Algoritmos e Estruturas de Dados II

ED2

Módulo 2 – *Structs* (Estruturas ou Registros)

Profa. Msc. **Cilene Aparecida Mainente**

1- Introdução

1. Defina um vetor para armazenar 100 nomes.

char nomes [100];

Armazenará 100 nomes ou um nome com até 100 caracteres ??

2. Defina um vetor para armazenar 100 notas.

```
float notas [100];
```

3. Defina um vetor para armazenar 100 nomes e 100 notas.

É possível armazenar tipos de dados diferentes em um mesmo vetor ?

Se fosse matriz, daria ?

2- Conceito

Structs ou Estruturas ou Registros

As *structs* podem armazenar diferentes tipos de dados em uma mesma constituição, ao contrário dos vetores e matrizes, que trabalham somente com um mesmo tipo.

Aqui estamos definindo uma estrutura chamada Nome, que contém diversos tipos de dados (a, b etc.) associados a variáveis (1, 2 etc).

```
struct Nome
    {
        tipo a variável1;
        tipo b variável2;
        ..........
        ..........
    };
```

Structs ou Estruturas ou Registros

Criando seus próprios tipos de dados.

Cada um dos componentes é chamado de **campo** ou **membro**.

Exemplo

A implementação de uma estrutura desse tipo **para armazenar os dados de uma pessoa, tais como RG, nome e salário,** seria algo assim:

```
struct Pessoa
    {
        int rg;
        char nome[30];
        float salario;
    };
```

3- Carga Inicial da *Struct*

1- Definindo um novo tipo de dado

A declaração da estrutura cria um novo tipo de dado.

Repare que o tipo da variável é a nova estrutura criada:

```
struct Pessoa
    {
        int rg;
        char nome[30];
        float salario;
    };
```

```
struct Aluno
    {
        int RA;
        char nome[50];
    } Paulo;
```

Pode-se também declarar a variável de outra forma – após a estrutura.

Ao lado, a variável **Paulo** é definida como sendo do tipo **Struct Aluno**.

Carga Inicial da Struct

A) Inicialmente, vamos definir uma variável do tipo da *Struct*:

struct Pessoa Pedro; (ou seja, *Pedro é uma Pessoa*)

B) Podemos então atribuir valores, como esses:

- **Pedro**.rg = 2324;
- **Pedro**.salario = 600.00;
- strcpy(**Pedro**.nome, “Pedro Silva”);

// usando a função strcpy( )
// por ser uma string

```
struct Pessoa
    {
        int rg;
        char nome[30];
        float salario;
    };
```

4- Acessando Membros da *Struct*

Acessando Membros da Struct

// Criar uma variavel do tipo Pessoa
Struct Pessoa Pedro;

Agora acessaremos os valores armazenados na estrutura:

printf (“Salario = %f \n”, **Pedro**.salario);
printf (“Nome = %s \n”, **Pedro**.nome);

```
struct Pessoa
    {
        int rg;
        char nome[30];
        float salario;
    };
```

Acessando Membros da *Struct*

```
struct Aluno {
  int RA;
  char nome[50];
  float nota;
};
int main() {
    struct Aluno Paulo;
    printf("RA: \n");   scanf("%d", &Paulo.RA);
    printf("Nota: \n"); scanf("%f", &Paulo.nota);
    printf("Nome: \n"); scanf("%s", Paulo.nome);
                        // ou gets(Paulo.nome);
    printf("Exibicao dos dados do aluno: \n");
    printf("RA: %d \n", Paulo.RA);
    printf("Nota: %.2f \n", Paulo.nota);
    printf("Nome: %s \n", Paulo.nome);
    // ou puts(Paulo.nome);
}
```

Leitura: ... **&Paulo.RA**

Impressão: **Paulo.RA**

Voltando ao Problema Original

Armazenar em uma mesma variável o nome e a nota de **01** aluno.

A) **Criar um novo tipo de dado**

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

B) **Criar uma variável do tipo Aluno**

```
struct Aluno alunoUSCS;
```

C) **Podemos alimentar diretamente os dados desse aluno ou ler**

```
alunoUSCS.nota = 10.0;
```

Exercício 1

Implemente a estrutura Aluno apresentada abaixo, em um programa que leia os dados de dois alunos e os armazene em duas variáveis <aluno1> e <aluno2>.

Ao final, apresentar em tela os dados dos dois alunos.

