- **Declaramos um Parâmetro da Classe Variável, quando pretendemos que a Chamada do Procedimento modifique o seu valor no Programa Principal;**
- **Declaramos um Parâmetro da Classe Valor, quando queremos evitar que a Chamada do Procedimento afecte o seu valor no Programa Principal.**

Mais um exemplo:

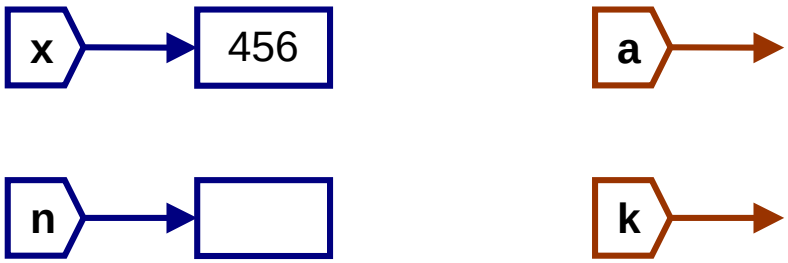
```
procedure ContarDigitos(a : integer; var k : integer);

          begin k:= 0;
                while a <> 0 do
                      begin k:= k + 1;
                            a:= a div 10
                      end
          end (* ContarDigitos *);

begin   (* Programa Principal, ou Procedimento Global *)
        ...
        (* Aqui x = 456 *)
        ContarDigitos(x, n);
        (* Aqui x = 456 e n = 3 *)
        ...
        ContarDigitos(n * (100 + 2 * n), j);
        (* Aqui j = 3 *)
        ...
end.
```

- **O valor do Parâmetro Actual** x = 456 **não foi afectado pela contagem dos seus dígitos;**
- **O Parâmetro Actual** n **foi usado para passar o resultado dessa contagem para o Programa;**
- **Um Parâmetro Actual, quando Declarado da Classe Valor, pode ser o valor de uma expressão, tal como** n * (100 + 2 * n)**.**

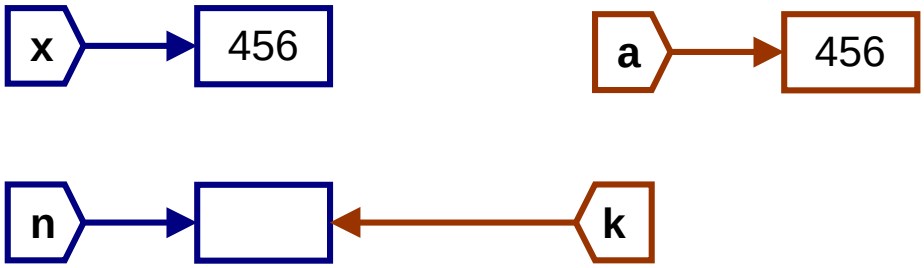
Mas qual é a diferença?

Análise de uma Chamada do Procedimento ContarDigitos:

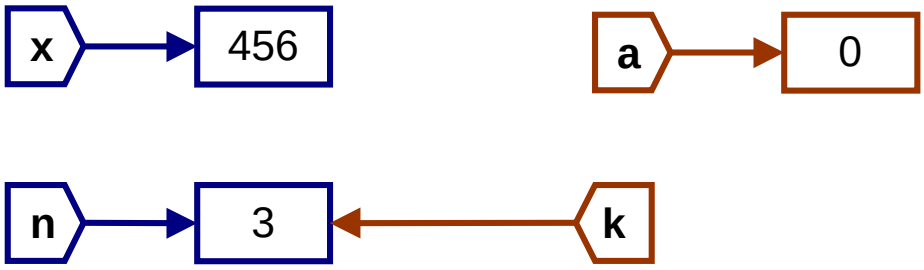
Antes da Chamada:



