Algoritmos II

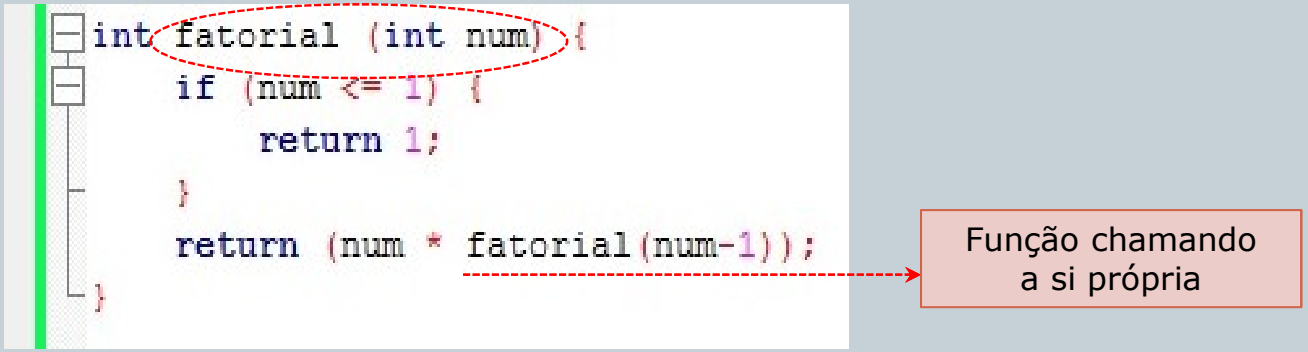
RECURSIVIDADE

Conceito

- Repetição pode ser obtida de 2 maneiras:
  - Laços (for, while, etc).
  - Chamada recursiva de métodos (recursão).
- Funções recursivas são funções que chamam a si mesmas para compor a solução de um problema.
- Uma função é dita recursiva quando dentro do seu código existe uma chamada para si mesma.

Recursão Direta

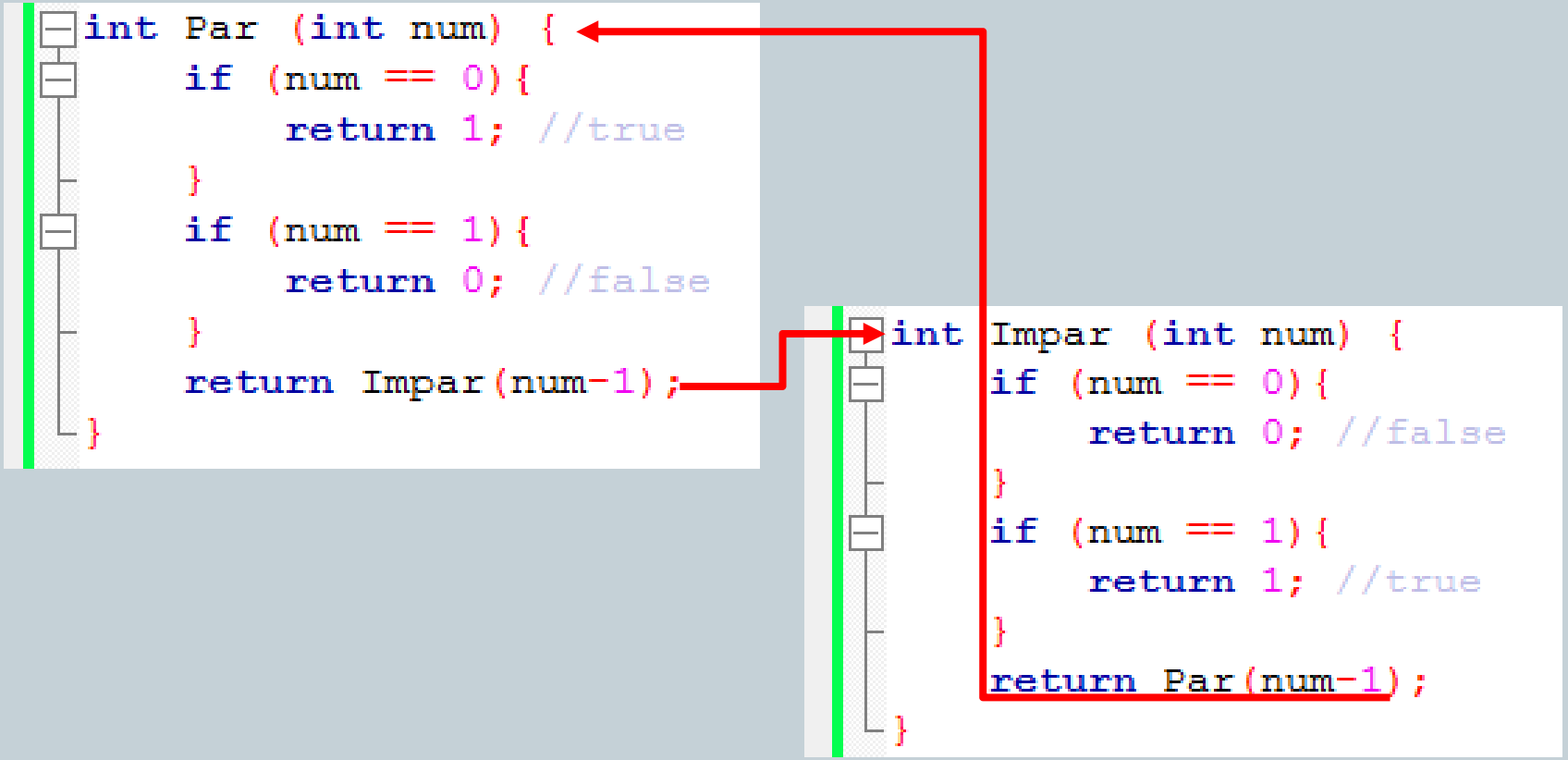
- Quando uma função chama a si mesma diretamente.



