Algoritmos e Estruturas de Dados I

1º Período Engenharia da Computação

Prof. Edwaldo Soares Rodrigues
Email: edwaldo.rodrigues@uemg.br
Material adaptado do prof. André Backes

UNIVERSIDADE
DO ESTADO DE MINAS GERAIS | UEMG
UNIDADE DIVINÓPOLIS

Structs – Estruturas/Registros

Variáveis

- As variáveis vistas até agora podem ser classificados em duas categorias:
  - simples: definidas por tipos **int**, **float**, **double** e **char**;
  - compostas homogêneas (ou seja, do mesmo tipo): 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

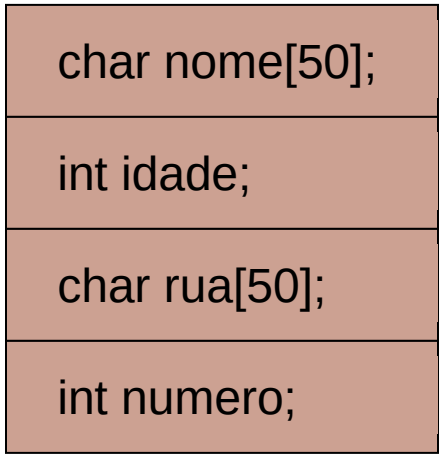
  - Pode ser declarada em qualquer escopo;
  - Ela é declarada da seguinte forma:

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

Estruturas

- Uma estrutura pode ser vista como um agrupamento de dados;
- Ex.: cadastro de pessoas:
  - Todas essas informações são da mesma pessoa, logo podemos agrupá-las;
  - Isso facilita também lidar com dados de outras pessoas no mesmo programa;

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



cadastro

Estruturas - declaração

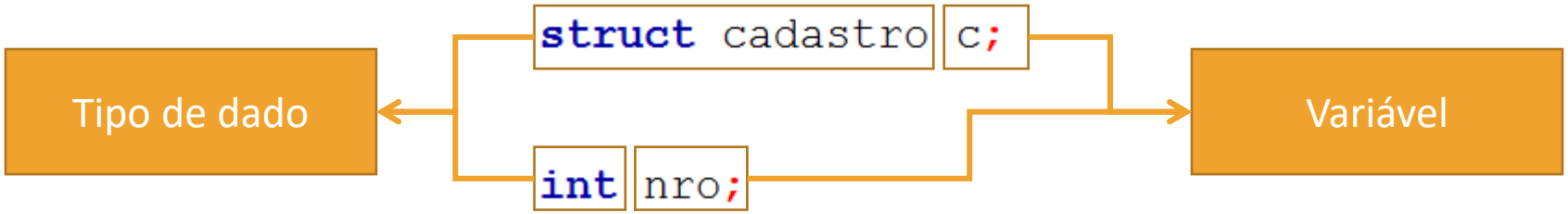
- Uma vez definida a estrutura, uma **variável** pode ser declarada de modo similar aos tipos já existentes:

```
struct cadastro c;
```

- Obs: por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável;

Estruturas - declaração

- Obs: por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável



Exercício

- Declare uma estrutura capaz de armazenar o número e 3 notas para um dado aluno;

Exercício - Solução

- Possíveis soluções

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

struct aluno {
    int num_aluno;
    int nota1;
    int nota2;
    int nota3;
};

struct aluno {
    int num_aluno;
    int nota[3];
};
```

Estruturas

- O uso de estruturas facilita na manipulação dos dados do programa. Imagine 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 da seguinte maneira:

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

//declarando 4 cadastros
struct cadastro c1, c2, c3, c4, c5;
```

Acesso às variáveis

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

```
//declarando a variável
struct cadastro c;

//acessando os seus campos
strcpy(c.nome,"João");
scanf("%d",&c.idade);
strcpy(c.rua,"Avenida 1");
c.numero = 1082;
```

Acesso às variáveis

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

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

struct ponto p1 = { 220, 110 };
```

Acesso às variáveis

- E se quiséssemos ler os valores das variáveis da estrutura do teclado?
  - Resposta: basta ler cada variável independentemente, respeitando seus tipos.

