UFABC

Recursão

Prof. Fabrício Olivetti de França

(com inspirações do slide do prof. Rodrigo Hausen)

Recursão

“Para entender recursão, você primeiro deve entender recursão!”

UFABC

Recursão

Forma de pensar em uma solução para um problema resolvendo primeiro instâncias menores (e talvez triviais) desse problema.

UFABC

Definição Recursiva

Na matemática muitas funções possuem definições **recursivas** ou **indutivas**.

Essas definições define valores bases para dada função e o restante dos valores são gerados utilizando a própria função.

UFABC

Fatorial

$$n! = \begin{cases} 1, & \text{se } n==0 \\ n \cdot (n-1)!, & \text{c. c.} \end{cases}$$

UFABC

Fatorial

Essa definição é indutiva pois para descobrir o valor de 5! você tem que induzir o valor 4!.

Para saber 4!, você terá que induzir o valor de 3!.

Para saber 3!, você terá que induzir o valor de 2!.

Para saber 2!, você terá que induzir o valor de 1!.

Para saber 1!, você terá que induzir o valor de 0!, que já está definido como 1.

UFABC

Fatorial

Então:

$1! = 1 \cdot 0! = 1 \cdot 1 = 1$

$2! = 2 \cdot 1! = 2 \cdot 1 = 2$

$3! = 3 \cdot 2! = 3 \cdot 2 = 6$

$4! = 4 \cdot 3! = 4 \cdot 6 = 24$

$5! = 5 \cdot 4! = 5 \cdot 24 = 120$

UFABC

Programando Funções Recursivas

Uma função recursiva deve ter:

- Pelo menos um caso base, para que ela possa terminar, e
- Pelo menos uma chamada recursiva.

UFABC

Programando Funções Recursivas

```
unsigned int fatorial (unsigned int n)
{
    if (n==0) return 1;
    return n * fatorial(n-1);
}
```

UFABC

Programando Funções Recursivas

```
unsigned int fatorial (unsigned int n)
{
    if (n==0) return 1;  /* Caso base */
    return n * fatorial(n-1);
}
```

UFABC

Programando Funções Recursivas

```
unsigned int fatorial (unsigned int n)
{
    if (n==0) return 1;  /* Caso base */
    return n * fatorial(n-1); /* Chamada recursiva */
}
```

UFABC

Fibonacci

Outro exemplo é a sequência infinita de Fibonacci, definida por:

$F(0) = F(1) = 1$

$F(n) = F(n-1) + F(n-2)$

UFABC

Fibonacci

Façam vocês!!

UFABC

O poder da recursão

"The power of recursion evidently lies in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions." - Niklaus Wirth

UFABC

Como a recursão funciona

```
… fatorial ( n )
    …
    return n * fatorial(n-1);
```

Nesse momento o resultado parcial e a instrução de operação é armazenado em uma pilha até que a chamada recursiva retorne.

UFABC

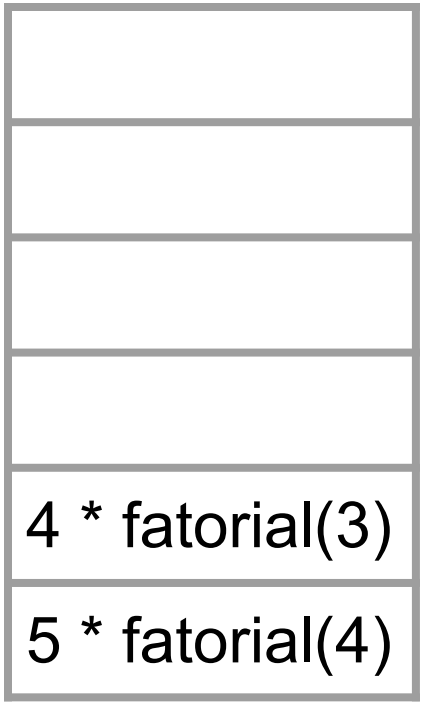
Estouro da pilha

> fatorial(5)

