Universidade Federal do Ceará

Centro de Ciências e Centro de Tecnologia

Departamento de Computação e Departamento de Engenharia Metalúrgica e de Materiais

Programação Computacional para Engenharia – 2013.2

Aula de 2013-11-25 (material complementar) – Estruturas (registros, struct's)

Conteúdo da aula:

1. Definição de estruturas.
2. Sintaxe e semântica de estruturas em C: definição, inicialização, acesso a membros (inclusive via ponteiros) e cópia.

Observações:

1. Como visto em sala, a inicialização de estruturas pode ser feita de duas formas: (1) na ordem em que os membros são listados na definição da estrutura e (2) pelo nome dos membros, em qualquer ordem:

```
struct Data { int dia, mes, ano; };
struct Data d1 = { 26, 11, 2013 };                    // Desde C90.
struct Data d2 = { .mes = 2, .dia = 9, .ano = 1983 }; // Desde C99.
struct Data d3 = { .ano = 1888 };                     // demais membros: 0.
struct Pessoa { char nome[100]; struct Data nasc; };
struct Pessoa p = { .nome = "Pablo" };                // nasc: 0, recursiv.
```

   Informação adicional: como ilustrado acima, é possível não inicializar todos os membros de uma estrutura: nesse caso, os membros omitidos são inicializados com zero. Caso um membro tenha, ele mesmo, tipo estrutura, a regra é aplicada recursivamente, e os membros dessa estrutura são inicializados com zero, etc.

   Fonte: Padrão C99, §6.7.8, itens 19, 21 e 10.

2. A regra acima de inicialização implícita se aplica também a vetores, bem como a forma de inicialização "designada" (independente da ordem):

```
char nome[10] = "Pablo"; // == { 'P', 'a', 'b', 'l', 'o', '\0', 0, 0, 0, 0 }
int v1[6] = { 0, 3, -1, 0 };        // == { 0, 3, -1, 0, 0, 0 }
int v2[6] = { [3] = -2, [1] = 3 }; // == { 0, 3, 0, -2, 0, 0 }
```

   Observação: no comentário da inicialização acima do vetor nome, \0 e 0 representam, na verdade, a mesma coisa – um caractere de valor zero –, e portanto nós poderíamos ter usado apenas \0, ou apenas 0; isso é explicado no item 5 abaixo.

3. O ponto-e-vírgula após a definição de um tipo estrutura tem um bom motivo: na verdade, toda tal definição pode também ser utilizada para declarar variáveis do tipo em questão:

```
struct ParOrd   { double x,y; } p;
struct Complexo { double r,i; } c = { 1, 2 };
```

   Assim, a diferença essencial entre "**int** i;" e "**struct** ParOrd { **double** x,y; } p;" é que este último comando também define um novo tipo.

4. Em sala, foi perguntado algo como se ".mes = 2" e ".mes = 02" são equivalentes. Para essa pergunta específica, a resposta é "sim", mas existe um detalhe técnico envolvido: em C, constantes inteiras iniciadas pelo dígito 0 indicam um número em base octal. Coincidentemente, 02 (octal) e 2 (decimal) representam o mesmo número, então não há problema, mas, por exemplo, 09 não é uma constante inteira válida em C, porque 9 não é um dígito octal. Constantes iniciadas por 0x ou 0X são de números representados na base hexadecimal.

   Fonte: Padrão C99, §6.4.4.1. (Mas veja a observação do item 5 sobre memorizar detalhes técnicos.)

5. Detalhes técnicos da linguagem C: o tipo **char** da linguagem C utiliza um único *byte*[1] para o armazenamento dos seus valores. O valor armazenado num *byte* pode ser interpretado como um número inteiro, e,

[1] Da Wikipédia: *The byte is a unit of digital information in computing and telecommunications that most commonly consists of eight bits. Historically, the byte was the number of bits used to encode a single character of text in a computer and for this reason it is the smallest addressable unit of memory in many computer architectures. The size of the byte has historically been hardware dependent and no definitive standards existed that mandated the size. The de facto standard of eight bits is a convenient power of two permitting the values 0 through 255 for one byte.*

de fato, as constantes de caractere da linguagem C representam números inteiros. É possível, inclusive, atribuir um valor inteiro (suficientemente pequeno) a uma variável do tipo **char**, e vice-versa:

```
char a = 98;        // a == 98 (decimal).
char b = '\142';    // 142 octal == 98 decimal, logo a == b.
char c = '\x62';    // 62 hexadecimal == 98 decimal, logo a == c.
char d = '\0';      // Zero octal. O mesmo que "char d = 0;"
char e = '0';       // O inteiro zero nao representa o simbolo 0, logo d != e.
char f = '1' + 2;   // f == '3'.
```

