



**UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI**  
**CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB**  
**CURSO DE SISTEMAS DE INFORMAÇÃO**  
**PICOS - PI**

# **RECURSIVIDADE**

Prof. Ma. Luana Batista da Cruz  
[luana.b.cruz@nca.ufma.br](mailto:luana.b.cruz@nca.ufma.br)

# Roteiro

2

- Recursividade
  - Direta e indireta
  - Com cauda e sem cauda

# Conceito de recursividade

3

- ❑ Repetição pode ser obtida de 2 maneiras:
  - Laços (for, while, do while)
  - É também um processo repetitivo caracterizado pelas chamadas a si mesma
  
- ❑ **Recursividade**
  - É uma técnica de programação na qual um método chama a si mesmo
  - Define conjuntos infinitos com comandos finitos
  - A recursividade é uma estratégia que pode ser utilizada sempre que o cálculo de uma função para o valor  $n$ , pode ser descrita a partir do cálculo desta mesma função para o termo anterior ( $n-1$ )

# Recursividade

4

## □ Estrutura básica

```
void ex_recursao(int n){  
    if (condicao_parada) //caso base  
        return;  
    else  
        ex_recursao(n-1); //recursão  
}
```

# Recursividade

5

## □ Estrutura básica

```
void ex_recursao(int n){  
    if (condicao_parada) //caso base  
        return;  
    else  
        ex_recursao(n-1); //recursão  
}
```

loop infinito

altera o estado da variável

# Recursividade

## Exemplo

6

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else  
        ate_zero(n-1); //recursão  
}
```

# Recursividade

7

- ❑ **Recursão direta** – quando uma função chama a si mesma diretamente
- ❑ **Recursão indireta** – quando uma função chama outra, e esta, por sua vez chama a primeira
- ❑ Para cada chamada de uma função, recursiva ou não, os parâmetros e as variáveis locais são empilhados na pilha de execução
- ❑ A recursividade nem sempre é a melhor solução, mesmo quando a definição matemática do problema é feita em termos recursivos

# Recursividade

## Direta

8

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else  
        ate_zero(n-1); //recursão  
}
```



# Recursividade

## Indireta

9

### ❏ Função para mostrar o antecessor e sucessor de um número

```
#include <stdio.h>
#include <stdlib.h>
int n = 0;

void antecessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O antecessor de %d = %d\n", n, n-1);
        sucessor(); //recursão
    }
}

void sucessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O sucessor de %d = %d\n", n, n+1);
        antecessor(); //recursão
    }
}

int main() {
    antecessor();
    return 0;
}
```

# Recursividade

## Indireta

10

### ❏ Função para mostrar o antecessor e sucessor de um número

```
#include <stdio.h>
#include <stdlib.h>
int n = 0;

void antecessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O antecessor de %d = %d\n", n, n-1);
        sucessor(); //recursão
    }
}

void sucessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O sucessor de %d = %d\n", n, n+1);
        antecessor(); //recursão
    }
}

int main() {
    antecessor();
    return 0;
}
```

Não altera o  
estado das  
variáveis

# Recursividade

## Indireta

11

### ❏ Função para mostrar o antecessor e sucessor de um número

```
#include <stdio.h>
#include <stdlib.h>
int n = 0;
```

```
void antecessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O antecessor de %d = %d\n", n, n-1);
        sucessor(); //recursão
    }
}
```

```
}
void sucessor() {
    if (n > 10) //caso base
        return;
    else{
        printf("O sucessor de %d = %d\n", n, n+1);
        n++;
        antecessor(); //recursão
    }
}
```

```
}
int main() {
    antecessor();
    return 0;
}
```

Não altera o estado das variáveis

Altera o estado da variável

# Recursividade

12

- ❑ **Recursão com cauda** – é um tipo especial de recursão, no qual não existe processamento a ser feito depois de encerrada a chamada recursiva. Sendo assim, não é necessário guardar o estado do processamento no momento da chamada recursiva
- ❑ **Recursão sem cauda** – a cada chamada recursiva realizada, é necessário guardar a posição do código onde foi feita a chamada para que continue a partir dali assim que receber o resultado

# Recursividade com cauda

## Exemplo

13

- ❑ Função para mostrar os números de **n até 0**:

