

Aula 16: Variáveis compostas - Registros

Joice Otsuka

*Adaptado do livro de Andre Backes

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 **arrays (vetores e matrizes)**
- A linguagem C permite a criação de novas estruturas a partir dos tipos básicos, podendo reunir tipos distintos (heterogêneos), por meio dos **registros**
- Registros na linguagem C são conhecidos como **struct**

Registros

- Variáveis compostas heterogêneas
 - Uma única variável pode ser composta por partes de diferentes tipos
 - Agrupar informações que fazem sentido apenas juntas
 - Exemplo:
 - Cadastro de um usuário:
 - Nome
 - Idade
 - Rua
 - Número

Até agora ...

- O uso de e registros/estruturas facilita na manipulação dos dados do programa. Imagine declarar 4 cadastros, para 4 pessoas diferentes:

```
char nome1[100], nome2[100], nome3[100], nome4[100];
int idade1, idade2, idade3, idade4;
char rua1[100], rua2[100], rua3[100], rua4[100];
int numero1, numero2, numero3, numero4;
```

Registros

- Utilizando um registro, o mesmo pode ser feito da seguinte maneira:

```
#include <stdio.h>

int main()
{
    struct cadastro
    {
        char nome[100];
        int idade;
        char rua[100];
        int numero;
    };
    struct cadastro c1,c2,c3,c4;
    return 0;
}
```

Registros

- Registro: a estrutura definida como um todo, composta por campos
- Campo: cada elemento que compõe um registro
- **Ao definir um registro, um novo tipo de dado composto é criado:**

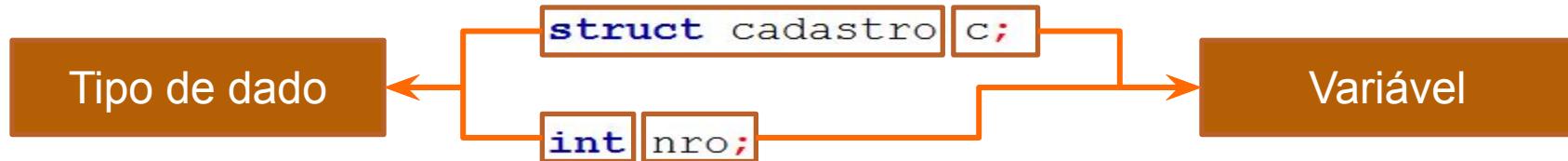
```
struct nomestruct {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
};
```

Registros

- Uma vez definida a estrutura de um registro, uma variável pode ser declarada usando o tipo definido:

```
struct cadastro c;
```

- 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 do RA e 3 notas para um dado aluno.

Exercício - Solução

- Possíveis soluções

```
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];  
};
```

Acesso aos campos de um registro

- Cada campo do registro pode ser acessado com o nome da variável seguido do operador ponto “.” e o nome do campo.
- Ex.:

```
int main(){
    struct cadastro{
        char nome[100];
        int idade;
        char rua[100];
        int numero;
    };
    struct cadastro c1, c2, c3;
    strcpy(c1.nome,"Luis Flavio");
    c1.idade = 20;
    strcpy(c1.rua, "Floriano Peixoto");
    c1.numero=358;
    printf("Nome: %s\n", c1.nome);
    printf("Idade: %d\n",c1.idade);
    printf("Endereço: %s, %d\n", c1.rua, c1.numero);
    return 0;
}
```

Inicialização

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

```
int main(){
    struct cadastro{
        char nome[100];
        int idade;
        char rua[100];
        int numero;
    };
    struct cadastro c1={"Luis Flavio", 20, "Floriano Peixoto", 358};
    printf("Nome: %s\n", c1.nome);
    printf("Idade: %d\n",c1.idade);
    printf("Endereço: %s, %d\n", c1.rua, c1.numero);
    return 0;
}
```

Atribuindo entrada de dados do teclado

- Podemos ler os valores dos campos do registro a partir do teclado.
- Ler cada campo independentemente, respeitando seus tipos.

```
int main(){
    struct cadastro{
        char nome[100];
        int idade;
        char rua[100];
        int numero;
    };
    struct cadastro c1;
    fgets(c1.nome,100,stdin);
    scanf("%d%c",&c1.idade);
    fgets(c1.rua,100,stdin);
    scanf("%d%c",&c1.numero);
    printf("Nome: %s", c1.nome);
    printf("Idade: %d\n",c1.idade);
    printf("Endereço: %s", c1.rua);
    printf("Número: %d\n",c1.numero);
    return 0;
}
```

