

# *Tecnologia em Análise e Desenvolvimento de Sistemas - TADS*

## ***Estrutura de Dados***

***Prof. Luciano Vargas Gonçalves***

*E-mail: [luciano.goncalves@riogrande.ifrs.edu.br](mailto:luciano.goncalves@riogrande.ifrs.edu.br)*



# Estrutura da Dados

- **Aula 3 – Struct - Programação em C**

# Structs

- Estrutura de Dados em C
  - Uma estrutura é ***um grupo ou conjunto de itens***, no qual cada ***item*** é identificado por um identificador próprio (tipo e nome), sendo cada um deles conhecido como um **membro** da estrutura.
    - Os vários MEMBROS formam um ***Struct***;
      - Uma ***Struct*** pode ser considerada um novo tipo de dados composto.
    - Estrutura é uma versão resumida de uma classe de OO, cada membro pode ser interpretado como um atributo de classe.

# Structs

- Estrutura de Dados em C

- Usa o comando **Struct** na sua definição;
- Necessita definir um nome para estrutura;
- Membros possuem um tipo e nome (semelhante a variáveis);
- Usa-se o comando de bloco de comandos { };

```
struct nome_da_estrutura{  
    tipo nome_parametro;  
    tipo nome_parametro;  
};
```

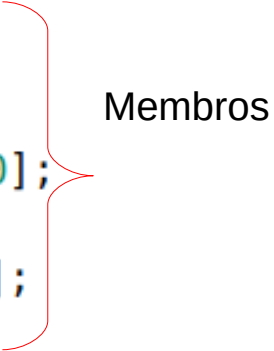
Membros da Struct

Definição

# Structs

- Estrutura de Dados em C
  - Exemplo de uma estrutura para armazenar os dados de uma pessoa – tipo Pessoa;

```
✓ struct pessoa{  
    int cod;  
    char nome [15];  
    char sobrenome [20];  
    int idade;  
    char telefone [10];  
};
```



Membros

Exemplo *Struct* Pessoa

# Structs – Declaração e Atribuição

- **Struct** pode ser vista como um novo tipo de dados definido pelo programador.
  - Um novo tipo de dados que se declara e/ou atribuir valores aos membros.
    - *Forma direta de atribuir informação, uso de chaves { }*

```
//DECLARACAO E ATRIBUICAO
```

```
struct pessoa maria = {2, "Maria", "Aparecida", 23, "45433333"};
```

Novo tipo de Dados



Nome atribuído a estrutura

Valores para os membros

Struct pessoa;

```
struct pessoa{  
    int cod;  
    char nome [15];  
    char sobrenome [20];  
    int idade;  
    char telefone [10];  
};
```

# Structs – Declaração e Atribuição

- Declaração separada da atribuição dos valores aos membros.
  - Usa o operador “.” para acessar os membros;

```
struct pessoa joao; //DECLARAÇÃO DE STRUCT
```

```
//ATRIBUIÇÃO DE VALORES
```

```
joao.cod = 1;  
joao.idade = 30;  
strcpy(joao.nome, "Joao Carlos");  
strcpy(joao.sobrenome, "Farias");  
strcpy(joao.telefone, "1212454533");
```

```
struct pessoa{  
    int cod;  
    char nome [15];  
    char sobrenome [20];  
    int idade;  
    char telefone [10];  
};
```

Struct pessoa;  
Tipo especial

struct.membro para atribuir valores;

STRCPY (comando para copiar uma String para uma variável ou membro)

# Structs

- Estrutura de Dados em C
  - Consultar valores de membros, **operador PONTO “.”**

```
//LEITURA DE UM STRUCT
```

```
printf("Pessoa: %s %s \n",joao.nome,joao.sobrenome);  
printf("\tCodigo: %d e idade %d \n",joao.cod,joao.idade);  
printf("\tTelefone: %s \n",joao.telefone);  
  
printf("Pessoa: %s %s \n",maria.nome,maria.sobrenome);  
printf("\tCodigo: %d e idade %d \n",maria.cod,maria.idade);  
printf("\tTelefone: %s \n",maria.telefone);
```

```
struct pessoa{  
    int cod;  
    char nome [15];  
    char sobrenome [20];  
    int idade;  
    char telefone [10];  
};
```

