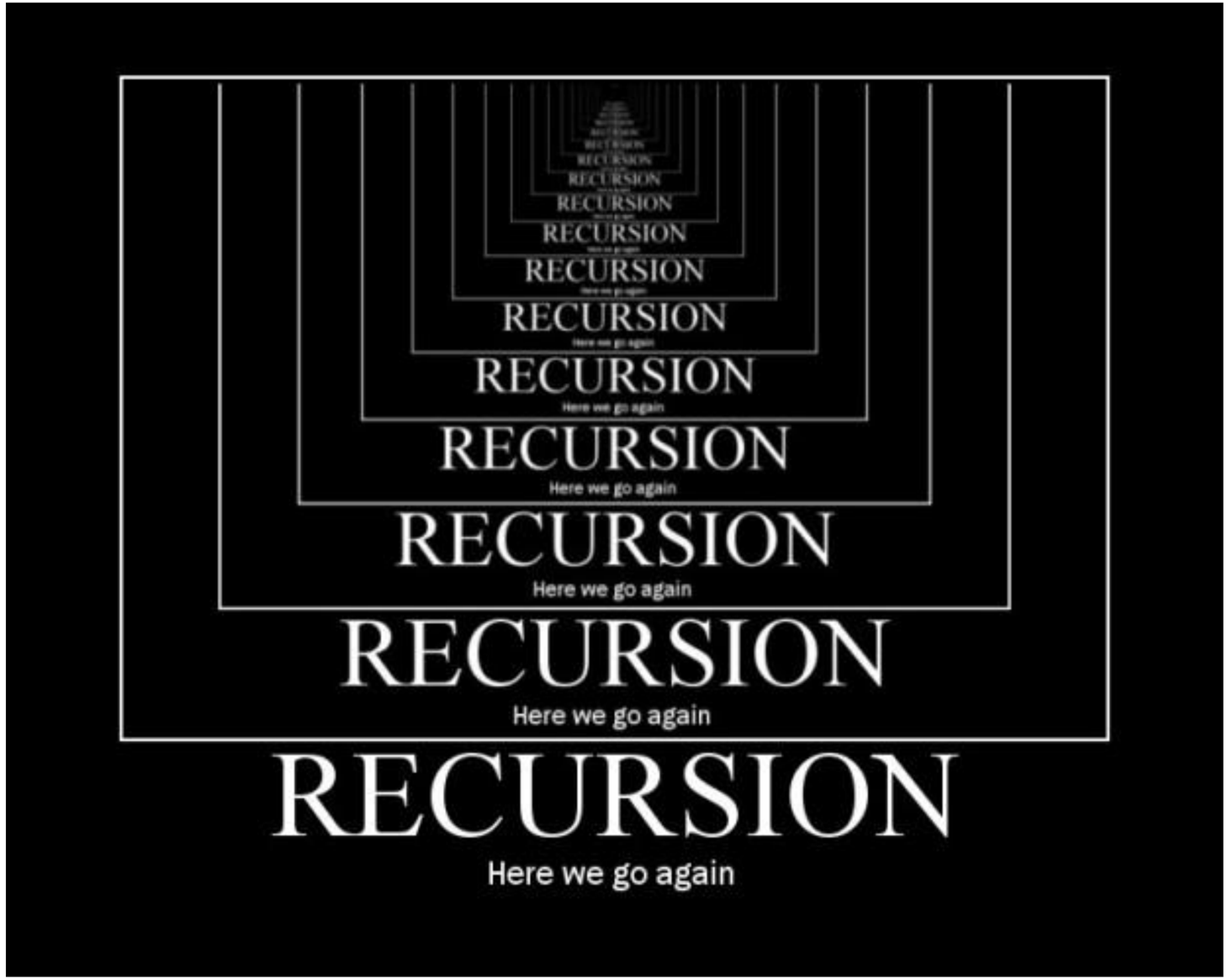
Recursão

Algoritmos

Recursão


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again

Recursividade

Recursividade

- É quando uma função envoca si mesmo para resolver um problema em uma instância menor!!!
- Programas recursivos são, em geral, mais simples de se escrever, analisar e entender!