O **%*c** solicita que um caractere seja ignorado (no caso será o '\n')

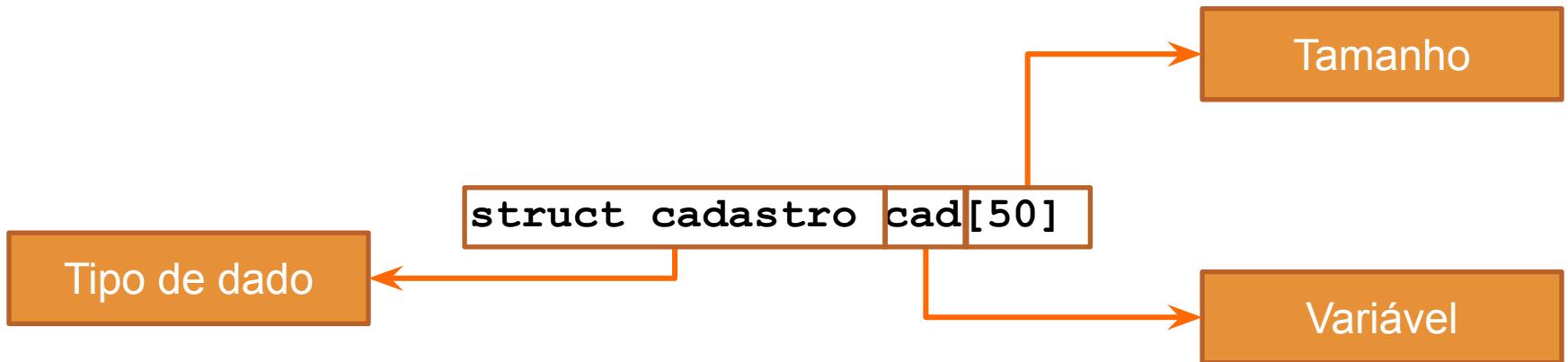
Acesso aos campos de um registro

- Note que cada campo dentro da estrutura pode ser acessado como se apenas ele existisse, não sofrendo nenhuma interferência dos demais.
 - 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**

Registros

Voltando ao exemplo anterior. E se, ao invés de cadastrar 3 pessoas, quisermos cadastrar 50 pessoas?

Vetor de registros



Vetor de registros

- **struct**: define um agrupamento de variáveis que podem ser de tipos diferentes;
- **vetor**: conjunto de elementos de mesmo tipo.

```
struct cadastro{  
    char nome[100];  
    int idade;  
    char rua[100];  
    int numero;  
};
```

```
char nome[100];  
int idade;  
char rua[100];  
int numero;
```

cad[0]

```
char nome[100];  
int idade;  
char rua[100];  
int numero;
```

cad[1]

```
char nome[100];  
int idade;  
char rua[100];  
int numero;
```

cad[2]

```
char nome[100];  
int idade;  
char rua[100];  
int numero;
```

... cad[49]

Vetor de registros

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

```
#include <stdio.h>
#define TAM 50

int main(){
    int i;
    struct cadastro{
        char nome[100];
        int idade;
        char rua[100];
        int numero;
    };
    struct cadastro c[TAM];
    for (i=0;i<TAM;i++){
        fgets(c[i].nome,100,stdin);
        scanf("%d%c",&c[i].idade);
        fgets(c[i].rua,100,stdin);
        scanf("%d%c",&c[i].numero);
    }
    return 0;
}
```

Exercício

- Utilizando a estrutura abaixo, faça um programa para ler o número, 3 notas e calcular/armazenar a média de 10 alunos.

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

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
}
```

Exercício

- Utilizando a estrutura abaixo, faça um programa para ler o número, 3 notas e calcular a média de 10 alunos.

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

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
}
```

Poderia ser um vetor

```
#include <stdio.h>

struct aluno{
    int num;
    float nota[3];
    float media;
};

int main()
{
    struct aluno alunos[10];
    float soma;

    for (int i=0;i<10;i++){
        soma=0;
        scanf("%d",&alunos[i].num);
        for (int j=0;j<3;j++){
            scanf("%f", &alunos[i].nota[j]);
            soma += alunos[i].nota[j];
        }
        alunos[i].media = soma/3;
        printf("Media: %.1f\n",alunos[i].media);
    }
    return 0;
}
```

Atribuição entre registros

- Atribuições entre registros só podem ser feitas quando os registros são do mesmo tipo (mesma estrutura)

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

```

/*
  Input: Um ponto em R^2
  Output: O mesmo ponto
*/
#include <stdio.h>
#include <math.h> // para sqrt e pow

int main(){
    struct{
        double x, y;
    } ponto_entrada;
    struct{
        double x, y;
    } ponto_saida;

    /* Entrada */
    printf("Coordenadas do ponto: ");
    scanf("%lf%lf", &ponto_entrada.x, &
        ponto_entrada.y);

    /* Copia de registros */
    ponto_saida = ponto_entrada;

    /* Resultado */
    printf("Ponto: (%g, %g)\n", ponto_saida.x,
        ponto_saida.y);

    return 0;
}

```

Qual o erro neste
programa?

