ALGORITMOS

TEMA-02

RECURSIVIDADE

Considere uma linguagem de programação qualquer, que aceite recursividade, de tal forma que a função Resultado(N) é implementada conforme algoritmo a seguir:

Função Resultado (inteiro: **N**)
Se N > 0
    Então Retorna **N***Resultado(**N**-1)
    Senão Retorna **1**
Fim-Se
Fim-Resultado

Neste caso, quanto vale Resultado(**6**) ?

Função Resultado (inteiro: **N**)
Se N > 0
    Então Retorna **N***Resultado(**N**-1)
    Senão Retorna **1**
Fim -Se
Fim-Resultado
Neste caso, quanto vale Resultado(**6**) ?

R(-2) = 1
R(-1) = 1
R(**0**) = 1
R(**1**) = **1** × R (1-1) = 1 × R (**0**) = 1 × 1 = 1
R(**2**) = **2** × R (2-1) = 2 × R (**1**) = 2 × 1 = 2
R(**3**) = **3** × R (3-1) = 3 × R (**2**) = 3 × 2 = 6
R(**4**) = **4** × R (4-1) = 4 × R (**3**) = 4 × 6 = 24
R(**5**) = **5** × R (5-1) = 5 × R (**4**) = 5 × 24 = 120
R(**6**) = **6** × R (6-1) = 6 × R (**5**) = 6 × 120 = **720**

Professor: Manuel Martins
Monitor: Raphael Machado

O que é RECURSIVIDADE NA ÁREA DE PROGRAMAÇÃO?

Em geral, uma rotina recursiva R pode ser expressa como uma composição formada por um conjunto de comandos C (que não contém chamadas R) e uma chamada (recursiva) à rotina R, onde:

R = [C, R]

Um algoritmo que para resolver um problema divide-o em subprogramas mais simples, cujas soluções requerem a aplicação dele mesmo.

É uma forma de resolver problemas por meio da divisão dos problemas em problemas menores de mesma natureza onde deve-se parar quando alcançarmos um caso trivial que conhecemos a solução.

Professor: Manuel Martins
Monitor: Raphael Machado

SABE COMO PODEMOS ENTENDER O CONCEITO DE RECURSIVIDADE?

Vamos brincar com **BONECAS RUSSAS!!**

Quem saberia me dizer, sem chutes, quantas Bonecas existem dentro desta Boneca?

1 2 3 4 5 6 7

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (inteiro: N)
Se N > 0
  Então Retorna N*Resultado(N-1)
  Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(N) =

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (6)
Se 6 > 0
   Então Retorna 6*Resultado(6-1)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*Resultado(6-1)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (6)
Se 6 > 0
   Então Retorna 6*Resultado(5)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*Resultado(5)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (5)
Se 5 > 0
   Então Retorna 5*Resultado(4)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*5*Resultado(4)

6*Resultado(5)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (4)
Se 4 > 0
   Então Retorna 4*Resultado(3)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*5*4*Resultado(3)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (3)
Se 3 > 0
   Então Retorna 3*Resultado(2)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*5*4*3*Resultado(2)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (2)
Se 2 > 0
   Então Retorna 2*Resultado(1)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*5*4*3*2*Resultado(1)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (1)
Se 1 > 0
   Então Retorna 1*Resultado(0)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

Resultado(6) = 6*5*4*3*2*Resultado(1)

Professor: Manuel Martins
Monitor: Raphael Machado

07 – Quanto vale Resultado (6)?

```
Função Resultado (0)
Se 0 > 0
   Então Retorna 0*Resultado(0)
   Senão Retorna 1
Fim-se
Fim-Resultado
```

FIM DO PROGRAMA???

Resultado(6) = 6*5*4*3*2*1*Resultado(0)

A FUNÇÃO RESULTADO(0) RETORNARÁ OS RESULTADOS DE FORMA RECURSIVA ATÉ RESULTADO(6), GERANDO O RESULTADO FINAL

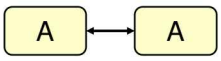
CONSTRUÇÃO DE ALGORITMOS RECURSIVIDADE

VÍDEO-02

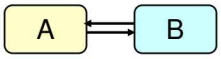
RECURSIVIDADE

As funções podem ser chamadas RECURSIVAMENTE, isto é, dentro do corpo de uma função podemos chamar novamente a PRÓPRIA FUNÇÃO.

Se uma função A chama a própria função A, dizemos que ocorre uma RECURSÃO DIRETA.



Se uma função A chama uma função B que, por sua vez, chama A, temos uma RECURSÃO INDIRETA.



RECURSIVIDADE

RECURSÃO - é um método de resolução de problemas que envolve quebrar um problema em subproblemas menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente. Normalmente recursão envolve uma função que chama a si mesma. Embora possa não parecer muito, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar.

Todos os ALGORITMOS RECURSIVOS devem obedecer a três leis importantes:

1- Um algoritmo recursivo deve ter um ***caso básico***.

2- Um algoritmo recursivo deve ***mudar*** o ***seu estado*** e ***se aproximar*** do caso básico.

3- Um algoritmo recursivo deve ***chamar a si mesmo***, recursivamente.

CASO BÁSICO = REGRA DE PARADA !