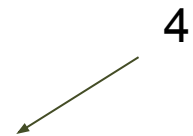
```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

# Recursividade com cauda

## Exemplo

14

- ❑ Função para mostrar os números de **n até 0**:



```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

# Recursividade com cauda

## Exemplo

15

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

4

# Recursividade com cauda

## Exemplo

16

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

4 3



# Recursividade com cauda

## Exemplo

17

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

4 3 2

# Recursividade com cauda

## Exemplo

18

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        printf("%d ", n);  
        ate_zero(n-1); //recursão  
    }  
}
```

4 3 2 1

# Recursividade sem cauda

## Exemplo

19

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

# Recursividade sem cauda

## Exemplo

20

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1  
                ate_zero(0)
```

# Recursividade sem cauda

## Exemplo

21

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1
```

# Recursividade sem cauda

## Exemplo

22

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1
```

# Recursividade sem cauda

## Exemplo

23

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1
```

1 2

# Recursividade sem cauda

## Exemplo

24

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1
```

1 2 3



# Recursividade sem cauda

## Exemplo

25

- ❑ Função para mostrar os números de **n até 0**:

```
void ate_zero(int n){  
    if (n == 0) //caso base  
        return;  
    else{  
        ate_zero(n-1); //recursão  
        printf("%d ", n);  
    }  
}
```

```
ate_zero(4) → 4  
    ate_zero(3) → 3  
        ate_zero(2) → 2  
            ate_zero(1) → 1
```

1 2 3 4

# Recursividade

26

- ❑ Internamente, quando qualquer chamada de função é feita dentro de um programa, é criado um **Registro de Ativação** na **Pilha de Execução** do programa
- ❑ O registro de ativação armazena os parâmetros e variáveis locais da função bem como o “ponto de retorno” no programa ou subprograma que chamou essa função
- ❑ Ao final da execução dessa função, o registro é desempilhado e a execução volta ao subprograma que chamou a função

# Recursividade

## Exemplo

27

### ❑ Fatorial

- É um número natural **inteiro positivo**, o qual é representado por  $n!$
- O fatorial de um número é calculado pela multiplicação desse número por todos os seus antecessores até chegar ao número 1

# Recursividade

## Exemplo

28

### Fatorial


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

# Recursividade

## Exemplo

29

### ❏ Fatorial

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

$$4!$$

# Recursividade

## Exemplo

30

### ❏ Fatorial

$$5! = 5 \times 4 \times \overbrace{3 \times 2 \times 1}^{3!}$$

$\underbrace{\hspace{10em}}_{4!}$

# Recursividade

## Exemplo

31

### ❑ Fatorial (tradicional)

- Se  $N \leq 0 \rightarrow 1$
- Se  $N > 0 \rightarrow N1 \times N2 \times N3 \times \dots \times N$

### ❑ Fatorial (recursivo)

- Se  $N \leq 0 \rightarrow 1$
- Se  $N > 0 \rightarrow N \times \text{Fat}(N - 1)$


# Recursividade

## Exemplo

32

### ❏ Fatorial

Tradicional




```
int fat(int n) {
    int resultado = 1;
    for (int i=n; i>0; i--)
        resultado = resultado * i;

    return resultado;
}

int main() {
    int f;
    f = fat(3);
    printf("%d", f);
}
```

Recursivo



```
int fat(int n) {
    if (n<=0)
        return 1;
    else
        return n * fat(n-1);
}

int main() {
    int f;
    f = fat(3);
    printf("%d", f);
}
```



