Sistemas Operacionais I

Laboratório 04

Revisão da Linguagem C
(Tipos Agregados Heterogêneos)

Agenda

- Estruturas (struct)
- Uniões (union)
- Enumeração (enum)
- Exercícios (Revisão)

Estruturas de dados

- Uma estrutura (struct) é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, agrupadas sob um único nome.
- Estruturas constituem um recurso importante para organizar os dados utilizados por um programa pois trata um grupo de valores como uma única variável.
- São chamadas de registros em outras linguagens de programação.

Estruturas de dados

- Estruturas (ou registros) são classificados como variáveis compostas heterogêneas, pois podem agrupar variáveis de tipos diferentes.
- Em contraposição, temos os vetores e matrizes, classificados como variáveis compostas homogêneas, pois somente agrupam variáveis do mesmo tipo.

Estruturas de dados

- Exemplo:

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

Estruturas de dados

- A palavra-chave **`struct`** informa ao compilador que um modelo de estrutura está sendo definido.
- "data" é uma etiqueta que dá nome à definição da estrutura.
- Uma definição de estrutura é um comando, por isso deve terminar em ponto-e-vírgula.

Estruturas de dados

- Os nomes declarados entre as chaves são os campos (ou membros) da estrutura.
- Os campos de uma mesma estrutura devem ter nomes diferentes.
- Porém, estruturas diferentes podem conter campos com o mesmo nome.

Estruturas de dados

- A definição de uma estrutura não reserva qualquer espaço na memória.
- Note que, no exemplo dado, nenhuma variável foi declarada de fato, apenas a forma dos dados foi definida.
- Essa definição, porém, cria um novo tipo de dados, que pode ser usado para declarar variáveis.

Declarando uma estrutura

- Duas maneiras de declarar a variável x do tipo data:

```
struct data
{
    int dia;
    int mes;
    int ano;
};
...
struct data x;
```

Dois comandos:
•Define estrutura como novo tipo
•Declara variável do novo tipo definido

ou

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

Um comando:
•Define estrutura e declara variável do novo tipo definido

Declarando uma estrutura

- Os campos de uma estrutura podem ser de qualquer tipo, inclusive uma estrutura previamente definida.

- Porém, o tipo dos campos não podem ser o do próprio tipo que está sendo definido.

Declarando uma estrutura

- A definição do formato de uma estrutura pode ser feita dentro da função principal (**`main`**) ou fora dela.

- Usualmente, declara-se fora da função principal, de modo que outras funções também possam "enxergar" a estrutura definida.

Estruturas de dados

- Forma geral de definição de uma estrutura:

```
struct <etiqueta> {
    <tipo> campo_1;
    <tipo> campo_2;
    ...
    <tipo> campo_n;
} <variáveis>;
```

Declarando uma estrutura utilizando **typedef**

- Novos tipos de dados podem ser definidos utilizando-se a palavra-chave **typedef**.

```
typedef struct nome_da_estrutura
{
    <tipo> campo_1;
    <tipo> campo_2;
    ...
    <tipo> campo_n;
} nome_do_tipo;
```

Declarando uma estrutura utilizando **typedef**

- Exemplo:

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

Declarando uma estrutura utilizando **typedef**

- O uso mais comum de **typedef** é com estruturas de dados, pois evita que a palavra-chave **struct** tenha de ser colocada toda vez que uma estrutura é declarada.

```
struct data dia_de_hoje;
tipoData dia_de_hoje;
```

Acessando os campos de uma estrutura

- Podemos acessar individualmente os campos de uma determinada estrutura como se fossem variáveis comuns.
- A sintaxe para acessar e manipular campos de estruturas é a seguinte:

```
<nome_da_variável>.<campo>
```

Leitura dos campos de uma estrutura

- A leitura dos campos de uma estrutura a partir do teclado deve ser feita campo a campo, como se fosse variáveis independentes.

```
printf ("Digite o nome do aluno: ");
scanf ("%s", &aluno.nome);

printf ("Digite a idade do aluno: ");
scanf ("%d", &aluno.idade);
```

Estruturas aninhadas

- Um campo de uma estrutura pode ser uma outra estrutura.
- Quando isso ocorre, temos uma estrutura aninhada.
- O padrão ANSI C especifica que as estruturas podem ser aninhadas até 15 níveis, mas a maioria dos compiladores permite mais.

