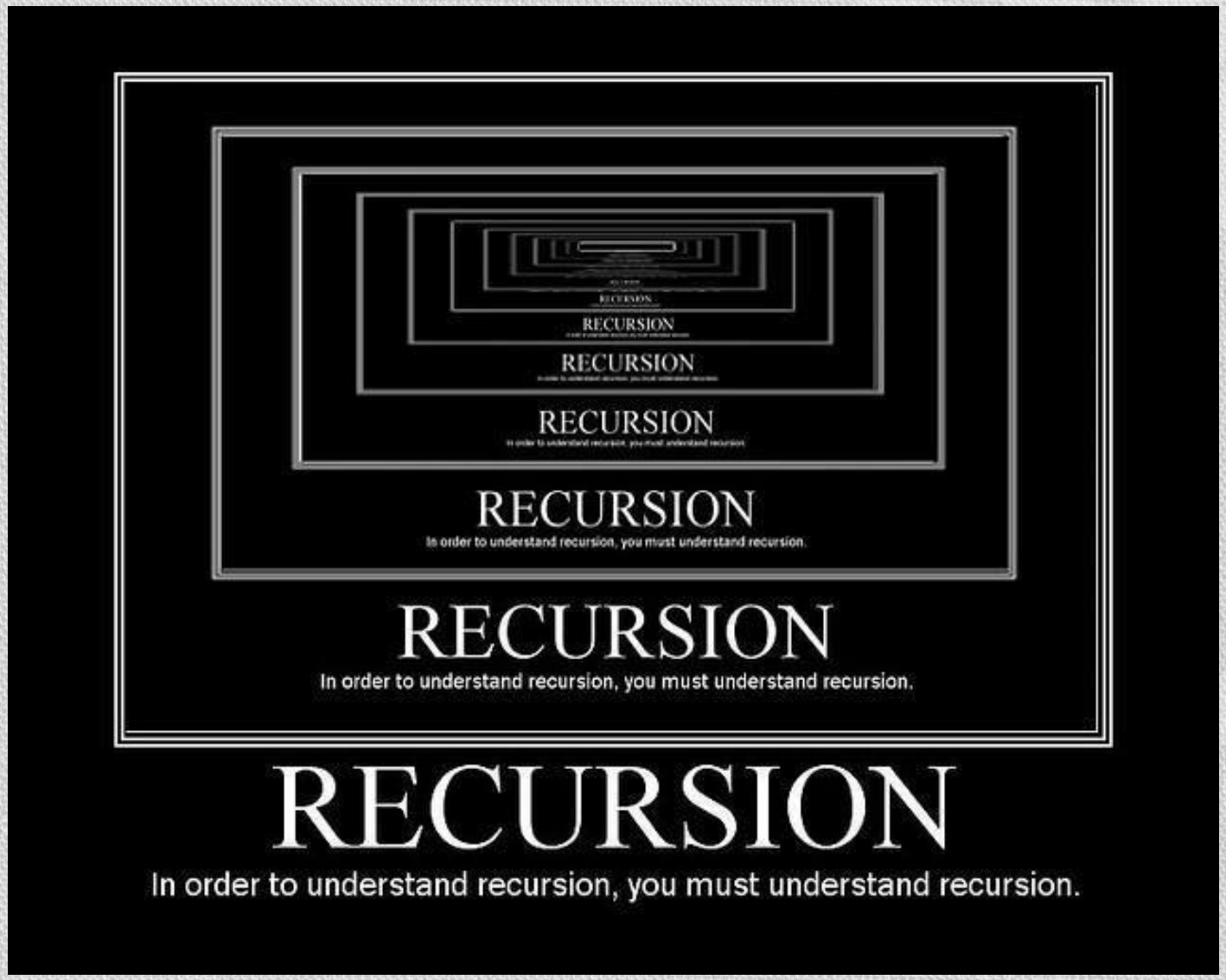
ESTRUTURA DE DADOS I

Recursividade

Profª Amanda Tameirão

Objetivos da aula de hoje

- Compreender o conceito de recursividade;
- Aplicar regras para criação de código recursivo;
- Desenvolver funções recursivas.


RECURSION
RECURSION
RECURSION
RECURSION
In order to understand recursion, you must understand recursion.
RECURSION
In order to understand recursion, you must understand recursion.
RECURSION
In order to understand recursion, you must understand recursion.