# Recursividade

## Exemplo

33

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

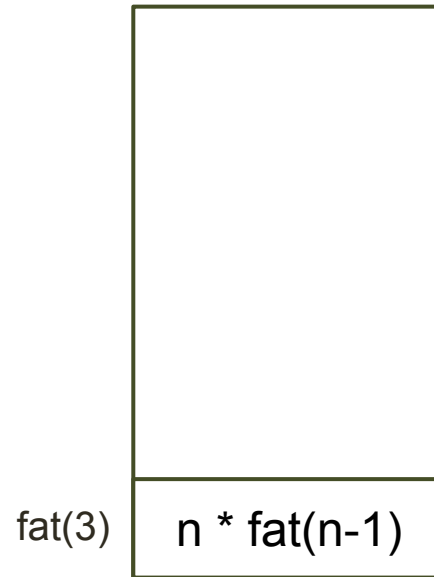
34

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

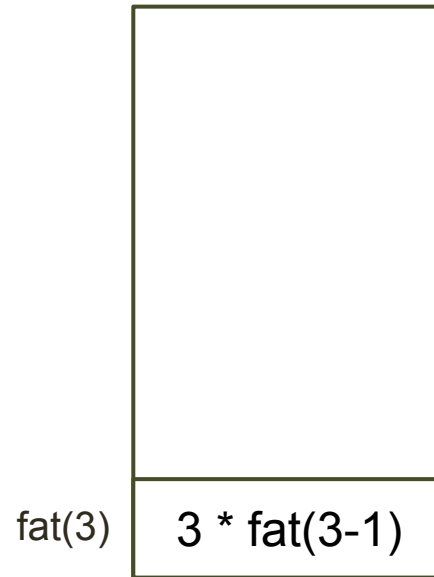
35

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

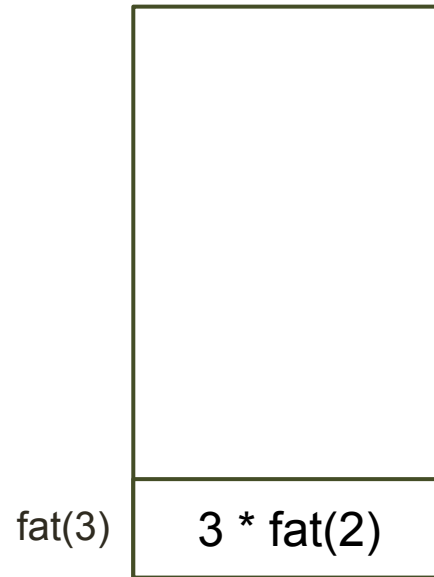
36

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

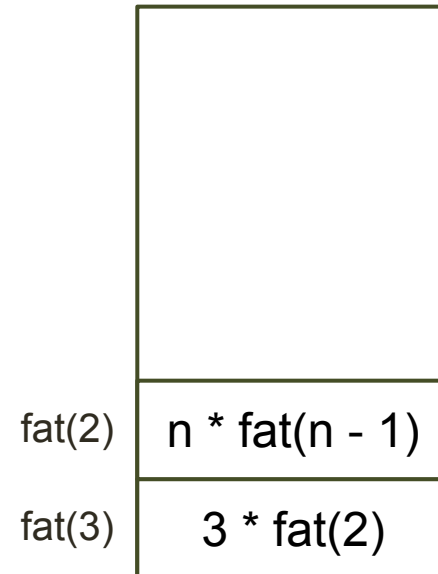
37

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

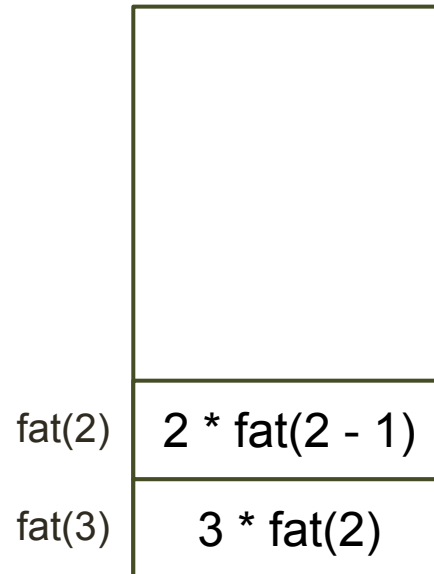
38

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

39

