

Funções

- Funções constituem ferramenta essencial para a modularização de código.
- Vantagens:
 - Permitem reaproveitamento de código.
 - Permitem divisão de tarefas.
 - Tornam código mais legível.

Programa: número de combinações

- Faça um programa em Scilab que:
 - leia 2 inteiros n e k ;
 - calcule e apresente o número de combinações de n por k , dado pela fórmula:

$$\binom{n}{k} = \frac{n!}{(n - k)! k!}$$

Programa: número de combinações

- Uma das formas de se calcular o fatorial de um número inteiro positivo qualquer é:

```
fat = 1;  
for i = 2:n  
    fat = fat * i;  
end
```

- No caso, o código deve ser adaptado no intuito de se calcular os fatoriais de **n**, **n-k** e **k**.

Programa: número de combinações

```
n = input("n= "); k = input("k= ");
fat_n = 1; // Cálculo do fatorial de n
for i = 2:n
    fat_n = fat_n * i;
end
fat_n_k = 1; // Cálculo do fatorial de n-k
for i = 2:(n-k)
    fat_n_k = fat_n_k * i;
end
fat_k = 1; // Cálculo do fatorial de k
for i = 2:k
    fat_k = fat_k * i;
end
nComb = fat_n / (fat_n_k * fat_k);
printf ("Resultado = %g", nComb);
```

Programa: número de combinações

Função:

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction
```

Cabeçalho
da função

Corpo da
função

Programa principal:

```
n = input("n= "); k = input("k= ");
nComb = fatorial(n) / (fatorial(n-k) * fatorial(k));
printf ("Resultado = %g", nComb);
```

Chamada da função

Parâmetros formais de uma função

Parâmetro formal de saída,
cujo valor é calculado e
retornado pela função

Parâmetro formal de entrada,
cujo valor deve ser fornecido
na chamada da função

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction
```

Parâmetros formais e reais de uma função

Uma função pode ter mais de um parâmetro formal de saída

Uma função pode ter mais de um parâmetro formal de entrada

```
function [r1, r2] = eq2g(a, b, c)
    delta = b^2 - 4*a*c;
    r1 = (-b + sqrt(delta)) / (2*a);
    r2 = (-b - sqrt(delta)) / (2*a);
endfunction
```

Parâmetros reais de saída

Chamada da função `eq2g` :

```
[raiz1, raiz2] = eq2g(x, y, z);
```

Parâmetros reais de entrada

Parâmetros formais e reais de uma função

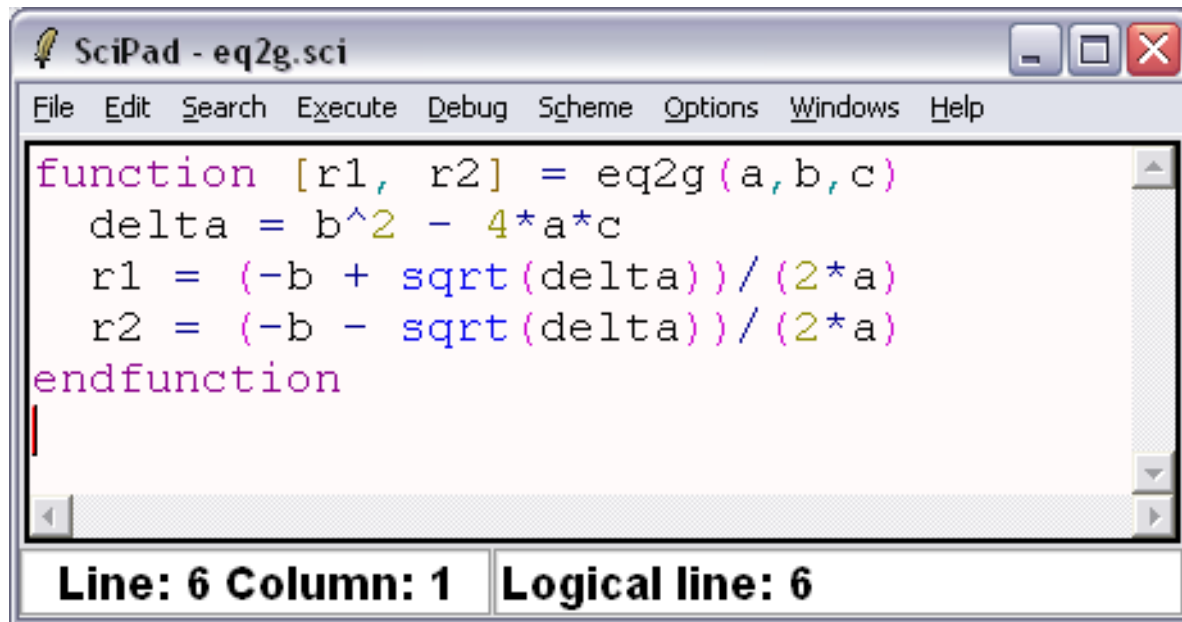
- Os parâmetros formais de entrada recebem os valores dos parâmetros reais de entrada.
 - Assim, o controle é transferido para a função, que trabalha sobre os parâmetros formais.

