

Laboratório de programação

Tipos definidos pelo programador

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

Estruturas

Declaração de variáveis tipo struct

Acessando os campos de uma estrutura

Inicialização de estruturas

Array de estruturas

Atribuição entre estruturas

Estruturas aninhadas

Ponteiros para estruturas

Estruturas em funções

- Os tipos de variáveis vistos até agora podem ser classificados em duas categorias:
 1. Tipos básicos: **char**, **int**, **float**, **double** e **void**.
 2. Tipos compostos homogêneos: *arrays*.
- A linguagem C permite criar novos tipos de dados a partir dos tipos básicos.
- Comandos usados para criar um novo tipo de dado:
 - Estruturas: comando **struct**
 - Uniões: comando **union**
 - Enumerações: comando **enum**
 - Renomear um tipo existente: comando **typedef**

- As estruturas são tipos de dados construídas usando variáveis de outros tipos. Na definição de uma estrutura é necessário usar a palavra reservada **struct**:

```
1 struct Cadastro {  
2     char nome[50];  
3     int idade;  
4     char rua[50];  
5     int numero;  
6 };
```


```
1 struct Aluno {  
2     char nome[50];  
3     int matricula;  
4     float nota[3];  
5 };
```

Warning!


A definição de uma estrutura não reserva espaço na memória, em vez disso ela cria um **novo tipo de dado** que é usado para declarar variáveis.

- As estruturas são tipos derivados de dados construídas usando variáveis de outros tipos. Examine a seguinte definição de estrutura:

```
1 struct Cadastro {  
2     char nome[50];  
3     int idade;  
4     char rua[50];  
5     int numero;  
6 };
```



```
1 struct Aluno {  
2     char nome[50];  
3     int matricula;  
4     float nota[3];  
5 };
```



Warning!

Depois do símbolo de fechar chaves (}) da estrutura, é necessário colocar um ponto e vírgula (;).

- Uma vez definida a estrutura, uma variável pode ser declarada de modo similar aos tipos já existentes:

```
1 struct Cadastro c;
```

Warning!

Por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável declarada.

- A declaração de variáveis também pode ser incorporada na definição da estrutura:

```
1 struct Carta {  
2     char face[10];  
3     char naipe[10];  
4     int valor;  
5 } carta1, carta2, carta3;
```

- O nome da estrutura é opcional. Por exemplo:

```
1 struct {  
2     char nome[80];  
3     int idade;  
4 } pessoa;
```

- Se uma definição de estrutura não possuir nome as variáveis do seu tipo só poderão ser declaradas na própria definição.

- Cada campo (variável) da estrutura pode ser acessado usando o operador "." (ponto).

```
1 struct Cadastro c;  
2 // ...  
3 printf("%s ", c.nome);  
4 printf("%d ", c.idade);  
5 printf("%s ", c.rua);  
6 printf("%d ", c.numero);  
7 // ...
```



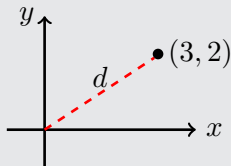
```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Cadastro {
5      char nome[50];
6      int idade;
7      char rua[50];
8      int numero;
9  };
10 int main() {
11     struct Cadastro c;
12     //Lê do teclado uma string e armazena no campo nome
13     scanf("%s", c.nome);
14     //Lê do teclado um valor inteiro e armazena no campo idade
15     scanf("%d", &c.idade);
16     //Atribui a string "Avenida Brasil" para o campo rua
17     scanf("%s", c.rua);
18     //Atribui o valor 1082 para o campo numero
19     scanf("%d", &c.numero);
20     return 0;
21 }
```

Exercício 22

Implemente um programa que leia o nome, a idade e o endereço de uma pessoa e armazene esses dados em uma estrutura. Em seguida, imprima na tela os dados da estrutura lida.

Exercício 23

Crie uma estrutura para representar as coordenadas de um ponto no plano (posições x e y). Em seguida, declare e leia do teclado um ponto e exiba a distância d dele até a origem das coordenadas, isto é, a posição $(0,0)$.



Warning!