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

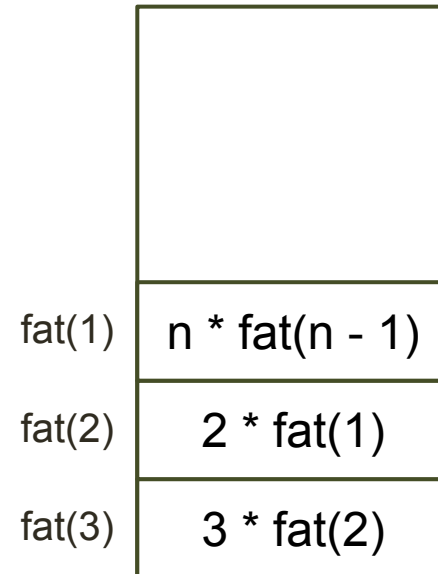
40

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução





# Recursividade

## Exemplo

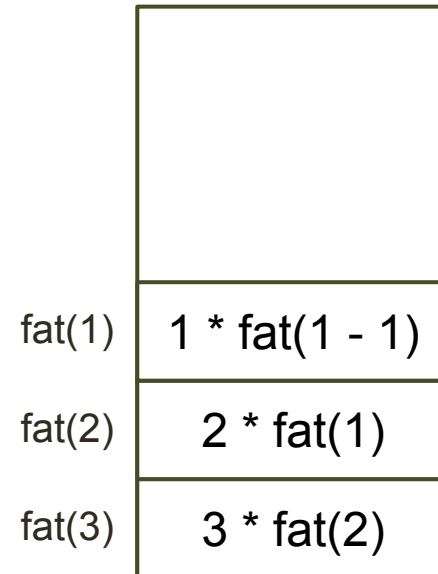
41

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

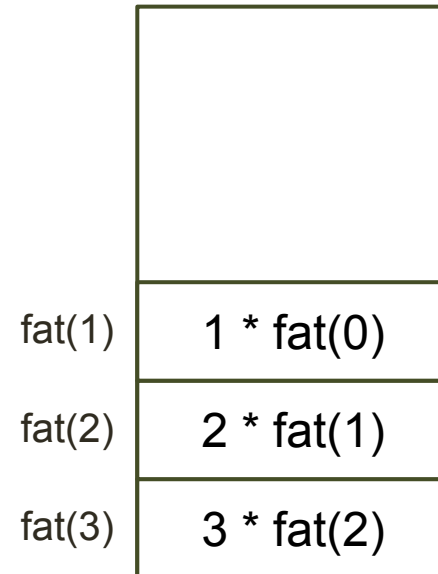
42

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

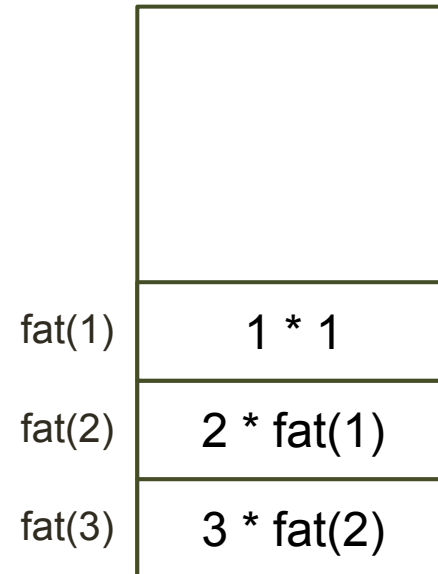
43

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

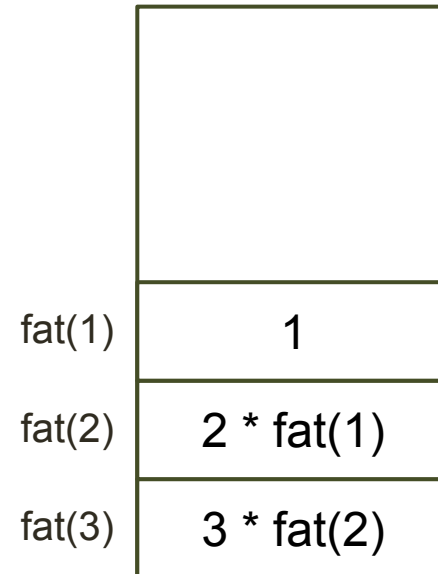
44

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

