Capítulo 0 : Conceitos Fundamentais

❒ Algoritmos (uma abordagem estruturada)

Um Algoritmo consiste numa Estruturação de Acções Elementares sobre os Dados

Acções Elementares { Atribuição; Chamada de Procedimento

• Atribuição

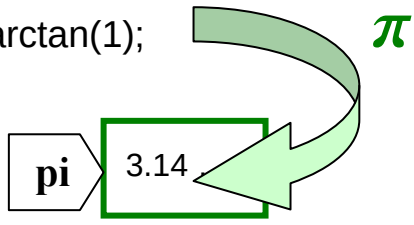
A Declaração de uma Variável:
Cria o Identificador;
Cria uma Célula de Memória com o Formado pretendido;
Associa o Identificador ao Endereço de Memória dessa Célula.

"Variável toma o valor de Expressão"!

A Variável foi Declarada.

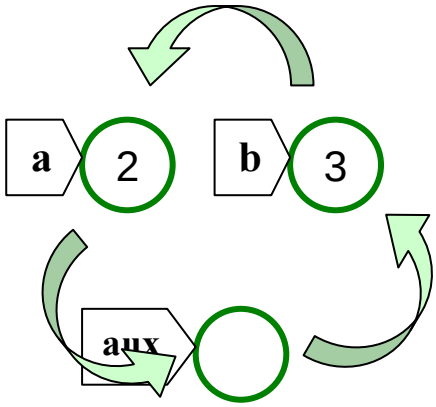
A Expressão é calculada e o seu valor final é registado na Célula de Memória associada à Variável.

```
var pi : real;
…
pi := 4* arctan(1);
```



Exemplo: Trocar os Valores de duas Variáveis

```
var a, b, aux : integer;
…

(* aqui a=2 e b=3 *)

aux := a:
a := b;
b := aux;

(* aqui a=3 e b=2 *)
```



- **Chamada de Procedimento**

Declaração de um Procedimento:

```
procedure trocar (var a, b : integer);
var aux : integer;
begin  aux := a;
       a := b;
       b := aux
end;
```

Após declarado o Procedimento, a sua Chamada no Programa Principal ou noutro sub-programa torna-se uma Acção Elementar:

```
...
read(x, y);
trocar(x, y);
writeln(x, y):
...
```

procedimentos implícitos

Estruturas de Base
- Sequência
- Decisão
- Repetição
- Parentetização

- **Sequência**

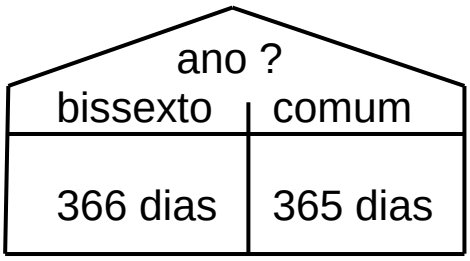
ler raio
calcular perímetro
escrever perímetro

separador

```
...
read(raio);
perimetro := 2 * pi * raio;
writeln(perimetro);
...
```

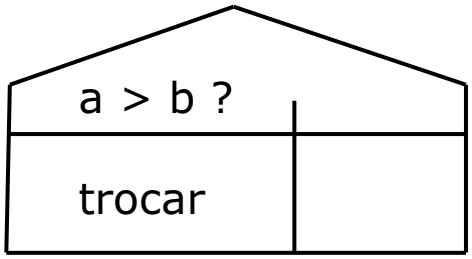
- **Decisão**

(**Alternativa:** forma completa)




```
...
if bissexto
then dias := 366
else dias := 365;
...
```

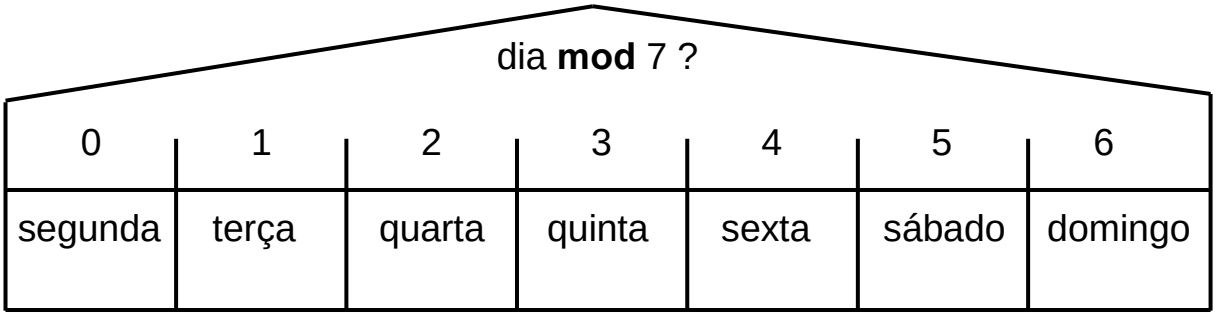
(**Alternativa:** forma parcial)




```
...
if a > b
then  begin  aux := a;
             a := b;
             b := aux
      end ;
...
```

- **Selecção múltipla**

Se o dia 1 de Outubro de 2002 foi uma Terça-Feira, como calcular os restantes?




```
...
case dia mod 7 of
        0 : writeln('Segunda Feira');
        1 : writeln('Terca Feira');
        2 : writeln('Quarta Feira');
        3 : writeln('Quinta Feira');
        4 : writeln('Sexta Feira');
        5 : writeln('Sabado');
        6 : writeln('Domingo')
end;
...
```

... e se o valor de dia $\notin$ [1..31]? (**validação do dados**)

```
...
if (dia >= 1) and (dia <= 31)
then   case  dia mod 7 of
                    0 : writeln('Segunda Feira');
                    1 : writeln('Terca Feira');
                    2 : writeln('Quarta Feira');
                    3 : writeln('Quinta Feira');
                    4 : writeln('Sexta Feira');
                    5 : writeln('Sabado');
                    6 : writeln('Domingo')
            end
else writeln(dia, 'Não é valor de dia de Outubro');
...
```

Problema: Calcular a data do dia seguinte a um dado dia.

Exemplo: O dia seguinte a 31/12/2002 é 1/1/2003.

CONSTRUÇÃO DESCENDENTE:
Decomposição das Estruturas,
até ao nível das Acções Elementares.

Construção Descendente:

