

# LÓGICA DE PROGRAMAÇÃO

PROF<sup>a</sup>. M.Sc. JULIANA H Q BENACCHIO

# Estrutura (Registros) - struct

- Tipos de dados definidos pelo usuário.
- Um registro agrupa várias variáveis numa só, formando um novo tipo de dados.
- Também são chamados de **Estruturas de Dados Heterogêneas**, pois um registro é uma coleção de uma ou mais variáveis agrupadas (campos), possivelmente de tipos diferentes, sob um mesmo nome.

# Estrutura (Registros) - struct

- Sintaxe:

```
struct nome_do_tipo_da_estrutura {  
    tipo_1 nome_1;  
    tipo_2 nome_2;  
    ...  
    tipo_n nome_n;  
};
```

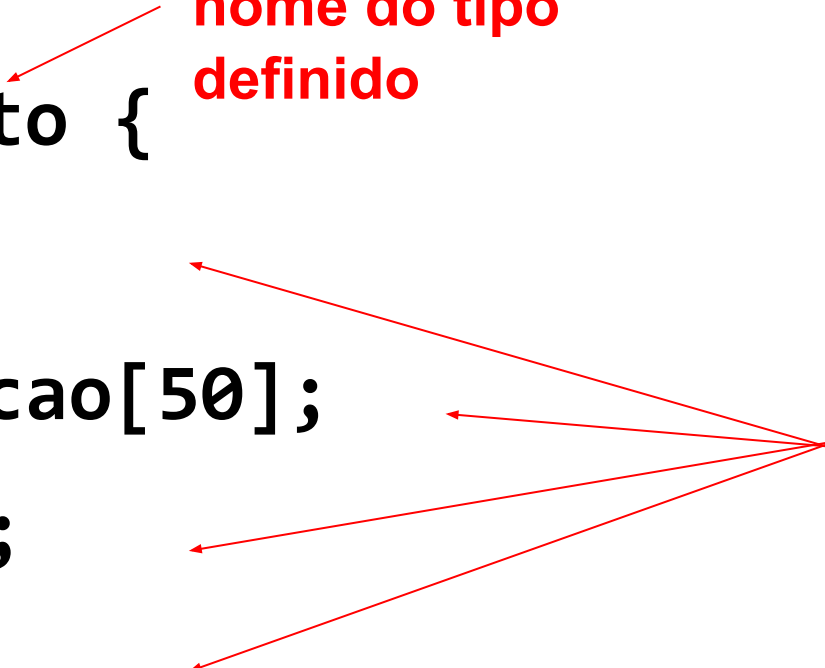
# Estrutura (Registros) - struct

- Por exemplo, a ficha cadastral de um produto:

```
struct TProduto {  
    int codigo;  
    char descricao[50];  
    float valor;  
    int qtde;  
};
```

**nome do tipo definido**

**campos**



# Estrutura (Registros) - struct

- Ou para armazenar as informações de um livro:

```
struct TLivro {  
    int cod;  
    char titulo[30];  
    char autor[20];  
    char editora[20];  
    int ano;  
};
```

# Estrutura (Registros) - `struct`

- A definição termina com um `(;)`. Isso ocorre porque uma definição de estrutura é um comando.
- Nos códigos anteriores, nenhuma variável foi de fato declarada. Apenas a forma dos dados foi definida.

# Estrutura (Registros) - `struct`

---

- Quando se define uma estrutura, está essencialmente definindo um tipo complexo de variável, não uma variável. Não existe uma variável desse tipo até que ela seja realmente declarada.

# Estrutura (Registros) - struct

- Declarar uma variável do tipo **TProduto**

```
struct TProduto estoque;
```

- Para preencher os dados, o acesso de um elemento da estrutura deve ser feito utilizando o ponto e o nome do campo.

```
estoque.codigo = 1;
```

```
strcpy(estoque.descricao, "Prego");
```



# Estrutura (Registros) - struct

- Também é possível declarar uma ou mais variáveis ao definir a estrutura.

```
struct TLivro {  
    char titulo[30];  
    char autor[20];  
    char editora[20];  
    int ano;  
} ficcao, romance, didatico;
```

# Estrutura (Registros) - struct

- Se for apenas uma variável estrutura, o nome da estrutura não é necessário

```
struct {  
    char titulo[30];  
    char autor[20];  
    char editora[20];  
    int ano;  
} livro_didatico;
```

# Estrutura (Registros) - struct

- Portanto, a forma geral de uma definição de estrutura é

```
struct identificador {  
    tipo nome_da_variavel;  
    tipo nome_da_variavel;  
    ...  
} variáveis_estrutura;
```

Onde `identificador`  
ou  
`variáveis_estrutura`  
podem ser omitidos,  
mas não ambos.

