



Aula 25: Estruturas heterogêneas

Introdução a Programação

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

BCC201 – 2019/2
Departamento de Computação – UFOP

Aulas anteriores

- Arquivos de texto
(conteúdo adiantado para entrega da primeira parte do TP)

Aula de hoje

- 1 Estruturas heterogêneas
- 2 Exemplos
- 3 Próxima aula

Aula de hoje

1 Estruturas heterogêneas

2 Exemplos

3 Próxima aula

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

```
1  #include <stdio.h>
2
3  struct Aluno {
4      int nMat;        // número de matrícula
5      float nota[3];  // três notas
6      float media;    // média aritmética
7  };                  // fim da definição da estrutura (com ;)
8
9  int main()
10 {
11     struct Aluno bart; // declara a variável do tipo 'struct Aluno'
12     bart.nMat = 1521001;
13     bart.nota[0] = 8.5;
14     bart.nota[1] = 9.5;
15     bart.nota[2] = 10.0;
16     bart.media = ( bart.nota[0]+ bart.nota[1] + bart.nota[2] ) / 3.0;
17     printf("Matrícula: %d\n", bart.nMat);
18     printf("Média      : %.1f\n", bart.media);
19     return 0;
20 }
```

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.

Observações

Alternativas para declaração:

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

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

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

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.

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.

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:

```
1  typedef struct {
2      int   pecas;
3      float preco;
4  } Venda;
5
6  int main()
7  {
8      Venda A = {20, 110.0};
9      Venda B = {3, 258.0};
10     Venda total;
11
12     // soma membro a membro
13     total.pecas = A.pecas + B.pecas;
14     total.preco = A.preco + B.preco;
15 }
```

Erro comum

```
1  // ERRO!
2  total = A + B;
```

Estruturas aninhadas

```
1  typedef struct {
2      int    dia;
3      char  mes[10];
4      int    ano;
5  } Data;
6
7  typedef struct {
8      int    pecas;
9      float preco;
10     Data  diaVenda;
11 } Venda;
12
13 int main()
14 {
15     // exemplo de declaração
16     Venda v = {20, 110.0, {7, "Novembro", 2015} };
17
18     // exemplo de uso:
19     printf("Ano da venda: %d", v.diaVenda.ano);
20
21     return 0;
22 }
```

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**:

```
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

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

```
1  typedef struct {
2      int    pecas;
3      float  preco;
4  } Venda;
5
6  // protótipo (com passagem por referência)
7  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):

```
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; v1->pecas ou (*v1).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 }
```

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  int main() {
2      Venda vendas[50]; // cria um array de estruturas
3      int n = 0, opcao; // inicialmente não temos nenhuma venda
4      do {
5          printf("Digite 1 para entrar uma venda\n");
6          printf("          2 para imprimir o total\n");
7          printf("          0 para terminar\n");
8          scanf("%d", &opcao);
9          switch (opcao) {
10             case 1:
11                 vendas[n] = novaVenda();
12                 n++;
13                 break;
14             case 2:
15                 imprimeTotal(vendas, n);
16                 break;
17             }
18         } while (opcao != 0);
19         return 0;
20     }

```

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 de hoje

1 Estruturas heterogêneas

2 Exemplos

3 Próxima aula

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.

Aula de hoje

1 Estruturas heterogêneas

2 Exemplos

3 Próxima aula

Próxima aula

- Mais sobre estruturas de dados heterogêneas



Perguntas?