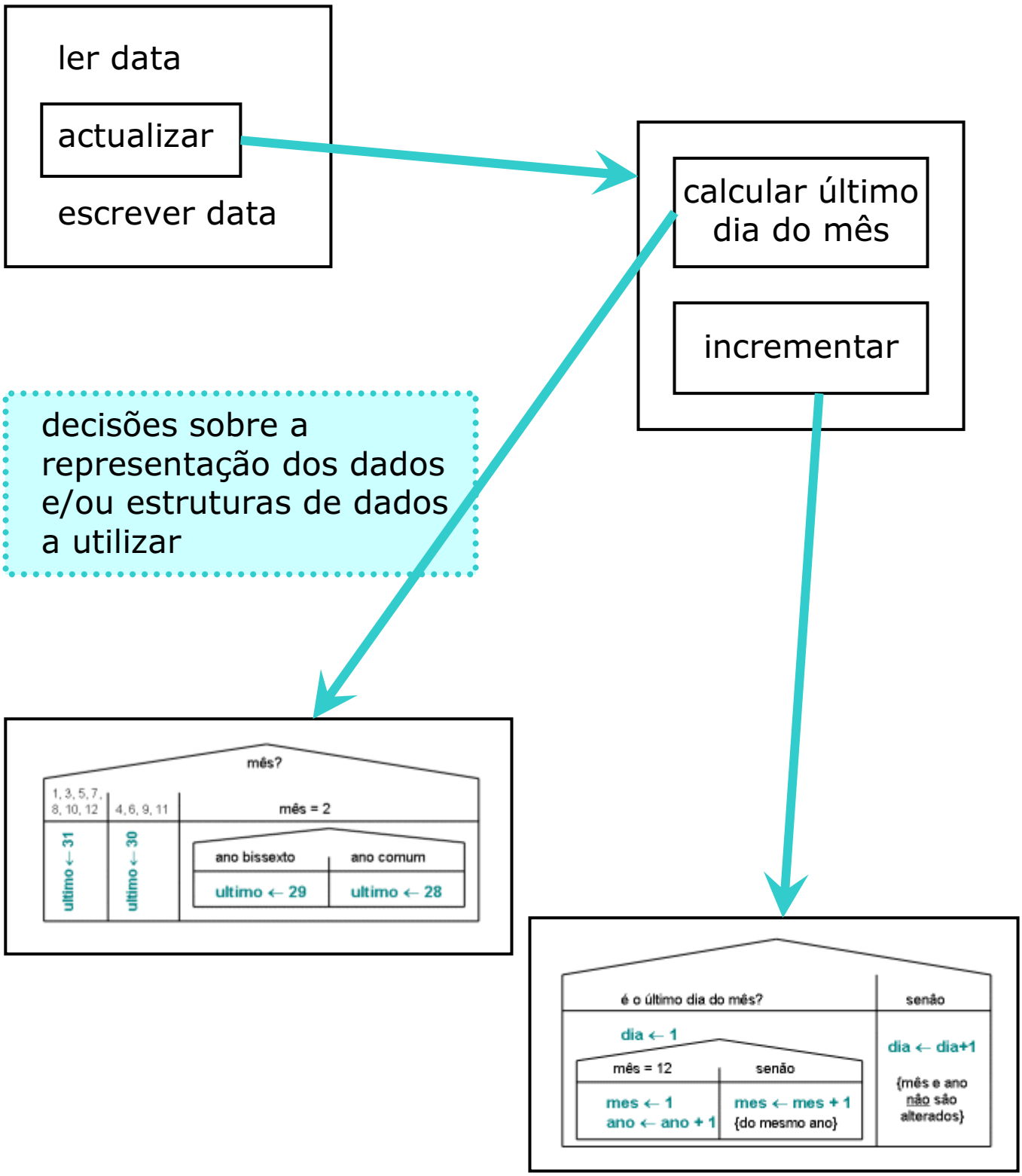
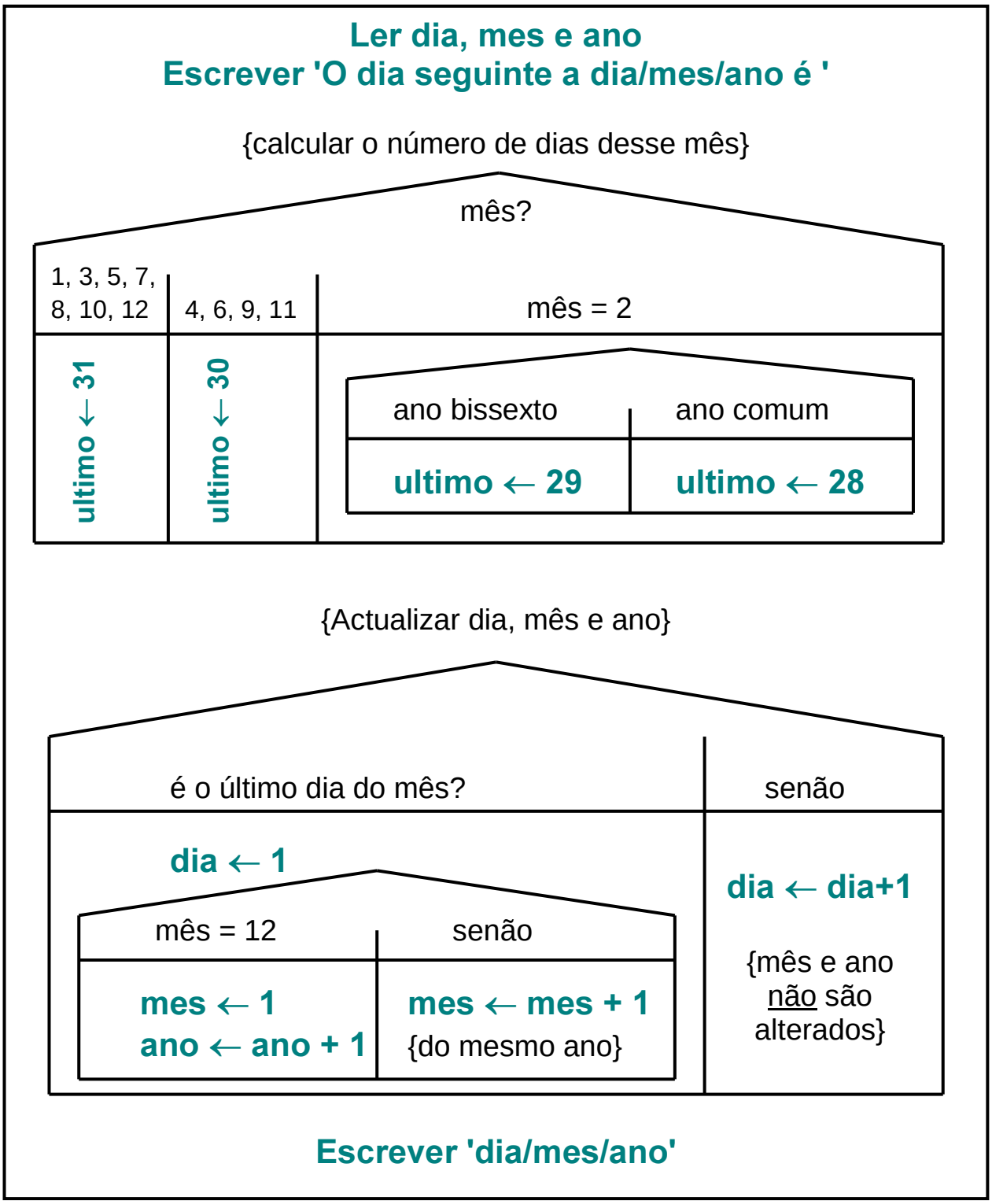


Diagrama de Estrutura:



Programa em Pascal:

```
program dia_seguinte(input, output);
var  dia: 1..31;
     mes: 1..12;
     ano: 1582..maxint;
     ultimo: 28..31;
     bissexto: boolean;

begin  writeln('Indique o dia, mes e ano da presente data');
       read(dia, mes, ano);
       write('Dia seguinte a ', dia:3, '/', mes:2, '/', ano:4, ' =');
       bissexto:= (ano mod 4 = 0) and (ano mod 100 <> 0)
                                          or (ano mod 400 = 0);

       (*Calculo do numero de dias do mes*)
       case mes of
            1, 3, 5, 7, 8, 10, 12 : ultimo := 31;
            4, 6, 9, 11 : ultimo := 30;
            2 :  if  bissexto
                 then  ultimo := 29
                 else  ultimo := 28
       end;

       (*Actualizacao de dia, mes e ano*)
       if  dia = ultimo
       then  begin  dia := 1;
                    if mes = 12
                    then  begin  mes := 1;
                                 ano := ano+1
                          end
                    else  mes := mes+1
             end
       else  dia := dia+1;

       writeln(dia:3, '/', mes:2, '/', ano:4)
end.
```

- A **Indentação** dever reflectir os sucessivos níveis de estrutura do algoritmo;
- Será adequada a utilização de **sub-domínios** para a **validação dos dados**?
- Como reage este programa para valores inválidos dos dados como, por exemplo, mes = 15 ?

