

Estrutura de Dados

Prof. Maurício Neto

O que vamos ver?

01

Ponteiros

02

Ponteiro para
Ponteiros

03

Typedef

01

Ponteiros

Ponteiros

O que é?

Ponteiro é uma variável que armazena o endereço de memória de outra variável. Em outras palavras, um ponteiro "**aponta**" para a localização na memória onde os dados estão armazenados, em vez de armazenar diretamente os dados em si. Isso permite trabalhar com dados indiretamente, manipulando os endereços de memória em vez dos próprios dados.

Em resumo, ponteiro é um apontador para o endereço de memória.

Ponteiros

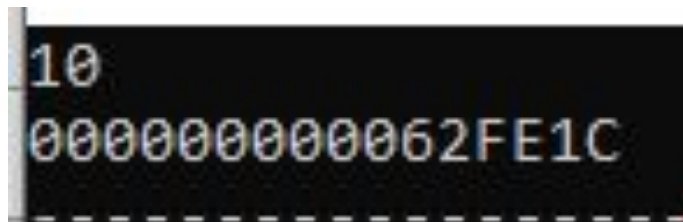
```
#include <stdio.h>

int main(){

    int var = 10;

    printf ("%d\n", var);
    printf ("%p", &var);

    return 0;
}
```



10
00000000000062FE1C

Ponteiros – Exemplo

```
int var = 10
```

Declaramos nossa variável

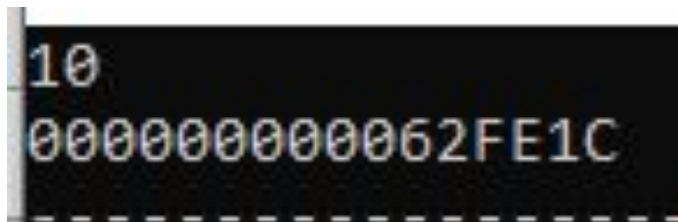
```
printf("%d \n", var);
```

Estamos utilizando o especificador %d, que é utilizado para imprimir valores inteiros.

```
printf("%p", &var);
```

%p é o especificador usado para imprimir endereços de memória.

&var é o operador de endereço, que retorna o endereço de memória da variável var.



Ponteiros – Exemplo

Em nosso exemplo estamos vendo o endereço **FÍSICO** da variável, no exemplo **62FE1C**, e seu **CONTEÚDO** é **10**.

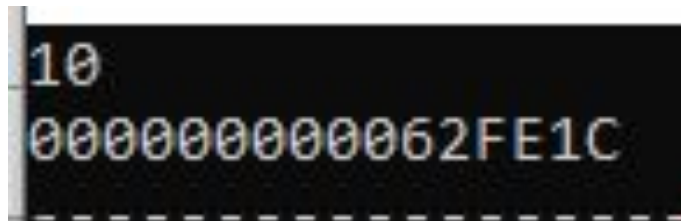
```
#include <stdio.h>

int main(){

    int var = 10;

    printf ("%d\n", var);
    printf ("%p", &var);

    return 0;
}
```



```
10
00000000000062FE1C
```

Ponteiros – Exemplo

Em C, utilizamos o * em frente ao nome da variável, para informar que é um **PONTEIRO**.

E vamos **apontar**, para o endereço da variável que criamos.

```
int var = 10;  
int *ptr;  
  
ptr = &var;
```


Ponteiros – Exemplo

	var				
	10				
6000	6001	6002	6003	6004	6005

Digamos que nossa variável **VAR**, ficou armazenada no endereço 6001, dentro dela vai existir o valor 10 que declaramos.

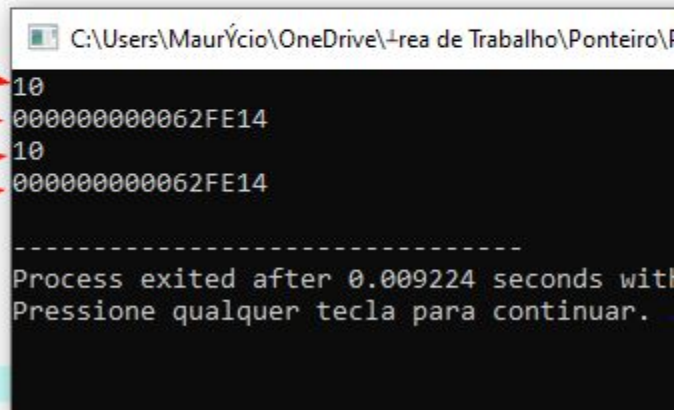
Ponteiros – Exemplo

	var		ptr		
	10		6001		
6000	6001	6002	6003	6004	6005

Digamos que nossa variável **PTR**, ficou armazenada no endereço 6003, dentro dela vai existir o valor 6001, que é o endereço da nossa variável **VAR**.

