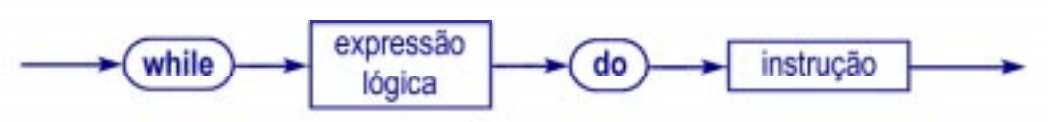
3. Estruturas de Repetição (Ciclos)

3. 1. Teste no Início do Ciclo (Repetição "enquanto"):



- **A Expressão Lógica é calculada e, enquanto se mantiver verdadeira, a Instrução (Simples ou Composta) é executada.**

Exemplo:

Imprimir uma tabela das potências inteiras de 2 inferiores a 1000.

```
i ← 1
potência ← 2
enquanto potência < 1000
escrever ( i, potência)
i ← i + 1
potência ← potência * 2
```

Teste (Cabeça do Ciclo)

Instrução Composta (Corpo do Ciclo)

(output)

1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512

Programa:

```
program potenciasde2 (output);
var i, potencia : integer;
begin  (* Inicializacoes *)
        i:= 1; potencia:= 2;
        while  potencia < 1000 do
                begin  writeln(i:5, potencia:5);
                        (* Formulas Iterativas *)
                        i:= i + 1;
                        potencia:= potencia * 2
                end
end.
```

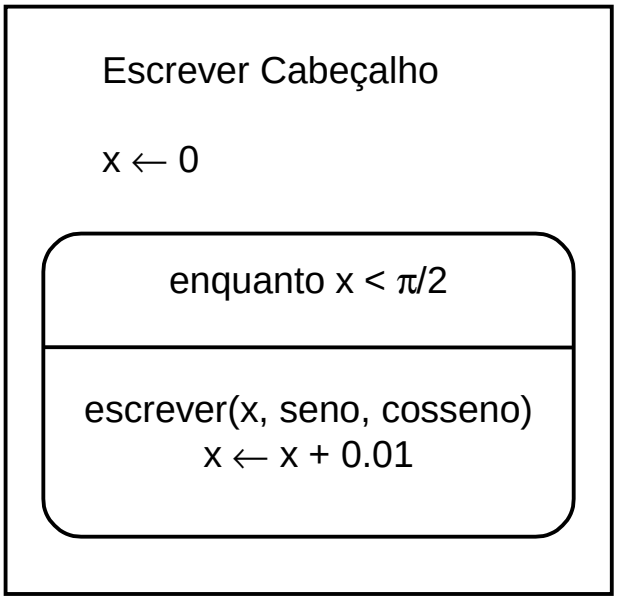
- **Se a Expressão Lógica for falsa à Entrada do Ciclo, este nunca chega a ser executado.**

Problema: Dado: $\mathbf{n} \in \mathbb{N}_0$
Resultado: 2^n

Programa:

```
program potenciade2 (input, output);
type natural = 0..maxint;
var i, n, potencia : natural;
begin  read(n);
        i:= 0; potencia:= 1;
        while  i < n  do
                begin  i:= i + 1;
                        potencia:= potencia * 2
                end;
        (* aqui i = n *)
        writeln('2 elevado a', n:5, ' = ', potencia:5)
end.
```

Problema: Imprimir uma tabela de senos e cossenos dos valores de $x \in [0 , \pi/2[$ com intervalos de 0.01 radianos.




```
   x   sin(x)   cos(x)

0.00   0.0000   1.0000
0.01   0.0100   1.0000
0.02   0.0200   0.9998
0.03   0.0300   0.9996
0.04   0.0400   0.9992
0.05   0.0500   0.9988
0.06   0.0600   0.9982

...

1.51   0.9982   0.0608
1.52   0.9987   0.0508
1.53   0.9992   0.0408
1.54   0.9995   0.0308
1.55   0.9998   0.0208
1.56   0.9999   0.0108
1.57   1.0000   0.0008
```

Programa:

```
program senosecossenos (output);
const pi = 3.14159265;
var x, pi2 : real;
begin  (* Cabeçalho da Tabela *)
       writeln('x' : 6, 'sin(x)' : 8, 'cos(x)' : 8);
       writeln;
       x:= 0;
       pi2:= pi/2;
       while  x < pi2 do
              begin  writeln(x:6:2, sin(x):8:4, cos(x):8:4);
                     x:= x + 0.01
              end
end.
```

Problema: Dados: $a, b \in \mathbb{N}_0$ não simultaneamente nulos,
Calcular: mdc(a, b) pelo Algoritmo de Euclides.

