

2º Semestre

Técnicas de Programação

Luiz Fernando Carvalho

luizfcarvalhoo@gmail.com

Vetor de Struct

- Cadastrando informações de 4 pessoas

```
1  #include<stdlib.h>
2  #include<stdio.h>
3
4  int main()
5  {
6      char nome1[30], nome2[30], nome3[30], nome4[30];
7      int idade1, idade2, idade3, idade4;
8      char rua1[30], rua2[30], rua3[30], rua4[30];
9      int numero1, numero2, numero3, numero4;
10
11
12      return 0;
13  }
```

Método trabalhoso e confuso. Sem **struct**.

```
1  #include<stdlib.h>
2  #include<stdio.h>
3
4  typedef struct{
5      char nome[30];
6      int idade;
7      char rua[30];
8      int numero;
9  } Pessoa;
10
11  int main()
12  {
13      Pessoa p1, p2, p3, p4;
14
15
16      Pessoa p[4];
17
18      return 0;
19  }
```

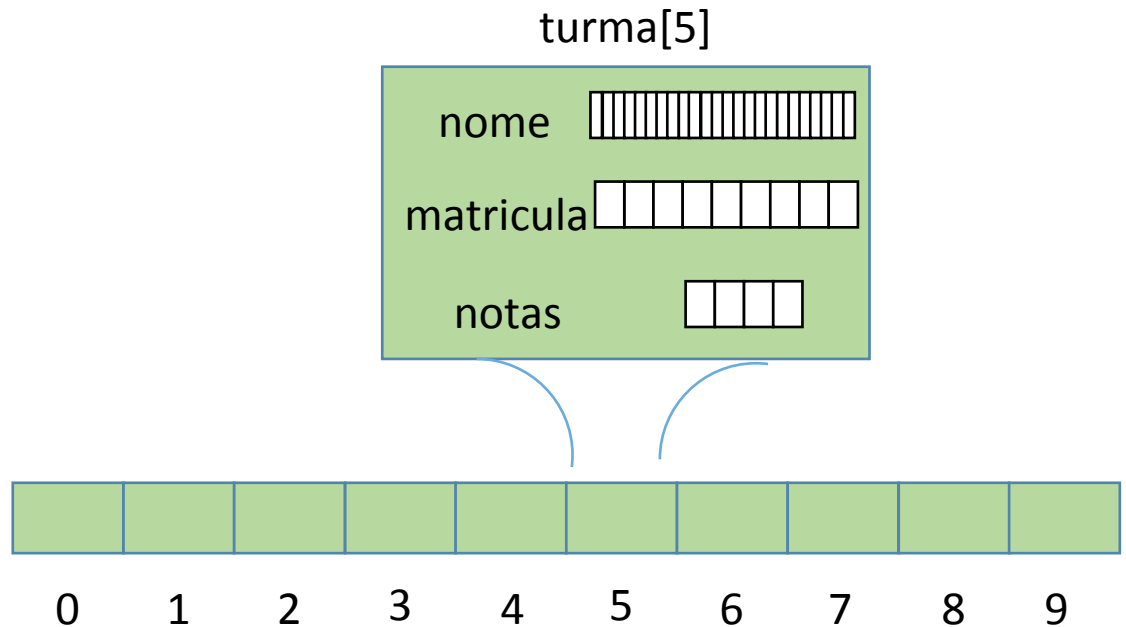
● Usando **struct**

● Usando um vetor de **struct**

Vetor de Struct

- Vetores são usados para guardar diversas variáveis do mesmo tipo (*float, int, char, etc.*)
- Porque não criar um vetor de **structs**?

```
typedef struct{  
    char nome[30];  
    char matricula[10];  
    float notas[4];  
}Aluno;  
  
int main()  
{  
    Aluno turma[10];  
    ...  
}
```



