

LPG0002 – Linguagem de Programação

Funções Recursivas

Prof^a Luciana Rita Guedes
Departamento de Ciência da Computação
UDESC / Joinville

Material elaborado por: Prof. Rui Jorge Tramontin Junior

Conteúdo

1. Introdução e definição
2. Exemplo: cálculo do fatorial
3. Exemplo passo a passo
4. Implementação e execução
5. Considerações

Introdução

- Recursividade é um método para solução de problemas que depende da solução de partes menores do problema principal;

Introdução

- Recursividade é um método para solução de problemas que depende da solução de partes menores do problema principal;
- Uma função é dita **recursiva** quando faz invocações a si mesma;

Introdução

- Recursividade é um método para solução de problemas que depende da solução de partes menores do problema principal;
- Uma função é dita **recursiva** quando faz invocações a si mesma;
 - Cada invocação resolve parte do problema;

Introdução

- Recursividade é um método para solução de problemas que depende da solução de partes menores do problema principal;
- Uma função é dita **recursiva** quando faz invocações a si mesma;
 - Cada invocação resolve parte do problema;
- Quando isso ocorre, as chamadas sucessivas da mesma função têm um efeito equivalente a um processo de repetição;

Introdução

- Recursividade é um método para solução de problemas que depende da solução de partes menores do problema principal;
- Uma função é dita **recursiva** quando faz invocações a si mesma;
 - Cada invocação resolve parte do problema;
- Quando isso ocorre, as chamadas sucessivas da mesma função têm um efeito equivalente a um processo de repetição;
 - **Recursão** é equivalente à **iteração** (*repetição*).

Introdução

- Para garantir que o processo se encerre em algum momento, a função precisa verificar em que condição um *caso base* deve ocorrer;

Introdução

- Para garantir que o processo se encerre em algum momento, a função precisa verificar em que condição um caso base deve ocorrer;
- Um caso base gera um valor trivial esperado para a função, sem a necessidade de realizar a recursão;

Introdução

- Para garantir que o processo se encerre em algum momento, a função precisa verificar em que condição um caso base deve ocorrer;
- Um caso base gera um valor trivial esperado para a função, sem a necessidade de realizar a recursão;
- Um exemplo simples para ilustrar o funcionamento típico de uma função recursiva é o cálculo do fatorial.

Definição Recursiva do Fatorial

$$x! \begin{cases} 1 & , se \ x = 0 & (caso \ base) \\ x \cdot (x - 1)! & , caso \ contrário & (caso \ recursivo) \end{cases}$$

Definição Recursiva do Fatorial

$$x! \begin{cases} 1 & , se \ x = 0 & \text{(caso base)} \\ x \cdot (x - 1)! & , caso \ contr\acute{a}rio & \text{(caso recursivo)} \end{cases}$$

- O caso recursivo é onde ocorre a recursão, ou seja, a função chama a si mesma;

Definição Recursiva do Fatorial

$$x! \begin{cases} 1 & , se \ x = 0 & \text{(caso base)} \\ x \cdot (x - 1)! & , caso \ contr\acute{a}rio & \text{(caso recursivo)} \end{cases}$$

- O caso recursivo é onde ocorre a recursão, ou seja, a função chama a si mesma;
- O caso base encerra o processo recursivo;

Definição Recursiva do Fatorial

$$x! \begin{cases} 1 & , se \ x = 0 & \text{(caso base)} \\ x \cdot (x - 1)! & , caso \ contr\acute{a}rio & \text{(caso recursivo)} \end{cases}$$

- O caso recursivo é onde ocorre a recursão, ou seja, a função chama a si mesma;
- O caso base encerra o processo recursivo;
- Uma função recursiva pode ter mais de um caso base e/ou mais de um caso recursivo.

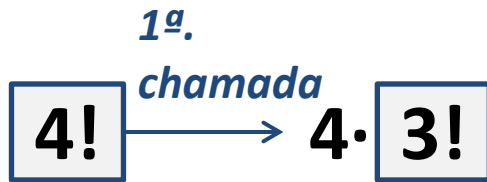
Exemplo de Recursão (1/10)

$$x! \begin{cases} 1 & , se \ x = 0 \\ x \cdot (x - 1)! & , caso \ contrário \end{cases}$$

4!

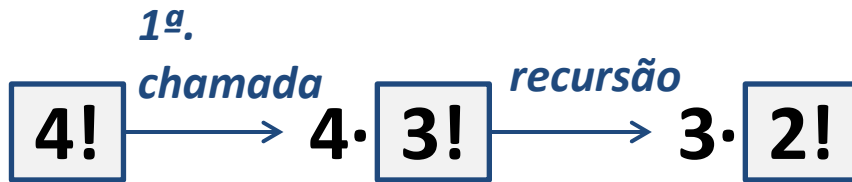
Exemplo de Recursão (2/10)

$$x! \begin{cases} 1 & , se \ x = 0 \\ x \cdot (x - 1)! & , caso \ contrário \end{cases}$$