UFABC

Estouro da pilha

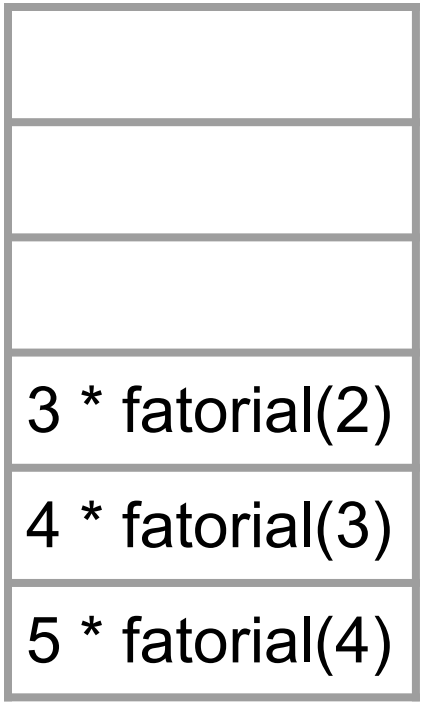
> fatorial(5)



UFABC

Estouro da pilha

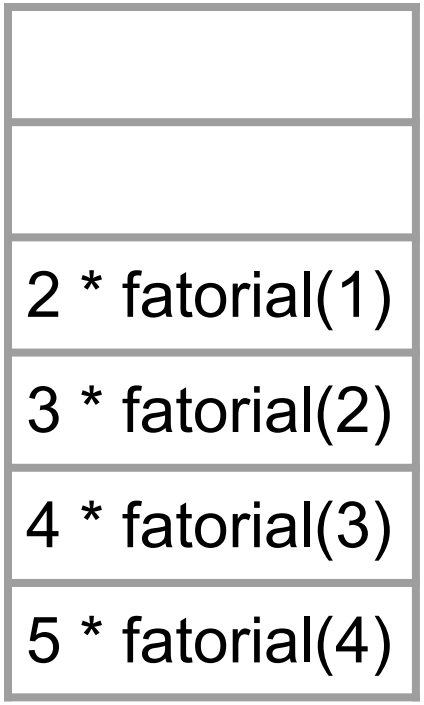
> fatorial(5)



UFABC

Estouro da pilha

> fatorial(5)



UFABC

Estouro da pilha

> fatorial(5)

UFABC

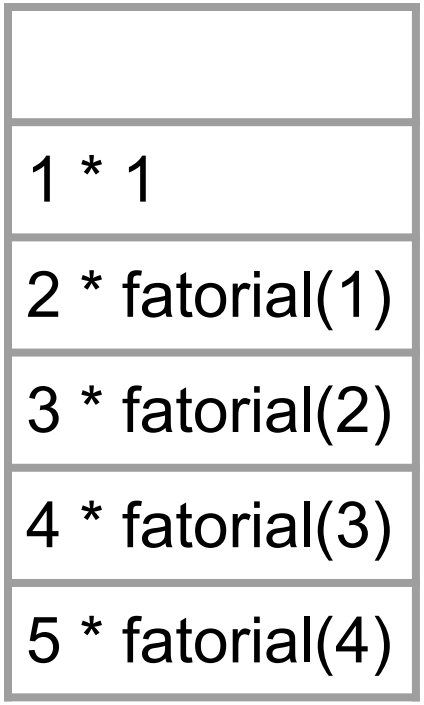
Estouro da pilha

> fatorial(5)