Vetor de Struct

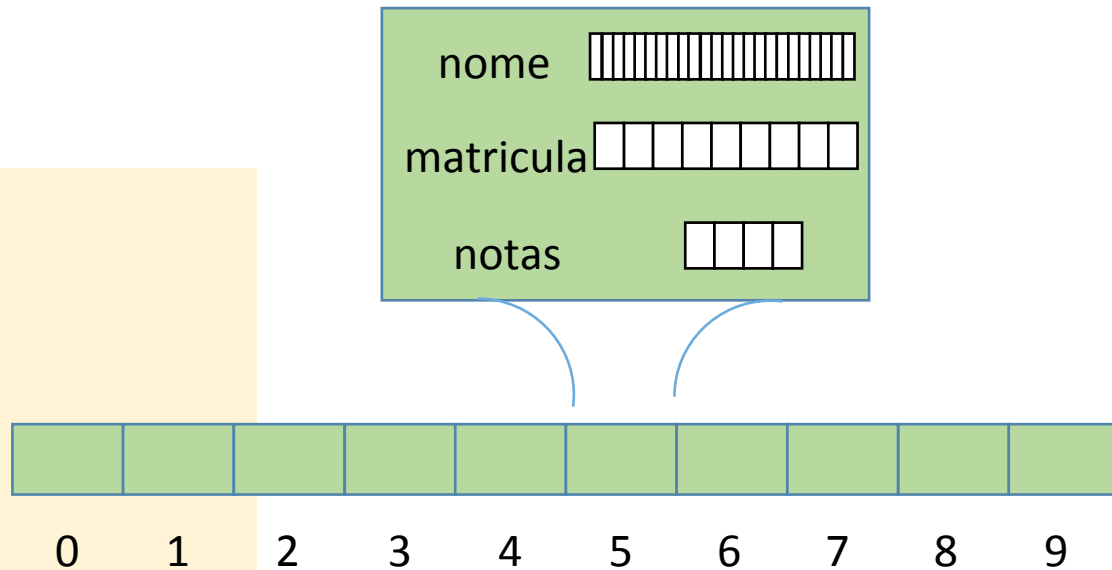
```
typedef struct{
    char nome[30];
    char matricula[10];
    float notas[4];
}Aluno;

int main()
{
    Aluno turma[10];

    strcpy(turma[5].nome, "Joao");
    strcpy(turma[5].matricula, "123456789");
    turma[5].notas[0] = 8.9;
    turma[5].notas[1] = 7.5;
    turma[5].notas[2] = 6.0;
    turma[5].notas[3] = 2.5;

    ...

}
```



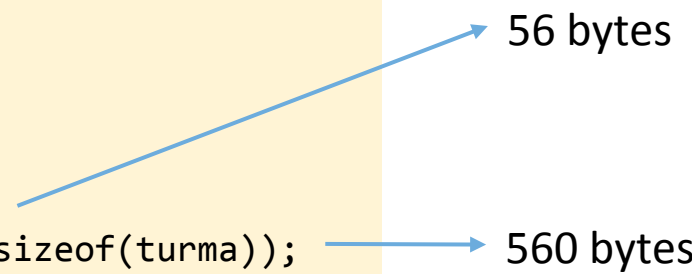
Vetor de Struct

```
typedef struct{
    char nome[30];
    char matricula[10];
    float notas[4];
}Aluno;

int main()
{
    Aluno turma[10];
    Aluno a;

    strcpy(turma[5].nome, "Joao");
    strcpy(turma[5].matricula, "123456789");
    turma[5].notas[0] = 8.9;
    turma[5].notas[1] = 7.5;
    turma[5].notas[2] = 6.0;
    turma[5].notas[3] = 2.5;

    printf("struct tem %d bytes", sizeof(a));
    printf("o vetor de struct tem %d bytes", sizeof(turma));
}
```



The diagram consists of two blue arrows pointing from the code to text labels on the right. The first arrow originates from the `sizeof(a)` argument in the first `printf` statement and points to the text "56 bytes". The second arrow originates from the `sizeof(turma)` argument in the second `printf` statement and points to the text "560 bytes".

Passando *struct* para funções

- É possível passar para funções:
 - variáveis membros da *struct*;
 - A variável *struct* como um todo.
- **Passagem por valor:** Uma cópia da variável é passada para a função;
- **Passagem por referência:** O endereço da variável é passado para a função.
- Quais as características de cada abordagem? Qual é melhor?

Passando *struct* para funções

- É possível passar variáveis membros de *struct* para funções

```
typedef struct{  
    int x, y, z;  
}Ponto;  
  
void imprime(int v){  
    printf("Valor: %d", v);  
}  
  
int main()  
{  
    Ponto p = {1, 2, 3};  
    imprime(p.x);  
    ...  
}
```

Tem que ser do mesmo tipo!