Struct pessoa



Operador “PONTO” para acessar um membro de uma *Struct* **ESTÁTICA**



# Exemplo1.c

- Exemplo completo
  - Estrutura Pessoa

Pessoa: Joao Carlos Farias  
Codigo: 1 e idade 30  
Telefone: 1212454533

Pessoa: Maria Aparecida  
Codigo: 2 e idade 23  
Telefone: 45433333

Pessoa: Maria Aparecida  
Codigo: 2 e idade 45  
Telefone: 222333

Saída no Terminal

```
1  #include <stdio.h>
2  struct pessoa{
3      int cod;
4      char nome [15];
5      char sobrenome [20];
6      int idade;
7      char telefone [10];
8  };
9  int main() {
10     //DECLARACAO DE OUTRA PESSOA E ATRIBUICAO
11     struct pessoa maria = {2,"Maria","Aparecida",23,"45433333"};
12     //DECLARAÇÃO DE STRUCT
13     struct pessoa joao;
14     //ATRIBUIÇÃO DE VALORES
15     joao.cod = 1;
16     joao.idade = 30;
17     strcpy(joao.nome,"Joao Carlos");
18     strcpy(joao.sobrenome, "Farias");
19     strcpy(joao.telefone, "1212454533");
20     //SAIDA DA STRUCT JOAO
21     printf("Pessoa: %s %s \n",joao.nome,joao.sobrenome);
22     printf("\tCodigo: %d e idade %d \n",joao.cod,joao.idade);
23     printf("\tTelefone: %s \n\n",joao.telefone);
24     //SAÍDA DA STRUCT MARIA
25     printf("Pessoa: %s %s \n",maria.nome,maria.sobrenome);
26     printf("\tCodigo: %d e idade %d \n",maria.cod,maria.idade);
27     printf("\tTelefone: %s \n\n",maria.telefone);
28     //PARA ALTERAR O VALOR DE UM MEMBRO
29     maria.idade = 45;
30     strcpy(maria.telefone ,"2222333");
31     //SAÍDA NO TERMINA - PRINT
32     printf("Pessoa: %s %s \n",maria.nome,maria.sobrenome);
33     printf("\tCodigo: %d e idade %d \n",maria.cod,maria.idade);
34     printf("\tTelefone: %s \n\n",maria.telefone);
35     return 0; }
```

# Exemplo1.c

- Exemplo completo
  - Estrutura Pessoa
  - Troca informações de Maria (idade e telefone)

```
//Para alterar o valor de um Membro
    maria.idade = 45;
    strcpy(maria.telefone , "2222333");

//SAÍDA NO TERMINA - PRINT
    printf("Pessoa: %s %s \n",maria.nome,maria.sobrenome);
    printf("\tCodigo: %d e idade %d \n",maria.cod,maria.idade);
    printf("\tTelefone: %s \n\n",maria.telefone);
    return 0;
```

Pessoa: Maria Aparecida  
Codigo: 2 e idade 23  
Telefone: 45433333



Pessoa: Maria Aparecida  
Codigo: 2 e idade 45  
Telefone: 2222333



# Comando TypeDef

- Fornece um mecanismo para criação de sinônimos para novos tipos de dados;
  - Elimina o uso do **comando Struct** na definição do tipo;

```
typedef struct nome_struct nome_tipo (apelido);
```

```
struct pessoa{  
    int cod;  
    char nome [15];  
    char sobrenome [20];  
    int idade;  
    char telefone [10];  
};
```

```
//REDEFINIÇÃO PARA PESSOA  
typedef struct pessoa Pessoa;
```

# Comando Typedef

- Exemplo, **Struct pessoa** passa se chamar de **Pessoa**;
- **Pessoa** é o novo tipo de dados para armazenar dados;  
Sem Typedef é obrigatório o uso do comando STRUCT na declaração do tipo pessoa


```
//DECLARAÇÃO DE OUTRA PESSOA E ATRIBUIÇÃO  
| struct pessoa maria = {2,"Maria","Aparecida",23,"45433333"};  
//DECLARAÇÃO DE STRUCT  
| struct pessoa joao;
```

Com Typedef não necessita o STRUCT na declaração do tipo pessoa

```
//REDEFINIÇÃO PARA PESSOA  
typedef struct pessoa Pessoa;  
//DECLARAÇÃO DO NOVO TIPO Pessoa  
Pessoa joao;  
Pessoa maria = {2,"Maria","Aparecida",23,"45433333"};
```

# Comando Typedef

- Exemplo ***typedef Struct pessoa*** igual a ***Pessoa***



```
typedef struct pessoa{  
    int cod;  
    char nome [10];  
    char sobrenome [20];  
    int idade;  
    char telefone [15];  
    Data nascimento;  
}Pessoa;
```

# Comando TypeDef

- Exemplo ***Struct pessoa*** igual a ***Pessoa***

```
//Declaração de uma variável Pessoa
```

```
Pessoa joao;
```

```
//Declaração e atribuição de valores a uma Pessoa
```

```
Pessoa maria = {2, "Maria", "Aparecida", 0, "53323244411", Hoje};
```

# Ponteiro de Struct

- **Podemos criar um ponteiro para acessar uma Struct**
  - **Struct** é semelhante aos tipos primitivos, podemos definir ponteiros para nos referenciarmos a estas estruturas;
  - Exemplo:
    - Estrutura Pessoa e Ponteiro de Pessoa;

```
//Declaração do Ponteiro
```

```
Pessoa maria, *ps_maria;
```

