Métodos de Ordenamentos:

Borbulhamento (BubbleSort)

Estratégia: Os elementos de um vector a[1..n] estão dispostos por ordem não decrescente quando,

$$\forall i \in [1..n-1] \Rightarrow a[i] \leq a[i+1].$$

Basta portanto identificar todos pares de elementos contíguos tais que a[i] > a[i+1] e trocá-los.

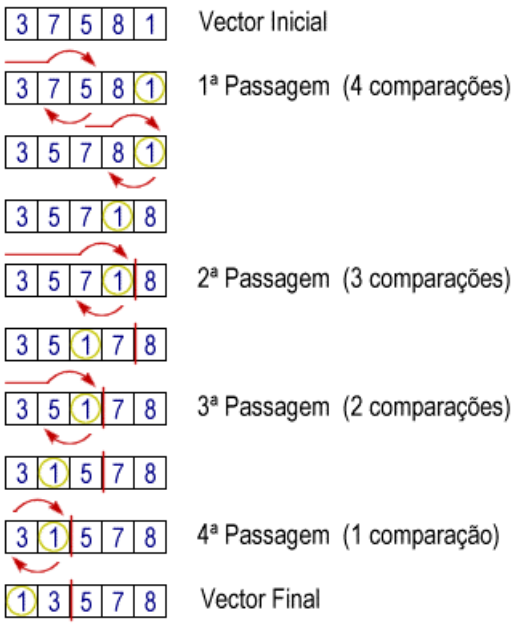
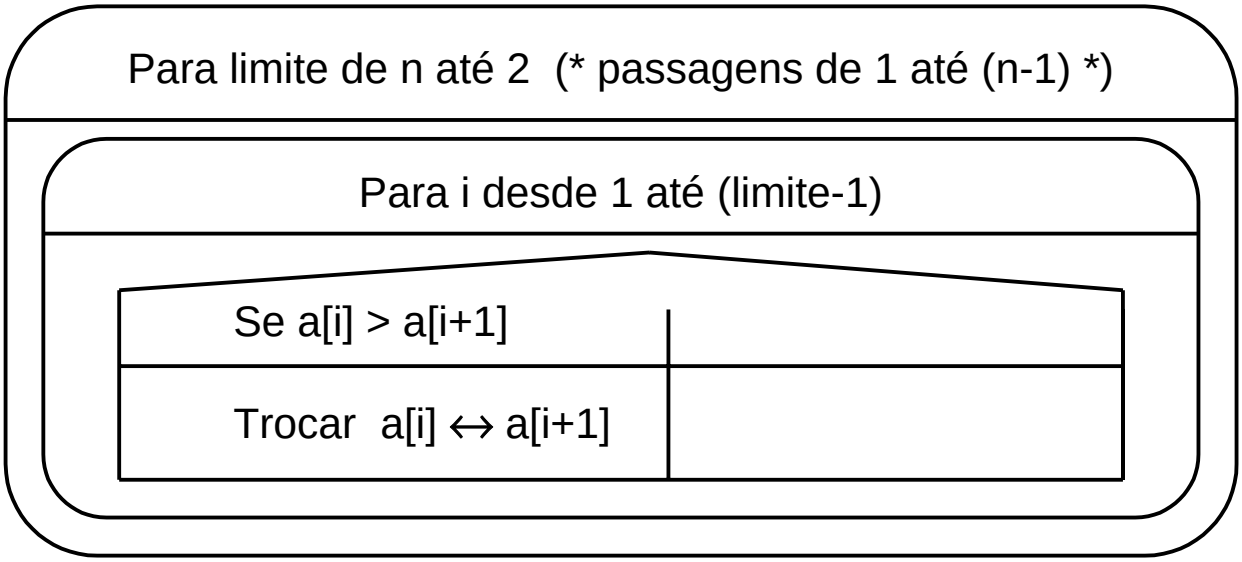


Diagrama de Estrutura:



1ª Versão:

```
(* Dado vector a[1..n] *)
for  limite:= n downto 2 do
     for  i:= 1 to limite-1 do
          if a[i] > a[i+1]
          then begin  aux:= a[i];
                      a[i]:= a[i+1];
                      a[i+1]:= aux
               end;
(* Resultado vector a[1..n] por ordem não decrescente *)
```

- **O BubbleSort é o mais simples dos Métodos de ordenamento, mas também o menos eficiente.**
- **Admite contudo vários melhoramentos e é também uma boa base para a construção de métodos mais elaborados.**

2ª Versão:

- **O melhoramento mais óbvio do BubbleSort consiste em parar o processo logo que se detecte que, ao longo de uma passagem, não foram efectuadas quaisquer trocas.**

```
var houvetrocas : boolean;
    ...
    limite:= n+1; houvetrocas:= true;
    while  houvetrocas do
           begin  limite:= limite-1;
                  houvetrocas:= false;
                  for  i:= 1 to limite-1 do
                       if a[i] > a[i+1]
                       then  begin  aux:= a[i];
                                    a[i]:= a[i+1];
                                    a[i+1]:= aux;
                                    houvetrocas:= true
                             end
           end;
```

Selecção Linear

Estratégia: Uma simples observação do comportamento do BubbleSort revela que, ao fim de cada passagem, apenas o último elemento de cada sub-vector é colocado na posição correcta. O método da Selecção Linear consiste em **seleccionar o maior elemento** de cada sub-vector e **trocá-lo com o último**.

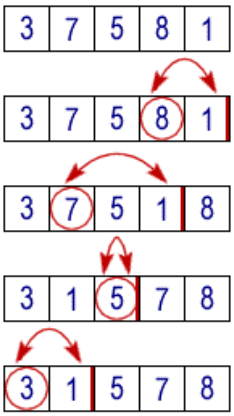
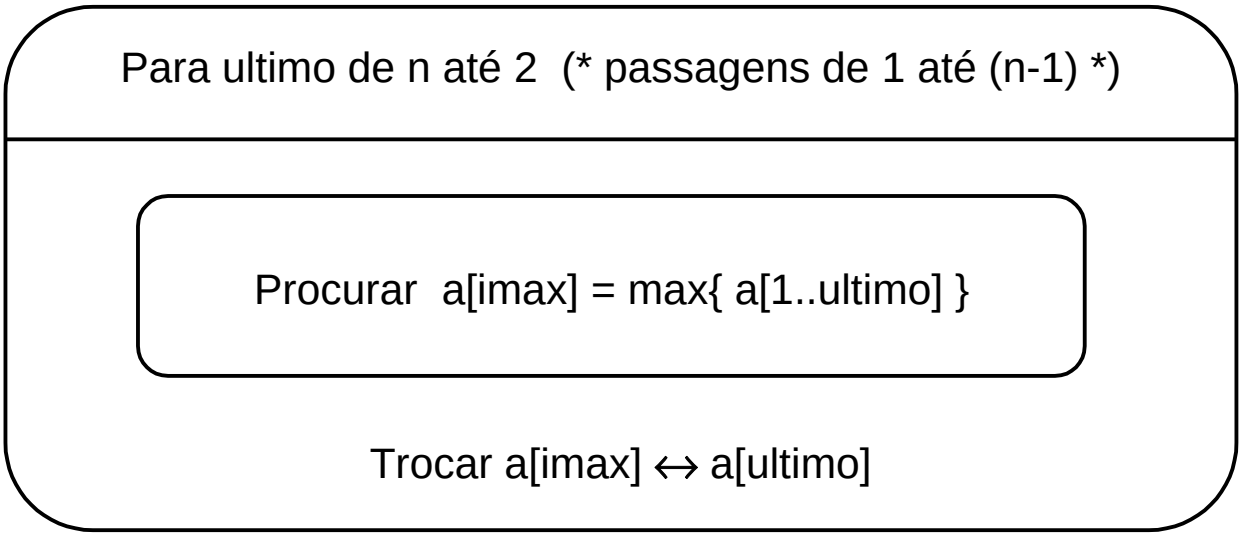


