



**INSTITUTO FEDERAL**  
Sul-rio-grandense

Câmpus  
Charqueadas

EDUCAÇÃO  
**PÚBLICA**  
**100%**  
GRATUITA

# Ponteiros em C

Programação Estruturada

Prof. André del Mestre

# Principais Conceitos

# Ponteiros

## Definição

- Variável
  - Eh um espaço reservado de memoria para guardar um **valor** que pode ser modificado pelo programa
- Ponteiro
  - Eh um espaço reservado de memoria para guardar o **endereço de memoria** de uma outra variável
  - Um ponteiro eh uma variável que armazena endereços de memoria

# Ponteiros

## Declaração

- O asterisco (\*) informa ao compilador que a variável armazena endereços para o tipo especificado
  - Exemplos:

```
// ptr eh um ponteiro para int. num eh uma variável inteira
int *ptr, num;
// ptr_ch eh um ponteiro para char. num_ch eh uma variável character
char *ptr_ch, num_ch;
// ptr_fl eh um ponteiro para float. num_fl eh uma variável float
float *ptr_fl, num_fl;
```

# Ponteiros

## Inicializacao

- Ponteiros apontam para uma posicao de memoria
  - **Atencao:** Ponteiros nao inicializados apontam para um lugar indefinido
  - De forma similar, uma variavel nao inicializada tem valor aleatorio, chamado de lixo de memoria
- Exemplo:

```
int *ptr, num;
```

Endereco	Variavel	Valor
119		
120	int *ptr	????
121		
122	int num	????
123		

# Ponteiros

## Inicializacao

- Ponteiros apontam para uma posicao de memoria
  - **Atencao:** Ponteiros nao inicializados apontam para um lugar indefinido
  - De forma similar, uma variavel nao inicializada tem conteudo aleatorio, chamado de lixo de memoria
- Exemplo:

```
int *ptr=NULL, num=4;
```

Valor especial NULL  
eh o endereco de  
nenhum lugar

Endereco	Variavel	Conteudo
119		
120	int *ptr	NULL
121		
122	int num	4
123		

# Ponteiros

## Inicializacao

- Ponteiros devem apontar para um lugar conhecido
  - Por exemplo, ponteiro aponta para uma variavel conhecida
  - **Atencao:** Pontoeiro de inteiro aponta apenas para variaveis inteiras!
- Veja o codigo abaixo:

```
int *ptr=NULL, num=4;  
ptr = &num;
```

Pontoeiro ptr aponta  
para o endereca da  
variavel num

Endereco	Variavel	Conteudo
119		
120	int *ptr	122
121		
122	int num	4
123		



# Ponteiros

## Utilização

- ANTES de conhecer ponteiros, só interessava o valor da variável
- DEPOIS de ver ponteiros, o endereço de memória da variável importa.
  - `*ptr` - acessa o valor apontado pelo ponteiro `ptr`

```
int *ptr=NULL, num=4;  
ptr = &num;  
→ printf("VALORES\n num=%i \n ptr=%i\n", num, *ptr);  
printf("ENDERECO\n num=%X \n ptr=%X\n", &num, ptr);
```

Valor de  
num

Valor  
apontado pelo  
ponteiro ptr

Endereco	Variavel	Valor
119		
120	int *ptr	122
121		
122	int num	4
123		



# Ponteiros

## Utilização

- ANTES de conhecer ponteiros, só interessava o valor da variável
- DEPOIS de ver ponteiros, o endereço de memória da variável importa.
  - `*ptr` - acessa o valor apontado pelo ponteiro `ptr`
  - `&num` - corresponde ao endereço de memória onde esta a variável `num`

```
int *ptr=NULL, num=4;
ptr = &num;
printf("VALORES\n num=%i \n ptr=%i\n", num, *ptr);
printf("ENDERECO\n num=%X \n ptr=%X\n", &num, ptr);
```

```
VALORES  num=4    ptr=4
ENDERECO num=122  ptr=122
```

Endereco	Variavel	Valor
119		
120	int *ptr	122
121		
122	int num	4
123		



# Ponteiros

## Utilização

- Exercício - Qual será a saída dos prints do trecho de código abaixo?

```
int *ptr=NULL, num=0, num2=0;
ptr = &num;   num=20;
printf("\n\nVALORES\n num=%i \n ptr=%i\n", num, *ptr);
printf("ENDERECO\n num=%X \n ptr=%X\n", &num, ptr);

*ptr = 10;
printf("\n\nVALORES\n num=%i \n ptr=%i\n", num, *ptr);
printf("ENDERECO\n num=%X \n ptr=%X\n", &num, ptr);

ptr = &num2; *ptr = 30;
printf("\n\nVALORES\n num=%i \n ptr=%i\n", num, *ptr);
printf("ENDERECO\n num=%X \n ptr=%X\n", &num, ptr);
```

Endereco	Variavel	Valor
119		
120	int *ptr	???
121		
122	int num	???
123	int num2	???