- Alterações feitas pela função sobre os parâmetros formais de entrada não afetam os parâmetros reais correspondentes.
 - Assim, variáveis criadas pela função não se misturam com variáveis de mesmo nome existentes no programa que chama a função.

- Os parâmetros reais de saída recebem os valores dos parâmetros formais de saída calculados pela função.
 - Assim, o controle é devolvido para o ponto de chamada.

Arquivo com uma função

- Uma função é escrita, normalmente, em um arquivo
 - com o mesmo nome da função;
 - com a extensão **.sci** (um programa tem a extensão **.sce**).



The image shows a screenshot of the SciPad editor window. The title bar reads "SciPad - eq2g.sci". The menu bar includes "File", "Edit", "Search", "Execute", "Debug", "Scheme", "Options", "Windows", and "Help". The main text area contains the following code:

```
function [r1, r2] = eq2g(a,b,c)
    delta = b^2 - 4*a*c
    r1 = (-b + sqrt(delta))/(2*a)
    r2 = (-b - sqrt(delta))/(2*a)
endfunction
```

At the bottom of the window, the status bar indicates "Line: 6 Column: 1 Logical line: 6".

- Para utilizar uma função em um programa Scilab, use **exec(<arquivo com a função>)** em tal programa.

Comando `exec`

```
exec("eq2g.sci")  
[raiz1,raiz2] = eq2g(x,y,z);
```

- Um programa Scilab só reconhece a existência de uma função criada pelo programador por meio do comando `exec`.
- O arquivo ".sci" com a função deve estar no mesmo diretório do programa que chama a função.

Programa principal e funções em um único arquivo

- Uma outra forma de se trabalhar com funções é construir um único arquivo onde funções precedem o programa principal.
 - Solução mais simples, porém dificulta reaproveitamento e manutenção.

```
function fat = fatorial(n)
    fat = 1;
    for i = 2:n
        fat = fat * i;
    end
endfunction

n = input("n= "); k = input("k= ");
nComb = fatorial(n) / (fatorial(n-k) * fatorial(k));
printf ("Resultado = %g", nComb);
```