Diagrama de Estrutura:



1ª Versão:

```
...
for ultimo:= n downto 2 do
    begin (* Procurar o maior elemento de a[1..ultimo] *)
          imax:= 1;
          for i:= 2 to ultimo do
              if a[i] > a[imax]
              then imax:= i;
          (* Trocar o maior com o último *)
          aux:= a[imax];
          a[imax]:= a[ultimo];
          a[ultimo]:= aux
    end;
...
```

Versões alternativas:

- **Procurar o menor e trocá-lo com o primeiro.**
- **Procurar o menor e também o maior e trocá-los, respectivamente, com o primeiro e com o último.**

Observação:

Borbulhamento

```
for limite:= n downto 2 do
    for i:= 1 to limite-1 do
        if a[i] > a[i+1]
        then trocar(a[i], a[i+1]);
```

Número de Comparações =
Número Máximo de Trocas =
n (n-1) / 2

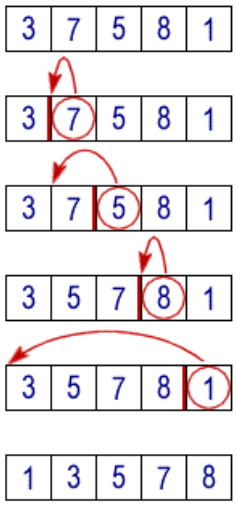
Selecção Linear

```
for ultimo:= n downto 2 do
    begin imax:= 1;
          for i:= 2 to ultimo do
              if a[i] > a[imax]
              then imax:= i;
          trocar(a[imax], a[ultimo])
    end;
```

Número de Comparações = n (n-1) / 2
Número Constante de Trocas = n-1

Ordenamento por Inserção Sequencial

Estratégia: Se um sub-vector (formado pelos primeiros elementos do vector dado) já estiver ordenado, basta inserir ordenadamente o elemento seguinte no sub-vector.



Algoritmo:

```
(* Dado vector a[1..n] *)
(* O sub-vector a[1..1] está ordenado*)

for fim:= 1 to n-1 do
    begin (* Inserir a[fim+1] no sub-vector ordenado a[1..fim] *)
          if a[fim+1] < a[fim]
          then  begin (* Procurar lugar em a[1..fim] *)
                      k:= fim;
                      while (k>=1) and (a[k] > a[fim+1]) do
                            begin a[k+1]:= a[k];
                                  k:= k-1
                            end;
                      (* Aqui (k=0) ou (a[k] <= a[fim+1]) *)
                      a[k+1]:= a[fim+1]
                end
          (* Caso contrário, a[fim+1] >= a[fim] *)
    end;
(* Resultado vector a[1..n] por ordem não decrescente *)
```