# Ponteiros

## Utilização

- Exercício - Qual será a saída dos prints do trecho de código abaixo?
  - Entrada: 3 5

```
int *ptr=NULL, num=0, num2=0;
ptr = &num;
scanf("%i %i", ptr, &num2);
printf("VALORES\n num=%i \n num2=%i\n", num, num2);
```

Endereco	Variavel	Valor
119		
120	int *ptr	???
121		
122	int num	???
123	int num2	???

# Operações com Ponteiros

# Operações com Ponteiros

- Operadores unários

- `&` – endereço de uma variável  
`ptr = &num;`
- `*` – valor apontado por ponteiro  
`*ptr = 4`

- Atribuição

- `ptr2 = ptr;`

- Operadores relacionais

- `==`, `!=`, `>=`, `<=`, `>`, `<`
- Exemplo

```
if(ptr2 == ptr){  
    printf("ptrs apontam p ends iguais");  
}
```

```
int main (){  
    int *ptr, *ptr2, num, num2;  
  
    ptr = &num;  
    *ptr = 4  
  
    num2 = *(&num); //num2 = num  
    ptr2 = &(*ptr); //ptr2 = ptr  
  
    if(ptr2 == ptr){  
        printf("Ponteiros apontam para o mesmo endereco\n");  
        printf("ptr=%X \n ptr2=%X\n", ptr, ptr2);  
    }else{  
        printf("Ponteiros apontam para enderecos diferentes\n");  
        printf("ptr=%X \n ptr2=%X\n", ptr, ptr2);  
    }  
}
```

# Operações com Ponteiros

## Operações aritméticas

- **Permitido** (endereços)

- Apenas soma e subtração de INTEIROS

```
ptr--;  
ptr=ptr+15;  
ptr+=2;
```

- **Proibido** (endereços)

- Divisão e multiplicação

```
ptr=ptr/2;
```

- Operações com números reais

```
ptr=2.5;
```

- Soma e subtração entre ponteiros

```
ptr=ptr+ptr2;
```

- **Permitido** (valores)

- Todas as operações aritméticas

```
*(ptr)=2.5;  
*(ptr)++;  
*ptr = *(ptr) * 5;
```

# Operações com Ponteiros

## Operações aritméticas

```
int main () {  
    char *ptr_ch=NULL, num_ch='A';  
    float *ptr_fl=NULL, num_fl=1.5;  
    double *ptr_db=NULL, num_db=4.5;  
  
    ptr_ch = &num_ch; //ptr_ch=120  
    ptr_fl = &num_fl; //ptr_fl=122  
    ptr_db = &num_db; //ptr_db=130  
  
    ptr_ch++; ptr_fl++; ptr_db++;  
  
}
```

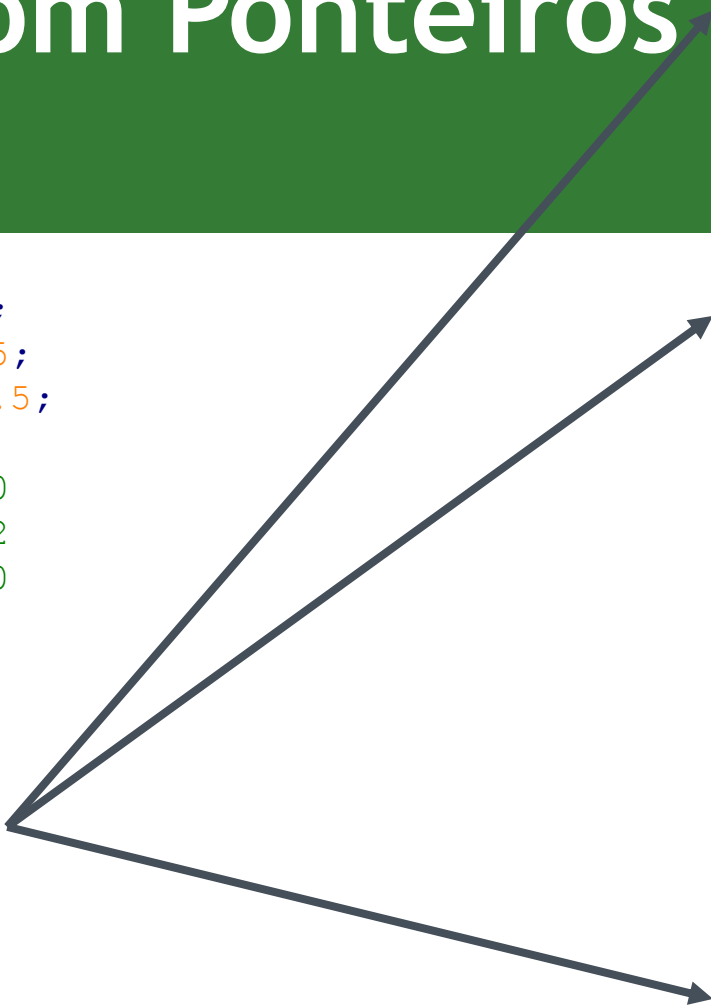
Endereco	Variavel	Valor
119		
120	num_ch	A
121		
122	num_fl	1.5
123		
124		
125		
126		
127		
128		
129		
130	num_db	3.5
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		