–

```
ps_maria = &maria;
```

# Ponteiro de Struct

- Utiliza-se o operador “->” seta para referenciar o membro do ponteiro;

```
ps_maria->cod = 1;
ps_maria->idade = 30;
strcpy(ps_maria->nome, "Maria da Graça");
strcpy(ps_maria->sobrenome, " Farias");
strcpy(ps_maria->telefone, "1212454533");

printf("\nPessoa:\n\tNome: %s %s \n", ps_maria->nome, ps_maria->sobrenome);
printf("\tCodigo: %d e idade: %d \n", ps_maria->cod, ps_maria->idade);
printf("\tTelefone: %s \n", ps_maria->telefone);
```



Membro referenciado por um ponteiro usa seta “->”



# Ponteiro de Struct

- Diferença entre Variável MARIA e o Ponteiro \*PS\_MARIA

```
Pessoa maria, *ps_maria;  
ps_maria = &maria;
```

```
printf("\nPessoa:\n\tNome: %s %s \n",ps_maria->nome,ps_maria->sobrenome);  
printf("\tCodigo: %d e idade: %d \n",ps_maria->cod,ps_maria->idade);  
printf("\tTelefone: %s \n",ps_maria->telefone);
```

---

```
printf("\nPessoa:\n\tNome: %s %s \n",maria.nome,maria.sobrenome);  
printf("\tCodigo: %d e idade: %d \n",maria.cod,maria.idade);  
printf("\tTelefone: %s \n",maria.telefone);
```

# Funções com Structs

- Passagem por Valor
  - A cópia de uma Pessoa é armazenada em uma variável **ps**
    - Mostra os dados de uma pessoa recebida por parâmetro;

```
void mostrarDadosPessoa(Pessoa ps){
```



- Passagem por Referência
  - O endereço de uma Pessoa é enviado para o ponteiro **\*ps**
    - *Função preenche os campos de uma Pessoa recebida por referência*

```
void lerDadosPessoa(Pessoa *ps, int cod){
```



# Funções com Structs

- Passagem por Valor, função ***mostraDadosPessoa (Pessoa ps);***

```
void mostrarDadosPessoa(Pessoa ps){  
    printf("\nPessoa:\n\t Nome: %s  %s \n",ps.nome,ps.sobrenome);  
    printf("\tCodigo: %d  e idade %d \n",ps.cod,ps.idade);  
    printf("\tTelefone: %s \n",ps.telefone);  
}
```

- Chamada da função

```
mostrarDadosPessoa(maria);
```

# Funções com Structs

- Passagem por Referência, função ***lerDadosCliente(Pessoa \*ps);***

```
void lerDadosPessoa(Pessoa *ps, int cod){  
    ps->cod = cod;  
    printf("\nInforme seu nome:");  
    scanf("%s",ps->nome);  
    printf("\nInforme seu Sobrenome:");  
    scanf("%s",ps->sobrenome);  
    printf("\nInforme sua idade:");  
    scanf("%d",&ps->idade);  
    printf("\nInforme seu telefone:");  
    scanf("%s",ps->telefone);  
}
```

- ***Chamada da Função***

```
lerDadosPessoa(&maria,codigo++);
```

# Exemplo2.c

- Chamada de funções
  - Passagem por **valor**
  - Passagem por **referência**

```
> typedef struct pessoa{ ...
    }Pessoa;

//PASSAGEM POR VALOR
> void mostrarDadosPessoa(Pessoa ps){ ...

//PASSAGEM POR REFERENCIA (ponteiro)
> void lerDadosPessoa(Pessoa *ps, int cod){ ...

int main(){
    system("clear");
    setlocale(LC_ALL, "");    //trabalhar com acentuação pt-br
    int codigo = 1;
    //declaração e alocação estática
    Pessoa joao, maria;

    //chamada das funções com Struct
    lerDadosPessoa(&joao,codigo++);    //passagem por referência
    mostrarDadosPessoa(joao);    //passagem por valor;

    lerDadosPessoa(&maria,codigo++);
    mostrarDadosPessoa(maria);
    exit(0);
}
```

# Alocação dinâmica de Struct

- **Alocação dinâmica – MALLOC**

- MALLOC = Operação de alocar memória para armazenamento de dados - Struct;
  - Retorna o endereço da memória alocada;
  - Sizeof → define o tamanho necessário a ser alocado.

```
//declaração de um ponteiro de Pessoa
```

```
Pessoa *paulo;
```

```
//inicialização do ponteiro;
```

```
paulo = NULL;
```

```
//Alocação dinâmica de memória para um STRUCT
```

```
paulo = (Pessoa*)malloc(sizeof(Pessoa));
```

