Estruturas de Dados 1

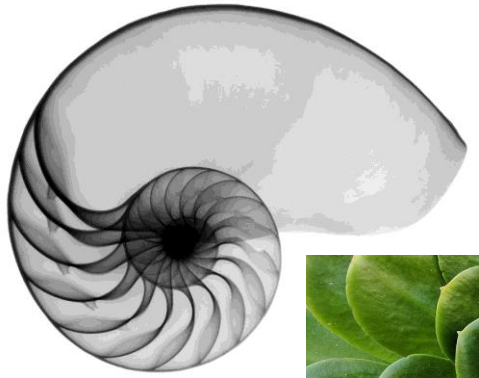
08 – Recursividade

Antonio Angelo de Souza Tartaglia

angelot@ifsp.edu.br

Recursividade

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema;
- É a capacidade que uma linguagem de programação possui, de permitir que uma função possa invocar a si mesma.
- Dessa forma, recursividade é um princípio que permite obter a solução de um problema, a partir do próprio problema;



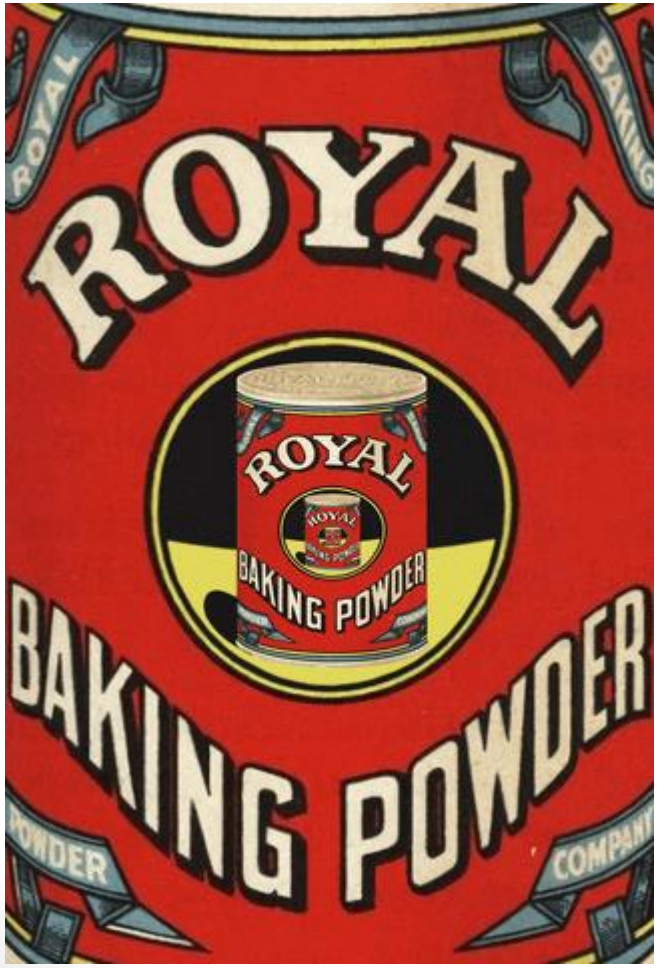
- Exemplos de recursividade:
  - Dois espelhos apontados um para o outro;
  - Caracóis, girassóis, folhas de algumas plantas.

