

# SI100

## Algoritmos e Programação de Computadores I

1º Semestre - 2018

### Tópico 8 – Registros

Unidade 12

Prof. Guilherme Palermo Coelho

[guilherme@ft.unicamp.br](mailto:guilherme@ft.unicamp.br)

# Roteiro

- Registros;
- Exercícios;
- Referências.

# REGISTROS OU ESTRUTURAS

# Introdução

- As variáveis (e vetores/matrizes) vistas até agora permitem armazenar dados de um único tipo primitivo:
  - Inteiros, ponto flutuante, caracteres...
- No entanto, suponha que seja necessário armazenar em um programa os seguintes dados de 100 pacientes:
  - Pressão sanguínea, peso, altura, nível de colesterol e se é fumante ou não (caractere);
- Como poderíamos guardar estes dados no programa?
  - Uma matriz 100x5? → Não, temos dados de tipos diferentes!
  - 5 vetores de 100 posições? → Possível solução (difícil gerenciar).
  - Como colocar todos os dados de um paciente em um mesmo lugar? → registros (ou *estruturas*).

# Registros ou Estruturas

- **Estruturas** são tipos de variáveis que agrupam dados *geralmente de tipos diferentes*;
- São diferentes de matrizes, já que estas agrupam dados similares:
  - Itens de dados em uma matriz: **elementos**;
  - Itens de dados em uma estrutura: **membros**;
- É possível criar vetores/matrizes de estruturas.
- Estruturas são criadas com o comando **struct** da linguagem C.

# Registros ou Estruturas

- Como declarar uma estrutura:

```
struct nome_da_estrutura {  
    tipo_1 nome_1;  
    tipo_2 nome_2;  
    tipo_3 nome_3;  
    ...  
    tipo_n nome_n;  
} var_1, var_2;
```

Cabeçalho da declaração, com o nome da estrutura.

Declaração dos tipos e nomes dos **membros** da estrutura

Declaração do nome de duas variáveis que serão do tipo desta estrutura (**opcional**).

# Registros ou Estruturas

- Exemplo de estrutura:

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

```
int main() {
```

```
    struct pessoa {
```

```
        char nome[15];  
        int idade;
```

```
    } cliente1;
```

```
    struct pessoa cliente2;
```

```
    ...
```

```
    return 0;
```

```
}
```

Membros da estrutura

Duas variáveis

Note que tanto “cliente1” quanto “cliente2”  
são do tipo “struct pessoa”.

# Estruturas: acesso aos membros

- Para acessar os membros de uma estrutura, digita-se o **nome da variável** seguido de um “.” e do **nome do membro**:

```
#include <stdio.h>

int main() {
    struct pessoa {
        char nome[15];
        int idade;
    } c1;

    c1.idade = 22;
    c1.nome[0] = 'A';
    c1.nome[1] = '\\0';

    printf("%s: %d anos\\n", c1.nome, c1.idade);
    return 0;
}
```

A: 22 anos



# Estruturas: inicialização

- É possível inicializar os valores de cada campo (membro) de uma estrutura de forma similar à utilizada em vetores:
  - Valores de cada membro entre “{” “}” e separados por vírgula.

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

```
int main() {
```

```
    struct pessoa {  
        char nome[15];  
        int idade;  
    };
```

```
    struct pessoa c1 = {"Ana", 27},  
    printf("%s: %d anos\n", c1.nome, c1.idade);  
    return 0;
```

```
}
```

Ana: 27 anos

Note os tipos diferentes!

# Estruturas: cópia

- Nas versões mais modernas da linguagem C, é possível copiar uma estrutura armazenada em uma variável através de uma atribuição simples:
  - Ao fazer esta atribuição, **todos os membros são copiados**.

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

```
int main() {
```

```
    struct pessoa {
```

```
        char nome[15];
```

```
        int idade;
```

```
    } c2;
```

```
    struct pessoa c1 = {"Ana", 27};
```

```
    c2 = c1;
```

```
    printf("%s: %d anos\n", c2.nome, c2.idade);
```

```
    return 0;
```

```
}
```

Ana: 27 anos

# Estruturas: cópia

- Uma alternativa seria copiar os campos individualmente:

```
#include <stdio.h>
#include <string.h>
int main() {
    struct pessoa {
        char nome[15];
        int idade;
    } c2;
    struct pessoa c1 = {"Ana", 27};

    strcpy(c2.nome, c1.nome);
    c2.idade = c1.idade;
    printf("%s: %d anos\n", c2.nome, c2.idade);
    return 0;
}
```

Ana: 27 anos

**Atenção:** note que o campo “nome” é uma *string* (vetor) e C não permite que vetores sejam copiados com atribuições diretas. Por isso a chamada a **strcpy()**.

# Estruturas: definição de tipo

- Registros são ferramentas poderosas em C. No entanto, a declaração de novas variáveis para uma estrutura exige que o nome da estrutura seja precedido da palavra **struct** em cada declaração:
  - Deixa a declaração de variáveis de registros mais complexa;
- Isto pode ser resolvido através da **definição** de um novo **tipo de dados** que será associado ao registro em questão;
  - Usa-se a palavra reservada **typedef**.

```
typedef struct nome_da_estrutura{  
    tipo_1 nome_1;  
    ...  
    tipo_n nome_n;  
} nome_do_novo_tipo;
```

Note que, quando combinado com **typedef**, o identificador colocado após o “}” corresponde ao **nome do novo tipo** criado.

