Etapa 8 — Recursão (Recursion)

Trilha: Lógica de Programação
Tema: Funções que chamam a si mesmas

1. Introdução

A **recursão** é uma das ideias mais poderosas e elegantes da lógica de programação.
Ela ocorre quando uma **função chama a si mesma** para resolver um problema menor, até chegar em uma **condição base**, que interrompe as chamadas.

Em resumo:

> "Um problema é resolvido resolvendo versões menores de si mesmo."

A recursão é muito usada em **matemática**, **algoritmos de busca e ordenação**, e **estruturas de dados** como **árvores e grafos**.

2. Estrutura de uma Função Recursiva

Toda função recursiva precisa ter **duas partes obrigatórias**:

1. **Caso base (base case)** → é a condição que **encerra a recursão**.
2. **Chamada recursiva** → é a parte onde a função **chama a si mesma** com parâmetros diferentes.

Pseudocódigo geral:

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

Dica:
Cada chamada deve **aproximar-se do caso base**, caso contrário o programa entra em **loop infinito**.

3. Exemplo Clássico: Fatorial

O **fatorial de um número** (n!) é o produto de todos os inteiros de n até 1.

Definição matemática:

- 0! = 1
- n! = n × (n – 1)!

Pseudocódigo recursivo:

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

Exemplo de execução:

fatorial(3)

= 3 * fatorial(2)

= 3 * (2 * fatorial(1))

= 3 * (2 * (1 * fatorial(0)))

= 3 * 2 * 1 * 1 = 6

Cada chamada é empilhada até atingir o caso base (`n = 0`), e então os resultados são desempilhados.

4. Pilha de Execução (Call Stack)

Quando uma função chama outra, o computador armazena as informações na **pilha de execução**.

Na recursão:

- Cada chamada **fica na pilha** até que o caso base seja alcançado.
- Quando o caso base é atingido, as funções começam a **retornar** os resultados na ordem inversa.

Se o caso base **nunca for atingido**, ocorre um erro de **estouro de pilha (stack overflow)**.

5. Caso Base (Base Case)

É o ponto de parada da recursão.
Sem ele, a função se chamaria infinitamente.

Exemplo: Contagem regressiva

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

Saída:

5

4

3

2

1

Fim!

6. Aplicações Comuns de Recursão

A recursão é usada em diversos contextos da computação:

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

7. Exemplo: Sequência de Fibonacci

A sequência de Fibonacci é formada pela soma dos dois números anteriores.

Definição matemática:

- f(0) = 0
- f(1) = 1
- f(n) = f(n - 1) + f(n - 2)

Pseudocódigo:

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

Saída:
`fibonacci(6)` → 8
(0, 1, 1, 2, 3, 5, 8, 13...)

8. Recursão vs Iteração

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

Conclusão:
Recursão é mais elegante para problemas que se dividem naturalmente em subproblemas.
Porém, loops são mais eficientes em termos de memória.

9. Vantagens e Desvantagens

Vantagens:

- Código mais curto e intuitivo.
- Facilita resolver problemas complexos com subdivisões.
- Base para algoritmos poderosos (buscas, árvores, etc.).

Desvantagens:

- Maior consumo de memória (pilha).
- Pode ser mais lenta que soluções iterativas.

- Difícil de depurar se não houver caso base bem definido.

10. Exemplo Avançado: Torre de Hanói

Problema clássico da recursão.
Mover **n discos** de um pino de origem para um de destino, usando um pino auxiliar, sem colocar um disco maior sobre um menor.

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
```

11. Recursão em Estruturas de Dados

Recursão é muito usada para percorrer **árvores binárias** e **grafos**.

Exemplo: Percurso em ordem (in-order traversal)

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

🧰 12. Dicas de Depuração

- Use mensagens (`escreva`) para acompanhar o fluxo.
- Desenhe o **fluxo da pilha** para entender o caminho das chamadas.
- Sempre verifique:
    - Há caso base?
    - Os parâmetros estão **diminuindo**?
    - A função realmente **retorna algo**?

13. Boas Práticas

- Sempre defina um **caso base** simples e seguro.
- Teste com **valores pequenos** primeiro.
- Documente o propósito da recursão.
- Prefira **recursão de cauda (tail recursion)** quando possível.
- Use-a quando realmente **simplificar o raciocínio** do problema.

14. Conclusão

- A recursão é uma técnica essencial que permite resolver problemas de forma **dividida e elegante**.

- Apesar de exigir atenção com a **pilha de execução** e o **caso base**, é uma ferramenta poderosa.

- Saber identificar quando aplicá-la é um marco de **maturidade lógica e algorítmica**.

- Na próxima etapa, entraremos em **Estruturas de Dados Simples**, onde aplicaremos recursão em listas, pilhas e filas.

15. Exercícios Práticos

1. Escreva uma função recursiva que calcule a soma dos números de 1 até n.

2. Implemente uma contagem regressiva recursiva de n até 1.

3. Crie uma função recursiva que exiba os elementos de um vetor.

4. Faça uma função recursiva que calcule o valor de base^expoente.

5. Desenvolva um programa recursivo para inverter uma string.

Gabarito

1. **Soma de 1 até n:**

