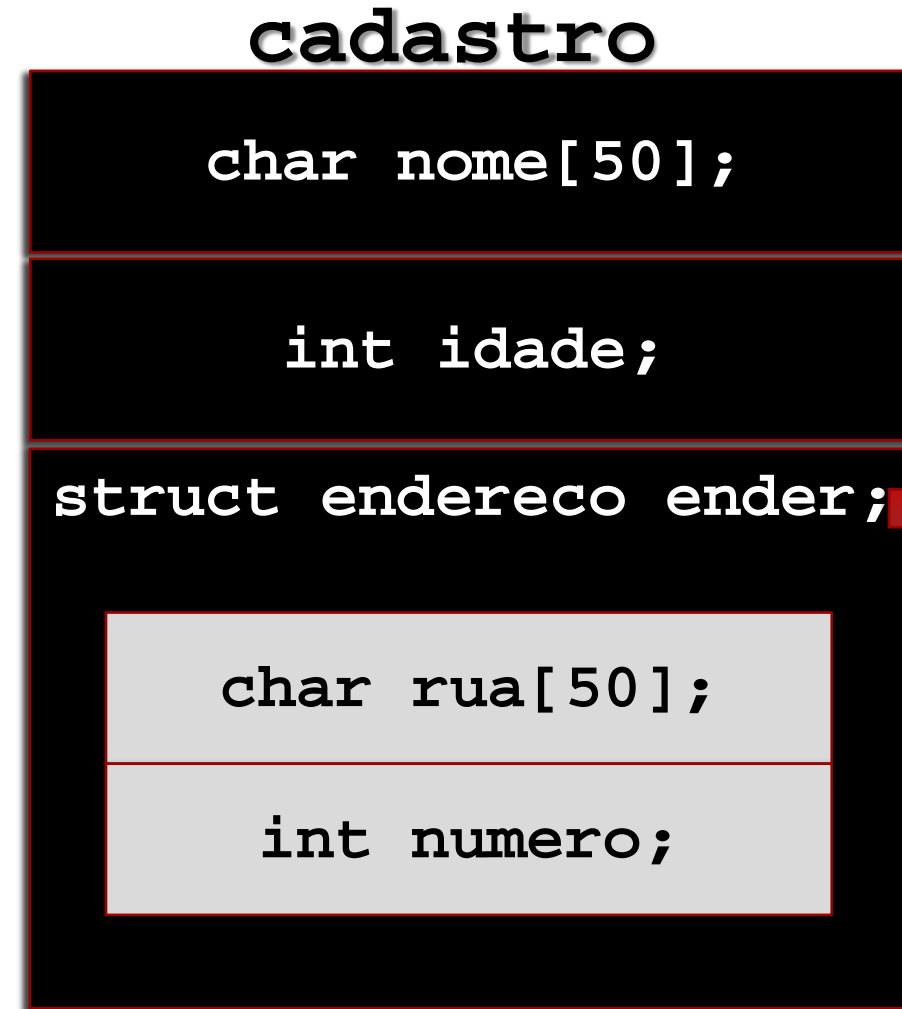
```
1  #include <stdio.h>
2
3  struct ponto {
4      int x;
5      int y;
6  };
7
8  struct novo_ponto {
9      int x;
10     int y;
11 };
12
13 void main() {
14     struct ponto p1, p2 = {1,2};
15     struct novo_ponto p3 = {3,4};
16
17     p1 = p3;
18     printf("p1 = %d e %d", p1.x, p1.y);
19 }
```

 **ERRO! Estruturas DIFERENTES!**

Estruturas Aninhadas

➤ Uma **estrutura** que contenha **OUTRA** estrutura dentro dela é conhecida como **ESTRUTURAS ANINHADAS**.

```
struct endereco{  
    char rua[50];  
    int numero;  
};  
  
struct cadastro{  
    char nome[50];  
    int idade;  
    struct endereco ender;  
};
```



Estrutura
aninhada

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

```
2  
3  struct endereco{  
4      char rua[50];  
5      int numero;  
6  };
```

```
7  
8  struct cadastro{  
9      char nome[50];  
10     int idade;  
11     struct endereco ender;  
12 };
```

```
13  
14 void main(){
```

```
15     struct cadastro c; ➡ Declarando uma variável da estrutura
```

```
16  
17     fflush(stdin);
```

```
18     fgets(c.nome, 50, stdin); ➡ Lê do teclado uma string e armazena em nome
```

```
19  
20     scanf("%d",&c.idade); ➡ Lê do teclado um inteiro e armazena em idade
```

```
21  
22     fflush(stdin);
```

```
23     fgets(c.ender.rua, 50, stdin); ➡ Lê do teclado uma string e armazena em  
24     rua da variável ender
```

```
25  
26     scanf("%d",&c.ender.numero); ➡ Lê do teclado um inteiro e armazena em  
27     numero da variável ender  
}
```

União: Union

União: Union

- A ideia básica é similar à da estrutura;
- Diferentemente das estruturas, **TODOS** os elementos contidos na união ocupam o **MESMO** espaço físico na memória:
 - O **espaço de memória** é reservado para o seu **MAIOR** elemento e **COMPARTILHA** essa memória com os demais.

```
union tipo{  
    short int x;  
    unsigned char c;  
};
```

- Essa união possui o nome **tipo** e **DUAS** variáveis:
 - **x** do tipo **short int** (**dois** bytes);
 - **c** do tipo **unsigned char** (**um** byte).
- Assim, uma variável declarada desse tipo: `union tipo t;`
- Ocupará **DOIS BYTES** na memória, que é o **tamanho** do **MAIOR** dos elementos da união (**short int**).

Enumerações: Enum

Enumerações: Enum

- É uma **LISTA** de constantes, em que cada **constante** possui um nome significativo;
- A **ideia básica** é criar apenas um **tipo de dado** que contenha **VÁRIAS** constantes, e uma variável desse tipo só poderá receber como valor uma dessas constantes;

*Ex_09.c x

```
1  #include <stdio.h>
2
3  enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado};
4
5  void main() {
6      printf("Domingo %i\n", Domingo);
7      printf("Segunda %i\n", Segunda);
8      printf("Terca %i\n", Terca);
9      printf("Quarta %i\n", Quarta);
10 }
11
```

D:\UFABC\Disciplinas\2021-2025\Q1\P

```
Domingo 0
Segunda 1
Terca 2
Quarta 3
```

```
Process returned 9 (0x9)   execu
Press any key to continue.
```

Comando Typedef

Comando: Typedef


- Permite que o programador defina os seus **próprios TIPOS** com base em outros tipos de dados existentes:
- Utiliza-se o comando **typedef**, cuja forma geral é:

```
typedef tipo_existente novo_nome;
```

- **tipo_existente**: é um tipo básico ou definido pelo programador (por exemplo, um `int`);
- **novo_nome**: é o nome para o novo tipo definindo.

Ex_10.c X

```
1  #include <stdio.h>
2
3  typedef int inteiro;
4
5  void main() {
6      int x = 10;
7      inteiro y = 20;
8      y = y + x;
9      printf("Soma = %d\n", y);
10 }
```

 D:\UFABC\Disciplinas\2021-2025\Q1\PE\A

Soma = 30

Process returned 10 (0xA) executi
Press any key to continue.

IMPORTANTE!!

O comando `typedef` **NÃO** cria um novo tipo. Ele apenas permite que você defina um **sinônimo** para um tipo já existente.

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011.