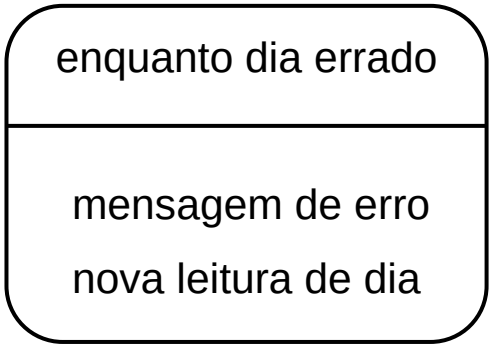
Exemplo de solução:

- Utilização de uma variável do tipo lógico; com estruturação das mensagens de erro.

```
...
var   dia, mes, ano: integer;
      bissexto, erro: boolean;

begin   erro := false;
        writeln('Indique o dia, mes e ano da presente data');
        read(dia);
        if (dia < 1) or (dia > 31)
        then erro := true;
        read(mes);
        if (mes < 1) or (mes > 12)
        then erro := true;
        ...
        if erro
        then   begin
                 write ('Valor errado nos dados:');
                 if (dia < 1) or (dia > 31)
                 then writeln (' dia fora de [1 .. 31]');
                 if (mes < 1) or (mes > 12)
                 then writeln (' mes fora de [1 .. 12]');
                 ...
               end
        else   begin
                 (*Dados válidos*)
                 ...
                 [Algoritmo Completo]
               end
end.
```

- **Repetição (ciclos)**

  (**Teste à entrada do Ciclo**)



- Um programa **robusto** deve permitir ao utilizador, várias tentativas ...

```
...
writeln('Escreva o dia de hoje');
read(dia);
while  (dia < 1) or (dia > 31) do
       begin  writeln('Valor de dia fora de [1 .. 31],
                                        por favor tente de novo');
              read(dia)
       end;
...
```

- ... e mesmo limitar o número de tentativas (t) :

```
...
read(dia);
t := 1;
erro := (dia < 1) or (dia > 31);
while  erro and (t < 10) do
       begin  writeln('Enganou-se, tente de novo');
              read(dia);
              t := t +1;
              erro := (dia < 1) or (dia > 31)
       end;
...
```

- Como **pára** este ciclo?
- Qual é o **Estado** do conjunto de variáveis envolvidas?

Se o Ciclo parou, então é **Falsa** a proposição lógica:

erro **and** (t < 10)

o que pode acontecer em três casos:

erro	t < 10	erro **and** (t < 10)	**ciclo**
F	F	F	parou
F	V	F	parou
V	F	F	parou
V	V	V	continua

apenas num desses três caso, a variável erro é Verdadeira.

Portanto, à saída do ciclo, podemos escrever:

```
if  erro
then  writeln('Enganou-se 10 vezes nos dados! Desisto!')
else  begin  (*Dados válidos*)
                ...
                [Algoritmo Completo]
      end;
...
```

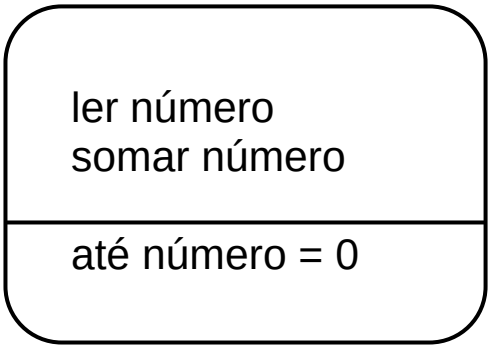
- Ou, de um modo mais formal:

not (erro **and** (t < 10)) =
not erro **or** (t ≥ 10) =

erro ⇒ (t ≥ 10)

- **Repetição (ciclos)**

  (**Teste à saída do Ciclo**)



Calcular a soma de vários números inteiros, dados pelo utilizador, um por linha. A última linha contem o número 0.

```
program somar(input, output);
var  numero, soma : integer;

begin  writeln('Escreva numeros inteiros, um por linha,
                                                terminando com zero.');
       soma:= 0;

       repeat  readln(numero);
               soma:=soma+numero;
       until   numero = 0;

       writeln('Soma = ', soma)
end.
```

- O Ciclo é executado pelo menos uma vez.
- O utilizador tem de escrever, pelo menos, o número zero.
- O zero é efectivamente somado.

- **Repetição (ciclos)**

  (Ciclo com Contador)

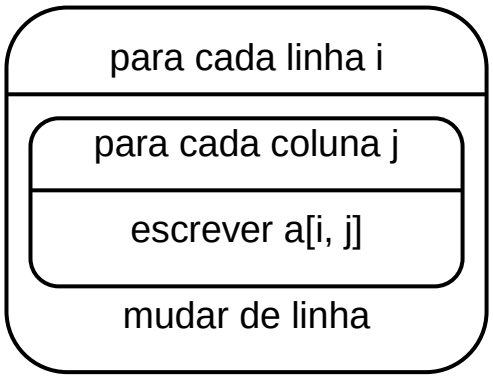
Somar os elementos de um vector x[1..n]

$$\sum_{k=1}^{n} x_k$$

```
soma:= 0;
for k:=1 to n do
     soma:= soma + x[k];

(* ou então *)
soma:= 0;
for k:=n downto 1 do
     soma:= soma + x[k];
```

Escrever uma matriz a[1..m, 1..n]

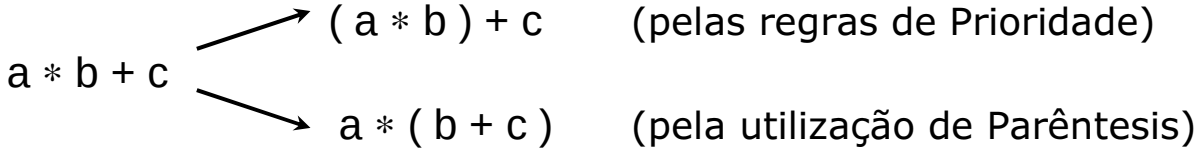
```
for i :=1 to m do
    begin  for j := 1 to n do
                write (a[i, j]:5:2);
           writeln
    end;
```



- A variável Contador e as duas Expressões são do mesmo tipo (escalares, não reais).
- Os valores das duas Expressões são calculados no início, não podendo ser alteradas durante a execução do Ciclo.
- O valor da Variável torna-se **indefinido** após a execução do Ciclo.

- **Parentetização** (*bracketing*)

De modo análogo ao caso das expressões,



existe uma possível ambiguidade na utilização encadeada de dois **if**s:

if cond1 **then if** cond2 **then** inst1 **else** inst2;

A qual dos dois **if**s pertence o **else**?

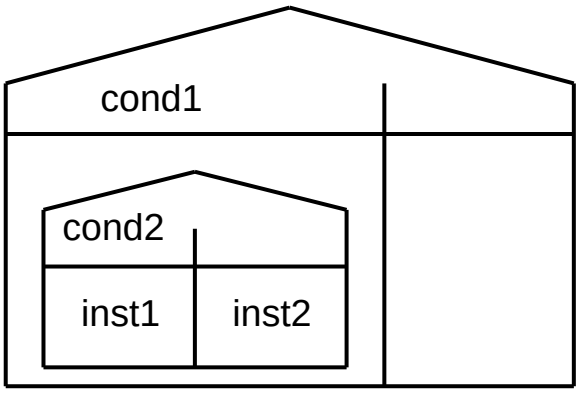
Por convenção, cada **else** corresponde ao último **if** indicado.