Recursão Indireta

- Recursão indireta – quando uma função chama outra, e esta, por sua vez chama a primeira.
  - Exemplo: verificar se um valor é um número Par ou um número Ímpar.
  - Definição de valor Par: número divisível por 2.
  - Definição de valor Ímpar: número não divisível por 2.
  - Definição recursiva de número Par: 0 é Par. n é Par. N é Par se n-1 é ímpar.
  - Definição recursiva de número Ímpar: 1 é Ímpar. N é Ímpar. N é Ímpar se n-1 é Par.

Recursão Indireta



Tipos de Recursão

- Linear: faz somente uma chamada recursiva (uma chamada a si mesmo). Exemplo: return (num * fatorial (num-1)).

- Binária: existem duas chamadas recursivas para cada caso não básico. Exemplo: return (Fibonacci(num-1) + Fibonacci(num-2));

- Múltiplas Chamadas: quando faz mais de duas chamadas recursivas. Exemplo: return (X(num-1) + X(num-2) + X (num-3));

Vantagens e Desvantagens

- Vantagens da recursão
  - Redução do tamanho do código fonte.
  - Maior clareza do algoritmo para problemas de definição naturalmente recursiva.
- Desvantagens da recursão
  - Baixo desempenho na execução devido ao tempo para gerenciamento das chamadas.
  - Dificuldade de depuração dos subprogramas recursivos, principalmente se a recursão for muito profunda.
- Resumindo
  - Na maioria das vezes a codificação na forma recursiva é mais simples (reduzida), mas a forma interativa tende a ser mais eficiente.

Recursividade

- Para todo algoritmo recursivo existe um outro correspondente iterativo (não recursivo), que executa a mesma tarefa.

Como criar uma Função Recursiva

1. Encontrar uma solução de como um problema pode ser dividido em passos menores.
2. Definir uma Regra Geral que seja válida para todos os outros casos.
3. Definir o Ponto de Parada.
4. Verificar se o Ponto de Parada é atingido, ou seja, o algoritmo não entrará em um looping infinito.

Construindo uma Função Recursiva

- Um dos exemplos mais utilizados para explicação de Recursividade é o Fatorial. O símbolo de Fatorial é !.

