

# Lógica de Programação: notas de aula

Prof. Jonatha Costa

2025

# Organização

---

## 1 Ponteiros

- Soluções com ponteiros
- Subfunções com ponteiros
- Iteração e recursão

## 2 Erros Aparentes

# O que é um Ponteiro?

---

## Definição:

- Um ponteiro em C é uma variável que armazena o endereço de memória de outra variável. Em vez de armazenar um valor diretamente, ele aponta para a localização na memória onde o valor está armazenado.

## Exemplo de ponteiro

```
int x = 10; // Variável inteira  
int *p = &x; // Ponteiro que armazena o endereço de x
```

*p* é declarado como int \*, ou seja, um ponteiro para um inteiro.

O operador & (endereço de) é utilizado para obter o endereço de *x*, que é atribuído ao ponteiro *p*.

O operador \* é usado para acessar o valor armazenado no endereço ao qual o ponteiro aponta. Por exemplo:

```
printf("%d", *p); // Imprime o valor de x, ou seja, 10.
```

# Ponteiro com *string*

---

## Definição:

- Uma string é, na verdade, um *array* de caracteres.  
Seja o trecho de código:  
`char *str = "Hello"; // str armazena o endereço do 'H'`
- O ponteiro **char** pode apontar para:
  - ① O primeiro caractere da string:  
`printf("%c", *str); // Imprime 'H'`
  - ② Um caractere de interesse do programador:  
`printf("%c", *(str + 1)); // Imprime 'e'`
  - ③ A string completa:  
`printf("%s", str); // Imprime 'Hello'`

# Variável vs Ponteiro

```
int x = 10;  
int *p;  
p = &x;
```

- x guarda 10.
- p guarda o endereço de x.
- \*p acessa o valor de x (10).

# Exemplo Completo

```
#include <stdio.h>
int main() {
int x = 10;
int *p;
p = &x;

printf("Valor de x: %d\n", x);
printf("Endereço de x: %p\n", &x);
printf("Valor de p: %p\n", p);
printf("Valor apontado por p: %d\n", *p);
return 0;
}
```

# Ponteiro Nulo e novos valores nos ponteiros

```
#include <stdio.h>
int main(){
int *p = NULL;
printf("Valor inicial do ponteiro: %p\n", p);
int x=10; p = &x;
printf("Novo valor apontado por p: %d\n", *p); // imprime 10
printf("Endereco atual de p: %p\n", p);
*p = 20; // Modifica o valor de x pelo ponteiro
printf("Novo valor apontado por p: %d\n", *p); // imprime 20
printf("Endereco atual de p: %p\n", p);
return 0;}
```

- `int *p = NULL` - Ponteiro nulo: indica que o ponteiro não aponta para lugar nenhum.  
Usado para segurança na inicialização.
- `*p = 20` - Modifica diretamente o valor de x para 20 através de p.

# Ponteiros e Vetores

```
int v[3] = {1, 2, 3};  
int *p = v;  
  
printf("%d\n", *p);  
printf("%d\n", *(p+1));  
printf("%d\n", *(p+2));
```

- Ponteiro caminha pelo vetor.

# Resumo

Expressão	Significado
int *p	Declara um ponteiro para int
p = &x	p recebe o endereço de x
*p	Valor apontado por p
&x	Endereço de x

Sempre leia:

- \*p como "valor apontado por p"
- &x como "endereço de x"

# Exercícios

---

Reconstrua os exercícios anteriores utilizando ponteiros.

# Organização

---

## 1 Ponteiros

Soluções com ponteiros

Subfunções com ponteiros

Iteração e recursão

## 2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Receber um nota e verificar aprovação com ponteiros

```
#include <stdio.h>
int main(){
int nota;
int *p;
p=&nota; // Endereco
printf("Informe uma nota: "); scanf("%d",p);
if(*p>=7) printf("Aprovado."); // *p: conteudo do endereco
else printf("Nao aprovado!");
return 0;
}
```

# Receber duas notas e verificar aprovação

## Solução com ponteiros sem vetores

```
#include <stdio.h>
int main(){
int nota1,nota2;
int *p1,*p2; // Definindo dois ponteiros
p1=&nota1;p2=&nota2; // Direcionando os ponteiros
float media;
printf("Informe a nota 1: ");scanf("%d",p1);
printf("Informe a nota 2: ");scanf("%d",p2);
media = (*p1 + *p2)/2.0; // Acessando o conteudo dos enderecos
if(media>=7) printf("Aprovado!");
else printf("Nao aprovado!");
return 0;
}
```