Estrutura de Dados 1

Recursividade

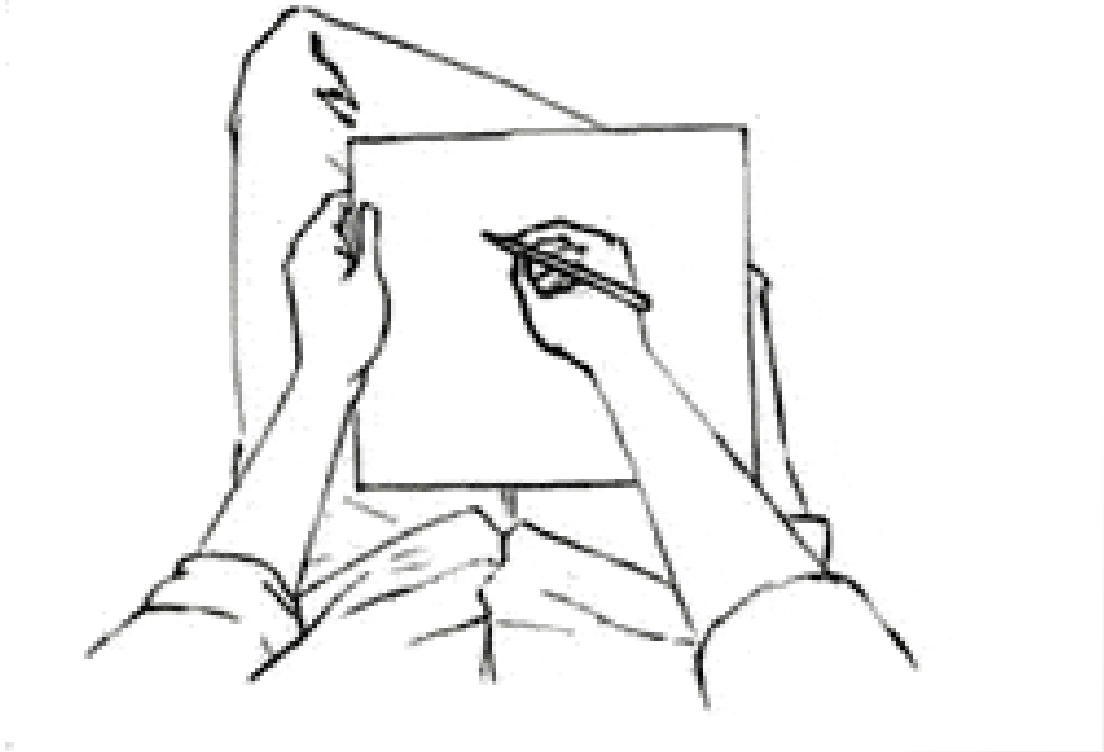
Estrutura de Dados 1

Recursividade



Estrutura de Dados 1

Recursividade



Torres de Hanoi

- Torres de Hanói é um jogo matemático inventado no ano de 1883, pelo matemático Eduard Lucas.
- Dispomos de **3 pinos**: “pino origem”, “pino de trabalho” e “pino destino”.
- O “pino origem” contém **n discos** empilhados por ordem crescente de tamanho (o maior disco fica embaixo).
- O objetivo do jogo é levar todos os discos do “pino origem” para o “pino destino”, utilizando o “pino de trabalho” para auxiliar a tarefa, e atendendo às seguintes restrições:

1. Apenas um disco pode ser movido por vez (o disco que estiver no topo da pilha de um dos pinos).
2. Um disco de tamanho maior nunca pode ser colocado sobre um disco de tamanho menor.

Torres de Hanoi – Solução Recursiva

- O jogo das Torres de Hanói também é muito apreciado por programadores e cientistas da computação porque possui uma **solução recursiva** que pode ser programada de uma maneira muito simples e elegante.

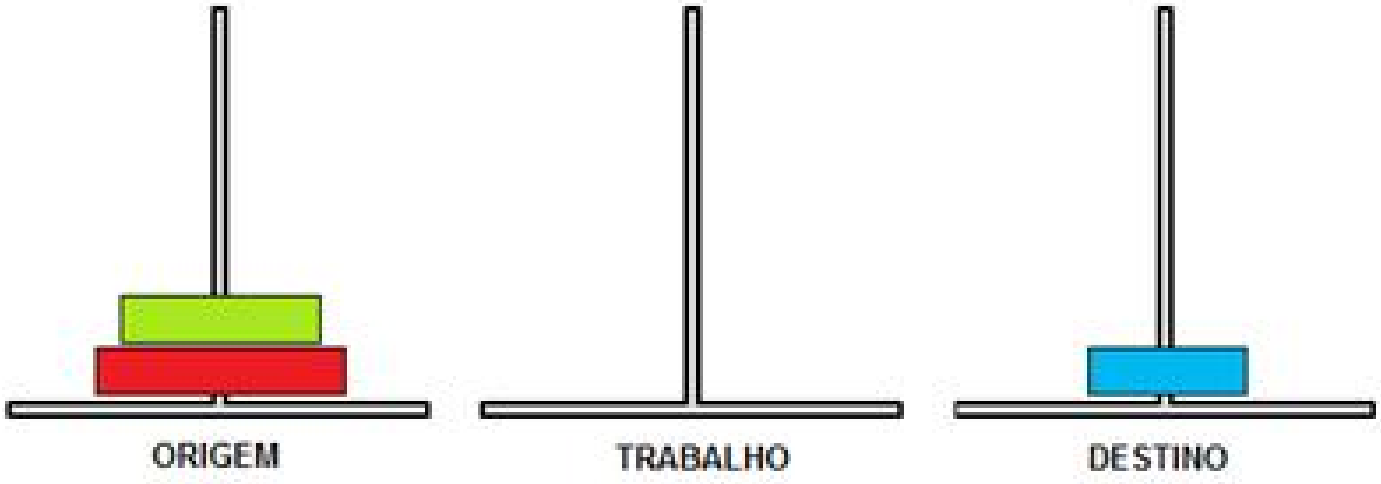
- Como toda solução recursiva, ela baseia-se na resolução de um problema de menor dimensão (ou seja, na resolução de um problema como um menor número de discos). Para resolver um jogo onde precisamos mover n discos, considerando n > 1, podemos executar os passos a seguir:

Torres de Hanoi – Solução Recursiva

- Mover n-1 discos para o “pino de trabalho”.
- Mover o n-ésimo pino (o maior de todos) do “pino origem” para o “pino destino”.
- Após isto, devemos resolver o problema da “Torre de Hanói” para os n-1 discos dispostos no “pino de trabalho”, movendo-os para o “pino destino” utilizando o mesmo princípio.

Torres de Hanoi – Solução Recursiva

- **PASSO 1:** Os movimentos 1, 2 e 3 mostram a transferência de n-1 discos do “pino origem” para o “pino de trabalho. Nesta caso, “pino destino” atua como auxiliar.
- **Movimento 1**