```
struct cadastro c;

gets(c.nome);//string
scanf("%d",&c.idade);//int
gets(c.rua);//string
scanf("%d",&c.numero);//int
```

Acesso às variáveis

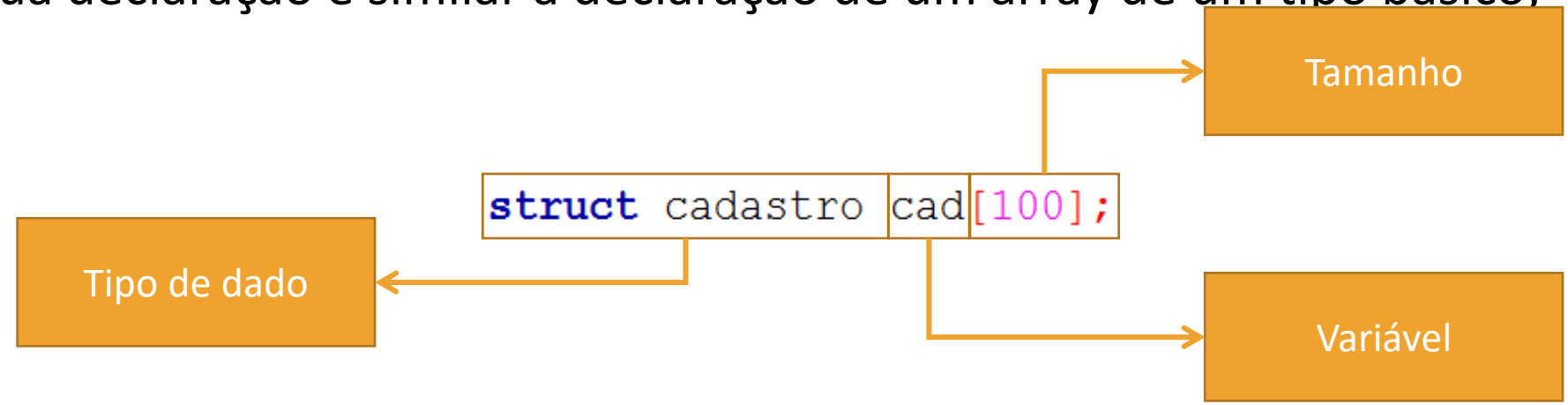
- Note que cada variável dentro da estrutura pode ser acessada como se apenas ela existisse, não sofrendo nenhuma interferência das outras.
  - Uma estrutura pode ser vista como um simples agrupamento de dados;
  - Se faço um **scanf** para **estrutura.idade**, isso não me obriga a fazer um **scanf**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

Estruturas

- Voltando ao exemplo anterior, se, ao invés de 4 cadastros, quisermos fazer 100 cadastros de pessoas?

Array de estruturas

- SOLUÇÃO: criar um **array de estruturas**.
  - Sua declaração é similar a declaração de um array de um tipo básico;

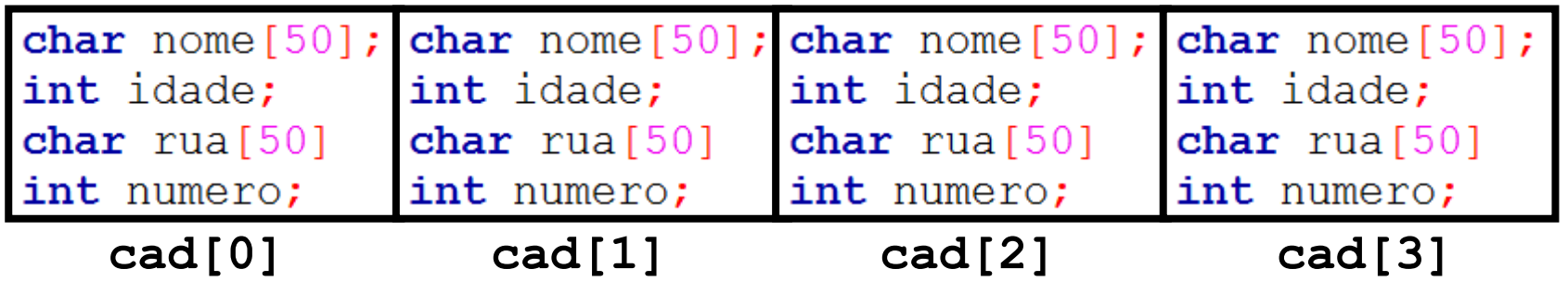


  - Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo **struct cadastro**;

Array de estruturas

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

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



Array de estruturas

- Num array de estruturas, o operador de ponto (**.**) vem depois dos colchetes (**[ ]**) do índice do **array**;

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

Exercício

- Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos:

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

Exercício - Solução

- Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos:

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


int main(){
    struct aluno a[10];
    int i;
    for(i=0;i<10;i++){
        scanf("%d",&a[i].num_aluno);
        scanf("%f",&a[i].nota1);
        scanf("%f",&a[i].nota2);
        scanf("%f",&a[i].nota3);
        a[i].media = (a[i].nota1 + a[i].nota2 + a[i].nota3)/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

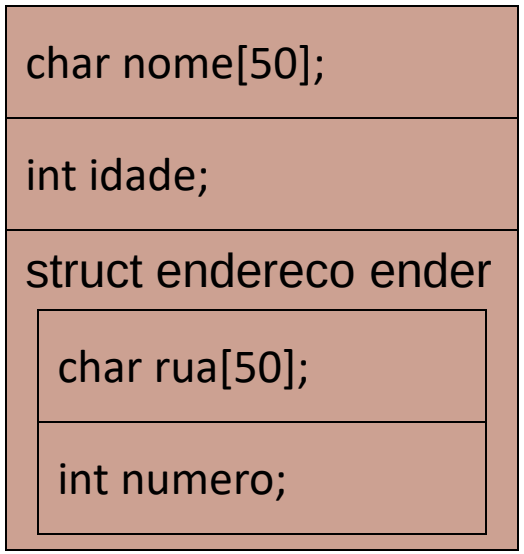
```
struct cadastro c[10];
c[1] = c[2]; //CORRETO
```

- Note que nesse caso, os tipos dos diferentes elementos do array são sempre IGUAIS.

Estruturas de estruturas

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

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



cadastro

Estruturas de estruturas

- Nesse caso, o acesso aos dados do **endereço** do cadastro é feito utilizando novamente o operador ponto “**.**”.