Recursividade

- Para usar a recursividade temos que aprender como o computador se comporta durante a recursão!
- Para isto entenderemos a recursão!!

Recursividade

- Exemplo:
  - Os pais de uma pessoa são seus ancestrais (caso base);
  - Os pais de qualquer ancestral são também ancestrais da pessoa inicialmente considerada (passo recursivo).

Recursividade

- Dividir: Divida o problema em subproblemas menores, até encontrar o caso base;
- Conquiste os subproblemas: resolva cada problema menor de forma isolada;
- Recombine a resolução dos subproblemas:por meio das resoluções dos problemas menores, resolva o maior.

Recursividade

- Os algoritmos recursivos chamam a si mesmos , porém passando parâmetros diferentes (que em tese são a “instância menor” do que algoritmo irá resolver).

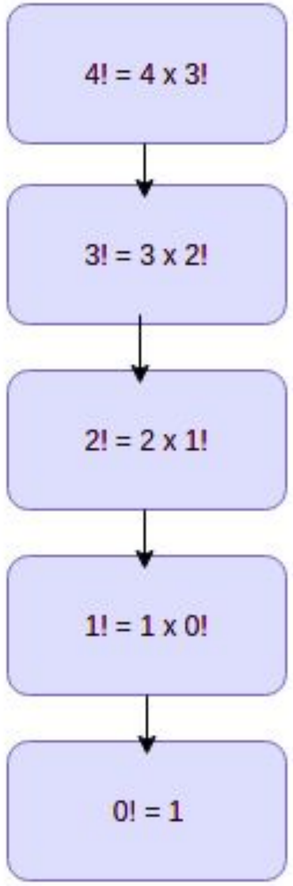
- Ao chamar a própria função com parâmetros diferentes podemos considerar que é um subproblema a ser resolvido.

Exemplo fatorial

- 4! = 4 x (4–1) x (4–2) x (4–3)4 = 4 x 3 x 2 x 1 = 24.
- Um fatorial de um número é o próprio número menos o fatorial do número menos um.
- 4! = 4 x 3! (o fatorial de 4, é 4 vezes o fatorial de 3)
- 3! = 3 x 2! (o fatorial de 3 é 3 vezes o fatorial de 2)
- 2! = 2 x 1! (o fatorial de 2 é 2 vezes o fatorial de 1)
- 1! = 1 x 0! (o fatorial de 1 é 1 vez o fatorial de 0)0! = o fatorial de zero é 1

Como fica nossa recursão?

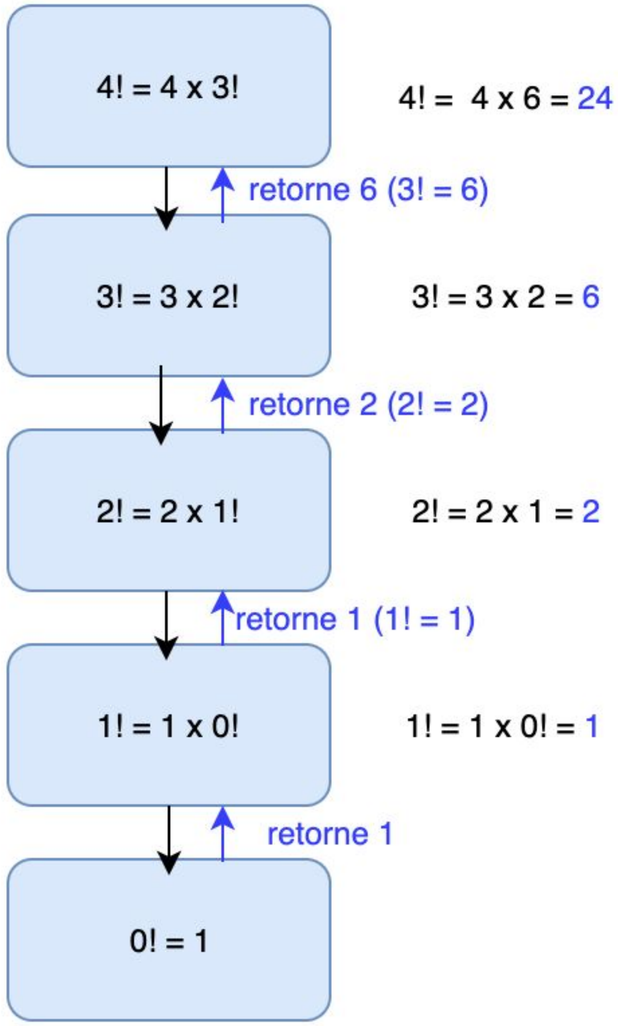
- Dividimos o problema em fatoriais menores até chegar no fatorial de zero.



