

# Estrutura de Dados II

## Procedimentos e Funções: RECURSIVIDADE

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

# Bibliografia



**Título:** Como Programa C – Sexta Edição

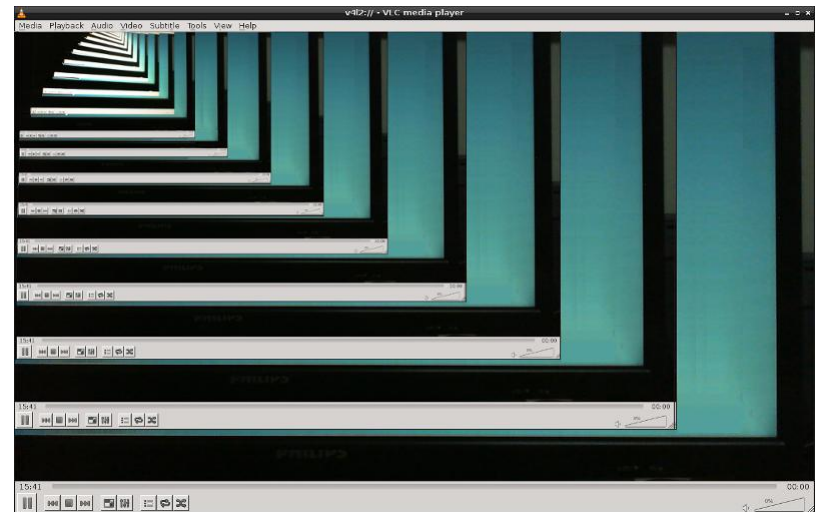
**Editora:** Pearson

**Autor:** Paul Deitel

**Capítulo 5.** Página 218

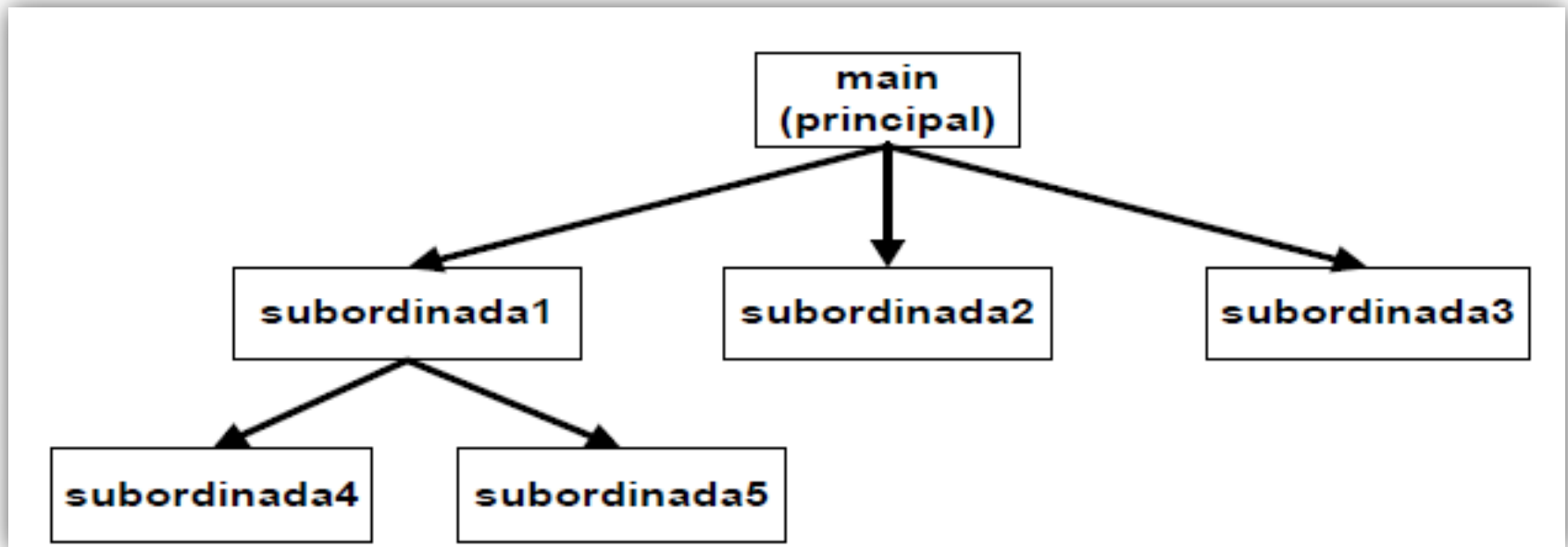
# O QUE É RECURSIVIDADE?