# Receber duas notas e verificar aprovação

## Solução com ponteiros e utilizando vetores

```
#include <stdio.h>
int main(){
int notas[2];
int *p;
p = notas;    // O ponteiro aponta para o inicio do vetor notas.
float media;
for(int i=0;i<2;i++)
{
printf("Informe a nota %d: ",i+1);scanf("%d",p+i);
}
media = (*(p+0) + *(p+1)) / 2.0;
if(media>=7) {printf("Aprovado!");}
else {printf("Nao aprovado!");}
return 0;
}
```

# Maior número entre 10

## Solução com ponteiros — substituindo &a e a[i]

```
#include <stdio.h>

int main() {
    int v[10], *p, maior;
    printf("Digite 10 numeros:\n");
    for (p = v; p < v + 10; p++) {
        // Substituindo convencional scanf("%d", &v[i]); por:
        scanf("%d", p); // p aponta para cada posicao do vetor
    }
    maior = *v; // acessa o valor da primeira posicao
    for (p = v + 1; p < v + 10; p++) {
        // Substitui: if (v[i] > maior)
        if (*p > maior)
            maior = *p;
    }
    printf("Maior numero: %d\n", maior);
    return 0;
}
```



# Maior, menor e diferença entre 10 números

## Solução com ponteiros

```
#include <stdio.h>

int main() {
    int v[10], *p, maior, menor;
    printf("Digite 10 numeros:\n");
    for (p = v; p < v + 10; p++) {
        // Substitui: scanf("%d", &v[i]);
        scanf("%d", p);
    }

    maior = menor = *v; // Substitui: v[0]
    for (p = v + 1; p < v + 10; p++) {
        // Substitui: v[i]
        if (*p > maior) maior = *p;
        if (*p < menor) menor = *p;
    }
    printf("Maior: %d\nMenor: %d\nDiferenca: %d\n", maior, menor, maior - menor);
    return 0;
}
```

# Números ímpares com ponteiros — substituindo índices

```
#include <stdio.h>

int main() {
int v[10], *p;

printf("Digite 10 numeros:\n");
for (p = v; p < v + 10; p++) {
// Substitui: scanf("%d", &v[i]);
scanf("%d", p);
}
printf("Impares:\n");
for (p = v; p < v + 10; p++) {
// Substitui: if (v[i] % 2 != 0)
if (*p % 2 != 0)
printf("%d ", *p); // Substitui: v[i]
}
return 0;
}
```

# Números primos usando ponteiros — sem índices

```
#include <stdio.h>
int main() {
int v[10], *p, j, primo;
printf("Digite 10 numeros:\n");
for (p = v; p < v + 10; p++) {
// Substitui: scanf("%d", &v[i]);
scanf("%d", p);}
printf("Primos:\n");
for (p = v; p < v + 10; p++) {
if (*p < 2) continue;
primo = 1;
// Substitui: for (j = 2; j <= v[i]/2; j++)
for (j = 2; j <= *p / 2; j++) {
if (*p % j == 0) {
primo = 0; break;
}}
if (primo) printf("%d ", *p); // Substitui: v[i]
}}
```

# Alocação dinâmica com ponteiros — malloc e aritmética

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *v, i, maior;
    v = malloc(10 * sizeof(int)); // Aloca dinamicamente espaço para 10
        inteiros
    printf("Digite 10 números:\n");
    for (i = 0; i < 10; i++) {
        // Substitui: &v[i]
        scanf("%d", v + i); // v + i -> endereço da posição i}
    maior = *v; // Substitui: v[0]
    for (i = 1; i < 10; i++) {
        if (*(v + i) > maior)// Substitui: v[i]
            maior = *(v + i); }
    printf("Maior: %d\n", maior);
    free(v); // Libera memória alocada
    return 0;}
```

# Organização

---

## 1 Ponteiros

Soluções com ponteiros

Subfunções com ponteiros

Iteração e recursão

## 2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Subfunção: void func(int\*) (simula retorno via ponteiro)

## Características:

- Não recebe parâmetros por valor.
- Retorna valor ao programa principal por meio de ponteiro.

## Definição:

```
void obterAno(int *p) {  
    *p = 2025;  
}
```

## Chamada no main():

```
1 int main() {  
2     int ano;  
3     obterAno(&ano);  
4     printf("%d\n", ano);  
5     return 0;  
6 }
```

# Subfunção: void func(int\*), void func(int\*, int\*)

## Características:

- Recebe valor por ponteiro.
- Pode imprimir ou retornar resultado por ponteiro.

## Definição:

```
void exibeQuadrado(int *a) {  
    printf("%d\n", (*a) * (*a));  
}  
  
void dobro(int *a, int *b) {  
    *b = 2 * (*a);  
}
```

## Chamada no main():

```
1 int main() {  
2     int a = 5, b;  
3     exibeQuadrado(&a);  
4  
5     dobro(&a, &b);  
6     printf("%d\n", b);  
7  
8     return 0;  
9 }
```

# Subfunção com vetores: void func(int\*, int\*, int)

## Características:

- Vetores são passados como ponteiros.
- Saída armazenada por referência.

## Definição:

```
void quadrado(int *in, int *out,
    int n) {
for (int i = 0; i < n; i++) {
    out[i] = in[i] * in[i];
}
}
```

## Chamada no main():

```
1 int main() {
2     int a[] = {1, 2, 3};
3     int b[3];
4
5     quadrado(a, b, 3);
6
7     for (int i = 0; i < 3; i++)
8         printf("%d ", b[i]);
9
10    return 0;
11 }
```

# Subfunção com matrizes: void func(int\*\*, int, int)

## Características:

- Passagem de matriz como ponteiro de ponteiros.
- Exige alocação dinâmica ou vetor de ponteiros.

## Definição:

```
void dobrarMatriz(int **m,
      int lin, int col) {
for (int i = 0; i < lin; i
    ++
) {
for (int j = 0; j < col; j
    ++
) {
    m[i][j] *= 2;
}}}
```

## Chamada no main():

```
1 int main() {
2     int lin = 2, col = 3; int *mat[2];
3     for (int i = 0; i < lin; i++)
4         mat[i] = malloc(col * sizeof(int));
5     for (int i = 0; i < lin; i++)
6         for (int j = 0; j < col; j++)
7             mat[i][j] = i * col + j + 1;
8     dobrarMatriz(mat, lin, col);
9     for (int i = 0; i < lin; i++) {
10        for (int j = 0; j < col; j++)
11            printf("%d ", mat[i][j]);
12            printf("\n");
13    }
14    return 0;
15 }
```

# Subfunção com matrizes: void func(int\*\*, int, int)

## Características:

- Passagem de matriz como um único bloco contínuo de memória
- Exige alocação dinâmica sem vetor de ponteiros.

## Definição:

```
void dobrarMatriz(int *m,
    int lin, int col) {
for (int i = 0; i < lin; i
    ++ ) {
for (int j = 0; j < col; j
    ++ ) {
        m[i*col+j] *= 2;
            // substitui m[i]
            ][j]
}}}
```

## Chamada no main():

```
1 int main() {
2     int lin = 2, col = 3, int *mat;
3     *mat = malloc(lin*col*sizeof(int));
4     for (int i = 0; i < lin; i++)
5         for (int j = 0; j < col; j++)
6             mat[i][j] = i * col + j + 1;
7     dobrarMatriz(mat, lin, col);
8     for (int i = 0; i < lin; i++) {
9         for (int j = 0; j < col; j++)
10            printf("%d ", mat[i][j]);
11            printf("\n");
12    }
13 }
```

# Tabela-resumo: Subfunções com Ponteiros em C

Tipos de subfunções em linguagem C (com ponteiros)

Tipo de função	Definição	Chamada	Características
void func(void)	void saudacao(void)	saudacao();	Sem parâmetros, sem retorno. Executa apenas uma ação.
void func(int*)	void obterAno(int *p)	obterAno(&x);	Retorna valor via ponteiro. Simula retorno escalar.
void func(int*)	void imprime(int *p)	imprime(&x);	Recebe endereço de um valor, sem retorno.
void func(int*, int*)	void dobro(int *x, int *y)	dobro(&a, &b);	Recebe valor via ponteiro e armazena resultado via ponteiro. Simula passagem por valor com retorno.
void func(int*, int)	void dobraVetor(int *v, int n)	dobraVetor(v, 3);	Passagem de vetor como ponteiro. Permite alterar o conteúdo original.
void func(int**, int, int)	void dobraMatriz(int **m, int lin, int col)	dobraMatriz(mat, 3, 3);	Passagem de matriz com ponteiro para ponteiro. Requer alocação dinâmica ou vetor de ponteiros.

# Organização

---

## 1 Ponteiros

Soluções com ponteiros

Subfunções com ponteiros

Iteração e recursão

## 2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Introdução

---

- Há dois conceitos fundamentais em programação de algoritmos:
  - ① Recursividade
  - ② Iteratividade
- Ambos podem resolver problemas semelhantes, mas possuem diferenças conceituais, de implementação e de desempenho.

# Questão: Como calcular fatorial?

---

- Exemplo: Calcular  $n!$
- Vamos comparar três implementações:

## Iterativo direto