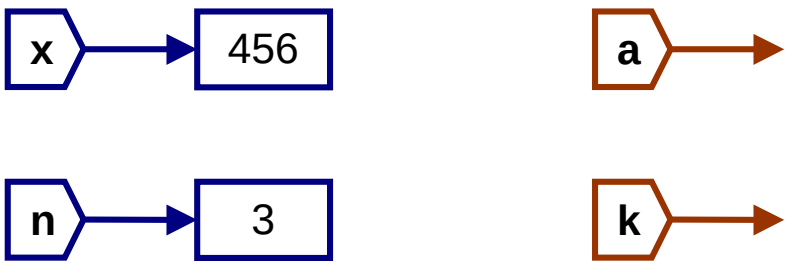
Início do Procedimento:



Fim do Procedimento:



Depois da Chamada:



Que acontece?

- Se o Parâmetro for da Classe Valor é gerada uma cópia temporária, que é destruída após a execução do Procedimento;
- Se o Parâmetro for da Classe Variável, a ligação entre o Parâmetro Formal e o Parâmetro Actual é feita apenas por referência.

Por isso:

- Se um Parâmetro Formal for da Classe Valor, o correspondente Parâmetro Actual pode ser uma expressão, cujo Valor é calculado no momento da Chamada;
  Para que esse Valor possa ser calculado, é necessário que todos os elementos da expressão estejam já definidos (i.e. todas as variáveis que ocorrem tenham valores atribuídos).

- Se um Parâmetro Formal for da Classe Variável, o correspondente Parâmetro Actual tem necessariamente de ser uma Variável.
  Essa Variável não precisa de estar já definida (i.e. pode não ter valor atribuído antes da chamada e, muitas vezes, não tem).

Correspondência entre a Lista de Parâmetros Formais e a Lista de Parâmetros Actuais

Regras:

- O Número de Parâmetros nas duas Listas deve ser o mesmo.
- Cada Parâmetro Actual corresponde ao Parâmetro Formal que ocupa a mesma posição na Lista.
- Parâmetros Formais e Actuais correspondentes devem ser da mesma Classe e do mesmo Tipo.

Problema:

Somar Fracções: a/b ← a/b + c/d

```
(* Asumindo type intnneg = 0 .. maxint; *)

procedure SomarFraccoes(var a, b : intnneg; c, d : intnneg);

          begin a:= a*d + b*c;
                b:= b*d                    (* Por esta ordem! *)
          end (* SomarFraccoes *);
```

Simplificar a Fracção: a/b

```
procedure SimplificarFraccao(var a, b : intnneg);
          var aa, bb, resto : intnneg;

          begin (* Calcular mdc(a, b) *)
                (* Cópias de a e de b *)
                aa:= a;
                bb:= b;
                while bb <> 0 do
                      begin resto:= aa mod bb;
                            aa:= bb;
                            bb:= resto
                      end;
                (* aa = mdc(a, b) *)
                if aa > 1
                then begin (* Só aqui os valores dos
                                        parâmetros são alterados *)
                           a:= a div aa;
                           b:= b div aa
                     end
          end (*SimplificarFraccao *);
```

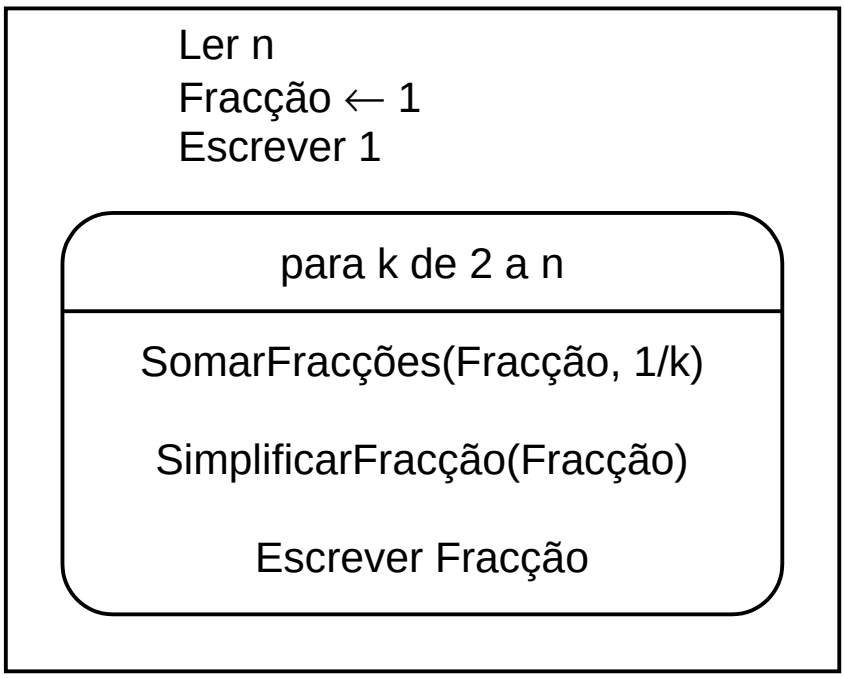
Problema: **Calcular as n primeiras Somas Parciais da Série Harmónica, em Aritmética de Racionais.**