—  
Cast (tipar)

Alocação de Memória

Endereço	Valor	Nome
xxxx01	xxxx03	*Paulo
xxxx02		
xxxx03	Nova Pessoa alocada	
Xxxx0..		
xxxx09		
xxxx06		

# Structs com ponteiros

- **Estrutura dados usando ponteiro \*.**

- Declarar o ponteiro para Cliente “\*paulo”;

```
//declaração de um ponteiro de Pessoa
```

```
Pessoa *paulo;
```

```
//inicialização do ponteiro;
```

```
paulo = NULL;
```

```
//Alocação dinâmica de memória para um STRUCT
```

```
paulo = (Pessoa*)malloc(sizeof(Pessoa));
```

Endereço	Valor	Nome
xxxx01	NULL	*paulo
xxxx02		
xxxx03	Nova Pessoa alocada	
Xxxx0..		
xxxx09		
xxxx06		

# Structs com ponteiros

- **Estrutura dados usando ponteiro \*.**

- Declarar o ponteiro para Cliente “\*paulo”;

```
//declaração de um ponteiro de Pessoa
```

```
Pessoa *paulo;
```

```
//inicialização do ponteiro;
```

```
paulo = NULL;
```

```
//Alocação dinâmica de memória para um STRUCT
```

```
paulo = (Pessoa*)malloc(sizeof(Pessoa));
```

Endereço	Valor	Nome
xxxx01	xxxx03	*paulo
xxxx02		
xxxx03	Nova Pessoa alocada	
Xxxx0..		
xxxx09		
xxxx06		



# MALLOC vs FREE

- Alocação com MALLOC
  - Reserva memória e retorna um ponteiro;
- ***Liberção de memória com FREE;***
  - Libera memória para realocação;

```
//declaração de um ponteiro de Pessoa
Pessoa *paulo;
//inicialização do ponteiro;
paulo = NULL;

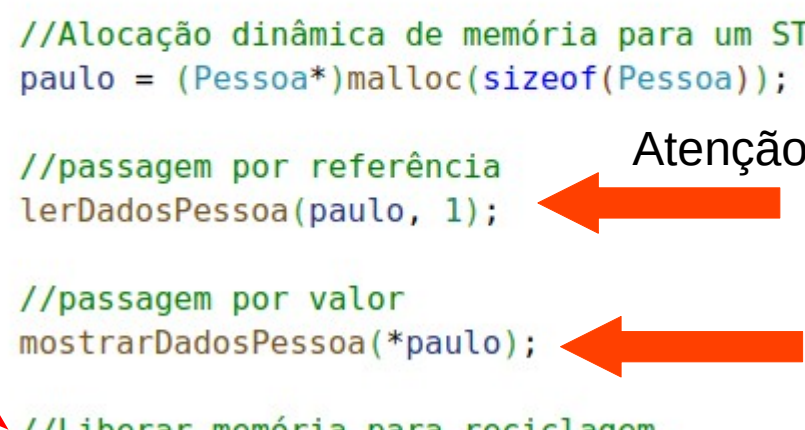
//Alocação dinâmica de memória para um STRUCT
paulo = (Pessoa*)malloc(sizeof(Pessoa));

//passagem por referência
lerDadosPessoa(paulo, 1);

//passagem por valor
mostrarDadosPessoa(*paulo);

//Liberar memória para reciclagem
free(paulo);
```

Atenção



# Structs Aninhadas

- Uma estrutura é composta por outra estrutura;

- Exemplo:

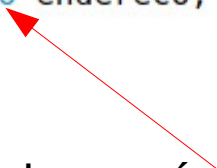
- Pessoa e Endereço
  - Pessoa tem Endereço

**ATENÇÃO:**

**Endereço** precisa ser declarado Antes de **Pessoa**

```
typedef struct endereco{  
    char nomeRua [15];  
    int numero;  
    int cep;  
}Endereco;
```

```
typedef struct pessoa{  
    int cod;  
    char nome [15];  
    Endereco endereco;    //Ligação entre as estruturas  
}Pessoa;
```



Endereço é membro, declarado dentro de pessoa, e é reservado memória para este.