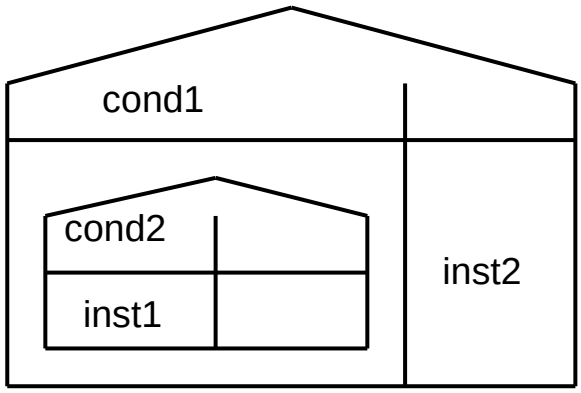
Pela Regra de Prioridade convencionada:

```
If cond1
then  if cond2
        then  inst1
        else  inst2;
```

Pela Utilização da Composição de Instruções:

```
If cond1
then  begin  if cond2
                then inst1
        end
else  inst2;
```



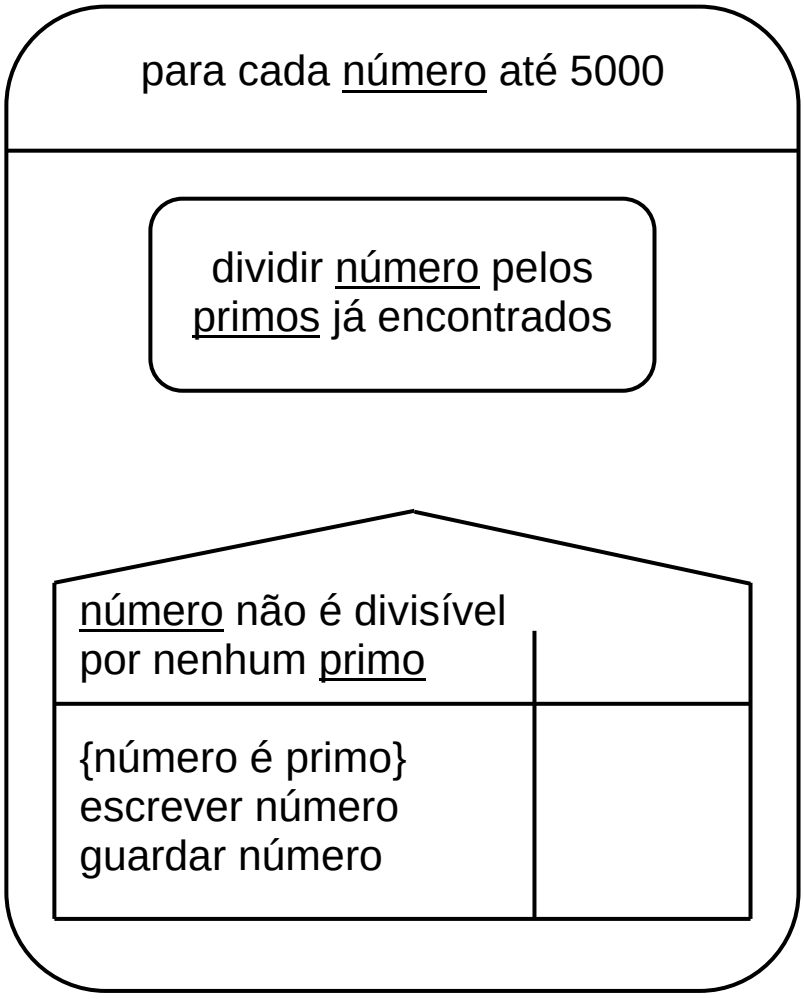


Problema: Listar todos os números primos até 5000

Estratégia:

> Dividir cada número até 5000
> por todos os primos já encontrados;
>
> Se um número não for divisível por nenhum primo,
> então também é primo.

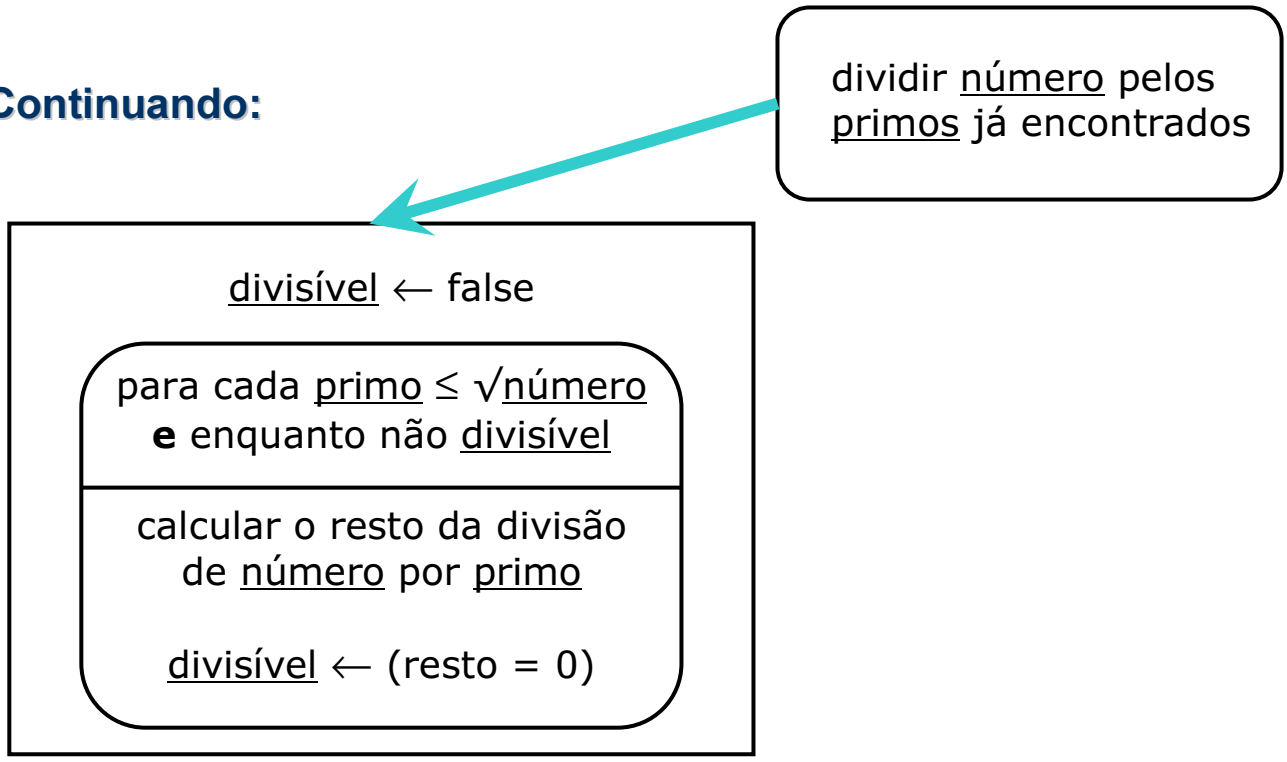
Abordagem Descendente:



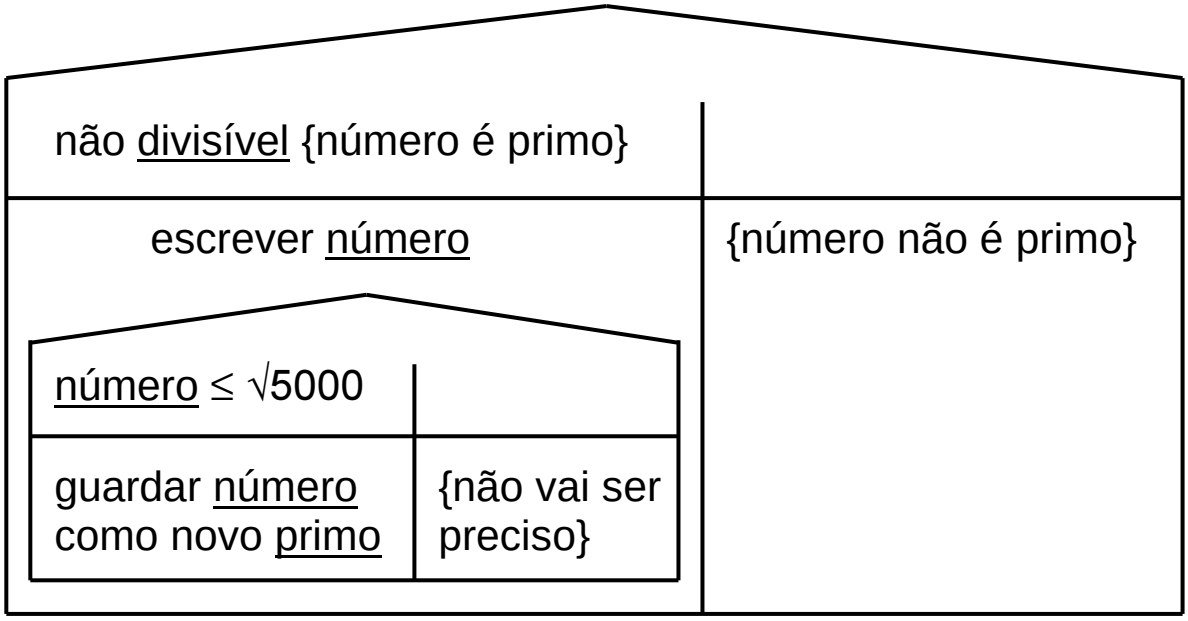
Um pouco de Teoria dos Números:

- O primeiro primo é 2;
- Depois do 2, só os ímpares podem ser primos;
- Se um número **não** tiver nenhum divisor menor ou igual à sua raiz quadrada, então é primo.

Continuando:



À saída deste ciclo, temos o valor final da variável lógica divisível:



Decisões sobre a Representação dos Dados:

A construção do Algoritmo foi Independente das Estruturas de Dados

Quantos primos será necessário guardar?

Aproximação de Legendre, para o número de primos até n:

$$\pi(n) \approx \frac{n}{\ln n - 1.08366}$$

portanto, para testar números até 5000, vão ser necessários $\pi(\sqrt{5000}) \approx 22$ primos.

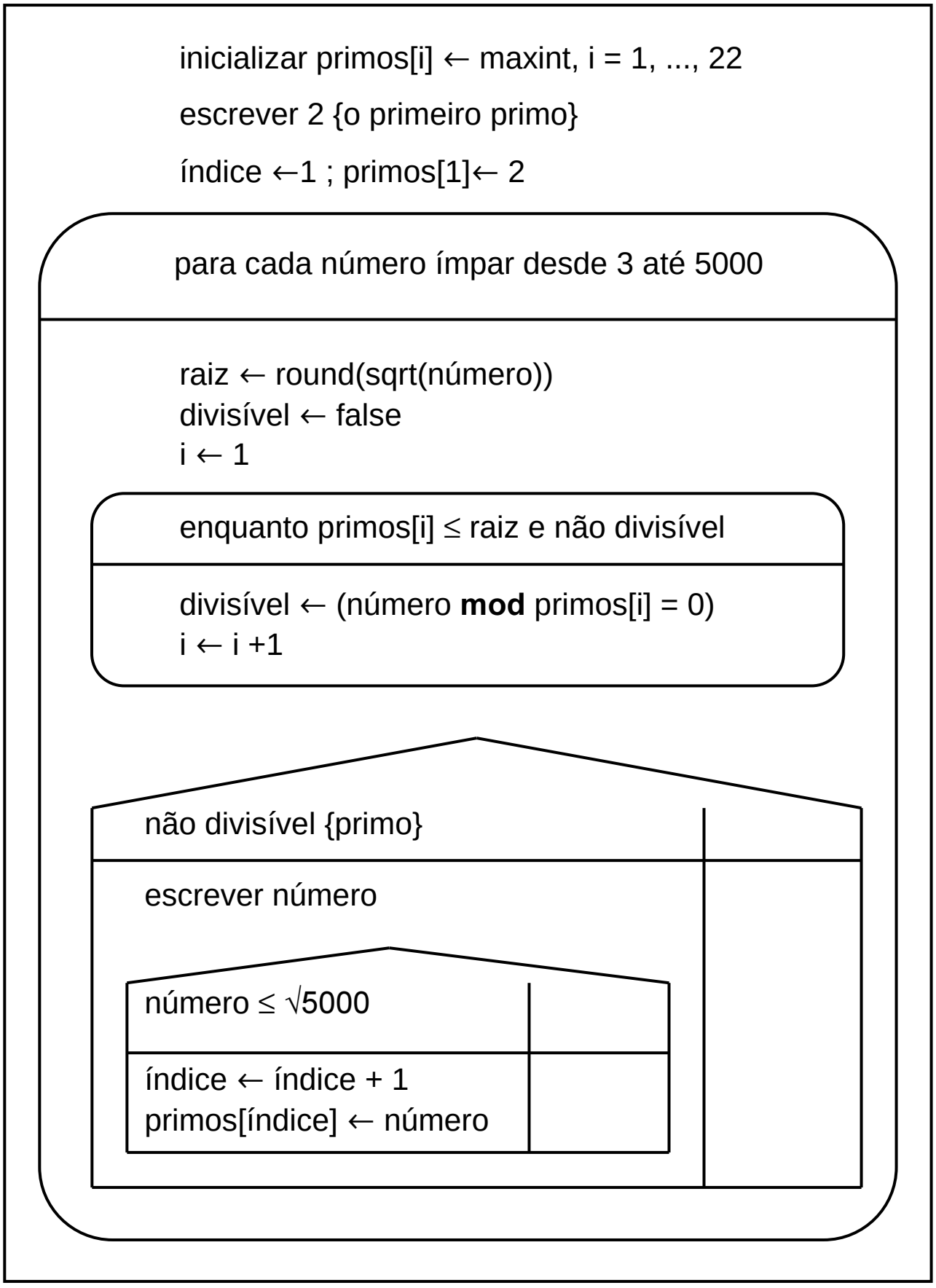
Como representar a lista de números primos?

- Em princípio pareceria adequada a utilização de uma **lista ligada**, pois terá **comprimento variável** e só serão feitos **acessos sequenciais**;
- Contudo, são apenas necessários 22 elementos. Por isso, neste caso, é aceitável a representação:

```
var primos : array [1..22] of integer;
```

- Vai ser necessário "simular", com um **array**, o funcionamento de uma lista ligada: acesso sequencial e comprimento variável;
- A variação do comprimento da lista é simulada com uma correcta inicialização dos elementos do **array**. Note-se que, neste caso, os elementos **não** devem ser inicializados a zero.

