Engenharia de Computação

# Fundamentos de Programação

## Aula 14 – Estrutura (Struct) e Definição de Tipo

**Prof. Muriel de Souza Godoi**
muriel@utfpr.edu.br

# Estruturas Heterogêneas

- Em um programa, há várias variáveis associadas à realização de uma tarefa específica;
  - É conveniente ter um modo de agrupar um conjunto de variáveis relacionadas;
- Vetores e matrizes agrupam uma série de variáveis homogêneas, ou seja, do **mesmo tipo**.
- Se quisermos um tipo de agrupamento heterogêneo, que englobe variáveis de **tipos diferentes**, no qual uma possa ser identificada por um nome específico usamos **estruturas** ou **registros**.

# Definição de Estrutura Heterogênea

- É um **conjunto de variáveis de diferentes tipos** que tem uma relação;
- Também é conhecido como **estrutura** ou **registro**
  - Por exemplo, uma **estrutura** que representa um Aluno pode conter as variáveis:
    - Nome, idade, peso e altura;
  - Essas variáveis são denominadas de **membros** ou **campos** da estrutura;
    - Criamos uma variável que contém dentro de si outras variáveis.
- Permite que o usuário “defina” **seus próprios tipos de dados** a partir dos tipos primitivos da linguagem (*int, float, char, etc*.);

# Definindo uma estrutura

- Uma **estrutura** em C é definida usando a palavra-chave **struct** seguida de um bloco (delimitado por chaves) contendo as declarações dos membros, como se fossem declaração de variáveis comuns.

```
struct Aluno{
    char nome[50];
    int idade;
    float peso;
    float altura;
};//struct
```

- **Atenção** para o ponto e vírgula.
  - A definição da struct é um comando.

- Declaração de 3 variáveis do tipo Aluno

```
struct Aluno aluno_1, aluno_2, aluno_3;
```

# Acesso aos membros

- Para acessar campos de uma estrutura, usamos o operador . (um ponto)
  - À esquerda o nome da variável que contém a estrutura
  - À direita o nome do membro (ou campo).
- No exemplo anterior:

```
struct Aluno{
    char nome[50];
    int idade;
    float peso;
    float altura;
};//struct
struct Aluno aluno_1, aluno_2, aluno_3;
```

- Pode-se acessar a idade do aluno_1 usando:

```
aluno_1.idade
```

# Resumo

- Definição de uma estrutura

```
struct nome_da_estrutura{
    //declaração dos membros
};//struct
```

- Declaração de uma variável do tipo struct

```
struct nome_da_estrutura nome_da_variável;
```

- Acesso aos membros da estrutura:

```
nome_da_variável.nome_do_membro;
```

# Inicializando

- Pode-se inicializar a struct de forma estática, como ocorre com variáveis comuns, vetores e matrizes;

```
struct Aluno{
    char nome[50];
    int idade;
    float peso;
    float altura;
};//struct

struct Aluno aluno1 = {“Rosinosberto”, 53, 72.5, 1.64 };
```

# Atribuindo

- Uma variável estrutura pode ser atribuída a outra do mesmo tipo por meio de uma atribuição simples

```
struct Aluno{
    char nome[50];
    int idade;
    float peso;
    float altura;
};//struct

struct Aluno meu_aluno = {“Rosinosberto”, 53, 72.5, 1.64 };
struct Aluno seu_aluno;

seu_aluno = meu_aluno;
```

# Vetor de Estruturas

- Podemos criar um vetor de structs

```
struct Cliente {
  char nome[21];
  char endereco[31];
  int idade;
};//struct

struct Cliente cadastro[3] = {
  { "Paulo Teste", "Rua do Torto, 1234", 56 },   //índice 0 do vetor
  { "Ze Roberto", "Rua dos Robertos, 4321", 65 },//índice 1 do vetor
  { "Maria Carla", "Av. Sem Fundo, 778", 28 }    //índice 2 do vetor
};
```

  - Ao atribuir uma posição para outra copia todos os membros

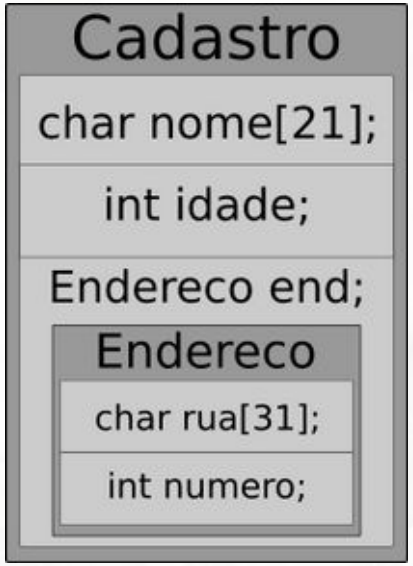
```
    cadastros[0] = cadastros[1];
```

# Estruturas Aninhadas

- Podemos aninhar estruturas

```
struct Endereco {
  char rua[31];
  int numero;
};//struct Endereço

struct Cadastro {
  char nome[21];
  int idade;
  struct Endereco end;
};//struct Cadastro
```




- E acessar os membros usando .

```
  struct Cadastro cad;
  cad.end.numero = 1234;
```

# Typedef – Criando novos tipos

- É possível definir um tipo baseado em outro tipo ou estrutura

```
typedef tipo_existente tipo_novo;
```

- No caso das estruturas:
  - Basta utilizar **typedef** na definição

```
typedef struct {
    char nome[50];
    int idade;
    float peso;
    float altura;
} Aluno;//struct

Aluno meu_aluno;
```

Agora **Aluno** é o nome do tipo

Agora não existe mais a necessidade de:

```
struct Aluno meu_aluno;
```

# Typedef

- Percebendo na prática a diferença do uso de estruturas com typedef

## Sem typedef

```c
struct MeuCadastro {
  char nome[21];
  char endereco[31];
  int idade;
};//struct MeuCadastro

int main( ) {
  struct MeuCadastro X[10];
  return 0;
}//main
```

## Com typedef

```c
typedef struct {
  char nome[21];
  char endereco[31];
  int idade;
} MeuCadastro;

int main( ) {
  MeuCadastro X[10];
 return 0;
}//main
```

# Exercícios Struct e Typedef

- **1)** Crie uma estrutura para representar as coordenadas de um ponto no plano (posições X e Y). Em seguida, declare e leia do teclado dois pontos e exiba a distância entre eles.
- **2)** Escreva um trecho de código para fazer a criação dos novos tipos de dados conforme solicitado abaixo:
  - Horário: composto de hora, minutos e segundos.
  - Data: composto de dia, mês e ano.
  - Compromisso: local, horário e texto que descreve o compromisso.

# Exercícios Struct e Typedef

- **3)** Construa uma estrutura aluno com nome, curso e 4 notas, média e situação. Leia as informações nome, curso e notas do teclado, calcule a média e armazene a situação do aluno.
  - media ≥ 7 → Aprovado;
  - 3 ≤ media < 7 → Exame;
  - media < 3 → Reprovado;
- **4)** Faça um programa que controla o consumo de energia dos eletrodomésticos de uma casa e:
  - Crie e leia 5 eletrodomésticos que contém nome (máximo 15 letras), potência (real, em kW) e tempo ativo por dia (real, em horas).
  - Leia um tempo t (em dias), calcule e mostre o consumo total na casa e o consumo relativo de cada eletrodoméstico (consumo/consumo total) nesse período de tempo. Apresente este último dado em porcentagem.