Aula: Estruturas heterogêneas

Introdução a Programação

Túlio Toffolo & Puca Huachi
http://www.toffolo.com.br

Departamento de Computação
Universidade Federal de Ouro Preto

Aula: Estruturas heterogêneas

Aula: Estruturas heterogêneas

Struct

- `struct`: palavra reservada que cria um novo tipo de dados.
- Tipos conhecidos: `char`, `int`, `float`, `double` e `void`.
- Estrutura: é um tipo de estrutura de dados heterogênea; agrupa itens de dados de diferentes tipos.
- Cada item de dado é denominado membro (ou campo);
- `struct` define um tipo de dados (estrutura): informa ao compilador o nome, o tamanho em bytes e a maneira como ela deve ser armazenada e recuperada da memória.
- Ao ser definido, o tipo passa a existir e pode ser utilizado para criar variáveis.

Exemplo: armazenando dados de um aluno

```c
#include <stdio.h>

struct Aluno {
    int nMat;       // número de matrícula
    float nota[3]; // três notas
    float media;   // média aritmética
};                 // fim da definição da estrutura (com ;)

int main()
{
    struct Aluno bart;   // declara a variável do tipo 'struct Aluno'
    bart.nMat = 1521001;
    bart.nota[0] = 8.5;
    bart.nota[1] = 9.5;
    bart.nota[2] = 10.0;
    bart.media = ( bart.nota[0]+ bart.nota[1] + bart.nota[2] ) / 3.0;
    printf("Matrícula: %d\n", bart.nMat);
    printf("Média    : %.1f\n", bart.media);
    return 0;
}
```

Observações

- A instrução `struct Aluno bart;` declara uma variável `bart` do tipo `struct Aluno`.
  - Memória é reservada para os membros: 4 bytes para nMat, 12 bytes para a matriz nota (3 floats) e 4 bytes para media.
- Os membros da estrutura são armazenados em sequência na memória.
- O operador **ponto** (.) conecta o nome de uma variável de estrutura a um membro dela.
- As declarações de uma variável simples e de uma variável de estrutura seguem o mesmo formato:

```
1  struct Aluno bart;
2  struct Aluno *ponteiroAluno;
3  int home;
```

Definição da Estrutura

- A definição de uma estrutura não cria uma variável.
- Define um tipo de dados (estrutura): informa ao compilador o nome, o tamanho em bytes e a maneira como ela deve ser armazenada e recuperada da memória.
- Não reserva memória.
- A estrutura é um tipo de dado cujo formato é definido pelo programador.

Inicializando estruturas

A inicialização é semelhante a inicialização das matrizes.

```
1  struct Data {
2      int  dia;
3      char mes[10];
4      int  ano;
5  };
6
7  struct Data natal = { 25, "Dezembro", 2016 };
8  struct Data niver = { 20, "Outubro",  1986 };
```

Obs.: as variáveis são inicializadas juntamente com suas declarações. Os valores atribuídos aos membros devem ser colocados **na ordem em que foram definidos na estrutura**, separados por vírgula e entre chaves.

Inicializando estruturas

Lembre-se, os membros também podem ser inicializados individualmente.

```
1  struct Data {
2  int  dia;
3  char mes[10];
4  int  ano;
5  };
6
7  struct Data natal, niver;
8  natal.dia = 25;
9  strcpy(natal.mes, "Dezembro");
10 natal.ano = 2020;
```

Observe que para separar o nome da variável dos seus membros é utilizado o operador **ponto** (.) entre eles.

Observações

Alternativas para declaração:

```c
struct Aluno {
    int nMat;        // número de matrícula
    float nota[3];   // três notas
    float media;     // média aritmética
} LisaSimpson;       // declaração da variável
```

```c
struct Aluno {
    int nMat;            // número de matrícula
    float nota[3];       // três notas
    float media;         // média aritmética
} Lisa, Marge, Meggie;   // declaração de múltiplas variáveis
```

```c
// podemos até suprimir o nome do tipo:
struct {             // struct sem nome
    int nMat;        // número de matrícula
    float nota[3];   // três notas
    float media;     // média aritmética
} Lisa, Marge, Meggie;
```

Atribuição entre estruturas

O uso de variáveis de estruturas é similar ao uso das variáveis que estamos acostumados a utilizar...

