

Linguagem de Programação I

- Estruturas -

Prof. Ulysses Santos Sousa
ulyssessousa@ifma.edu.br

Aula 05

Roteiro

- Estruturas
- Struct dentro de struct
- Acessando o campo de uma estrutura
- Exemplo
- Vetor de struct
- Exemplo 2
- Tipos de dados enumerados: enum
- Uniões

Estruturas

- **Conceitos**

- Em C, é uma coleção de variáveis referenciadas por um nome, fornecendo uma maneira conveniente de se ter informações relacionadas agrupadas.
- Uma *Definição de Estrutura* forma um modelo que pode ser usado para criar variáveis de estruturas.
- As variáveis que correspondem à estrutura são chamadas *membros de estrutura* (ou elementos, ou campos)

Estruturas

- Estruturas (struct)
 - Permitem o agrupamento de valores de tipos diferentes.
- Forma geral:

```
struct nome_do_tipo_da_estrutura
{
    tipo_1 nome_1;
    tipo_2 nome_2;
    ...
    tipo_n nome_n;
} variáveis_estrutura;
```

Estruturas

- Declaração
 - Exemplo:

```
struct tipo_endereco
{
    char rua[50];
    int nr;
    char bairro[20];
    char cidade[20];
    char estado[2];
    int cep;
} endereco;
```

Estruturas

- **Entendendo a declaração**
 - struct
 - Comando utilizado para criar a estrutura.
 - tipo_endereco
 - É o especificador de tipo da estrutura.
 - char rua[50]; int nr; char bairro[20]; char cidade[20]; char estado[2]; int cep
 - São as variáveis que compreendem a estrutura.
 - endereco
 - É o nome da variável.

Definição e inicialização na mesma instrução

- Podemos inicializar uma variável do tipo estrutura da seguinte forma:

```
struct Data{  
    int dia;  
    char mês[10];  
    int ano;  
} natal = {25, "dezembro", 2016},  
trabalhador = {1, "maio", 2017};
```

Struct dentro de struct

- Uma struct também pode fazer parte de outra struct.
- Exemplo:

```
struct tipo_dados2
{
    int cod_filial;
    char nome[32];
} filial;

struct tipo_dados3
{
    int cod;
    float vlr;
    int qtde;
    struct tipo_dados2 filial;
} componente;
```


Acessando o campo de uma estrutura

- Para acessar os campos de uma estrutura devemos utilizar o ponto (.)
- Exemplo:
 - componente.cod;
 - componente.vlr;
 - componente.qtde;
 - componente.filial.cod_filial;
 - componente.filial.nome.

Exemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     struct tipo_endereco
7     {
8         int numero;
9         char rua[40];
10        char bairro[30];
11    }endereco;
12
13    struct tipo_funcionario
14    {
15        int cod;
16        char nome[40];
17        struct tipo_endereco endereco;
18    }funcionario;
19
```

Exemplo (cont.)

```
20      /*leitura de dados*/
21      printf("Dados do funcionario:\n");
22      printf("Digite o codigo: ");
23      scanf("%d", &funcionario.cod);
24      printf("Digite o nome: ");
25      fflush(stdin);
26      gets(funcionario.nome);
27      printf("Digite o numero da residencia: ");
28      scanf("%d", &funcionario.endereco.numero);
29      printf("Digite o nome da rua: ");
30      fflush(stdin);
31      gets(funcionario.endereco.rua);
32      printf("Digite o nome do bairro: ");
33      fflush(stdin);
34      gets(funcionario.endereco.bairro);
35
```

Exemplo (cont.)

```
36  /*Mostrando os valores na tela*/
37  printf("\nDados do funcionario:\n");
38  printf("codigo: %d\n", funcionario.cod);
39  printf("nome: %s\n", funcionario.nome);
40  printf("Numero da casa: %d\n", funcionario.endereco.numero);
41  printf("Rua: %s\n", funcionario.endereco.rua);
42  printf("Bairro: %s\n", funcionario.endereco.bairro);
43  return 0;
44 }
```

Vetor de Struct

- No exemplo anterior era possível armazenar os dados referentes a apenas um funcionário.
- Para armazenar dados de mais de um funcionário é possível utilizar vetores de estrutura.

Exemplo 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     struct tipo_endereco
7     {
8         int numero;
9         char rua[40];
10        char bairro[30];
11    }endereco;
12
13    struct tipo_funcionario
14    {
15        int cod;
16        char nome[40];
17        struct tipo_endereco endereco;
18    }funcionario[3];
19
20    int i;
21
```

Exemplo 2 (cont.)

```
22  /*leitura de dados*/
23  for (i = 0; i < 3; i++)
24  {
25      printf("Digite os dados do funcionario %d:\n", i + 1);
26      printf("Digite o codigo: ");
27      scanf("%d", &funcionario[i].cod);
28      printf("Digite o nome: ");
29      fflush(stdin);
30      gets(funcionario[i].nome);
31      printf("Digite o numero da residencia: ");
32      scanf("%d", &funcionario[i].endereco.numero);
33      printf("Digite o nome da rua: ");
34      fflush(stdin);
35      gets(funcionario[i].endereco.rua);
36      printf("Digite o nome do bairro: ");
37      fflush(stdin);
38      gets(funcionario[i].endereco.bairro);
39  }
40
```