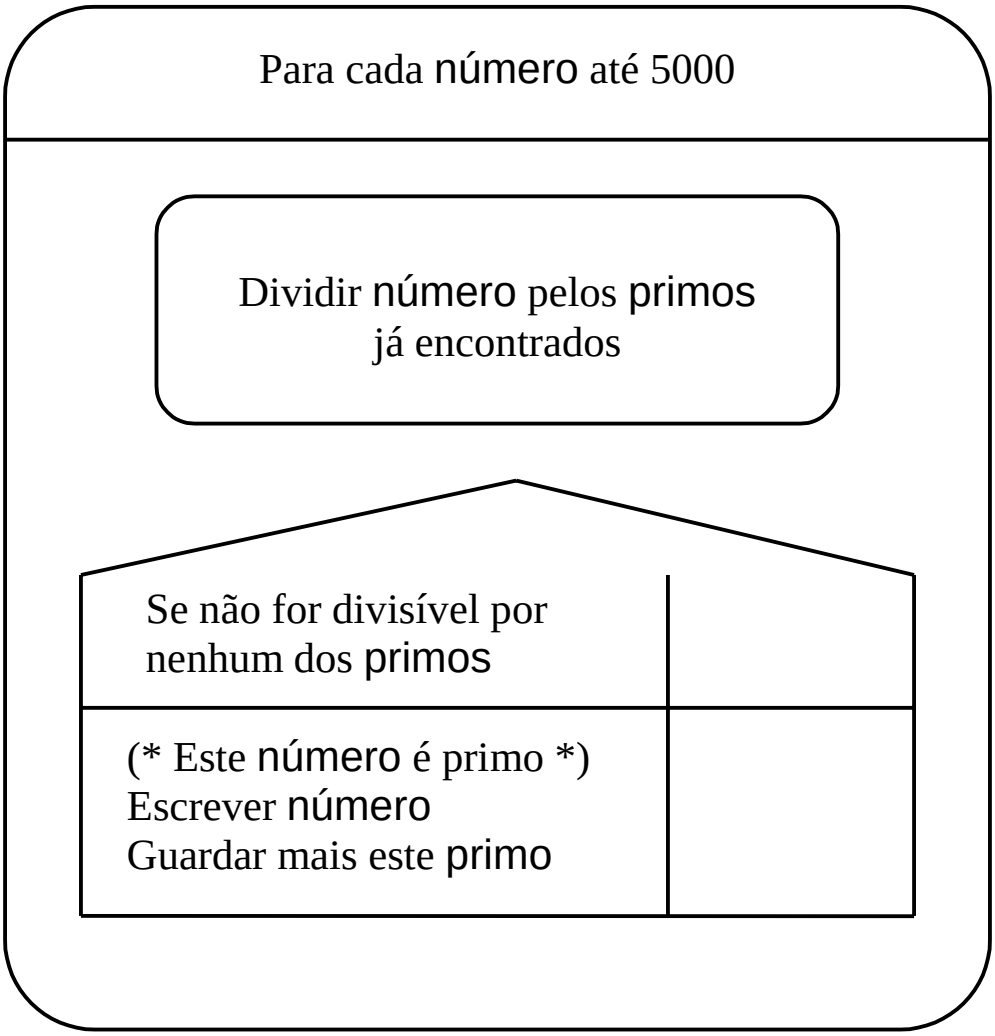
Mais alguns exemplos com Números Primos:

Problema: Listar os Números Primos até 5000.

Estratégia:

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

Abordagem:



Alguma Teoria dos Números:

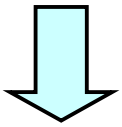
- **O primeiro número primo é o 2.**
- **Nenhum número par, além do 2, pode ser primo.**
- **Se um número não tiver nenhum divisor menor ou igual a $\sqrt{numero}$, então é primo.**
- **Quantos serão os números primos até 5000? $\pi(5000)=?$**

  Aproximação de Legendre: $\pi(n) = n / (\ln n - 1.08366)$
- **Quantos primos será necessário guardar, para testar os números até 5000?**