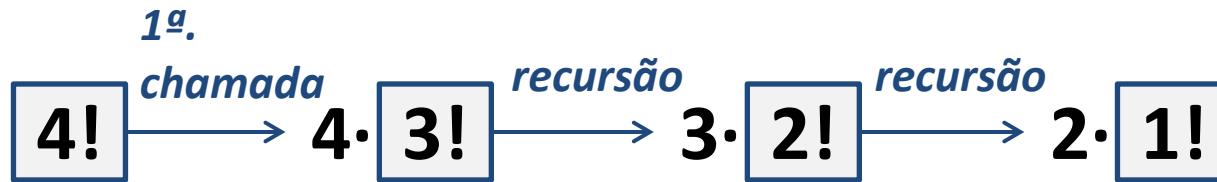
Exemplo de Recursão (3/10)

$$x! \begin{cases} 1 & , se \ x = 0 \\ x \cdot (x - 1)! & , caso \ contrário \end{cases}$$



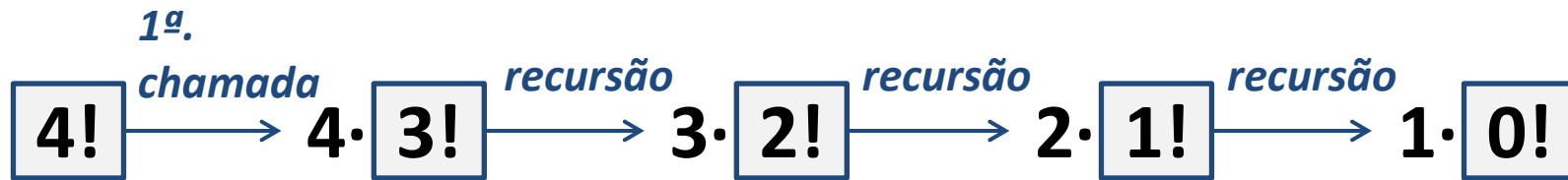
Exemplo de Recursão (4/10)

$$x! \begin{cases} 1 & , se \ x = 0 \\ x \cdot (x - 1)! & , caso \ contrário \end{cases}$$



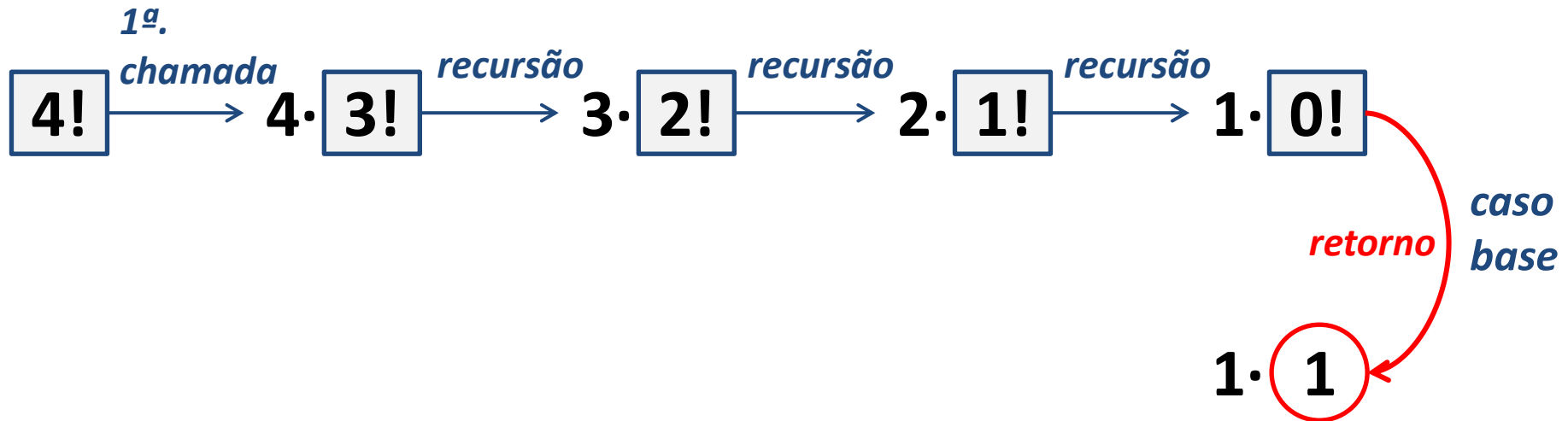
Exemplo de Recursão (5/10)

$$x! \begin{cases} 1 & , se \ x = 0 \\ x \cdot (x - 1)! & , caso \ contrário \end{cases}$$