Encadeamento de chamadas

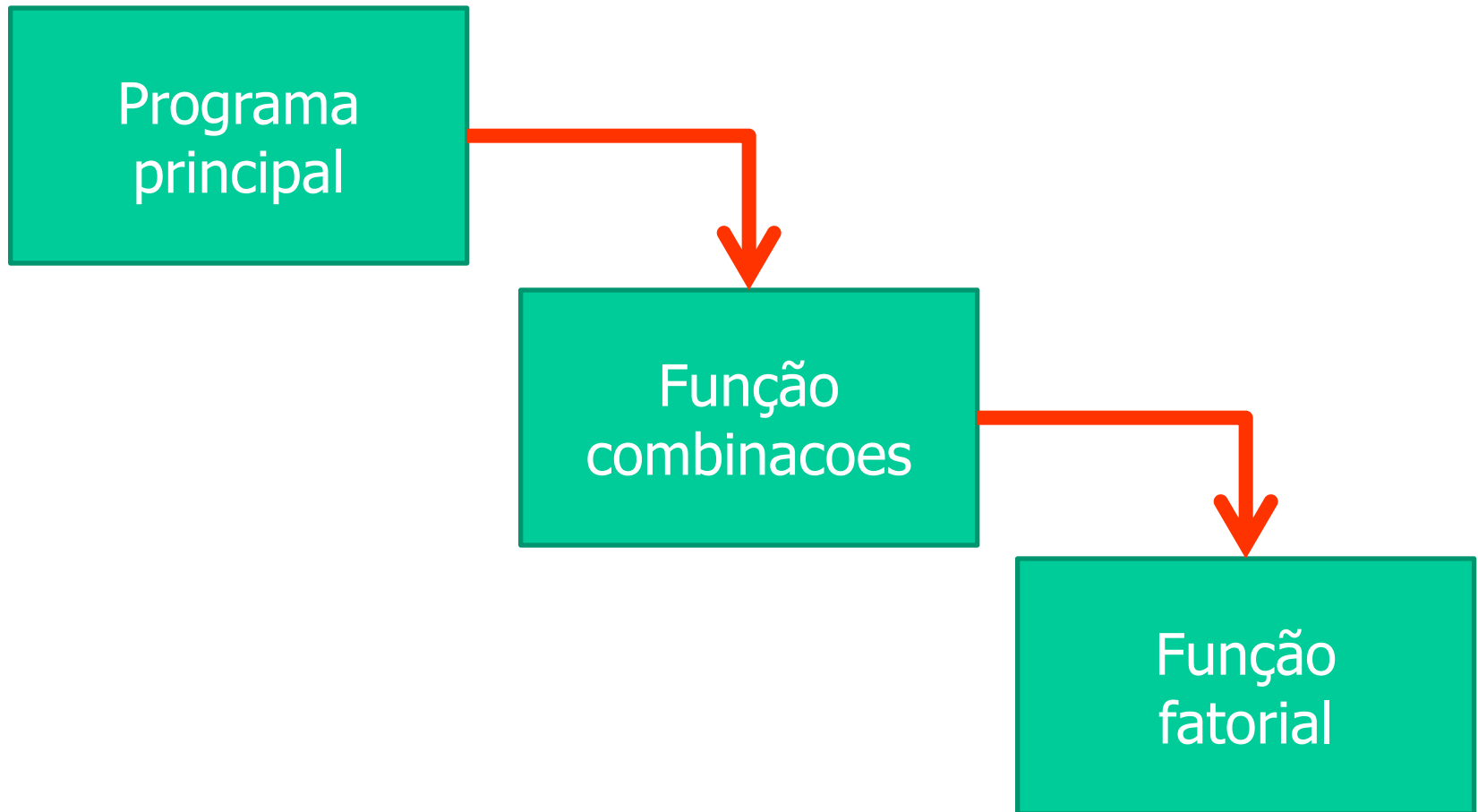
```
function nComb = combinacoes(n,k)
    nComb = fatorial(n)/...
        (fatorial(n-k) * fatorial(k));
endfunction
```

O programa que calcula o número de comparações transformado em função

Programa principal:

```
exec("combinacoes.sci");
exec("fatorial.sci");
n = input("n= "); k = input("k= ");
printf("nComb(%d,%d) = %d",n,k,combinacoes(n,k));
```

Encadeamento de chamadas



Função: soma dos elementos de um vetor

- Faça uma função para calcular a soma dos elementos de um vetor de valores numéricos.
 - Dados de entrada: um vetor (**parâmetro de entrada**).
 - Dados de saída: soma dos elementos do vetor (**parâmetro de saída**).

```
function s = soma(A)
    // Calcula a soma dos
    // elementos do vetor A
    s = 0;
    for k = 1:length(A)
        s = s + A(k);
    end
endfunction
```