Passagem por valor

Passando *struct* para funções

- É possível passar variáveis membros de *struct* para funções

```
typedef struct{
    int x, y, z;
}Ponto;

void incrementa_imprime(int *v){
    *v = *v + 1;
    printf("Valor: %d", *v);
}

int main()
{
    Ponto p;

    imprime(&p.y);
    ...
}
```

O operador & precede o nome da estrutura, não o nome da variável membro!

Passagem por referência

Passando *struct* para funções

- É possível passar variáveis membros de *struct* para funções

```
typedef struct{
    char nome[30];
    char matricula[10];
    float notas[4];
}Aluno;

void imprimeNotas(float n[]){
    int i;
    for(i=0;i<4;i++)
        printf(" %f ", n[i]);
}

int main()
{
    Aluno a;
    imprimeNotas(a.notas);
}
```

Passa o vetor notas do Aluno a como parâmetro para a função imprimeNotas

lembre-se que um vetor sempre é passado por referência

Passando *struct* para funções

- Quando uma *struct* inteira é passada por parâmetros, é usado a **passagem por valor**;
- **IMPORTANTE:** Em alguns compiladores antigos, estruturas inteiras não podiam ser passadas como parâmetros
 - Eram tratadas como matrizes e apenas um ponteiro para a estrutura era passado
- Lembre-se que o tipo do parâmetro deve coincidir com o tipo da *struct*.

```
1  typedef struct{
2      int x, y, z;
3  }Ponto;
4
5  void funcao(Ponto v){
6      ...
7  }
8
9  int main(){
10     Ponto p;
11     funcao(p);
```

Passando *struct* para funções

- É possível passar *struct* inteiras para funções

```
1  typedef struct{
2      char nome[30];
3      char matricula[10];
4      float notas[4];
5  }Aluno;
6
7  void imprimeAluno(Aluno a){
8      int i;
9
10     puts(a.nome);
11     puts(a.matricula);
12
13     for(i=0;i<4;i++)
14         printf(" %f ", a.notas[i]);
15 }
```

```
16  int main(){
17      Aluno a;
18      int i;
19
20      scanf("%s", a.nome);
21      scanf("%s", a.matricula);
22
23      for(i=0;i<4;i++)
24          scanf("%f", &a.notas[i]);
25
26      imprimeAluno(a);
27
28      return 0;
29 }
```

Passando *struct* para funções

- É possível passar *struct* inteiras para funções
 - Existe um prejuízo em passar a *struct* inteira, exceto as mais simples
 - Tempo necessário para copiar e enviar as variáveis membros para as funções;
 - Em *structs* simples, esse tempo extra não é tão grande;
 - Se existem várias variáveis membros ou se algumas delas são vetores e matrizes, a performance pode ser comprometida.
 - A solução é passar apenas **um ponteiro** para uma função

Passando *struct* para funções

- Quando um ponteiro para uma estrutura é passado para uma função, apenas o endereço é necessário;
- Isso contribui para chamadas muito mais rápidas para funções;
- Além disso, passando um ponteiro, é possível alterar o conteúdo das variáveis membros diretamente.

Passando *struct* para funções

- É possível passar para função um ponteiro para *struct*

```
typedef struct{
    int x, y, z;
}Ponto;

void altera(Ponto *v){
    (*v).x = (*v).x + 1;
    (*v).y = (*v).y + 1;
    (*v).z = (*v).z + 1;
}

int main()
{
    Ponto p = {1, 2, 3};

    altera(&p);
    ...
}
```

Equivalentes



```
typedef struct{
    int x, y, z;
}Ponto;

void altera(Ponto *v){
    v->x = v->x + 1;
    v->y = v->y + 1;
    v->z = v->z + 1;
}

int main()
{
    Ponto p = {1, 2, 3};

    altera(&p);
    ...
}
```

Passando *struct* para funções

- É possível passar para função um ponteiro para *struct*
 - O operador -> é chamado de seta;
 - A seta é usada no lugar do operador ponto quando se está acessando um membro da estrutura por meio de um ponteiro para a estrutura

```
1 typedef struct{
2     int x, y;
3 }Ponto;
4
5 void altera(Ponto *v){
6     v->x = 2;
7     v->y = 8;
8 }
9
10 int main()
11 {
12     Ponto p = {1, 5};
13     altera(&p);
14     printf("x = %d, y = %d", p.x, p.y);
15 }
```

Lembre-se de usar o operador **ponto** para acessar elementos de uma *struct* quando estiver operando na própria *struct* (linha 14). Quando tiver um ponteiro para a estrutura, use o operador **seta** (linhas 6 e 7).