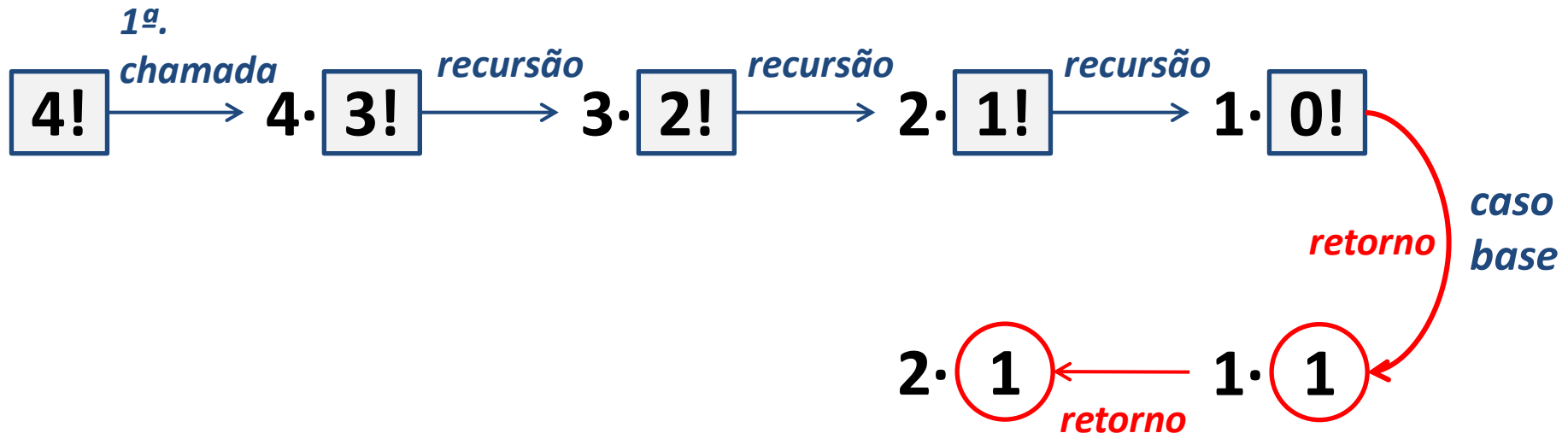
Exemplo de Recursão (6/10)

$$x! = \begin{cases} 1 & , \text{se } x = 0 \\ x \cdot (x - 1)! & , \text{caso contrário} \end{cases}$$



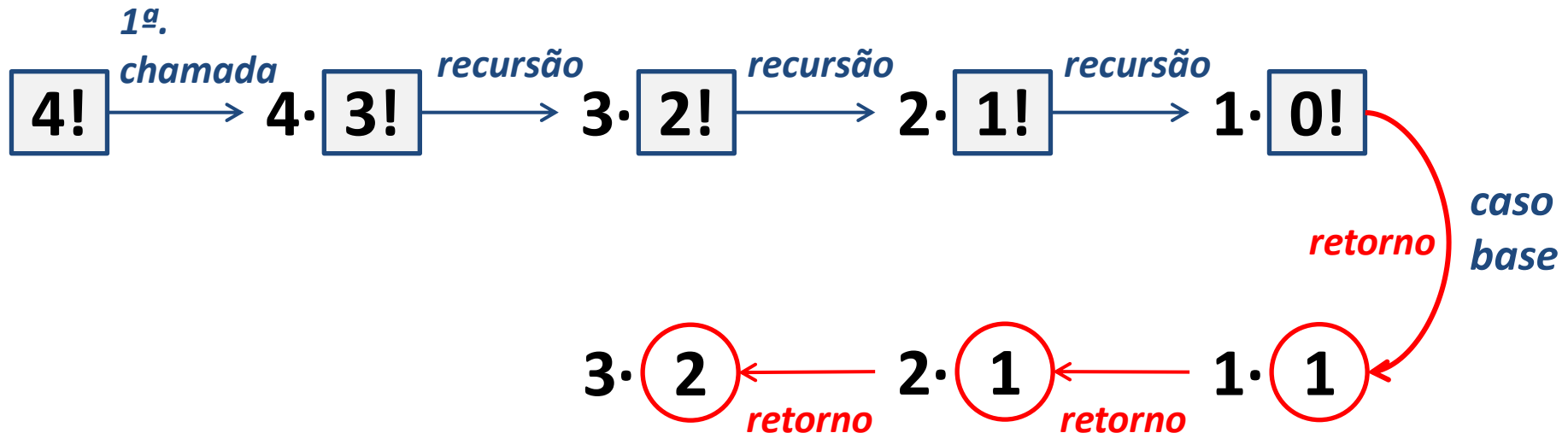
Exemplo de Recursão (7/10)

$$x! = \begin{cases} 1 & , \text{se } x = 0 \\ x \cdot (x - 1)! & , \text{caso contrário} \end{cases}$$



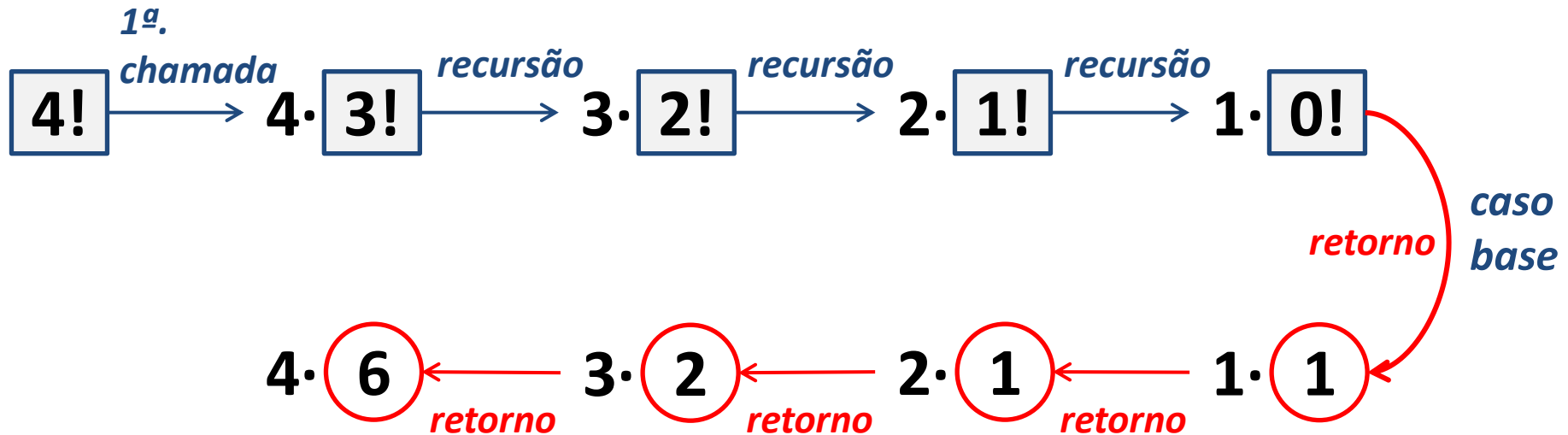
Exemplo de Recursão (8/10)