Torres de Hanoi – Solução Recursiva

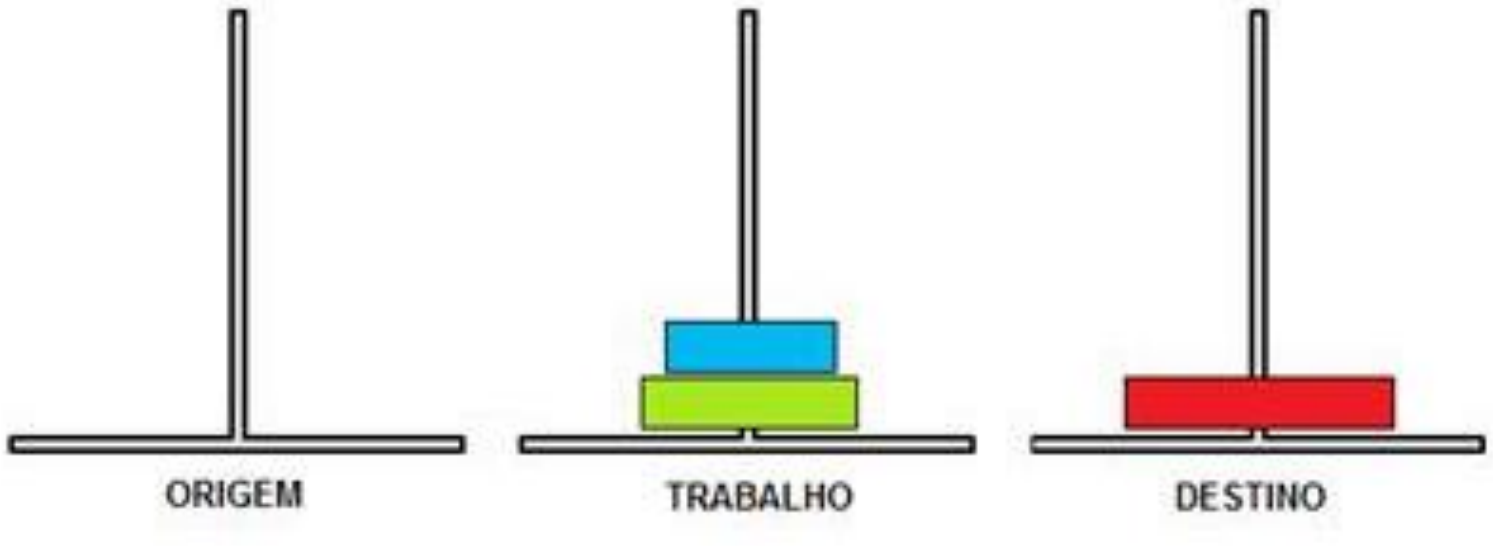
- **Movimento 2**

Torres de Hanoi – Solução Recursiva

- **Movimento 3**

Torres de Hanoi – Solução Recursiva

- **PASSO 2** : O movimento 4 mostra a transferência do maior disco do “pino origem” para o “pino destino”
- **Movimento 4**

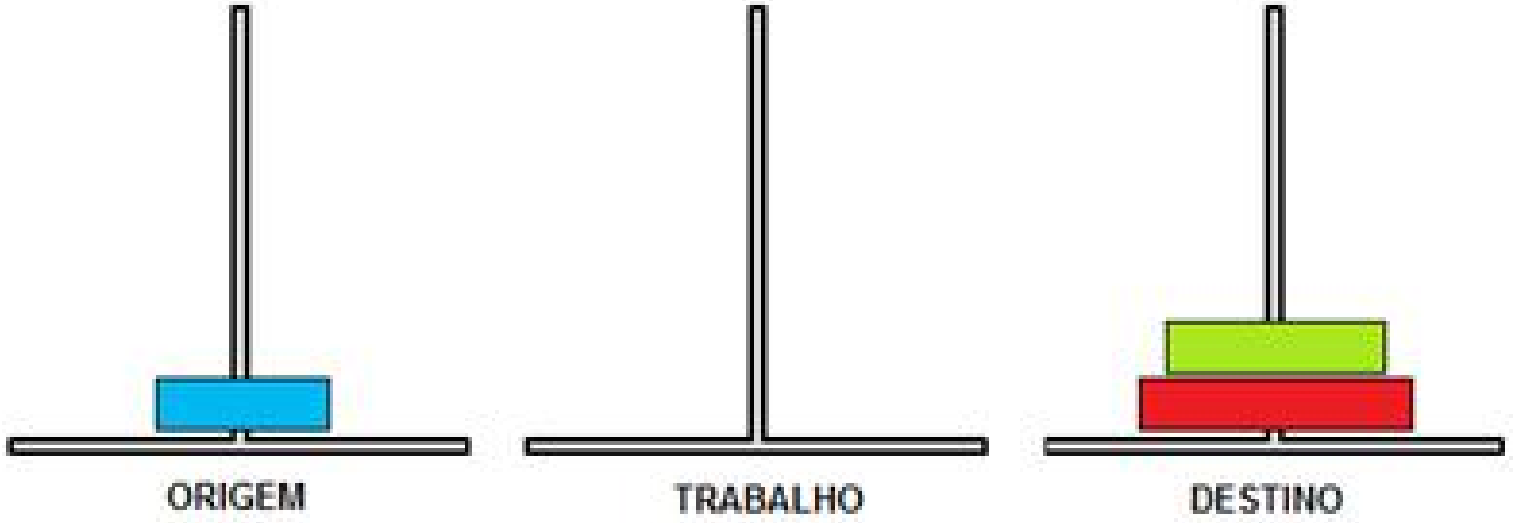


Torres de Hanoi – Solução Recursiva

- **PASSO 3**: Por fim, os movimentos 5, 6 e 7 ilustram a transferência dos n-1 discos do “pino de trabalho” para o “pino destino”. Veja que, desta vez, o “pino de origem” é que atua como área de armazenamento auxiliar.
- **Movimento 5**