```
int main()
{
    int p=1;
    for (int i=1;i<=n;i++)
    {
        p*=i;
    }
    printf("%d",p);
    return 0;
}
```

### Função Iterativa

```
int fatorial(int k)
{
    int p=1;
    for (int i=1;i<=k;i++)
        p*=i;
    return p;
}
```

```
int main()
{
    int fac = fatorial(n);
    printf("%d",fac);
    return 0;
}
```

### Função Recursiva

```
int fatorial(int k)
{
    if (k==0 || k==1)
        return 1;
    else
        return k * fatorial(k-1);
}
```

```
int main()
{
    int fac = fatorial(n);
    printf("%d",fac);
    return 0;
}
```

# Conclusão Comparativa

- **Iterativo direto:** simples, rápido, menos modular.
- **Função Iterativa:** modularidade e eficiência.
- **Função Recursiva:** elegante conceitualmente, mas usa pilha de chamadas<sup>1</sup>, consumindo mais memória.

---

<sup>1</sup> **Pilha de chamadas:** Cada chamada recursiva cria um novo registro na **pilha de execução** do programa, armazenando variáveis locais e o ponto de retorno. Em grandes profundidades, isso pode causar *stack overflow*.



# Organização

---

1 Ponteiros

2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Organização

---

## 1 Ponteiros

Soluções com ponteiros

Subfunções com ponteiros

Iteração e recursão

## 2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Problema no Código?

Considere o *script* a seguir e analise se há erros!

Código contém erros?

```
#include <stdio.h>
int main() {
    int a = 5;
    char b;
    b = (a > 7) ? "Aprovado!" : "Nao aprovado!";
    printf("%c", b);
    return 0;
}
```

# Problema no Código?

Considere o *script* a seguir e analise se há erros!

Código contém erros?

```
#include <stdio.h>
int main() {
    int a = 5;
    char b;
    b = (a > 7) ? "Aprovado!" : "Nao aprovado!";
    printf("%c", b);
    return 0;
}
```

Erro:

- b é **char**, mas o operador ternário retorna **strings**.
- %c espera um único caractere, o que causa erro.

# Solução 1

## Código Corrigido

```
#include <stdio.h>
int main() {
int a = 5;
char *b;
b = (a > 7) ? "Aprovado!" : "Nao aprovado!";
printf("%s", b);
return 0;
}
```

## Mudanças:

- b alterado para um **ponteiro char \***
- Utilizando %s no printf para imprimir strings.

# Código Corrigido com Ponteiro: Solução 01

## Código corrigido com ponteiro

```
#include <stdio.h>
int main() {
    int a = 5;
    char *b;
    b = (a > 7) ? "Aprovado" : "Nao aprovado";
    printf("%s", b);
}
```

### Como funciona no compilador?

- 1 As *strings* literais "Aprovado" e "Não Aprovado" são armazenadas em uma área especial da memória, chamada seção de texto ou seção de dados constantes.
- 2 Essas strings são armazenadas de forma estática, ou seja, são criadas no momento da compilação e permanecem na memória durante a execução do programa.
- 3 O ponteiro *b* então recebe o endereço da *string* "Aprovado" (no caso da condição ser verdadeira). O endereço de memória dessa *string* é atribuído a *b*, e o ponteiro pode ser usado para acessar o conteúdo dessa *string*.
- 4 Quando a função `printf("%s", b)` é chamada, ela imprime o conteúdo da *string* que o ponteiro *b* aponta. Como *b* aponta para a string "Aprovado", o comando `printf` imprimirá "Aprovado".

# Código Corrigido com Ponteiro: Solução 02

## Como funciona no compilador?

### Código Corrigido sem declarar o ponteiro