Caso base

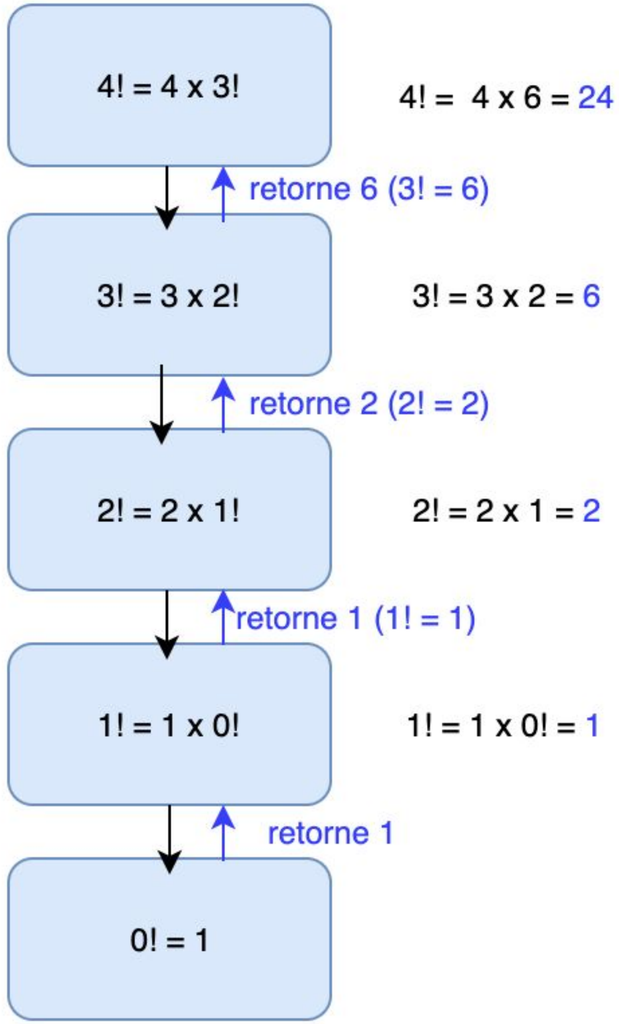
Como fica nossa recursão?

- E o que acontece quando chegarmos ao zero?
- Podemos retornar 1, pois 0! equivale a 1 e é o menor fatorial possível.



Como fica nossa recursão?

- Quebre o problema maior em subproblemas, até chegar ao caso base;
- Identifique o caso base (o caso base nesse caso é o fatorial de 0 que é igual a 1)
- A partir do caso base, comece a resolver os subproblemas um a um;
- Por fim, resolva o problema maior.

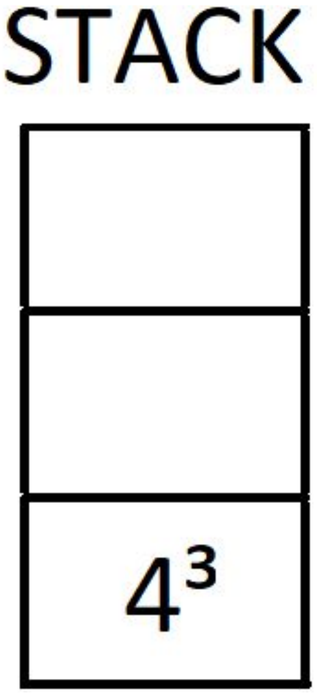


Como o computador faz?