$$x! = \begin{cases} 1 & , \text{se } x = 0 \\ x \cdot (x - 1)! & , \text{caso contrário} \end{cases}$$



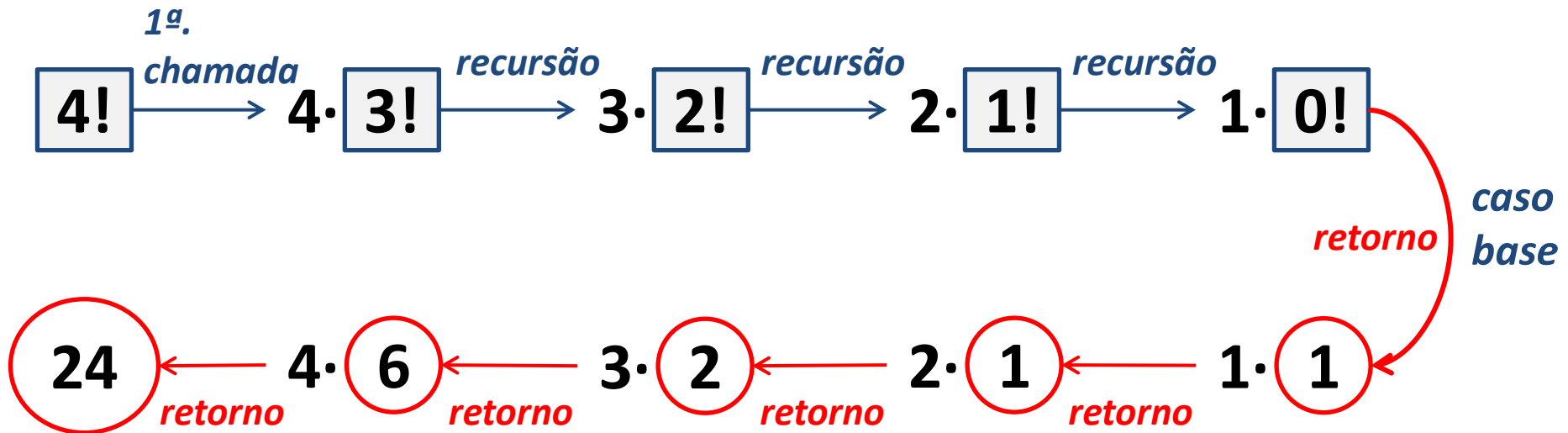
Exemplo de Recursão (9/10)

$$x! = \begin{cases} 1 & , \text{se } x = 0 \\ x \cdot (x - 1)! & , \text{caso contrário} \end{cases}$$



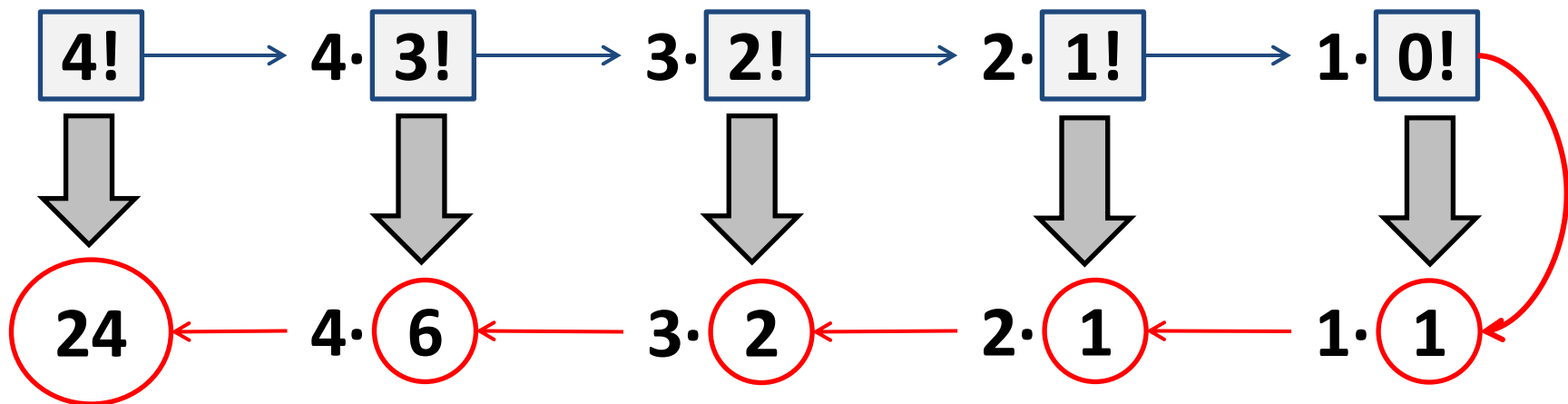
Exemplo de Recursão (10/10)

$$x! = \begin{cases} 1 & , \text{se } x = 0 \\ x \cdot (x - 1)! & , \text{caso contrário} \end{cases}$$