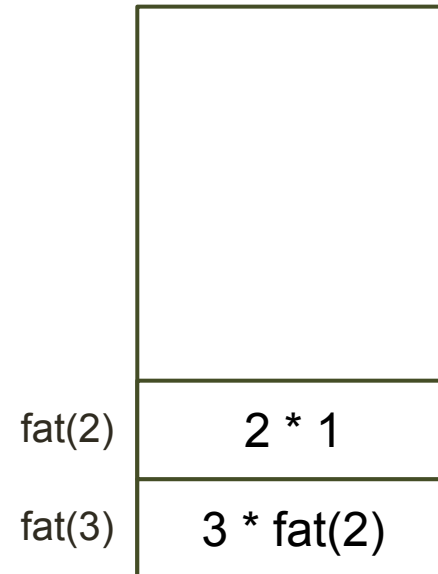
45

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

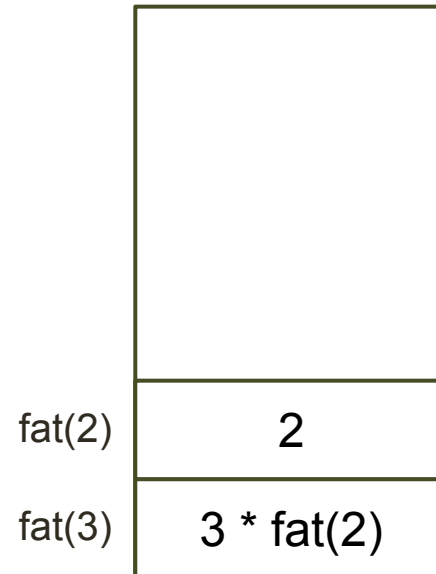
46

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

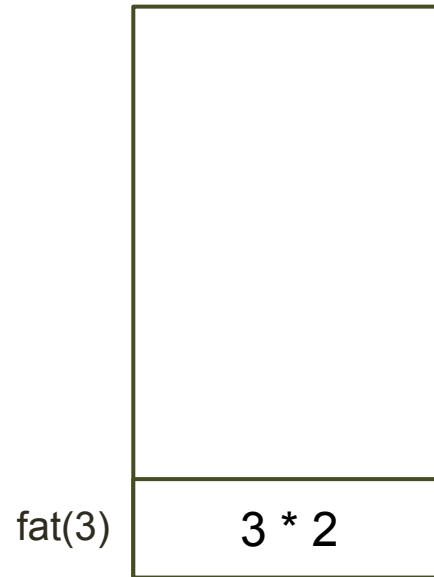
47

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

Pilha de execução



# Recursividade

## Exemplo

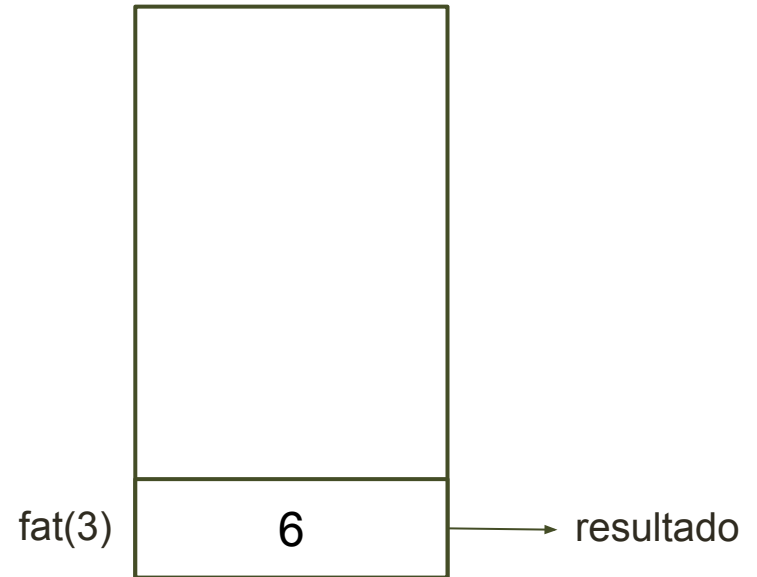
48

### ❏ Fatorial

```
int fat(int n) {  
    if (n<=0)  
        return 1;  
    else  
        return n * fat(n-1);  
}
```

```
int main() {  
    int f;  
    f = fat(3);  
    printf("%d", f);  
}
```