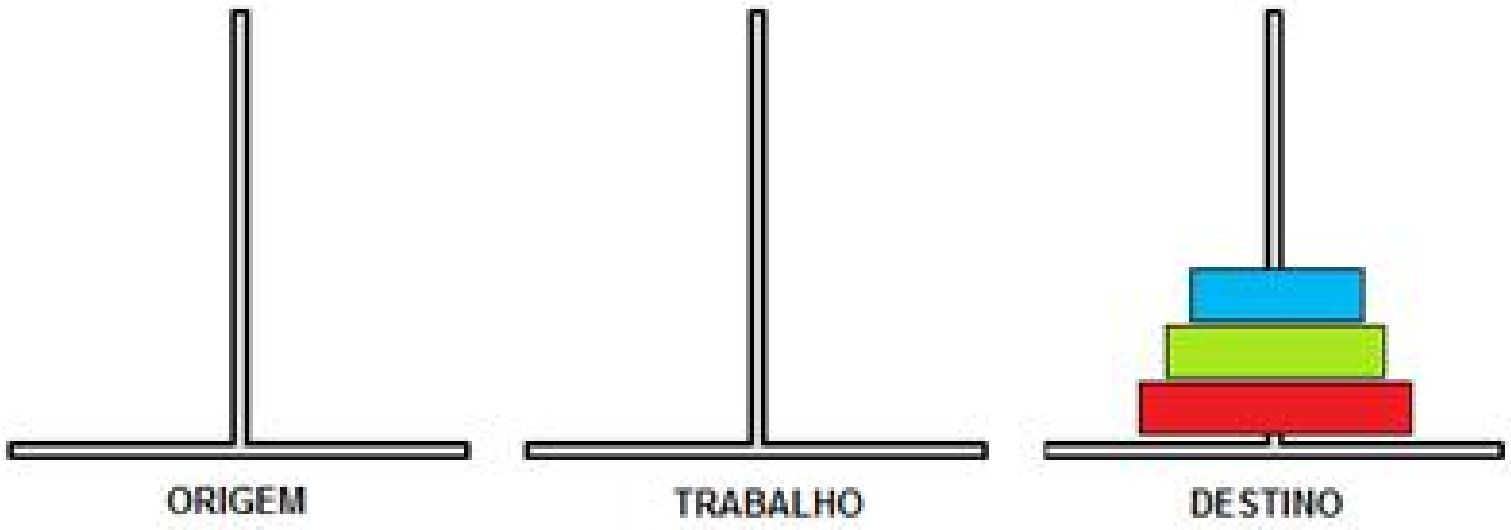
Torres de Hanoi – Solução Recursiva

- **Movimento 6**



Torres de Hanoi – Solução Recursiva

- **Movimento 7**



Recursividade

- **Agora leia o arquivo “Recursividade.pdf”, disponível na plataforma Moodle.**

Função Recursiva

- Ser uma função recursiva é a capacidade que esta possui de chamar a si própria.
  - Exemplo – usaremos o cálculo de fatorial:

$$n! = n * (n-1)!$$

- Quando uma função recursiva chama a si mesma, imagine que uma “nova cópia” da função passa a ser executada.
  - Os parâmetros locais da segunda cópia são independentes dos parâmetros locais da primeira.

Recursão é uma forma de implementar um laço através de chamadas sucessivas a mesma função.

Recursividade

- A ideia básica da recursão é dividir para conquistar:
  - problema maior → problema menor;
  - solucionar os problemas menores;
  - Combinar soluções.

Problemas menores são mais fáceis de solucionar...

Combinar a solução de cada um desses problemas menores, para chegar na solução do problema maior.

```
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 //CASO BASE
0! = 1 //CASO BASE

n! = n * (n - 1)!
1! = 1
0! = 1
```

Generalizando para a linguagem C...

Tipos de Função Recursiva

- Recursão direta ou simples:
  - Quando a função recursiva chama a si mesma diretamente;

- Recursão indireta ou composta:
  - Quando a função chama outra função, e esta, por sua vez chama a primeira.

Função Recursiva – Condição de parada

- Em funções recursivas pode ocorrer um problema de terminação do programa: loop interminável ou infinito

- Para que isso não ocorra, ou seja, para que a recursão não seja infinita, é necessário que haja uma **condição de parada**. Esta condição é que determinará quando a função deverá parar de criar novas instâncias de si mesma; quando será a hora de parar de chamar a si própria.

Função Recursiva – Regras para escrita

- A implementação do critério de término (**CASO BASE**), das chamadas deve obrigatoriamente ser realizada antes da chamada recursiva, isto é, qual a condição ou condições que se devem verificar para que a função pare de invocar a si própria;

- Só depois de escrito o critério de término é que se deverá ser escrito o passo da chamada recursiva da função, e sempre relativa a um subconjunto, diminuído, que deverá se aproximar mais do caso base.

Função Recursiva – Passos Básicos

- Ao inicializar o algoritmo:
  - É comum que algoritmos recursivos necessitem de um ponto de partida, o qual normalmente é passado por parâmetro;
  - Verificar se o valor avaliado bate com o **CASO BASE** ou **CONDIÇÃO DE PARADA**;
  - Redefinir a resposta em um subproblema menor;
  - Rodar o algoritmo no subproblema redefinido;
  - Combinar o resultado para formular a resposta;
  - Devolver o resultado.

Função Recursiva - Estrutura

- Função(lista_de_parâmetros)
  - **CASO BASE** (condição de parada)
    - Teste de término de recursão;
      - Se teste ok, retorna neste ponto;
    - É possível que haja mais de um caso base.

