Etapa 8 — Recursão (Recursion)

Trilha: Lógica de Programação

O que é Recursão?

- **Recursão (Recursion)** é quando uma **função chama a si mesma**.
- Permite resolver problemas dividindo-os em **subproblemas menores**.
- Necessita de uma **condição de parada (base case)** para evitar loops infinitos.
- Exemplo prático: cálculo do **fatorial**.

Estrutura de uma Função Recursiva

- **Pseudocódigo:**

```
funcao nome(param)
    se condicao_base entao
        retorna valor
    senao
        // chamada recursiva
        retorna nome(novo_param)
    fimse
fimfuncao
```

- Deve sempre ter uma **condição base** e uma **chamada recursiva**.

Exemplo: Fatorial Recursivo

- **Pseudocódigo:**

```
funcao fatorial(n)
    se n = 0 entao
        retorna 1
    senao
        retorna n * fatorial(n - 1)
    fimse
fimfuncao
```

- **Cálculo de fatorial(3):**
  → 3 × fatorial(2)
  → 3 × (2 × fatorial(1))
  → 3 × (2 × (1 × fatorial(0)))
  → 3 × 2 × 1 × 1 = 6

Como Funciona a Pilha de Execução (Call Stack)

- Cada chamada recursiva é **armazenada na pilha (stack)**.
- Quando atinge o caso base, as chamadas começam a **retornar**.
- A pilha desempilha as funções em ordem inversa à chamada (LIFO — *Last In, First Out*).
- **Atenção:** recursão excessiva pode causar **estouro de pilha (stack overflow)**.

Caso Base (Base Case)

- É a **condição que encerra a recursão**.
- Sem ela, a função nunca para.
- Deve ser simples e retornar um valor direto.
- **Exemplo:**

```
funcao contagem_regressiva(n)
    se n = 0 entao
        escreva "Fim!"
    senao
        escreva n
        contagem_regressiva(n - 1)
    fimse
fimfuncao
```

Exemplos de Aplicações de Recursão

1. **Matemática**: fatorial, sequência de Fibonacci.
2. **Estruturas de dados**: percorrer árvores e grafos.
3. **Busca e ordenação**: binary search, merge sort, quick sort.
4. **Problemas lógicos**: Torre de Hanói, labirintos, permutações.

Exemplo: Fibonacci Recursivo

- **Pseudocódigo:**

```
funcao fibonacci(n)
    se n <= 1 entao
        retorna n
    senao
        retorna fibonacci(n - 1) + fibonacci(n - 2)
    fimse
fimfuncao
```

- **Saída:**

fibonacci(5) → 5

(0, 1, 1, 2, 3, 5, 8...)

Diferença entre Recursão e Iteração

Característica	**Recursão**	**Iteração**
Estrutura	Função chama a si mesma	Usa loops (for, while)
Controle	Caso base	Condição de parada
Memória	Usa pilha (stack)	Usa variável de controle
Leitura	Mais conceitual	Mais prática e direta

- **Dica:** muitas soluções recursivas podem ser transformadas em iterativas.

Vantagens e Desvantagens

- **Vantagens:**
  - Código mais **curto e elegante**.
  - Natural para resolver problemas divididos em partes menores.
- **Desvantagens:**
  - Maior uso de **memória** (pilha).
  - Pode ser mais **lento** que a iteração.
  - Difícil de depurar se for mal implementado.

Boas Práticas

- Sempre defina **caso base** claro.
- Teste com **valores pequenos** antes de escalar.
- Prefira **recursão de cauda (tail recursion)** quando possível.
- Use **comentários explicativos** para facilitar leitura.

Exemplo Avançado: Torre de Hanói

- **Pseudocódigo:**

```
procedimento hanoi(n, origem, destino, auxiliar)
    se n = 1 entao
        escreva "Mover disco 1 de", origem, "para", destino
    senao
        hanoi(n - 1, origem, auxiliar, destino)
        escreva "Mover disco", n, "de", origem, "para", destino
        hanoi(n - 1, auxiliar, destino, origem)
    fimse
fimprocedimento
```

- Resolve o problema com **3 pinos e n discos**, aplicando o mesmo padrão recursivo.

Recursão em Estruturas de Dados

- Usada para **percorrer árvores (trees)** e **grafos (graphs)**.
- Exemplo clássico: traversal em árvore binária.
- **Exemplo:**

```
procedimento percorrer_no(no)
    se no ≠ nulo entao
        percorrer_no(no.esquerda)
        escreva no.valor
        percorrer_no(no.direita)
    fimse
fimprocedimento
```

Dicas de Depuração (Debugging)

- Use **prints** para acompanhar chamadas.
- Identifique quando e como o caso base é atingido.
- Desenhe o **fluxo da pilha** em papel se necessário.
- Se o código travar, verifique:
  - Caso base correto?
  - Parâmetros estão diminuindo?

Boas Práticas Finais

- Cada chamada recursiva deve **aproximar-se do caso base**.
- Teste para **n = 0 ou n = 1** sempre.
- Prefira recursão **somente quando** ela for mais clara que um loop.

“Recursão é poderosa, mas requer disciplina.”

Conclusão

- Recursão é uma **estratégia de repetição conceitual**, onde a função resolve versões menores do mesmo problema.
- É fundamental em **estruturas de dados**, **buscas**, e **algoritmos matemáticos**.
- Saber identificar quando aplicá-la é um sinal de **maturidade em lógica**.
- Dominar recursão abre caminho para entender **estruturas complexas** e **algoritmos avançados**.