# Operações com Ponteiros

## Operações aritméticas

```
int main () {  
    char *ptr_ch=NULL, num_ch='A';  
    float *ptr_fl=NULL, num_fl=1.5;  
    double *ptr_db=NULL, num_db=4.5;  
  
    ptr_ch = &num_ch; //ptr_ch=120  
    ptr_fl = &num_fl; //ptr_fl=122  
    ptr_db = &num_db; //ptr_db=130  
  
    ptr_ch++; ptr_fl++; ptr_db++;  
  
    //ptr_ch=121  
    //ptr_fl=126  
    //ptr_db=138  
}
```

Endereco	Variavel	Valor
119		
120	num_ch	A
121		
122	num_fl	1.5
123		
124		
125		
126		
127		
128		
129		
130	num_db	3.5
131		
132		
133		
134		
135		
136		
137		
138		
139		
140		





# Ponteiros e Funções

# Ponteiros e Funções

## Passagem de parâmetro

- **Passagem por valor**

- É feita uma cópia do valor dentro da função
- Operações ocorrem na cópia
- Sem **return**, todas as variáveis da função são encerradas ao fim da função

```
void incrementa(int num) {  
    num++;  
    printf("Valor DENTRO da funcao = %i\n", num);  
}  
  
int main () {  
    int num=5;  
    printf("Valor ANTES da funcao = %i\n", num);  
    incrementa(num);  
    printf("Valor DEPOIS da funcao = %i\n", num);  
}
```

Endereco	Variavel	Conteudo
119		
120	int num (incrementa)	6
121		
122	int num (main)	5
123		

```
Valor ANTES da funcao = 5  
Valor DENTRO da funcao = 6  
Valor DEPOIS da funcao = 5
```

# Ponteiros e Funções

## Passagem de parâmetro

- **Passagem por referência**

- O endereço é copiado para dentro da função
- Alterações de valor ocorrem a partir do endereço passado como parâmetro
- Qualquer alteração de valor dentro da função, vai se refletir fora da função.
- Observe a sintaxe!
  - `&` para passar o end. para função
  - `*` para operar valor dentro da função

Endereco	Variavel	Conteudo
119		
120	int *ptr (incrementa)	122
121		
122	int num (main)	5
123		



```
void incrementa(int *ptr){
    *(ptr)++;
    printf("Valor DENTRO da funcao = %i\n", *ptr);
}

int main (){
    int num=5;
    printf("Valor ANTES da funcao = %i\n", num);
    incrementa(&num);
    printf("Valor DEPOIS da funcao = %i\n", num);
}
```

```
Valor ANTES da funcao = 5
Valor DENTRO da funcao = 6
Valor DEPOIS da funcao = 6
```

# Ponteiros e Funções

## Passagem de parâmetro

- **Passagem por referência**

- Você PODE passar ponteiros como parâmetro!

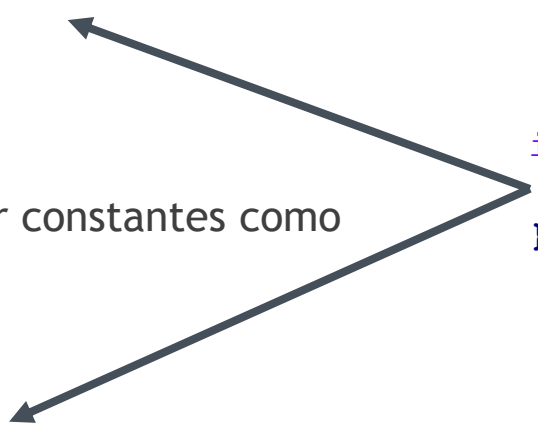
```
int num, *ptr;  
ptr=&num;  
scanf("%i", ptr);  
incrementa(ptr);  
incrementa(&num);
```