Qualidade que se podem **reaplicar sucessivamente** às estruturas resultantes de sua **aplicação anterior**, explicando assim o conceito execução infinita.



# RECURSIVIDADE - FUNÇÕES

Geralmente, os programas que analisamos são estruturados como funções que fazem chamadas entre si de uma maneira disciplinada e hierárquica.



# O QUE É RECURSIVIDADE?

Uma *função recursiva* é uma função que **chama a si mesma**, ou diretamente ou indiretamente por meio de outra função.

# O QUE É RECURSIVIDADE?

- Implementações **iterativas** tendem a ser mais **eficientes** (performance) que as recursivas.
- O código de uma função recursiva é mais simples e claro (?).
- Para alguns tipos de problemas, é útil ter funções que chamem a si mesmas.
  - **Cálculo Repetitivos com a mesma fórmula:**
    - Fatorial.
    - Verificar os valores de um determinado intervalo.
    - Fórmulas com decremento.

# RECURSIVIDADE

Os métodos recursivos para solução de problemas possuem vários **elementos em comum**.

Uma função só sabe resolver os casos simples.

*Se a função for chamada em um problema mais complexo, ela divide o **problema em duas partes**:*

- a) Uma parte que a função sabe como resolver
- b) e outra que ela não sabe (**chamada recursiva**).

# ELEMENTOS DA RECURSIVIDADE

**Ponto de Parada ou Condição de Parada:** é o ponto onde a função será encerrada.

**Regra Geral:** é o método que reduz a resolução do problema através da invocação recursiva de casos menores, que por sua vez são resolvidos pela resolução de casos ainda menores pela própria função, assim sucessivamente até atingir o “**ponto de parada**” que finaliza a função.



# RECURSIVIDADE

Para tornar viável a recursão, a segunda parte deve ser parecida com o problema original, mas ser uma versão um pouco mais simples ou menor do que ele.

Por esse novo problema ser parecido com o problema original, a função chama (lança) uma nova cópia de si mesma para lidar com o problema menor — o que é conhecido por ***chamada recursiva*** ou ***etapa de recursão***.

A etapa de recursão também inclui a palavra-chave **return** porque seu resultado **será combinado com a parte do problema que a função** sabe como resolver para formar um resultado que será enviado de volta para a função original de chamada, possivelmente **main**.

# RECURSIVIDADE

Uma função pode chamar a si própria por **UM NÚMERO LIMITADO** de vezes.

Esse limite é dado pelo **tamanho da pilha**. Se o valor correspondente ao tamanho máximo da pilha for atingido, haverá um estouro da pilha ou **Stack Overflow**.

Cada vez que uma função é chamada de forma recursiva, são armazenados uma cópia dos seus parâmetros, de modo a não perder os valores dos parâmetros das chamadas anteriores.

Ao final da execução, os dados são desempilhados e a execução volta ao subprograma que chamou a função

# QUANDO USAR?

**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  $(n-1)$ .**



# INTERATIVO - EXEMPLO

//Calculo de Fatorial de 5 - Convencional

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

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

```
main()
```

```
{    int i = 5;
```

```
    int fatorial = i;
```

```
    while (i > 1)
```

```
    {
```

```
        fatorial = fatorial * (i - 1);
```

```
        i--;
```

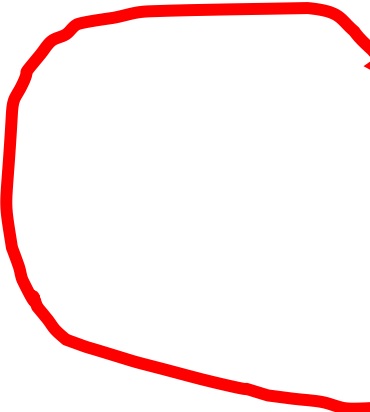
```
    }
```

```
    printf("Fatorial de 5 eh = %d ", fatorial); }
```