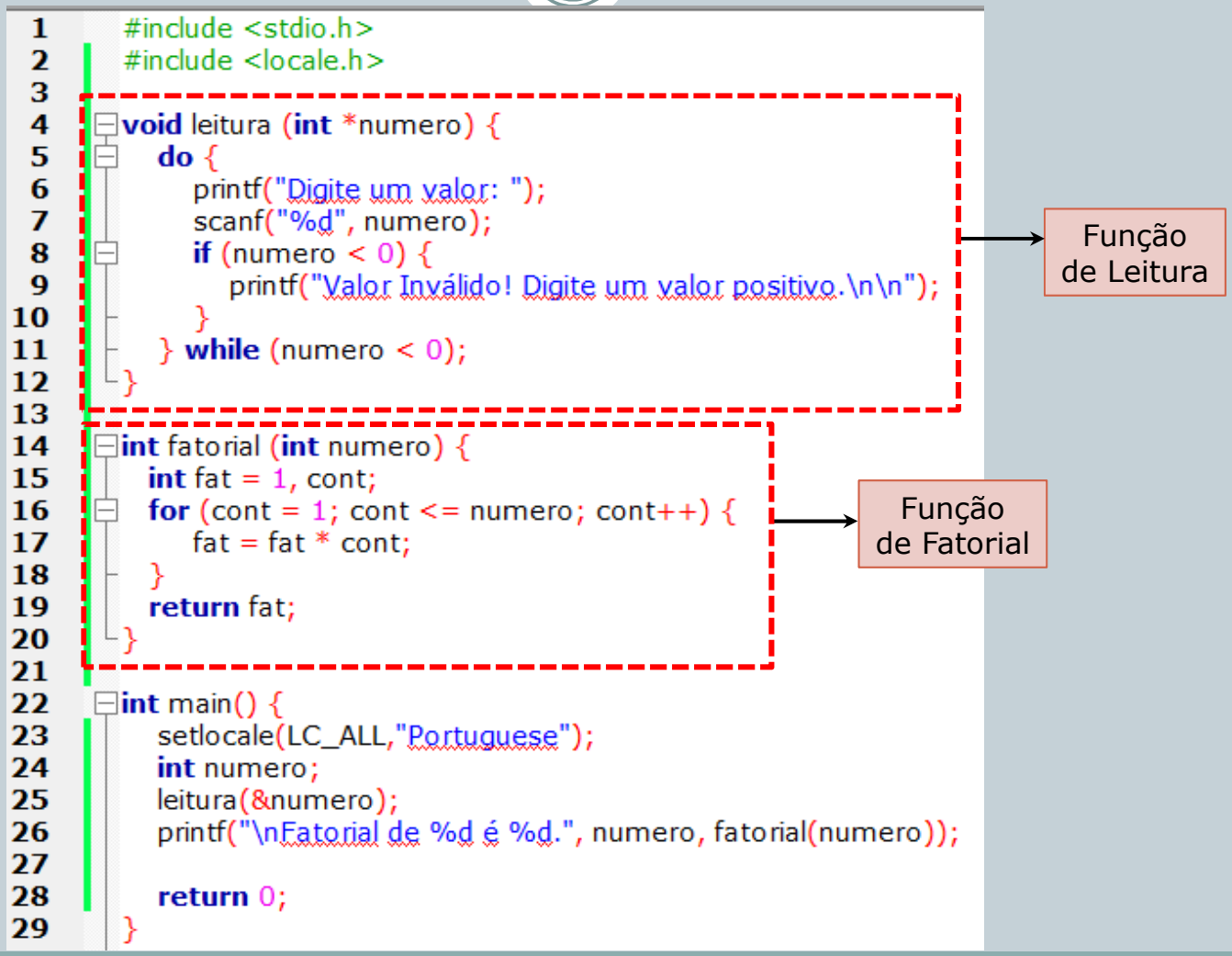
- **5!** = 5 * 4 * 3 * 2 * 1 = 120
- **4!** = 4 * 3 * 2 * 1 = 24
- **3!** = 3 * 2 * 1 = 6
- **2!** = 2 * 1 = 2
- **1!** = 1 e **0!** = 1

ou seja

N! = 1, se N <= 0.

*N! = 1 * 2 * 3 * … * N se N > 0.*

Funções não Recursivas (Fatorial)



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

void leitura (int *numero) {
    do {
        printf("Digite um valor: ");
        scanf("%d", numero);
        if (numero < 0) {
            printf("Valor Inválido! Digite um valor positivo.\n\n");
        }
    } while (numero < 0);
}

int fatorial (int numero) {
    int fat = 1, cont;
    for (cont = 1; cont <= numero; cont++) {
        fat = fat * cont;
    }
    return fat;
}

int main() {
    setlocale(LC_ALL,"Portuguese");
    int numero;
    leitura(&numero);
    printf("\nFatorial de %d é %d.", numero, fatorial(numero));