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

p.e., para n = 6, o programa deve escrever:

n	H(n)
1	1
2	3/2
3	11/6
4	25/12
5	137/60
6	49/20

Algoritmo:



Exercício: **Escrever o programa completo.**

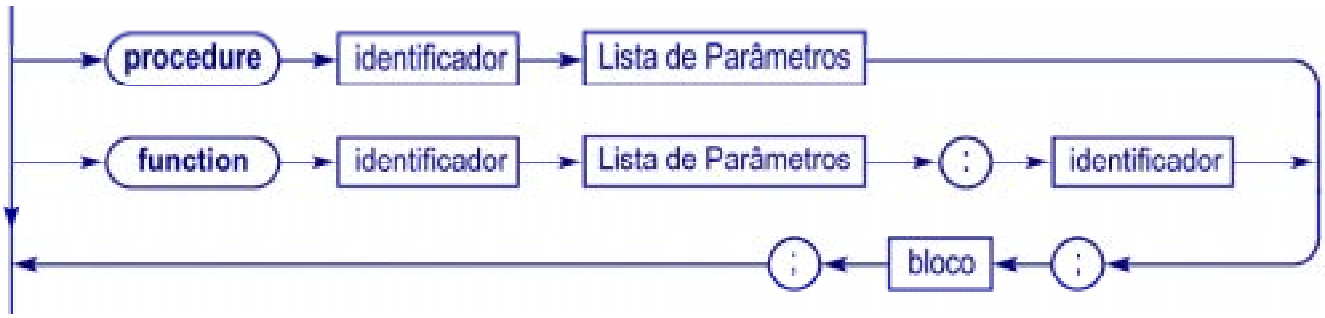
5.4. Funções

- **Uma Função é um Subprograma que calcula e fornece um valor (escalar), para ser utilizado como um Factor de uma Expressão.**

Diagramas de sintaxe:

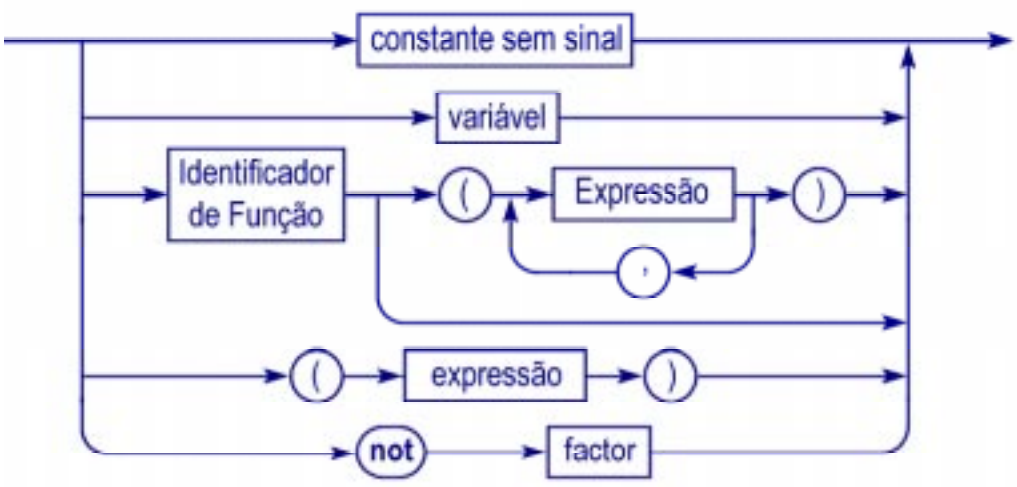
Declaração de Função

(Bloco)

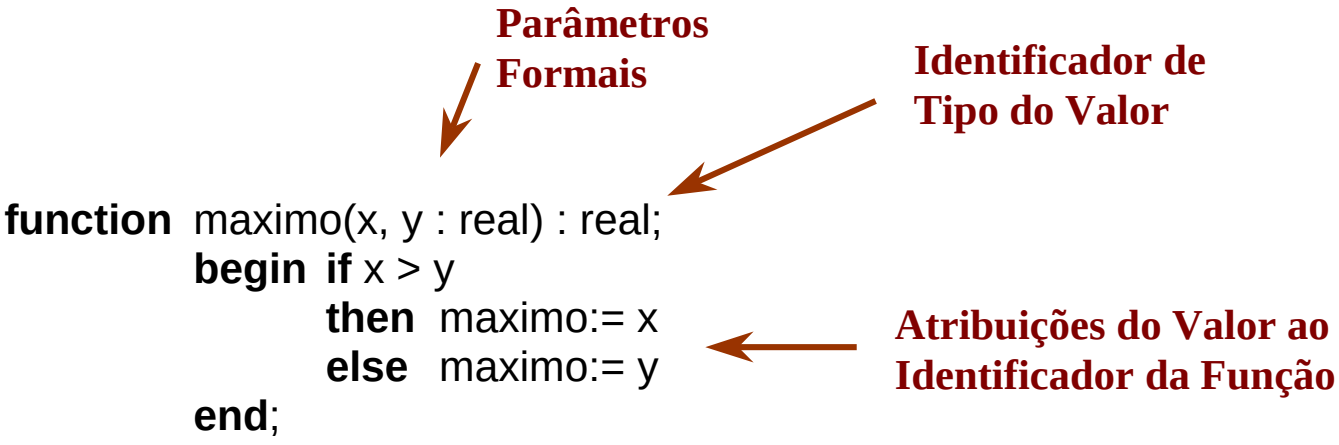


Chamada de Função

(Expressão: Expressão Simples: Termo: Factor)



Exemplo:




```
function  maximo(x, y : real) : real;
          begin  if x > y
                 then  maximo:= x
                 else  maximo:= y
          end;

...

begin   (* Programa Principal, ou Procedimento Global *)
        ...
        m:= 2 * maximo(a, b);
        ...
        z:= maximo( sin(x), cos(x));
        ...
        w:= sqrt(abs(maximo(m, n)));
        ...
        a:= maximo(maximo(m, z), w);
        ...
end
```

Chamadas da Função maximo **e de Funções pré-definidas.**

O Identificador da Função ocorre:

1. **No Cabeçalho da Declaração da Função;**
2. **No Bloco da Função: no Lado Esquerdo de Instruções de Atribuição (sem os Parâmetros);**
3. **No Programa Principal (ou em Procedimentos Globais): no Lado Direito de Instruções de Atribuição (com os Parâmetros).**

Em princípio:

- **As Regras referentes aos Domínios dos Indentificadores são as mesmas que as dos Procedimentos;**
- **As Regras referentes à Correspondência e aos modos de Ligação entre os Parâmetros Formais e Actuais são as mesmas que as dos Procedimentos.**

Mas!:

- **As Funções não devem usar Parâmetros da Classe Variável;**
- **As Funções não devem conter Instruções de Entrada/Saída;**
- **As Funções não devem alterar o conteúdo de Variáveis Globais.**