Atribuição entre registros

- 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 os tipos dos elementos do array são sempre IGUAIS.

Registros de registros

- Inicialização de uma estrutura de estruturas:

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

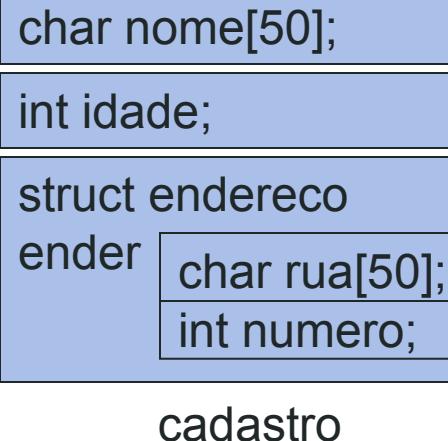
struct retangulo {
    struct ponto inicio, fim;
};

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

Registros de registros

- Sendo um registro um tipo de dado, podemos declarar um registro que utilize outro registro previamente definido:

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



Registros de registros

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

```
int main(){
    struct endereco{
        char rua[100];
        int numero;
    };
    struct cadastro{
        char nome[100];
        int idade;
        struct endereco ender;
    };
    struct cadastro c;
    fgets(c.nome,100,stdin);
    scanf("%d%c",&c.idade);
    fgets(c.ender.rua,100,stdin);
    scanf("%d%c",&c.ender.numero);
    return 0;
}
```

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**

```
#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

```
#include <stdio.h>

typedef struct {
    char nome[32];
    int idade;
} cadastro;

int main()
{
    cadastro aluno1 = {"Luiz Silva", 18};
    printf("Nome: %s -- Idade: %d\n",aluno1.nome,aluno1.idade);
    return 0;
}
```

Exercício

Escreva um código em C que defina um registro com os dados abaixo:

- Data com dia, mês e ano.

E outro registro contendo:

- Nome do empregado (máximo 30 caracteres);
- Data de contratação (use o registro de data);
- Data de demissão (use o registro de data);

O programa deve ler os dados de um registro e apresentá-los.

```
#include <stdio.h>
typedef struct {
    int dia;
    int mes;
    int ano;
} data;

typedef struct {
    char nome[32];
    data contratacao;
    data demissao;
} cadastro;

int main()
{
    cadastro c;
    fgets(c.nome,32,stdin);
    scanf("%d %d %d",&c.contratacao.dia,&c.contratacao.mes,&c.contratacao.ano);
    scanf("%d %d %d",&c.demissao.dia,&c.demissao.mes,&c.demissao.ano);
    printf("Nome: %s",c.nome);
    printf("Data de contrataC#o: %d/%d%d\n",c.contratacao.dia,c.contratacao.mes,c.contratacao.ano);
    printf("Data de demissC#o: %d/%d/%d\n",c.demissao.dia,c.demissao.mes,c.demissao.ano);
    return 0;
}
```

Exercícios

1. Escreva um programa que cadastre o nome, o RA e duas notas de 100 alunos. Em seguida, imprima um relatório com a matrícula, o nome e a média dos alunos:
 - a. Com a maior média da turma
 - b. Com a menor média da turma
 - c. Com média acima da média da turma.

Exercício

Escreva um programa que cadastre o nome, a altura, o peso, o cpf e gênero de 100 pessoas. Com os dados cadastrados, solicite o CPF de uma pessoa e localize e imprima o seu IMC. Obs: dado o peso p em quilogramas e a altura h em centímetros, o IMC é calculado por:

$$\text{IMC} = p/(h^*h)$$

Exercício

Escreva um programa em C que use registros para representar pontos no espaço cartesiano \mathbb{R}^3 . O programa deve ler os dados de dois pontos e apresentar o ponto médio entre eles.

O ponto médio entre (x_1, y_1, z_1) e (x_2, y_2, z_2) é dado por

$$\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}, \frac{z_1 + z_2}{2} \right).$$

Referências

- BACKES, André. **Linguagem C:** completa e descomplicada. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3. Disponível na Biblioteca.