Introdução à Recursão

Sei o valor de 26!, como isso ajuda a calcular 27!? RESP: $27! = 27 \cdot 26!$.

Sei o valor de 52!, como isso ajuda a calcular 53!? RESP: $53! = 53 \cdot 52!$.

Generalizando, podemos calcular $n!$ da seguinte forma:

$$f(n) = n \cdot f(n-1)$$

Qual o problema na função definida acima? RESPOSTA: Não pára!

O correto seria escrever:

$$\mathrm{f}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot \mathrm{f}(n-1) & \text{se } n > 0. \end{cases}$$

Seria possível usar algo parecido em linguagens de programação? SIM!

Exemplo. *Em linguagem C:*

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

Exemplo. *Em linguagem Pascal:*

```
function f(n: integer): integer;
begin
  if n = 0 then
     f := 1
  else
     f := n * f(n-1);
end;
```

O método de definir uma função (matemática ou computacional) a partir dela mesma é chamado de *recursão*.

Definição. *Uma função definida recursivamente é chamada de* função recursiva *ou relação de recorrência. Tal função sempre tem dois elementos:*

1. ***base****: o(s) caso(s) computados sem necessidade de recursão.*

2. ***recursão****: computação recursiva para os casos que não são base.*

Uma vantagem: É necessário pouco esforço para traduzir a função recursiva matemática para a função recursiva em código de computador.

Exemplo. *Defina x^n como uma função (matemática) recursiva e em seguida traduza para código de computador:*

$$x^n = \begin{cases} 1 & \text{se } n = 0 \\ x \cdot x^{n-1} & \text{se } n > 0. \end{cases}$$

```
float potencia (float x, int n) {
  if (n==0) return 1.0;
  return x * potencia (x,n-1);
}
```

Compare com a versão não recursiva:

```
float potencia_nao_recursiva (float x, int n) {
  if (n == 0)
      return 1;

  float result = x;

  while (n>1) {
     result = result * x;
     n = n - 1;
  }
  return result;
}
```

Geralmente, os código recursivos tem algumas vantagens:

- O código recursivo é mais intuitivo: parece mais com a definição matemática.
- Aumenta a legibilidade, simplifica a codificação.
- Grande parte de códigos recursivos são menores.

Passo a Passo para Códigos Recursivos

O passo-a-passo para escrever um código recursivo é:

1. Escreva o protótipo da sua função: definir as entradas e saídas.
2. Escreva o código para o caso base.
3. Tome um exemplo concreto e responda: como usar o valor já calculado de um exemplo mais próxima da base para calcular o valor do exemplo mais distante da base?
4. Generalize o passo anterior para escrever o código da parte recursiva.

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

1. *Protótipo:* `void count_down (int n)`
2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down(8)`. *Ideia: imprime* 8 *e chama* `count_down(7)`.
4. *Código da parte recursiva:*

```
   else {
      printf("%d, ", n);
      count_down (n - 1);
   }
}
```

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$.

1. *Protótipo:* `void count_down_up (int n)`
2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down_up(4)`. *Ideia: imprime* 4, *chama* `count_down_up(3)`, *imprime* 4.

4. *Código da parte recursiva:*

```
    else {
        printf("%d, ", n);
        count_down_up (n - 1);
        printf("%d, ", n);
    }
}
```

Sequência de Fibonacci

A *sequência de Fibonacci* são os números na seguinte sequência:

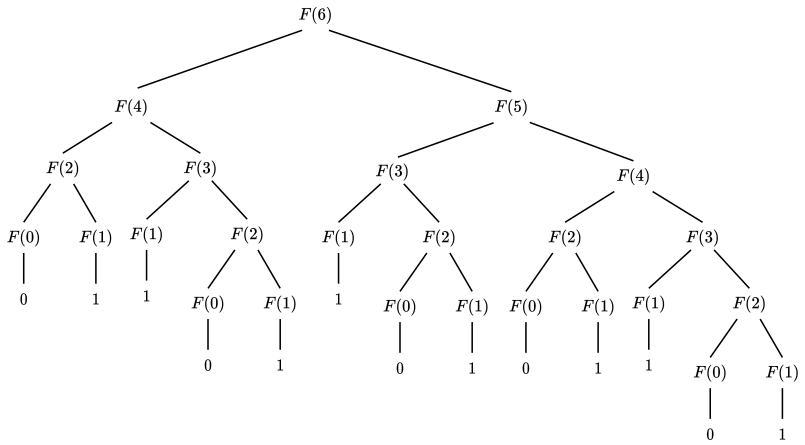
$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$$

Por definição, os primeiros dois números da sequência são 0 e 1, e cada número subsequente é a soma dos dois anteriores.

Como calcular o n-ésimo número da sequência de Fibonacci?

$$F(n) = \begin{cases} n & \text{se } n \leq 1 \\ F(n-1) + F(n-2) & \text{se } n > 1. \end{cases}$$

Em sala: Escreva o código recursivo e iterativo (não-recursivo) que calcula o $F(n)$.



Teorema. *O número de adições que a versão recursiva de $F(n)$ executa é igual a $F(n+1)-1$.*

n	$F(n+1)$	num. de adições	num. chamadas de função
6	13	12	25
10	89	88	177
15	987	986	1973
20	10946	10945	21891
25	121393	121392	242785
30	1346269	1346268	2692537

Uma desvantagem da recursão: chamada de função é custosa.