Exemplo

```
1  #include<stdio.h>
2  #include<unistd.h>
3  #define DELAY 999999
4
5  typedef struct{
6      int horas;
7      int minutos;
8      int segundos;
9  }My_time;
10
11 void atualiza(My_time *t){
12     t->segundos++;
13     if(t->segundos == 60){
14         t->segundos = 0;
15         t->minutos++;
16     }
17     if(t->minutos == 60){
18         t->minutos = 0;
19         t->horas++;
20     }
21     if(t->horas == 24)
22         t->horas = 0;
23     delay();
24 }
```

```
25 void imprime(My_time *t){
26     printf("%02d:", t->horas);
27     printf("%02d:", t->minutos);
28     printf("%02d\n", t->segundos);
29 }
30
31 void delay(){
32     usleep(DELAY);
33 }
34
35 int main(){
36     My_time tempo;
37
38     tempo.horas = 0;
39     tempo.minutos = 0;
40     tempo.segundos = 0;
41
42     for(;;){
43         atualiza(&tempo);
44         imprime(&tempo);
45     }
46
47     return 0;
48 }
```


Passando *struct* para funções

- É possível passar um vetor de *struct* para funções

```
1 typedef struct{
2     char nome[30];
3     char matricula[10];
4     float notas[4];
5 }Aluno;
6
7 void leAlunos(Aluno v[]){
8     int i, j;
9
10    for(i=0;i<5;i++){
11        scanf("%s", v[i].nome);
12        scanf("%s", v[i].matricula);
13        for(j=0;j<4;j++)
14            scanf("%f", &v[i].notas[j]);
15    }
16 }
```

```
17 int main(){
18     Aluno a[5];
19
20     leAlunos(a);
21
22     return 0;
23 }
```

Não se esqueça que vetores são sempre passados como referência!!!

Passando *struct* para funções

- É possível passar um vetor de *struct* para funções

```
1 typedef struct{
2     int x, y;
3 }Ponto;
4
5 int somaVetor(Ponto v[]){
6     int i, soma=0;
7
8     for(i=0;i<5;i++){
9         soma += v[i].x;
10        soma += v[i].y;
11    }
12
13    return soma;
14 }
```

```
15 int main(){
16     Ponto p[5] =
17     {{1,2}, {3,4}, {5,6}, {7,8}, {9,10}};
18
19     int resultado;
20
21     resultado = somaVetor(p);
22     printf("A soma e': %d", resultado);
23
24     return 0;
25 }
```

Não se esqueça que vetores
são sempre passados como
referência!!!

Retornando *struct* de funções

- É possível retornar uma *struct* a partir de uma função

```
1 typedef struct{
2     int hora, minuto, segundo;
3 }Horario;
4
5 Horario iniciaHorario(){
6     Horario h;
7
8     h.hora = 0;
9     h.minuto = 0;
10    h.segundo = 0;
11
12    return h;
13 }
```

```
int main(){
    Horario agora;

    agora = iniciaHorario();
    ...
}
```

A variável *agora* tem que ser do tipo *Horario*

Agora não ficamos restritos a retornar apenas tipos primitivos da linguagem C (*int*, *float*, *char*...)

Retornando *struct* de funções: Exemplo

```
1 typedef struct{
2     char modelo[20], placa[8];
3     int ano;
4 }Carro;
5
6 Carro iniciaCarro(char *m, char *p, int a){
7     Carro c;
8
9     strcpy(c.modelo, m);
10    strcpy(c.placa, p);
11    c.ano = a;
12
13    return c;
14 }
```

```
int main(){
    Carro novo_carro;

    novo_carro = iniciaCarro("Ferrari", "abc1234", 2018);
    ...
}
```

