



UNIVERSIDADE
FEDERAL DO CEARÁ



CK0179 – Programação Computacional para Engenharia: Estruturas (structs)

Prof. Maurício Moreira Neto

Objetivo

- Aprender a criar novos tipos de dados estruturados a partir de dados primitivos
- Aprender a usar as estruturas para resolução de problemas na linguagem C

Variáveis

- As variáveis que foram vistas são classificadas por duas categorias:
 - Simples: definidas por tipos **int**, **float**, **double** e **char**
 - Compostas Homogêneas: definidas por **array**
- No entanto, a linguagem C permite que se criem novas estruturas a partir dos tipos básicos
 - **struct**

Estruturas

- Uma estrutura pode ser vista como um novo tipo de dado, que é formado por composição de variáveis de outros tipos
 - Pode ser declarada em qualquer escopo
 - A declaração é feita da seguinte maneira:

```
struct{  
    tipo1 campo1;  
    tipo2 campo2;  
    tipo3 campo3;  
    ...  
    tipoN campoN;  
};
```

Estruturas

- Uma estrutura pode ser vista como um agrupamento de dados
- Exemplo: cadastro de pessoas
 - Todas essas informações são da mesma pessoa, logo, é possível agrupá-los
 - Isso facilita também lidar com dados de outras pessoas no mesmo programa

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

Cadastro

char nome[50];
int idade;
char rua[50];
int numero;

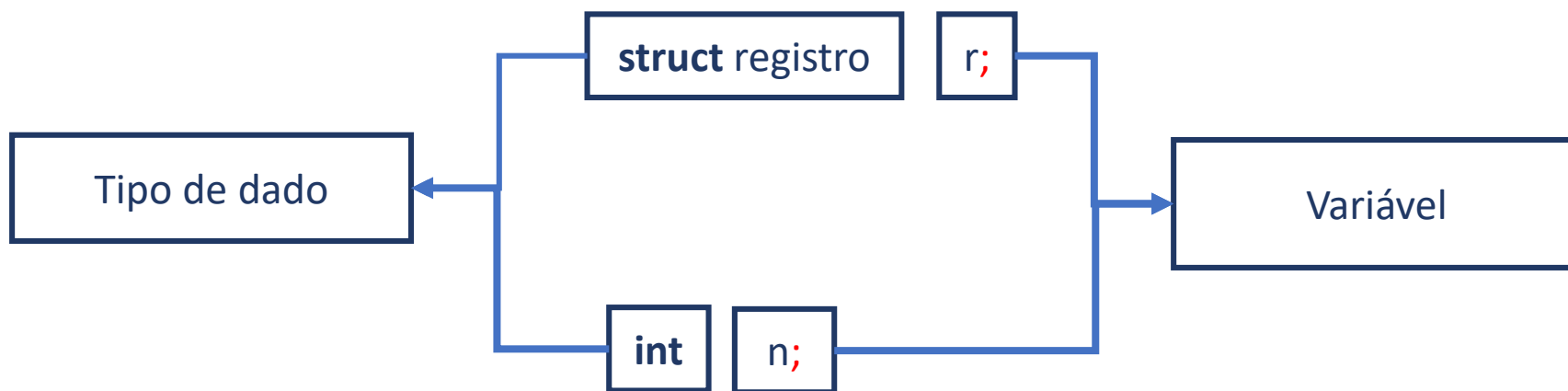
Estruturas – Declaração

- Uma vez definida a estrutura, uma variável pode ser declarada de modo similar aos tipos já existentes:

```
struct registro r;
```

- Note: Usa-se a palavra **struct** antes do tipo da nova variável

Estruturas – Declaração



Estruturas – Declaração

- **Exercício:** Declare uma estrutura que seja capaz de armazenar 3 notas para um dado aluno.

Estruturas – Declaração

- Soluções possíveis:

```
struct aluno{  
    int numero_aluno;  
    int nota1, nota2, nota3;  
};
```

1

2

```
struct aluno{  
    int numero_aluno;  
    int notas[3];  
};
```

Estruturas

- O uso de estruturas facilita na manipulação dos dados do programa. Declarar 4 cadastros, para 4 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;
```

Estruturas

- Utilizando uma estrutura, o mesmo pode ser feito desta forma:

```
struct registro{  
    char nome[50];  
    int idade;  
    char rua[50];  
    int numero;  
};  
struct registro r1, r2, r3, r4;
```

Acessando às variáveis

- Como é feito o acesso às variáveis da estrutura?
 - Cada variável da estrutura pode ser acessada com o operador ponto “.”
- Exemplo:

```
// declarando a variavel
struct registro r;
// acessando os campos
strcpy(r.nome, "mauricio");
scanf("%d", &r.idade);
strcpy(r.rua, "Fco Farias Filho");
r.numero = 1111;
```

Acessando às variáveis

- Como nos arrays, uma estrutura pode ser previamente inicializada:

```
struct ponto{  
    int x;  
    int y;  
};  
struct ponto p = {100, 200}
```

Acessando às variáveis

- E como ler os valores das variáveis da estrutura do teclado:
 - Neste caso, baste ler cada variável independentemente, respeitando seus tipos

```
struct registro r;
```

```
gets(r.nome); // string
```

```
scanf("%d", &r.idade); //inteiro
```

```
gets(r.rua); // string
```

```
scanf("%d", &r.numero); //inteiro
```

Acessando às variáveis

- Cada variável dentro da estrutura pode ser acessado como se apenas ela existisse, não sofrendo nenhuma interferência das outras
 - Uma estrutura pode ser vista como um simples agrupamento de dados
 - Se faço um **scanf** para **estrutura.idade**, isso não me obriga a fazer um **scanf** para **estrutura.numero**

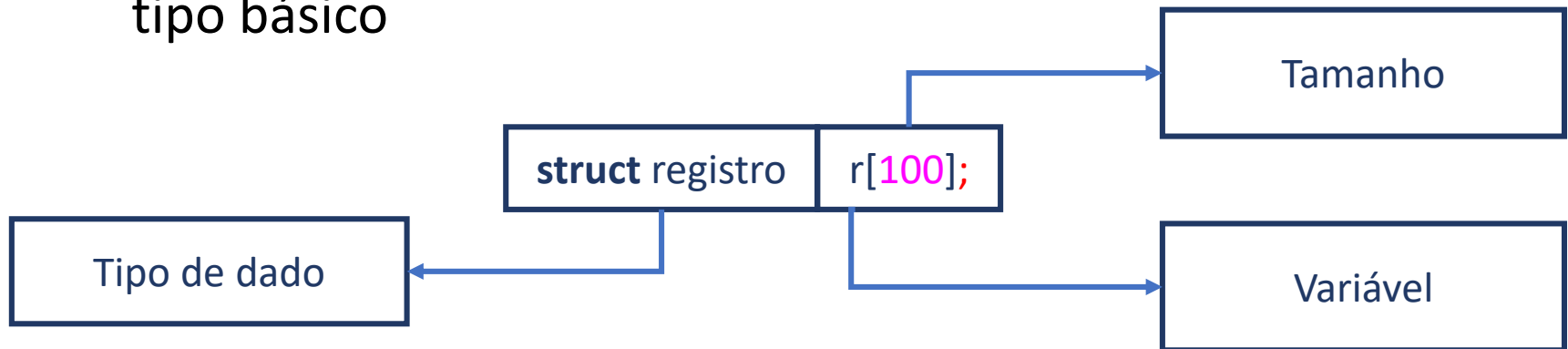
Estruturas

- Lembrando o exemplo anterior, se quiséssemos fazer 100 cadastros de pessoas?



Array de Estruturas

- Possível solução:
 - Criar um **array de estruturas**
 - Sua declaração é similar a declaração de um array de um tipo básico



- Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo **struct registro**

Array de Estruturas

- Lembrando:

- **struct**: define um “conjunto” de variáveis que pode ser de tipos diferentes
- **array**: é uma “lista” de elementos de mesmo tipo

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

r[0]

r[1]

r[2]

r[3]

```
char nome[50];  
int idade;  
char rua[50];  
int numero;
```

```
char nome[50];  
int idade;  
char rua[50];  
int numero;
```

```
char nome[50];  
int idade;  
char rua[50];  
int numero;
```

```
char nome[50];  
int idade;  
char rua[50];  
int numero;
```

Array de Estruturas

- Em um array de estrutura, o operador de ponto (.) vem depois dos colchetes ([]) do índice do array

```
int main( ){
    struct registro r[4];
    int n;
    for (n = 0; n < 4; n++) {
        gets(r[i].nome);
        scanf("%d", &r[n].idade);
        gets(r[i].rua);
        scanf("%d", &r[n].numero);
    }
    return 0;
}
```

Exercício

- Utilizando a estrutura a seguir, faça um programa que lê o número e as 3 notas de 10 alunos.

```
struct aluno {  
    int num_aluno;  
    float nota1, nota2, nota3;  
    float media;  
};
```

Solução

```
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};

int main() {
    int n;
    struct aluno al[10];
    for (n = 0 ; n < 10 ; n++) {
        scanf("%d", &al[n].num_aluno);
        scanf("%d", &al[n].num_aluno);
        scanf("%d", &al[n].num_aluno);
        scanf("%d", &al[n].num_aluno);
        al[n].media = (a[i].nota1 + a[i].nota1 + a[i].nota1)/3.0;
    }
}
```

Atribuição Entre Estruturas

- Atribuições entre estruturas só podem ser feitas quando as estruturas são AS MESMAS, ou seja, possuem o mesmo nome!