Conceito

Um algoritmo é considerado recursivo quando sua execução chama a si mesmo, ou chama uma sequencia de outros, onde um deles volta a chamar o primeiro.

Também conhecido como:

Recursão
Recorrência

Objetivo

A proposta consiste em dividir um problema complexo, em problemas menores.

Cada uma das pequenas soluções fará parte da solução final.

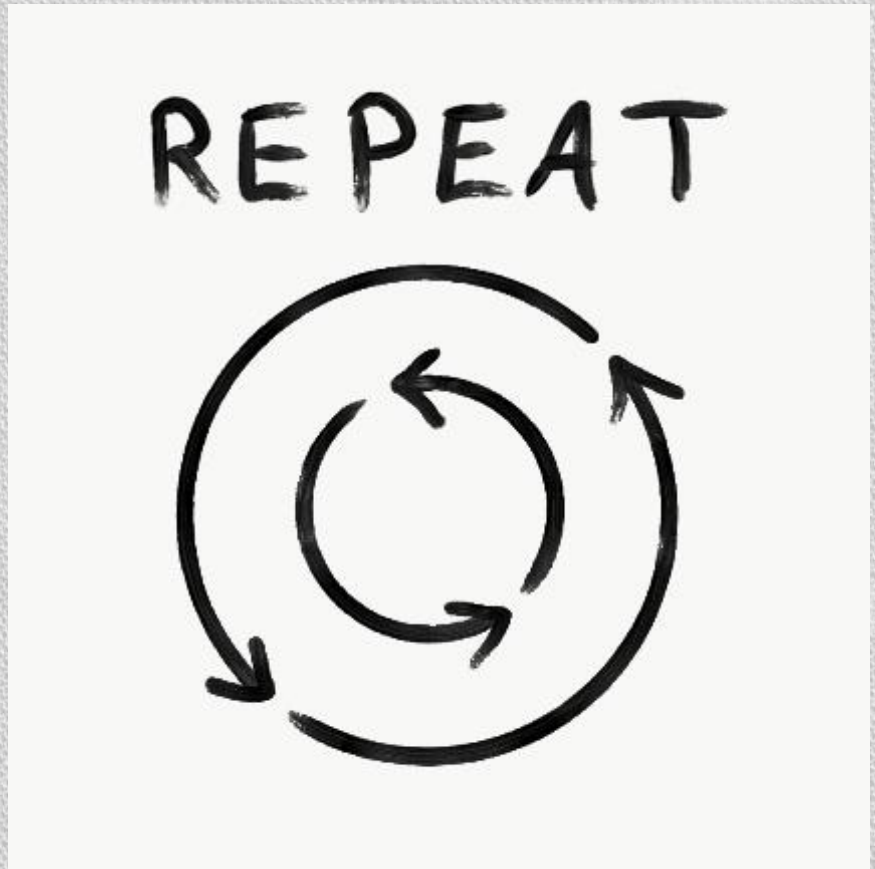
Dividir para conquistar

Faz pate de uma classe de algoritmos conhecidos como **Dividir para conquistar.**

Dividir seus problemas em problemas menores, e mais fáceis de resolver.