Estrutura Aninhadas

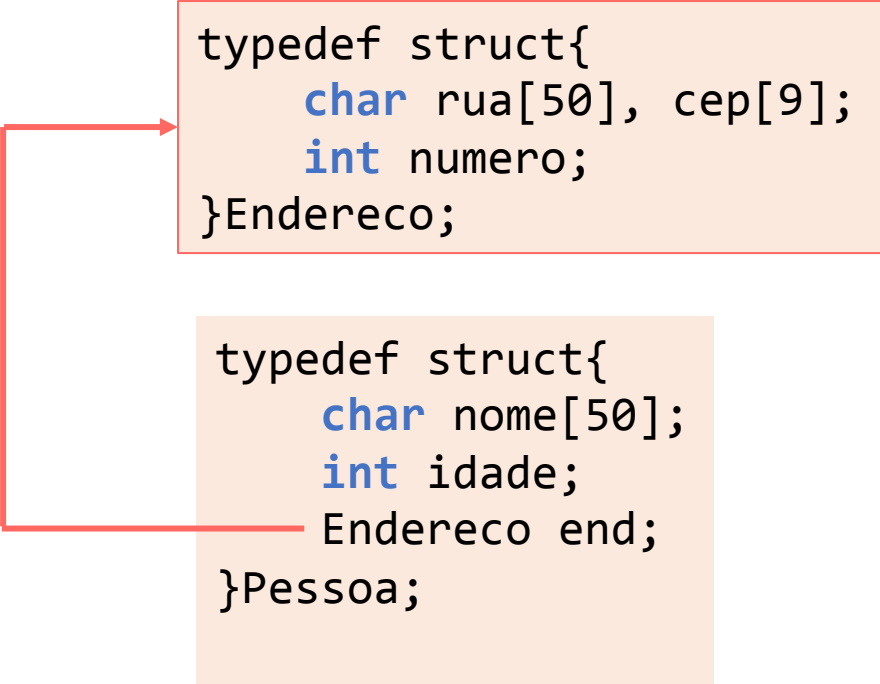
- Assim como estruturas de controle `while`, `for` e `if` podem ser aninhadas, as estruturas também podem:

```
typedef struct{  
    char nome[50], rua[50], cep[9];  
    int idade, numero;  
}Pessoa;
```

- As variáveis `nome` e `idade` estão relacionadas à pessoa;
- As variáveis `rua`, `cep` e `numero` estão relacionadas ao endereço da pessoa.

Estrutura Aninhadas

- Então, pode-se dividir a estrutura em duas:



```
typedef struct{  
    char rua[50], cep[9];  
    int numero;  
}Endereco;
```

```
typedef struct{  
    char nome[50];  
    int idade;  
    Endereco end;  
}Pessoa;
```

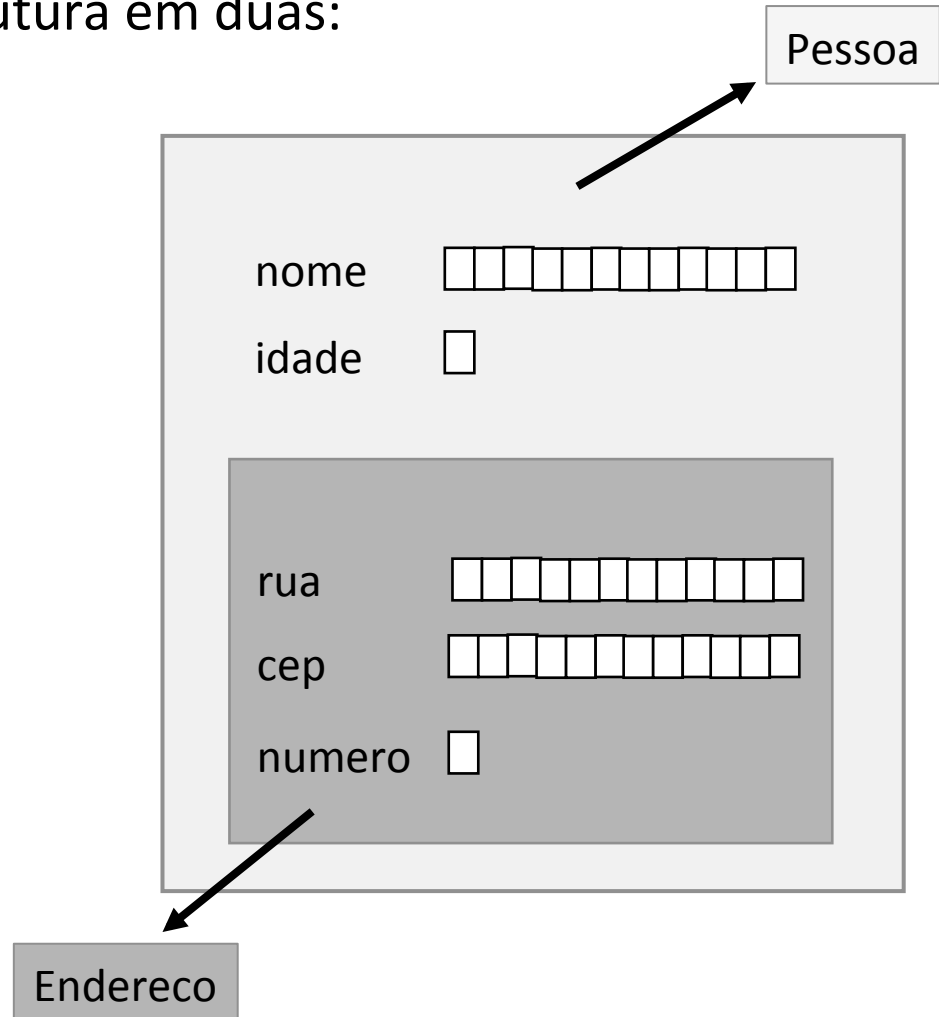
Qual a vantagem de se fazer essa divisão?