```
struct cadastro c;

//leitura
gets(c.nome);
scanf("%d",&c.idade);
gets(c.ender.rua);
scanf("%d",& c.ender.numero);

//atribuição
strcpy(c.nome,"João");
c.idade = 34;
strcpy(c.ender.rua,"Avenida 1");
c.ender.numero = 131;
```

Estruturas de estruturas

- Inicialização de uma estrutura de estruturas:

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

struct retangulo {
    struct ponto inicio, fim;
};

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

Comando typedef

- A linguagem C permite que o programador defina os 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**. Ele apenas cria um sinônimo (**inteiro**) para o tipo **int;**

```c
#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

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

```
struct cadastro{
    char nome[100];
    int idade;
};
//Usando Typedef
typedef struct cadastro{
    char nome[100];
    int idade;
}CadAlunos;

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

Neste exemplo, verifica-se a possibilidade de se declarar uma struct sem usar o typedef e usando o typedef.

Verifica-se que na função main que quando for criar uma variável do tipo da struct, é mais simples criar quando a mesma foi desenvolvida usando typedef.

Portanto, recomenda-se o uso da segunda opção;

Comando typedef

- Apesar de ser possível criar estruturas sem o uso de typedef, não recomendo que façam assim;

- A maioria dos exemplos que visualizarem, sejam em livros, internet e afins utiliza-se do typedef;

Exercícios

- Crie uma scruct para representar a matrícula (inteiro), as notas, n1, n2 e n3 e média geral, de um aluno, além de seu nome e curso que está cursando. Em seguida, leia por meio do teclado os dados de 5 alunos e armazene. Em seguida, faça:

  - a) apresente os dados do aluno que obteve a maior média geral;
  - b) Apresente os dados do aluno que obteve a maior nota entre as três recebidas;
  - c) Apresente os dados do aluno que se encontra na posição 1, em uma ordem crescente dos alunos de acordo com ordenação de seus nomes;

Algoritmos e Estruturas de Dados I

- Bibliografia:
  - Básica:
    - CORMEN, Thomas; RIVEST, Ronald, STEIN, Clifford, LEISERSON, Charles. Algoritmos. Rio de Janeiro: Elsevier, 2002.
    - DEITEL, Paul; DEITEL, Harvey. C++ como programar. 5. ed. São Paulo: Pearson, 2006.
    - MELO, Ana Cristina Vieira de; SILVA, Flávio Soares Corrêa da. Princípios de linguagens de programação. São Paulo: Edgard Blücher, 2003.
  - Complementar:
    - ASCENCIO, A. F. G. & CAMPOS, E. A. V. Fundamentos da programação de computadores. 2. ed. São Paulo: Pearson Education, 2007.
    - MEDINA, Marcelo, FERTIG, Cristina. Algoritmos e programação: teoria e prática. Novatec. 2005.
    - MIZRAHI, V. V.. Treinamento em linguagem C: módulo 1. São Paulo: Makron Books, 2008.
    - PUGA, S. & RISSETTI, G. Lógica de programação e estruturas de dados com aplicações em java. São Paulo: Prentice Hall, 2004.
    - ZIVIANI, Nívio. Projeto de Algoritmos com Implementação em Pascal e C. Cengage Learning. 2010.

Algoritmos e Estruturas de Dados I