As Funções só devem ser usadas no sentido Matemático do termo

Exemplo:

```
function mdc(x, y : intnneg) : intnneg;
         var resto: intnneg;
         begin while y <> 0 do
                        begin resto:= x mod y;
                              x:= y;
                              y:= resto
                        end;
               mdc:= x
         end;
```

- **Não são necessárias cópias (explícitas) de** x **e de** y **pois, sendo os Parâmetros Formais da Classe Valor, as alterações não afectam os valores dos Parâmetros Actuais.**

```
procedure SimplificarFraccao(var a, b : intnneg);
          var divisor : intnneg;

          function mdc(x, y : intnneg) : intnneg;
                   var resto: intnneg;
                   begin while y <> 0 do
                                 begin resto:= x mod y;
                                       x:= y;
                                       y:= resto
                                 end;
                         mdc:= x
                   end (* mdc *);

          begin divisor:= mdc(a, b);
                if divisor > 1
                then begin a:= a div divisor;
                           b:= b div divisor
                     end
          end (* SimplificarFraccao *);
```

Porque foi utilizada a variável divisor **?**
Uma versão apararentemente análoga seria:

```
          begin if mdc(a, b) > 1
                then begin a:= a div mdc(a, b);
                           b:= b div mdc(a, b)
                     end
          end (* SimplificarFraccao *);
```

Contudo:

1. **A mesma Função teria sido calculada 3 vezes (ineficiência).**
2. **Neste caso particular, o resultado estaria efectivamente errado, pois a chamada** a:= a **div** mdc(a, b) **altera o valor da variável** a **que, por sua vez, serve de argumento à segunda chamada** b:= b **div** mdc(a, b)**.**

- **Por vezes, torna-se útil a utilização de Funções com Resultado do Tipo Lógico (**boolean**):**

Exemplo1:

```
function perfeito(numero : intnneg) : boolean;
var   soma, n : intnneg;

begin  soma:= 1;
          for  n:=2 to (numero div 2) do
               if numero mod n = 0
               then soma:= soma + n;
          perfeito:= (soma = numero)
end(* perfeito *);
```

Exercício: function SomaDivisores(numero : intnneg) : intnneg;

Exemplo2:

```
function primo(numero : intnneg) : boolean;
var   limite, n : intnneg;
        p : boolean;

begin  limite:= trunc(sqrt(numero));
          p:= true;
          n:= 2;
          while  p and (n <= limite) do
                    begin  p:= (numero mod n) <> 0;
                              n:= n+1
                    end;
          primo:= p
end(* primo *);
```

- **Foi necessária a utilização da variável auxiliar (**p : boolean**) pois a Expressão Lógica (**primo **and** (n <= limite)**) iria incluir uma Chamada da própria Função** primo**, sem Parâmetros, o que provocaria um ERRO.**

Exemplo: Função para, dado um inteiro não negativo, calcular e fornecer o número obtido por inversão da ordem dos seus dígitos.

```
function reverso(numero : intnneg) : intnneg;
var   inv, digito : intnneg;
begin  inv:= 0;
        while  numero > 0  do
                begin  digito:= numero mod 10;
                        inv:= inv * 10 + digito;
                        numero:= numero div 10
                end;
        reverso:= inv
end(* reverso *);
```

Exemplo de utilização:

```
capicua:= (numero = reverso(numero));
```

- **Não foi necessária uma cópia explícita de** numero**;**
- **Foi necessária a variável local auxiliar** inv**, pois a Instrução de Atribuição** reverso:= reverso * 10 + digito; **iria incluir uma tentativa ERRADA de Chamada da Função** reverso **(lado direito).**

Mais um Exemplo:

```
function factorial(n : intnneg) : intnneg;
var   i, fa : intnneg;

begin  fa:= 1;
        for  i:=1 to n do
             fa:= fa * i;
        factorial:= fa
end(* factorial *);
```