Exemplos:

mdc(380,55) = ?

a	b	resto
380	**55**	50
55	50	5
50	5	0
5	**0**	

mdc(380,55) = 5

mdc(55,380) = ?

a	b	resto
55	**380**	55
380	55	50
55	50	5
50	5	0
5	**0**	

mdc(55,380) = 5

Programa:

```
program Euclides (input, output);
type natural = 0..maxint;
var a, b, resto : natural;
begin  writeln('Escreva dois numeros naturais, nao
                                        simultaneamente nulos');
       read(a, b);
       while  b <> 0 do
              begin  resto:= a mod b;
                     a:= b;
                     b:= resto
              end;
       (* aqui b = 0 *)
       writeln('mdc = ', a)
end.
```

Notas:

- No programa anterior, os valores iniciais de a e b são destruídos.
- Para o valor inicial b=0, o ciclo nunca é executado, pois mdc(a,0) = a.

Problema: Calcular também o mmc(a,b) = a * b / mdc(a,b).

Programa:

```
program Euclides2 (input, output);
type natural = 0..maxint;
var a, aa, b, bb, resto : natural;
begin  writeln('Escreva dois numeros naturais, nao
                                        simultaneamente nulos');
       read(a, b);
       (* tirar cópias de a e de b *)
       aa:= a;
       bb:= b;

       (* aplicar o Algoritmo de Euclides às cópias *)
       while  bb <> 0 do
              begin  resto:= aa mod bb;
                     aa:= bb;
                     bb:= resto
              end;
       (* aqui bb = 0  e  aa = mdc(a,b) *)

       writeln('mdc = ', aa, ' e mmc = ', a*b div aa)
end.
```

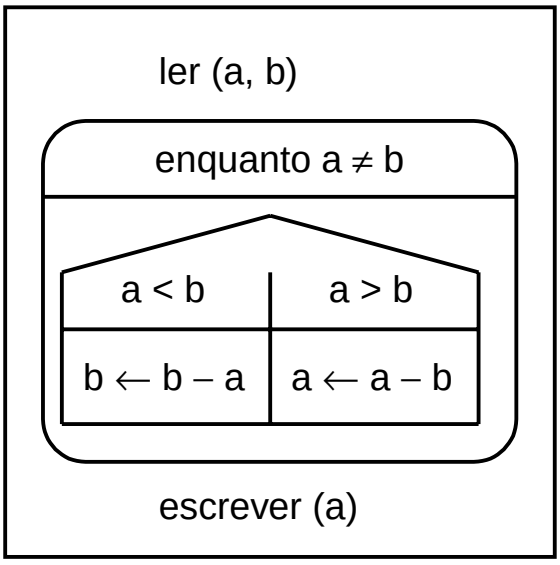
Exercício: Aperfeiçoar a escrita dos resultados, por forma a que escreva:

mdc(380,55) = 5 e mmc(380,55) = 4180

Algoritmo original de Euclides:

Na sua sua forma original, o Algoritmo de Euclides para o cálculo do mdc não utilizava divisões. Consistia em:

Dados dois números inteiros e positivos, subtrair sucessivamente o menor ao maior, até ficarem iguais.



a	b
9	**30**
9	21
9	12
9	3
6	3
3	**3**

mdc(9,30)=3

Programa:

```
program EuclidesOriginal (input, output);
type intpos = 1..maxint;
var a, b : intpos;
begin  writeln('Escreva dois numeros inteiros e positivos');
       read(a, b);
       while  a <> b do
              if  a < b
              then  b:= b – a
              else  a:= a – b;
       (* aqui a = b *)
       writeln('mdc = ', a)
  end.
```

Problema: Calcular $\mathbf{e^x}$ por desenvolvimento em **Série de Taylor** desprezando termos, em grandeza, inferiores a 10^{-7}.

Um pouco de Análise Infinitesimal:

Desenvolvimento em série de Taylor da função exponencial:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots, \textit{convergente } \forall x \in \mathbb{R}$$

Somas parciais de ordem n e $n+1$:

$$\begin{aligned} S_n &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \\ S_{n+1} &= S_n + \frac{x^{n+1}}{(n+1)!} \end{aligned}$$

Termos de ordem n e $n+1$:

$$\begin{aligned} t_n &= \frac{x^n}{n!} \\ t_{n+1} &= \frac{x^{n+1}}{(n+1)!} = t_n \frac{x}{n+1} \end{aligned}$$