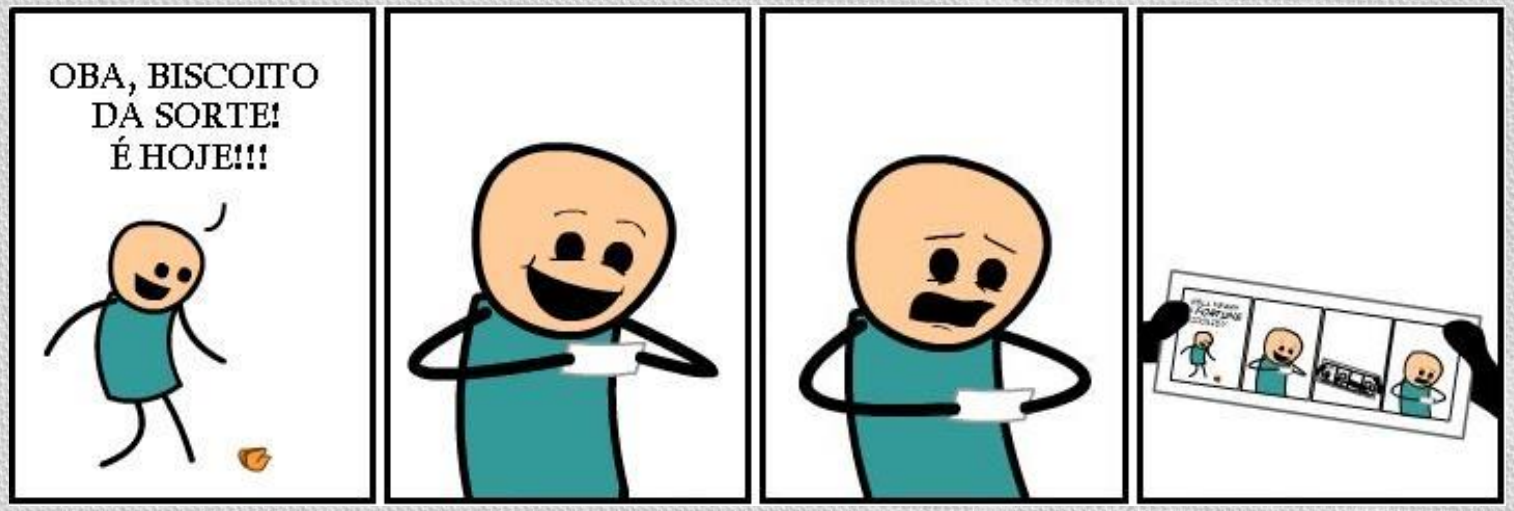
Paradigma Funcional

A recursividade foi criada para atender a necessidade de repetião, em linguagens que não possuem estrutura de repetição



Dúvida comum

Se uma função chamará ela mesma, como garantir que não irá gerar um loop eterno?



Siga a estratégia!!!!

Estratégia

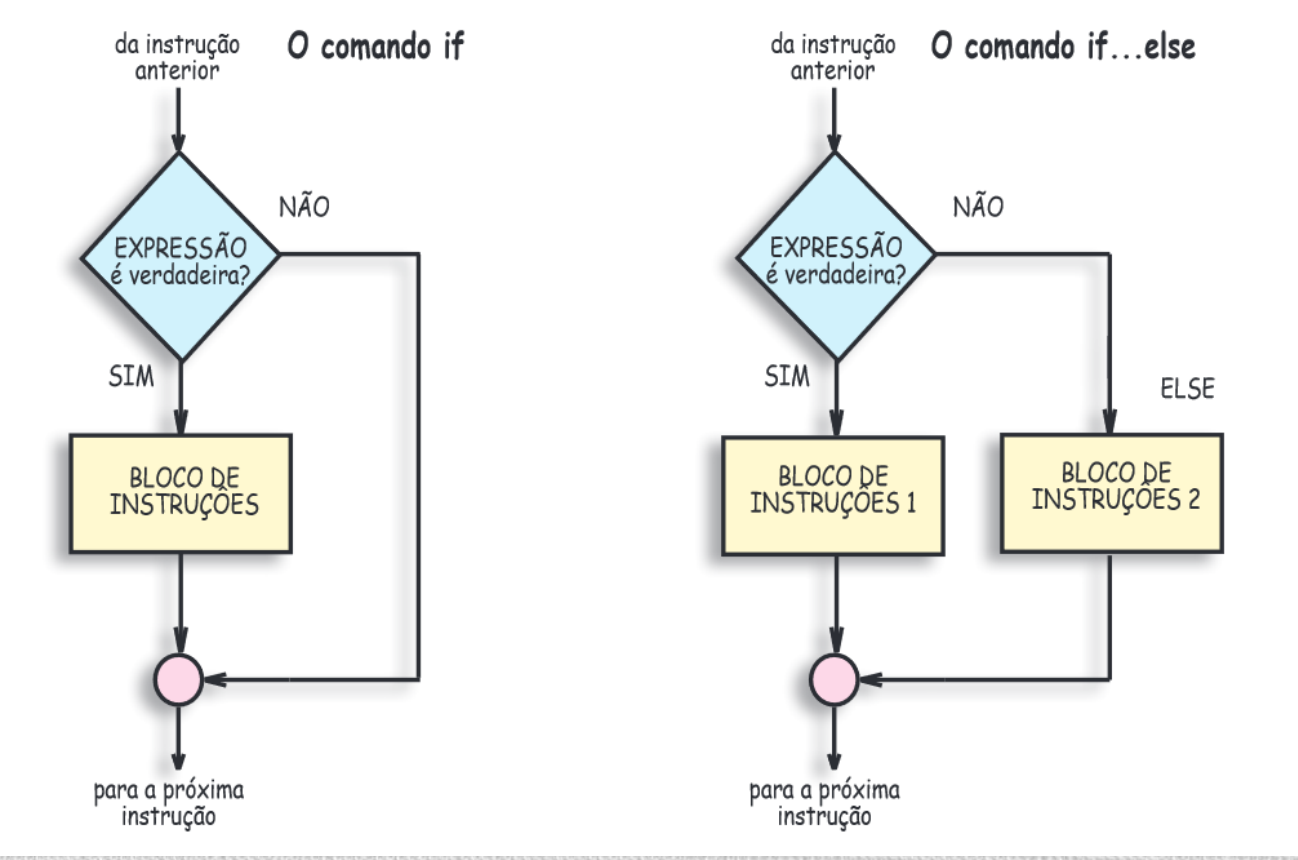
Para criar um código recursivo, é importante pensar em duas partes:

1ª Parte – Conhecida como âncora, ou caso base, é aqui que seu código irá finalizar a repetição, por isso, garanta que ela sempre existirá.

2ª Parte – Caso recursivo, aqui seu código irá chamar uma nova execução, dele mesmo, porém, como novos parâmetros para entrada. Executará até "encontrar" a primeira parte.

Importante

Utilize apenas estrutura de decisão, nunca estrutura de repetição, afinal, a repetição é feita a a partir das novas chamadas do mesmo código.



Estratégia

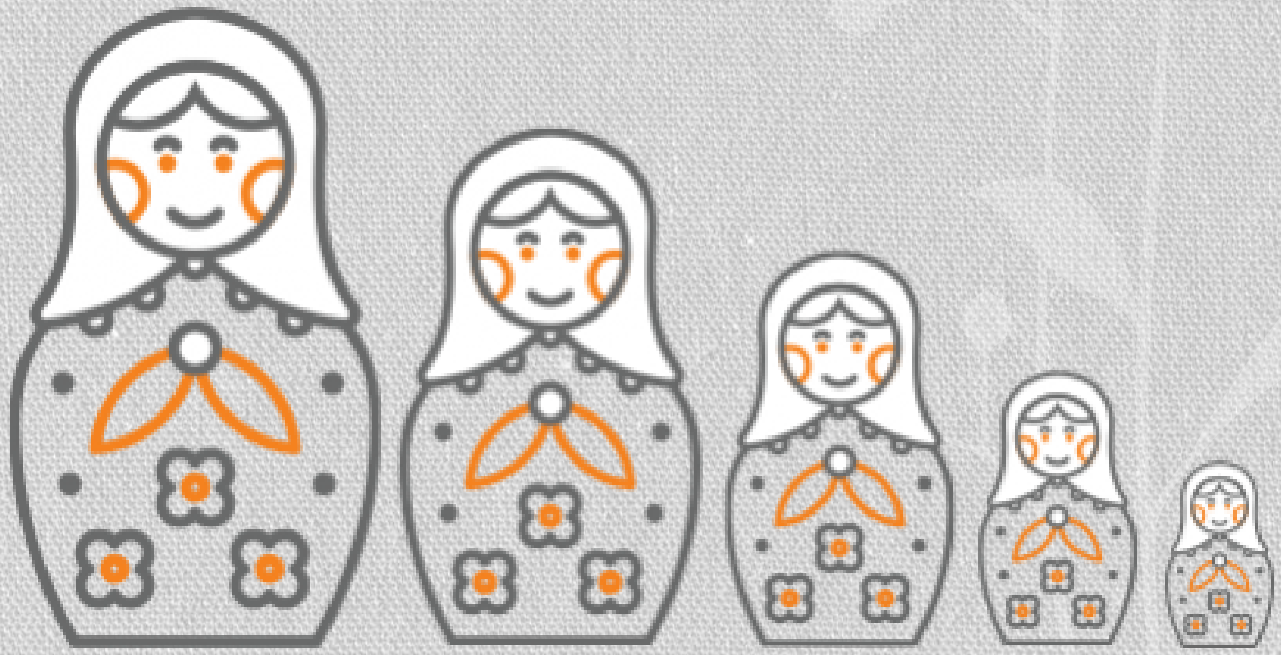
se

então – Caso base (finaliza a repetição e retorna os valores necessários)

senão – nova chamada da função (executada a partir desta instância, com novos parâmetros)

Relembrando

Ser recursivo é uma forma de repetir sem utilizar estrutura de repetição, a estratégia é utilizar estrutura de decisão e, a partir das novas chamadas, repetir todo o código da função.



