FAMEPI

Faculdade do Amazonas de Ensino, Pesquisa e Inovação

FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

Prof Pedro Corrêa

# Objetivo da aula

**Utilizar a técnica de recursividade em programação na Linguagem C**

# Sumário

# Sumário

# Algoritmos e Estrutura de Dados 2

## GENERALIDADES

- Devido à característica operacional e de retorno de conteúdo de uma função:

  - É possível desenvolver funções que fazem chamadas a si mesmas;

- Esse efeito denomina-se **RECURSIVIDADE** ou **RECURSÃO**

# Sumário

# Sumário

# Algoritmos e Estrutura de Dados 2

## DEFINIÇÃO

- A recursão é uma técnica que define um problema em termos de uma ou mais versões menores deste mesmo problema

- Esta ferramenta pode ser utilizada sempre que for possível expressar a solução de um problema em função do próprio problema

# Algoritmos e Estrutura de Dados 2

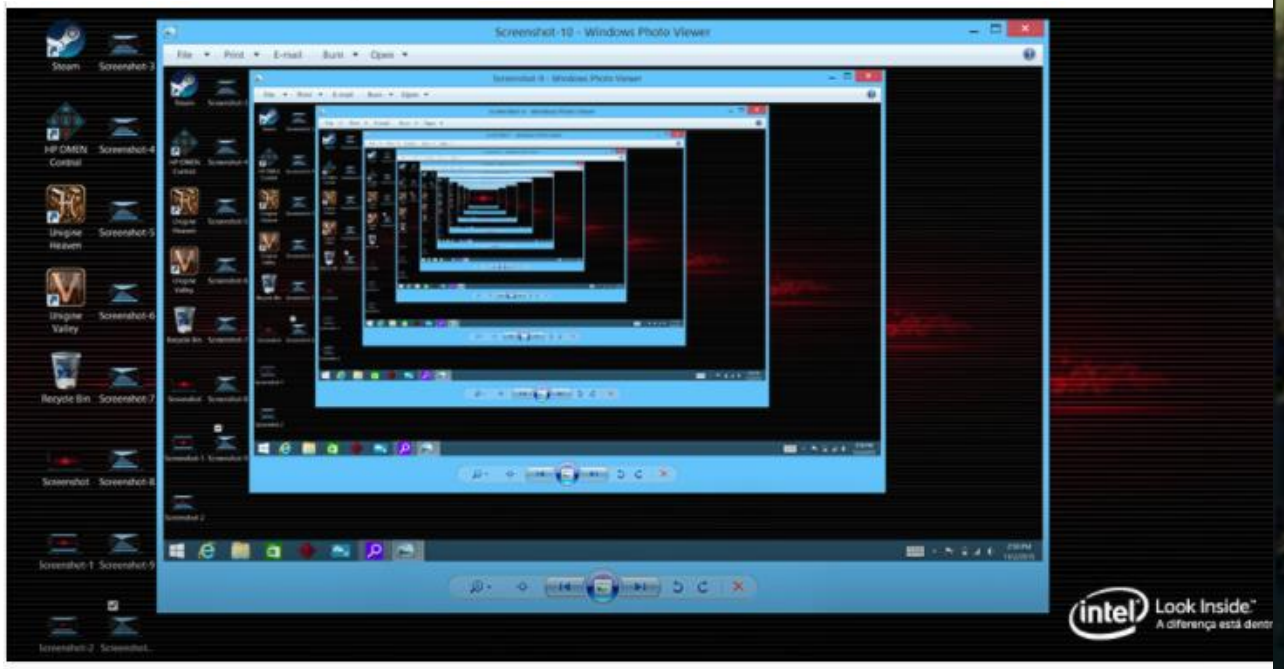
## DEFINIÇÃO

- As funções recursivas são em sua maioria soluções mais elegantes e simples, se comparadas a funções tradicionais ou iterativas, já que executam tarefas repetitivas sem utilizar nenhuma estrutura de repetição, como for, while ou do-while.

- Porém essa elegância e simplicidade têm um preço que requer muita atenção em sua implementação.