# Pessoa e Endereço

- Associação entre Pessoa e Endereço;
  - Pessoa joao;
  - Endereço minhaCasa;



```
Pessoa joao, maria; //dois Pessoas novos
Endereco minhaCasa;
joao.cod = 1;
strcpy(joao.nome, "João Cesar");
maria.cod = 2;
strcpy(maria.nome, "Maria Cesar");
strcpy(minhaCasa.nomeRua, "Rua 24 Maio");
minhaCasa.numero = 332;
minhaCasa.cep = 96500333;
//associação entre Pessoa e endereco
joao.endereco = minhaCasa;
maria.endereco = minhaCasa;
mostrarDadosPessoa(joao);
mostrarDadosPessoa(maria);
```

# Structs Aninhadas

- Exemplo de Pessoa e Endereço com função

```
void mostrarDadosPessoa(Pessoa ps){  
    printf("\nPessoa: %s Código: %d \n",ps.nome,ps.cod);  
    printf("Endereço: %s Número:%d Cep:%d \n",ps.endereco.nomeRua,ps.endereco.numero,ps.endereco.cep);  
}
```

*Struct.membro.membro*

```
void lerDadosPessoa(Pessoa *ps, int cod){  
    ps->cod = cod;  
    printf("\nInforme seu nome:");  
    scanf("%s",ps->nome);  
    printf("\nInforme sua Rua:");  
    scanf("%s",ps->endereco.nomeRua);  
    printf("\nInforme Numero Casa:");  
    scanf("%d",&ps->endereco.numero);  
    printf("\nInforme seu CEP:");  
    scanf("%d",&ps->endereco.cep);  
}
```

*Ponteiro->membro.membro*

# Exemplo3.c

- Associação Cliente e Endereço

João e Maria residem  
no mesmo endereço

Pessoa: João Cesar Código: 1  
Endereço: Rua 24 Maio Número:332 Cep:96500333

Pessoa: Maria Cesar Código: 2  
Endereço: Rua 24 Maio Número:332 Cep:96500333

```
Pessoa joao, maria; //dois Pessoas novos
Endereco minhaCasa;
joao.cod = 1;
strcpy(joao.nome, "João Cesar");
maria.cod = 2;
strcpy(maria.nome, "Maria Cesar");
strcpy(minhaCasa.nomeRua, "Rua 24 Maio");
minhaCasa.numero = 332;
minhaCasa.cep = 96500333;
//associação entre Pessoa e endereco
joao.endereco = minhaCasa;
maria.endereco = minhaCasa;
mostrarDadosPessoa(joao);
mostrarDadosPessoa(maria);
```

# Structs Aninhadas – Exemplo 2

- **Ponteiro como Membro de Struct**

- Estrutura CASAL
  - Composta de uma data;
  - Dois ponteiros para Pessoa;
    - Marido e Esposa;

Ponteiros para Struct Pessoa



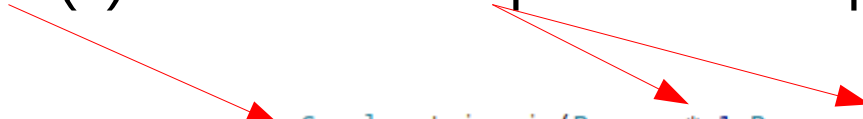
```
typedef struct {  
    char nomeRua [15];  
    int numero;  
    char cep [10];  
}Endereco;
```

```
typedef struct {  
    int cod;  
    int idade;  
    char nome [15];  
    Endereco end;  
}Pessoa;
```

```
typedef struct casal{  
    char data[10];  
    Pessoa *marido; //dois ponteiros para Pessoas  
    Pessoa *esposa;  
}Casal;
```

# Structs Aninhadas – Exemplo 2

- Função matrimônio
  - Retorna um Casal (c) e recebe dois ponteiros de pessoas



```
Casal matrimonio(Pessoa *p1,Pessoa *p2){
    Casal c;
    strcpy(c.data,"02/06/20");
    c.marido = p1;
    c.esposa = p2;
    return c;
}

void imprimeCertidaoCasamento(Casal cs){
    printf("\n\nCertidão de Casamento!\n");
    printf("\n Nada data %s Casaram-se neste cartório ",cs.data);
    printf("\n \t%s e %s",cs.marido->nome,cs.esposa->nome);
    printf("\n\n Dou fé a este Matrimonio!\n");
}
```

# Structs Aninhadas – Exemplo 2

- Passagem por referência
  - Pessoa &joao e &maria
- Retorno por Valor
  - Casal joaoEmaria
- 

```
int main(){
    system("clear");
    setlocale(LC_ALL, "");
    Pessoa joao,maria;
    joao.cod = 1;
    joao.idade = 30;
    strcpy(joao.nome,"Joao Carlos");
    strcpy(joao.endereco.nomeRua, "Duque de caxias");
    joao.endereco.numero = 54;
    strcpy(joao.endereco.cep, "96234-234");
    maria.cod = 2;
    maria.idade = 26;
    strcpy(maria.nome,"Maria Aparecida");
    strcpy(maria.endereco.nomeRua, "Duque de caxias");
    maria.endereco.numero = 45;
    strcpy(maria.endereco.cep, "96234-234");

    //matrimonio é o relacionamento entre duas pessoas
    Casal joaoEmaria = matrimonio(&joao,&maria);
    imprimeCertidaoCasamento(joaoEmaria);
    exit(0);
}
```