Exemplo de Recursão

- Cada chamada e seu respectivo retorno. (↓)



Implementação

- Uma vez definida a solução, a implementação de funções recursivas é muito simples;
 - Basta mapear cada caso de acordo com as condições;

Implementação

- Uma vez definida a solução, a implementação de funções recursivas é muito simples;
 - Basta mapear cada caso de acordo com as condições;
- Exemplo:

```
int fatorial(int x){  
    if(x == 0)  
        return 1; // caso base  
    else  
        return x * fatorial(x-1); // recursão  
}
```

Execução (1/15)

Código

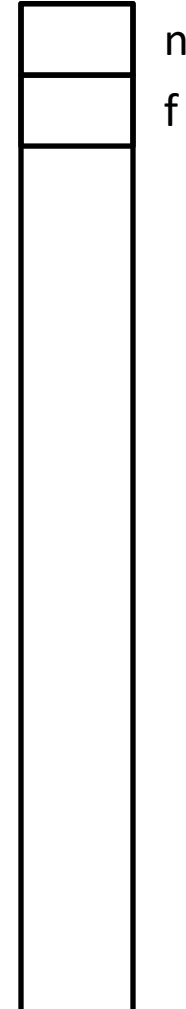
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (2/15)

Código

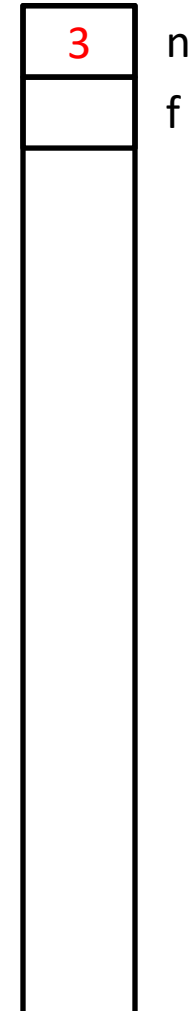
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (3/15)

Código

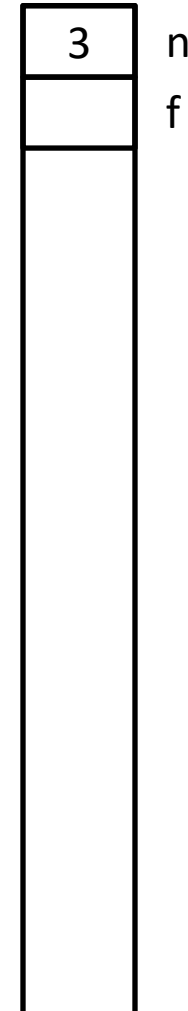
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (4/15)

Código

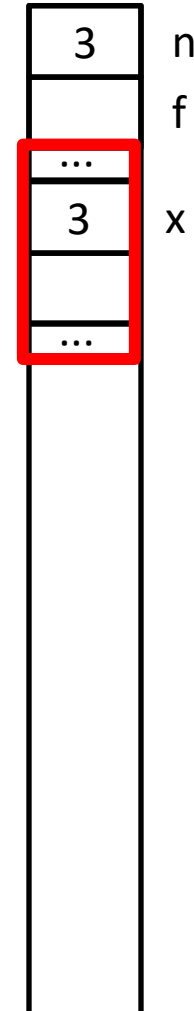
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

→ int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (5/15)

Código

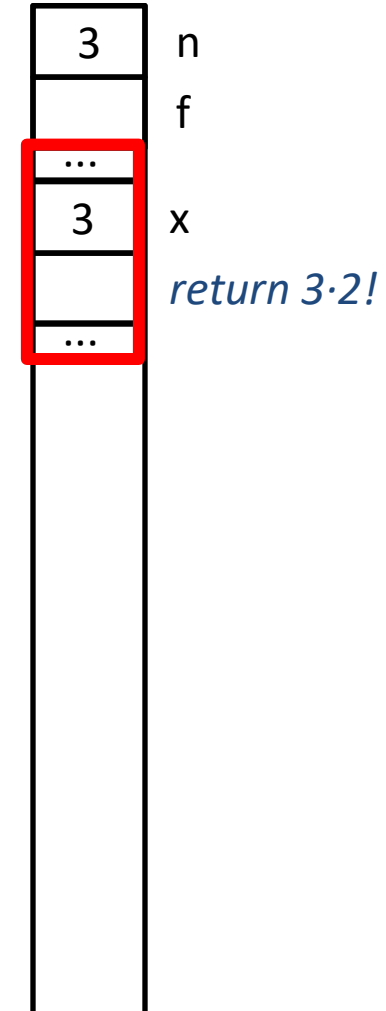
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (6/15)