Vetor de estruturas

- Usado quando precisamos de diversas cópias de uma estrutura.
- Por exemplo, cada cliente de uma locadora de vídeo constitui um elemento de um vetor, cujo tipo é uma estrutura de dados que define as características de cada cliente.

```
struct etiqueta variável[dimensão];
```

Union

- Uma **union** é uma posição de memória que é compartilhada por duas ou mais variáveis diferentes, geralmente de tipos diferentes, em momentos diferentes.
- Sua definição é semelhante à de estrutura:

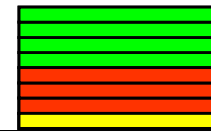
```
union <identificador> {
    <tipo> campo_1;
    <tipo> campo_2;
    ...
    <tipo> campo_n;
} <variáveis>;
```

Union
:: Exemplo

- O espaço ocupado por uma union na memória corresponde ao maior tamanho de variável que ela contém. Exemplo:

```
struct exemplo {
        int             a;
        char            b;
        double          c;
}
```

```
union exemplo {
        int             a;
        char            b;
        double          c;
}
```



Union
:: Exemplo

- Código para guardar um float e um inteiro em uma mesma posição de memória:

```
union ieee754 {
        float  num_real;
        int            num_hexa;
};
```

Palavra chave **enum**

- **enum** é a abreviação de ENUMERATE.
- Podemos usar esta palavra chave para declarar e inicializar uma seqüência de constates inteiras.
- Sua principal vantagem é quando não se quer inicializar todas as constantes e cada uma precisa ter um valor único.
- Por padrão, a primeira constante equivale a zero. As restantes equivalem à anterior incrementada em um.

Palavra chave **enum** :: Exemplo

```
enum colors {RED, GREEN, BLUE};
```

- **colors** é o nome dado para o grupo de constantes (opcional).
- Se não é atribuído um valor para as constantes, o valor padrão atribuído para o primeiro elemento na lista (RED em nosso caso) será 0 (zero).
- As demais constantes com valor indefinido terão valor o da constante anterior mais 1 (um).
- No nosso caso, GREEN = 1 e BLUE = 2.

Palavra chave **enum**

- Pode-se atribuir valores para as constantes.

```
enum colors {RED = 1,
        YELLOW,
        GREEN = 6,
        BLUE};
```

- Agora RED = 1, YELLOW = 2, GREEN = 6 e BLUE = 7.
- Geralmente usa-se letras maiúsculas para dar nome às constantes.

Variáveis **enum**

- Constantes enumeradas são do tipo inteiro, então **`int x = RED;`** está correto.
- Porém, você pode criar o seu próprio tipo de dado, por exemplo **`colors`**.

```
enum colors corfundo;
```

- Declara uma variável chamada **`corfundo`**, a qual é do tipo de dado enumerado **`colors`**.

Exercícios

- Escreva uma função que receba dois structs do tipo dma (dia, mês e ano), cada um representando uma data válida, e devolva o número de dias que decorreram entre as duas datas.
- Escreva uma função que receba um número inteiro que representa um intervalo de tempo medido em minutos e devolva o correspondente número de horas e minutos (por exemplo, converte 131 minutos em 2 horas e 11 minutos). Use uma struct como a seguinte:

```
struct hm {
  int horas;
  int minutos;
};
```

Exercícios

- Escreva um programa que preencha uma variável estrutura e depois mostre-a na tela. A estrutura deverá conter campos para: nome, endereço, idade, telefone, data. Sendo que data deverá ser uma estrutura com os campos: dia, mês e ano.
- Modifique o programa anterior, copiando os campos de nome e data para uma outra variável estrutura e exibindo os conteúdos da duas variáveis.
- Imagine a seguinte situação: você precisa armazenar na memória os dados referentes a 60 alunos (nome e série) e suas notas ao longo do ano (4 bimestres) em 4 disciplinas, inglês, francês, matemática e português. Como você poderia organizar essas informações para serem armazenadas na memória? Faça um programa para isso.
- Escreva um programa que usando tipos enumerados escreva a data do dia, na forma p. ex. 12 de Outubro de 1996.