- Pilha de execução:
  - Toda vez que fazemos uma chamada de função dentro do programa, o SO reserva memória para as variáveis e parâmetros desta função.
  - A função no topo da pilha é função sendo executada no momento.
  - Quando uma função termina de ser executada ela é removida da pilha.
  - Quando ocorre uma nova chamada de função, a função chamada é colocada imediatamente no topo da pilha e começa a ser executada.

Como o computador faz? (Exponenciação)

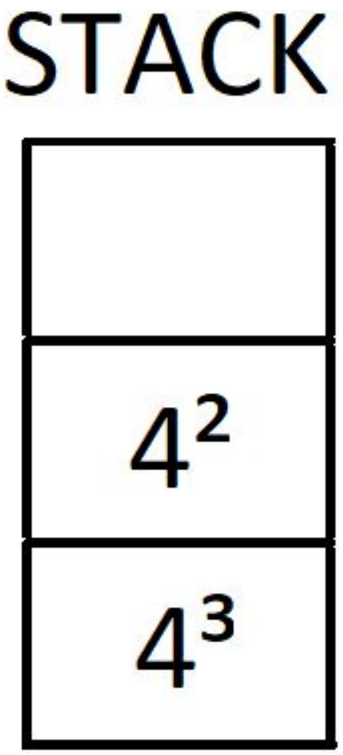
pow(4,3)



Como o computador faz?

pow(4,3)

↳ 4 * pow(4,2)



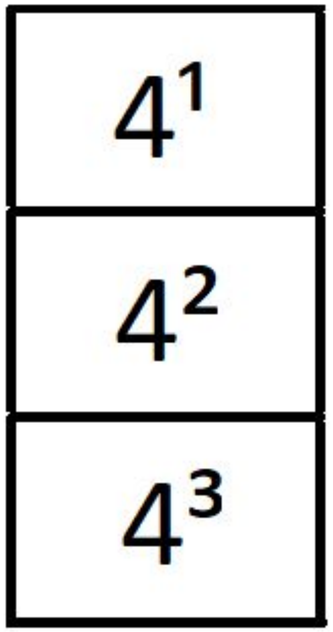
Como o computador faz?

pow(4,3)

↳ 4 * pow(4,2)

↳ 4 * pow(4,1)

STACK



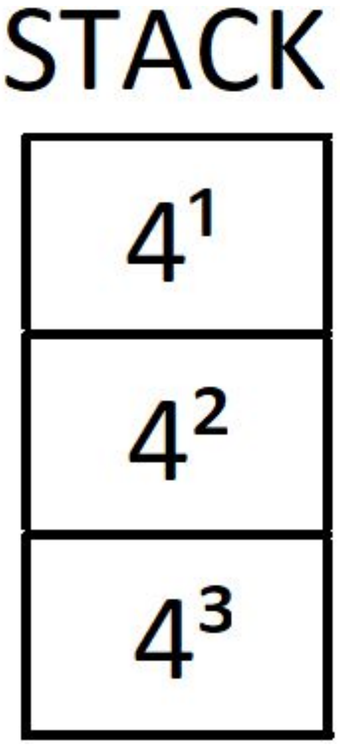
Como o computador faz?

pow(4,3)