# INTERATIVO - EXEMPLO

## Execução While:

Início	1º Execução (i > 1) ?	2º Execução (i > 1) ?	3º Execução (i > 1) ?	4º Execução (i > 1) ?
i = 5 Fatorial = 5	i = 4 Fatorial = 20	i = 3 Fatorial = 60	i = 2 Fatorial = 120	i = 1 Fatorial = 120



```
while (i > 1)
{
    fatorial = fatorial * (i-1);
    i--;
}
```

# RECURSIVIDADE - EXEMPLO

```
//Calculo de Fatorial de 5
#include <stdio.h>
#include <stdlib.h>

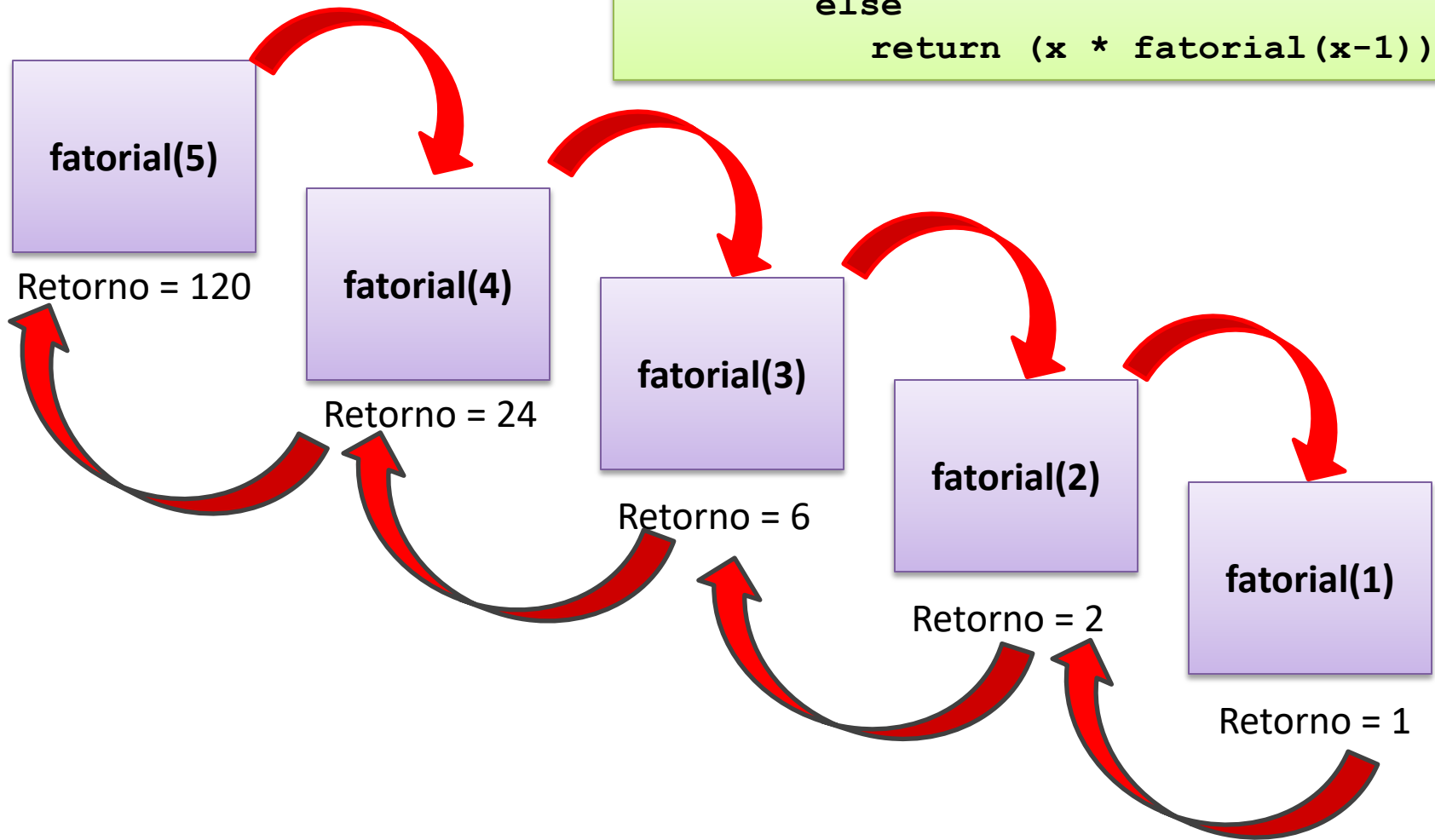
int fatorial (int x)
{
    if (x ==1) //Ponto de Parada
        return (1);
    else
        return (x * fatorial(x-1));
}

main()
{
    printf("Fatorial de 5 eh = %d ", fatorial(5));
}
```