UFABC

Estouro da pilha

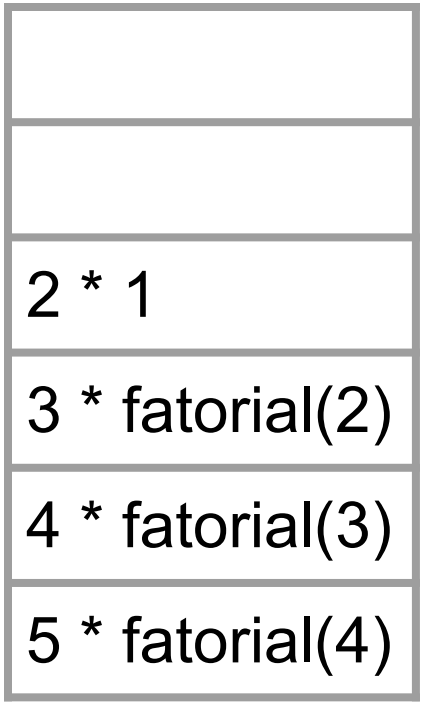
> fatorial(5)



UFABC

Estouro da pilha

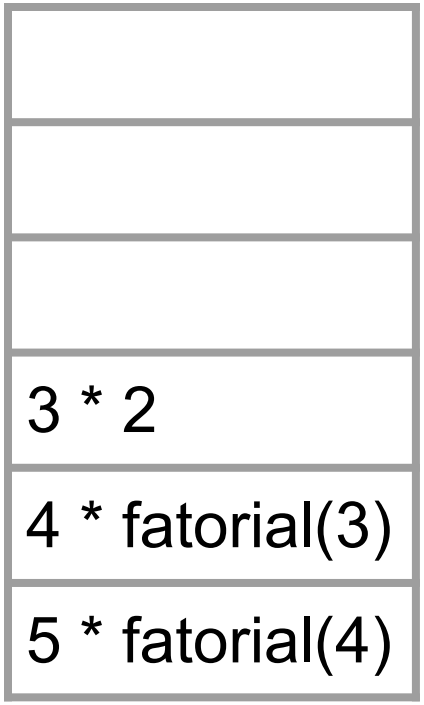
> fatorial(5)



UFABC

Estouro da pilha

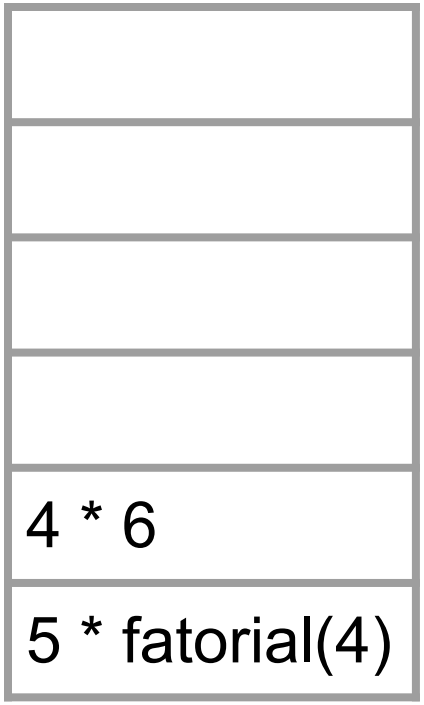
> fatorial(5)