Se não existir, a função continuará sendo executada e chamando a si própria indefinidamente até esgotar a memória disponível!

  - **CASO RECURSIVO** (Regra Geral)
    - É a alma da recursão;
    - Permite ao programador redefinir o problema para um novo, que se aproxime mais do caso base.

É necessária a mudança do valor do parâmetro passado, de forma que a recursão se aproxime do **caso base** e chegue a um término.

Calculo iterativo de Fatorial

```c
#include <stdio.h>
#include <stdlib.h>

int fatorial_iterativo(int n);

int main(){
    int n;
    int resultado;
    printf("Entre com o numero para calculo de fatorial: ");
    scanf("%d", &n);

    resultado = fatorial_iterativo(n);
    printf("\n\nO resultado do fatorial do numero %d e: %d\n\n\n", n, resultado);
    system ("pause");
}

int fatorial_iterativo(int n){
    int t, f;
    f = 1;
    for(t = 1; t <= n; t++){
        f = f * t;
    }
    return f;
}
```

```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 08 - Recursividade\material de apoio
Entre com o numero para calculo de fatorial: 5

O resultado do fatorial do numero 5 e: 120

Pressione qualquer tecla para continuar. . .
```

Calculo Recursivo de Fatorial

```c
#include <stdio.h>
#include <stdlib.h>

int fatorial_recursivo(int n);

int main(){
    int n;
    int resultado;
    printf("Entre com o numero para calculo de fatorial: ");
    scanf("%d", &n);

    resultado = fatorial_recursivo(n);
    printf("\n\nO resultado do fatorial do numero %d e: %d\n\n\n", n, resultado);
    system ("pause");

}

int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```

```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aulas\Aula 08 - Recursividade\material de apoio\
Entre com o numero para calculo de fatorial: 5

O resultado do fatorial do numero 5 e: 120

Pressione qualquer tecla para continuar. . .
```

Função Recursiva

Exemplo de calculo recursivo de Fatorial

- $n! = \begin{cases} 1, & se\ n = 0 \\ n * (n-1)!, & se\ n > 0 \end{cases}$

```
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 //CASO BASE
0! = 1 //CASO BASE

n! = n * (n - 1)!
1! = 1
0! = 1
```

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```

Sempre que uma função é chamada, “quem chamou a função” é pausado e aguarda o retorno dessa chamada exatamente onde parou.

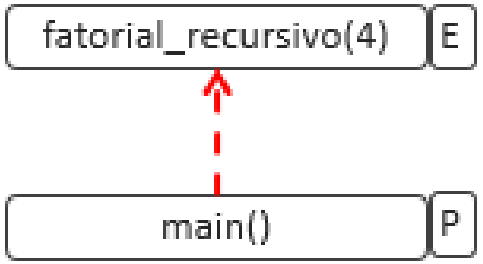
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!

fatorial_recursivo(4)

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



Função Recursiva

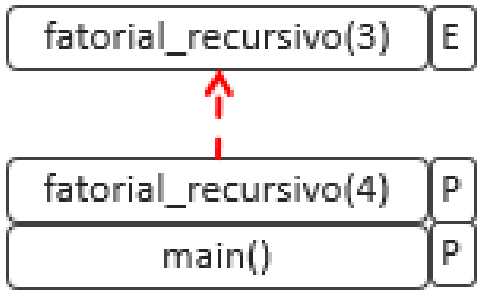
Exemplo de calculo recursivo de Fatorial

- Exemplo 4!

fatorial_recursivo(4)

4 * fatorial_recursivo(3)

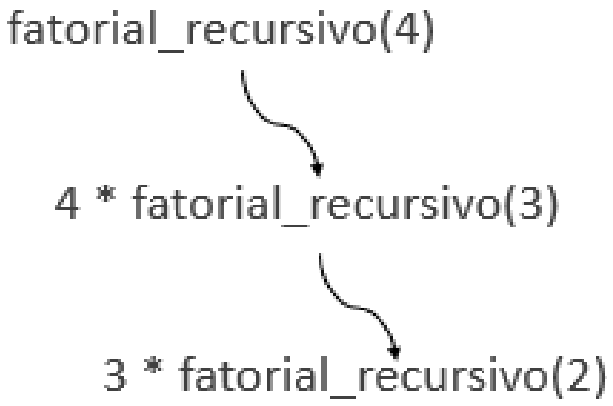
```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



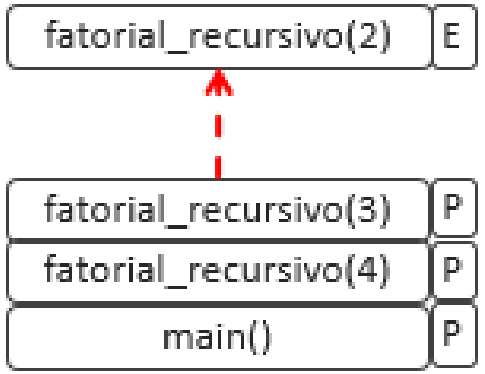
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!