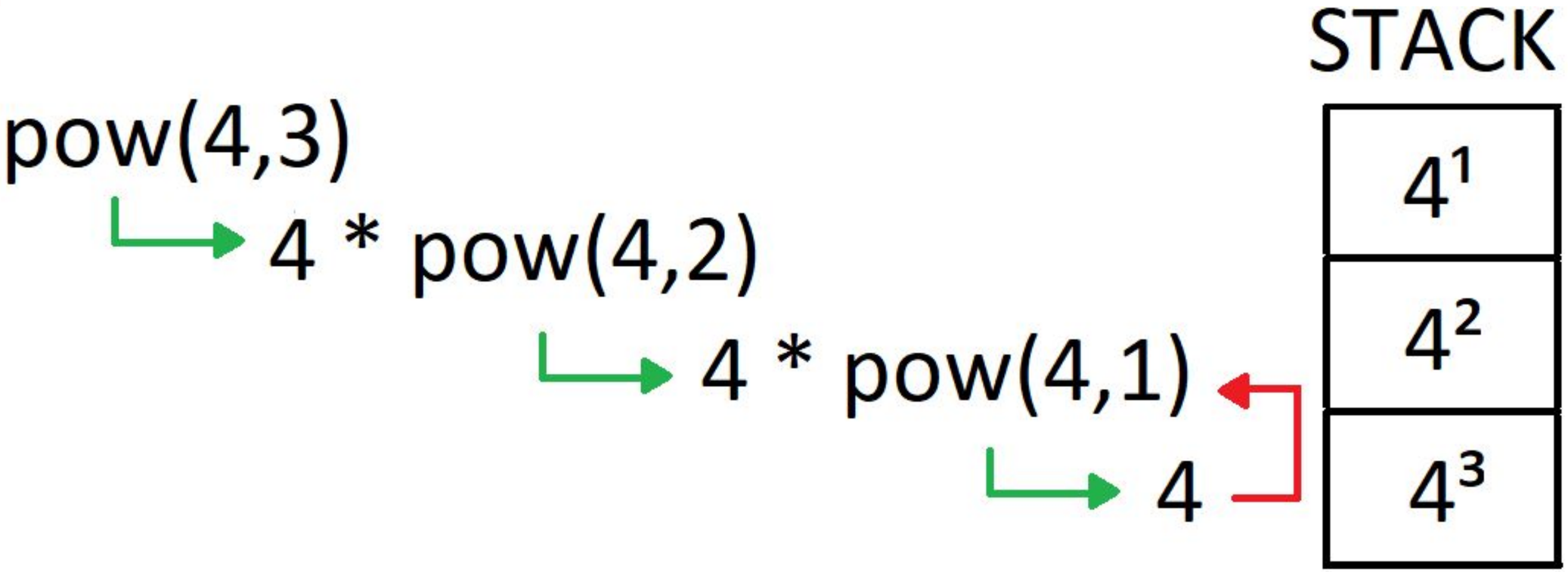
↳ 4 * pow(4,2)

↳ 4 * pow(4,1)

↳ 4



Como o computador faz?



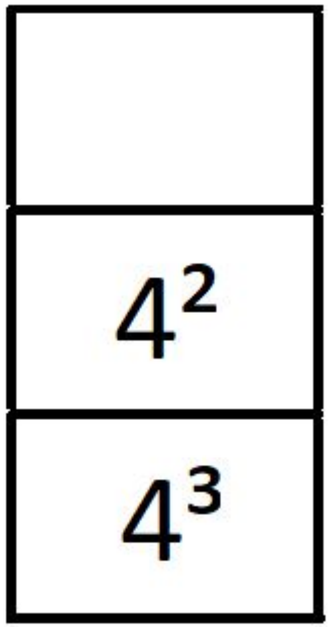
Como o computador faz?

pow(4,3)

→ 4 * pow(4,2)

→ 4 * 4

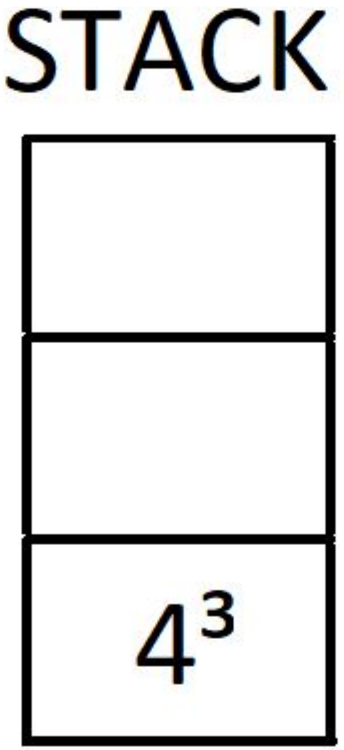
STACK



Como o computador faz?