RECURSIVIDADE

Diversas implementações ficam muito mais fáceis usando **recursividade**. Por outro lado, implementações **NÃO RECURSIVAS** tendem a ser mais eficientes.

Para cada chamada de uma função recursiva, os parâmetros e as variáveis locais são empilhados na pilha de execução. Assim, mesmo quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada. As variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

RECURSIVIDADE

As implementações **recursivas** devem ser **pensadas** considerando-se a **definição recursiva do problema** que **desejamos resolver**. Por exemplo, o valor do fatorial de um número pode ser definido de forma recursiva:

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

Quanto vale 1! ?
E 0!

ANÁLISE COMBINATÓRIA

FATORIAL ⇒ UM BREVE RESUMO !

$$n! = \prod_{k=1}^{n} k \quad \forall n \in N \quad \prod = \text{PRODUTÓRIO}$$

EXEMPLOS

1! = 1
2! =1×2 = 2
3! =1×2×3 = 6
4! =1×2×3×4 = 24
5! =1×2×3×4×5 =120

6! =1×2×3×4×5×6 = **720**
5!

ANÁLISE COMBINATÓRIA

FATORIAL ⇒ UM BREVE RESUMO !

(n+1)! = n! × (n+1) ⇒ FUNÇÃO RECURSIVA

EXEMPLOS

2! = 1!×2
3! = 2!×3 = 1×2×3
4! = 3!×4 = 1×2×3×4
5! = 4!×5 = 1×2×3×4×5
6! = 5!×6 = 1×2×3×4×5×6
.....
10! = 9!×10 = 1×2×3×4×5×6×7×8×9×10
9!

(n+1)! = n! × (n+1)

ANÁLISE COMBINATÓRIA

FATORIAL ⇒ UM BREVE RESUMO !

(n+1)! = n! × (n+1) ⇒ FUNÇÃO RECURSIVA

Fazendo n=**0** vem:
(**0**+1)! = **0**! × (**0**+1)
(1)! = **0** ! × 1
1 = **0**! × 1
1=**0**!
0! = 1 ⇒ **FATORIAL DE 0 (ZERO) É 1 !**

OLHOU...E VIU !

FEROZ !

RECURSIVIDADE

As implementações recursivas devem ser pensadas considerando-se a definição recursiva do problema que desejamos resolver. Por exemplo, o valor do fatorial de um número pode ser definido de forma recursiva:

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

```
/* Função recursiva para calculo do fatorial */
int fat (int n)
{
if (n==0)
      return 1;
 else
      return n*fat(n-1);
}
```

```c
// ALGORITMO Q3 ATIVIDADE SUPERVISIONADA
#include <stdio.h>
int resultado(int);  // Protótipo da Função
main()
{
int n;
printf("Entre com o numero desejado: ");
scanf("%d", &n);
printf("%d ! = %d\n", n, resultado (n));
}
int resultado (int num) // Função Resultado
{
        if(num > 1)
                return(num*resultado(num-1));  // Recursividade
        else
                return(1);
}
```

FUNÇÃO FATORIAL

Consideremos a função fatorial abaixo

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

Vamos executar a função para **n = 4**

```c
int fatorial (int n);          n = 4
{
  if (n==0)
     return 1;
  else
     return n*fatorial(n-1);
}
```

EMPILHA fatorial(4)
EMPILHA fatorial(3)
EMPILHA fatorial(2)
EMPILHA fatorial(1)
EMPILHA fatorial(0)

PILHA DE EXECUÇÃO

fatorial(0) → return 1 !!!!
fatorial(1) → return 1*fatorial(0)
fatorial(2) → return 2*fatorial(1)
fatorial(3) → return 3*fatorial(2)
fatorial(4) → return 4*fatorial(3)

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

DESEMPILHA fatorial(0)

PILHA DE EXECUÇÃO

fatorial(0) → return 1 !!!!
fatorial(1) → return 1*fatorial(0)
fatorial(2) → return 2*fatorial(1)
fatorial(3) → return 3*fatorial(2)
fatorial(4) → return 4*fatorial(3)

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

DESEMPILHA fatorial(1)

PILHA DE EXECUÇÃO

fatorial(1) → return 1*1
fatorial(2) → return 2*fatorial(1)
fatorial(3) → return 3*fatorial(2)
fatorial(4) → return 4*fatorial(3)

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

DESEMPILHA fatorial(2)

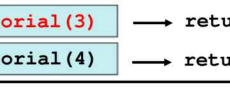
PILHA DE EXECUÇÃO

fatorial(2) → return 2*(1*1)
fatorial(3) → return 3*fatorial(2)
fatorial(4) → return 4*fatorial(3)

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

PILHA DE EXECUÇÃO

DESEMPILHA fatorial(3)



fatorial(3) → return 3*(2*1*1)

fatorial(4) → return 4*fatorial(3)

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

PILHA DE EXECUÇÃO

DESEMPILHA fatorial(4)

fatorial(4) → return 4*(3*2*1*1)

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

PILHA DE EXECUÇÃO

Fatorial(4)= 24

FEROZ !

VAZIA!

OLHOU...E VIU!

ALGORITMOS FLUTUA ACIMA DO BEM...E DO MAL!

Carpe Diem!