# RECURSIVIDADE - EXEMPLO

Execução Recursiva:

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



# RECURSIVIDADE - EXEMPLO

Função para verificar os valores pares entre 100 e 150

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

bool verificar (int x){
    if (x % 2 != 0)
        return (false);
    else
        return (true);}

main() {
    int x = 50;
    int y = 150;
    while (x <= y)
    {
        if (verificar(x))
            printf("\n%d Eh par", x);
        x++;
    }
}
```

**ATENÇÃO:**  
Executa de 100  
até 150 com laço  
de repetição.



# RECURSIVIDADE - EXEMPLO

Função recursiva para verificar os valores pares entre 100 e 150.

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

void imprime_pares(int x, int y)
{
    if (x >= y+1)
        return;
    else if (x % 2 == 0)
        printf ("Valor %d eh par \n", x);

    imprime_pares(x+1, y);
}

main()
{
    imprime_pares(50, 150);
}
```

**ATENÇÃO:**  
Executa de 100  
até 150 sem laço  
de repetição.

# Recursividade 01

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

int ff (int n)
{
    if (n <= 1) return 1;
    else return ff(n-1);
}

main ()
{
    printf ("%d", ff(7));
}
```

**Usando teste de mesa**, qual será o valor impresso no final do programa?

# Recursividade 02

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

int ff (int n)
{
    if (n <= 1) return n;
    else return ff(n-1);
}

main ()
{
    printf ("%d", ff(7));
}
```

