Universidade Federal de Ouro Preto
Campus João Monlevade

CSI103 – Algoritmos e Estruturas de Dados I

Revisão de Conteúdo
Registros

Prof. Mateus Ferreira Satler

Índice

1. Registros
2. Exercícios
3. Referências Bibliográficas

1. Registros

- Um registro (**`struct`**) é a forma de agrupar variáveis em C, sejam elas de mesmo tipo ou de tipos diferentes.

- As variáveis unidas em um registro se relacionam de forma a criar um contexto maior

- Exemplos de uso de registros:
  - **Registro de Alunos**: armazena nome, matrícula, médias, faltas, etc.
  - **Registro de Pacientes**: armazena nome, endereço, convênio, histórico hospitalar

1. Registros

- Um registro é declarado com a palavra-reservada **struct**

```
struct nome_tipo_registro {
   tipo_1 variavel_1;
   tipo_2 variavel_2;
   ...
   tipo_n variavel_n;
};
```

- Devemos declarar variáveis deste novo tipo assim:

```
struct nome_tipo_registro variavel_registro;
```

1. Registros

- Um registro define um novo tipo de dados com nome **`nome_tipo_registro`** que possui os campos **`variavel_i`**
- O campo **`variavel_i`** é uma variável do tipo **`tipo_i`** e será acessível a partir de uma variável do novo tipo **`nome_tipo_registro`**.
- **`variavel_registro`** é uma variável do tipo **`nome_tipo_registro`** e possui, internamente, os campos **`variavel_i`** do tipo declarado.
- A declaração de um registro pode ser feita dentro de uma sub-rotina ou fora dela.

1. Registros

```
struct regAluno {
    char nome[50];
    int idade;
    char sexo;
    int turma;
};

int main ( ) {
    struct regAluno aluno1, aluno2;
}
```

1. Registros

- Os campos de um registro podem ser acessados individualmente a partir de variáveis do tipo do registro da seguinte forma:

```
variavel_registro.variavel_i
```

- Cada campo **`variavel_registro.variavel_i`** se comporta como uma variável do tipo do campo, ou seja, **`tipo_i`**

- Isto significa que todas as operações válidas para variáveis do tipo **`tipo_i`** são válidas para o campo acessado por **`variavel_registro.variavel_i`**

1. Registros

- Na inicialização de registros, os campos obedecem à ordem de declaração:

```
struct contaBancaria {
    int numero;
    char idCorrentista[15];
    float saldo;
};

struct contaBancaria conta = { 1, "MG12345", 100.0 };
```

1. Registros

- Pode-se copiar o conteúdo de uma estrutura para outra utilizando uma atribuição simples.

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

int main () {
   struct ponto p = {1,2}, q;
   q.x = 3; q.y = 4;
   q = p;
   q.x = 5;
}
```

	End.	Cont.	
p	0x200		x
	0x300		y
q	0x400		x
	0x500		y
	0x600		
	0x700		
	0x800		

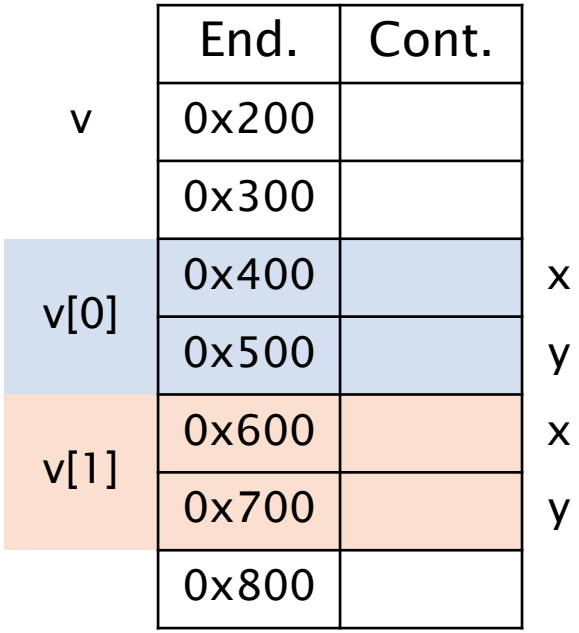
1. Registros

- Pode-se criar vetores de registros da mesma maneira que se criam vetores de tipos primitivos (**int**, **float**, **char**, **double**).

- É necessário definir a estrutura antes de declarar o vetor.

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

int main () {
   struct ponto v[2];
   v[0].x = 3; v[0].y = 4;
   v[1].x = 5; v[1].y = 6;
}
```