Vamos a um exemplo

O cálculo do fatorial é um clássico da recursividade.

Vamos relembrar:

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$5! = (5) \times (5-1) \times (4-1) \times (3-1) \times (2-1)$

Isso significa que:

fatorial(1) = 1

fatorial(2) = 1 x 2 = 2

fatorial(3) = 1 x 2 x 3 = 6

fatorial(4) = 1 x 2 x 3 x 4 = 24

fatorial(5) = 1 x 2 x 3 x 4 x 5 = 120

Solução iterativa (loops)

```
int fatorialIterativo (int n) {
        int  i,
             fat =1;

        for (i=2; i<=n; i++) {
                fat * = i;
        }

        return fat;
}
```

Utiliza o **processamento** a seu favor.

Efetua apenas a **repetição dos comandos** existentes no loop.

Vamos a um exemplo

O cálculo do fatorial é um clássico da recursividade.

Vamos relembrar:

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$5! = (5) \times (5-1) \times (4-1) \times (3-1) \times (2-1)$

Isso significa que:

fatorial(1) = 1

fatorial(2) = 1 x 2 = 2

fatorial(3) = 1 x 2 x 3 = 6

fatorial(4) = 1 x 2 x 3 x 4 = 24

fatorial(5) = 1 x 2 x 3 x 4 x 5 = 120

Proposta da recursividade

fatorial(N) =

1, se N <= 1

N x fatorial(N-1), se N > 1 (não negativos)

Ou seja:

fatorial(1) = 1

fatorial(2) = 2 x fatorial (1) = 2

fatorial(3) = 3 x fatorial (2) = 6

fatorial(4) = 4 x fatorial (3) = 24

fatorial(5) = 5 x fatorial (4) = 120

Exemplo na estratégia

se n<=1

então – retorna 1 (caso base)

senão – executa novamente com n – 1 multiplicado ao n da instância.