Exemplo 2 (cont.)

```
41
42  /*Mostrando os valores na tela*/
43  for (i = 0; i < 3; i++)
44  {
45      printf("\nDados do funcionario %d:\n", i + 1);
46      printf("codigo: %d\n", funcionario[i].cod);
47      printf("nome: %s\n", funcionario[i].nome);
48      printf("Numero da casa: %d\n", funcionario[i].endereco.numero);
49      printf("Rua: %s\n", funcionario[i].endereco.rua);
50      printf("Bairro: %s\n", funcionario[i].endereco.bairro);
51  }
52
53  return 0;
54 }
```


Novos nomes para tipos existentes

- **typedef**
 - Declarações com typedef não criam novos tipos de dados, apenas criam novos nomes (sinônimos) para os tipos existentes.
- **Sintaxe:**
 - `typedef tipo_existente sinônimo;`

Novos nomes para tipos existentes

- typedef (exemplo):

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      typedef int inteiro;
7      inteiro x;
8      x = 10;
9      printf("%d\n", x);
10     return 0;
11 }
```

A palavra *inteiro* foi utilizada no lugar da palavra *int*.

Usando *typedef* com *struct*

- 1ª Forma:

```
struct Aluno
{
    int matricula;
    float nota[3];
    float media;
};

typedef struct Aluno Aluno;

Aluno jose;
```

Declaração de uma
variável do tipo ***Aluno***.

Usando *typedef* com *struct*

- 2ª Forma:

```
typedef struct Aluno
{
    int matricula;
    float nota[3];
    float media;
}Aluno;
```

```
Aluno jose;
```

Declaração de uma
variável do tipo ***Aluno***.

Usando *typedef* com *struct*

- 3ª Forma:

```
typedef struct
{
    int matricula;
    float nota[3];
    float media;
}Aluno;
```

```
Aluno jose;
```

Declaração de uma
variável do tipo **Aluno**.

Tipos de dados enumerados: enum

- Constituem um conjunto de constantes inteiras, em que cada uma delas é representada por um nome.
- São usados quando conhecemos o conjunto de valores que uma variável pode assumir.

Tipos de dados enumerados: enum

- A palavra enum enumera a lista de nomes automaticamente, dando-lhes números em sequência (0, 1, 2 etc).
- Vantagem:
 - Torna o programa mais claro.

Tipos de dados enumerados: enum

- Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum Mes {Jan =1, Fev, Mar, Abr, Mai, Jun,
5           Jul, Ago, Set, Out, Nov, Dez};
6
7  int main()
8  {
9      enum Mes m1, m2, m3;
10     m1 = Abr;
11     m2 = Jun;
```


