VARIÁVEIS COMPOSTAS HETEROGÊNEAS STRUCTS

PROFA. DRA. GIANI CARLA ITO

Como você faria um programa para calcular as notas de 100 alunos?

Como você faria um programa que além de registrar as notas, precise colocar os nomes, idade, endereço, telefone destes 100 alunos?

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Tema

- Variáveis compostas heterogêneas (struct).

Objetivos

- Entender como manipular estruturas;
- Entender a diferença entre um vetor (variáveis compostas homogêneas) e um estruturas (variáveis compostas heterogêneas);
- Desenvolver algoritmos com estruturas

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Problematização

- Em algumas situações no desenvolvimento de algoritmos, nos deparamos com problemas onde é desejável agrupar sob o mesmo nome um conjunto de variáveis de tipos distintos.
- Por exemplo agrupar os diferentes tipos de dados que pode conter o registro escolar de um aluno.

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Structs

- É uma maneira que temos para agrupar variáveis

- Se tiver um conjunto de dados que precisa unir de forma única, use struct

Structs

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

Cadastro pessoa Fisica

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

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

Número: 00001
Origem: _____Destino: _____
Data: ___/___/_____ Horário: _____:_____
Poltrona: _____ Distância: _____ km

Exemplo: Passagem de ônibus

Descrita como um conjunto de dados logicamente relacionados, porém de tipos diferentes, tais como:

Número da passagem (inteiro),

Origem e destino (caractere),

Data (caractere),

Horário (caractere),

Poltrona (inteiro),

Distância (real).

Structs

Exemplo: Dados de um aluno.

- Nome
- Matricula
- Sexo
- Cotista
- NotaSisu

```
nome: string;
matricula: integer;
sexo: char;
cotista: boolean;
notaSisu: real;
```

ALUNO

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**

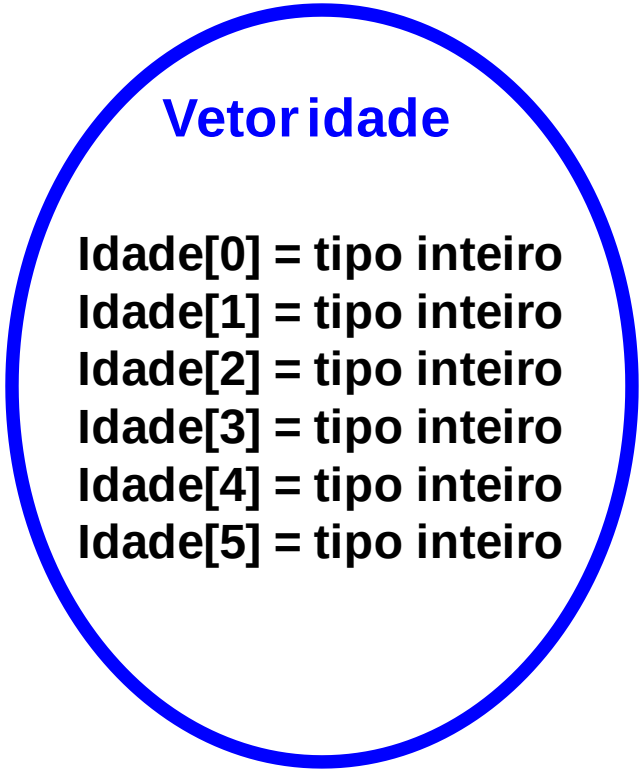
UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Variáveis Compostas Heterogêneas ou Registro

- Quando declaramos um vetor (Variáveis compostas homogêneas) sabemos que todas as variáveis (posições) contidas naquele vetor possuem o mesmo nome e mesmo tipo, porém são diferenciadas pelo seu índice.
- Ao declarar um registro (Variáveis compostas heterogêneas) sabemos que todas as variáveis contidas naquele registro possuem o nome do registro, no entanto elas podem ou não possuir o mesmo tipo.

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Structs



UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

O que são tipos de dados?

- Além dos tipos primitivos algumas linguagens permitem que o usuário defina novos tipos de dados através de declarações específicas
- Para utilizar a estrutura denominada registros deve-se informar **obrigatoriamente** as variáveis que estarão associadas ao registro.

OS TIPOS DEVEM SER DEFINIDOS ANTES DA DECLARAÇÃO DAS VARIÁVEIS

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 - declaração

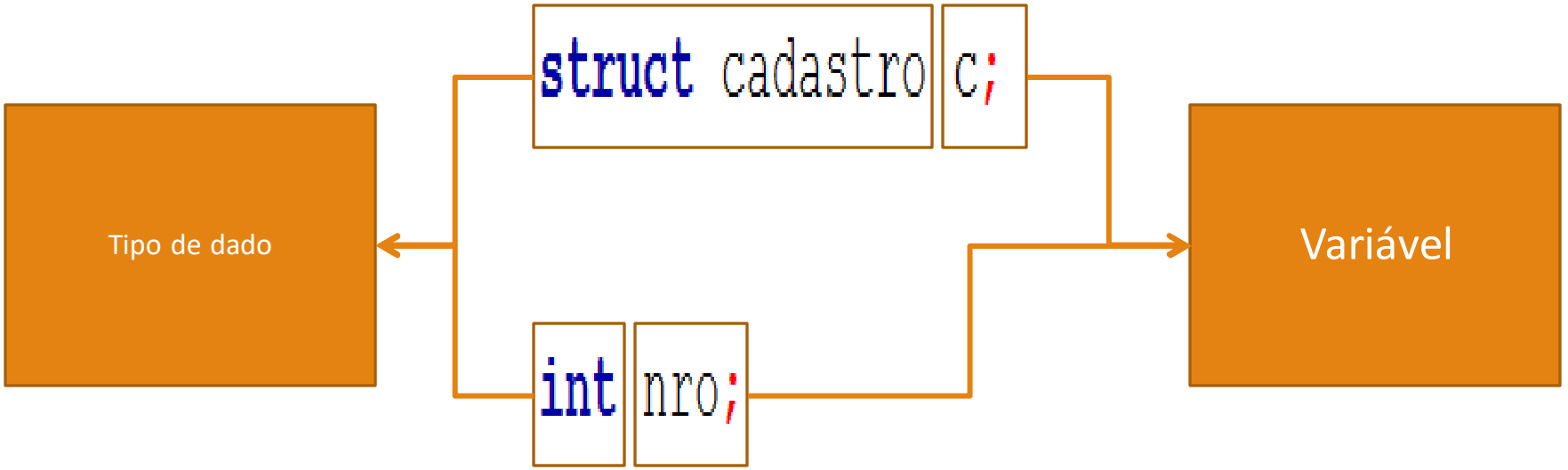
Uma vez definida a estrutura, uma **variável** pode ser declarada de modo similar aos tipos já existentes:

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

```
struct cadastro c;
```

Estruturas - declaração

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 “.”.

```
//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.

- Um **scanf** para **estrutura.idade**, não obriga a fazer um **scanf** para **estrutura.numero**

Estruturas

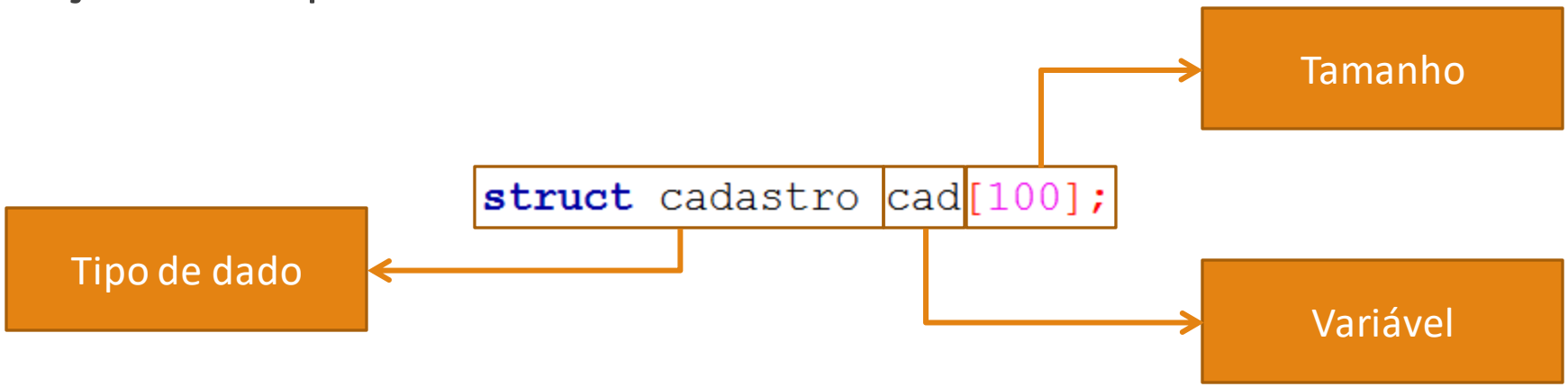
Voltando ao exemplo anterior, se, ao invés de 5 cadastros, quisermos fazer 100 cadastros dos alunos?

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

- **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;
};
```

cad[0]	cad[1]	cad[2]	cad[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;

```
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;
}
```

Array de estruturas

NUM ARRAY DE ESTRUTURAS, O OPERADOR DE PONTO (.) VEM DEPOIS DOS COLCHETES ([ ]) DO ÍNDICE DO **ARRAY**.

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;
};
```

```c
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

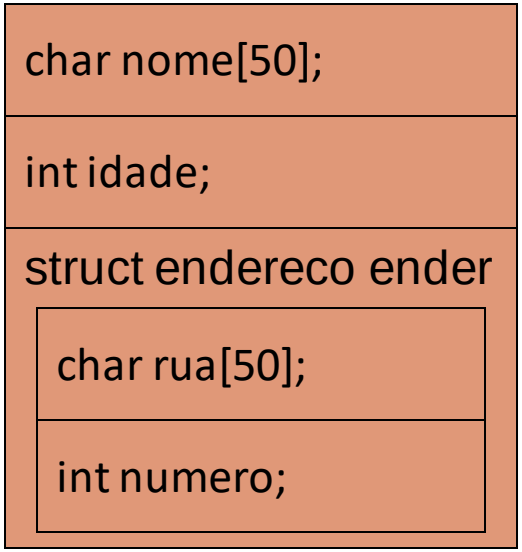
```
struct cadastro c[10];
c[1] = c[2]; //CORRETO
```

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;`

```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 COMANDO **TYPEDEF** NÃO CRIA UM NOVO TIPO CHAMADO **INTEIRO**. ELE APENAS CRIA UM SINÔNIMO (**INTEIRO**) PARA O TIPO **INT**

```c
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;
}
```

Comando typedef

O COMANDO TYPEDEF É USADO PARA CRIAR "SINÔNIMO" OU UM "ALIAS" PARA TIPOS DE DADOS EXISTENTES.

Exercícios

1. Defina um registro para informações de cartão de crédito.
2. Defina um registro para um cliente de uma empresa.
3. Defina um registro para cadastro de carros para uma concessionária.
4. Defina um registro para cadastro de roupas para uma loja.

Referências

BACKES, A. NOTAS DE AULA