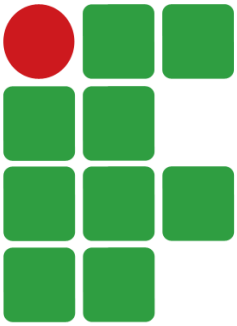


INSTITUTO FEDERAL
Mato Grosso
Campus Cuiabá - Octayde Jorge da Silva

LISTA 4 - RECURSIVIDADE

Aluno: Vitor Bruno de Oliveira Barth

Professor: Ruy de Oliveira

Disciplina: Algoritmos II

Cuiabá

2016

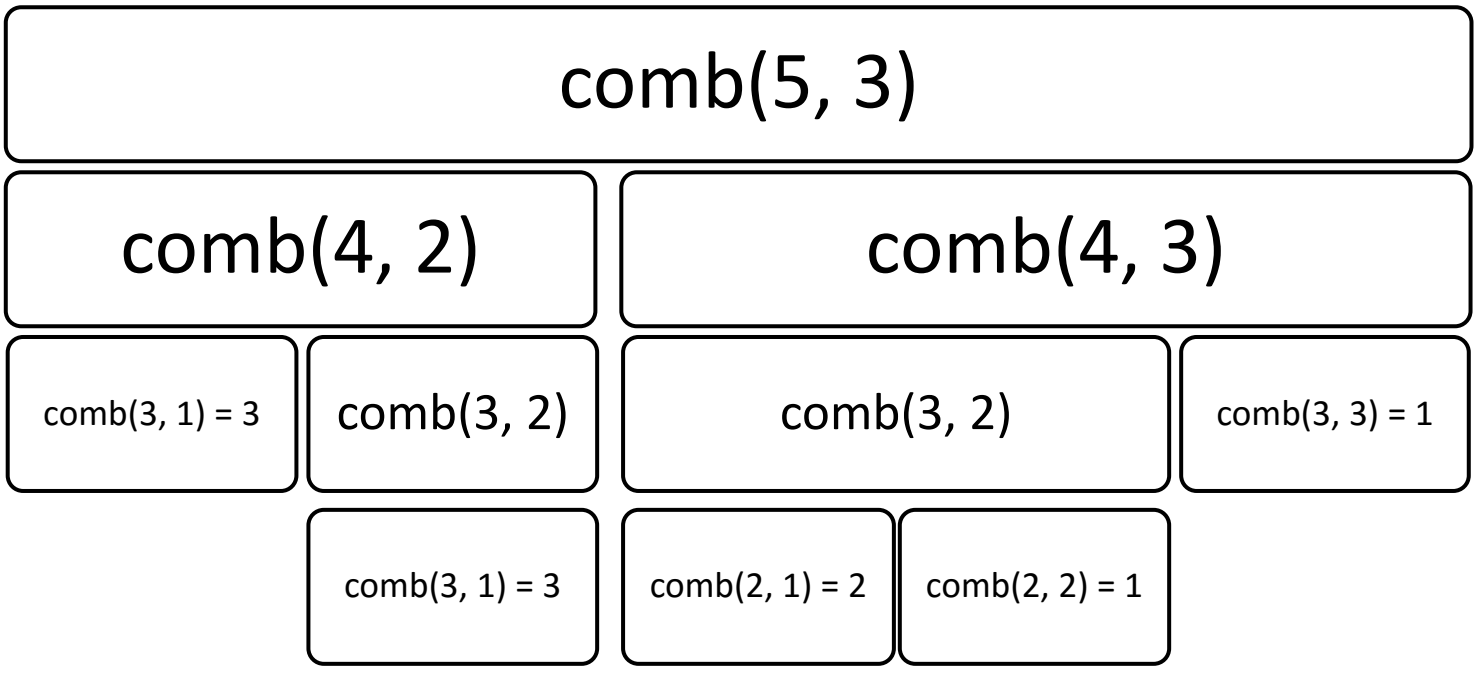
1) Considere a função ***Comb(n, k)***, que representa o número de grupos distintos com ***k*** pessoas que podem ser formados a partir de ***n*** pessoas. Por exemplo, *Comb*(4, 3) = 4, pois com 4 pessoas (A, B, C, D), é possível formar 4 diferentes grupos: ABC, ABD, ACD e BCD. Sabe-se que:

$$Comb(n,k) = \begin{cases} n & \text{se } k = 1 \\ 1 & \text{se } k = n \\ Comb(n-1,k-1) + Comb(n-1,k) & \text{se } 1 < k < n \end{cases}$$

