Linguagem C

Estruturas

MsC. Douglas Santiago Kridi

Programação I - 2018.2

Bacharelado em Ciência da Computação

Universidade Estadual do Piauí

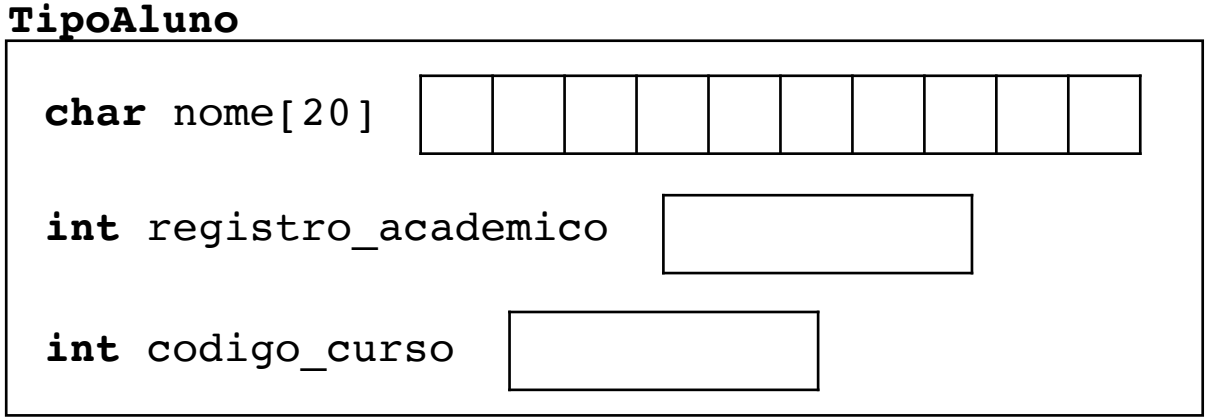
douglaskridi@gmail.com

Introdução

- Estruturas são um tipo de dados definido pelo programador, capaz de armazenar, *sob um mesmo nome de variável*, diversos dados inter-relacionados e possivelmente de *tipos diferentes*.

Introdução

- Uma *estrutura*, ou *registro*, é formada por uma coleção de uma ou mais variáveis declaradas juntas sob um único nome e manipuladas simultaneamente nas operações.
- Para diferenciar das variáveis convencionais, as variáveis que formam uma estrutura são chamadas de *atributos* da estrutura.
  - Note que a estrutura é um conceito diferente do vetor, que é uma coleção de valores do mesmo tipo e acessíveis por meio de um índice.



Sintaxe básica

- A forma geral da declaração de estruturas:

```
struct {
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

```
struct {
    ...
    tipo vetor[tamanho];
    tipo matriz[linhas][colunas];
    ...
} variavel;
```

- Exemplo: Aluno e Funcionário

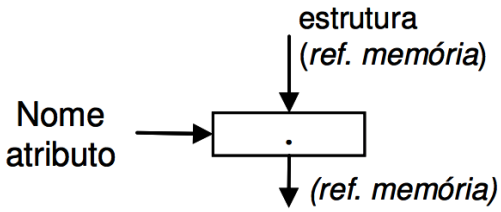
```
struct {
    char nome[50];
    int registro_academico;
    int codigo_curso;
} aluno;
```

```
struct {
    char nome[50];
    float salario;
    int codigo_cargo;
} funcionario;
```

Independentes, pois são de estruturas diferentes!

Acesso ao conteúdo

- Para modificar ou ler um valor de um atributo, escrevemos o nome da variável e o nome do atributo, separados por um *ponto*.
- O *operador de seleção de atributo*, na linguagem C é representado pelo ponto ‘.’.



- Exemplo: Para modificar o atributo *registro_academico* da variável aluno:

```
strcpy(aluno.nome, “Fulano”);
aluno.registro_academico = 991122;
aluno.codigo_curso = 3;
```

Acesso ao conteúdo

- Exemplo: Declaração e inicialização

```
struct {
    char nome[50];
    int registro_academico;
    int codigo_curso;
} aluno1, aluno2, aluno3;
```

```
struct {
    char nome[50];
    int registro_academico;
    int codigo_curso;
} aluno2 = {"Fulano", 991122, 3};
```

- Exemplo: Cópia

```
aluno1 = aluno2;
```

Acesso ao conteúdo

- Exemplo:

```
struct {
    char nome[50];
    float salario;
    int codigo_cargo;
} funcionario;
```

Acesso:
```
funcionario.salario
```

Atribuição:
```
funcionario.salario = 1000.0;
```

Impressão:
```
printf("A variável: %s", funcionario.nome);
```

Leitura:
```
scanf("%f", &funcionario.salario);
```

Declaração de tipo

- É possível dar um nome (um identificador) para uma estrutura.
- Desta forma, o compilador cria um *novo tipo de dados* baseado na definição da estrutura.
- Em outra parte do programa, este tipo poderá ser utilizado para declarar variáveis com esta estrutura, sem necessidade de definir novamente todos os seus atributos.
- Garante reuso da estrutura.

Declaração de tipo

- O nome da estrutura é escrito após a palavra struct.

```
struct nome_estrutura{
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
} variavel;
```

- Observe que o nome da estrutura é escrito antes das chaves { }, enquanto que o nome da variável é escrito depois das chaves.

- Depois basta utilizar a forma abreviada com apenas o nome da estrutura:

```
struct nome_estrutura variavel;
```

- A palavra chave *struct* *continua obrigatória* para deixar explícito tratar-se de uma declaração de variável tipo estrutura.

Declaração de tipo

- É possível definir estruturas nomeadas, sem sequer declarar variáveis.

```
struct nome_estrutura{
    tipo1 atributo1;
    tipo2 atributo2;
    tipo3 atributo3;
    ...
};
```

- Neste caso, o compilador apenas lembra a definição do novo tipo de dados formado pela estrutura.

Declarações aninhadas

- Exemplo:
  - Estruturas aninhadas:

```
struct {
    char nome[50];
    int codigo_curso;
    int dia_matricula;
    int mes_matricula;
    int ano_matricula;
} aluno;
```

```
struct {
    char nome[50];
    int codigo_curso;
    struct {
        int dia,mes,ano;
    } data_matricula;
} aluno;
```

  - Acessando:

```
aluno.data_matricula.dia = 02;
aluno.data_matricula.mes = 03;
aluno.data_matricula.ano = 2017;
```

Declarações aninhadas

- Exemplo:
  - Estruturas independentes:

```
struct TipoData{
        int dia,mes,ano;
};
```

```
struct TipoAluno{
    char nome[50];
    int codigo_curso;
    struct TipoData data_matricula;
} aluno;
```

  - Acessando:

```
aluno.data_matricula.dia = 02;
aluno.data_matricula.mes = 03;
aluno.data_matricula.ano = 2017;
```

Vetores de etsruturas

- Como uma estrutura pode ser um tipo de dado, é possível construir um vetor de estruturas (de um mesmo tipo).
- Exemplo:

```
struct TipoAluno{
    char nome[50];
    int codigo_curso;
    struct TipoData data_matricula;
};

struct TipoAluno turma[50];
```

  - O exemplo define *turma* como um vetor de estruturas do tipo Aluno.
  - Cada elemento desse vetor (ex. *turma[i]*) tem os campos definidos da estrutura que podem ser usados individualmente (ex. *turma[i].curso*).

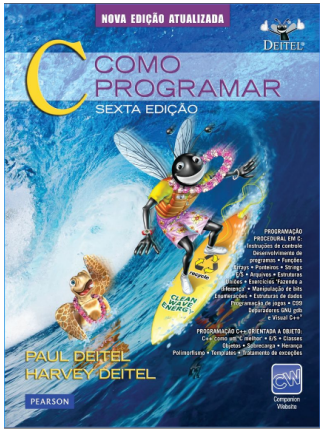
Pratique:

1. Crie as seguintes Estruturas (como tipos)

Pessoas	Veículos	Estoque
– nome	– marca	– código
– endereço	– modelo	– descrição
– telefone.	– cor	– valor unitário
	– placa	– quantidade

2. Acesse cada atributo de cada Struct, leia e imprima valores para cada um.
3. Em seguida, crie um vetor para cada um dos tipos recém criado e faça leituras e impressões a partir destes vetores.

Bibliografia Básica



C: COMO PROGRAMAR

DEITEL, Harvey M.; DEITEL, Paul J. Editora Pearson - 6ª ed. 2011

Fundamentos da Programação de Computadores

Ascencio, Ana F. G., Campos, Edilene A. V. de, - Editora Pearson 2012

Lógica de Programação e Estrutura de Dados

Puga, Sandra. Risseti, Gerson. – Ed. Pearson - 2016

Lógica de Programação Algorítmica (Apostila)

Guedes, Sergio. - Editora Pearson/Ser - 2014