1. Registros

- Pode-se passar um registro como parâmetro de subrotina.

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

void imprimePonto (struct ponto p) {
   printf("Coordenadas (%d, %d)", p.x, p.y);
}

int main () {
   struct ponto p = {1,2};
   imprimePonto (p);
}
```

1.1. Sinônimos de tipos: typedef

- Pode-se associar uma definição de **struct** a um novo tipo de dados, evitando a repetição da palavra **struct** pelo código:

```
typedef int inteiros;
typedef char caracteres;

typedef struct contabancaria {
    int numero;
    char idCorrentista[15];
    float saldo;
} CB;

int main () {
  CB conta1, conta2; }
```

1.2. Alocação Dinâmica de Registros

- O operador **->** facilita o acesso aos registros, permitindo acessar diretamente o conteúdo de um campo do registro

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

Ponto* ponto_ptr;
ponto_ptr = (Ponto*) malloc (sizeof(Ponto));
ponto_ptr->x = 3;
ponto_ptr->y = 4;
free(ponto_ptr);
```

2. Exercícios

1. Escreva um algoritmo completo na linguagem C seguindo as instruções abaixo. Leia atentamente todos os itens antes de iniciar seu desenvolvimento:

   a) Crie uma estrutura chamada livro. Cada livro terá os seguintes dados: título (150 caracteres), autor (150 caracteres), preços (vetor de números reais com 6 posições). O vetor "preços" representa o preço do livro nos últimos 6 meses.

   b) Crie um procedimento que peça ao usuário para digitar todos os dados de um conjunto de livros. Os parâmetros do procedimento são: um vetor de livros e seu tamanho.

   c) Crie um procedimento para imprimir o preço médio de cada livro do vetor de livros. O preço médio será calculado pela média aritmética de todos os preços de cada livro. Os parâmetros deste procedimento são: um vetor de livros e seu tamanho.

   d) Faça uma função principal (main) para criar dinamicamente um vetor de 5 livros usando a estrutura do item (a) e, posteriormente, “chame” os procedimentos dos itens (b) e (c). Imprima também as informações do livro mais caro e do livro mais barato.

2. Exercícios

2. Considere as seguintes estruturas:

```
typedef struct aluno_reg {
{
    char nome[50];
    int matricula;
}aluno;
```

```
typedef struct disciplina_reg

    char nome[50];
    int código;
    aluno * matriculados;
    int num_matriculados;
} disciplina;
```

Escreva um programa na linguagem C que use **alocação dinâmica** para criar 2 (duas) disciplinas. Em seguida, peça ao usuário para digitar as informações (nome e matrícula) de vários alunos para cada disciplina. Seu programa deve perguntar se o usuário quer adicionar mais alunos, e se a resposta for positiva, deve alocar espaço **dinamicamente** para inclusão desse aluno. Por último, o programa deve imprimir na tela as informações das duas disciplinas e dos alunos matriculados nelas.

3. Referências Bibliográficas

- Material de aula do Prof. Ricardo Anido, da UNICAMP: http://www.ic.unicamp.br/~ranido/mc102/

- Material de aula da Profa. Virgínia F. Mota: https://sites.google.com/site/virginiaferm/home/disciplinas

- DEITEL, P; DEITEL, H. C How to Program. 6a Ed. Pearson, 2010.