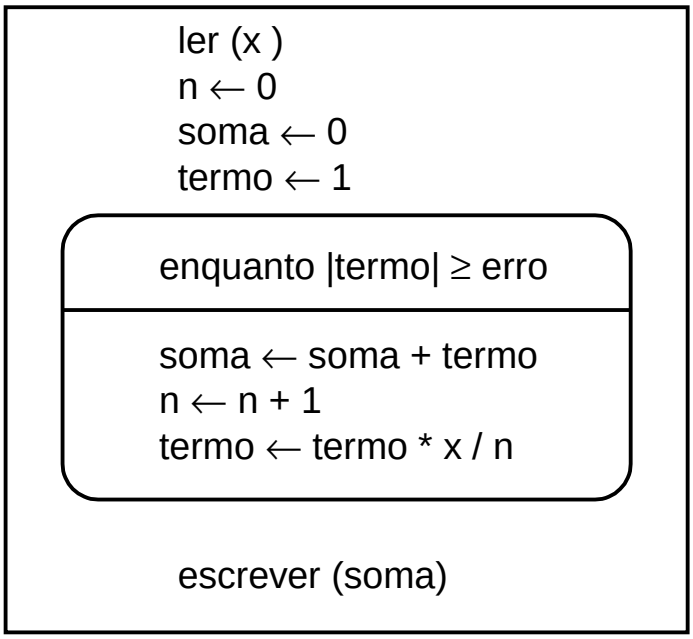
A série é convergente $\forall x \in \mathbb{R}$, isto é:

$$\lim_{n\to\infty} \{S_n\} = e^x$$

E, nesse caso:

$$\lim_{n\to\infty} \{t_n\} = 0$$

Diagrama de Estrutura do Algoritmo:



Programa:

```
program exponencial (input, output);
const erro = 1.0E-7;
var x, termo, soma : real;
begin  writeln('Escreva um numero real);
       read(x);
       (* inicializações *)
       soma:= 0;
       n:= 0;
       termo:= 1;
       while  abs(termo) >= erro do
              begin  (* somar termo válido *)
                     soma:= soma + termo;
                     (* calcular novo termo *)
                     n:= n + 1;
                     termo:= termo * x / n
              end;
       (* aqui abs(termo) < erro *)
       writeln('Valor da soma = ', soma:20:7)
  end.
```

Notas:

- Caso particular para x=1: e = 2.7182818284…

  Resultados do programa:

n	termo	soma
0	1.0000000	0.0000000
1	1.0000000	1.0000000
2	0.5000000	2.0000000
3	0.1666667	2.5000000
4	0.0416667	2.6666667
5	0.0083333	2.7083333
6	0.0013889	2.7166667
7	0.0001984	2.7180556
8	0.0000248	2.7182540
9	0.0000028	2.7182788
10	0.0000003	2.7182815

- Que garantia temos, mesmo no caso geral, de que o ciclo vai parar?

$$\lim_{n\to\infty}\{t_n\} = 0 \iff \forall \varepsilon \in \mathbb{R}_+,\ \exists N : \forall n \geq N \Rightarrow |t_n| < \varepsilon$$

- Que pode acontecer, quando não conhecemos a convergência da série?

Exercícios:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,\ \forall x \in \mathbb{R}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots,\ \forall x \in \mathbb{R}$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots,\ -1 < x \leq 1$$

Outra vez o exemplo: Se o dia 1 de Novembro de 1999 foi uma Segunda-Feira calcular os restantes.
(ver Cap.III, pág. 9)

Um programa educado e paciente, para com o utilizador distraído, que escreve valores de dia ∉ [1, 30]:

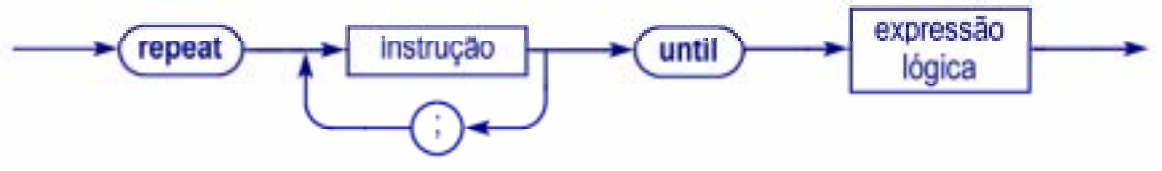
```
program Novembro2(input, output);
var dia : integer;
begin  writeln('Por favor, indique o valor de um dia de Novembro');
       read(dia);

       (* Validação dos Dados *)
       while  (dia < 1) or (dia > 30) do
              begin  writeln(dia, 'Não é valor de dia de Novembro,
                                                    tente outra vez');
                     read(dia)
              end;
       (* Aqui (dia >= 1) and (dia <= 30) *)

       write('O dia ', dia, ' de Novembro de 1999 é ');

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

Mas não será paciência a mais?
E se o utilizador nunca acertar?

Vamos impôr um limite ao número de “distrações”:

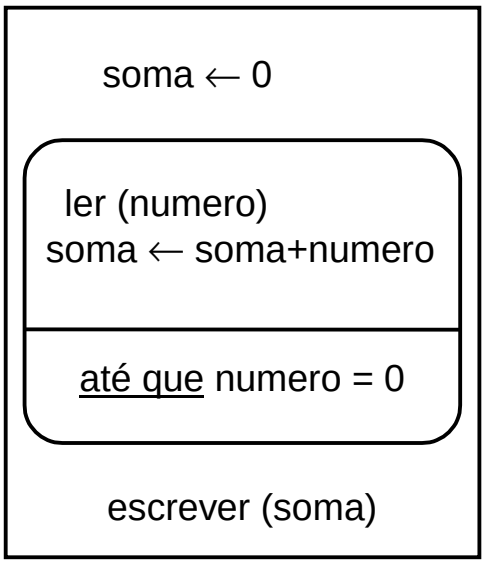
```
program Novembro3(input, output);
var  dia, tentativas : integer;
     erro : boolean;

begin writeln('Por favor, indique o valor de um dia de Novembro');
      read(dia);