# Exemplo3.c

- Exemplo de Casal
  - Com uso da função matrimônio

```
5 typedef struct {
6     char nomeRua [15];
7     int numero;
8     char cep [10];
9 }Endereco;
10 typedef struct {
11     int cod;
12     int idade;
13     char nome [15];
14     Endereco endereco;
15 }Pessoa;
16 typedef struct casal{
17     char data[10];
18     Pessoa *marido; //dois ponteiros para Pessoas
19     Pessoa *esposa;
20 }Casal;
21 Casal matrimonio(Pessoa *p1,Pessoa *p2){
22     Casal c;
23     strcpy(c.data,"02/06/20");
24     c.marido = p1;
25     c.esposa = p2;
26     return c; }
27 void imprimeCertidaoCasamento(Casal cs){
28     printf("\n\nCertidão de Casamento!\n");
29     printf("\n Nada data %s Casaram-se neste cartório ",cs.data);
30     printf("\n \t%s e %s",cs.marido->nome,cs.esposa->nome);
31     printf("\n\n Dou fé a este Matrimonio!\n"); }
32 int main(){
33     system("clear");
34     setlocale(LC_ALL, "");
35     Pessoa joao,maria;
36     joao.cod = 1;   joao.idade = 30;
37     strcpy(joao.nome,"Joao Carlos");
38     strcpy(joao.endereco.nomeRua, "Duque de caxias");
39     joao.endereco.numero = 54;
40     strcpy(joao.endereco.cep, "96234-234");
41     maria.cod = 2;   maria.idade = 26;
42     strcpy(maria.nome,"Maria Aparecida");
43     strcpy(maria.endereco.nomeRua, "Duque de caxias");
44     maria.endereco.numero = 45;
45     strcpy(maria.endereco.cep, "96234-234");
46
47     //matrimonio é o relacionamento entre duas pessoas
48     Casal joaoEmaria = matrimonio(&joao,&maria);
49     imprimeCertidaoCasamento(joaoEmaria);
50     exit(0);
51 }
```

# Exemplo3.c

- Estrutura DATA
  - Dia,mês,ano

Certidão de Casamento!

Nada data 15/3/2022 Casaram-se neste cartório

Joao Carlos e Maria Aparecida

Dou fé a este Matrimonio!

Saída no terminal

```
void imprimeCertidaoCasamento(Casal cs){
    printf("\n\n\t\t Certidão de Casamento!\n");
    printf("\n\t Nada data %d/%d/%d Casaram-se neste cartório\n ",
           cs.dataCasamento.dia,cs.dataCasamento.mes,cs.dataCasamento.ano);
    printf("\n\t\t %s e %s",cs.marido->nome,cs.esposa->nome);
    printf("\n\n\t\t Dou fé a este Matrimonio!\n\n\n");
}

int main(){
    system("clear");
    setlocale(LC_ALL, "");
    Pessoa joao,maria;
    Data hoje = {15,03,2023} ;
    joao.cod = 1;
    joao.idade = 30;
    strcpy(joao.nome,"Joao Carlos");
    strcpy(joao.endereco.nomeRua, "Duque de caxias");
    joao.endereco.numero = 54;
    strcpy(joao.endereco.cep, "96234-234");
    maria.cod = 2;
    maria.idade = 26;
    strcpy(maria.nome,"Maria Aparecida");
    strcpy(maria.endereco.nomeRua, "Duque de caxias");
    maria.endereco.numero = 45;
    strcpy(maria.endereco.cep, "96234-234");

    //matrimonio é o relacionamento entre duas pessoas
    Casal joaoEmaria = matrimonio(&joao,&maria,hoje);
    imprimeCertidaoCasamento(joaoEmaria);
    exit(0);
}
```

# Array de struct

- Criando um array estático de estruturas Pessoas
  - VetorPessoas [4]
  - Mesmo processo de array de tipos primitivos;
  - Cada Elemento do Array é uma Struct
  - Uso do operador “.” ponto;
  - Manipular o vetor é semelhante a um struct

```
//DECLARAÇÃO DE UM VETOR DE 4 Pessoas
Pessoa vetorPessoas[4];
```

```
//Cada posição do vetor é uma Pessoa struct.
```

```
for(int i=0;i<4;i++){
    vetorPessoas[i].cod = i+1;
    //Entrada de Dados
    printf("\nInforme seu nome:");
    scanf("%s",vetorPessoas[i].nome);
    printf("\nInforme seu Sobrenome:");
    scanf("%s",vetorPessoas[i].sobrenome);
    printf("\nInforme sua idade:");
    scanf("%d",&vetorPessoas[i].idade);
    printf("\nInforme seu telefone:");
    scanf("%s",vetorPessoas[i].telefone);
}
```

