Recursividade

Revisão Estrutura de Dados I

📘 Recursividade em Programação

🔹 Conceito

- A **recursividade** é uma técnica de programação na qual uma função chama a si mesma, direta ou indiretamente, para resolver um problema.
- Esse paradigma é especialmente útil em problemas que podem ser divididos em **subproblemas menores de mesma natureza**.
- Exemplo clássico: o cálculo do **fatorial** de um número, em que

$$n! = n \times (n-1)! \quad \text{com } 0! = 1$$

📘 Recursividade em Programação

🔹 **Para que serve**

- A recursividade é indicada quando:
- O problema pode ser **dividido em subproblemas menores e similares**.
- Há uma **definição matemática ou lógica recursiva**, como em árvores, grafos, fatorial, sequência de Fibonacci etc.
- Estruturas de dados naturalmente recursivas (como árvores e listas encadeadas) precisam ser percorridas.

📘 Recursividade em Programação

🔹 Vantagens

- **Clareza e elegância**: código mais limpo e próximo da formulação matemática.
- **Facilidade de implementação** em problemas naturalmente recursivos (árvores, grafos, divisão e conquista).
- **Redução de complexidade lógica**: evita uso de muitos loops e variáveis auxiliares.

📘 Recursividade em Programação

- 🔹 **Desvantagens**

- **Uso maior de memória**, pois cada chamada recursiva empilha variáveis na **stack**.
- **Menor eficiência** em alguns casos, podendo ser mais lento que soluções iterativas.
- **Risco de estouro de pilha (stack overflow)** se a condição de parada não for bem definida.
- **Dificuldade de depuração**: nem sempre é simples acompanhar chamadas recursivas.

📘 Recursividade em Programação

- **Exemplo 1 – Cálculo do Fatorial**

```c
#include <stdio.h>

// Função recursiva para calcular fatorial
int fatorial(int n) {
    if (n == 0 || n == 1) {
        return 1; // Caso base: fatorial de 0 ou 1 é 1
    }
    return n * fatorial(n - 1); // Passo recursivo
}

int main() {
    int num = 5;
    printf("Fatorial de %d = %d\n", num, fatorial(num));
    return 0;
}
```

📘 Recursividade em Programação

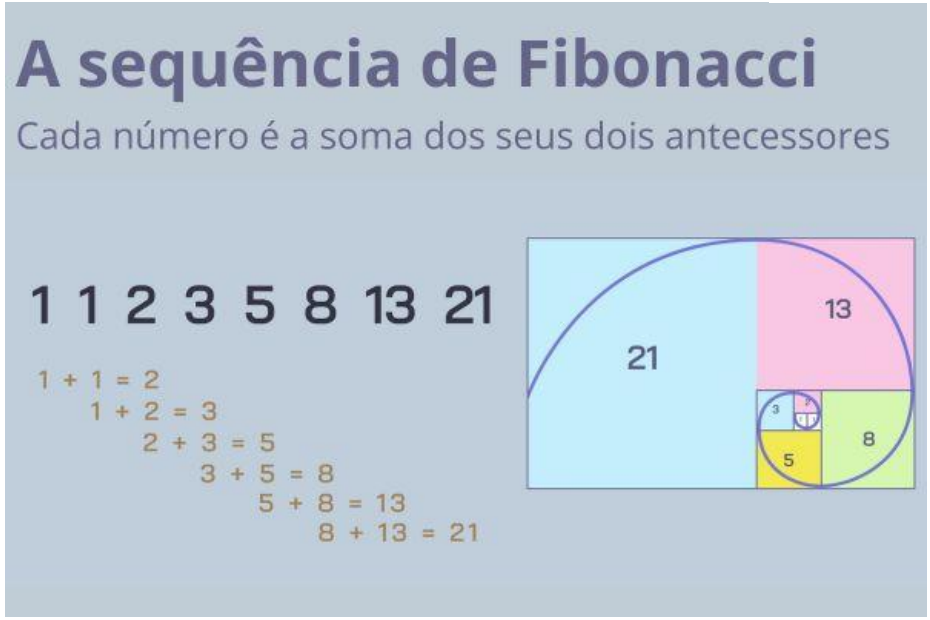
- **Exemplo 2 – Sequencia de Fibonacci**

