5950257 - Programação de Computadores

Aula 12 - Structs

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Objetivos de Aprendizagem

1. **Diferenciar** variáveis homogêneas (arrays) de variáveis heterogêneas (structs).
2. **Compreender** a necessidade lógica de agrupar dados relacionados sob um único identificador.
3. **Implementar** estruturas (`struct`) em C para modelar objetos do mundo real.
4. **Utilizar** estruturas aninhadas para criar modelos de dados complexos (composição).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Variáveis Compostas

- Em programação, nem sempre os dados que queremos manipular são do mesmo tipo (inteiros, floats, chars).
- **Problema:** Como representar um "Aluno" ou um "Produto"? Eles possuem nomes (string), preços (float) e IDs (int).
- **Solução:** Variáveis Compostas Heterogêneas.
  - Permitem agrupar dados de tipos diferentes que possuem uma **unidade lógica**.
  - Exemplo: Um registro de "Funcionário" une nome, salário e cargo em um único conceito.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conceito de Registro

Um Registro é um conjunto de posições de memória conhecidas por um mesmo nome, mas individualizadas por identificadores internos.

- Enquanto um *Array* organiza dados do mesmo tipo, o *Registro* organiza dados de tipos diferentes que pertencem ao mesmo contexto.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Struct

O compilador utiliza a palavra-chave **struct** para definir um novo tipo de dado personalizado.

- **Sintaxe de Definição:**

```
struct nome_da_estrutura {
    tipo nome_do_elemento_1;
    tipo nome_do_elemento_2;
    // ... outros elementos
};
```

> **Observação:** A definição de uma **struct** não aloca memória imediatamente; ela cria um "molde" (blueprint) para o compilador.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Struct: Definindo uma Nova Estrutura

Exemplo de estrutura:

```c
// Definição da Estrutura
struct Pessoa {
    char nome[50];
    int idade;
    float altura;
};
```

> **Alocação de memória:** O compilador calcula automaticamente o espaço necessário somando os tamanhos de todos os membros.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Struct: Criando Instâncias

Para instanciar uma variável do tipo **struct**, basta utilizar o nome especificado na definição dela:

```c
// Declaração e Inicialização Simultânea
struct Pessoa p1 = { "Ana", 25, 1.65 };

// Declaração Separada
struct Pessoa p2;
p2.nome[0] = 'A'; // Acesso direto ao membro
```

> Note que a inicialização pode ser feita no momento da declaração ou posteriormente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Struct: Accesando Componentes

Quando você tem a variável instanciada é possível usar o operador ponto `.` para acessar cada componente da estrutura:

```
struct Pessoa p1 = { "Ana", 25, 1.65 };
p1.idade = 26;
printf("%d", p1.altura);
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo

```c
#include <stdio.h>
#include <string.h> // Necessário para usar a função strcpy

// DEFINIÇÃO
struct endereco {
    char rua[50];         // Um array de caracteres para armazenar o nome da rua
    unsigned int numero;  // Um inteiro sem sinal (não negativo) para o número
};

int main() {
    // INSTANCIAÇÃO
    struct endereco casa1;

    // ATRIBUIÇÃO DE VALORES
    strcpy(casa1.rua, "Rua das Flores");
    casa1.numero = 120;

    // IMPRESSÃO DE DADOS
    printf("Logradouro: %s\n", casa1.rua);
    printf("Número:     %u\n", casa1.numero);

    return 0;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Structs Aninhadas (Composição)

Uma estrutura pode conter outra estrutura como membro. Isso permite criar modelos de dados complexos e modulares.

Exemplo: Em vez de colocar todos os campos de endereço em uma struct "Pessoa", criamos uma struct "Endereco" e a incluímos dentro da struct "Pessoa".

```
struct endereco {
    char rua[50];
    int numero;
};

struct pessoa {
    char nome[40];
    struct endereco ondeMora; // Estrutura aninhada
};
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Array de Structs

Quando precisamos gerenciar múltiplos registros do mesmo tipo (ex: uma lista de alunos, um inventário de produtos).

Sintaxe de Acesso: **nome_da_variável[índice].membro**

```c
struct aluno {
    char nome[30];
    float nota;
};

// Criação de uma lista de 50 alunos:
struct aluno lista_alunos[50];

// Acesso ao nome do primeiro aluno:
printf("%s", lista_alunos[0].nome);
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício

Implemente um sistema de cadastro de alunos.