ElementoVetor.membro da struct

# Array de struct

- Criando um array de **ponteiros** para estruturas Pessoas
  - Cada posição do vetor tem um ponteiro para uma struct Pessoa;
  - As estruturas não foram alocadas no momento da criação do vetor;
    - Precisa usar o Malloc para cada elemento do vetor

```
//DECLARAÇÃO DE UM VETOR DE PONTEIROS PARA 4 Pessoas
//Cada posição do vetor é um ponteiro para uma struct.
Pessoa *vetorPessoas[4];

for(int i=0;i<4;i++){
    //alocação de memória para cada um dos Pessoas
    vetorPessoas[i] = (Pessoa*)malloc(sizeof(Pessoa));
    vetorPessoas[i]->cod = i+1;    //Entrada de Dados com ponteiros
    printf("\nInforme seu nome:");
    scanf("%s",vetorPessoas[i]->nome);
    printf("\nInforme seu Sobrenome:");
    scanf("%s",vetorPessoas[i]->sobrenome);
    printf("\nInforme sua idade:");
    scanf("%d",&vetorPessoas[i]->idade);
    printf("\nInforme seu telefone:");
    scanf("%s",vetorPessoas[i]->telefone);
}
```

Ponteiro->membro da struct

# Array de struct

- As estruturas não foram alocadas até o momento;
  - Precisa usar o Malloc

```
Cliente *vetorClientes[4]; //DECLARAÇÃO DE UM VETOR DE PONTEIROS PARA CLIENTES

for(int i=0;i<4;i++){
    //reserva memória para cada cliente
    vetorClientes[i] = (Cliente*)malloc(sizeof(Cliente));
```



# Exemplo4.c

- Vetor de ponteiros \*
  - 4 elementos ponteiros de Pessoa

```
typedef struct pessoa{
    int cod;
    char nome [15];
    char sobrenome [20];
    int idade;
    char telefone [10];
}Pessoa;

int main()
{
    //DECLARAÇÃO DE UM VETOR DE PONTEIROS PARA 5 Pessoas
    //Cada posição do vetor é um ponteiro para uma struct.
    Pessoa *vetorPessoas[4];

    for(int i=0;i<4;i++){
        //alocação de memória para cada um dos Pessoas
        vetorPessoas[i] = (Pessoa*)malloc(sizeof(Pessoa));
        vetorPessoas[i]->cod = i+1;    //Entrada de Dados com ponteiros
        printf("\nInforme seu nome:");
        scanf("%s",vetorPessoas[i]->nome);
        printf("\nInforme seu Sobrenome:");
        scanf("%s",vetorPessoas[i]->sobrenome);
        printf("\nInforme sua idade:");
        scanf("%d",&vetorPessoas[i]->idade);
        printf("\nInforme seu telefone:");
        scanf("%s",vetorPessoas[i]->telefone);
    }
    return 0;
}
```

# Exemplo4.c

- Vetor de ponteiros
  - Vetor já é um ponteiro, se os elementos forem ponteiros, temos ponteiro de ponteiro (\*\*ps)

```
typedef struct pessoa{
    int cod;
    char nome [15];
    char sobrenome [20];
    int idade;
    char telefone [10];
}Pessoa;

> void mostraVetor(Pessoa **ps, int n){ ...

> void preencheVetor(Pessoa **ps, int n){ ...

int main()
{

    //DECLARAÇÃO DE UM VETOR DE PONTEIROS PARA 4 Pessoas
    //Cada posição do vetor é um ponteiro para uma struct.
    Pessoa *vetorPessoas[2];

    preencheVetor(vetorPessoas, 2);

    mostraVetor(vetorPessoas, 2);

    return 0;
}
```



# Exemplo4.c

- Cada elemento do vetor é um ponteiro;

```
void mostraVetor(Pessoa **ps, int n){
//Mostra  UM STRUCT
    Pessoa *p;
    for(int i=0;i<n;i++){
        p = ps[i];
        printf("Pessoa: %s %s \n",p->nome,p->sobrenome);
        printf("\tCodigo: %d e idade %d \n",p->cod,p->idade);
        printf("\tTelefone: %s \n",p->telefone);
    }
}

void preencheVetor(Pessoa **ps, int n){
    for(int i=0;i<n;i++){
        //alocação de memória para cada um dos Pessoas
        ps[i] = (Pessoa*)malloc(sizeof(Pessoa));
        ps[i]->cod = i+1; //Entrada de Dados com ponteiros
        printf("\nInforme seu nome:");
        scanf("%s",ps[i]->nome);
        printf("\nInforme seu Sobrenome:");
        scanf("%s",ps[i]->sobrenome);
        printf("\nInforme sua idade:");
        scanf("%d",&ps[i]->idade);
        printf("\nInforme seu telefone:");
        scanf("%s",ps[i]->telefone);
    }
}
```