Código

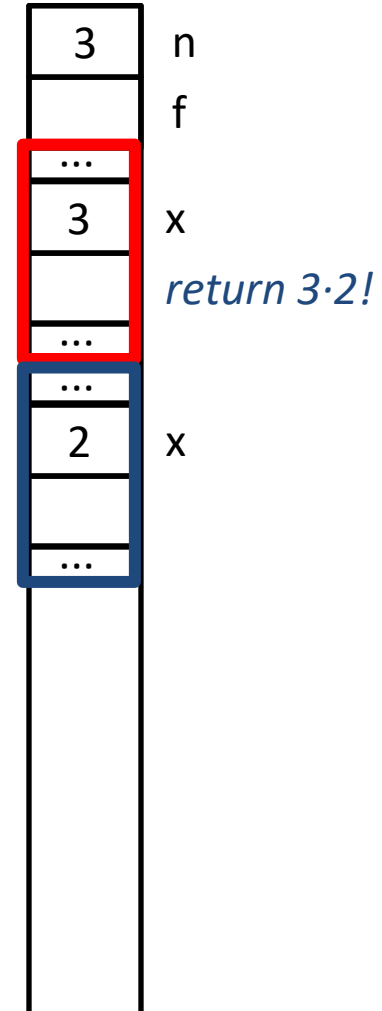
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

→ int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (7/15)

Código

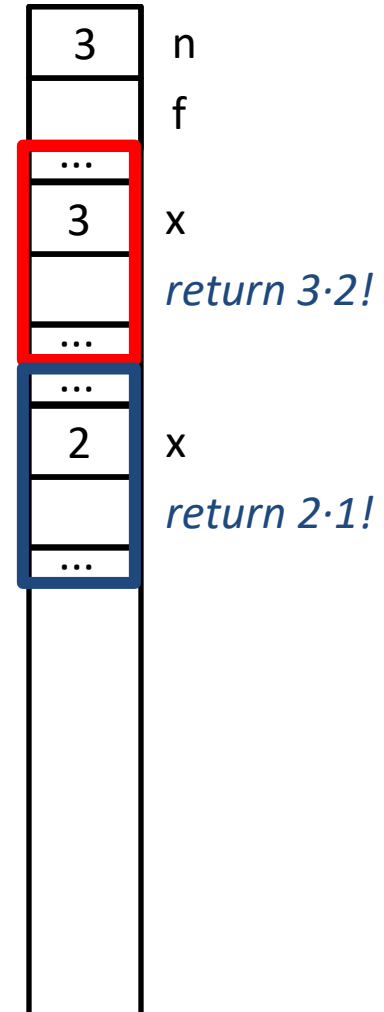
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (8/15)

Código

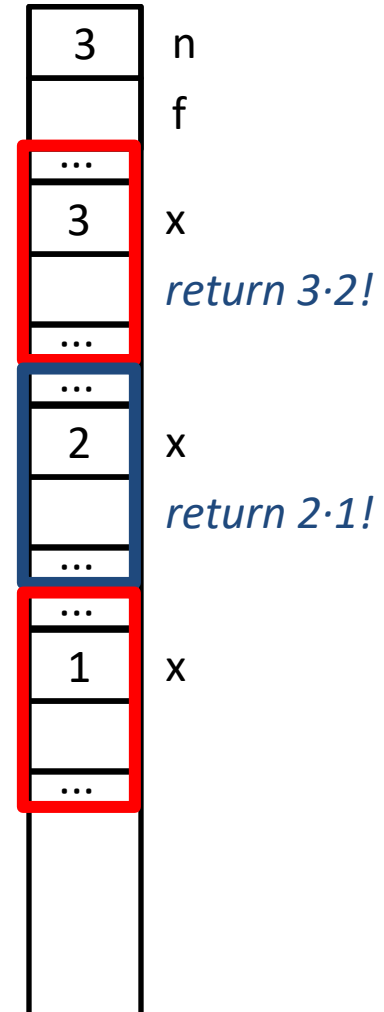
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

→ int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (9/15)