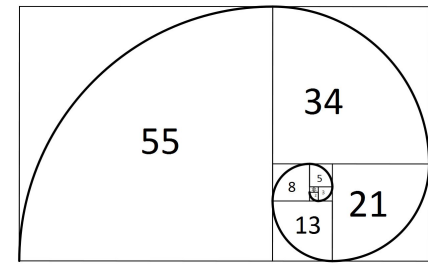
Pilha de execução





# Recursividade

## Exemplo



49

□ Outro exemplo (Série de Fibonacci):

-  $F_0 \rightarrow F_1 \rightarrow 1$

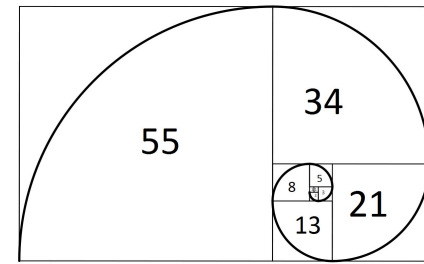
-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$





# Recursividade

## Exemplo



52

□ Outro exemplo (Série de Fibonacci):

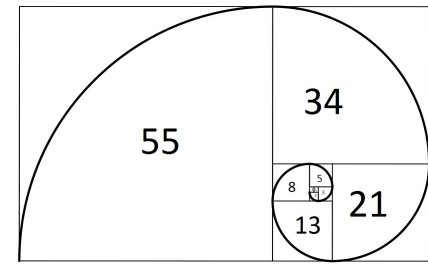
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2								
1	1	2								

# Recursividade

## Exemplo



53

□ Outro exemplo (Série de Fibonacci):

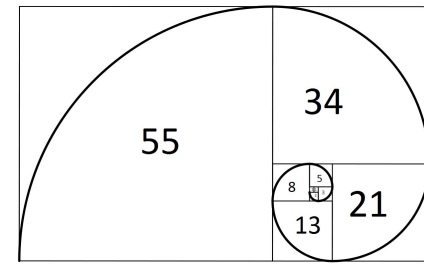
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3							
1	1	2								

# Recursividade

## Exemplo



54

□ Outro exemplo (Série de Fibonacci):

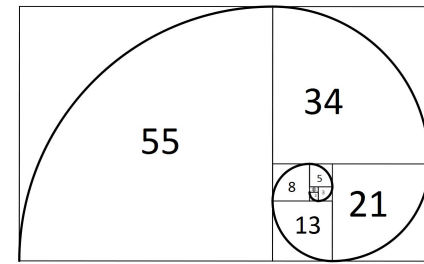
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3							
1	1	2	3							

# Recursividade

## Exemplo



55

□ Outro exemplo (Série de Fibonacci):

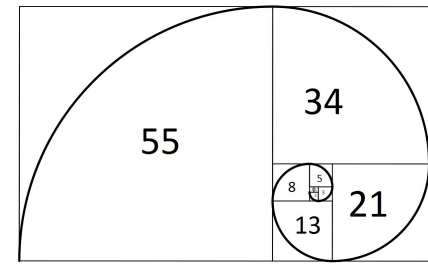
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4						
1	1	2	3							

# Recursividade

## Exemplo



56

□ Outro exemplo (Série de Fibonacci):

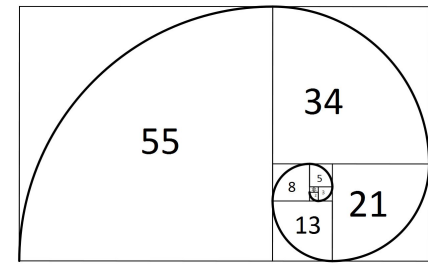
- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4						
1	1	2	3	5						



# Recursividade

## Exemplo



57

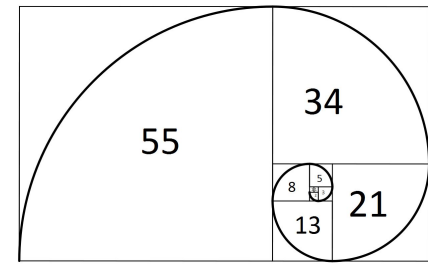
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5					
1	1	2	3	5						

# Recursividade

## Exemplo



58

□ Outro exemplo (Série de Fibonacci):