      (* Validação dos Dados *)
      tentativas:= 1;
      erro:= (dia < 1) or (dia > 30);

      while  erro and (tentativas < 10) do
             begin writeln('Enganou-se. Tente outra vez');
                   read(dia);
                   tentativas:= tentativas + 1;
                   erro:= (dia < 1) or (dia > 30)
             end;
      (* Aqui (not erro) or (tentativas >= 10) *)

      if erro
      then writeln('Enganou-se 10 vezes. Eu desisto!')
      else begin write('O dia ', dia, ' de Novembro de 1999 é ');
                 case dia mod 7 of
                       0 : writeln('um Domingo');
                       1 : writeln('uma Segunda Feira');
                       2 : writeln('uma Terca Feira');
                       3 : writeln('uma Quarta Feira');
                       4 : writeln('uma Quinta Feira');
                       5 : writeln('uma Sexta Feira');
                       6 : writeln('um Sabado')
                 end
           end
end.
```

3.2. Teste no Fim do Ciclo (Repetição "até que"):



- **A Instrução (ou Sequência de Instruções) é repetida até que a Expressão Lógica se torne verdadeira.**
- **A Sequência de Instruções é executada pelo menos uma vez.**

Problema: Calcular a soma de vários números inteiros, dados pelo utilizador, um por linha. A última linha contem 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 utilizador tem de escrever, pelo menos, o número zero.**
- **O zero é efectivamente somado.**

Problema: O dono de um hotel, resolveu estabelecer os seus preços de uma forma bastante original:

- Pela primeira noite cobrava 100 000$00, pela segunda noite 50 000$00 e, de um modo geral, pela noite de ordem **n** cobrava **100 / n!** milhares de escudos.

Um cliente aceitou ficar, mas apenas enquanto o preço de uma noite fosse superior a 10$00.
Quantas noites ficou e qual o total da sua conta?

Programa:

```
program hotel(output);
var   noites : integer;
      preco, total : real;
begin  noites:= 1;
       preco:= 1.0E5;
       total:= 0;
       repeat   total:= total + preco;
                (* Calcular a próxima *)
                noites:= noites + 1;
                preco:= preco/noites
       until  preco <= 10;
       noites:= noites - 1;
       writeln;
       writeln('Ficou', noites:6, 'noites e pagou', total:12:2)
end.
```

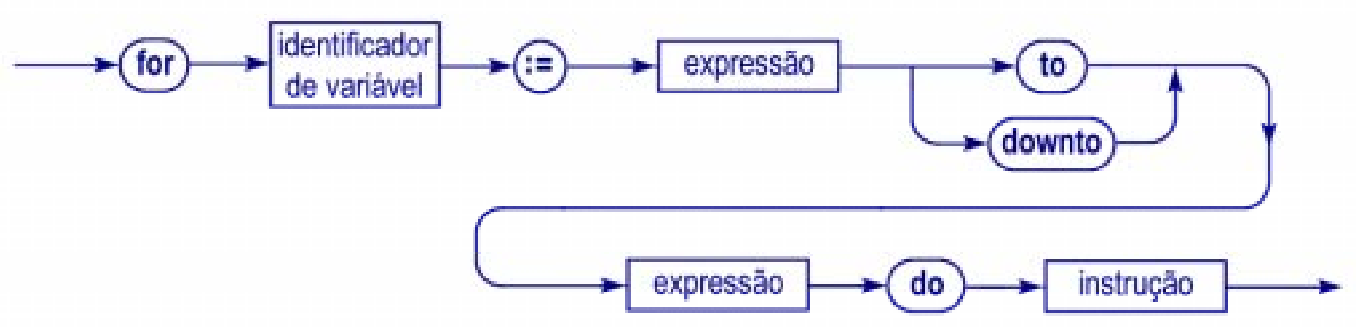
```
noites       preco       total
     1   100000.00   100000.00
     2    50000.00   150000.00
     3    16666.67   166666.67
     4     4166.67   170833.33
     5      833.33   171666.67
     6      138.89   171805.56
     7       19.84   171825.40
     8        2.48
Ficou 7 noites e pagou  171825.40
```

E mais uma vez o programa da Calculadora:
(ver Cap.II - pág. 8 e Cap.III - pág. 10)