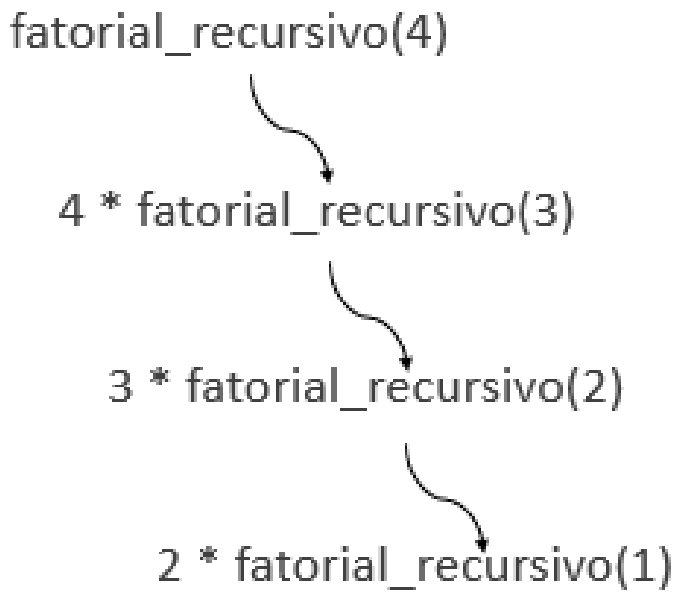

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



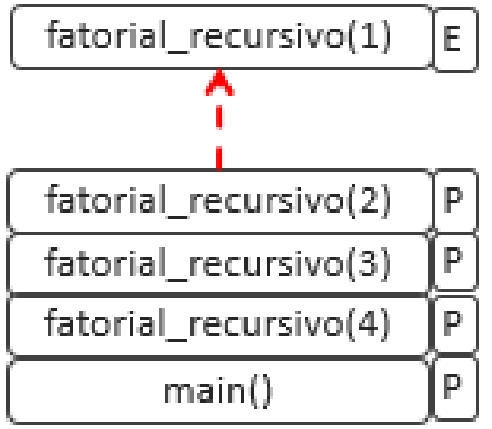
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!



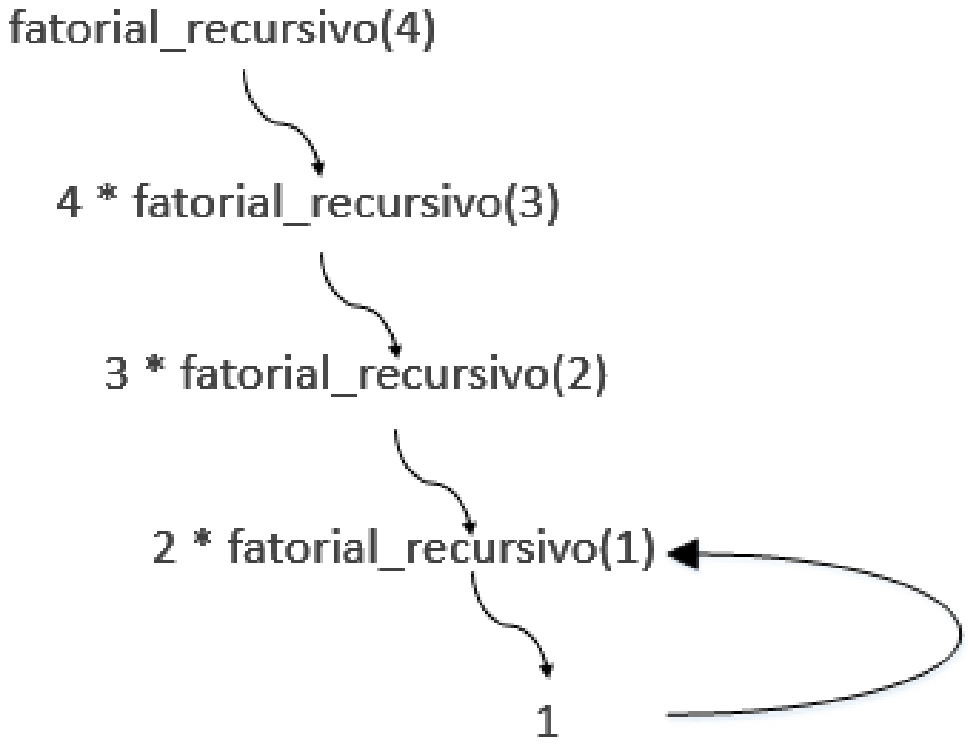

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



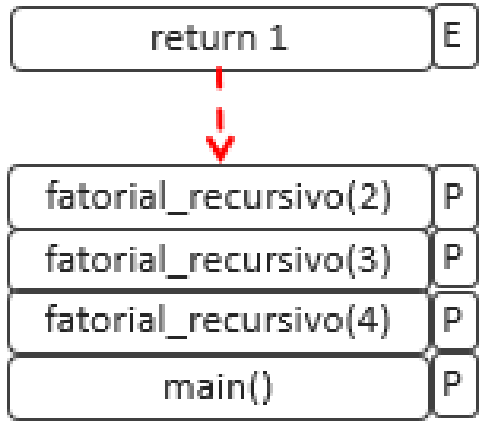
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!