A notação '\ooo' é utilizada para denotar um valor inteiro de até 3 dígitos octais que caiba num *byte*. Em particular, '\0' denota o número zero em octal, e portanto a constante '\0', que nós conhecêramos como o delimitador de fim de *strings*, é simplesmente um outro nome para o inteiro zero – mas ela é utilizada mesmo assim, para reforçar a intenção da representação de um valor armazenável pelo tipo **char**.

Observe que '\0' e '0' representam números distintos: a primeira constante representa o número zero, enquanto a segunda representa o número inteiro que, no conjunto de caracteres do ambiente de execução do programa, é utilizado para representar o símbolo "0".

A notação '\xhh' é utilizada para denotar um inteiro (que caiba num *byte*) por meio de até 2 dígitos hexadecimais (0 ... 9, a ... f, A ... F).

Fonte: Padrão C99, §6.4.4.4.

Atenção: a especificação formal da linguagem C envolve muitos detalhes; o padrão C99 garante, por exemplo, que as constantes '0' a '9' assumem valores inteiros consecutivos (§5.2.1, item 3). Na prática, esses detalhes são mais frequentemente consultados quando necessário do que memorizados. Como qualquer outro programador, você pode consultar um texto de referência para ter acesso a essas informações, mas lembre que o foco da nossa disciplina não está nesses detalhes técnicos, mas sim na lógica da programação e na utilização da linguagem C para a escrita de programas reais.

Exercícios:

1. Escreva uma função "**int** bissexto (**int** a)" que retorna 1 se a é ano bissexto, e que retorna zero em caso contrário, utilizando as seguintes regras (fonte omitida por razões didáticas):
   (a) São bissextos todos os anos múltiplos de 400: 1600, 2000, 2400, 2800...
   (b) Também são bissextos todos os demais anos que são múltiplos de 4 mas não de 100: 1896, 1904, 1908, 1912, 1916...
   (c) Os demais anos não são bissextos.

   Dica: lembre que, em C, a % b denota o resto da divisão de a por b se ambos são inteiros e b não é zero.

2. Escreva uma função "Data proxima_data (Data d)" que recebe uma "**struct** Data {**int** dia, mes, ano;};" e retorna a data seguinte. A sua função deve levar em consideração os anos bissextos, utilizando, para isso, a função do exercício 1.

   Dica: uma função como a deste exercício envolve uma análise de casos; talvez você possa simplificar o seu código definindo um vetor ou matriz com informações relevantes, por exemplo sobre os meses do ano.

3. Escreva uma função "estatistica_salarios" como abaixo:

```
struct TFunc { char nome[100]; float salario; };
void estatistica_salarios (int n, struct TFunc vf[n],
                           float *min, float *max, float *med, float *dm);
```

   que recebe um vetor com os dados de $n \geq 1$ funcionários e que calcula: o menor salário, o maior salário, a média dos salários e o desvio médio dos salários. Os dados calculados devem ser atribuídos aos objetos apontados pelos ponteiros que a função recebe (exemplo: "*min = ...;"). O desvio médio de um multiconjunto $\{x_1, \ldots, x_n\}$ de média $M = \frac{1}{n}\sum_{i=1}^{n} x_i$ é dado por $\frac{1}{n}\sum_{i=1}^{n}|x_i - M|$.

4. Defina um tipo estrutura "DadosEstatisticos" que tenha como membros os 4 dados estatísticos da questão anterior (mínimo, máximo, média e desvio médio, todos **float**), e então escreva uma função "estatistica_salarios_2" que receba como argumento apenas um vetor de funcionários (e o tamanho deste), e que retorne os 4 dados estatísticos em questão por meio de uma estrutura. A função em questão não deve repetir o código da anterior, mas sim realizar uma chamada a esta última para a realização dos cálculos.