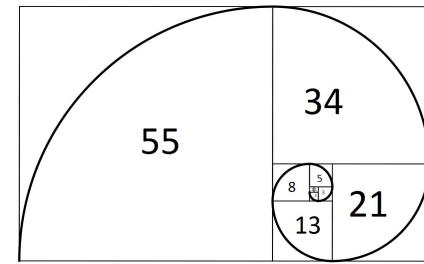
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5					
1	1	2	3	5	8					

# Recursividade

## Exemplo



59

□ Outro exemplo (Série de Fibonacci):

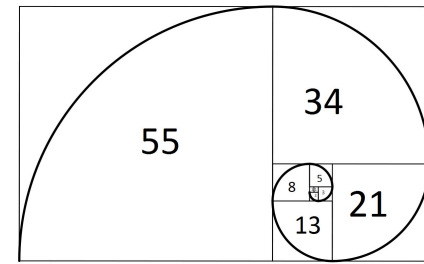
-  $F_0 \rightarrow F_1 \rightarrow 1$

-  $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6				
1	1	2	3	5	8					

# Recursividade

## Exemplo



60

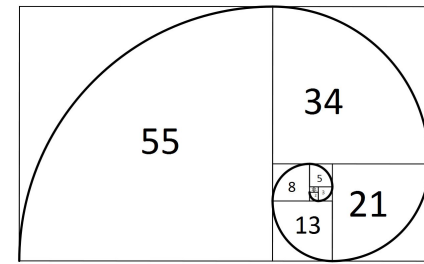
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6				
1	1	2	3	5	8	13				

# Recursividade

## Exemplo



61

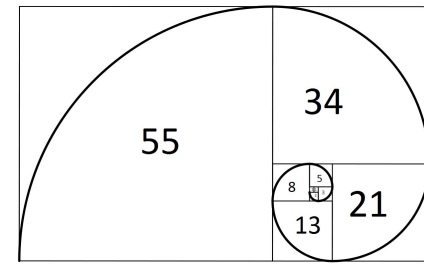
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7			
1	1	2	3	5	8	13				

# Recursividade

## Exemplo



62

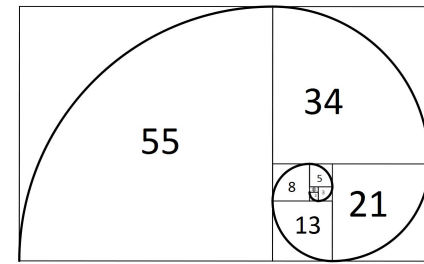
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7			
1	1	2	3	5	8	13	21			

# Recursividade

## Exemplo



63

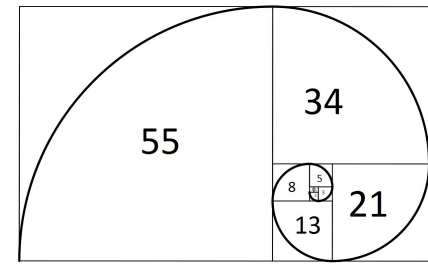
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8		
1	1	2	3	5	8	13	21			

# Recursividade

## Exemplo



64

□ Outro exemplo (Série de Fibonacci):

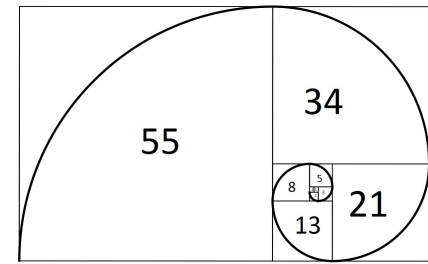
- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8		
1	1	2	3	5	8	13	21	34		



# Recursividade

## Exemplo



65

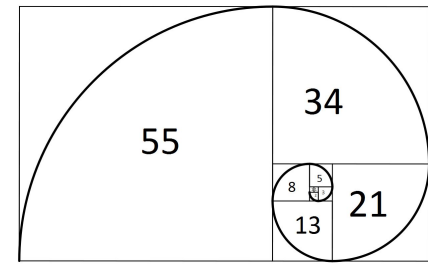
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8	9	
1	1	2	3	5	8	13	21	34		