```
program calculadora4(input, output);
var   oper: char;
      x, y, res: real;
      opcerto, divcerta: boolean;

begin   repeat  writeln('Qual a Operacao?');
                opcerto:= true;
                divcerta:= true;
                read(x, oper, y);
                case oper of
                      '+': res := x+y;
                      '–': res := x–y;
                      '*': res := x*y;
                      '/':  if y = 0
                            then divcerta:= false
                            else res := x/y;
                otherwise  opcerto:= false
                end;

                if  opcerto and divcerta
                then  writeln(x, oper, y, '=', res)
                else  begin  write('ERRO:  ');
                             if opcerto
                             then write('Divisão por zero, ')
                             else  write('Operador nao Previsto, ');
                             writeln('Tente outra vez.')
                      end
        until opcerto and divcerta
end.
```

Exercício: **Incluir a possibilidade do utilizador responder se Sim/Não pretende resolver outra expressão.**

3.3. Ciclo com Contador:



- **O Ciclo é executado um número fixo de vezes.**
- **O Incremento da Variável pode ser +1 (to) ou –1(downto).**

Exemplo1: n! = 1 * 2 * 3 * 4 * … * n

```
factorial:= 1;
for  i:= 1 to n do
     factorial:= factorial * i ;
```

Exemplo2: n! = n * (n–1) * (n–2) * … * 3 * 2 * 1

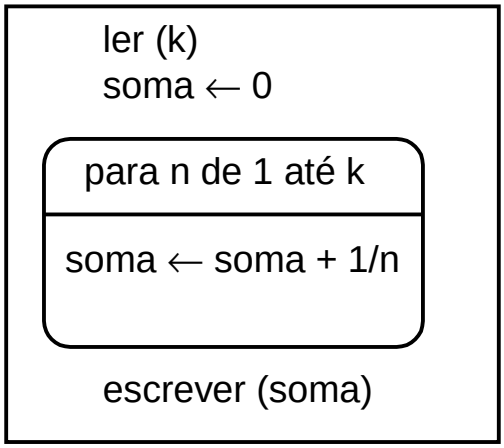
```
factorial:= 1;
for  i:= n downto 1 do
     factorial:= factorial * i ;
```

A Variável e as Expressões:

- **São do mesmo Tipo;**
- **São Escalares (não Reais);**
- **Não podem ser alteradas dentro do Ciclo.**

Problema: Calcular a soma dos k primeiros termos da Série Harmónica

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



Programa:

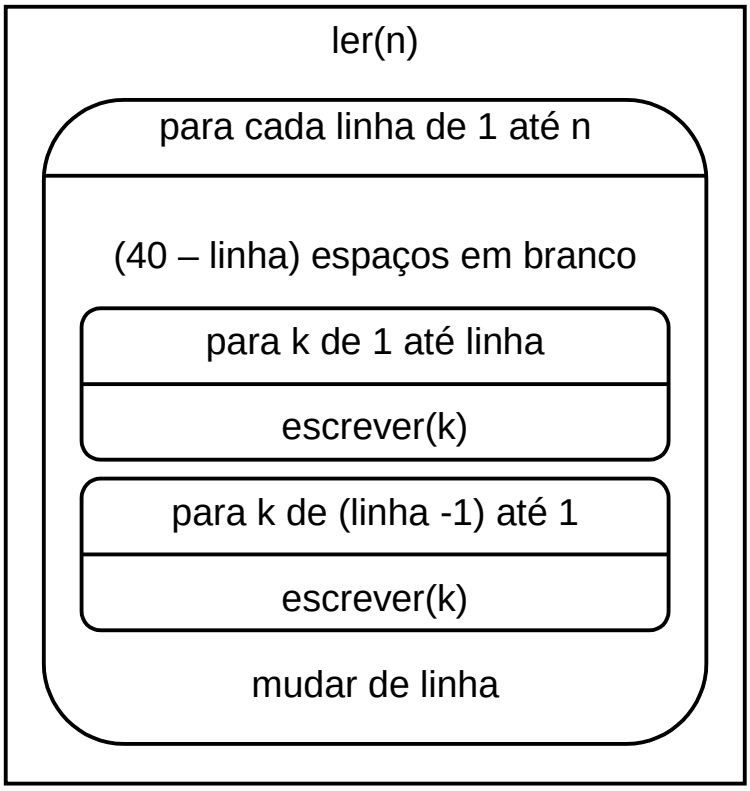
```
program Harmonica(input, output);
var  k, n : integer;
     soma : real;