    return 0;
}
```

Função de Leitura

Função de Fatorial

Regra Geral

- Regra Geral: reduz a resolução do problema através da chamada recursiva de casos menores, que são resolvidos pela chamada de casos ainda menores da própria função, e assim seguindo até atingir o Ponto de Parada.

$$5! = 5 * 4 * 3 * 2 * 1$$

- Analisando, seria o mesmo que
- 5!
- 5 * 4!
- 5 * 4 * 3!
- 5 * 4 * 3 * 2!
- 5 * 4 * 3 * 2 * 1.

Então, a Regra Geral seria:

N! = N * (N-1)!, para N >0.

Ponto de Parada

- Ponto de Parada: ponto onde a função será encerrada, e normalmente é o limite inferior ou o limite superior da Regra Geral.

- Analisando, todos os fatoriais encerram em 1.
  - 5! = 5 * 4 * 3 * 2 * 1
  - 4! = 4 * 3 * 2 * 1
  - 3! = 3 * 2 * 1

Então, o Ponto de Parada seria:

N = 1, para N <= 1.

Funções Recursivas (Fatorial)

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

void leitura (int *numero) {
        printf("Digite um valor: ");
        scanf("%d", numero);
        if (numero < 0) {
            printf("Valor Inválido! Digite um valor positivo.\n\n");
            leitura(&numero);
        }
}

int fatorial (int numero) {
    if (numero <= 1) {
        return 1;
    }
    return numero * fatorial (numero - 1);
}

int main() {
    setlocale(LC_ALL,"Portuguese");
    int numero;
    leitura(&numero);
    printf("\nFatorial de %d é %d.", numero, fatorial(numero));

    return 0;
}
```