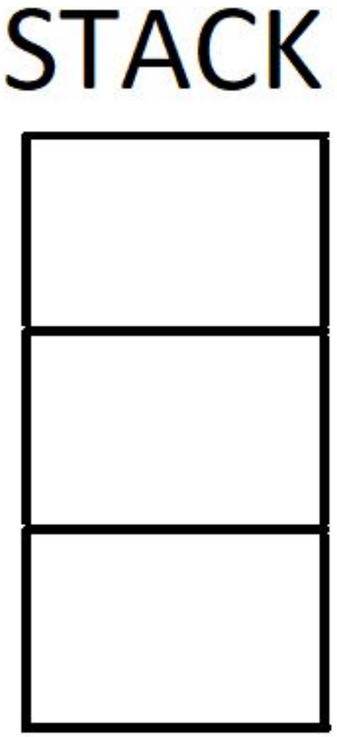
pow(4,3)

→ 4 * 16 ←



Como o computador faz?

64



Como o computador faz?

- Uma chamada recursiva nada mais é que uma simples chamada de função.
- Tudo que acontece é que várias instâncias da mesma função ficam empilhadas.
- Cada instância da função na pilha tem suas próprias variáveis locais, por isso o que acontece em uma instância de uma função recursiva não influencia o que acontece em outras instâncias.

Exemplo exponenciação

```
int pow_recursivo(int x, int n){
    if(n == 1)
        return x;

    return x * pow_recursivo(x,n-1);
}
```

Como fica o Fatorial?

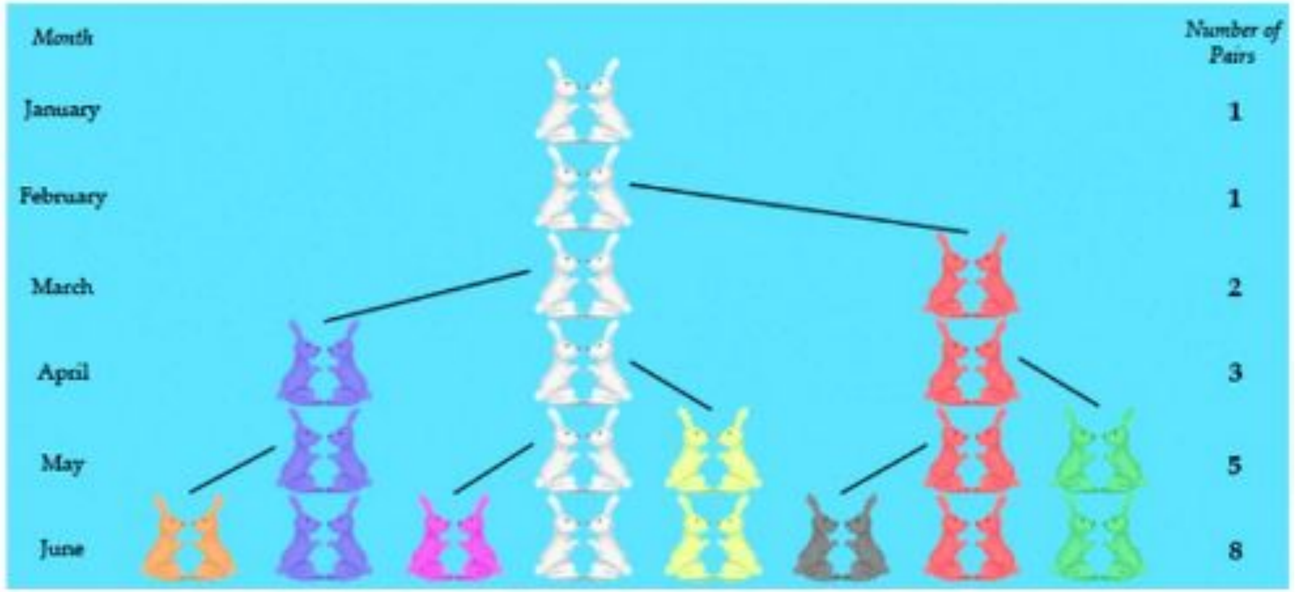
```c
#include <stdio.h>

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

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", F(num));
```

Fibonacci

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Fibonacci

- Os números de Fibonacci foram propostos por Leonardo di Pisa (Fibonacci), em 1202, como uma solução para o problema de determinaro tamanho da população de coelhos
  - (*) fonte http://www.oxfordmathcenter.com/drupal7/node/487.



Fibonacci