1. Defina uma **struct Aluno** contendo: **nome**, **matricula** e **nota**.
2. Crie um array de estruturas para armazenar até 10 alunos.
3. O programa deve permitir ao usuário cadastrar quantos alunos ele desejar.
4. **Condição de Parada:** O programa deve parar de aceitar novos cadastros quando o usuário digitar **-1** no campo da matrícula.
5. Ao final, exiba a lista completa de alunos cadastrados.
6. Exiba também a média de notas dos alunos.

Passando Estruturas como Argumentos de Função

Passagem por valor: A função recebe uma **cópia completa** de toda a estrutura.

- Qualquer alteração feita dentro da função **não afeta** o original (isolamento).
- Se a **struct** for grande, copiar todos os bytes leva tempo e memória extra.

```c
// Passagem por valor (apenas leitura)
void exibir_dados(struct Paciente p) {
    printf("[Função Valor] Nome: %s, Sanguíneo: %s\n", p.nome, p.tipo_sanguineo);
    // Se tentarmos mudar aqui: p.id_registro = 0; -> Apenas a cópia é alterada!
}
```

Passando Estruturas como Argumentos de Função (cont.)

Passagem por referência: A função recebe apenas o **endereço de memória** do original (um ponteiro).

- Não há cópia de dados, apenas a passagem de um endereço, independentemente do tamanho da struct.
- Se a função modificar os dados através deste ponteiro (**`*p`**), o dado original será alterado permanentemente.

```c
// Passagem por referência (permite modificação)
void atualizar_sanguineo(struct Paciente *p, const char* novo_tipo) {
    // Usamos o operador '*' para desreferenciar e acessar o conteúdo no endereço 'p'.
    strcpy(p->tipo_sanguineo, novo_tipo);
}
```

Manipulando Estruturas via Ponteiros

Quando temos ponteiros para estruturas, usamos o operador de seta `->` para acessar componentes, como no exemplo do slide anterior.

```c
struct Pessoa p1 = { "Ana", 25, 1.65 };
struct Pessoa *ptr = &p1; // ptr aponta para p1

// Acesso via ponteiro (equivalente a p1.nome)
printf("%s", ptr->nome);
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Conclusão

Structs são a base para a criação de tipos de dados personalizados em C. Elas permitem unir tipos heterogêneos em uma unidade lógica.

Acesso aos Componentes:

- Variável direta: **variavel.membro** (Operador **.**).
- Via ponteiro: **ponteiro->membro** (Operador **->**).

Semântica de Chamada:

- Passar por Valor: Cria cópia (seguro, mas lento para structs grandes).
- Passar por Ponteiro: Passa endereço (rápido, permite modificação direta).

Material Adicional:

- Vídeo no YouTube: Estruturas em Linguagem C - structs pelo Prof. Gustavo Kikee e
- Exemplos de código: Estruturas - Prof. Carlos Maziero)

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercícios e Questões

Exercício 1

Crie um programa em C que defina uma estrutura chamada **Produto**. Ela deve conter: **nome** (string), **preco** (float) e **codigo** (int). O programa deve ler os dados de **um** produto e exibir na tela no formato abaixo.

```
Produto: [Nome] | Pre_R$ [Preço] | ID: [Código]
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exercício 2

Você deve criar um programa em C que simule o atendimento médico. O objetivo é verificar se um paciente está apto para receber um medicamento específico, baseado no seu tipo sanguíneo e na sua idade mínima.

1. Utilize a **struct Paciente** (Nome, ID, Ano de Nascimento, Tipo Sanguíneo).
2. Crie uma função chamada **verificaMaiorIdade(struct Paciente p)** que recebe o paciente por valor e retorna um booleano (**int** onde 1 é verdadeiro) se ele tiver mais de 18 anos.
3. Crie a função abaixo que recebe o paciente por ponteiro e verifica se o tipo sanguíneo do paciente for compatível com o medicamento.

```
void verificaMedicamento(struct Paciente *p, const char* medicamento) {...}
```

Questões

1. Explique a diferença fundamental entre um Array de Inteiros e uma Estrutura (**struct**) em C.
2. O que acontece com o espaço de memória quando declaramos uma **struct** mas não criamos nenhuma variável a partir dela?
3. Por que o uso de estruturas aninhadas é considerado uma boa prática?
4. Explique em termos de memória por que passar uma **struct** por valor pode ser ineficiente, especialmente se a estrutura contiver muitos campos grandes (ex: arrays de caracteres).