- Uma variável estrutura pode ser atribuída à outra variável do mesmo tipo por meio de uma atribuição simples.

```
1  struct Data natal = { 25, "Dezembro", 2016 };
2
3  struct Data natalDesteAno;
4  natalDesteAno = natal;
```

- **Importante**:
  - valores dos membros da estrutura são atribuídos de uma única vez;
  - a atribuição entre vetores/matrizes deve ser feita elemento por elemento.

Exemplo

Faça um programa que recebe um determinado tempo em segundos e converte para horas, minutos e segundos e imprime esses valores. Utilize a seguinte estrutura:

```
1  struct Tempo {
2      int horas;
3      int minutos;
4      int segundos;
5  };
```

Exemplo

```c
#include <stdio.h>

struct Tempo {
    int horas;
    int minutos;
    int segundos;
};

int main()
{
    struct Tempo tempo;
    int segundos;

    printf("\nDigite o tempo em segundos: ");
    scanf("%d", &segundos);

    tempo.horas = segundos / 3600;
    tempo.minutos = (segundos % 3600) / 60;
    tempo.segundos = (segundos % 3600) % 60;

    printf("\nTempo em horas, minutos e segundos:");
    printf(" %d:%d:%d\n\n", tempo.horas, tempo.minutos, tempo.segundos);

    return 0;
}
```

O comando *typedef*

- O comando `typedef` define um apelido (*alias*) para um tipo.
- Em geral, apelidos simplificam o uso de estruturas em C.
- Exemplo:

```
1  typedef struct { // não precisamos definir o nome aqui
2      int  dia;
3      char mes[10];
4      int  ano;
5  } Data;          // 'apelido' (novo nome) para a estrutura: Data
```

- Uso simplificado (omitimos a palavra *struct* ao declarar variáveis):

```
1  Data natal = { 25, "Dezembro", 2016 };
2
3  Data natalDesteAno;
4  natalDesteAno = natal;
```

Exemplo de uso e operações em structs:

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

int main()
{
    Venda A = {20, 110.0};
    Venda B = {3, 258.0};
    Venda total;

    // soma membro a membro
    total.pecas = A.pecas + B.pecas;
    total.preco = A.preco + B.preco;
}
```

Erro comum

```c
// ERRO!
total = A + B;
```

Estruturas aninhadas

```c
typedef struct {
    int  dia;
    char mes[10];
    int  ano;
} Data;

typedef struct {
    int   pecas;
    float preco;
    Data  diaVenda;
} Venda;

int main()
{
    // exemplo de declaração
    Venda v = {20, 110.0, {7, "Novembro", 2015} };

    // exemplo de uso:
    printf("Ano da venda: %d", v.diaVenda.ano);

    return 0;
}
```

Estruturas em funções

As estruturas podem ser passadas como argumentos de funções da mesma maneira que as variáveis simples.

- O nome de uma estrutura em C não é um endereço, portanto ela pode ser passada por **valor**.
- Exemplo: função que recebe duas estruturas como argumento e imprime os valores da soma de seus membros.

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  // protótipo (com passagem por valor)
7  void imprimeTotal(Venda v1, Venda v2);
```

Estruturas em funções

Exemplo utilizando passagem por **valor**:

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

void imprimeTotal(Venda v1, Venda v2)
{
    Venda total = {0, 0.0};
    total.pecas = v1.pecas + v2.pecas;
    total.preco = v1.preco + v2.preco;
    printf("Nro peças:   %d\n", total.pecas);
    printf("Preço total: %.2f\n", total.preco);
}

int main()
{
    Venda v1 = {1, 20}, v2 = {3, 10};
    imprimeTotal(v1, v2);
    return 0;
}
```

Estruturas em funções

- Podemos usar ponteiros para fazer passagem por **referência**:

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

// protótipo (com passagem por referência)
void imprimeTotal(Venda *v1, Venda *v2);
```