Diagrama de Estrutura:



Programa em Pascal:

```
program ListaPrimos5000(output);
{Este programa imprime os numeros primos ate 5000}

const  max = 5000;   {adaptabilidade}
       lim = 22;     {aproximacao de Legendre para pi(sqrt(5000))}
       raizdemax = 71;
var    primos : array[1..lim] of integer;  {lista de numeros primos}
       i, raiz, indice, numero : integer;
       divisivel : boolean;

begin  writeln('Lista dos Números Primos até ', max); writeln;
       {inicializacao dos elementos da lista}
       for i:=1 to lim do
           primos[i]:= maxint;
       {o primeiro numero primo}
       indice:= 1; primos[1]:=2;
       writeln(2);

       for  numero:= 3 to max do
            if odd(numero)
            then  begin  {dividir pelos primos ja encontrados}
                         raiz:= trunc(sqrt(numero));
                         divisivel:= false;
                         i:= 1;
                         while  not divisivel and (primos[i] <= raiz) do
                                begin  divisivel:= (numero mod primos[i]) = 0;
                                       i:= i+1
                                end;

                         if not divisivel
                         then  begin  {numero primo}
                                      writeln(numero);
                                      if numero <= raizdemax
                                      then  begin  {acrescentar a lista}
                                                   indice:= indice+1;
                                                   primos[indice]:= numero
                                            end
                               end
                  end
end{ListaPrimos5000}.
```

❒ Sub-programas

Subdivisão dos programas em unidades funcionais.

- Procedimento (**procedure**): realiza uma acção
- Função (**function**) : calcula e fornece um valor

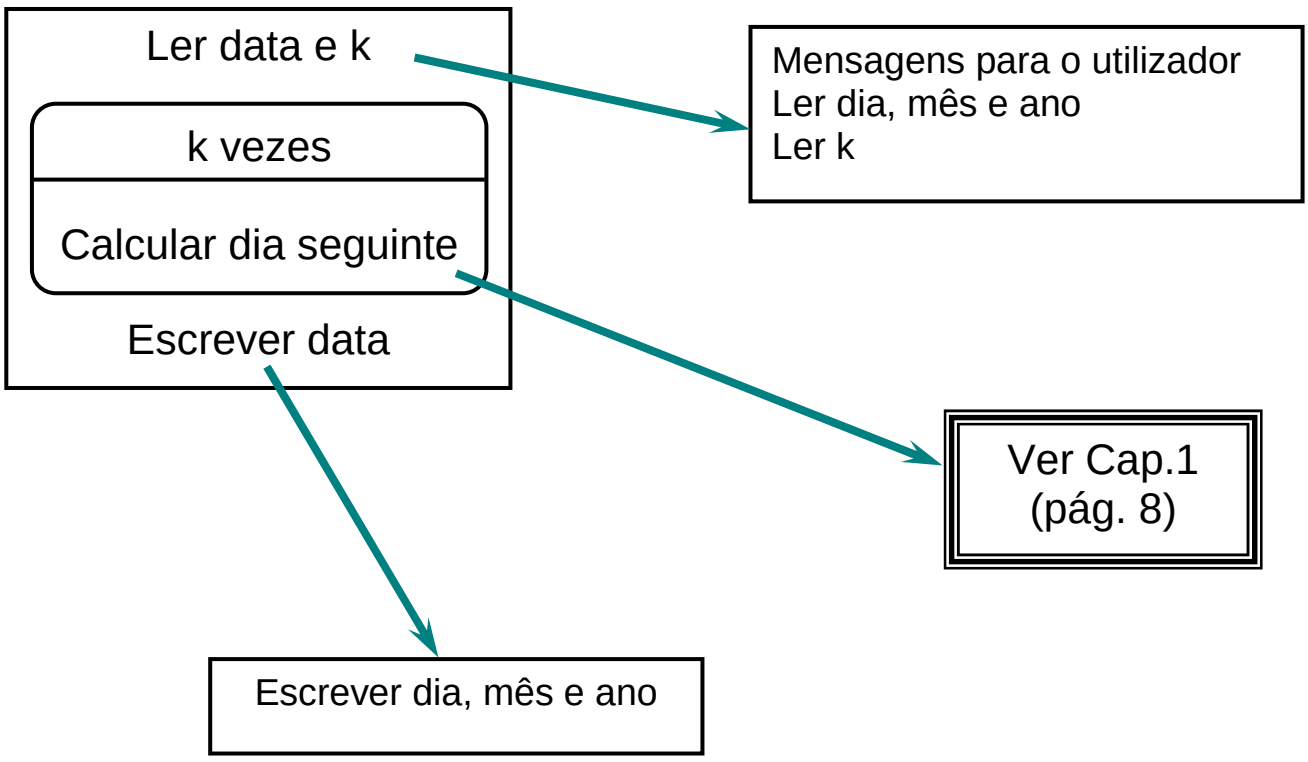
MODULARIDADE:

- Facilita a elaboração do(s) algoritmo(s);
- Aumenta a Clareza dos programas;
- Simplifica futuras Modificações;
- …

• Procedimentos

Dada uma data, pelos valores de dia, mês e ano, e também um valor inteiro e positivo k. Qual será a data, k dias após a data inicial?

Construção Descendente:



Primeira Versão do Programa:

```
program IncrementarData1(input, output);
var   dia, mes, ano, ultimo, i, k : integer;
      bissexto: boolean;

{ Declarações dos Procedimentos }

procedure  DiaSeguinte;
           begin   bissexto:= (ano mod 4 = 0) and (ano mod 100 <> 0) or (ano mod 400 = 0);
                   case mes of
                         1, 3, 5, 7, 8, 10, 12 : ultimo := 31;
                         4, 6, 9, 11 : ultimo := 30;
                         2 :  if  bissexto
                              then ultimo := 29
                              else  ultimo := 28
                   end;
                   if  dia = ultimo
                   then   begin  dia := 1;
                                 if mes = 12
                                 then  begin   mes := 1;
                                               ano := ano+1
                                       end
                                 else  mes := mes+1
                          end
                   else  dia := dia+1
           end {DiaSeguinte };

procedure  LerDados;
           begin   writeln('Indique o dia, mes e ano da presente data');
                   readln(dia, mes, ano);
                   writeln('Quantos dias após a presente data?');
                   readln(k)
           end { LerDados };

procedure  EscreverData;
           begin   write(k, ' Dias após a presente data são: ');
                   writeln(dia:3, '/', mes:2, '/', ano:4)
           end { EscreverData };


begin { Programa Principal }
      LerDados;
      for  i:= 1 to k do
           DiaSeguinte;
      EscreverData
end { IncrementarData1 }.
```

{ Chamadas dos Procedimentos }

Uma **correcta** utilização de Subprogramas reduz:

- O comprimento total do Programa.
- As áreas de potenciais erros.
- O esforço de programação.