- Você NÃO PODE passar constantes como parâmetro!

```
int num, *ptr;  
ptr=&num;  
scanf("%i", 4);  
incrementa(7);
```

```
void incrementa(int *ptr){  
    *(ptr)++;  
    printf("Valor DENTRO da funcao = %i\n", *ptr);  
}
```

```
int main () {  
  
}
```



# Ponteiros e Funções

## Passagem de parâmetro

- **Passagem por referência**

- Você PODE modificar mais de uma variável utilizando apenas uma função!

```
ANTES = 5 A 3.3
DENTRO = 6 B 4.3
DEPOIS = 6 B 4.3
```

```
void incrementa_tudo(int *ptr, char *ptr_ch, float *ptr_fl){

    *(ptr)++;
    *(ptr_ch)++;
    *ptr_fl = *ptr_fl + 1;

    printf("DEPOIS = %i %c %.1f\n", *ptr, *ptr_ch, *ptr_fl);
}

int main (){
    int num=5; char ch='A'; float fl=3.3;

    printf("ANTES = %i %c %.1f\n", num, ch, fl);

    incrementa_tudo(&num, &ch, &fl);

    printf("DEPOIS = %i %c %.1f\n", num, ch, fl);
}
```

# Ponteiros e Funções

## Passagem de parâmetro

- **Passagem por referência**
  - O que faz a função `guess_what`?

```
ANTES = 3 5
DEPOIS = ? ?
```

```
void guess_what(int *ptr1, int *ptr2){
    int temp=0;

    temp = *ptr1;
    *ptr1= *ptr2;
    *ptr2= temp;
}

int main(){
    int num=3; num2=5;

    printf("ANTES da funcao = %i %i\n", num, num2);
    guess_what(&num, &num2);
    printf("DEPOIS da funcao = %i %i\n", num, num2);
}
```

# Ponteiros e Vetores

# Ponteiros e Vetores


## Relação

- Vetores são agrupamentos de mesmo tipo na memória
  - Elementos agrupados continuamente, sem fragmentação
  - Reserva quantidade sequencial de dados na memória
  - O vetor é um endereço para o início da sequência na memória
  - O nome do vetor SEM índice é um ponteiro para o primeiro elemento do vetor

```
int vet[5]={4,2,0,1,3}, *ptr;
```

```
ptr=vet;
```

Endereco	Variavel	Conteudo
116		
120	int *ptr	128
124		
128	int vet[0]	4
132	int vet[1]	2
136	int vet[2]	0
140	int vet[3]	1
144	int vet[4]	3
148		
152		





# Ponteiros e Vetores

## Relação


- O número entre colchetes é o deslocamento a partir do início do array
  - Exemplo:
    - `*(ptr+4)` é equivalente a `ptr[4]`

```
int vet[5]={4,2,0,1,3}, *ptr;
```

```
ptr=vet;
```

```
printf("ptr[0]=%i %i\n", *ptr, vet[0]);  
printf("ptr[0]+4=%i %i\n", *ptr+4, vet[0]+4);  
printf("ptr[4]=%i %i\n", *(ptr+4), vet[4]);
```

Endereco	Variavel	Conteudo
116		
120	int *ptr	128
124		
128	int vet[0]	4
132	int vet[1]	2
136	int vet[2]	0
140	int vet[3]	1
144	int vet[4]	3
148		
152		



```
ptr[0] = 4 4  
ptr[0]+4 = 8 8  
ptr[4] = 3 3
```

# Ponteiros e Vetores

## Relação


- No código abaixo, afirma-se
  - `*ptr` é equivalente a `vet[0]`
  - `*(ptr+indice)` é equivalente a `vet[indice]`
  - `vet` é equivalente a `&vet[0]`
  - `(ptr+indice)` é equivalente a `&vet[indice]`

```
int vet[5]={4,2,0,1,3}, *ptr;
```

```
ptr=vet;
```

```
printf("ptr[0]=%i %i\n", *ptr, vet[0]);  
printf("ptr[0]+4=%i %i\n", *ptr+4, vet[0]+4);  
printf("ptr[4]=%i %i\n", *(ptr+4), vet[4]);
```

Endereco	Variavel	Conteudo
116		
120	int *ptr	128
124		
128	int vet[0]	4
132	int vet[1]	2
136	int vet[2]	0
140	int vet[3]	1
144	int vet[4]	3
148		
152		



```
ptr[0] = 4 4  
ptr[0]+4 = 8 8  
ptr[4] = 3 3
```

