Funções Recursivas

Professor **Alexandre Moreno**

RECURSIVIDADE

Definições:

1. A recursividade é um mecanismo que permite uma função cahamar a si mesma, ***direta*** ou ***indiretamente***.

2. Uma função é dita recursiva quando possui uma chamada a si própria (ZIVIANI, 2004; GUIMARÃES; LAGES, 1994)

Três leis básicas da Recursividade

1. Deve possuir um fim previsto;
2. A cada execução, deve estar mais próximo do fim;
3. Deve ser recursiva;

Tipos de recursão:

- **Recursão direta**
  a função **A** chama a própria função **A**;
- **Recursão indireta**
  a função **A** chama uma função **B** que, por sua vez, chama a função **A**

Q+A WITH AMY

Linguagens que permitem uma função chamar a si própria são ditas **recursivas**. Linguagens que não permitem são ditas **iterativas** ou **não recursivas.**

FATORIAL (relembrando)

5! = 5 * 4 * 3 * 2 * 1 = **120**

ou

5! = 1 * 2 * 3 * 4 * 5 = **120**

Fatorial

Note que há um **laço de repetição** que vai de **1 até n**, multiplicando o resultado anterior com o contador.

5! = 1 * 2 * 3 * 4 * 5 = **120**

Fatorial

Versão Iterativa

Laço de repetição que vai de **1** até **n**, multiplicando o resultado anterior com o contador.

```
funcao fatorial(n: inteiro): inteiro
var i, aux: inteiro
inicio
    aux ← 1
    para i ← 1 ate n faca
        aux ← aux * i
    fimpara
    RETORNE aux
fimfuncao
```

Fatorial

Versão Iterativa

Laço de repetição que vai de **1** até **n**, multiplicando o resultado anterior com o contador.

Versão em **linguagem C**

```c
int fatorial(int n){
   int i,aux;
   aux=1;
   for (i=1; i<=n; i++)
      aux=aux*i;
   return aux;
}
```

Fatorial

Pensando em recursão

Note que **n!** é **n*(n-1)!**

O processo será encerrado quando ocorrer **0!**

5! = 5*4!

4! = 4*3!

3! = 3*2!

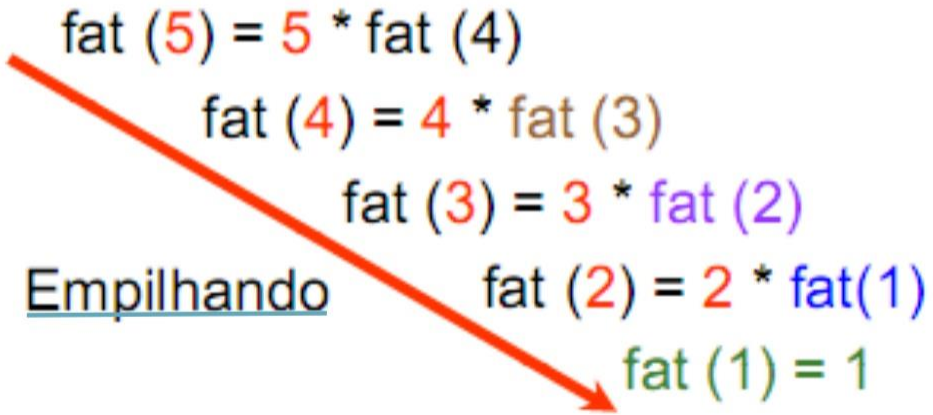
2! = 2*1!

1! = 1*0!