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

Exercício 1 - Resolução

Implemente a estrutura Aluno apresentada abaixo, em um programa que leia os dados de dois alunos e os armazene em duas variáveis <aluno1> e <aluno2>.

Ao final, apresentar em tela os dados dos dois alunos.

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

```
int main () {
    struct Aluno aluno1, aluno2;

    printf("Digite nome:");   scanf ("%s",  aluno1.nome);
    printf("Digite nota:");   scanf ("%f", &aluno1.nota);

    printf("Digite nome:");   scanf ("%s",  aluno2.nome);
    printf("Digite nota:");   scanf ("%f", &aluno2.nota);

    printf ("%s nota=%.2f" , aluno1.nome,aluno1.nota);
    printf ("%s nota=%.2f" , aluno2.nome,aluno2.nota);
}
```

5- Vetor de *Structs*

Vetor de Structs

Uma possibilidade interessante é expressar as estruturas
por meio de vetores, tornando possível um encadeamento de dados.

Exemplo:
Armazenar o **título** e **ano** de gravação de 05 **CD**'s.

- O título pode ter no máximo 30 caracteres e o ano é inteiro.

Vetor de Structs

Estrutura para Armazenar os dados de um CD:

```
struct CD {
    char titulo[30];
     int ano;
};
```

Vetor de Structs

Ex. Para armazenar 01 CD, usamos: **struct CD cantigasNinar;**

Estrutura para Armazenar os dados de **5** CDs:

```
struct CD colecao[5];
```

Vetor de Structs

```
struct CD {
    char titulo[30];
     int ano;
};
```

```
struct CD colecao[5];
```

Atualizando os dados dos CD´s

```
colecao[0].ano = 1967;
printf("%d \n", colecao[0].ano);
strcpy(colecao[0].titulo, "The Beatles");
printf("%s \n", colecao[0].titulo);
```

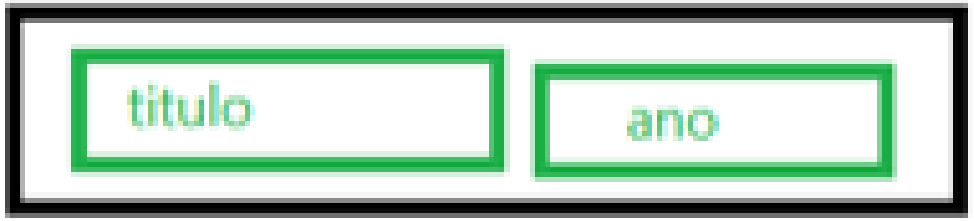
```
colecao[1].ano = 1984;
printf("%d \n", colecao[1].ano);
strcpy(colecao[1].titulo, "The Smiths");
printf("%s \n", colecao[1].titulo);
```

```
struct CD {
    char titulo[30];
    int ano;
};
```

```
struct CD colecao[5];
```

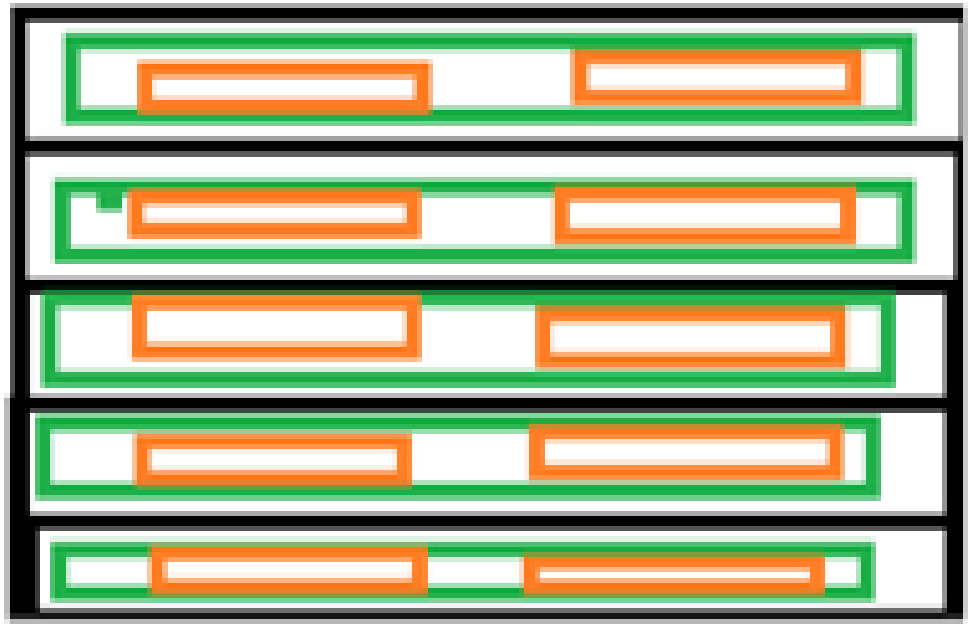
Atualizando os dados dos CD´s

struct CD



struct CD colecao [5];

colecao



colecao[0].ano = 2020;

colecao[4].ano = 1970;