Estrutura Aninhadas

- Então, pode-se dividir a estrutura em duas:

```
typedef struct{  
    char rua[50], cep[9];  
    int numero;  
}Endereco;
```

```
typedef struct{  
    char nome[50];  
    int idade;  
    Endereco end;  
}Pessoa;
```



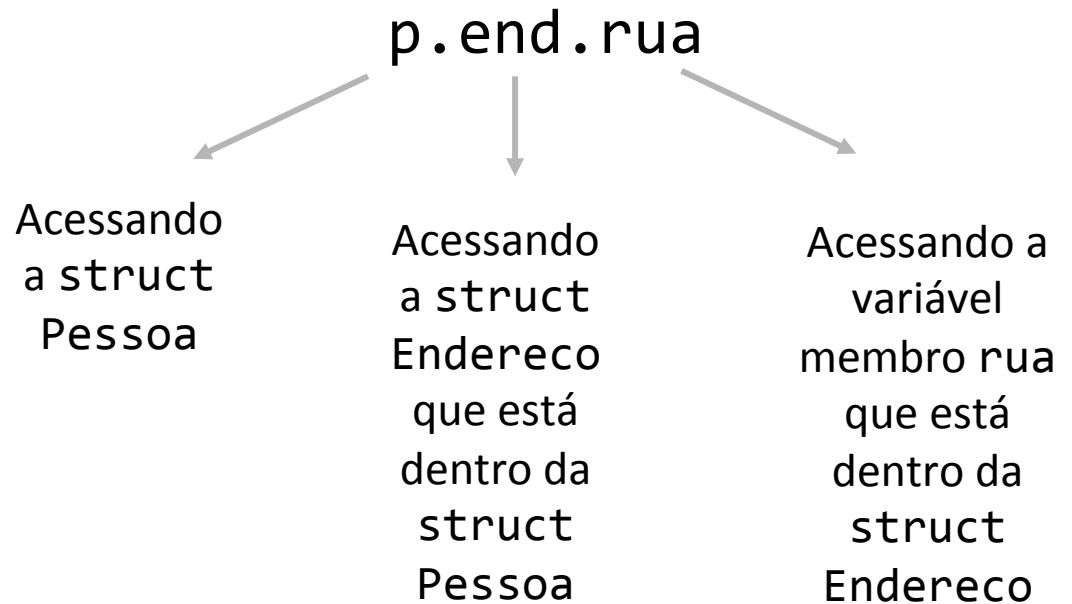
Estrutura Aninhadas

```
typedef struct{  
    char rua[50], cep[9];  
    int numero;  
}Endereco;
```

```
typedef struct{  
    char nome[50];  
    int idade;  
    Endereco end;  
}Pessoa;
```

```
int main(){  
    Pessoa p;  
    ...
```