```
struct cadastro c1, c2;  
c1 = c2; //CORRETO
```

```
struct cadastro c1;  
struct ficha c2;  
c1 = c2; //ERRADO!! TIPOS DIFERENTES
```

Atribuição Entre Estruturas

- No caso de estarmos trabalhando com arrays, a atribuição entre diferentes elementos do array é válida

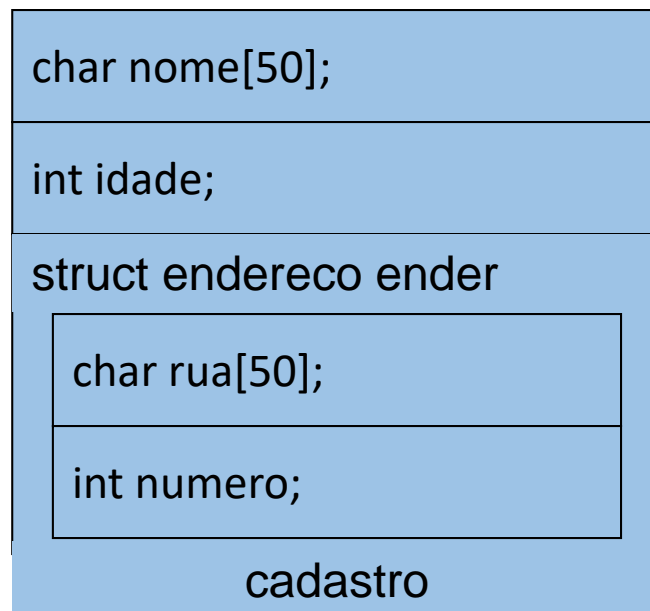
```
struct registro r[10];  
r[1] = r[2];
```

- Note que: os tipos dos diferentes elementos do array são sempre IGUAIS

Criando Estruturas de Estruturas

- Se uma estrutura é um tipo de dado, podemos declarar uma estrutura que utilize outra estrutura previamente definida:

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



Criando Estrutura de Estruturas

- O acesso aos dados do **endereço** do registro é feito utilizando novamente o operador ponto “.”

```
struct registro r;  
// leitura  
gets(r.nome);  
scanf("%d", &r.idade);  
gets(r.ender.rua);  
scanf("%d", &r.ender.numero);  
// atribuicao  
strcpy(r.nome, "Joao");  
r.idade = 29;  
strcpy(r.ender.rua, "Fco Farias");  
r.ender.numero = 233;
```

Criando Estrutura de Estruturas

- Inicialização de uma estrutura de estruturas:

```
struct ponto{  
    int x, y;  
};
```

```
struct retangulo{  
    struct ponto inicio, fim;  
};
```

```
struct retângulo r = {{10,20},{30,40}};
```

Comando TYPEDEF

- A linguagem C permite que o programador defina seus próprios tipos com base em outros tipos de dados existentes
- Para isso, utiliza-se o comando ***typedef***, cuja forma geral é:
 - `Typedef tipo_existente novo_nome;`

Comando TYPEDEF

- Exemplo

- Note que o comando **typedef** não cria um novo tipo chamado **inteiro**. Apenas cria um sinônimo (**inteiro**) para o tipo **int**

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

typedef int inteiro;

int main() {
    int x = 10;
    inteiro y = 20;
    y = y + x;
    printf("Soma = %d\n", y);

    return 0;
}
```

Comando TYPEDEF

- **typedef** é muito utilizado para definir nomes mais simples para estrutura, evitando carregar a palavra **struct** sempre que referenciamos a estruturas

```
struct cadastro{
    char nome[300];
    int idade;
};
// redefinindo o tipo struct cadastro
typedef struct cadastro CadAlunos;

int main(){
    struct cadastro aluno1;
    CadAlunos aluno2;

    return 0;
}
```

Dúvidas?



Referências

- André Luiz Villar Forbellone, Henri Frederico Eberspächer, Lógica de programação (terceira edição), Pearson, 2005, ISBN 9788576050247.
- Ulysses de Oliveira, Programando em C - Volume I - Fundamentos, editora Ciência Moderna, 2008, ISBN 9788573936599.
- **Slides baseados no material do site "Linguagem C Descomplicado"**
 - <https://programacaodescomplicada.wordpress.com/complementar/>