Ponteiros – Exemplo

```
int main(){  
  
    int var = 10;  
    int *ptr;  
  
    ptr = &var;  
  
    printf ("%d\n", var);  
    printf ("%p\n", &var);  
  
    printf ("%d\n", *ptr);  
    printf ("%p\n", ptr);  
  
    return 0;  
}
```



```
C:\Users\Maurício\OneDrive\Área de Trabalho\Ponteiro\F  
10  
000000000062FE14  
10  
000000000062FE14  
-----  
Process exited after 0.009224 seconds with  
Pressione qualquer tecla para continuar.
```

Ponteiros – Exemplo

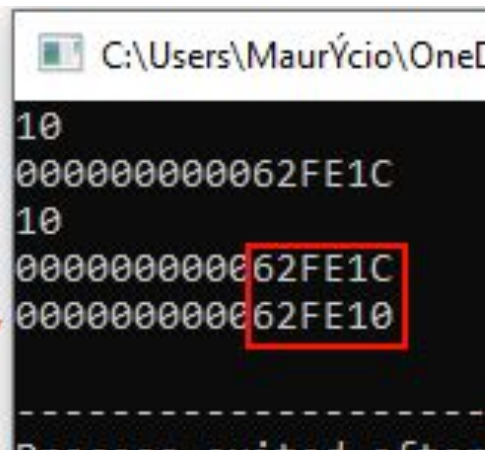
Quando utilizamos *, estamos utilizando o conteúdo do endereço apontado pelo ponteiro.

Se imprimirmos o endereço do ponteiro que criamos, vamos ver o endereço do **PTR**.

```
printf ("%d\n", var);  
printf ("%p\n", &var);
```

```
printf ("%d\n", *ptr);  
printf ("%p\n", ptr);
```

```
printf ("%p\n", &ptr);
```



```
C:\Users\Maurício\One[  
10  
000000000062FE1C  
10  
000000000062FE1C  
000000000062FE10  
-----  
Program ended with...
```

Ponteiros – Exemplo

Se alterarmos o valor do nosso **PONTEIRO**, ele vai alterar o **CONTEÚDO**, do endereço da variável.

```
*ptr = 20;  
  
printf ("\n\n");  
printf ("%d\n", var);  
printf ("%p\n", &var);  
printf ("%d\n", *ptr);  
printf ("%p\n", ptr);  
printf ("%p\n", &ptr);
```

```
20  
0000000000062FE1C  
20  
0000000000062FE1C  
0000000000062FE10
```

Process exited after

Ponteiros – Exemplo

```
int main(){  
  
    int var = 10;  
    int *ptr;  
  
    ptr = &var;  
  
    printf ("%d\n", var);  
    printf ("%p\n", &var);  
    printf ("%d\n", *ptr);  
    printf ("%p\n", ptr);  
    printf ("%p\n", &ptr);  
  
    *ptr = 20;  
  
    printf ("\n\n");  
    printf ("%d\n", var);  
    printf ("%p\n", &var);  
    printf ("%d\n", *ptr);  
    printf ("%p\n", ptr);  
    printf ("%p\n", &ptr);  
  
    return 0;  
}
```

```
10  
000000000062FE1C  
10  
000000000062FE1C  
000000000062FE10  
  
20  
000000000062FE1C  
20  
000000000062FE1C  
000000000062FE10
```

Quando usar ponteiros?