0! = 1

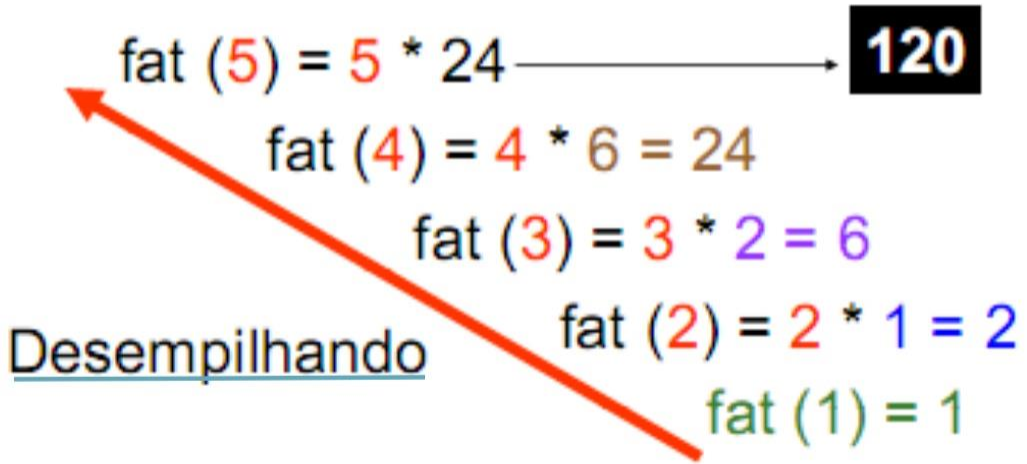
FATORIAL:
Problema clássico de recursão

fat (n) = n * fat (n-1) ➔ até que n = 1



FATORIAL:
Problema clássico de recursão

fat (n) = n * fat (n-1) ➔ até que n = 1



Versão Recursiva

Substituição do laço de repetição pela chamada de si mesma.

```
funcao fatorial(n: inteiro): inteiro
inicio
    se n = 0 entao
        retorne 1
      senao
        retorne n*fatorial(n-1)
    fimse
fimfuncao
```

Versão Recursiva

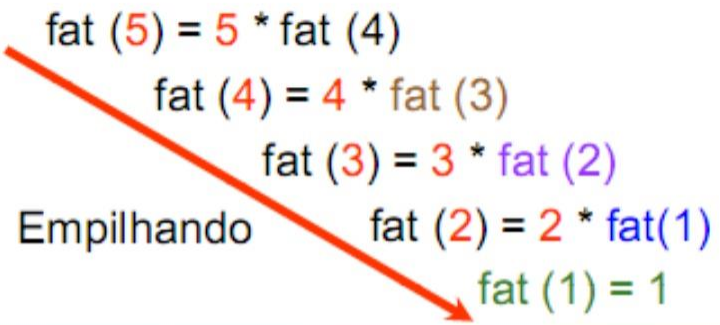
Substituição do laço de repetição pela chamada de si mesma.

Versão em **linguagem C**

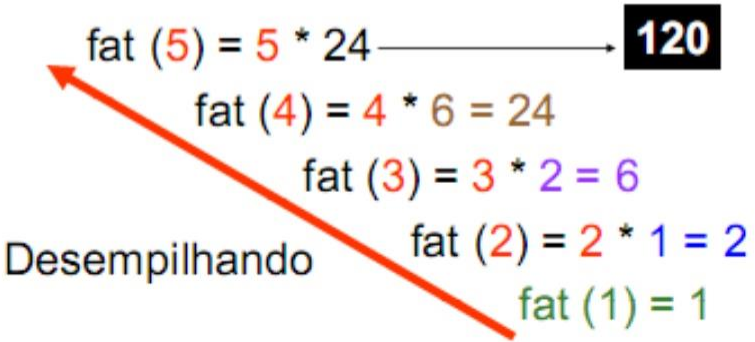
```c
int fatorial(int n){
   if (n==0)
      return 1;
   else
      return n*fatorial(n-1);
}
```

```
funcao fatorial(n: inteiro): inteiro
inicio
    se n = 0 entao
        retorne 1
      senao
        retorne n*fatorial(n-1)
    fimse
fimfuncao
```

fat (n) = n * fat (n-1) ➔ até que n = 1



fat (n) = n * fat (n-1) ➔ até que n = 1



FATORIAL iterativo versus recursivo

```
funcao fatorial(n: inteiro): inteiro
var i, aux: inteiro
inicio
    aux ← 1
    para i ← 1 ate n faca
        aux ← aux * i
    fimpara
    retorne aux
fimfuncao
```

```
funcao fatorial(n: inteiro): inteiro
inicio
    se n = 0 entao
        retorne 1
     senao
        retorne n*fatorial(n-1)
    fimse
fimfuncao
```

FATORIAL iterativo versus recursivo

Versões em **linguagem C**

```c
int fatorial(int n){
  int i,aux;
  aux=1;
  for (i=1; i<=n; i++)
    aux=aux*i;
  return aux;
}
```

Iterativo

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

Recursivo

POTÊNCIA iterativo versus recursivo

```
funcao potencia(ba,ex: inteiro): inteiro
var i, aux: inteiro
inicio
    aux ← 1
    para i ← 1 ate ex faca
        aux ← aux * ba
    fimpara
    retorne aux
fimfuncao
```

```
funcao potencia(ba,ex: inteiro): inteiro
inicio
    se ex = 0 entao
        retorne 1
    senao
        retorne ba * potencia(ba, ex - 1)
    fimse
fimfuncao
```

POTÊNCIA iterativo versus recursivo

Versões em **linguagem C**

```c
int potencia(int ba, int ex){
  int i,aux;
  aux=1;
  for (i=1; i<=ex; i++)
    aux=aux*ba;
  return aux;
}
```

Iterativo

```c
int potencia(int ba, int ex){
  if (ex==0)
    return 1;
  else
    return (ba * potencia(ba, ex-1));
}
```

Recursivo

Qual o maior problema relacionado a recursividade?

São as funções que NUNCA terminam!