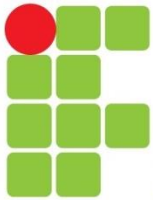

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MATO GROSSO DO SUL

Algoritmos II

Estruturas/Struct

Prof. Rogério Alves dos Santos Antoniassi
Prof. Alex Fernando de Araujo

Introdução

- Estrutura é um tipo de dado definido pelo usuário que permite agrupar múltiplas variáveis (até mesmo de tipos diferentes).

- Também chamada de registros.

- Definição segundo Ascencio e Campos: "Uma variável composta heterogênea, também conhecida como registro, é uma estrutura de dados que agrupa itens que podem ser de tipos diferentes." (Ascencio e Campos, "Fundamentos da Programação de Computadores", p. 211).

Exemplos de uso

- Para representar entidades do mundo real (ex: um aluno com nome, matrícula, notas).

- Para organizar dados que logicamente pertencem juntos (ex: uma data com dia, mês, ano).

- Para retornar múltiplos valores de uma função (agrupados em uma struct).

Vantagens

- Organização Lógica: Mantém dados relacionados juntos.
- Legibilidade: Torna o código mais claro e fácil de entender.
- Manutenção: Facilita a modificação e expansão de estruturas de dados.
- Modularidade: Permite passar conjuntos complexos de dados para funções de forma encapsulada.
- Modelagem: Ideal para representar objetos ou conceitos do mundo real.

Declarando uma estrutura

- Sintaxe Geral: "A palavra-chave struct inicia a declaração da estrutura." (Backes, "Linguagem C - Completa e Descomplicada", p. 231).

```
struct nome_da_tag {
        tipo_do_membro1 nome_membro1;
        tipo_do_membro2 nome_membro2;
        // ... demais
        };
```

Exemplo

- Exemplo (Adaptado de Ascencio e Campos, p. 213):

```c
struct Data {
    int dia;
    int mes;
    int ano;
};

struct Aluno {
    char nome[100];
    int matricula;
    float media_geral;
    struct Data data_nascimento; // Estrutura aninhada
};
```

Definindo Variáveis do Tipo Estrutura

- Definida a estrutura, você pode declarar variáveis desse tipo:

```
struct Aluno aluno1;
struct Data hoje;
```

Acessando Membros da Estrutura

- Sintaxe: nome_da_variavel_estrutura.nome_do_membro

- Exemplo (Continuando o anterior, adaptado de Ascencio e Campos, p. 213-214):

```c
#include <stdio.h>
#include <string.h>
// (Definições das structs Data e Aluno aqui)
int main() {
    struct Aluno aluno1;

    strcpy(aluno1.nome, "Rogério Antoniassi");
    aluno1.matricula = 1234123;
    aluno1.media_geral = 7.5;
    aluno1.data_nascimento.dia = 22;
    aluno1.data_nascimento.mes = 9;
    aluno1.data_nascimento.ano = 2000;

    printf("Nome: %s\n", aluno1.nome);
    printf("Matricula: %d\n", aluno1.matricula);
    printf("Data de Nascimento: %d/%d/%d\n",
           aluno1.data_nascimento.dia,
           aluno1.data_nascimento.mes,
           aluno1.data_nascimento.ano);

    return 0;
}
```

Arrays de Estruturas

- Podemos criar arrays cujos elementos são estruturas. Isso é útil para armazenar uma coleção de registros.

- Declaração (Adaptado de Ascencio e Campos, p. 216):

```
struct Aluno turma[30]; // Declara um array para 30 alunos
```

- Acessando Membros:

```
nome_do_array[indice].nome_do_membro
```

Arrays de Estruturas - Exemplo

```c
#include <stdio.h>
#include <string.h>

// (Definição da struct Aluno aqui)

int main() {
    struct Aluno turma[2]; // Um pequeno array para exemplo

    strcpy(turma[0].nome, "Aluno Teste");
    turma[0].matricula = 2025002;
    turma[0].media_geral = 7.0;

    strcpy(turma[1].nome, "Aluno Teste 1");
    turma[1].matricula = 2025003;
    turma[1].media_geral = 7.5;

    for (int i = 0; i < 2; i++) {
        printf("Aluno %d: %s, Matricula: %d\n",
               i + 1, turma[i].nome, turma[i].matricula);
    }
    return 0;
}
```

Inicialização de Estruturas

- Estruturas podem ser inicializadas no momento da sua declaração, fornecendo valores para seus membros entre chaves {}.