# Estruturas: definição de tipo

- Exemplo:

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

```
int main() {
```

```
    typedef struct pessoa {  
        char nome[15];  
        int idade;  
    } Pessoa;
```

```
    Pessoa c1 = {"Ana", 27};
```

```
    Pessoa c2 = c1;
```

```
    printf("%s: %d anos\n", c2.nome, c2.idade);  
    return 0;
```

```
}
```

Ana: 27 anos

# Estruturas: vetores e matrizes

- De maneira análoga aos tipos primitivos de dados, a linguagem C também permite a criação de **vetores** e **matrizes** de estruturas:

```
#include <stdio.h>

int main() {
    struct livro{
        char titulo[15];
        char autor[15];
        double preco;
    };
    struct livro bib[50];

    ...
}
```

Vetor de 50 posições. Cada posição contém um elemento **struct livro**.

# Estruturas: vetores e matrizes

```
#include <stdio.h>
#include <string.h>
int main() {
    struct livro{
        char titulo[15];
        char autor[15];
        double preco;
    };
    struct livro bib[50];

    strcpy(bib[0].titulo, "Iracema");
    strcpy(bib[0].autor, "Jose de Alencar");
    bib[0].preco = 10.0;

    bib[1] = bib[0];

    printf("%s, %c\n", bib[1].titulo, bib[1].autor[2]);
    return 0;
}
```

Iracema, s

# Estruturas aninhadas

- Por fim, também é possível criar **estruturas aninhadas** em C:
  - Estruturas que contêm membros que são outras estruturas.

```
#include <stdio.h>
#include <string.h>
int main() {
    struct tipo_endereco{
        int numero;
        int CEP;
        char rua[100];
        char cidade[100];
    };
    typedef struct pessoa {
        char nome[15];
        int idade;
        struct tipo_endereco endereco;
    } Pessoa;
    // continua...
```



# Estruturas aninhadas

```
//continuação:
```

```
    struct tipo_endereco endereco = {310, 13000, "Av.  
Brasil", "Campinas-SP"};  
    Pessoa p;  
  
    strcpy(p.nome, "Ana Flores");  
    p.idade = 25;  
    p.endereco = endereco;  
  
    p.endereco.numero = 320;  
  
    printf("%s, %d anos\n%s, %d, %s\n", p.nome, p.idade,  
p.endereco.rua, p.endereco.numero, p.endereco.cidade);  
  
    return 0;  
}
```

```
Ana Flores, 25 anos  
Av. Brasil, 320, Campinas-SP
```

# EXERCÍCIOS

# Exercícios

1. Escreva um programa que defina uma estrutura chamada “Ponto”, que deverá conter as coordenadas  $x$  e  $y$  de quaisquer pontos no espaço bidimensional, leia as coordenadas de dois pontos quaisquer do teclado, calcule a distância euclidiana entre eles e exiba o resultado na tela.
2. Escreva um programa que, a partir do que foi desenvolvido no Exercício 1, leia dois pontos e indique se eles são iguais. Dado que quando trabalhamos com pontos flutuantes não faz muito sentido buscarmos igualdade exata, considere que dois pontos são iguais quando a distância euclidiana entre eles for inferior a 0,0000001.
3. Defina um tipo “Retangulo” para armazenar as coordenadas dos pontos inferior esquerdo e superior direito de um retângulo contido no primeiro quadrante do plano cartesiano e cujos lados são paralelos aos eixos (utilize a estrutura “Ponto” do Ex. 1). Faça um programa que leia tais coordenadas e calcule a **área** do retângulo.

# Exercícios

4. Escreva um programa que, utilizando a estrutura definida no Ex. 3, leia as coordenadas de dois retângulos e indique se o segundo retângulo está **totalmente contido** no primeiro retângulo ou não.
5. Escreva um programa que contenha uma estrutura chamada *corpo*, que possua os campos **altura** (tipo *float*), **peso** (tipo *float*) e **nome** (tipo *string*). Este programa deve ler os dados de um usuário, armazená-los em uma variável da estrutura *corpo* e, em seguida, acessar estes valores armazenados, calcular o IMC (índice de massa corporal, dado pela divisão entre *peso* e o quadrado da *altura*) e exibir o resultado na tela.

# Exercícios

6. Escreva um programa que:

- Contenha uma estrutura com três membros do tipo **int**, chamados “hora”, “minuto” e “segundo”. Defina um tipo associado a esta estrutura chamado “Tempo”.
- Possua outra estrutura que armazene dados de controle em um estacionamento. Esta estrutura deve armazenar a **placa** do carro, sua **marca** e os instantes de **entrada** e de **saída** do veículo. *Entrada* e *saida* devem ser do tipo “Tempo” previamente definido.
- Leia do teclado e armazene os dados de estadia de **cinco** veículos em um vetor.
- Calcule o tempo total de permanência de cada veículo no estacionamento e exiba os resultados na tela (em conjunto com a marca e a placa do veículo).

# Exercícios

- **Observação:**

- Os seguintes exercícios devem ser entregues via SuSy.
  - Exercício 2;
  - Exercício 4;
  - Exercício 5.
- Veja os enunciados atualizados no site do sistema:
  - <https://susy.ic.unicamp.br:9999/si100a> (Turma A);
  - <https://susy.ic.unicamp.br:9999/si100b> (Turma B).

# REFERÊNCIAS

# Referências

- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo – Módulo 1*. 2a Edição, Pearson Makron Books, 2005.
- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo – Módulo 2*. 2a Edição, Pearson Makron Books, 2005.
- ASCENCIO, A. F. G. & DE CAMPOS, E. A. V., *Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++*. Pearson Prentice Hall, 2003.