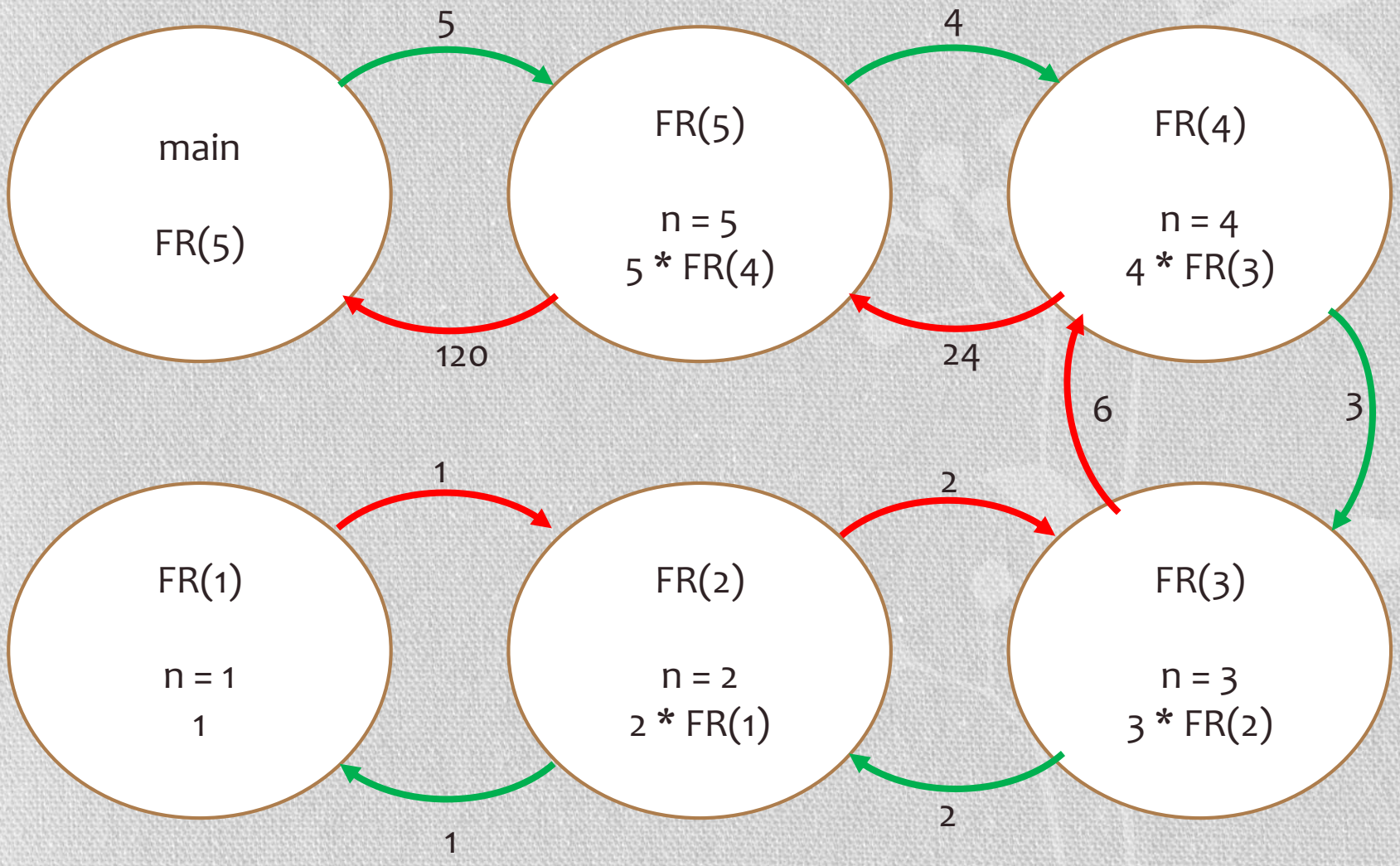
Solução iterativa (loops)

```
int fatorialRecursivo (int n) {
        if (n <= 1) {
                return 1;
        } else {
                return n * fatorialRecursivo(n – 1);
        }
}
```

Utiliza a memória a seu favor.

Efetua a repetição de todo o código da função, porém, em nova instância.

Simulação – FatorialRecursivo = FR



Recursividade x Iteração

Existem vantagens e desvantangens para utilizar qualquer uma das estratégias.

Recursividade	Iteração
Utiliza estrutura de seleção	Utiliza estrutura de repetição
Funções repetidas	Repetições de algumas linhas de códigos
Termina no caso base	Termina na condição do loop
Pode executar infinitamente	Pode executar infinitamente
Lento	Rápido
Simples e de fácil manutenção	Complicadas e de difícil manutenção

Recursividade - Desvantagens

Uma das maiores desvantagens da técnica é o uso excessivo de memória, podendo gerar estouro de pilha (*stack overflow*).

Mas lembre-se....

Existem linguagens que são funcionais, ou seja, são executadas a partir de funções.
Para estas, não existem estruturas de repetição disponíveis, apenas recursão.

Recursividade de cauda

Uma outra forma de desenvolver códigos recursivos é utilizar recursividade de cauda, com ela, é possível criar códigos que aproveitem melhor o processamento utilizado.

A idéia é “levar um resultado parcial” até chegar ao caso base, assim, o caso base terá a solução final, e não precisará retornar com os resultados para cálculo.

Recursividade de cauda - regras

A recursividade de cauda soluciona o problema do estouro de pilha, já que outros processamentos não serão deixados para trás.

A chamada recursiva é a última operação a ser executada.

