Linguagem de Programação I

Aula 9 – Ponteiros

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

Agenda

Conceito de Ponteiros

Vantagens e Desvantagens

Inicializando Ponteiros

Passagem por Referência

Referências

Ponteiros

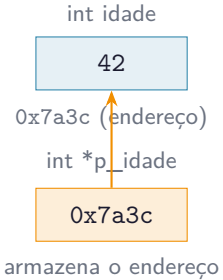
O recurso mais poderoso da linguagem C

O que é um Ponteiro?

Definição

Ponteiros são **variáveis que armazenam o endereço de memória** de outras variáveis. Dizemos que um ponteiro **"aponta"** para uma variável quando contém o endereço dela.

Ponteiros podem apontar para qualquer tipo: `int`, `float`, `double`, `char`, structs...



Declaração

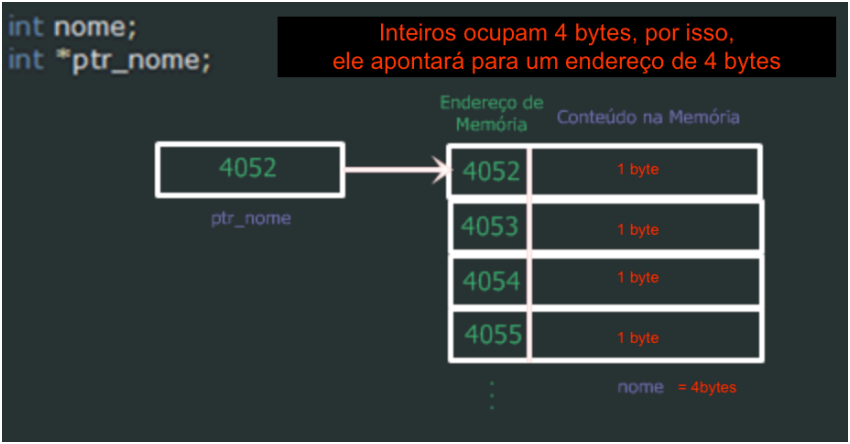
```
tipo *nome_ponteiro;
```

O asterisco `*` indica que a variável é um ponteiro.

Declaração de Ponteiros

```
int *nome;
struct alunoDaTurma *aluno1;
float *idade;
```

Ponteiro para Inteiro



Vantagens e Desvantagens

dos Ponteiros

Vantagens dos Ponteiros

▪ Acesso compartilhado

Várias partes do programa podem ter ponteiros apontando para a **mesma variável**, sem precisar percorrer estruturas completas como vetores ou matrizes. [Schildt e Mayer 1997]

▪ Alocação dinâmica

Com ponteiros, podemos **alocar apenas a memória que precisamos**, à medida que vamos precisando — sem desperdiçar espaço como num vetor de tamanho fixo.

▪ Dado sempre atualizado

Se o dado apontado for alterado, **todos os ponteiros** que apontam para aquele endereço verão o valor atualizado automaticamente — o endereço permanece o mesmo.

▪ Desvantagem

O uso de ponteiros exige **domínio de programação**. Qualquer erro pode fazer você perder o endereço apontado e, sem referência, a informação é **perdida na memória**.

Inicializando Ponteiros

O operador & e o operador *

O operador & — Obtendo o endereço

▪ Como inicializar um ponteiro?

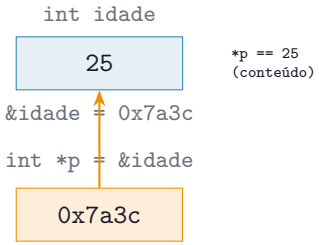
Os endereços de memória são determinados pelo compilador e realocados a cada execução — impossível saber de antemão.

A solução é usar o operador **&** para obter o endereço de uma variável e atribuí-lo ao ponteiro:

▪ Sintaxe

```
int idade = 25;
int *p = &idade;
```

Agora `p` contém o endereço de `idade`.



Inicializando Ponteiros — Exemplo

```
int contador=20;
int *ponteiro;
ponteiro = &contador;
```

Desreferenciando com * — Modificando o valor

O operador * (desreferência)

Uma vez que temos `p = &idade`, a expressão `*p` é **equivalente à própria variável** `idade`.

Para mudar o valor de `idade` para 70, basta: `*p = 70;`

```c
int idade, *ponteiro_idade;
idade = 40; //inicializando a idade com o valor 40
ponteiro_idade = &idade; // apontando ponteiro_idade para idade
*ponteiro_idade = 70; //modificando o valor da variável idade para 70
```

Passagem por Referência

Ponteiros como parâmetros de funções

Passagem de Structs por Referência

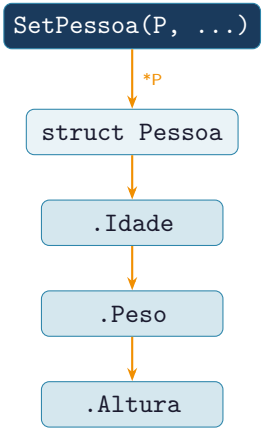
Como funciona?

Para passar uma struct por referência, passamos um **ponteiro para a struct** como parâmetro da função. Isso evita copiar toda a struct e permite modificá-la diretamente.

Duas formas de acessar campos

```
void SetPessoa(Pessoa *P, ...)  {
  (*P).Idade = idade; // (*ptr).campo
  P->Peso  = peso; // ptr->campo (atalho)
  P->Altura = altura;
}
```

`P->campo` é equivalente a `(*P).campo`



Passagem por Referência — Exemplo

```c
#include <stdio.h>
typedef struct pessoa{
    float Peso;
    int Idade;
    float Altura;
} Pessoa;

void ImprimePessoa(Pessoa P){
  printf("Idade: %d  Peso: %5.2f Altura: %5.2f\n", P.Idade, P.Peso, P.Altura);
}

void SetPessoa(Pessoa *P, int idade, float peso, float altura){
    (*P).Idade = idade;  // o campo pode ser acessado desta forma
    P->Peso = peso;      // ou desta
    P->Altura = altura;
}

int main(){
    Pessoa Joao;
    SetPessoa(&Joao, 15, 70.5, 1.75);
    ImprimePessoa(Joao);
    return 0;
}
```

Referências I

SCHILDT, H.; MAYER, R. *C completo e total*. [S.l.]: Pearson University, 1997. ISBN 9788534605953.