```
funcao soma(n)
  se n = 0 entao retorna 0
  senao retorna n + soma(n - 1)
fimfuncao
```

2. **Contagem regressiva:**

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

3. **Exibir vetor:**

```
procedimento exibir(vetor, i)
  se i < tamanho(vetor) entao
     escreva vetor[i]
     exibir(vetor, i + 1)
  fimse
```

```
fimprocedimento
```

4. **Potenciação:**

```
funcao potencia(b, e)
  se e = 0 entao retorna 1
  senao retorna b * potencia(b, e - 1)
fimfuncao
```

5. **Inverter string:**

```
funcao inverter(texto)
  se comprimento(texto) = 0 entao retorna ""
  senao retorna inverter(subtexto(texto, 2, fim)) + texto[1]
fimfuncao
```

16. Exercícios de Múltipla Escolha

1. Conceito básico de recursão

O que caracteriza uma função recursiva?

a) Uma função que chama outras funções auxiliares.
b) Uma função que chama a si mesma durante sua execução.
c) Uma função que utiliza apenas estruturas de repetição (*loops*).
d) Uma função que não possui retorno.

2. Caso base

Por que o **caso base (base case)** é essencial em uma função recursiva?

a) Porque ele define a condição que encerra as chamadas recursivas.
b) Porque ele permite que o código use variáveis globais.
c) Porque ele é o primeiro passo da recursão, não o último.
d) Porque ele garante que a função use menos memória.

3. Execução da recursão

Qual é a **principal estrutura de memória** usada para armazenar as chamadas recursivas?

a) Vetor (*array*)
b) Lista encadeada (*linked list*)
c) Pilha (*stack*)
d) Fila (*queue*)

4. Exemplo de fatorial

Considere o seguinte pseudocódigo:

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

Qual será o resultado de `fatorial(3)`?

a) 3
b) 5
c) 6
d) 9

5. Comparando recursão e iteração

Em relação à **recursão e iteração**, qual das alternativas é **verdadeira**?

a) A recursão sempre usa menos memória do que a iteração.
b) A recursão não precisa de uma condição de parada.
c) A recursão usa a pilha de chamadas para armazenar estados intermediários.
d) A iteração é obrigatoriamente mais lenta que a recursão.

6. Erros comuns

O que acontece se uma função recursiva **não possuir um caso base**?

a) Ela retorna um valor indefinido.
b) Ela entra em um loop infinito e pode causar *stack overflow*.

c) Ela termina imediatamente com erro de sintaxe.
d) Ela é convertida automaticamente em uma função iterativa.

7. Fibonacci

Dado o pseudocódigo:

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

Qual o valor de `fibonacci(5)`?

a) 3
b) 4
c) 5
d) 8

8. Aplicações práticas

Em qual dos seguintes problemas a **recursão** é mais naturalmente aplicada?

a) Calcular o total de vendas em um vetor.
b) Ordenar uma lista com *merge sort*.
c) Somar dois números simples.
d) Exibir uma mensagem em tela.

9. Eficiência

Qual é uma desvantagem da recursão?

a) Maior clareza do código.
b) Menor uso de variáveis.
c) Maior uso de memória devido à pilha de chamadas.
d) Execução garantida mais rápida.

10. Recursão indireta

O que é **recursão indireta**?

a) Quando uma função chama a si mesma diretamente.
b) Quando uma função é chamada várias vezes seguidas.
c) Quando duas ou mais funções se chamam mutuamente.
d) Quando a recursão é feita dentro de um laço *for*.

GABARITO

Questão	Resposta
1	b
2	a
3	c
4	c
5	c
6	b
7	c
8	b
9	c
10	c