Código

```
#include <stdio.h>

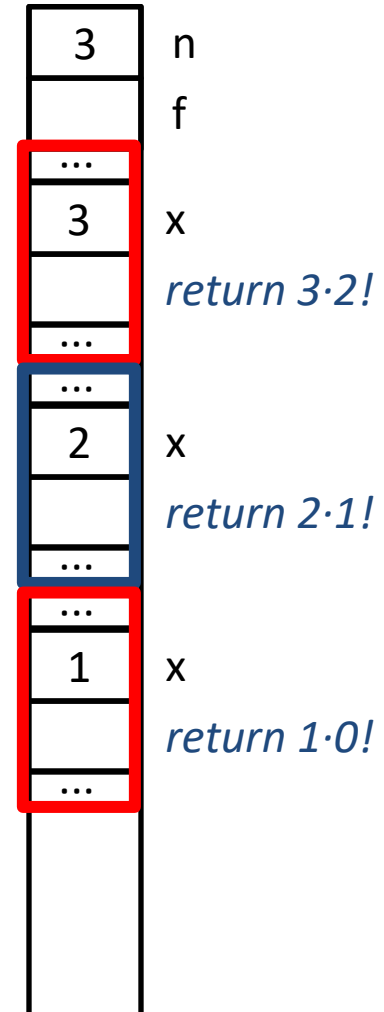
int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```



Memória (pilha)



Execução (10/15)

Código

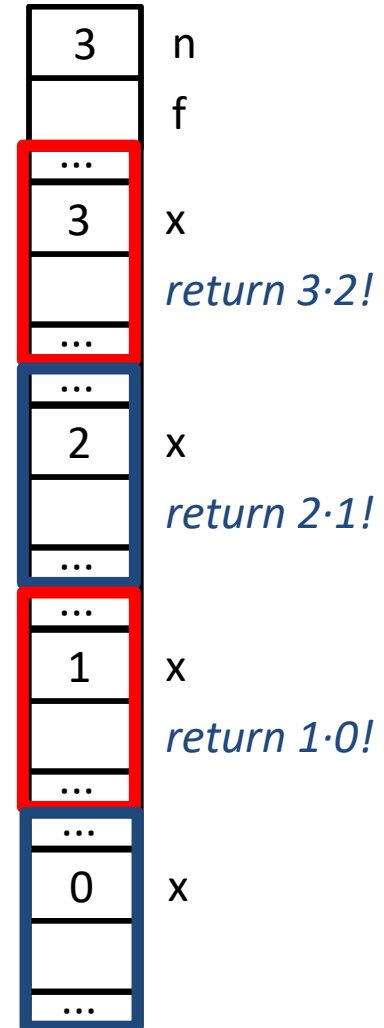
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

→ int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (11/15)

Código

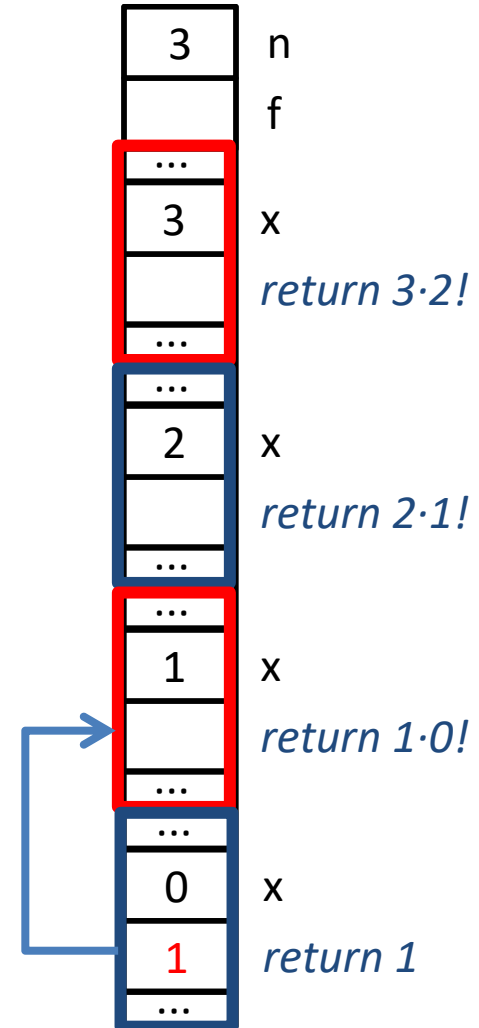
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (12/15)

Código

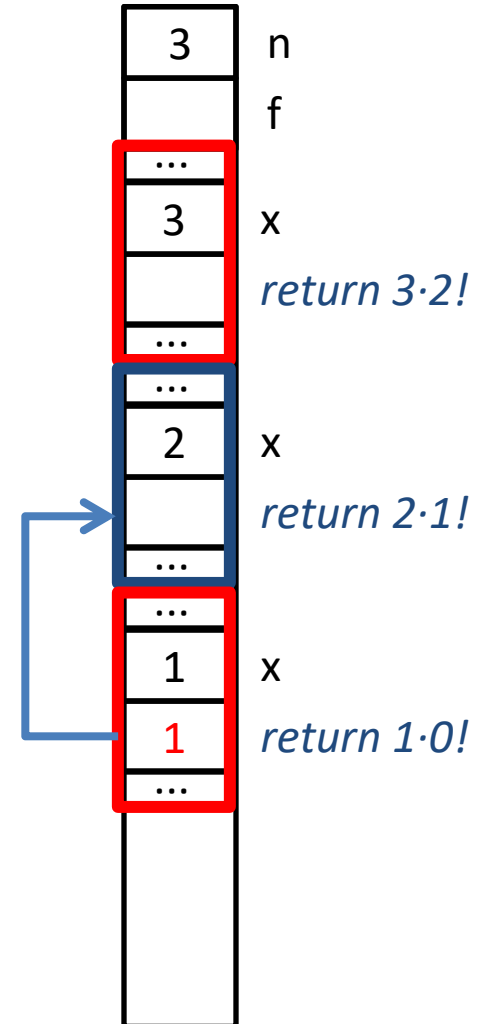
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (13/15)

Código

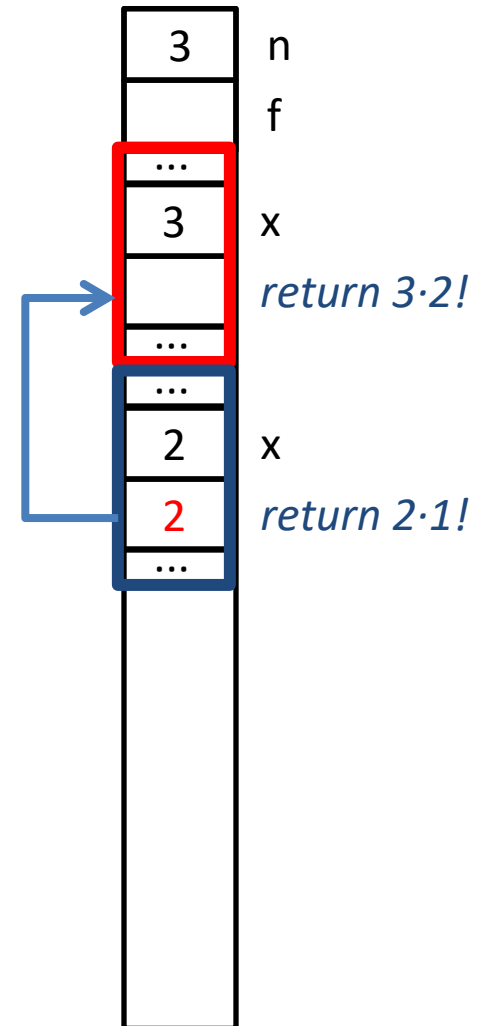
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (14/15)

Código

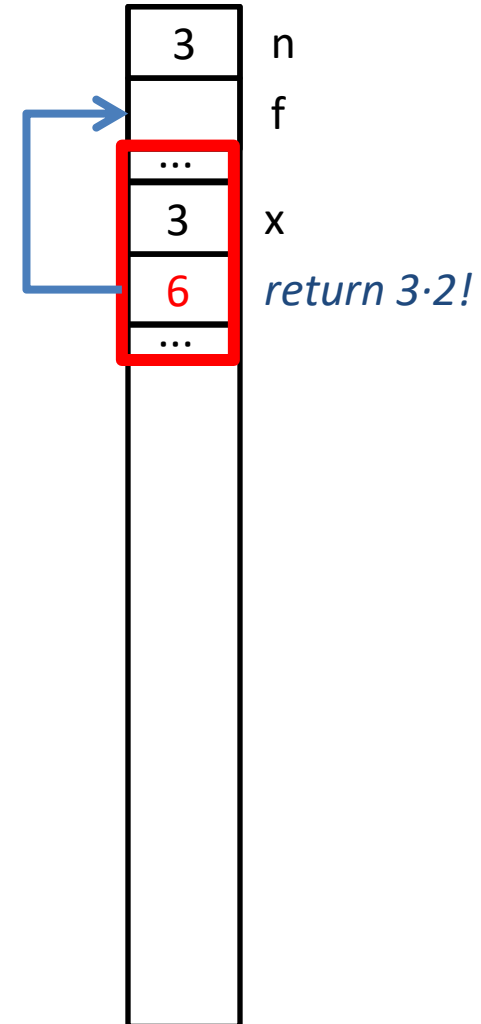
```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)