```
#include <stdio.h>
int main() {
    int a = 5;
    printf("%s", (a > 7) ?
        "Aprovado" : "Não
        aprovado");
    return 0;
}
```

- 1 As *strings* literais, como "Aprovado!" e "Não aprovado!", são tratadas pelo compilador como *arrays* de caracteres constantes, mas, **internamente**, elas são representadas como **ponteiros** para o primeiro caractere da *string*.
- 2 Quando o compilador encontra "Aprovado!", ele sabe que isso é um ponteiro para o primeiro caractere da *string* ('A'), e esse ponteiro é o que é utilizado quando a *string* é passada para funções como `printf`.
- 3 A função `printf` espera um argumento do tipo `char*` para o especificador de formato `%s`, que indica uma *string*. A expressão ternária (`a > 7`) ? "Aprovado": "Não aprovado" retorna diretamente um ponteiro para a *string* apropriada.
- 4 Portanto, não é necessário declarar um ponteiro adicional, porque o tipo de "Aprovado" ou "Não aprovado" já é `char*`, ou seja, um ponteiro para o primeiro caractere da *string*. Quando a expressão ternária é avaliada, ela simplesmente retorna esse ponteiro, que é então passado para o `printf`.

# Organização

---

## 1 Ponteiros

Soluções com ponteiros

Subfunções com ponteiros

Iteração e recursão

## 2 Erros Aparentes

Um aparente erro e solução

Exercícios com ternários e ponteiros

# Exercícios de ternários e ponteiros: bloco 1

- 1 **Operador Ternário com Números:** Escreva um programa que utilize o operador ternário para verificar se um número é positivo, negativo ou zero. O programa deve imprimir a mensagem correspondente:
  - Se o número for positivo, deve imprimir “Número positivo”.
  - Se o número for negativo, deve imprimir “Número negativo”.
  - Se o número for zero, deve imprimir “Número zero”.
- 2 **Uso de #define para Definir Constantes:** Utilize a diretiva `#define` para definir uma constante para o valor de PI e calcule a área de um círculo de raio 5. A fórmula para calcular a área de um círculo é:

$$A = \pi \times r^2$$

Em que `r` é o raio do círculo.

- 3 **Ponteiro para String:** Implemente um programa que utilize um ponteiro para armazenar e imprimir uma string. O programa deve armazenar a string “Bem-vindo ao C!” em um ponteiro de caractere e imprimi-la utilizando `printf`.
- 4 **Operador Ternário com Ponteiros:** Escreva um programa que, utilizando o operador ternário, decida qual das duas variáveis ponteiro `ptr1` ou `ptr2` deve ser utilizada com base no valor de um número inteiro `a`. Se `a > 10`, o programa deve usar `ptr1`, caso contrário, `ptr2`. Ambas as variáveis ponteiro devem apontar para um valor inteiro.

# Exercícios de ternários e ponteiros: bloco 2

---

- 5 **Uso de #define para Função de Cálculo:** Utilizando `#define`, crie uma macro chamada `SQUARE(x)` que calcula o quadrado de um número `x`. Use essa macro para calcular o quadrado de um número inserido pelo usuário e imprima o resultado.
- 6 **Uso de Ponteiros para Funções:** Escreva uma função que receba um ponteiro para um número inteiro e altere seu valor para 100. No programa principal, crie uma variável inteira, passe seu ponteiro para a função e imprima o valor alterado.
- 7 **Operador Ternário e Arrays:** Dado um array de inteiros, escreva um programa que utilize o operador ternário para verificar se o primeiro elemento é maior que 10. Se for, imprima "Maior que 10", caso contrário, imprima "Menor ou igual a 10".
- 8 **Manipulação de Ponteiros em Arrays:** Crie um programa que utilize ponteiros para manipular um array de inteiros. O programa deve imprimir os elementos do array, acessando-os através de ponteiros.
- 9 **Estrutura com Ponteiros:** Defina uma estrutura chamada `Pessoa` com os campos `nome` e `idade`. Crie um ponteiro para uma variável do tipo `Pessoa`, atribua valores a esses campos e imprima as informações.
- 10 **Uso de #define para Definir Tipos:** Utilize a diretiva `#define` para criar um tipo de dado `float32`, que seja equivalente a `float`. Em seguida, crie uma variável desse tipo e imprima seu valor.

# Referências

---

- **Veja material auxiliar em:** <https://github.com/jonathacosta-IA/PL>
  - *Slides* em: [https://github.com/JonathaCosta-IA/PL/tree/main/A-PL\\_Slides](https://github.com/JonathaCosta-IA/PL/tree/main/A-PL_Slides)
  - Códigos em: [https://github.com/JonathaCosta-IA/PL/tree/main/B\\_PL\\_Codes](https://github.com/JonathaCosta-IA/PL/tree/main/B_PL_Codes)
- **Referência basilares**
  - PUD da Disciplina de Lógica de Programação
  - DEITEL, P. J.; DEITEL, H. M. C: Como programar. 6. ed. São Paulo: Pearson, 2011. E-book. Disponível em: <https://plataforma.bvirtual.com.br>. Acesso em: 28 jun. 2025.