O Domínio dos Identificadores:

- O **Bloco** de um Subprograma é equivalente ao Bloco de um Programa.
- É portanto sintacticamente possível declarar Constantes, Tipos, Variáveis e outros Subprogramas, no Bloco de um Subprograma.
- É conveniente declarar no Bloco de um Subprograma todos os identificadores que são utilizados unicamente nesse Subprograma.
- Os identificadores declarados no Bloco de um Subprograma chamam-se **locais** a esse Subprograma, não podendo ser utilizados fora dele.
- Os identificadores declarados no Bloco do Programa Principal são **globais**, podendo ser utilizados no Programa Principal e em todos os seus Subprogramas.

p. ex.:

```
var  ultimo : integer;
     bissexto: boolean;    {São locais ao Procedimento DiaSeguinte}

var  dia, mes, ano, i, k : integer;          {São Variáveis globais}
```

```
program IncrementarData2(input, output);
var   dia, mes, ano, i, k : integer;
procedure  DiaSeguinte;
    var  ultimo : integer;
         bissexto: boolean;
    begin   bissexto:= (ano mod 4 = 0) and
                                (ano mod 100 <> 0) or (ano mod 400 = 0);
            case mes of
                    1, 3, 5, 7, 8, 10, 12 : ultimo := 31;
                    4, 6, 9, 11 : ultimo := 30;
                    2 : if  bissexto
                        then ultimo := 29
                        else  ultimo := 28
            end;
            if  dia = ultimo
            then   begin   dia := 1;
                           if mes = 12
                           then   begin   mes := 1;
                                          ano := ano+1
                                  end
                           else  mes := mes+1
                   end
            else  dia := dia+1
    end { DiaSeguinte };
procedure  LerDados;
           begin   writeln('Indique o dia, mes e ano da presente data');
                   readln(dia, mes, ano);
                   writeln('Quantos dias após a presente data?');
                   readln(k)
           end { LerDados };
procedure  EscreverData;
           begin   write(k, ' Dias após a presente data são: ');
                   writeln(dia:3, '/', mes:2, '/', ano:4)
           end { EscreverData };

begin { Programa Principal }
      LerDados;
      for  i:= 1 to k do
           DiaSeguinte;
      EscreverData
end { IncrementarData2 }.
```

Bloco do Programa Principal

Bloco do Procedimento DiaSeguinte

De um modo geral, a utilização de **Declarações Locais**:

- Torna claro que as Variáveis Locais só têm significado dentro dos Subprogramas em que foram declaradas, tornando o programa **mais legível**.
- Assegura que a utilização dessas Variáveis, fora dos Subprogramas onde foram declaradas, provoque um erro imediatamente detectado pelo Compilador, **facilitando a detecção e correcção dos erros**.
- Facilita a minimização do espaço de Memória usado por essas Variáveis, assim tornando o programa **mais eficiente**.

Procedimentos englobados noutros Procedimentos:

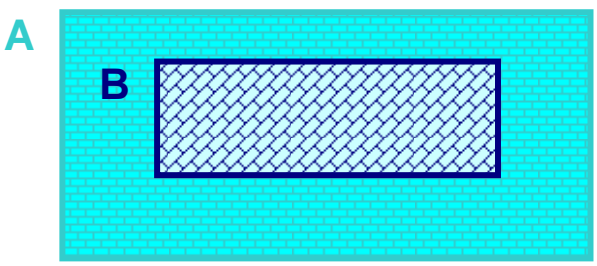
- No Bloco de um Subprograma podem ser declarados: Constantes, Tipos, Variáveis e, em particular, outros Subprogramas.

Domínio dos Identificadores:

Regras:

1. O Domínio de um Identificador é o Bloco no qual está declarado e todos os Blocos englobados nesse Bloco, exceptuando a Regra 2.

2. Quando um Identificador é **re**declarado num Bloco B, englobado por A, a nova declaração sobrepõe-se à anterior, no Bloco B e em todos os Blocos englobados por B.



Exemplo de Procedimentos Englobados:

```
procedure DiaSeguinte;
var ultimo : integer;

procedure CalcularUltimo;
        var bissexto: boolean;
        begin { CalcularUltimo }
                bissexto:= (ano mod 4 = 0) and
                   (ano mod 100 <> 0) or (ano mod 400 = 0);
                case mes of
                        1, 3, 5, 7, 8, 10, 12 : ultimo := 31;
                        4, 6, 9, 11 : ultimo := 30;
                        2 :  if  bissexto
                             then ultimo := 29
                             else  ultimo := 28
                end;
        end { CalcularUltimo };

procedure Actualizar;
        begin { Actualizar }
                if  dia = ultimo
                then begin dia := 1;
                           if mes = 12
                           then begin mes := 1;
                                      ano := ano+1
                                end
                           else  mes := mes+1
                     end
                else  dia := dia+1
        end { Actualizar };

begin { DiaSeguinte }
        CalcularUltimo;
        Actualizar
end { DiaSeguinte };
```

- A Variável ultimo é **local** a DiaSeguinte e **global** a CalcularUltimo e Actualizar. A Variável bissexto é **local** a CalcularUltimo.
- Os Procedimentos CalcularUltimo e Actualizar são **locais** ao Procedimento DiaSeguinte.

Que fazer?

- Evitar declarações globais!
- Declarar cada Identificador no Bloco em que é usado.
- Quando um Identificador tem de ser usado em dois ou mais Procedimentos para representar o mesmo objecto, deve ser declarado no Bloco que imediatamente os engloba.
- Localizar sempre todos contadores de ciclos.

Não esquecendo que:

- As declarações locais aumentam a liberdade de escolha dos Indentificadores a usar, porque identificadores locais a Procedimentos "disjuntos" são totalmente independentes.

Como Comunicam os Módulos?

A utilização de Parâmetros:

Parâmetros Formais

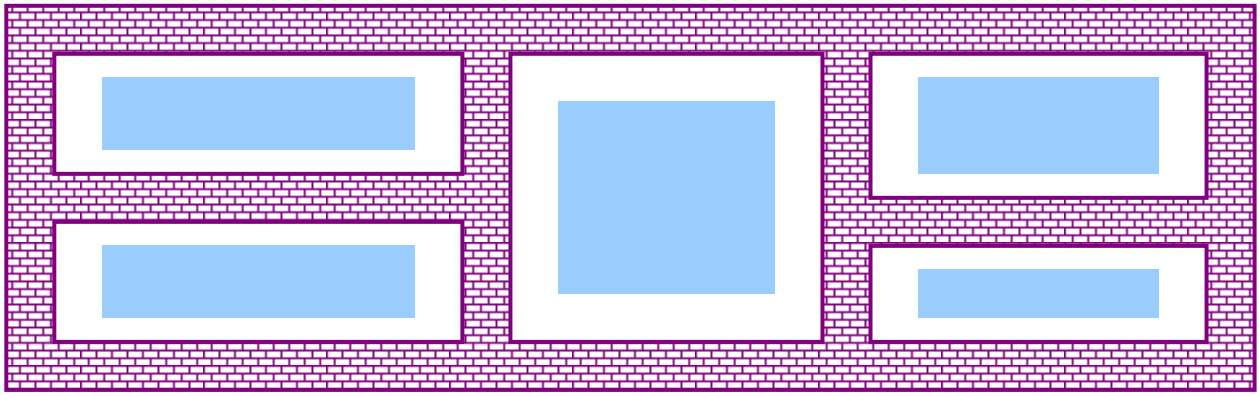
```
program MiasDatas (input, output);
var dia, mes, ano, ... : integer;

procedure DiaSeguinte(var dia, mes, ano : integer);

          var  ultimo : integer;
               bissexto: boolean;
          begin ...
                  ...
          end { DiaSeguinte };

begin ...
      { Aqui 29/2/2000 }
      DiaSeguinte(dia, mes, ano);
      { Aqui 1/3/2000 }
      ...
end { MaisDatas}.
```

Parâmetros Actuais

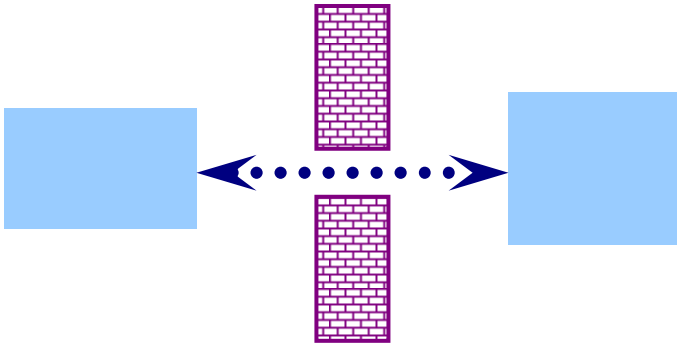
- Na declaração de um Procedimento é definida uma Lista de **Parâmetros Formais**;
- Na chamada do Procedimento, estes são substituídos pelos correspondentes **Parâmetros Actuais** (ou Reais).
- O número de Parâmetros nas duas listas é necessariamente igual e a correspondência é determinada pela posição na lista.
- Parâmetros correspondentes são do mesmo tipo e classe.