Lembre-se: uma estrutura pode ser vista como um simples agrupamento de dados.

Warning!

Como cada campo é independente dos demais, outros operadores podem ser aplicados a cada campo. Por exemplo, pode-se comparar a idade de dois cadastros.

```
1  struct Cadastro a, b;  
2  // ...  
3  if (a.idade > b.idade) {  
4      // ...  
5  }  
6  // ...
```

- As estruturas podem ser inicializadas usando listas de inicializadores como arrays.
- Seus campos podem ser nomeados ou não:
 - Caso os nomes dos campos sejam omitidos, a ordem de declaração deve ser obedecida.
 - Usando os nomes dos campos pode-se atribuir os valores em qualquer ordem.

```
1 struct Carta {  
2     char face;  
3     char naipe[10];  
4     int valor;  
5 };  
6 // ...  
7 struct Carta c = {'A', "Copas", 1};  
8 struct Carta d = {.valor=12, .face='Q', .naipe="Espadas"};
```

- A declaração de um array de estruturas é similar à declaração de um array de um tipo básico.

```
1 struct Usuario u[20];  
2 struct Carta baralho[52];  
3 struct Cadastro c[10];
```

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct Usuario {
5      char login[10];
6      int senha;
7  };
8
9  int main() {
10     struct Usuario u[3];
11     int i;
12
13     for (i=0; i<3; i++) {
14         scanf("%s", u[i].login);
15         scanf("%d", &u[i].senha);
16     }
17
18     return 0;
19 }
```

Exercício 24

Crie uma estrutura capaz de armazenar o primeiro nome e a idade de uma pessoa. Agora, escreva um programa que leia os dados de 50 pessoas. Calcule e exiba os nomes da pessoa mais nova e da mais velha.

1. Baixe o arquivo `name_age.txt`
2. Execute o comando abaixo no terminal

```
./a.out < name_age.txt
```

OBS: se você criou o executável do seu programa com outro nome substitua o `a.out` pelo nome do seu executável.

- A atribuição entre duas variáveis de estrutura faz com que os conteúdos das variáveis contidas dentro de uma estrutura sejam copiados para a outra.

Warning

Atribuições entre estruturas só podem ser feitas quando as estruturas são AS MESMAS, ou seja, quando possuem o mesmo nome!


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct Ponto {
4      int x;
5      int y;
6  };
7  struct NovoPonto {
8      int x;
9      int y;
10 };
11 int main(){
12     struct Ponto p1, p2 = {1,2};
13     struct NovoPonto p3 = {3,4};
14     p1 = p2;
15     printf("p1 = %d e %d", p1.x,p1.y);
16     //ERRO! TIPOS DIFERENTES
17     p1 = p3;
18     printf("p1 = %d e %d", p1.x,p1.y);
19     return 0;
20 }
```

- No caso de trabalharmos com arrays de estruturas, a atribuição entre diferentes elementos do array também é válida.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Usuario {
5      char login[10];
6      int senha;
7  };
8
9  int main(){
10     struct Usuario u[3];
11     //...
12     u[0] = u[2];
13     //...
14     return 0;
15 }
```

- Estruturas podem ainda ser aninhadas criando tipos mais complexos.

```
1 struct Endereco {  
2     char rua[50];  
3     int numero;  
4 };  
5 struct Cadastro {  
6     char nome[50];  
7     int idade;  
8     struct Endereco endereco;  
9 };
```

- A variável **endereco** também é uma estrutura. Sendo assim, o operador ponto (.) é novamente utilizado para acessar as variáveis dentro dessa estrutura.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct Endereco {
5      char rua[50];
6      int numero;
7  };
8  struct Cadastro {
9      char nome[50];
10     int idade;
11     struct Endereco endereco;
12 };
13
14 int main() {
15     struct Cadastro c = {"Noeh", 82, {"R. dos Alagados", 105}};
16
17     printf("%s %d anos: %s, %d\n", c.nome, c.idade,
18                                     c.endereco.rua, c.endereco.numero);
19     return 0;
20 }
```