Em termos matemáticos, a sequência é definida recursivamente pela fórmula abaixo, sendo o primeiro termo F_1= 1:

$$F_n = F_{n-1} + F_{n-2},$$

e valores iniciais

$$F_1 = 1,\ F_2 = 1.$$



📘 Recursividade em Programação

- **Exemplo 2 –** Sequência de Fibonacci

```c
// Função recursiva para calcular n-ésimo termo de Fibonacci
int fibonacci(int n) {
    if (n == 0) return 0; // Caso base 1
    if (n == 1) return 1; // Caso base 2
    return fibonacci(n - 1) + fibonacci(n - 2); // Passo recursivo
}

int main() {
    int termos = 10;
    printf("Sequência de Fibonacci até %d termos:\n", termos);
    for (int i = 0; i < termos; i++) {
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```

📌 Observações:

- Fibonacci é um clássico exemplo recursivo.
- **Custo computacional elevado**, pois há muitas chamadas redundantes.

📘 Recursividade em Programação

🔹 **Exemplo 3 – Cálculo do Máximo Divisor Comum (MDC) com Recursividade**

- O **MDC** entre dois números inteiros positivos ***a*** e ***b*** é o maior número que divide ambos sem deixar resto.
- Matematicamente, podemos defini-lo pelo **Algoritmo de Euclides**, que funciona assim:

$$\text{mdc}(a,b) = \begin{cases} a & \text{se } b = 0 \\ \text{mdc}(b, a \bmod b) & \text{se } b \neq 0 \end{cases}$$

Ou seja:

- Se $b = 0$, então $\text{mdc}(a,b) = a$.
- Caso contrário, chamamos a função novamente, trocando os papéis dos números, até que b se torne zero.

📘 Recursividade em Programação

- **Explicação Matemática do Exemplo**

Suponha que queremos calcular o **MDC(48, 18)**:

1. $\text{mdc}(48, 18) \rightarrow$ como $b \neq 0$, fazemos $\text{mdc}(18, 48 \bmod 18)$.
2. $48 \bmod 18 = 12$, então $\text{mdc}(18, 12)$.
3. $\text{mdc}(18, 12) \rightarrow$ novamente $b \neq 0$, logo $\text{mdc}(12, 18 \bmod 12)$.
4. $18 \bmod 12 = 6$, então $\text{mdc}(12, 6)$.
5. $\text{mdc}(12, 6) \rightarrow$ como $b \neq 0$, calculamos $\text{mdc}(6, 12 \bmod 6)$.
6. $12 \bmod 6 = 0$, então $\text{mdc}(6, 0) = 6$.

Portanto, $\text{MDC}(48, 18) = 6$.

📘 Recursividade em Programação

- **Explicação Matemática do Exemplo**

```
int func_mdc(int a, int b) {
    // Parada: quando b for zero, o resultado é “a”
    if (b == 0) {
        return a;
    }
    // Passo recursivo: chama a função trocando os
valores
    return func_mdc(b, a % b);
}
```

📘 Recursividade em Programação

- **Exemplo 3 – MDC**

```c
#include <stdio.h>

// Função recursiva para calcular o MDC usando o Algoritmo de Euclides
int mdc(int a, int b) {
    // Caso base: quando b for zero, o resultado é a
    if (b == 0) {
        return a;
    }
    // Passo recursivo: chama a função trocando os valores
    return mdc(b, a % b);
}

int main() {
    int x, y;

    printf("Digite dois numeros inteiros positivos: ");
    scanf("%d %d", &x, &y);

    printf("O MDC de %d e %d é: %d\n", x, y, mdc(x, y));

    return 0;
}
```

Exemplo 4 – Inversão de uma string

Elabore uma função recursiva que imprima uma string invertida.

Exemplo 4 – Inversão de uma string

Enunciado:
Faça uma função recursiva que imprima uma string invertida.