UFABC

Estouro da pilha

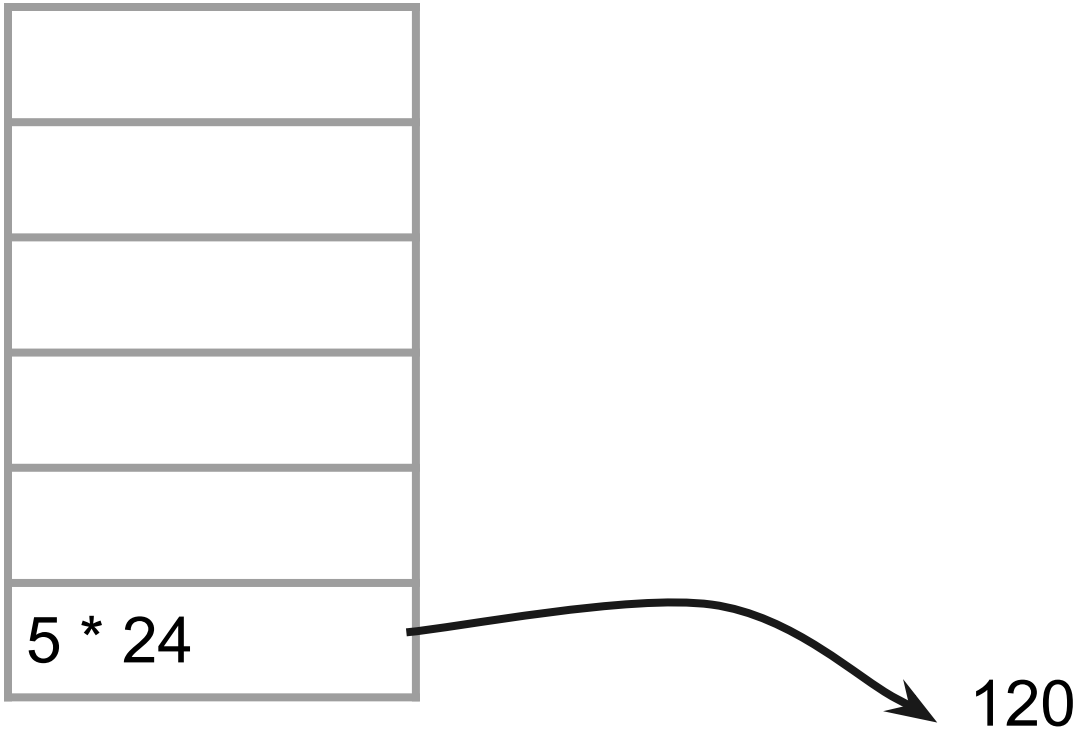
> fatorial(5)



UFABC

Estouro da pilha

> fatorial(5)