Estruturas em funções

Exemplo utilizando **ponteiros**:

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  void imprimeTotal(Venda *v1, Venda *v2)
7  {
8      Venda total = {0, 0.0};
9      total.pecas = (*v1).pecas + (*v2).pecas;
10     total.preco = (*v1).preco + (*v2).preco;
11     printf("Nro peças:   %d\n", total.pecas);
12     printf("Preço total: %.2f\n", total.preco);
13 }
14
15 int main()
16 {
17     Venda v1 = {1, 20}, v2 = {3, 10};
18     imprimeTotal(&v1, &v2);
19     return 0;
20 }
```

Estruturas em funções

Exemplo utilizando **ponteiros** (alternativa):

```c
typedef struct {
    int   pecas;
    float preco;
} Venda;

void imprimeTotal(Venda *v1, Venda *v2)
{
    Venda total = {0, 0.0};
    total.pecas = v1->pecas + v2->pecas; //v1->pecas ou (*v1).pecas
    total.preco = v1->preco + v2->preco;
    printf("Nro peças:   %d\n", total.pecas);
    printf("Preço total: %.2f\n", total.preco);
}

int main()
{
    Venda v1 = {1, 20}, v2 = {3, 10};
    imprimeTotal(&v1, &v2);
    return 0;
}
```

Observe que neste exemplo utilizamos o operador **seta** (**->**) para acessar

Vetores/matrizes de estruturas

- Uma lista de peças e preços é composta por várias vendas (provavelmente mais de duas).
- Cada venda pode ser descrita por uma variável do tipo `Venda`.
- As diversas vendas podem ser armazenadas em um vetor de estruturas.

```
1  void imprimeTotal(Venda v[], int n);
```

```
1  Venda novaVenda();
```

```c
int main() {
    Venda vendas[50];  // cria um array de estruturas
    int n = 0, opcao; // inicialmente não temos nenhuma venda
    do {
        printf("Digite 1 para entrar uma venda\n");
        printf("       2 para imprimir o total\n");
        printf("       0 para terminar\n");
        scanf("%d", &opcao);
        switch (opcao) {
            case 1:
                vendas[n] = novaVenda();
                n++;
                break;
            case 2:
                imprimeTotal(vendas, n);
                break;
        }
    } while (opcao != 0);
    return 0;
}
```

Função que cria e retorna uma nova venda (usando `scanf`):

```
1  Venda novaVenda()
2  {
3      Venda v;
4      printf("Digite a quantidade e o valor da venda:\n");
5      scanf("%d %f", &v.pecas, &v.preco);
6      return v;
7  }
```

Função que calcula o total de várias vendas:

```
1  void imprimeTotal(Venda v[], int n)
2  {
3      Venda total = {0, 0.0};
4      for (int i = 0; i < n; i++) {
5          total.pecas += v[i].pecas;
6          total.preco += v[i].preco;
7      }
8      printf("Nro peças:   %d\n", total.pecas);
9      printf("Preço total: %f\n", total.preco);
10 }
```

Aula: Estruturas heterogêneas

Exercícios

Exercício 1

Crie um programa que apresente, na tela de abertura, um menu com opções para:

1. inserir uma nova pessoa: nome, altura e ano de nascimento;
2. listar todos os nomes e respectivas alturas;
3. listar os nomes das pessoas que nasceram depois de um certo ano.

Cada uma destas opções deve ser implementada em uma função separada. O programa deve ser capaz de armazenar dados de até 100 pessoas.

Exercícios

Exercício 2

Crie uma estrutura (*struct*) para armazenar dados de um funcionário:

1. nome (até 50 caracteres),
2. função (até 50 caracteres),
3. idade,
4. salário.

Em seguida, crie um programa que lê os dados de n funcionários.

Perguntas?