```c
void inverterString(char str[], int n) {
    //'\0': É o caractere nulo. Em C, todas as strings são arrays de caracteres terminados com este caractere especial.
    //Ele marca o fim da string.
    //if(str[n] == '\0') {
    //    return;
    //}

    //Se o número n atingir o tamanho da string, significa que chegamos ao final da string.
    if(n == strlen(str)) {
        return; // caso base
    }

    inverterString(str, n + 1); // chamada recursiva

    printf("%c", str[n]); // imprimir caractere após a chamada recursiva

}

int main() {

    char palavra[] = "Universidade";

    printf("Palavra Original: %s\n", palavra);
    printf("Palavra Invertida: ");
    inverterString(palavra,0);

    return 0;
}
```

Exemplo 5 – Soma dos elementos de um vetor

Implemente uma função recursiva que calcule a soma dos elementos de um vetor de inteiros.

Exemplo 5 – Soma dos elementos de um vetor

Enunciado:
Implemente uma função recursiva que calcule a soma dos elementos de um vetor de inteiros.

```c
#include <stdio.h>

// Função recursiva para somar os elementos do vetor
int somaVetor(int vetor[], int tamanho) {
    if (tamanho == 0) {
        return 0; // Caso base: vetor vazio tem soma zero
    }
    return vetor[tamanho - 1] + somaVetor(vetor, tamanho - 1);
    // Soma o último elemento com o restante
}

int main() {
    int v[] = {1, 2, 3, 4, 5};
    int tamanho = 5;
    printf("Soma dos elementos = %d\n", somaVetor(v, tamanho));
    return 0;
}
```

Desafio

Desafio 1 – Torres de Hanói

- **Enunciado:**
- O problema das Torres de Hanói consiste em mover todos os discos de um pino de origem para um pino de destino, utilizando um pino auxiliar.
  As regras são:
- Apenas um disco pode ser movido por vez.
- Um disco maior nunca pode ficar sobre um disco menor.
- Implemente em C um programa recursivo que resolva o problema das Torres de Hanói para **n discos**, exibindo os movimentos passo a passo.

Desafio 1 – Torres de Hanói

- **Resolução:**
- A recursão é aplicada da seguinte forma:
- Para mover **n discos** do pino A para o pino C (usando B como auxiliar):
  - Mova os **n-1 discos** de A para B (usando C como auxiliar).
  - Mova o maior disco de A para C.
  - Mova os **n-1 discos** de B para C (usando A como auxiliar).

Desafio 1 – Torres de Hanói

- **Resolução:**
- A recursão é aplicada da seguinte forma:
- Para mover **n discos** do pino A para o pino C (usando B como auxiliar):
  - Mova os **n-1 discos** de A para B (usando C como auxiliar).
  - Mova o maior disco de A para C.
  - Mova os **n-1 discos** de B para C (usando A como auxiliar).

```c
#include <stdio.h>

// Função recursiva para resolver as Torres de Hanói
void hanoi(int n, char origem, char destino, char auxiliar) {
    // Caso base: se há apenas 1 disco
    if (n == 1) {
        printf("Mover disco 1 de %c para %c\n", origem, destino);
        return;
    }

    // Passo 1: mover n-1 discos da origem para o auxiliar
    hanoi(n - 1, origem, auxiliar, destino);

    // Passo 2: mover o disco maior para o destino
    printf("Mover disco %d de %c para %c\n", n, origem, destino);

    // Passo 3: mover os n-1 discos do auxiliar para o destino
    hanoi(n - 1, auxiliar, destino, origem);
}

int main() {
    int n;

    printf("Digite o número de discos: ");
    scanf("%d", &n);

    printf("\nSequência de movimentos:\n");
    hanoi(n, 'A', 'C', 'B');  // A = origem, C = destino, B = auxiliar

    return 0;
}
```