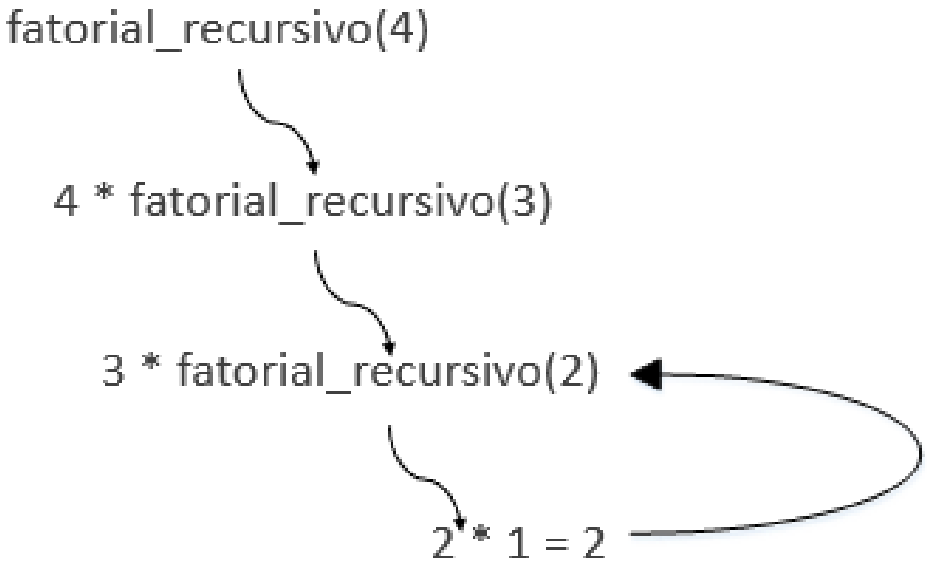

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



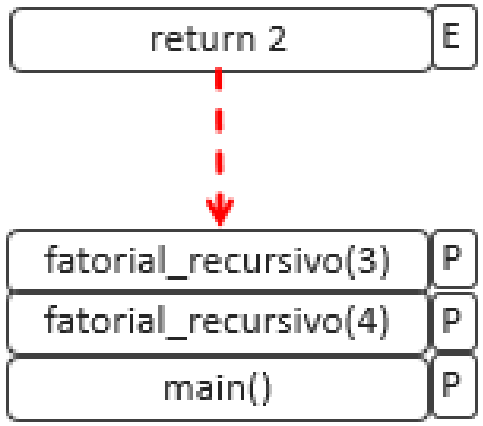
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!



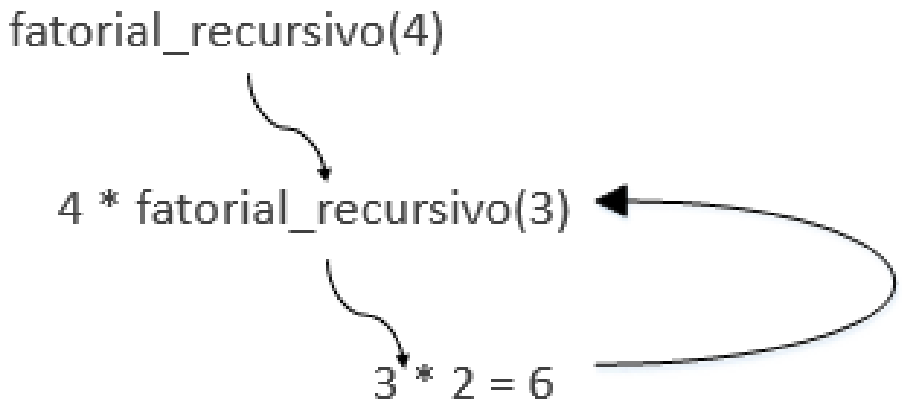

```
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



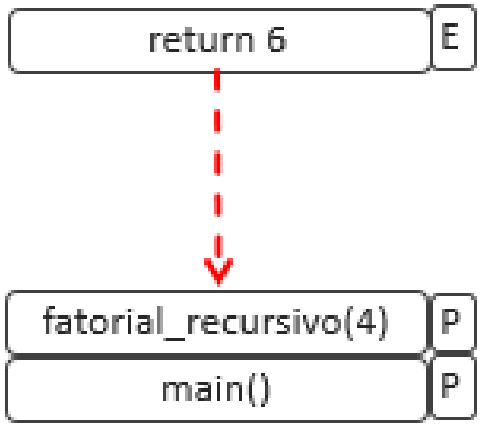
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!



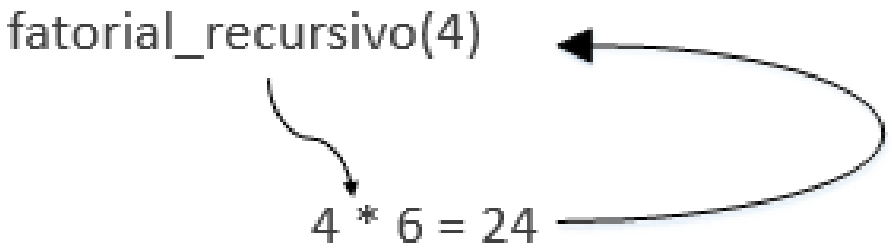

```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



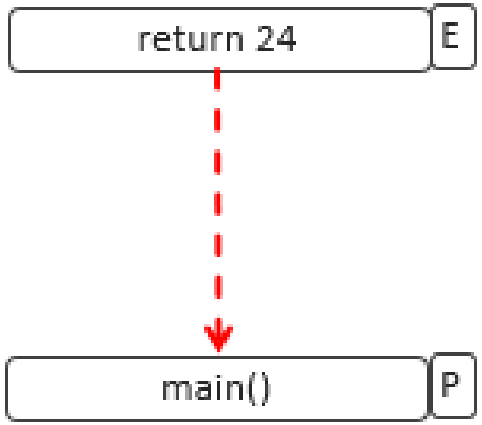
Função Recursiva

Exemplo de calculo recursivo de Fatorial

- Exemplo 4!




```c
int fatorial_recursivo(int n){
    //condição de parada
    if (n == 0 || n == 1){
        return 1;
    }
    //rechamada da função ou passo recursivo
    return n * fatorial_recursivo(n-1);
}
```