  $\pi(\sqrt{5000}) \approx 22$

Continuando…

Dividir número pelos primos
já encontrados



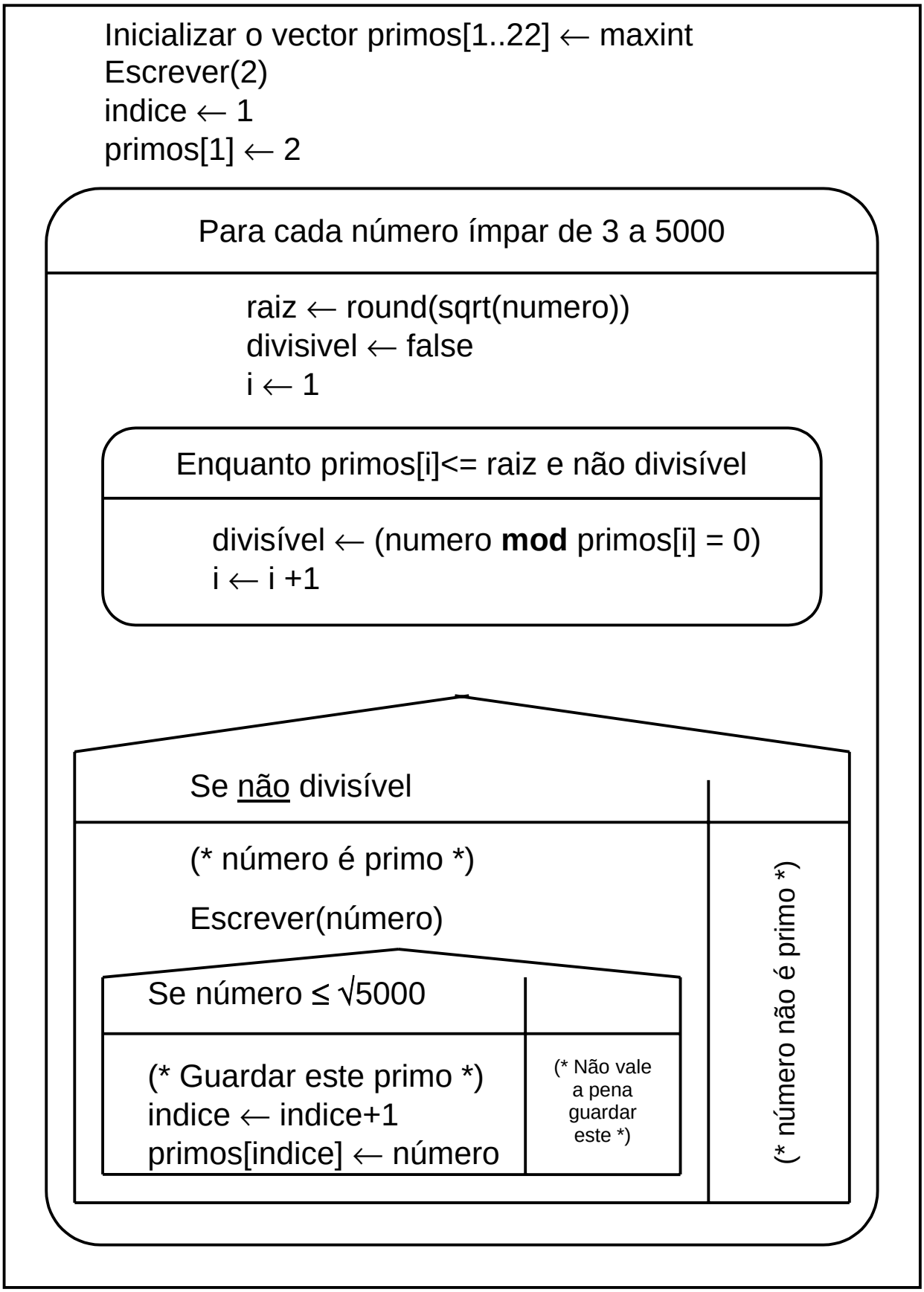
divisível ← false

Para cada primo ≤ √número e
enquanto não for divisível

(* Dividir o número por este primo *)

divisível ← ((número **mod** primo) = 0)