UFABC

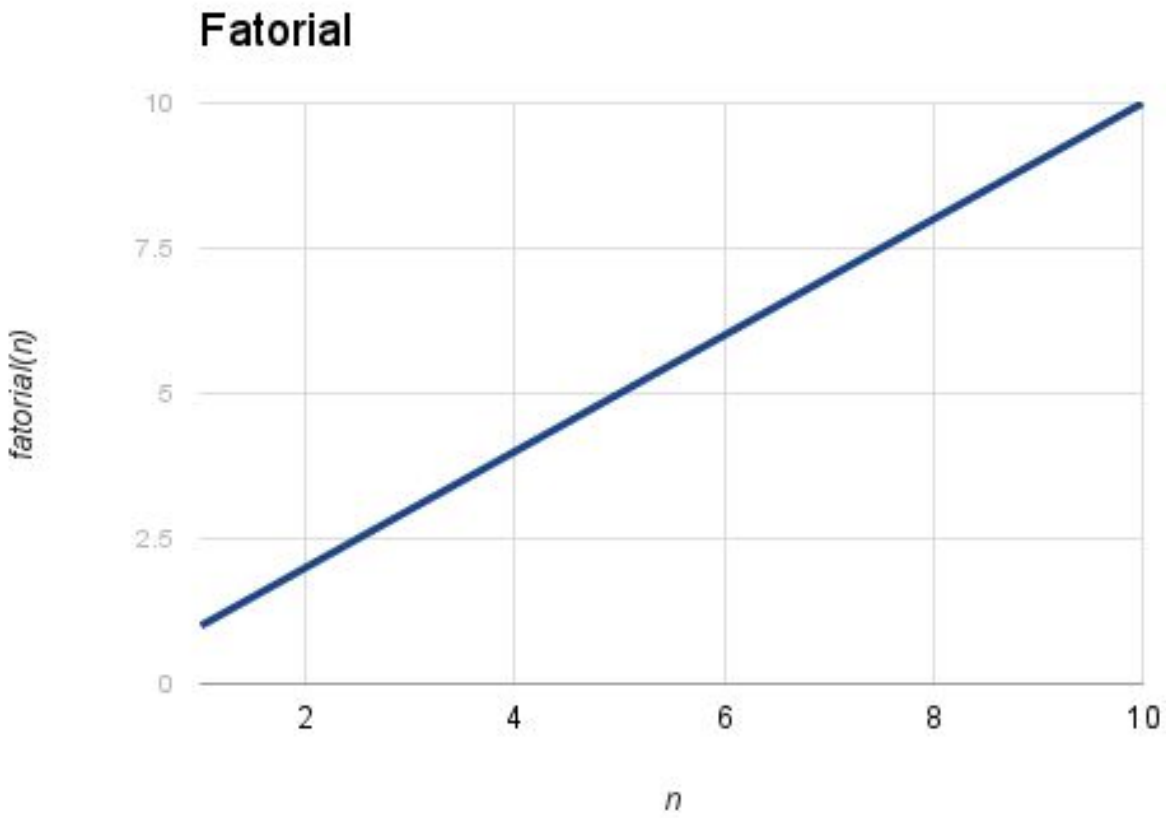
Estouro da pilha

Se a quantidade de chamadas recursivas forem maiores do que o tamanho da pilha, ocorre o erro conhecido como **estouro de pilha** ou **stack overflow**.

Esse é um outro caso que pode ocorrer **segmentation fault**.

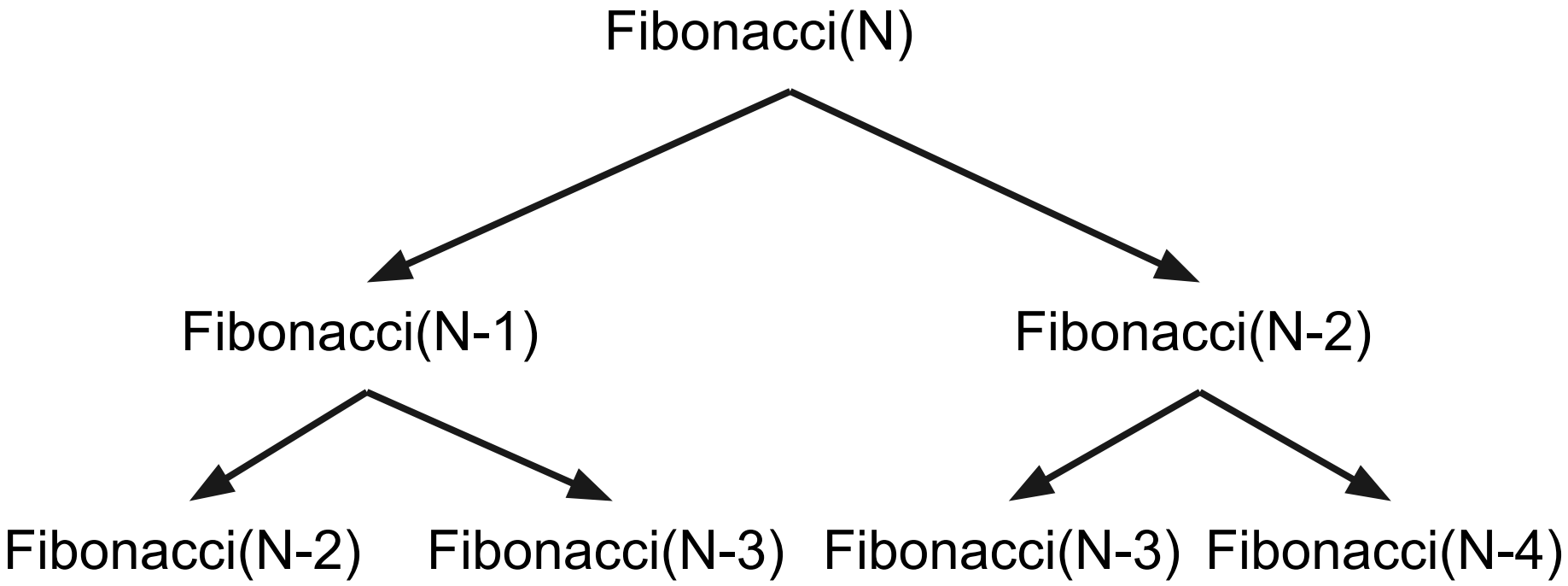
UFABC

Uso da Pilha x N - fatorial



FABC

Uso da pilha x N - Fibonacci



...