- Sintaxe e Exemplo (Adaptado de Damas, p. 186):

```
struct Ponto {
    int x;
    int y;
};

struct Ponto p1 = {2, 5};
```

Inicialização de Estruturas

```
struct Aluno aluno_exemplo = {
    "Aluno Teste",          // nome
    1234123,                  // matricula
    7.5,                      // media_geral
    {22, 9, 2000}             // data_nascimento (dia, mes, ano)
};

printf("Nome: %s, Data Nasc: %d/%d/%d\n",
       aluno_exemplo.nome,
       aluno_exemplo.data_nascimento.dia,
       aluno_exemplo.data_nascimento.mes,
       aluno_exemplo.data_nascimento.ano);
```

typedef e Estruturas

- A palavra-chave typedef pode ser usada para criar um "apelido" (um novo nome) para um tipo de estrutura. Isso pode tornar o código mais legível e simplificar a declaração de variáveis.
- Exemplo (Adaptado de Backes, p. 242, e Damas, p. 190):

```c
// Sem typedef
struct _Horario {
    int hora;
    int minuto;
    int segundo;
};
struct _Horario agora_sem_typedef;

// Com typedef
typedef struct {
    int hora;
    int minuto;
    int segundo;
} Horario; // 'Horario' é o novo nome para o tipo struct

Horario agora_com_typedef; // Declaração mais simples
```

typedef e Estruturas

```c
// Ou nomeando a tag e usando typedef
typedef struct _Data {
    int dia;
    int mes;
    int ano;
} DataInfo;

DataInfo data_evento; // Uso do tipo definido com typedef
// struct _Data outra_data; // Também é válido se a tag _Data for mantida
```

Ponteiros para Estruturas

- Acessando Membros via Ponteiro: Há duas formas principais:
  - Operador Seta (->): É a forma mais comum e legível. ptr_produto->preco
- Desreferenciação e Ponto ((*).):
  - (*ptr_produto).preco

- "Para aceder aos membros de uma estrutura através de um ponteiro para essa estrutura utiliza-se o operador -> (seta)." (Damas, "Linguagem C", p. 187).

- "O operador -> é usado quando temos um ponteiro para uma estrutura." (Backes, "Linguagem C - Completa e Descomplicada", p. 247).

Ponteiros para Estruturas - Exemplo

```c
#include <stdio.h>
#include <string.h>

typedef struct {
    char titulo[100];
    char autor[100];
    int ano_publicacao;
} LivroInfo;

// Função recebe um ponteiro para LivroInfo
void atualizarPrecoLivro(LivroInfo *livro, int novo_ano) {
    if (livro != NULL) { // Boa prática: verificar se o ponteiro não é nulo
        livro->ano_publicacao = novo_ano;
        // Exemplo: strcpy(livro->autor, "Autor Atualizado");
        printf("Informacoes do livro atualizadas dentro da funcao.\n");
    }
}

int main() {
    LivroInfo meu_livro = {"Programacao em C", "Autor Original", 2020};

    printf("Ano antes: %d\n", meu_livro.ano_publicacao);

    atualizarPrecoLivro(&meu_livro, 2024); // Passa o endereço da struct

    printf("Ano depois: %d\n", meu_livro.ano_publicacao);
    // printf("Autor depois: %s\n", meu_livro.autor); // Se alterado na função

    return 0;
}
```

Atribuição entre Estruturas

- É possível atribuir o conteúdo de uma variável de estrutura diretamente a outra, desde que ambas sejam do mesmo tipo de estrutura. Isso realiza uma cópia membro a membro.
- Exemplo (Adaptado de Ascencio e Campos, p. 215):

```
struct Coordenada pontoA;
struct Coordenada pontoB;

pontoA.x = 5;
pontoA.y = 15;
strcpy(pontoA.rotulo, "Origem");

pontoB = pontoA; // Atribuição direta! Todos os membros são copiados.
```

Referências

- ASCENCIO, Ana Fernanda Gomes; DE CAMPOS, Edilene Aparecida Veneruchi. **Fundamentos da programação de computadores**. Pearson Educación, 2008.
- BACKES, André. ***Linguagem C: completa e descomplicada***. 2. ed., 4. imp. Rio de Janeiro: LTC, 2023.
- CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. Rio de Janeiro: Elsevier, c2012.
- DAMAS, Luís. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.
- EVARISTO, Jaime. **Aprendendo a programar: programando em linguagem C**. Rio de Janeiro: Book Express, 2001. 205 p.
- MEDINA, Marco. **Algoritmos e programação: teoria e prática**. São Paulo: Novatec, 2006.
- MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C**. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, Marcos. **Estrutura de Dados com Algoritmos e C**. Rio de Janeiro: Brasport, 2008.