Lógica de programação_Aula 9

Caderno:	Aulas		
Criada em:	28/07/2020 15:17	**Atualizada ...**	28/07/2020 19:40
Autor:	cbrmesquita@gmail.com		

Parte teórica

Variáveis compostas heterogêneas

Principal : **Registro** (também chamadas estruturas)

Há um outro tipo de variável composta heterogênea chamada união, mas eu não cheguei a aprender o precisar a usar ela.

Vetores/Matrizes/Strings -> Variáveis compostas homogêneas (mesmo tipo de dado armazenado)

Já em um registro, teremos variáveis com tipos diferentes, mas que possuem algo em comum, como pertencer a algo.

Em c, o registro é chamado **STRUCT**

Exemplo: Criando uma estrutura para armazenar uma data.

```
struct {
   int dia;
   int mes;
   int ano;
} hoje;
```

para armazenar dados para esse registro "hoje", podemos:

```
hoje.dia = 28;
hoje.mes = 7;
hoje.ano = 2020;
```

Mas, no exemplo anterior, da forma como foi declarada, não poderíamos criar outra variáveis do tipo hoje.

Para solucionar esse problema, nós rotulamos as estruturas:

```
typedef struct data {
   int dia;
   int mes;
   int ano;
};
```

É como se criássemos um "tipo de dado" data, que pode receber um dia, mes e ano. Assim, podemos declarar variáveis que sejam desse "tipo":

```
struct data hoje;
```

struct data ontem, amanha;

(tem que usar o struct junto, se não o compilador não reconhece)

ou, para facilitar ainda mais, podemos nomea-la também:

```
typedef struct data {
   int dia;
   int mes;
   int ano;
} DATA;
```

Assim, podemos criar as estruturas da seguinte forma:

```
DATA hoje;
DATA amanha, ontem;
```

ou podemos fazer simplesmente:

```
typedef struct {
   int dia;
   int mes;
   int ano;
} DATA;
```

Mas esse ultimo caso não pode ser utilizado para caso de estruturas recursivas (que serão vistas mais para a frente).

Essa declaração pode ser feita dentro ou fora da main.

Atribuição de valores

```
DATA hoje = { 28, 7, 2020 };
```

Observação, caso um dos elementos for um vetor ou uma matriz, os dados irão ser armazenados no vetor/matriz até preenche-lo e assim passar para a próxima variavel.

Ex: Se o "mês" do exemplo anterior fosse um vetor:
int mes[1]

então, para preencher corretamente os dados teriamos:

```
DATA hoje = { 28, 0 , 7, 2020};
```

É possível também que uma estrutura(1) tenha como elemento outra estrutura(2), mas a estrutura(2) precisa ter sido previamente definida.

```
typedef struct {
   char nome[31];
   char fone[21];
   DATA nasc;
} PESSOA;
```

Para acessar um dado na struct usamos:

registro.variavelinterna

e se for um vetor:

registro.variavelinterna[posicao]

e se for um registro dentro do outro

registro1.registro2.variavelinterna2

para receber os dados é a mesma coisa.

Vetor de estruturas

Criar um vetor de estruturas é uma forma de nos possibilitar de criar diversos registros que tenham origens diferentes mas uma mesma categoria, mas ele é só uma das formas de fazer, exemplo:

Podemos cadastrar dados de diversos alunos, criar diferentes compromissos em uma agenda e assim por diante.

formas de declaração:
1_)

```
typedef struct data{
    int dia;
    int mes;
    int ano;
}DATA[10];
```

2_)

ou

```
typedef struct {
    int dia;
    int mes;
    int ano;
} DATA;
```

-na main-

```
DATA hoje[5];
```

Diferenças: na 1 ao usar um elemento, utilizaremos DATA[i].dia, já na 2 usamos hoje[i].dia

(um vetor de estruturas/registros pode ser chamado de tabela)

Parte prática

Ex1:

1. Escreva um trecho de código para fazer a criação dos novos tipos de dados conforme solicitado abaixo:
• Horário: composto de hora, minutos e segundos.
• Data: composto de dia, mês e ano.
• Compromisso: composto de uma data,horário e texto que descreve o compromisso.

Ex2:

3. Construa uma estrutura aluno com nome, número de matrícula e curso. Leia do usuário a informção de 5 alunos, armazene em vetor dessa estrutura e imprima os dados na tela.