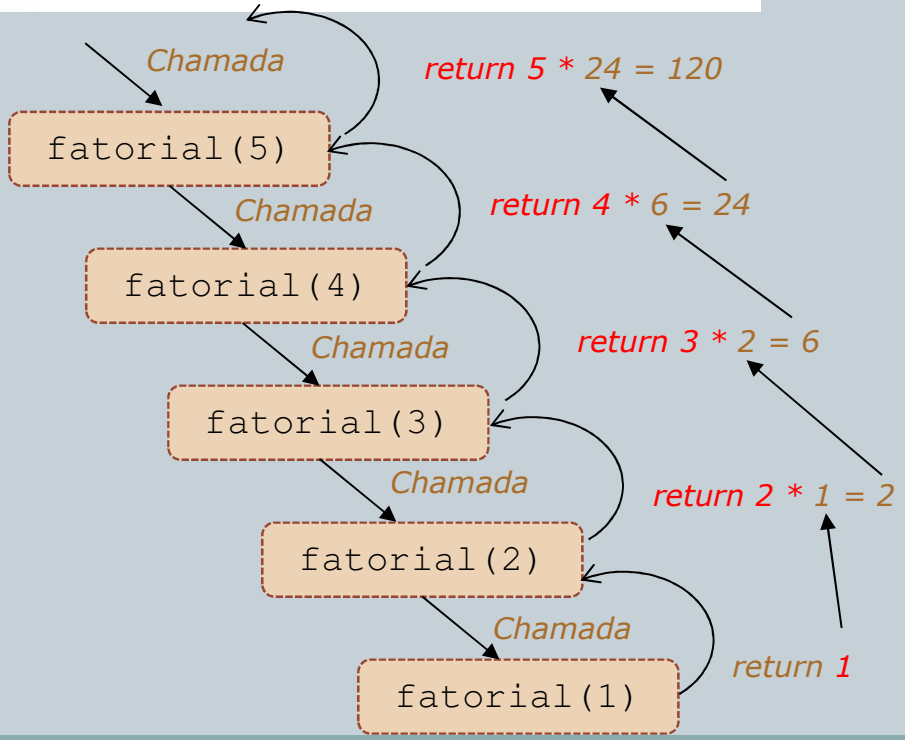
Função Recursiva de Leitura

Função Recursiva de Fatorial

Teste de Mesa

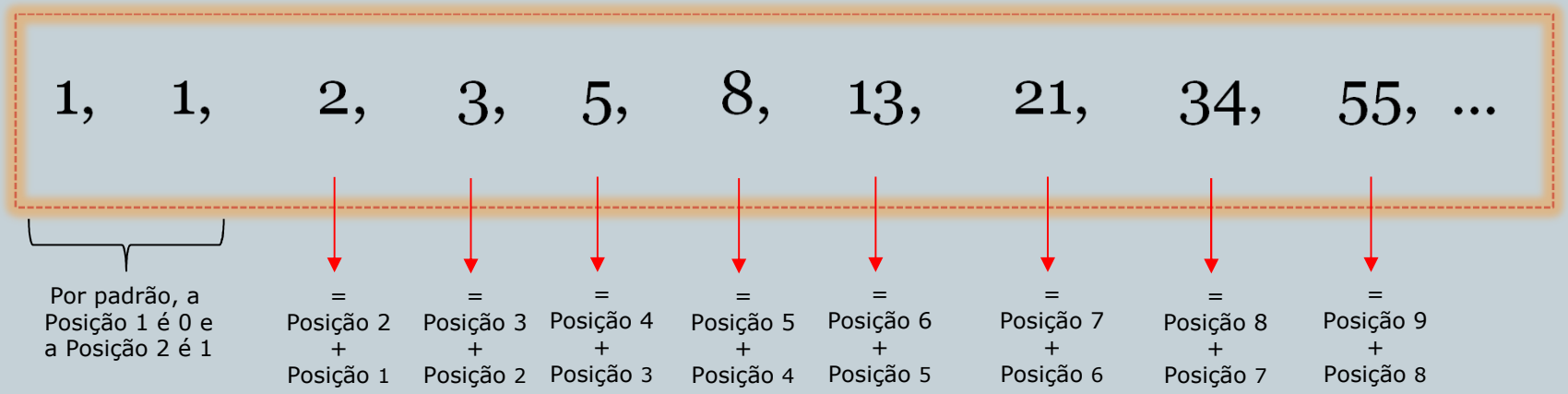
```
int fatorial (int num) {
    if (num <= 1) {
        return 1;
    }
    return (num * fatorial(num-1));
}
```

- Considerando que o usuário digitou num = 5



Construindo uma Função Recursiva

- Um dos exemplos mais utilizados para explicação de Recursividade é a Série de Fibonacci. Considerando que se quer saber o elemento de uma posição N na série.



- Por exemplo, se o usuário digitar que quer saber o elemento da posição 6, a resposta deverá ser 8.

Funções não Recursivas (Fibonacci)

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

void leitura (int *posicao) {
    do {
        printf("Digite uma posição: ");
        scanf("%d", posicao);
        if (posicao <= 0) {
            printf("Posição Inválida! Digite um valor maior que 0.\n\n");
        }
    } while (posicao <= 0);
}

void fibonacci (int posicao) {
    int ant = 1, pen = 1, prox, cont;
    for (cont = 1; cont <= posicao; cont++) {
        printf("%d\t",ant);
        prox = ant + pen;
        ant = pen;
        pen = prox;
    }
}

int main() {
    setlocale(LC_ALL,"Portuguese");
    int posicao;
    leitura(&posicao);
    fibonacci(posicao);
    return 0;
}
```

Função de Leitura

Função de Fibonacci

Regra Geral

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

- Analisando, pode-se verificar que o próximo elemento é gerado somando-se a posição anterior com a antepenúltima posição. Por exemplo, a Posição 4 é adquirida somando-se a Posição 3 e a Posição 2.

Então, a Regra Geral seria:

Fibo(pos) = 1, se posição <= 2
Fibo(pos) = Fibo(pos-1) + Fibo(pos-2), se posição > 2

Ponto de Parada

- Analisando, a série para a esquerda depende do valor digitado pelo usuário, mas o início é sempre o mesmo (1). Como o usuário entra com a posição final, o Ponto de Parada será as duas primeiras posições iniciais, pois ambas retornam 1.

Então, o Ponto de Parada seria:

Fibo(pos) = 1, para N <= 2.

Funções Recursivas (Fibonacci)

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

void leitura (int *posicao) {
        printf("Digite uma posição: ");
        scanf("%d", posicao);
        if (posicao <= 0) {
            printf("Posição Inválida! Digite um valor maior que 0.\n\n");
            leitura(&posicao);
        }
}

int fibonacci (int posicao) {
    if (posicao <= 2) {
        return 1;
    }
    return fibonacci(posicao - 1) + fibonacci(posicao - 2);
}

int main() {
    setlocale(LC_ALL,"Portuguese");
    int posicao;
    leitura(&posicao);
    printf("\nO elemento na posição %d é %d.", posicao, fibonacci(posicao));

    return 0;
}
```

Função Recursiva de Leitura

Função Recursiva de Fibonacci

Teste de Mesa