Para acessar as variáveis rua, cep e numero por meio da struct Pessoa:



Estrutura Aninhadas

```
typedef struct{
    char rua[50], cep[9];
    int numero;
}Endereco;

typedef struct{
    char nome[50];
    int idade;
    Endereco end;
}Pessoa;
```

```
int main(){
    Pessoa p;

    strcpy(p.nome, "Cardoso");
    p.idade = 45;
    strcpy(p.end.cep, "123456789");
    p.end.numero = 4;
    strcpy(p.end.rua, "Rua Alan Turing");

    puts(p.nome);
    printf("%d", p.idade);
    puts(p.end.cep);
    puts(p.end.rua);
    printf("%d", p.end.numero);

    return 0;
}
```

Estrutura Aninhadas

```
typedef struct{  
    int rodas, capacidade;  
    char marca[20], modelo[20];  
    char combustivel[10];  
    float peso;  
}Veiculo;  
  
typedef struct{  
    int cilindradas;  
    Veiculo v;  
}Motocicleta;  
  
typedef struct{  
    int motores;  
    float alturaMaxima;  
    Veiculo v;  
}Aviao;
```

int rodas, capacidade
char marca, modelo, combustivel
float peso

int motores
float alturaMaxima



Exercícios

1. Construa uma estrutura aluno com nome, numero de matrícula e curso. Leia do usuário a informação de 5 alunos, armazene em vetor dessa estrutura e imprima os dados na tela.
2. Ordene o vetor do exercício anterior de acordo com o nome do aluno. Ordene em ordem alfabética.

Exercícios

3. Crie uma estrutura representando os alunos de um determinado curso. A estrutura deve conter a matrícula do aluno, nome, nota da primeira prova, da segunda, da terceira e da quarta prova.
 - Permita ao usuário entrar com os dados de 5 alunos.
 - Encontre o aluno com maior nota da primeira prova.
 - Encontre o aluno com maior media geral.
 - Encontre o aluno com menor media geral.
 - Para cada aluno diga se ele foi aprovado ou reprovado, considerando o valor 6 para aprovação.

Exercícios

4. Defina uma estrutura que irá representar bandas de música. Essa estrutura deve ter o nome da banda, que tipo de música ela toca, o número de integrantes e em que posição do ranking essa banda está dentre as suas 5 bandas favoritas;
5. Crie uma função para preencher as 5 estruturas de bandas criadas no exemplo passado. Após criar e preencher, exiba todas as informações das bandas/estruturas. Não se esqueça de usar o operador -> para preencher os membros das structs;
6. Crie uma função que peça ao usuário um número de 1 até 5. Em seguida, seu programa deve exibir informações da banda cuja posição no seu ranking é a que foi solicitada pelo usuário;

Exercícios

7. Crie uma função em C que peça ao usuário um tipo de música e exiba as bandas com esse tipo de música no seu ranking. Que função da `string.h` você usaria para comparar as *strings* que representam o tipo de banda?
8. Crie uma função que peça o nome de uma banda ao usuário e diga se ela está entre suas bandas favoritas ou não;
9. Agora junte tudo e crie uma aplicação que exibe um menu com as opções de preencher as estruturas e todas as opções das questões passadas.

Exercícios

10. Faça um programa que seja uma agenda de compromissos e:

- Crie e leia um vetor de 5 estruturas de dados com: compromisso (máximo 60 letras) e data. A data deve ser outra estrutura de dados contendo dia, mês e ano.
- Leia dois inteiros M e A e mostre todos os compromissos do mês M do ano A . Repita o procedimento ate ler $M = 0$.

12. Faça um programa para resolver equações do segundo grau que deverá tratar os casos particulares.

```
typedef struct{  
    int retorno;  
    float x1, x2;  
} Raizes;
```

- Devolve -1 se delta < 0 e informe que não existe raízes reais;
- Devolve 0 se a = 0 e informe que não é uma equação do 2º grau;
- Devolve 1 se delta=0, raízes iguais
- Devolve 2 se delta>0, raízes diferentes

Exercícios

12. Baseado em um baralho tradicional (cada carta tem um naipe e um valor), implemente a parte de distribuição (sorteio) de cartas para 2 jogadores. Considere que cada jogador irá receber 5 cartas. Exiba na tela as cartas que cada jogador recebeu;