Função que retorna o número de elementos de um vetor passado como parâmetro

Programa principal: teste da função **soma**

```
exec ("soma.sci") ;  
a = int(10*rand(1,4))  
sa = soma(a) ;  
printf("\n Soma = %g\n\n", sa) ;  
  
b = int(10*rand(1,6))  
sb = soma(b) ;  
printf("\n Soma = %g\n\n", sb) ;  
  
c = int(10*rand(1,9))  
sc = soma(c) ;  
printf("\n Soma = %g\n\n", sc) ;
```

A falta do ";" faz com que o vetor seja impresso

Execução do programa principal

a =
3. 3. 2. 5.
Soma = 13

b =
4. 3. 5. 5. 4. 2.
Soma = 23

c =
6. 4. 9. 0. 4. 2. 4. 2. 1.
Soma = 32

Função: menor elemento de um vetor

- Faça uma função para retornar o menor elemento de um vetor de valores numéricos.
 - Dados de entrada: um vetor (**parâmetro de entrada**).
 - Dados de saída: menor elemento do vetor (**parâmetro de saída**).

```
function m = menor(A)
// Encontra o menor elemento do vetor A
m = A(1);
for k = 2:length(A)
    if (A(k) < m) then
        m = A(k);
    end
end
endfunction
```

Programa principal: teste da função **menor**

```
exec ("menor.sci") ;  
  
a = int(10*rand(1,4)) ;  
ma = menor(a) ;  
printf("\n Menor = %g\n\n", ma) ;  
  
b = int(10*rand(1,6)) ;  
mb = menor(b) ;  
printf("\n Menor = %g\n\n", mb) ;  
  
c = int(10*rand(1,9)) ;  
mc = menor(c) ;  
printf("\n Menor = %g\n\n", mc) ;
```

Função: número primo

- Faça uma função para retornar se um determinado número inteiro maior que 1 é primo ou não.
 - Dados de entrada: um número (**parâmetro de entrada**).
 - Dados de saída: valor lógico (**parâmetro de saída**).

```
function primo = ehPrimo(n)
    d = 2;
    while modulo(n,d) ~= 0
        d = d + 1;
    end
    primo = (d == n);
endfunction
```

Programa principal: teste da função **ehPrimo**

```
exec ("ehPrimo.sci");  
  
for i = 2:100  
    if (ehPrimo(i)) then  
        printf ("%g é primo\n", i);  
    end  
end
```

Recursividade

- Sabe-se que uma função pode chamar outra função
 - que pode chamar outra função,
 - que pode chamar outra função,
 - e assim sucessivamente ...

- Uma função também pode chamar a si própria.
 - Nesta caso, a função é dita recursiva.

- Pode-se criar uma função recursiva para se resolver um determinado problema quando a definição de tal problema baseia-se nele próprio.

Função: fatorial recursivo

- Uma definição recursiva formal de fatorial é:
 - $1! = 1$; e
 - $n! = n \times (n-1)!$, para $n > 1$.

```
function fat = fatorialR(n)
    if n == 1 then
        fat = 1
    else
        fat = n * fatorialR(n-1)
    end
endfunction
```

