RECURSIVIDADE

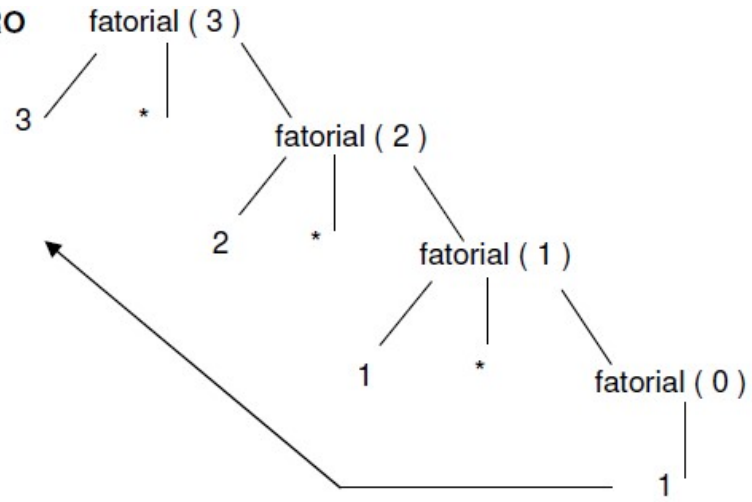
- A recursividade pode ser compreendida quando na prática surge a necessidade de se definir alguma coisa em função dela mesma.
- Em muitos problemas, uma solução recursiva é muito mais simples, natural ou intuitiva do que uma solução repetitiva ou iterativa.
- Uma subrotina é recursiva se definida em termos de si mesma.

Considere a seguinte definição de fatorial

$$n! = \begin{cases} 1 & , se\ n = 0 \\ n\,(n-1)! & , se\ n > 0 \end{cases}$$

Com esta definição, o fatorial é definido em seus próprios termos.

```
FUNÇÃO fatorial (INTEIRO n) : INTEIRO
INÍCIO
    SE n = 0 ENTÃO
        fatorial ← 1
    SENÃO
        fatorial ← n * fatorial ( n – 1 )
    FIM SE
FIM
```



Estrutura básica de uma função/método recursivo:

Se condição de parada

Fim

Senão

Chamada recursiva (n -1)

Considerações:

- Usar quando o problema é definido recursivamente ou seus dados
- Garantir um fim (garantir que a função a ser chamada é menos abrangente que a atual)
- São mais lentas que funções iterativas
- Erros na implementação ou limitações de máquina podem levar a estouro de pilha

RECURSIVIDADE

Uma subrotina pode ser:

- **diretamente recursiva**: P ativa P;
- **indiretamente recursiva**: P ativa Q ativa R....ativa P

A natureza recursiva do problema não garante que um algoritmo recursivo seja a melhor solução. Todo algoritmo recursivo pode ser transformado num algoritmo não recursivo que, apesar de ser, normalmente, mais complexo e menos claro, muitas vezes é mais eficiente em relação a espaço e tempo.

Passos para desenvolvimento de algoritmos recursivos

Passo 1: Entender o problema.
Exemplo: Série de Fibonacci -> 0 1 1 2 3 5 8 13 21 34 55...

Passo 2: Formular o problema em uma (ou mais) subrotinas recursivos
Fibonacci(1) = 0
Fibonacci(2) = 1
Fibonacci(3) = 1
Fibonacci(4) = 2
Fibonacci(5) = 3
Fibonacci(6) = 5

Qual é a ordem?

Fibonacci(1) = 0
Fibonacci(2) = 1
Fibonacci(n) = Fibonacci(n -1) + Fibonacci(n - 2)

Passo 3: Implementar.

```
long fibonacci (int n) {
     if (n == 1)
          return 1;
     else
          if (n == 2)
               return 1;
          else
               return fibonacci (n – 1) + fibonacci (n – 2);
}
```

RECURSIVIDADE EM VETORES

- Uso de dados com características recursivas.

$$\text{Vetor} = \begin{cases} \text{Se tamanho}=1, [1^{o}\ \text{elemento}] \\ \text{Se tamanho}>1, [1^{o}\ \text{elemento}] + \text{Vetor com tamanho-1} \end{cases}$$

Vantagens:

- Preocupa-se somente com o dado atual
- Diminui consideravelmente a complexidade das estruturas de dados complexas

Exemplo – contagem de elementos maiores que N (recursivo):

```
TIPO TVETINT = VETOR [1..100] DE INTEIRO
FUNCAO maiores_N (n, tamanho : INTEIRO; var vetor: TVETINT ) : INTEIRO
INICIO
   SE tamanho = 0 ENTAO
      maiores_N ← 0    // serve tb para inicializar o somatorio
   SENAO
      // verifica se o ultimo elemento eh maior que n
      SE vetor[tamanho] > n ENTAO
            maiores_N ← 1 + maiores_N(n, tamanho-1, vetor)
      SENAO
            maiores_N ← maiores_ N(n, tamanho-1, vetor)
      FIM SE
   FIM SE
FIM
```

RECURSIVIDADE EM VETORES

Exemplo – acha maior elemento do vetor (recursivo):

```
TIPO TVETINT = VETOR [1..100] DE INTEIRO

FUNCAO acha_maior (tamanho: INTEIRO; var vetor: TVETINT) : INTEIRO
VAR
      maiorElementoRestoLista : INTEIRO
INICIO
   SE tamanho = 1 ENTAO
      acha_maior ← vetor [1]
   SENAO          // busca o maior elemento do resto da lista
      maiorElementoRestoLista ← acha_maior (tamanho-1, vetor)
      SE vetor[tamanho] > maiorElementoRestoLista ENTAO
            acha_maior ← vetor [tamanho]
      SENAO
            acha_maior ← maiorElementoRestoLista
      FIM SE
   FIM SE
FIM
```

Exemplo – verifica a existência de um elemento qualquer no vetor (recursivo):

```
TIPO TVETINT = VETOR [1..100] DE INTEIRO

FUNCAO existeElem (elemento, tamanho: INTEIRO; var vetor: TVETINT): LOGICO
INICIO
   SE tamanho = 1 ENTAO
      SE vetor[1] = elemento ENTAO
            existeElem ← VERDADE
      SENAO
            existeElem ← FALSO
      FIM SE
   SENAO
      SE vetor[tamanho] = elemento ENTAO
            existeElem ← VERDADE
      SENAO
            existeElem ← existeElem (elemento, tamanho-1, vetor)
      FIM SE
   FIM SE
FIM
```