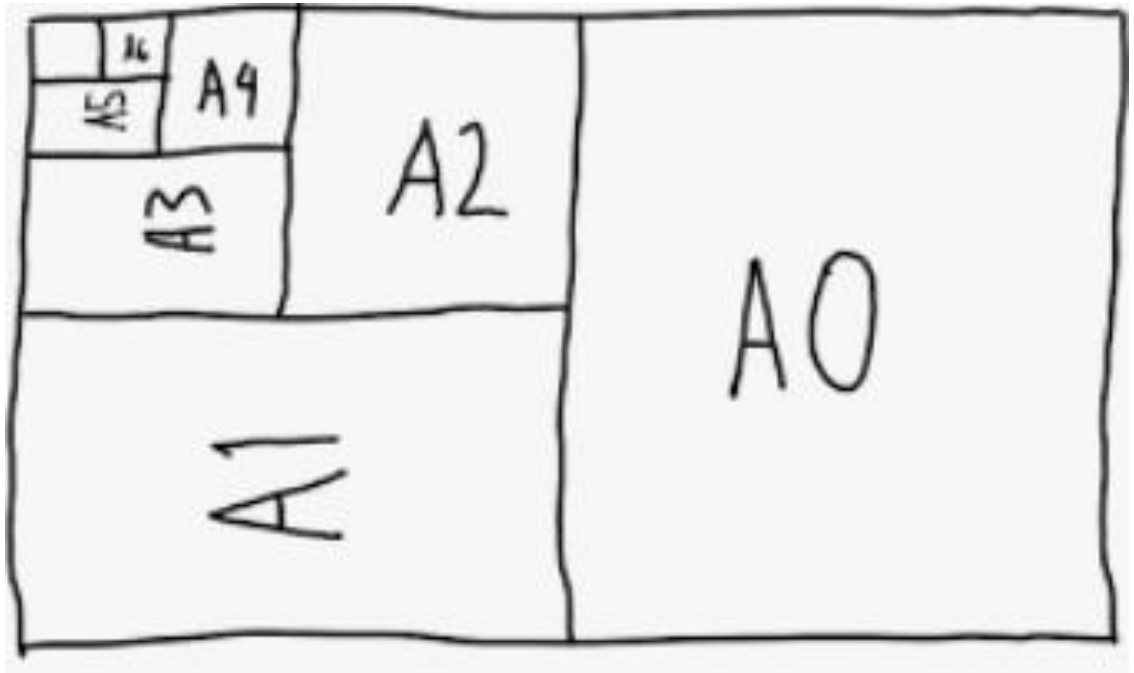
# Algoritmos e Estrutura de Dados 2

## EXEMPLOS



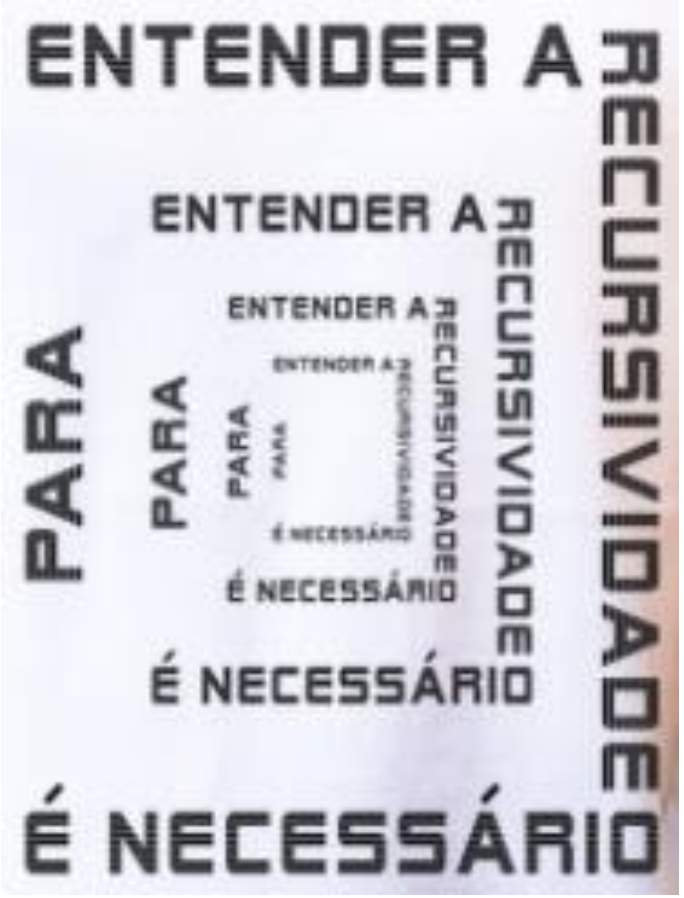

# Algoritmos e Estrutura de Dados 2

## EXEMPLOS




# Algoritmos e Estrutura de Dados 2

## EXEMPLOS


"Para saber recursão, tem que saber recursão"