UFABC

Uso da pilha x N - Fibonacci

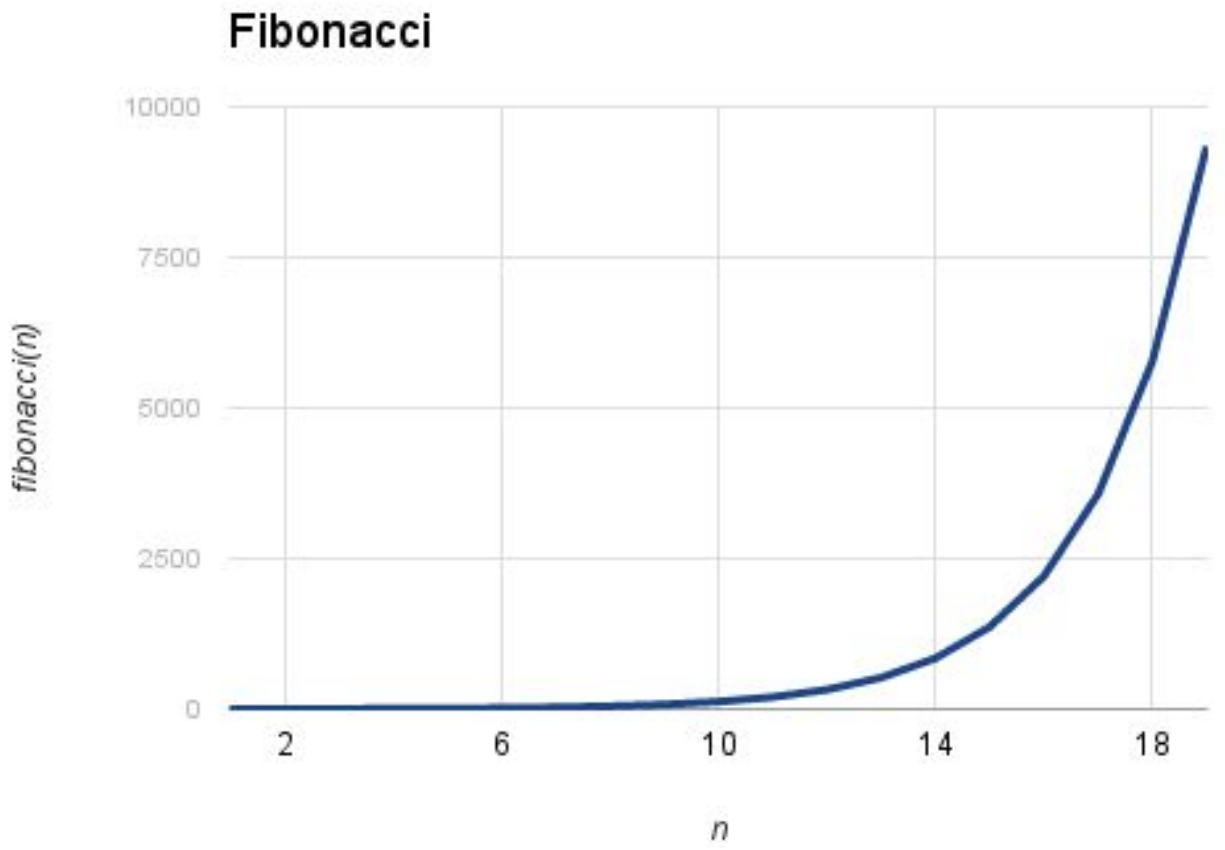
Com T(n) representando o número de chamadas recursivas do algoritmo de Fibonacci para a entrada n.