Vamos utilizar ponteiro com variáveis simples (int, float, char), nesses casos precisamos utilizar o & para indicar onde é o endereço da memória que o valor vai ser armazenado.

```
int idade;  
  
printf ("Digite sua idade: ");  
scanf ("%d", &idade);  
  
printf ("%d", idade);
```

Mas, se utilizamos um **ARRAY**, para o nome, não precisamos utilizar &, pois o próprio array já é o ponteiro.

```
char nome[10];  
  
printf ("Digite seu nome: ");  
scanf ("%s", nome);  
  
printf ("%s", nome);
```

Ponteiros – Exemplo

```
#define pi 3.14159
```

```
int main() {
```

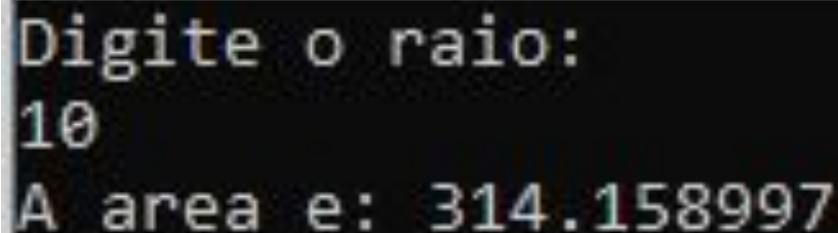
```
    float raio, area;
```

```
    printf("Digite o raio: \n");
```

```
    scanf("%f", &raio);
```

```
    area = pi * raio * raio;
```

```
    printf("A area e: %f\n", area);
```

A screenshot of a terminal window with a black background and white text. It shows the execution of a C program. The first line is the prompt "Digite o raio:", followed by the user input "10". The second line shows the output "A area e: 314.158997".

```
Digite o raio:  
10  
A area e: 314.158997
```


Ponteiros – Exemplo

```
#define pi 3.14159

int main() {

    float raio, area;

    printf("Digite o raio: \n");
    scanf("%f", &raio);

    area = pi * raio * raio;

    printf("A area e: %f\n", area);
```

```
Digite o raio:
10

-----
Process exited after 3.218
Pressione qualquer tecla pa
```

Especificadores mais usados

%d: Números inteiros (decimal).

%f: Números em ponto flutuante (decimal).

%c: Caracteres individuais.

%s: Strings de caracteres.

%x: Números inteiros em hexadecimal (minúsculo ou maiúsculo).

%u: Números inteiros sem sinal.

%p: Ponteiros.

Atividade

1 - Faça um código, onde o usuário digita três números, e seja impresso na tela os três números e que exiba também sua posição na memória.

```
Numeros digitados e seus endereços de memória:  
Numero 1: 10 (Endereco: 000000000062FE1C)  
Numero 2: 20 (Endereco: 000000000062FE18)  
Numero 3: 30 (Endereco: 000000000062FE14)
```

2 - Escreva um programa que declare uma variável inteira e um ponteiro para inteiro. Atribua um valor à variável e faça o ponteiro apontar para ela. Em seguida, imprima o valor da variável, o valor apontado pelo ponteiro, o valor da memória da variável e o valor do ponteiro na memória.

```
Valor da variavel: 10  
Valor apontado pelo ponteiro: 10  
Valor na memoria da variavel: 6487580  
Valor apontado da memoria do ponteiro: 6487568
```

02

Ponteiro para Ponteiros

Ponteiro para Ponteiro

Quando usamos um *, estamos informando que é um ponteiro que aponta para a região de memória. Quando colocamos **, estamos informando que é um ponteiro para ponteiro, é uma variável ponteiro que vai apontar para outra variável que também é um ponteiro.

```
int a = 10;  
int *b = &a;  
int **c = &b;
```

Ponteiro para Ponteiro

```
int a = 10;  
int *b = &a;  
int **c = &b;
```

`int a = 10;` cria uma variável inteira a com o valor 10.

`int *b = &a;` cria um ponteiro b que aponta para a.

`int **c = &b;` cria um ponteiro para ponteiro c que aponta para b.