FAMEPI
Faculdade do Amazonas de Ensino, Pesquisa e Inovação
FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

## EXEMPLOS

# Sumário

# Sumário

# Algoritmos e Estrutura de Dados 2

## Escrever uma função que devolve o fatorial de um nr

**n! = n * (n-1) * (n-2) * (n-3) *....* 1**

**exemplo:**

**4! = 4 * 3 * 2 * 1 = 24**

**Implementação iterativa**

```
#include<stdio.h>
int fat(int n){
     int f, i;
     if (n == 0)
               return 1;
      else {
               f = 1;
               for ( i = n ; i > 1 ; i-- )
                     f = f * i;   // f *= i;
               return f;
        }
}
int main( ){
    int nr;
    printf("\n\n Entre com um número: ");
    scanf("%d", &nr);
    if (num >= 0)
       printf("\n\n Fat do nr %d =  %d" , nr, fat(nr) );

return 0;
}
```

# Algoritmos e Estrutura de Dados 2

## Recursividade

- É uma estratégia que pode ser utilizada sempre que o cálculo de uma função para um valor **n** qualquer, pode ser descrita a partir do cálculo desta mesma função para o termo anterior (n-1).

**Exemplo – Função fatorial:**

**n! = n * (n-1) * (n-2) * (n-3) *....* 1**
**(n-1)! = (n-1) * (n-2) * (n-3) *....* 1**

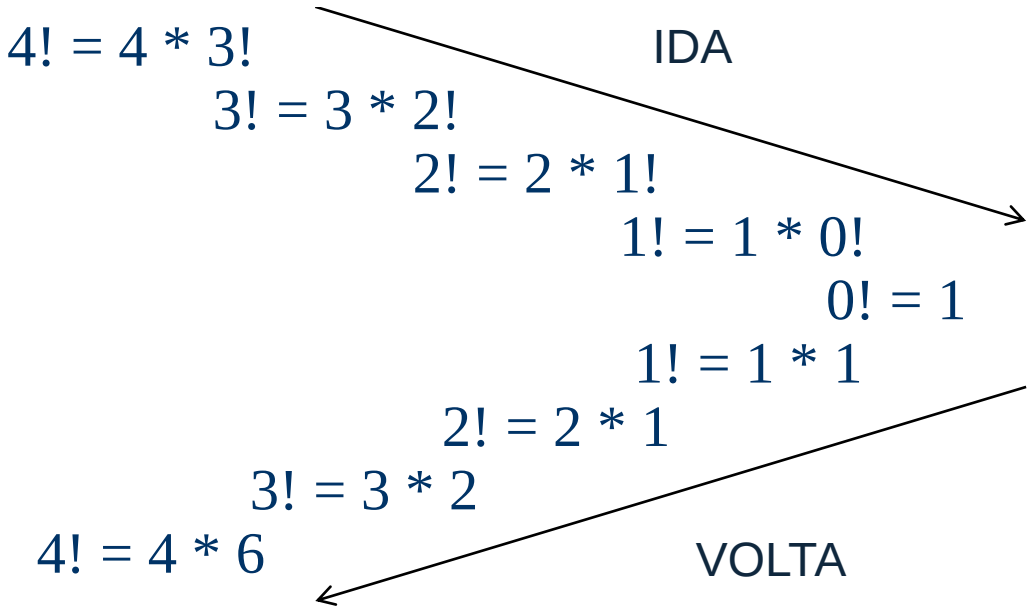
**logo:**

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

# Algoritmos e Estrutura de Dados 2

## Exemplo – Função Fatorial

$$fat(n) = \begin{cases} 1, \text{ se } n = 0 \text{ (solução trivial)} \\ n \times fat(n-1), \text{ se } n > 0 \text{ (solução recursiva)} \end{cases}$$



# Algoritmos e Estrutura de Dados 2

## Código – Função Fatorial

```
#include<stdio.h>

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

int main( ){
   int nr;
   printf("\n\n Entre com um nr: ");
   scanf("%d", &nr);
   if (num >= 0)
      printf("\n\n O fatorial do nr %d =  %d" , nr, fat(nr) );

return 0;
}
```

**Implementação recursiva**

Condição de parada

Chamada a si mesma

Chamada da Função

FAMEPI
Faculdade do Amazonas de Ensino, Pesquisa e Inovação
FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

## Recursividade

- Uma função recursiva é aquela que faz uma chamada a si mesma direta (a função A chama a própria função A) ou indireta (a função A chama uma função B que, por sua vez, chama A)

- Quando uma função recursiva está sendo executada, são alocados novos parâmetros e variáveis locais na memória, para cada instância da função em questão.