$T(0) = T(1) = 0$

$T(n) = 2 + T(n-1) + T(n-2) \sim \text{Fibonacci}(n) \sim \phi^n$

UFABC

Uso da pilha x N - Fibonacci



Quando usar recursão

- Quando a solução for representada mais naturalmente com recursão e,
- Quando não houver riscos de estouro de pilha ou,
- Como protótipo de uma solução iterativa.

UFABC

Quando não usar recursão

- Quando há riscos de estouro de pilha ou,
- A solução é tão simples quanto na forma iterativa

UFABC

Recursão Caudal

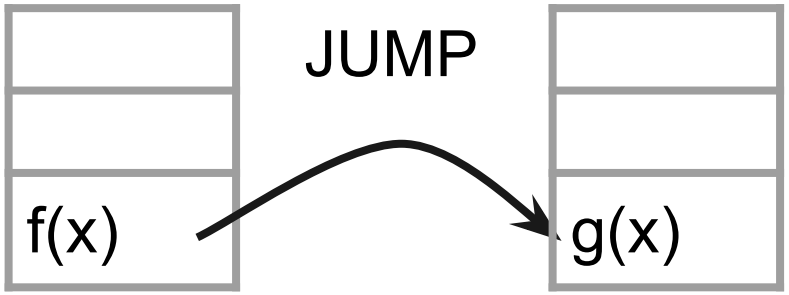
Quando a última instrução de uma rotina ou função é a chamada de uma função temos o que é chamado de **chamada caudal** ou **tail call**.

Esse tipo de chamada é especificamente útil para o compilador durante a otimização do código, pois não é necessário usar espaço da pilha.

UFABC

Recursão Caudal

```
int f( int x )
{
    return g(x);
}
```



UFABC

Recursão Caudal

Quando a última instrução de uma função é a chamada da própria função temos o que é chamado de **recursão caudal** ou **tail recursion**.

Nesse caso o compilador pode facilmente otimizar o código para um processo iterativo.

UFABC

Máximo Divisor Comum

O MDC(m, n) pode ser definido recursivamente como:

MDC(m, n) = MDC(n, m % n) se n != 0 senão m

UFABC

Máximo Divisor Comum

```
unsigned int mdc(unsigned int m, unsigned int n)
{
    if (n==0) return m;
    return mdc(n, m%n);
}
```

UFABC

Máximo Divisor Comum

Reparem que a última instrução do MDC é apenas a chamada da própria função. Logo ela é uma recursão caudal.

UFABC

Máximo Divisor Comum

O compilador transforma a função para algo como:

```
unsigned int mdc(unsigned int m, unsigned int n)
{
    inicio:
        if (n==0) return m;
        tmp = n;
        n = m % n;
        m = n;
        jump inicio
}
```

UFABC

Recursão não-caudal → Caudal

```
unsigned int fatorial (unsigned int n)
{
    if (n==0) return 1;
    return n * fatorial(n-1);
}
```

UFABC

Recursão não-caudal → Caudal

```
unsigned int fatorialTR (unsigned int n, unsigned base)
{
    if (n==0) return base;
    return fatorialTR(n-1, n*base);
}

unsigned int fatorial (unsigned int n, unsigned base)
{
    return fatorialTR(n, 1);
}
```

UFABC

Ex01 - Somatória

A somatória dos números inteiros de 1 até n pode ser definida como:

$S(n) = 1$, se $n==1$

$n + S(n-1)$, se $n>1$

Crie um algoritmo recursivo para calcular a somatória e uma versão caudal para esse algoritmo.

UFABC

Ex01 - Somatória

Desconsiderando o uso da pilha, quanto de memória esse algoritmo utilizada?

Quantas instruções são executadas? (conte chamadas de função como instrução)

UFABC