6- Definindo um Tipo

Definição de Tipos (typedef)

A palavra reservada **typedef** tem o propósito de associar nomes alternativos a tipos de dados já existentes.

Ou seja, cria um **apelido** para o tipo.

Pode ser utilizado para qualquer tipo de dado ou estrutura.

Forma Geral: **typedef <tipoDeDado> <apelidoDoTipoDeDado>;**

Exemplo 01 – Apelido para o tipo int

No exemplo abaixo, cria-se o apelido "KmPorHora" para o tipo int.

```
typedef int kmPorHora;          // definindo novo tipo int
```

Ao definir uma variável inteira, pode-se utilizar o apelido:

```
kmPorHora velocidade_atual; // usando o novo tipo
```

Typedef com Struct

Typedef pode ser usado para simplificar a declaração de estruturas já criadas, como esta:

```
struct MinhaStruct{
    int data1;
    char data2;
};
```

Uma nova declaração simplificadora poderia ser:

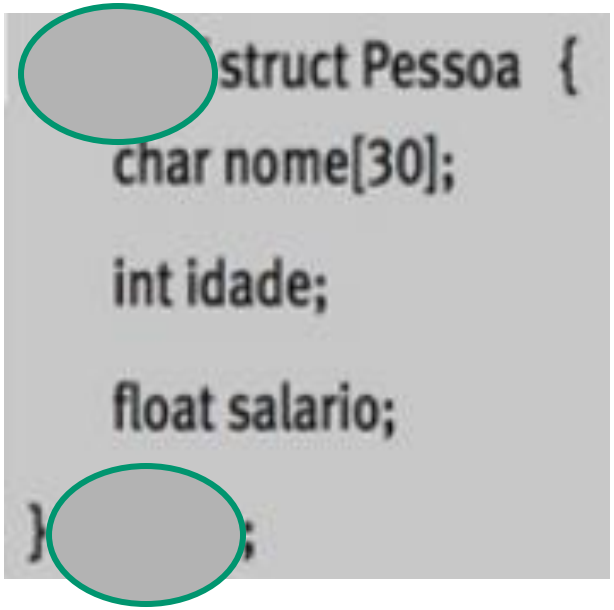
```
typedef struct MinhaStruct novo_tipo;
```

ou

```
typedef struct MinhaStruct {
    int dado1;
    char dado2;
} novo_tipo;
```

ou

Exemplo 02 – Sem o typedef



Dessa forma, para criar uma pessoa chamada Joao, usamos:

struct Pessoa joao;

Exemplo 02 – Apelido para uma Struct

```
typedef struct Pessoa  {
    char nome[30];
    int idade;
    float salario;
} PESSOA;
```

Dessa forma, para criar uma pessoa chamada Joao, usamos:

PESSOA joao; // Onde PESSOA é o apelido da estrutura

7- Passagem de *Structs* para Funções

Passando uma *Struct* para uma Função

Sempre interessante poder passar uma estrutura de dados
como argumento de uma função,
dadas as já comentadas vantagens
que uma função apresenta em programação.

Passando uma *Struct* para uma Função - Exemplo

```
typedef struct Pessoa  {
    char nome[30];
    int idade;
    float salario;
} PESSOA;
```

Passando uma Struct para uma Função

```
main() {
    struct Pessoa p = {“Paulo”, 35, 2000};
    mostrar(p);
}
```

Poderia ser escrito da seguinte forma:

```
main () {
   PESSOA p;
   strcpy(p.nome, "Paulo");
   p.idade = 35;
   p.salario = 2000;

   // PESSOA p={"Paulo",35,2000};
}
```

Passando uma *Struct* para uma Função

```
void mostrar (struct Pessoa x) {
    printf("Nome: %s \n", x.nome);
    printf("Idade: %d \n", x.idade);
    printf("Salario: %.2f \n", x.salario);
}
```