- Usando o `scanf` com estruturas aninhadas.

```
1  int main() {  
2      struct Cadastro c;  
3  
4      scanf("%s", c.nome);  
5      scanf("%d", &c.idade);  
6  
7      scanf("%s", c.endereco.rua);  
8      scanf("%d", &c.endereco.numero);  
9  
10     return 0;  
11 }
```

- Define-se um ponteiro para uma estrutura de forma análoga aos tipos básicos.

```
1 struct Ponto {  
2     int x, y;  
3 };  
4 // ...  
5 struct Ponto r = {3, 7};  
6 struct Ponto *p;  
7 // ...  
8 p = &r;
```

- Contudo para acessar os campos da estrutura usando o ponteiro é necessário obedecer a seguinte sintaxe:

```
1 // ...  
2 printf("%d %d", (*p).x, (*p).y);
```

- Desreferenciamento de um ponteiro para uma estrutura

desreferenciamento → $(*p).x$ ← *acesso ao campo x*

- Para simplificar essa sintaxe pode-se usar o operador seta (\rightarrow):

$p \rightarrow x$

- As duas sintaxes são equivalentes.

```
1 // ...  
2 printf("%d", (*p).x);  
3 printf("%d", p->y);
```

- Estruturas podem ser passadas como parâmetros a uma função tanto por **valor** como por **referência**.
- Para passar uma estrutura chamada por referência, passe o endereço da variável da estrutura.

```
1 struct Usuario u;  
2 //...  
3 printUsuario(u); // passagem por valor  
4 //...  
5 scanUsuario(&u); // passagem por referencia
```



```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct Usuario {
5      char login[10];
6      int senha;
7  };
8
9  void printUsuario(struct Usuario u) {
10     printf("Login: %s Senha: %d\n", u.login, u.senha);
11 }
12
13 int main() {
14     struct Usuario u = {"root", 123456};
15     printUsuario(u);
16     return 0;
17 }
```

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  struct Usuario {
5      char login[10];
6      int senha;
7  };
8
9  void scanUsuario(struct Usuario *u) {
10     scanf("%s", u->login);
11     scanf("%d", &u->senha);
12 }
13
14 int main() {
15     struct Usuario u;
16     scanUsuario(&u);
17     return 0;
18 }
```

Exercício 25

Crie uma estrutura representando um Site. Essa estrutura deve conter o domínio do site (ex: sitefalso.com), seu endereço de IP (ex: 161.136.195.15) e o número de acessos por dia. Agora, escreva um programa que leia os dados de 300 sites e os exiba por ordem de número de acessos, do mais acessado para o menos acessado.

1. Baixe o arquivo `website_hits.txt`
2. Execute o comando abaixo no terminal

```
./a.out < website_hits.txt
```

OBS: se você criou o executável do seu programa com outro nome substitua o `a.out` pelo nome do seu executável.

Exercício 26

(a) Crie uma estrutura Peça (struct Peça) contendo os campos: *nome* e *cor*. (b) Depois crie uma matriz 8×8 de ponteiros para o tipo struct Peça chamada **tabuleiro**. (c) inicialize a matriz de ponteiros com NULL. (d) peça ao usuário as posições (linha e coluna) para a inicialização de duas peças e as adicione ao tabuleiro. (e) Crie uma função que recebe o tabuleiro e o imprime para verificar se as peças estão nas posições corretas seguindo a notação na figura abaixo. Diferencie as casas brancas das pretas.

	a	b	c	d	e	f	g	h	
8	a8	b8	c8	d8	e8	f8	g8	h8	8
7	a7	b7	c7	d7	e7	f7	g7	h7	7
6	a6	b6	c6	d6	e6	f6	g6	h6	6
5	a5	b5	c5	d5	e5	f5	g5	h5	5
4	a4	b4	c4	d4	e4	f4	g4	h4	4
3	a3	b3	c3	d3	e3	f3	g3	h3	3
2	a2	b2	c2	d2	e2	f2	g2	h2	2
1	a1	b1	c1	d1	e1	f1	g1	h1	1
	a	b	c	d	e	f	g	h	