- Isto faz as funções recursivas serem mais lentas do que as funções iterativas

# Algoritmos e Estrutura de Dados 2

## Recursividade

- Há três regras para a Recursão

1. Saber quando parar
2. Decidir como aplicar o próximo passo
3. Analisar o problema de forma que o mesmo possa ser dividido em problemas menores

# Algoritmos e Estrutura de Dados 2

## Recursividade

- **1ª Regra: Saber quando parar**
  - Qualquer função recursiva deve fazer uma checagem para verificar se a jornada já foi completada antes da nova chamada recursiva

- **2ª Regra: Decidir como fazer o próximo passo**
  - Quebrar um problema em subproblemas que possam ser resolvidos instantaneamente.

# Algoritmos e Estrutura de Dados 2

## Recursividade

- **3ª Regra: analisar o problema de forma que o mesmo possa ser dividido em problemas menores**
  - Achar uma maneira da função chamar a si mesma recursivamente e cujo parâmetro é um problema menor resultante da segunda regra

# Algoritmos e Estrutura de Dados 2

## Exemplo – Somatório

- Escrever uma função que recebe como parâmetro um inteiro positivo n e retorna a soma de todos os números inteiros entre 0 e n:

$$5 = 5 + 4 + 3 + 2 + 1 = \mathbf{15}$$

$$\sum n = \begin{cases} 1, & \text{se } n = 1 \\ n + (n-1), & \text{se } n > 1 \end{cases}$$

FAMEPI
Faculdade do Amazonas de Ensino, Pesquisa e Inovação
FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

## Exemplo – Somatório

$$\sum n = \begin{cases} 1, & \text{se } n = 1 \\ n + (n-1), & \text{se } n > 1 \end{cases}$$

IDA

$5 = 5 + (5-1)$

$(5-1) = 4 + (4-1)$

$(4-1) = 3 + (3-1)$

$(3-1) = 2 + (2-1)$

$(2-1) = 1$

$(3-1) = 2 + 1$

$(4-1) = 3 + 3$

$(5-1) = 4 + 6$

$5 = 5 + 10$

VOLTA

# Algoritmos e Estrutura de Dados 2

## Código – Função Somatório

```
#include<stdio.h>

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

int main( ){
   int nr;
   printf("\n\n Entre com um número: ");
   scanf("%d", &nr);
   if (nr > 0)
      printf("\n\n O somatorio do número %d:  %d" , nr, somat(nr) );

return 0;
}
```

Condição de parada

Chamada a si mesma

Chamada da Função

FAMEPI
Faculdade do Amazonas de Ensino, Pesquisa e Inovação
FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

## Código – Função série de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...)

**Implementação iterativa**

```
#include<stdio.h>

int fib(int n){
      int f1 = 0, f2 = 1, f3, i;
      for( i  = 1 ; i <=  n ; i++ ) {
            f3 = f2 + f1;
            f1 = f2 ;   f2 = f3 ;
       }
       return f1;
}

int main( ){
    int nr;
    printf("\n\n Entre com um número: ");
    scanf("%d", &nr);
    if (nr > 0)
        printf("\n\n %do. nr da serie de Fibonacci =  %d" , nr, fib(nr) );

return 0;
}
```

FAMEPI
Faculdade do Amazonas de Ensino, Pesquisa e Inovação
FUNDAÇÃO MATIAS MACHLINE

# Algoritmos e Estrutura de Dados 2

## Código – Função série de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...)

**Implementação recursiva**

```
#include<stdio.h>

int fib(int n){

      if(n <= 1) return n;

       return fib(n-1) + fib(n-2);
}

int main( ){
    int nr;
   printf("\n\n Entre com um nr: ");
   scanf("%d", &nr);
   if (nr > 0)
      printf("\n\n %do. nr da serie de Fibonacci =  %d" , nr, fib(nr) );

return 0;
}
```

Condição de parada

Chamada a si mesma

Chamada da Função

# Sumário

# Sumário

# Algoritmos e Estrutura de Dados 2

## Conclusão

- As funções recursivas são em sua maioria soluções mais elegantes e simples ➔ não precisam dos laços for, while ou do-while.

- Essa elegância e simplicidade têm preço:
  - ❖ São alocados novos parâmetros e variáveis locais na memória, para cada instância da função em questão.
  - ❖ São mais lentas do que as funções iterativas