# Exercício 1

- Defina uma estrutura para armazenar dados de um apartamento:
  - Ex: Nome do condomínio, número, andar, quantidade cômodos, valor do aluguel, valor condomínio, box e etc.

```
1
2
3
4
5     typedef struct apartamento{
6         char condomio [30];
7         int num, andar, qtd_comodos,box;
8         double v_aluguel, v_condominio;
9     }Apartamento;
10
```
  - Crie dois apartamentos, atribua valores aos seus membros (ap101, ap201);
  - Crie uma função para imprimir os valores de um apartamento (passagem por val `void mostraDadosApartamento(Apartamento ap){`
  - Crie uma `void alteraAluguelEm10porcentos(Apartamento *ap){` rtamento (passagem por referência);

# Exercício 1

- Saída do Exercício 1

Apartamento:

Residencial: Avenida Brasil Nº= 101 Andar= 1 Box= 10  
Aluguel: 300.500000 e Condomínio 93.000000

Apartamento:

Residencial: Avenida Brasil Nº= 201 Andar= 2 Box= 20  
Aluguel: 300.330000 e Condomínio 100.000000

Apartamento:

Residencial: Avenida Brasil Nº= 101 Andar= 1 Box= 10  
Aluguel: 330.550000 e Condomínio 103.000000

—



Com aumento de 10%

# Exercício 2

- Crie dois ponteiros de apartamentos, utilize alocação dinâmica (malloc). Após, atribua valores aos campos dos apartamentos (ex. 301 e 401).
  - Após mostre o apartamento criado (301);
- Defina uma função que recebe um apartamento e retorna o box do apartamento;

```
int retornaBoxApartamento(Apartamento ap){
```

- Mostre apenas o box do apartamento 301;
- Defina um função para criar um novo apartamento, a função recebe por parâmetro os dados do apartamento (Condomínio, andar, número, etc) e retorna o ponteiro do novo apartamento (atribua ao 401);

```
Apartamento criaNovoApartamento(char *condominio, int num, int andar, double v_aluguel, double v_condominio, int box){
```

- Mostre os dados do apartamento 401 criado;

# Exercício 2

## Exemplo de saída das funções anteriores

Apartamento:

Residencial: Avenida Brasil N°= 301 Andar= 3 Box= 23

Aluguel: 300.330000 e Condomínio 100.000000

Box do apartamento 301 éo box = 23

Apartamento:

Residencial: Avenida Brasil N°= 401 Andar= 4 Box= 17

Aluguel: 200.000000 e Condomínio 100.000000

ch 5 14 17

# Exercício 3

- Defina dois proprietários e atribua aos apartamentos 101 e 201;
- Implemente a função `mostraProprietario`;  

```
void mostraProprietario(Apartamento ap){  
    // ...  
}
```
- Aloque espaço para um novo proprietário (use Malloc), e troque o dono do apartamento 101 para o novo proprietário.
  - Use o Malloc

# Exercício 3 - Apartamentos

- Crie a estrutura proprietário, com nome e cpf como membros;
- Adicione a estrutura Apartamento o membro, Proprietário;

```
typedef struct proprietario{  
    char nome [30];  
    char cpf [14];  
}Proprietario;
```

Qual a diferença entre  
proprietário Ponteiro e Variável?

```
typedef struct apartamento{  
    char condomio [30];  
    int num, andar, qtd_comodos,box;  
    double v_aluguel, v_condominio;  
    Endereco *ed;  
    Proprietario *dono;  
}Apartamento;
```



Ponteiro para Proprietário

# Exercício 4 - Apartamentos

- Defina uma **Struct Data**, com dia, mês e Ano de membro;
- Insira na **Struct Apartamento** a data de compra;

```
5  typedef struct proprietario{
6      char nome [30];
7      char cpf [14];
8  }Proprietario;
9
10 typedef struct data{
11     int dia,mes,ano;
12 }Data;
13
14 typedef struct apartamento{
15     char condomio [30];
16     int num, andar, qtd_comodos,box;
17     double v_aluguel, v_condominio;
18     Proprietario *dono;
19     Data compra;
20 }Apartamento;
```

# Exercício 4 - Apartamentos

- Crie um função para vender um apartamento, que recebe um apartamento, um ponteiro para proprietário (comprador) que se tornará o novo dono, e um data;

```
void venderApartamento(Apartamento *ap, Proprietario *pr, Data dt){
```

- Atualize a função para mostrar todos os detalhes do apartamento;

```
void mostraDadosApartamento(Apartamento ap){
```



# Exercício 5 - Apartamentos

- Crie um bloco de apartamentos com 4 apartamentos (vetor de apartamentos), defina os atributos de cada apartamento;
- Imprima o bloco de apartamentos com todos os dados dos apartamentos;
- Associe com a estrutura apartamento, um proprietário (pessoa) com a data de compra. Defina um mesmo proprietário para mais de um apartamento;



Dúvidas ??