begin  writeln('Quantos termos?');
       read(k);
       soma:= 0;
       for n:=1 to k do
            soma:= soma + 1/n;
       writeln(soma)
end.
```

- **Os valores das duas Expressões são calculados no início e funcionam como constantes, durante a execução do Ciclo.**
- **No caso do Incremento Positivo (to) se, o valor da primeira Expressão for superior ao da segunda, o Ciclo não é executado.**
- **No caso do Incremento Negativo (downto) se, o valor da primeira Expressão for inferior ao da segunda, o Ciclo não é executado.**
- **O valor da Variável torna-se indefinido após a execução do Ciclo.**

Problema:

Construir uma pirâmide de números como, por exemplo, para n=6:

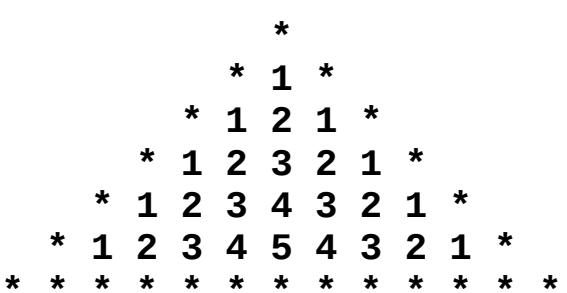
```
     1
   1 2 1
 1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
```

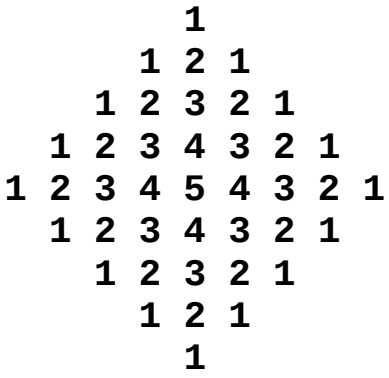


Programa:

```
program Piramide(input, output);
var   linha, k, n : integer;

begin  writeln('Qual o tamanho da piramide?');
       read(n);

       for  linha:=1 to n do