Ponteiro para Ponteiro

```
printf("O endereço de A: %p\tConteudo de A: %d\n", &a, a);  
printf("O endereço de B: %p\tConteudo de B: %p\n", &b, b);  
printf("O endereço de C: %p\tConteudo de C: %p\n", &c, c);
```

```
O endereço de A: 0x7ffdb01d9ff4 Conteudo de A: 10  
O endereço de B: 0x7ffdb01d9ff8 Conteudo de B: 0x7ffdb01d9ff4  
O endereço de C: 0x7ffdb01da000 Conteudo de C: 0x7ffdb01d9ff8
```

Ponteiro para Ponteiro

Quando solicitamos para imprimir na tela o conteúdo de ***b** ou ****c**, ele vai apontar para 10, pois **c** recebeu o endereço de **b** e **b** recebeu o endereço de **a**.

```
printf ("Conteudo apontado por B: %d\n", *b);  
printf ("Conteudo apontado por C: %d\n", **c);
```

```
Conteudo apontado por B: 10  
Conteudo apontado por C: 10
```


Atividade

Escreva um programa em C que crie uma variável inteira e um ponteiro que aponte para essa variável. Em seguida, crie um ponteiro para ponteiro que aponte para o ponteiro inicial. Modifique o valor da variável original utilizando o ponteiro para ponteiro e imprima o valor da variável antes e depois da modificação.

```
Valor original do ponteiro: a = 5  
Valor modificado do ponteiro: a = 15
```

03

Typedef

Typedef

É uma palavra-chave em C que permite criar um novo nome (**alias**) para um tipo de dado existente.

Facilita a leitura e a manutenção do código.

Typedef – Por que usar?

Clareza: Nomes mais descritivos ajudam a entender o propósito dos dados.

Facilidade de Modificação: Alterar o tipo de dados em um único lugar, se necessário.

Organização: Melhora a organização do código, especialmente em estruturas complexas.

Typedef

```
typedef tipo_original novo_nome;
```

```
struct cadastroAlunos {  
    char nome[50];  
    char matricula[20];  
};
```

```
typedef struct cadastroAlunos* Alunos;
```

Typedef

struct cadastroAlunos: Define uma estrutura que contém os dados de um aluno, incluindo o nome e a matrícula.

typedef struct cadastroAlunos* Alunos: Cria um novo tipo chamado Alunos, que é um ponteiro para a estrutura cadastroAlunos. Isso facilita a manipulação de múltiplos registros de alunos.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // Definição da estrutura
6  struct cadastroAlunos {
7      char nome[50];
8      char matricula[20];
9  };
10
11 // Criação do typedef
12 typedef struct cadastroAlunos* Alunos;
13
14 int main() {
15     // Alocação de memória para um aluno
16     Alunos aluno1 = (Alunos)malloc(sizeof(struct cadastroAlunos));
17
18     // Atribuição de valores
19     strcpy(aluno1->nome, "João Silva");
20     strcpy(aluno1->matricula, "20240001");
21
22     // Exibição dos dados
23     printf("Nome: %s\n", aluno1->nome);
24     printf("Matrícula: %s\n", aluno1->matricula);
25
26     // Liberação da memória
27     free(aluno1);
28
29     return 0;
30 }
```

Typedef

Definição da Estrutura: A estrutura cadastroAlunos é definida com dois campos: nome e matricula.

Alias com typedef: O typedef cria um tipo Alunos que é um ponteiro para struct cadastroAlunos.

Alocação de Memória: Usamos malloc para alocar memória para um novo registro de aluno.

Atribuição de Valores: Os valores são atribuídos aos campos da estrutura usando o operador ->, que é usado para acessar membros de uma estrutura através de um ponteiro.

Exibição dos Dados: Os dados do aluno são exibidos na tela.

Liberação de Memória: A memória alocada é liberada com free para evitar vazamentos de memória.

Typedef

Tipos básicos

```
typedef int inteiro;  
inteiro a = 10; // 'a' é do tipo 'int', mas usando o alias 'inteiro'
```

Ponteiros

```
typedef int* int_ptr;  
int_ptr p; // 'p' é um ponteiro para um inteiro
```

Estruturas

```
typedef struct {  
    char nome[50];  
    int idade;  
} Pessoa;  
  
Pessoa p1; // Declaração de uma variável do tipo 'Pessoa'
```

Atividade

Cleitinho TI, está precisando de um sistema de cadastro de funcionários. O sistema deve armazenar informações básicas, como: Nome (string), ID do funcionário (int) e salário (float).

Input

```
Cadastro de Funcionário
-----
Nome: Maurício
ID: 1010
Salário: 199,99
```

Output

```
Informações do Funcionário
-----
Nome: Maurício
ID: 1010
Salário: 199.00
```