Recursividade - Vantagens

- Soluções recursivas são, geralmente, mais elegantes do que suas contrapartes iterativas;
- Cria versões mais claras e simples de vários algoritmos;
- Em grande parte dos casos a solução recursiva é menor do que a solução iterativa;
- Podem ser escritas mais rapidamente;

Recursividade - Desvantagens

- Recursão é lenta, ao chamar uma função, é necessário copiar os parâmetros e inicializar as variáveis. Isso demanda tempo;
- Recursões grandes consomem grandes quantidades de memória, pois é necessário guardar o estado das funções que estão esperando a próxima retornar;
- Um deslize na definição da condição de parada pode fazer seu programa chamar a função indefinidamente.

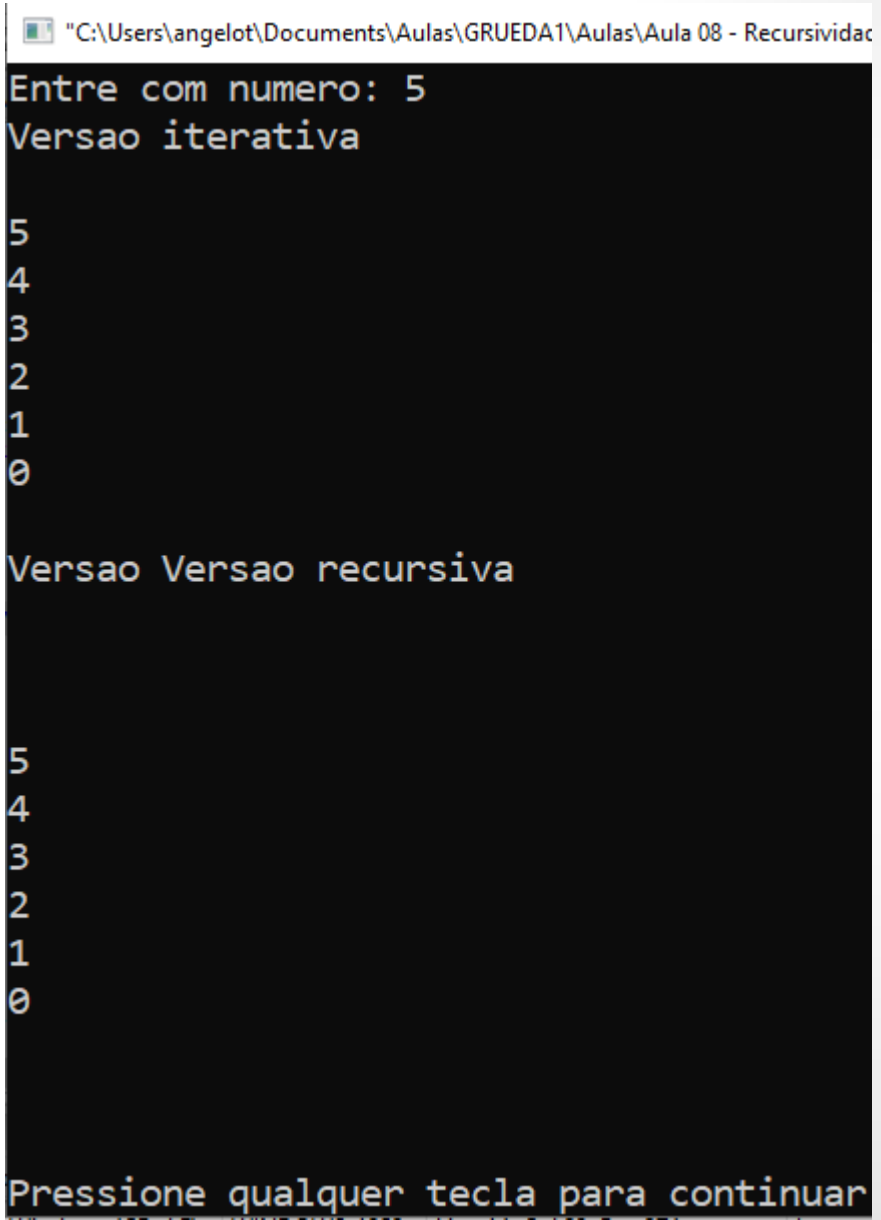
SEMPRE ASSEGURE QUE HAVERÁ UMA CONDIÇÃO DE PARADA.

Estrutura de Dados 1

Atividade 1

- Faça um programa que utilize duas funções uma iterativa e uma recursiva que recebam um valor n passado pelo usuário, e imprima em contagem regressiva a partir deste valor. Exemplo: se o usuário passar o valor 5, o programa imprimirá 5, 4, 3, 2, 1, 0;
- As funções devem ser executadas seguidamente e seus resultados exibidos de forma identificada;

- Entregue no Moodle como atividade 1.



Atividade 2

- Crie um programa para determinar o que a função recursiva abaixo calcula.

```
int func(int n){
    if(n == 0){
        return 0;
    }
    return (n + func(n-1));
}
```

- Escreva uma função iterativa para atingir o mesmo objetivo e adicione esta função no mesmo programa da função recursiva, e deixe a cargo do usuário a escolha do método de cálculo.
- Entregue no Moodle como atividade 2.