            begin write(' ':40-linha);
                  for k:=1 to linha do
                      write(k:2);
                  for k:=linha-1 downto 1 do
                      write(k:2);
                  writeln
            end
end.
```

Exercícios:



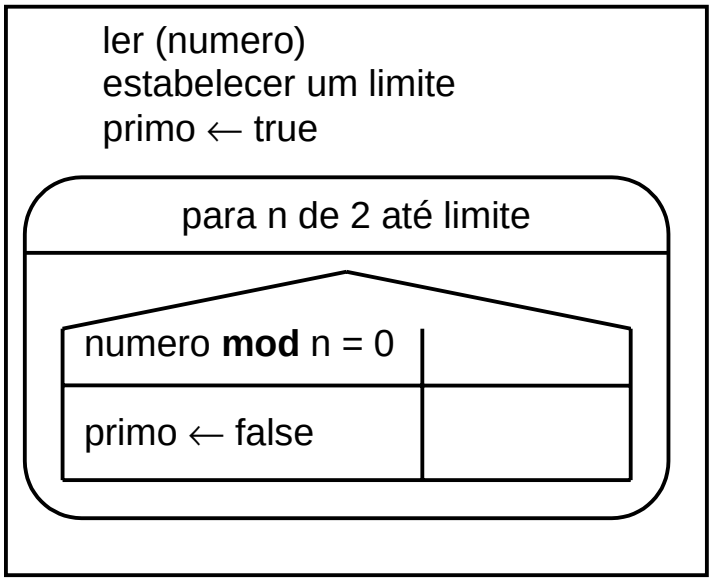


Problema: Verificar se um dado número natural é ou não um Número Primo.

Algoritmo:

Mas que limite?

n/2 ? $\sqrt{n}$?



Teorema: Se um dado número natural n não possuir divisores ≤ sqrt(n), então n é um Número Primo.

Demonstração (por absurdo):

Suponhamos que n, sem divisores ≤ sqrt(n), **não era** um Número Primo. Então, existiriam a,b > sqrt(n), tais que **n = a * b**.
Nesse caso teríamos:

$$n = a * b > (sqrt(n))^2 = n$$

ou, **n > n**, o que **seria absurdo**.

- **Com base neste resultado, podemos utilizar o valor de:**
  limite = trunc(sqrt(número)).

Programa:

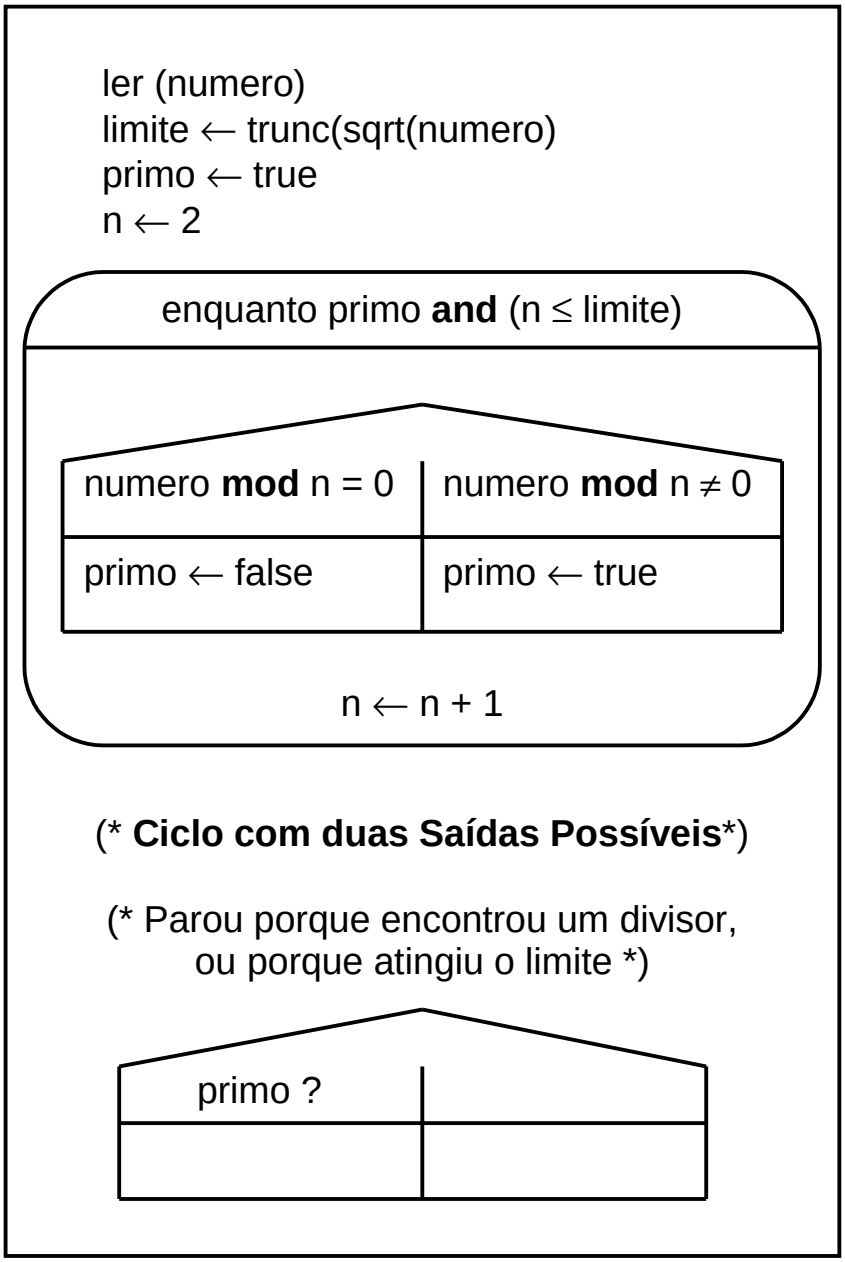
```
program TestaPrimos(input, output);
var   numero, limite, n : 1..maxint;
      primo : boolean;