Programa principal: teste da função **fatorialR**

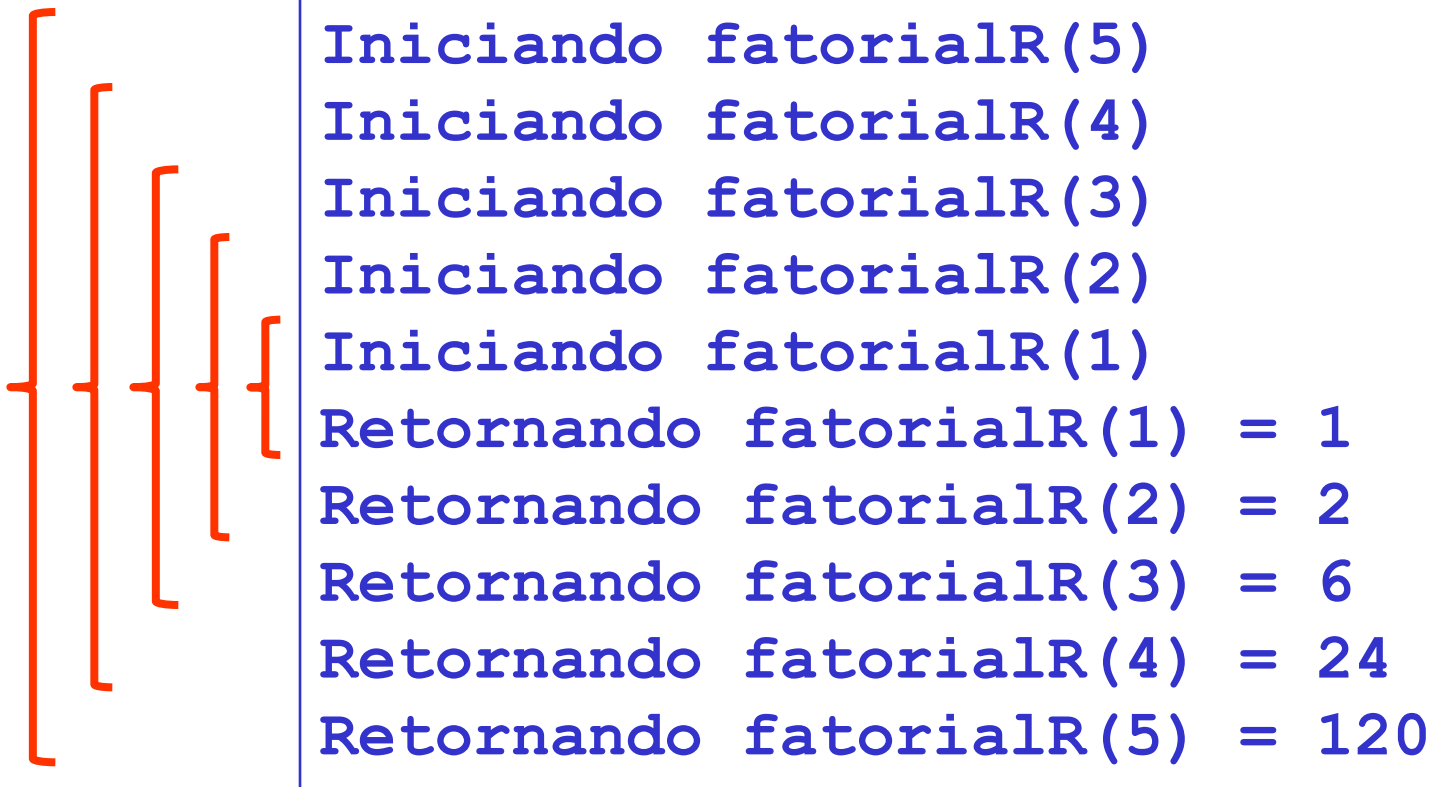
```
exec("fatorialR.sci");  
  
n = input("n = ");  
  
while n > 0 do  
    printf("%d! = %d\n",n,fatorialR(n));  
    n = input("n = ");  
end
```