# Ponteiros e Vetores

## Relação

- Usando vetor

```
int vet[5]={4,2,0,1,3}, *ptr, i;  
ptr=vet;  
for(i=0;i<5;i++){  
    printf("%i \n", ptr[i]);  
}
```

- Usando ponteiro

```
int vet[5]={4,2,0,1,3}, *ptr, i;  
ptr=vet;  
for(i=0;i<5;i++){  
    printf("%i \n", *(ptr+i));  
}
```

# Ponteiros e Vetores

## Vetores como parâmetro de funções

- Manipule vetores em funções utilizando passagem de parâmetros por referência
  - Exemplo 1

```
#define TAM 5
void adicionaNumAoVetor(int * vet, int tamanho, int num){
    int i;
    // vet[i] = vet[i] + num;
    for(i=0;i<tamanho;i++)
        *(vet+i) = *(vet+i) + num;
}
int main(){
    int vet[TAM]={4,2,0,1,3};
    adicionaNumAoVetor(vet, TAM, -1);
    return 0;
}
```

Endereco	Variavel	Conteudo
116		
120	int vet[0]	4
124	int vet[1]	2
128	int vet[2]	0
132	int vet[3]	1
136	int vet[4]	3
140		
144		
148		
152		
156		
160		

**ANTES de** adicionaNumAoVetor()

# Ponteiros e Vetores

## Vetores como parâmetro de funções

- Manipule vetores em funções utilizando passagem de parâmetros por referência
  - Exemplo 1

```
#define TAM 5
void adicionaNumAoVetor(int * vet, int tamanho, int num){
    int i;
    // vet[i] = vet[i] + num;
    for(i=0;i<tamanho;i++)
        *(vet+i) = *(vet+i) + num;
}
int main(){
    int vet[TAM]={4,2,0,1,3};
    adicionaNumAoVetor(vet, TAM, -1);
    return 0;
}
```

Endereco	Variavel	Conteudo
116		
120	int vet[0]	3
124	int vet[1]	1
128	int vet[2]	-1
132	int vet[3]	0
136	int vet[4]	2
140		
144		
148		
152		
156		
160		

DEPOIS de adicionaNumAoVetor()

# Ponteiros e Vetores

## Vetores como parâmetro de funções

- Manipule vetores em funções utilizando passagem de parâmetros por referência
  - Exemplo 2

```
#define TAM 5
void somaVetores(int *vet1, int *vet2, int tamanho){
    int i;
    // vet1[i] = vet1[i] + vet2[i];
    for(int i=0;i<tamanho;i++)
        *(vet1+i) = *(vet1+i) + *(vet2+i);
}
int main(){
    int vet[TAM]={4,2,0,1,3}, outro_vet[TAM]={7,9,5,6,8};
    somaVetores(vet, outro_vet, TAM);
    return 0;
}
```

Endereco	Variavel	Conteudo
116		
120	int vet[0]	4
124	int vet[1]	2
128	int vet[2]	0
132	int vet[3]	1
136	int vet[4]	3
140		
144	int outro_vet[0]	7
148	int outro_vet[1]	9
152	int outro_vet[2]	5
156	int outro_vet[3]	6
160	int outro_vet[4]	8

ANTES de somaVetores()

# Ponteiros e Vetores

## Vetores como parâmetro de funções

- Manipule vetores em funções utilizando passagem de parâmetros por referência
  - Exemplo 2

```
#define TAM 5
void somaVetores(int *vet1, int *vet2, int tamanho){
    int i;
    // vet1[i] = vet1[i] + vet2[i];
    for(int i=0;i<tamanho;i++)
        *(vet1+i) = *(vet1+i) + *(vet2+i);
}
int main(){
    int vet[TAM]={4,2,0,1,3}, outro_vet[TAM]={7,9,5,6,8};
    somaVetores(vet, outro_vet, TAM);
    return 0;
}
```

Endereco	Variavel	Conteudo
116		
120	int vet[0]	11
124	int vet[1]	11
128	int vet[2]	5
132	int vet[3]	7
136	int vet[4]	11
140		
144	int outro_vet[0]	7
148	int outro_vet[1]	9
152	int outro_vet[2]	5
156	int outro_vet[3]	6
160	int outro_vet[4]	8

DEPOIS de somaVetores()

MUITO  
OBRIGADO

Prof. André del Mestre

[www.ifsul.edu.br](http://www.ifsul.edu.br)  
[almmartins@charqueadas.ifsul.edu.br](mailto:almmartins@charqueadas.ifsul.edu.br)