Usando *typedef* com *enum*

- Exemplo:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef enum {Jan =1, Fev, Mar, Abr, Mai, Jun,
5               Jul, Ago, Set, Out, Nov, Dez} Mes;
6
7  int main()
8  {
9      Mes m1, m2, m3;
10     m1 = Abr;
11     m2 = Jun;
```

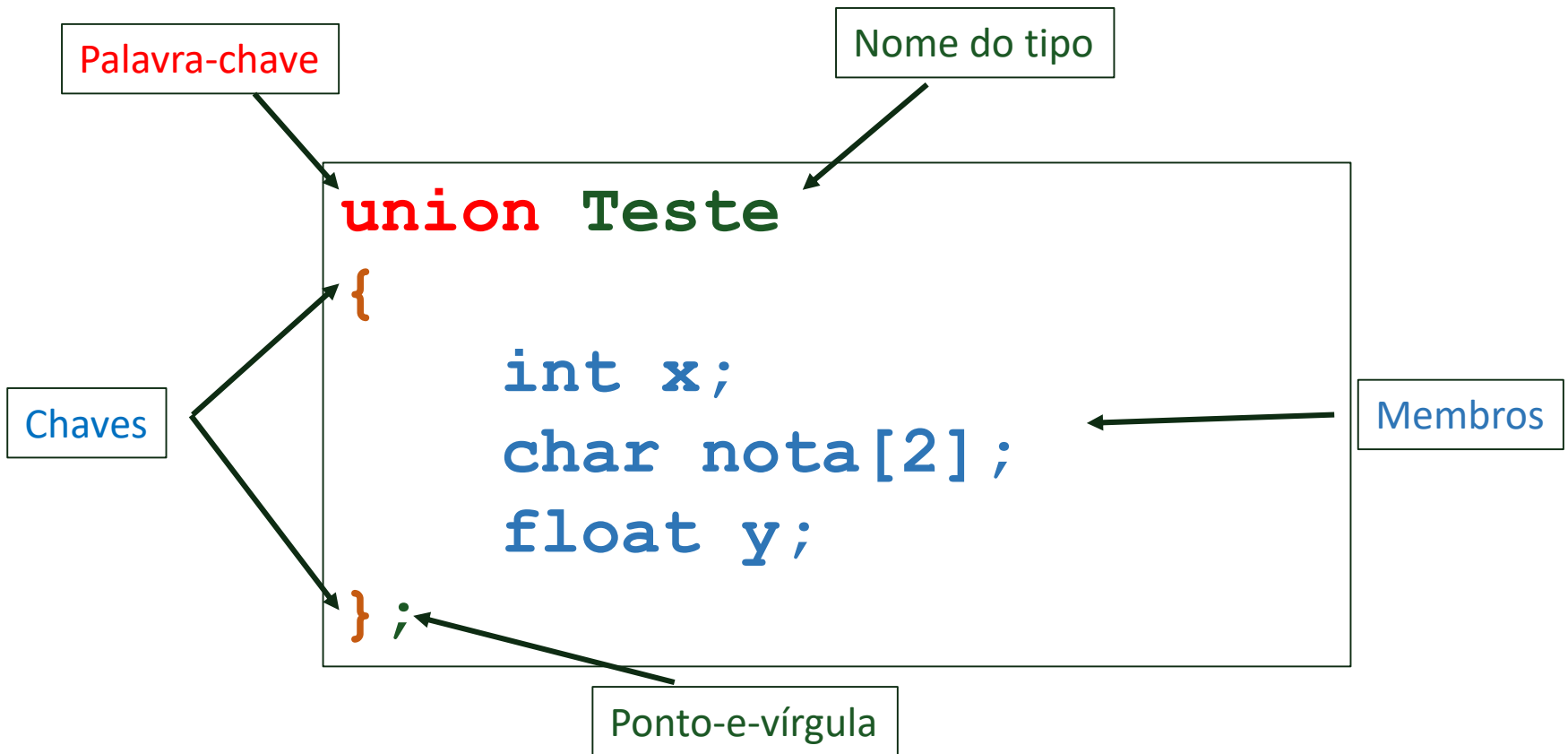
Uniões

- É definida através da palavra *union*
 - a sintaxe de definição é idêntica à da palavra *struct*.
 - Porém, utiliza um mesmo espaço de memória a ser compartilhado com diferentes membros.

Uma union é o meio pelo qual um pedaço de memória é tratado ora como uma variável de um certo tipo, ora como outra variável de outro tipo. Uniões podem ser usadas para poupar memória.

Uniãos

- Sintaxe:




Uniãos

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef union{
6      int idade;
7      char nome[10];
8      float altura;
9  }Pessoa;
10
11  typedef enum {
12      tipoInt = 1, tipoStr, tipoFloat
13  };
14
15  int main()
16  {
17      int opcao;
18      Pessoa p;
19      printf("Informe uma das opções:\n");
20      printf("1 - idade\n2 - nome\n3 - altura\n");
21      scanf("%d", &opcao);
22      fflush(stdin);
23      switch(opcao)
24      {
25          case tipoInt:
26              scanf("%d", &p.idade); break;
27          case tipoStr:
28              gets(p.nome); break;
29          case tipoFloat:
30              scanf("%f", &p.altura);
31      }
32      return 0;
33  }
```

Unões e Estruturas

- Exemplo:

```
typedef struct{  
    int x, y;  
}DuplaInt;  
  
typedef struct{  
    float a, b;  
}DuplaFloat;  
  
typedef union{  
    DuplaInt di;  
    DuplaFloat df;  
}dupla;
```

Unões Anônimas

- **Definição:**

- São uniões definidas sem a especificação do nome do tipo.

- **Características:**

- Só podem ser definidas como membros de estruturas.
- São acessadas diretamente, por meio do nome da variável *struct*, como se fossem membros da estrutura.

Deve-se indicar a cada tempo qual é o membro da **union** corrente, pois existirá somente um membro de cada vez.

Unões Anônimas

```
typedef struct{
    int tipo;
    union{
        char a;
        int b;
        float c;
    };
}Teste;

int main()
{
    Teste teste;
    teste.a = 'a';
    printf("teste = %c\n", teste.a);
    teste.b = 15;
    printf("teste = %d\n", teste.b);
    teste.c = 5.4;
    printf("teste = %.2f\n", teste.c);
    return 0;
}
```

Operador *sizeof*

- Opera sobre o nome de um tipo de dado ou de uma variável e resulta o seu tamanho em bytes.
 - Uma variável do tipo **union** tem o tamanho do maior membro.
- **Exemplo:**
 - `sizeof(int)`
 - `sizeof(union Pessoa);`

Exercício

- Faça um programa para cadastrar alunos e professores em um vetor de struct. Os dados dos alunos são: nome, serie e nível. Os dados dos professores são: nome, escolaridade, disciplina. O programa deverá exibir os dados de todos os alunos e professores cadastrados.

Referências

- MIZRAHI, V. V. Treinamento em Linguagem C. 2ª Edição. São Paulo: Person Prentice Hall, 2008.
- SANTANNA, Solimara Ravani de. Lógica de programação e automação. Curitiba: Livro Técnico, 2012. 144 p. ISBN 8563687340.