

## Programação II: Recusividade

Universidade Federal de Sergipe – UFS  
 Curso: Sistema de Informação  
 Campus: Itabaiana  
 Professor: Thiago S. Reis Santos  
 e-mail: thiago.ufs@gmail.com

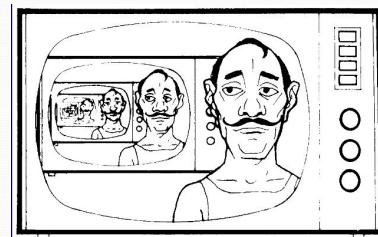
## Roteiro

- O que é Recursividade?
- Técnicas para desenvolver algoritmos recursivos
- Definição Matemática de Fatorial
- Exercício – Resolvido
- Recusividade
- Exercícios
- Torre de Hanói

## O que é Recusividade?

- Recusividade significa algo ser definido em termos de si próprio.
- Em termos de programação: a recursividade é uma técnica em que uma rotina, no processo de execução de suas tarefas, chama a si mesma;
- A recursão é uma técnica poderosa em definições matemáticas;

## O que é Recusividade?



Uma Imagem Recusiva.

## Técnicas para desenvolver algoritmos recursivos

- A recursão em programação se dá através de sub-rotinas, geralmente funções;
- Assim sendo cria-se uma função que em seu corpo terá uma chamada a si própria;
- Todo cuidado é pouco ao se usar funções recursivas, pois facilmente podemos gerar um "loop infinito".

## Técnicas para desenvolver algoritmos recursivos

- A primeira coisa que temos de pensar é em um critério de parada. Isto impede que a função chame a si própria "infinita" vezes;
- Este é um requisito fundamental: as chamadas recursivas de um procedimento P devem estar sujeitas à uma condição B (condição de parada), a qual, em algum instante, se torna verdadeira.

*"Finitude dinâmica"*

## Definição Matemática de Fatorial

$$x! = \begin{cases} \text{se } x = 0 \Rightarrow 0! = 1 \\ \text{se } x = 1 \Rightarrow 1! = 1 \\ \text{se } x > 0 \Rightarrow x! = x * (x-1)! \end{cases}$$

• A definição de fatorial é de natureza recursiva, ou seja, existe uma chamada a ela mesma em  $x * (x-1)!$

• A condição de parada é quando  $x$  é decrementado até o valor 1. Dessa forma encerra-se as multiplicações sucessivas.

## Exercício - Resolvido

- Escreva uma função em Java para o cálculo do fatorial de forma recursiva e de forma iterativa.

```
// Algoritmo Iterativo
public long factorial(int x) {
    if(x<0) //Número negativo
        return -1;
    long fat = 1;
    while (x>-1)
        fat *= x;
    return fat;
}
```

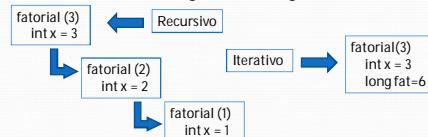
```
//Algoritmo Recursivo
public long factorial(int x) {
    if(x<0) //Número negativo
        return -1;
    if(x == 0 || x == 1)
        return 1;
    return x * factorial(x-1);
}
```

## Recursividade

- Há algoritmos que são mais eficientes quando implementados de forma recursiva, entretanto rotinas recursivas requerem área para armazenamento dos valores dos objetos a cada chamada;
- Isto acarreta num risco de exceder o espaço disponível de memória e ocorrer uma falha no programa;

## Recursividade

- Todo algoritmo recursivo pode ser transformado para um iterativo.
- As implementações recursivas tendem a ser mais custosas tanto em termos de memória como também de cpu, já que é alocada uma pilha de processamento e de memória muito grande (no geral).



## Recursividade

- Entretanto os algoritmos escritos de forma recursiva tendem a ser mais elegantes, uma programação mais enxuta e de mais fácil entendimento pelo ser humano.

## Recursividade

- Tipos de recursão:
  - Se um procedimento P contiver uma referência explícita a si próprio, este procedimento é dito diretamente recursivo;
  - Se P contiver uma referência a outro procedimento Q, que por sua vez contém uma referência direta ou indireta a P, então P é dito indiretamente recursivo;

## Exercícios

- 1º) Faça um programa em Java que calculo o valor de Fibonacci para um "n" fornecido pelo usuário.

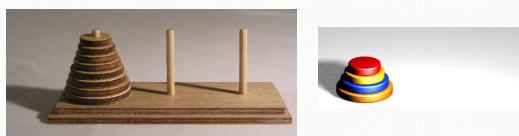
$$\begin{cases} \text{se } n = 0 \Rightarrow fib(0) = 1 \\ fib(n) \text{ se } n = 1 \Rightarrow fib(1) = 1 \\ \text{se } n > 1 \Rightarrow fib(n) = fib(n-1) + fib(n-2) \end{cases}$$

## Exercícios

- 2º) Faça a implementação de forma recursiva do rastreamento da solução do problema da Torre de Hanói sendo o número de disco será informado pelo usuário (máximo 12).
- A Torre de Hanói é um quebra-cabeça que consiste em uma base contendo n pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo.

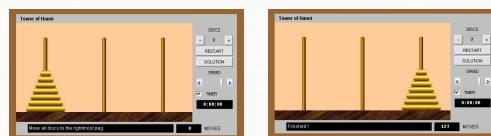
## Torre de Hanói

- O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação.



## Torre de Hanói

- [http://www.prof2000.pt/users/pjca/jogos\\_ficheiros/hanoi/Torre%20de%20Hanoi.html](http://www.prof2000.pt/users/pjca/jogos_ficheiros/hanoi/Torre%20de%20Hanoi.html)



## Torre de Hanói

- É interessante observar que o número mínimo de "movimentos" para conseguir transferir todos os discos da primeira estaca à terceira é  $2^n - 1$ , sendo n o número de discos. Logo:
- Para solucionar um Hanói de 3 discos, são necessários  $2^3 - 1$  movimentos = 7 movimentos
- Para solucionar um Hanói de 7 discos, são necessários 127 movimentos
- Para solucionar um Hanói de 15 discos, são necessários 32.767 movimentos

## Exercícios

- A resolução do 2º exercício deve ser a impressão dos movimentos dos discos.
- Ex.: Caso seja com 5 disco a solução deve ser assim:
  - 1º passo – movendo disco A da 1ª torre para 3ª torre
  - 2º passo – movendo disco B da 1ª torre para 2ª torre
  - 3º passo – movendo disco A da 3ª torre para 2ª torre
  - 4º passo – movendo disco C da 1ª torre para 3ª torre

## Referências

- HORSTMANN, Cay S.; CORNELL, Gary. **Core Java 2: Volume I - Fundamentos.** Tradução: Andreza Gonçalves e Marcelo Soares. 7<sup>a</sup> ed. Rio de Janeiro: Alta Books editora, 2005.
- Slides da Professora Maily.
- Wikipedia,  
[http://pt.wikipedia.org/wiki/Torre\\_de\\_Hanoi](http://pt.wikipedia.org/wiki/Torre_de_Hanoi)