Uma *imagem* da Programação Estruturada:

A independência dos Módulos



A Comunicação entre os Módulos



Classes de Parâmetros (Modos de Ligação):

- Classe Variável - Ligação por Referência
- Classe Valor - Ligação por Valor

Exemplo:

```
program simples (output);
var x : integer;

procedure  alterar(var a : integer);
           begin a:= 1;
           end;

begin  x:= 0;
       alterar(x);
       writeln(x)
end.
```

Resultado: 1

O Parâmetro Formal **a** foi declarado da Classe Variável.

A chamada do Procedimento alterou o Valor do Parâmetro Actual **x** no Programa Principal.

```
program facil (output);
var x : integer;

procedure  mudar(a : integer);
           begin a:= 1;
           end;

begin  x:= 0;
       mudar(x);
       writeln(x)
end.
```

Resultado: 0

O Parâmetro Formal **a** foi declarado da Classe Valor.

A chamada do Procedimento não mudou o Valor do Parâmetro Actual **x** no Programa Principal.

Que fazer?

- 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.

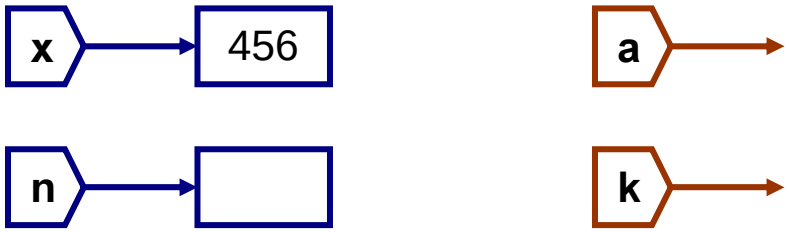
Mais um exemplo: Contar os dígitos de um número.

```
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.
```

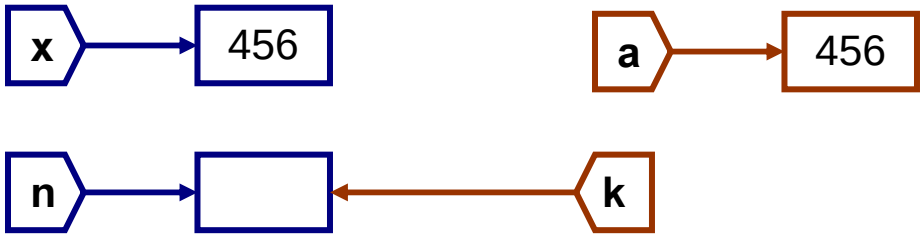
- O valor do Parâmetro Actual x = 456 não foi afectado pela contagem dos seus dígitos;
- O Parâmetro Actual n foi usado para passar o resultado dessa contagem para o Programa;
- Um Parâmetro Actual, quando Declarado da Classe Valor, pode ser o valor de uma expressão, tal como n * (100 + 2 * n).

Mas como funciona?

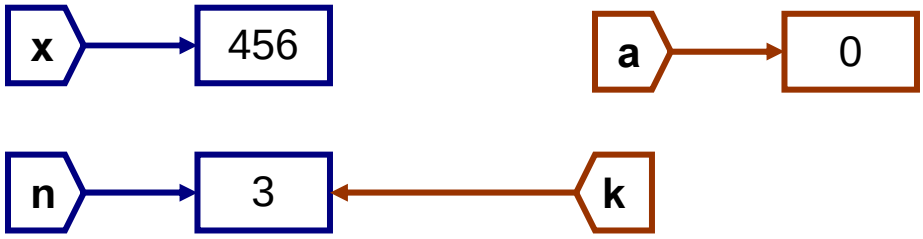
Antes da Chamada:



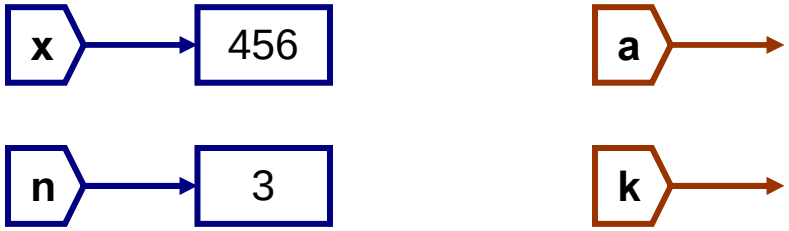
Início do Procedimento:



Fim do Procedimento:



Depois da Chamada:

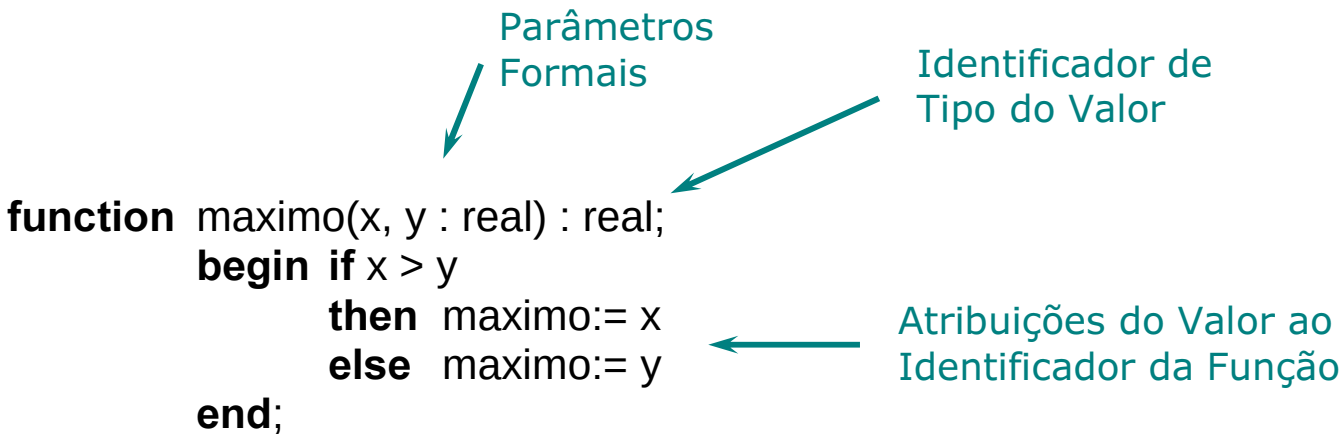


- Se o Parâmetro for da Classe Valor é gerada uma **cópia temporária**, que é destruída após a execução do Procedimento;

- Se o Parâmetro for da Classe Variável, a ligação entre o Parâmetro Formal e o Parâmetro Actual é feita apenas **por referência**.

- **Funções**

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




```
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);
        ...
```

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 Identificadores 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

```
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.

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

```
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: Construir a função,
function SomaDivisores(numero : intnneg) : intnneg;
e utilizá-la na função perfeito (...) anterior.

```
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 gerar uma Chamada da própria Função primo, sem Parâmetros, o que provocaria um ERRO.

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).

```
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 *);
```

- Necessidade da variável local auxiliar fa.