begin writeln('Qual o numero?');
      read(numero);
      limite:= trunc(sqrt(numero));
      primo:= true;

      for  n:=2 to limite do
          if numero mod n = 0
          then primo:= false;

      if primo
      then  writeln(numero, 'É um Número Primo')
      else  writeln(numero, 'Não é um Número Primo')
end.
```

- **Note-se que a variável lógica primo só mantem o valor verdadeiro inicial, se não for encontrado nenhum divisor.**
- **Se a variável lógica tomar, pelo menos uma vez o valor falso, nunca mais pode passar a ser verdadeira.**

- **Mas o que acontece, se for encontrado um divisor, logo nas primeiras tentativas?**

Versão mais Eficiente:

Testar só até encontrar o primeiro divisor.



Programa mais Eficiente:

```
program TestaPrimos2(input, output);
var  numero, limite, n : 1..maxint;
     primo : boolean;

begin writeln('Qual o numero?');
      read(numero);
      limite:= trunc(sqrt(numero));
      primo:= true;
      n:= 2;

      while  primo and (n <= limite) do
             begin  primo:= (numero mod n) <> 0;
                    n:= n+1
             end;

      (* Aqui not primo or (n>limite) *)

      if primo
      then  writeln(numero, 'É um Número Primo')
      else  writeln(numero, 'Não é um Número Primo')
end.
```

Exercícios:

- Qual seria outro teste possível, após a Saída do Ciclo?
- O mesmo programa, sem a variável lógica.
- O mesmo problema, com um ciclo **repeat**.

- Imprimir a Lista de todos os Números Primos até 1000.
- Imprimir a Lista dos 1000 primeiros Números Primos.

Problema: No século I D.C., os números inteiros positivos dividiam-se em três categorias:

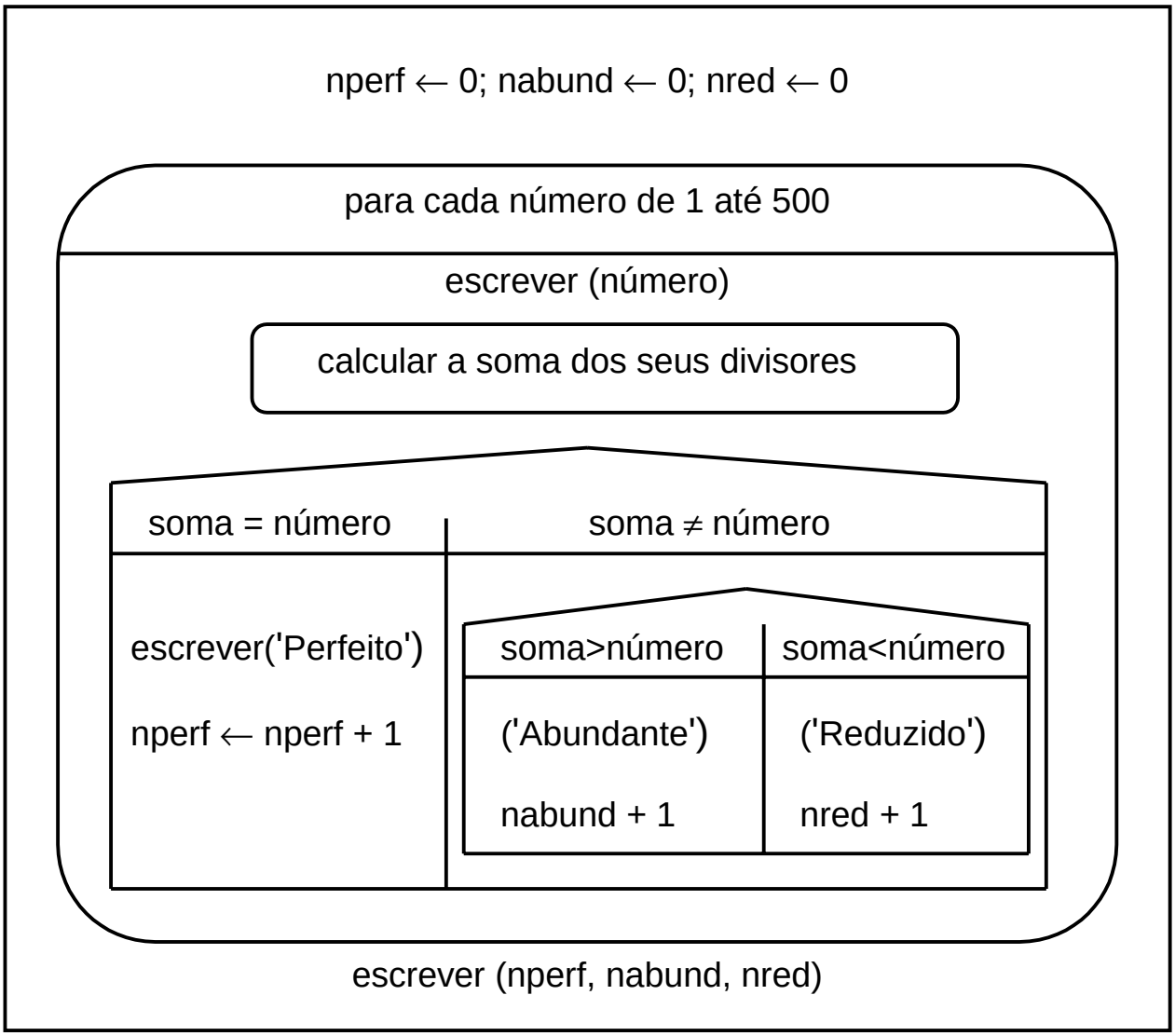
Perfeitos – aqueles que são iguais à soma dos seus divisores.
(p.ex. 6 = 1 + 2 + 3)

Abundantes – inferiores à soma dos seus divisores.
(p.ex. 12 < 1 + 2 + 3 + 4 + 6 = 16)

Reduzidos – superiores à soma dos seus divisores.
(p.ex. 9 > 1 + 3 = 4)

Note-se que, neste caso, eram considerados divisores de um número, a unidade mas não o próprio número.

Imprima a lista dos inteiros de 1 até 500, classificando-os em perfeitos, abundantes ou reduzidos. Calcule também a quantidade de elementos de cada categoria.



Programa:

```
program Historico(output);
const max = 500;
var   numero, n, soma, nperf, nabund, nred : 0..max;

begin  nperf:= 0;
       nabund:= 0;
       nred:= 0;

       for  numero:=1 to max do
            begin  write(numero);
                   soma:= 1;
                   for  n:=2 to (numero div 2) do
                        if numero mod n = 0
                        then soma:= soma + n;
                   if soma = numero
                   then  begin  writeln(' Número Perfeito');
                                nperf:= nperf+1
                         end
                   else  if soma > numero
                         then  begin  writeln(' Número Abundante');
                                      nabund:= nabund+1
                               end
                         else  begin  writeln(' Número Reduzido');
                                      nred:= nred+1
                               end
            end;

       writeln;
       writeln('Total de', nperf, 'Números Perfeitos');
       writeln('Total de', nabund, 'Números Abundantes');
       writeln('Total de', nred, 'Números Reduzidos')
end.
```

Exercício: Incluir também a classe dos Números Primos.