# Recursividade

## Exemplo



66

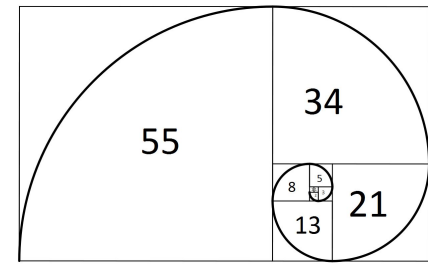
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8	9	
1	1	2	3	5	8	13	21	34	55	

# Recursividade

## Exemplo



67

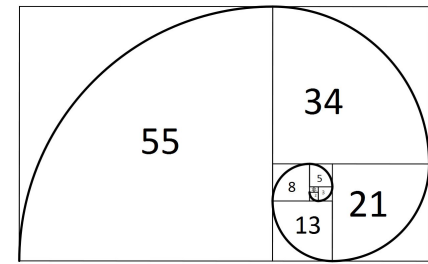
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	5	8	13	21	34	55	

# Recursividade

## Exemplo



68

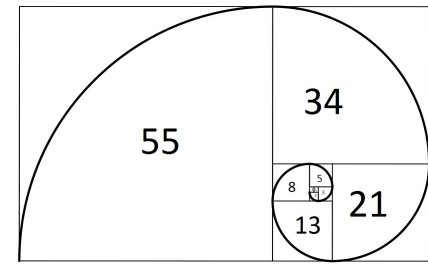
□ Outro exemplo (Série de Fibonacci):

- $F_0 \rightarrow F_1 \rightarrow 1$
- $F_n \rightarrow F_{n-1} + F_{n-2} \quad n > 1,$

0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	5	8	13	21	34	55	89

# Recursividade

## Exemplo



69

### ❏ Outro exemplo (Série de Fibonacci):

```
int fib(int n) {  
    if (n<2)  
        return 1;  
    else  
        return fib(n-1) + fib(n-2);  
}
```

```
int main() {  
    int f;  
    f = fib(4);  
    printf("%d", f);  
}
```



# Recursividade

71

## ❑ Vantagens

- A utilização de uma função recursiva pode simplificar a solução de alguns problemas
- Pode-se obter um código mais conciso e eficaz nessas situações
- Redução do tamanho do código fonte
- Uma solução recursiva pode, por outro lado, eliminar a necessidade de manter o controle manual sobre uma série de variáveis normalmente associadas aos métodos alternativos à recursividade
- Maior clareza do algoritmo para problemas de definição **naturalmente recursiva**

# Recursividade

72

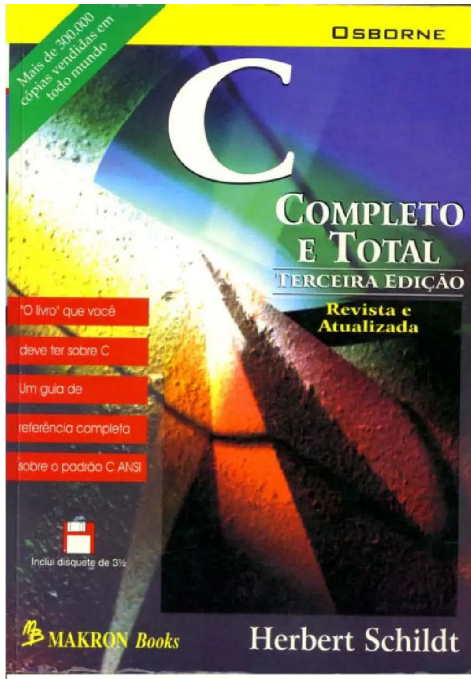
## ❑ Desvantagens

- As funções recursivas são geralmente mais lentas e ocupam mais memória do que as funções iterativas equivalentes, uma vez que são feitas muitas chamadas consecutivas a funções
- Dificuldade de depuração dos subprogramas recursivos, principalmente se a recursão for muito profunda
- Um erro de implementação pode levar ao esgotamento dos recursos associados à pilha que gere a chamada a funções



# Referências

73



SCHILD, Herbert. **C completo e total**. Makron, 3ª edição revista e atualizada, 1997.



SZWARCHFITER, J. **Estruturas de Dados e seus algoritmos**. 3 ed. Rio de Janeiro: LTC, 2010.