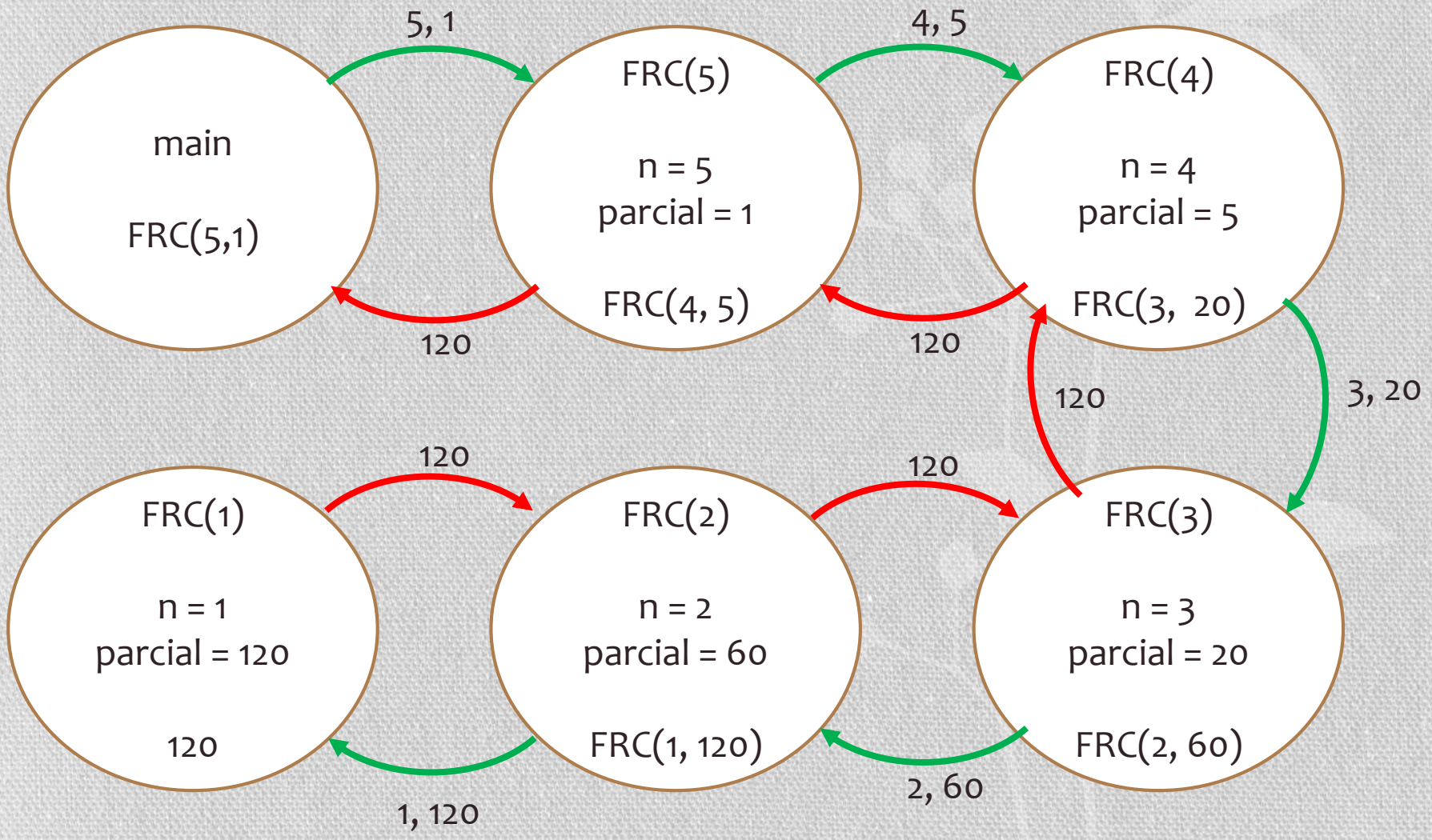
O resultado parcial é carregado de função em função.

Não deixa processamento para “trás”.

Solução recursiva em cauda (Fatorial)

```
int fatorialRecursivoCauda (int n, int parcial) {
    if (n <= 1)
        return parcial;
    else
        return fatorialRecursivoCauda (n-1, parcial * n);
}
```

Simulação – FatorialRecursivoCauda = FRC



EDI

Próxima aula

Exercício aplicando recursividade

Bibliografia

- ZIVIANI, Nivio. **Projeto de algoritmos**: com implementações em Pascal e C. 2ed. São Paulo: Pioneira Thomson Learning, 2004. 552 p.
- CORMEN, T. H. et alii. **Algoritmos**: Teoria e Prática. 1ed. Rio de Janeiro: Campus, 2002. 916pp.
- Notas de aula do **Prof. Flavio Lapper**
- Notas de aula do **prof. Rodrigo Oliveira**. Disponível em http://www.ic.unicamp.br/~oliveira/doc/mc102_2s2004/Aula19.pdf. Acesso em 10 fev 2017.