# Exemplo 1

```
#include <stdio.h>

int main()
{
    struct TProduto {
        int codigo;
        char descricao[50];
        float valor;
        int qtde;
    };

    struct TProduto estoque;
    float total;
```

# Exemplo 1

```
estoque.codigo = 1;
printf("Dados para o Produto\n");
printf("Descricao: ");
fgets(estoque.descricao, 50, stdin);
printf("Valor: ");
scanf("%f", &estoque.valor);
printf("Quantidade: ");
scanf("%d", &estoque.qtde);
total = estoque.valor * estoque.qtde;
printf("Total em estoque = %.2f", total);

return 0;
}
```

# Estrutura (Registros) - struct

- Exemplo: Ficha pessoal com dados de contato.
- Uma estrutura pode fazer parte de outra

```
struct ficha_pessoal
{
    char nome [50];
    char telefone[12];
    struct tipo_endereco endereco;
};
```

# Estrutura (Registros) - struct

- Deve ser declarada antes da sua utilização

```
struct tipo_endereco {  
    char rua [50];  
    char numero[5];  
    char bairro [20];  
    char cidade [30];  
    char sigla_estado [3];  
    char cep[10];  
};
```

# Estrutura (Registros) - struct

- Para acessar os campos da estrutura interna endereço que faz parte da estrutura ficha, acessa-se primeiro a variável do tipo e depois o campo:

```
ficha.endereco.numero = 1250;
```



# Exemplo 2

...

```
struct tipo_endereco {  
    char rua [50];  
    char numero[5];  
    char bairro [20];  
    char cidade [30];  
    char sigla_estado [3];  
    char cep[10];  
};  
  
struct ficha_pessoal  
{  
    char nome [50];  
    char telefone[12];  
    struct tipo_endereco endereco;  
};  
  
struct ficha_pessoal contatos;
```

# Exemplo 2

```
printf("Inserir Contato\n");  
printf("Nome: ");  
fgets(contatos.nome, 50, stdin);  
printf("Telefone: ");  
fgets(contatos.telefone, 12, stdin);  
printf("Endereço\n");  
printf("Rua: ");  
fgets(contatos.endereco.rua, 50, stdin);  
printf("Numero: ");  
fgets(contatos.endereco.numero, 5, stdin);  
printf("Bairro: ");  
fgets(contatos.endereco.bairro, 20, stdin);
```

...

# Atribuição de estruturas

- A informação contida em uma estrutura pode ser atribuída a outra estrutura do mesmo tipo
- Em vez de atribuir os valores de todos os elementos separadamente, é possível utilizar apenas um único comando de atribuição

# Exemplo 3

```
struct coordenada{  
    int x;  
    int y;  
} p1, p2;  
  
p1.x = 5;  
p1.y = 3;  
  
p2 = p1;  
  
printf("Coordenadas\n");  
printf("Ponto 1: %d,%d\n", p1.x, p1.y);  
printf("Ponto 2: %d,%d\n", p2.x, p2.y);
```

- Uma estrutura é como qualquer outro tipo de dado no C
- Portanto, é possível criar vetores de estruturas
- Por exemplo, para criar um vetor de 100 produtos:

```
struct TProduto estoque[100];
```

- Para acessar o conteúdo de uma estrutura específica, deve-se indexar o nome da estrutura e então acessar o campo:

```
estoque[4].valor = 2.50;
```

```
printf("%s", estoque[4].descricao);
```

# Vetor de Estruturas

```
estoque[0].codigo = 1;  
strcpy(estoque[0].descricao, "prego");  
estoque[0].valor = 0.20;  
estoque[0].qtde = 1000;
```

# Vetor de Estruturas

---

```
estoque[1].codigo = 2;  
strcpy(estoque[1].descricao, "parafuso");  
estoque[1].valor = 0.50;  
estoque[1].qtde = 500;
```



# Exemplo 4

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

int main()
{
    struct TProduto {
        int codigo;
        char descricao[50];
        float valor;
        int qtde;
    };

    struct TProduto estoque[5];
    int i, b;
    float total;
```

# Exemplo 4

```
for (i=0; i < 5; i++){  
    estoque[i].codigo = i+1;  
    printf("Dados para o Produto %d\n", (i+1));  
    printf("Descricao: ");  
    __fpurge(stdin); //limpa o buffer do teclado  
    fgets(estoque[i].descricao, 50, stdin);  
    printf("Valor: ");  
    scanf("%f", &estoque[i].valor);  
    printf("Quantidade: ");  
    scanf("%d", &estoque[i].qtde);  
}
```

# Exemplo 4

```
printf("\nQual produto deseja visualizar: ");
scanf("%d", &b);
printf("Dados do Produto %d\n", b);
printf("Descricao: %s\n", estoque[b-1].descricao);
printf("Valor: %.2f\n", estoque[b-1].valor);
printf("Quantidade: %d\n", estoque[b-1].qtde);
total = estoque[b-1].valor * estoque[b-1].qtde;
printf("Total em estoque = %.2f", total);

return 0;

}
```

# Exemplo 4



```
Consola
Dados para o Produto 1
Descricao: prego
Valor: 0.5
Quantidade: 100

Qual produto deseja visualizar: 1
Dados do Produto 1
Descricao: prego

Valor: 0.50
Quantidade: 100
Total em estoque = 50.00

-----
(program exited with code: 0)
Press return to continue
█
```

# Exemplo 4

- Retirando o ' \n ' do fgets

```
for (i=0; i < 5; i++){
    estoque[i].codigo = i+1;
    printf("Dados para o Produto %d\n", (i+1));
    printf("Descricao: ");
    __fpurge(stdin);
    fgets(estoque[i].descricao, 50 , stdin);
    estoque[i].descricao[strlen(estoque[i].descricao)-1] = '\0';
    printf("Valor: ");
    scanf("%f", &estoque[i].valor);
    printf("Quantidade: ");
    scanf("%d", &estoque[i].qtde);
}
```

# Exemplo 4



```
Consola
Dados para o Produto 1
Descricao: prego
Valor: 0.50
Quantidade: 100

Qual produto deseja visualizar: 1
Dados do Produto 1
Descricao: prego
Valor: 0.50
Quantidade: 100
Total em estoque = 50.00

-----
(program exited with code: 0)
Press return to continue
█
```