Implemente em portugol uma função recursiva para *Comb* (*n, k*) e mostre o diagrama de execução para chamada *Comb* (5, 3). Sabendo-se ainda *Comb* (*n, k*) = *n*! / (*k*! (*n*-*k*)!), implemente uma função não recursiva de *Comb* (*n, k*).

```
algoritmo { calcula uma combinação através de recursividade }
    declare n, k, comb
    leia n, k
    imprima comb(n, k)
fim-algoritmo

subrotina comb(n, k numéricos)
    declare resultado numerico
    se k = 1 então {caso base }
        resultado <- n
    senão
        se k=n então
            comb(1, 1) { se k = n, comb = 1 }
        senão
            resultado <- comb(n-1,k-1) + comb (n-1,k)
        fim-se
    fim-se
    retorna resultado
fim-subrotina
```



```
algoritmo { resolve combinação através de fatorial }
        declare n, k, comb
        leia n, k
        imprima fatorial(n)/(fatorial(k)*fatorial(n-k))
fim-algoritmo


subrotina fatorial(n numéricos)
        se n = 0 então
                resultado <- 1
        senão
                resultado <- n*fatorial(n-1)
        fim-se
fim-subrotina
```

2 - Implemente recursivamente uma função Max que retorne o maior valor armazenado em um vetor V, contendo n números inteiros.

```
algoritmo { retorne o maior valor de um vetor }
        declare i, n numerico
        leia n
        declare v1[n] numérico
        leia v1
        max(v1, n)
fim-algoritmo

subrotina max(v1, n numéricos)
        declare resultado numéricos
        se n = 2 então
                resultado <- maior(v1[0], v1[1])
        senão
                resultado <- maior(v1[n], max(v1, n-1))
        fim-se
        retorna resultado
fim-subrotina


subrotina maior(a, b, numéricos)
        declare resultado
        se a > b
                resultado <- a
        senão
                resultado <- b
        fim-se
        retorne resultado
fim-subrotina
```

3) Dada a implementação em portugol da função abaixo:

```
função F(N : natural) : natural
início
  se N < 4 então
    retorne 3 N
   senão
    retorne 2 * F(N - 4) + 5
fim
```

Quais são os valores de F(3) e de F(7)?

F(3) = 3*3 = 9

F(7) = 2*F(3)+5 = 23

F(3) = 3*3 = 9

4) O cálculo da raiz quadrada de um número real x pode ser feito através do seguinte algoritmo:

$$RaizQ(x, x_0, \varepsilon) = \begin{cases} x_0 & se\ |\ x_0^2 - x\ | \leq \varepsilon \\ RaizQ(x, \frac{x_0^2 + x}{2\,x_0}, \varepsilon) & \text{caso contrário} \end{cases}$$

em que x_0 é uma aproximação inicial do valor $\sqrt{x}$ e ε é um erro admissível. Implemente o algoritmo em Portugol e mostre o diagrama de execução para a chamada $RaizQ(13, 3.2, 0.001)$.

```
algoritmo
    declare raiz, aprox, erro numéricos
    leia raiz, aprox, erro
    imprima raizq(raiz, aprox, erro)
fim-algoritmo

subrotina raizq (r, a, e numérico)
    se ((a*a)-r) <= e então { caso base }
        retorne a
    senão
        raizq(r, (((a*a)+r)/(2*a)), e)
    fim-se
fim-subrotina
```

RaizQ(13, 3.2, 0.001)

RaizQ(13, 3.63125, 0.001)

RaizQ(13, 3.60564, 0.001)

3.60564^2 = 13,0006 < 0.001

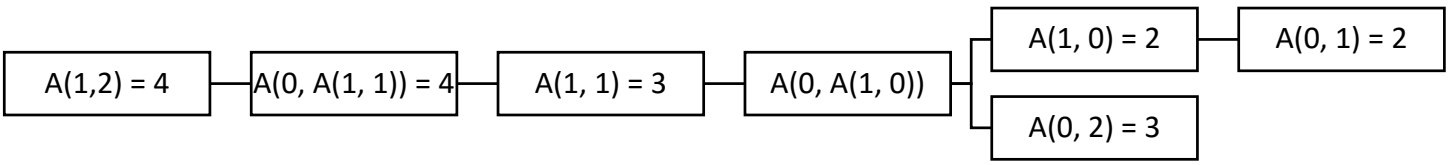
5) Dada a definição da função de Ackerman:

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(n-1,1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1, A(m,n-1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

válida para valores inteiros não negativos de m e n, implemente uma versão recursiva do algoritmo e faça o diagrama de execução de $A(1, 2)$.

```
algoritmo
    declare m, n, x numérico
    leia m, n  x
    escreva A(m, n)
fim algoritmo

subrotina A (m, n numérico)
    se m = 0 então { caso base }
        resultado <- n+1
    senão se m > 0 && n = 0 então
        resultado <- A(m-1, 1)
    senão
        resultado <- A(m-1, A(m, n-1))
    fim-se
    retorne resultado
fim-subrotina
```



6) A função $f(x, n) = x^n$, em que x é um número real e n um número inteiro, pode ser calculada eficientemente como nos exemplos abaixo:

$x^0 = 1$; $x^1 = x$; $x^2 = x^2$; $x^3 = x\,x^2$; $x^4 = (x^2)^2$ $x^5 = x\,(x^2)^2$; $x^6 = (x\,x^2)^2$; $x^{11} = x\,((x\,(x^2)^2)^2$; $x^{-2} = 1/x^2$ etc.

Elabore a definição recursiva de $f(x, n)$ e implemente um algoritmo recursivo para $f(x,n)$.

```
algoritmo
    declare x, n numérico
    leia x, n
    imprima f(x, n)
fim-algoritmo

subrotina f(x, n numérico)
    declare resultado
    se n < 0
        resultado <- 1/f(x, n*-1)
    senão se n = 0 então
        resultado <- 1
    senão se n = 1 então
        resultado <- x
    senão se n = 2 então
        resultado <- x²
    senão se n%2 = 0 então
        resultado <- f(fx, n/2), 2)
    senão
        resultado <- x*f(x, n-1)
    fim-se
    retorne resultado
fim-subrotina
```

7) A recursividade pode ser utilizada para gerar todas as possíveis permutações de um conjunto de símbolos. Por exemplo, existem seis permutações no conjunto de símbolos A, B e C: ABC, ACB, BAC, BCA, CBA e CAB. O conjunto de permutações de N símbolos é gerado tomando-se cada símbolo por vez e prefixando-o a todas as permutações que resultam dos $(N - 1)$ símbolos restantes. Conseqüentemente, permutações num conjunto de símbolos podem ser especificadas em termos de permutações num conjunto menor de símbolos. Formule um algoritmo recursivo para este problema.

```
algoritmo
        declare num, i numérico
        leia num
        declare str[num] literal
        leia str
        permut(str, 0)
fim algoritmo

subrotina permut(str literal, k numérico)
        declare i, tam numérico
        tam <- tamanho(str)
        se k = tam então
                para  i de 0 até num faça
                                imprima str[i]
                fim para
        senão
                para i de k até tam -1 faça
                        troca(str, k, i)
                        permut(str, k+1)
                        troca(str, i, k)
                fim para
        fim se
fim-subrotina

subrotina troca(str literal, a, b numérico)
        declare aux literal
        aux <- str[a]
        str[a] <- str[b]
        str[b] = aux
fim-subrotina
```

8) Considere uma partida de futebol entre duas equipes A x B, cujo placar final é m x n, em que m e n são os números de gols marcados por A e B, respectivamente. Implemente um algoritmo recursivo que imprima todas as possíveis sucessões de gols marcados. Por exemplo, para um resultado de 3 x 1 as possíveis sucessões de gols são "A A A B", "A A B A", "A B A A" e "B A A A".

```
algoritmo
    declare golsT1, golsT2, i, j numérico
    leia golsT1, golsT2
    imprima calcula(golsT1, golsT2)
fim-algoritmo

subrotina calcula(golsT1, golsT2 numérico)
    declare totalGols
    totalGols <- golsT1+golsT2
    se totalGols = 2
        imprima AB, BA
    senão
        se golsT1 > 1
            imprima A
                para i de 0 a golsT1 faça
                    calcula(1, golsT2-1)
                fim-para
        senão se golsT2 > 1
            imprima B
                para i de 0 a golsT2 faça
                    calcula(golsT1-1, 1)
                fim-para
        fim-se
    fim-se
fim-subrotina
```