Função: fatorial recursivo com mensagens

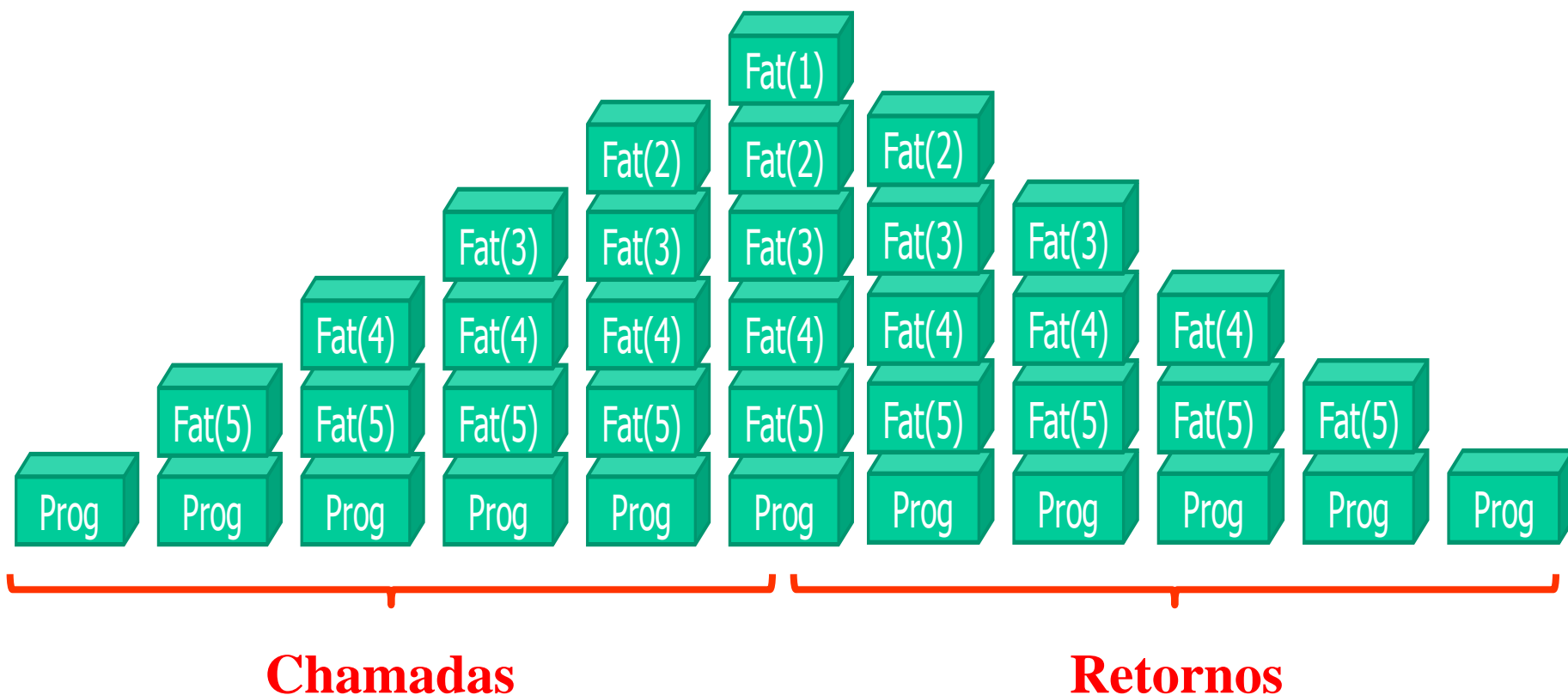
```
function fat = fatorialR(n)
    printf("\nIniciando fatorialR(%d)",n);
    if n == 1 then
        fat = 1
    else
        fat = n * fatorialR(n-1)
    end
    printf("\nRetornando fatorialR(%d) = %d",n,fat)
endfunction
```


Função: fatorial recursivo com mensagens

■ Execução de `fatorialR(5)`:



Pilha de execução - chamadas e retornos



Função recursiva: menor elemento de um vetor

- É possível formular o algoritmo de descoberta do menor elemento em um vetor como uma função recursiva.
- Uma possibilidade é:
 - Se $\text{length}(\mathbf{A}) == 1$, o menor valor em \mathbf{A} é $\mathbf{A}(1)$.
 - Se $\text{length}(\mathbf{A}) > 1$, o menor valor em \mathbf{A} é o menor dentre (o menor valor na metade esquerda de \mathbf{A}) e (o menor valor na metade direita de \mathbf{A}).

Função recursiva: menor elemento de um vetor

```
function m = menorR(A)
    if (length(A) == 1) then
        m = A(1);
    else
        metade = int(length(A)/2);
        menorEsq = menorR(A(1:metade));
        menorDir = menorR(A(metade+1:length(A)));
        if (menorEsq <= menorDir) then
            m = menorEsq;
        else
            m = menorDir;
        end
    end
endfunction
```