Execução (15/15)

Código

```
#include <stdio.h>

int fatorial(int x);

int main(){
    int n;
    scanf("%d", &n);
    int f = fatorial(n);
    printf("%d! = %d\n", n, f);
    return 0;
}

int fatorial(int x){
    if(x == 0)
        return 1;
    else
        return x * fatorial(x-1);
}
```

Memória (pilha)

3	n
6	f

Considerações

- O cálculo do fatorial é um exemplo muito simples;
 - Porém, serve para mostrar os princípios básicos;

Considerações

- O cálculo do fatorial é um exemplo muito simples;
 - Porém, serve para mostrar os princípios básicos;
- Outros exemplos a serem apresentados futuramente:

Considerações

- O cálculo do fatorial é um exemplo muito simples;
 - Porém, serve para mostrar os princípios básicos;
- Outros exemplos a serem apresentados futuramente:
 - Cálculo da potenciação;

Considerações

- O cálculo do fatorial é um exemplo muito simples;
 - Porém, serve para mostrar os princípios básicos;
- Outros exemplos a serem apresentados futuramente:
 - Cálculo da potenciação;
 - Geração do n -ésimo termo da sequência de Fibonacci;

Considerações

- O cálculo do fatorial é um exemplo muito simples;
 - Porém, serve para mostrar os princípios básicos;
- Outros exemplos a serem apresentados futuramente:
 - Cálculo da potenciação;
 - Geração do n -ésimo termo da sequência de Fibonacci;
 - Torres de Hanoi.

Exemplo Prático 1:

definição recursiva da potenciação

$$b^e \begin{cases} 1 & , se e = 0 & (caso base) \\ \left(\frac{1}{b}\right)^{-e} & , se e < 0 & (caso recursivo) \\ b \cdot b^{e-1} & , caso contrário & (caso recursivo) \end{cases}$$

Exemplo Prático 2:

n-ésimo termo da seq. de Fibonacci

$$fib(n) \begin{cases} 1 & , se \ n = 1 \ ou \ 2 \quad (base) \\ fib(n-1) + fib(n-2) & , caso \ contrário \quad (rec.) \end{cases}$$

Considerações

- Pensamento recursivo nem sempre é trivial;
- Entretanto, pode ser mais fácil implementar um algoritmo recursivamente;
 - *Torres de Hanoi*;
- Problemas e limitações:
 - Gasto de memória (a cada recursão);
 - Processamento desnecessário (*Fibonnacci*).