**Usando teste de mesa,** qual será o valor impresso no final do programa?

# Recursividade 03

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

int ff (int n)
{
    if (n <= 1) return n;
    else return ff(n-1) + 1;
}

main ()
{
    printf ("%d", ff(7));
}
```

**Usando teste de mesa,** qual será o valor impresso no final do programa?

# Recursividade 04



ERRO

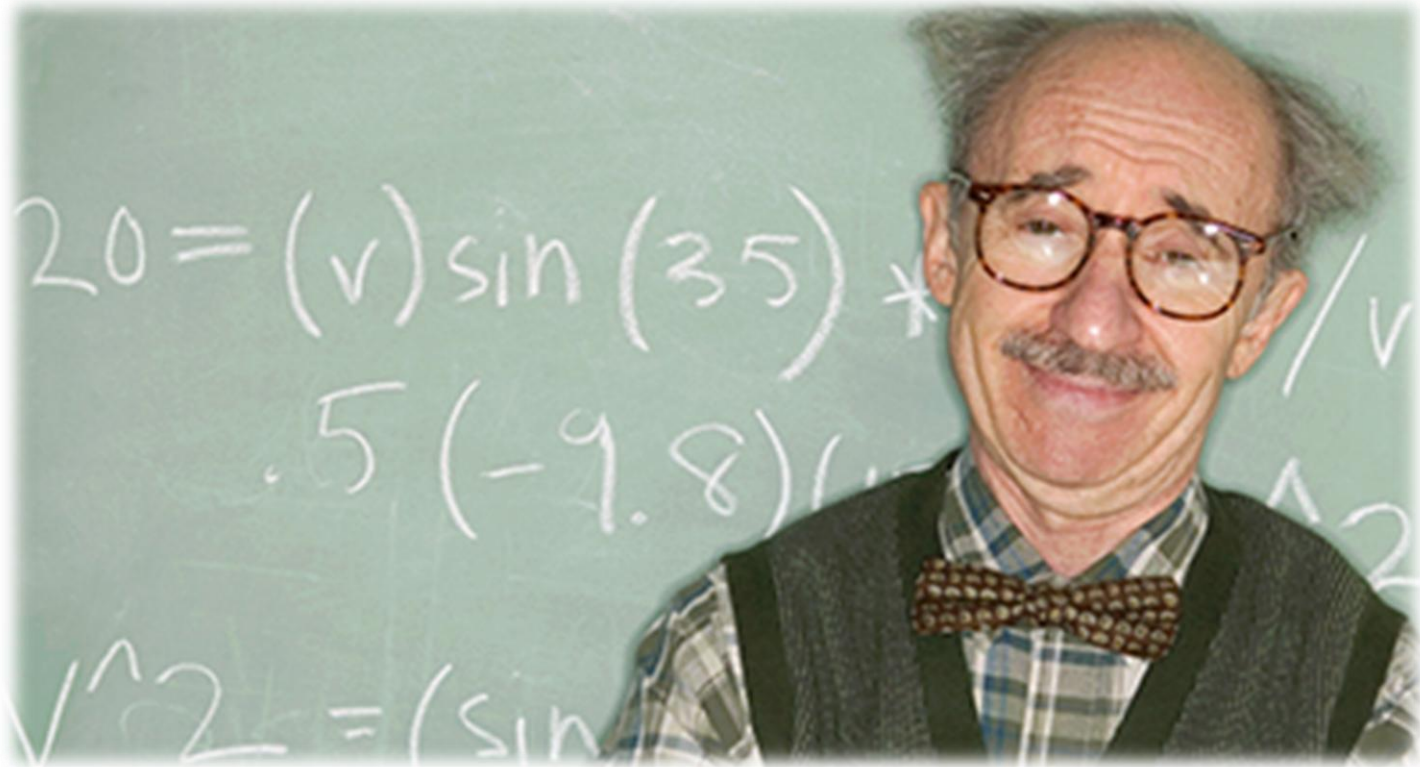
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int ff (int n)
{
    return ff(n-1) + 1;
    if (n <= 1) return n;
}

main ()
{
    printf ("%d", ff(7));
}
```

**Usando teste de mesa,** qual será o valor impresso no final do programa?

# Exercícios



# Atividade 01

Fazer uma função recursiva que receba um valor como parâmetro e mostre o somatório do valor.

**Exemplo:** Valor = 5

Calcular  $5+4+3+2+1$

# Atividade 02

Faça uma função que receba, por parâmetro, um valor inteiro e positivo e retorne o **número de divisores** do valor lido.

- a) Implementar usando laço de repetição (**while ou for**) e outra usando **recursividade**.
- b) Comparar o tempo gasto na execução de cada implementação.



# Atividade 03

Faça um algoritmo recursivo que dado um valor decimal, imprime seu correspondente em binário.

Implementar usando laço de repetição (while ou for) e outra usando recursividade.

# Atividade 04

Escreva uma função recursiva que determine quantas vezes um CARATERE X ocorre em um STRING Y.

**Por exemplo**, o caractere A ocorre 3 vezes em "abacate".

Implementar usando laço de repetição (while ou for) e outra usando recursividade.

# Atividade 05 (A)

Qual será a saída gerada das funções dos algoritmos abaixo, considerando o valor inicial de  $n=4$ :

```
a) func (int n)
{
    if (n == 0)
        printf("fim");
    else
    {
        printf(n);
        func(n-1);
    }
}
```

# Atividade 05 (B)

Qual será a saída gerada das funções dos algoritmos abaixo, considerando o valor inicial de  $n=4$ :

```
b) func (int n)
{
    if (n == 0)
        printf("fim");
    else
    {
        func(n-1);
        printf(n);
    }
}
```

# Atividade 05 (C)

Qual será a saída gerada das funções dos algoritmos abaixo, considerando o valor inicial de  $n=4$ :

```
c) func (int n)
{
    if (n == 0)
        printf("fim");
    else
    {
        printf(n);
        func(n-1);
        printf(n);
    }
}
```

# Atividade 05 (D)

Qual será a saída gerada das funções dos algoritmos abaixo, considerando o valor inicial de  $n=4$ :

```
d) func (int n)
{
    if (n == 0)
        printf("fim");
    else
    {
        func(n-1);
        printf(n);
        func(n-1);
    }
}
```