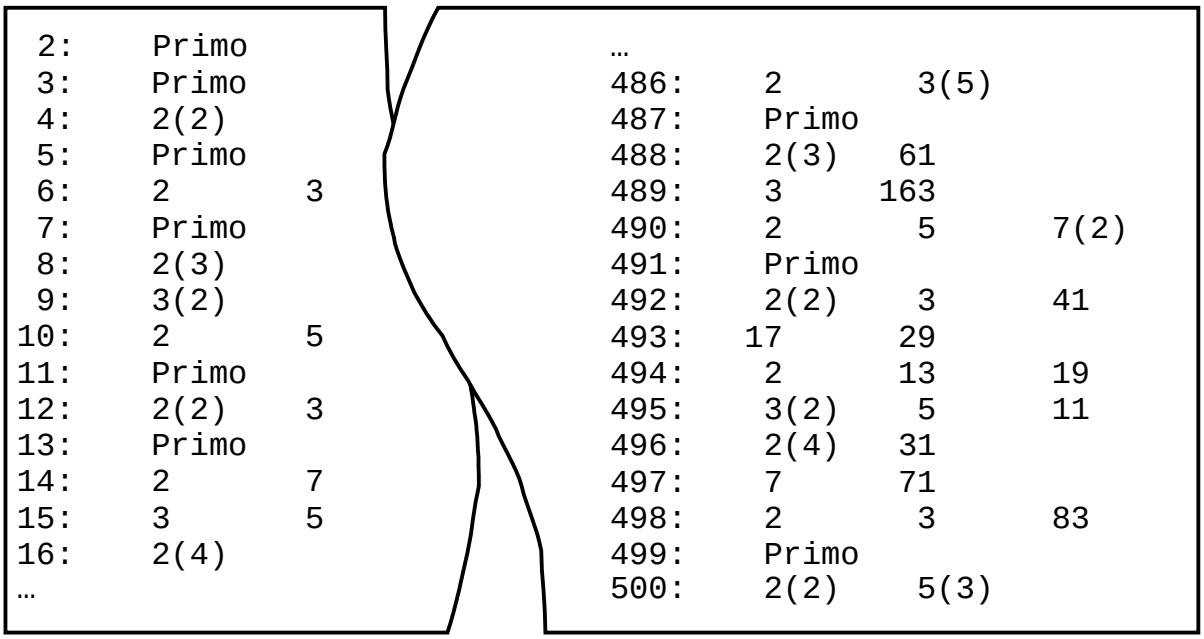
Diagrama de Estrutura:



Programa:

```
program primos5000 (output);
(* Este programa imprime os numeros primos ate 5000 *)
const n = 5000;
      raizden = 71;
      lim = 22; (* Aproximacao de Legendre para pi(sqrt(5000)) *)
var  primos : array[1..lim] of integer; (* Lista de numeros primos *)
     i, raiz, indice, numero : integer;
     divisivel : boolean;

begin (* Inicialização 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 n do
           if odd(numero)
           then begin  (* Dividir pelos primos ja encontrados,
                          ate raiz de numero ou encontrar um divisor *)
                       raiz:= round(sqrt(numero));
                       i:= 1;
                       divisivel:= false;
                       while  (primos[i] <= raiz) and not divisivel do
                              begin divisivel:= numero mod primos[i] = 0;
                                    i:= i + 1
                              end;

                       if not divisivel
                       then begin (* numero é primo *)
                                  writeln(numero);
                                  if numero<= raizden
                                  then begin (* Acrescenta-lo à lista *)
                                             indice:= indice+1;
                                             primos[indice]:= numero
                                       end
                                  (* Senao nao vai ser necessario *)
                            end
                end
end (* primos5000*) .
```

Problema: (ver Cap.III-Exs. Págs. 11-12)
Decompor os números de **2** a **n** em factores primos, indicando os respectivos graus de divisibilidade.



```
 2:    Primo
 3:    Primo
 4:    2(2)
 5:    Primo
 6:    2       3
 7:    Primo
 8:    2(3)
 9:    3(2)
10:    2       5
11:    Primo
12:    2(2)    3
13:    Primo
14:    2       7
15:    3       5
16:    2(4)
…
```

```
…
486:    2       3(5)
487:    Primo
488:    2(3)   61
489:    3     163
490:    2       5       7(2)
491:    Primo
492:    2(2)    3       41
493:   17      29
494:    2      13       19
495:    3(2)    5       11
496:    2(4)   31
497:    7      71
498:    2       3       83
499:    Primo
500:    2(2)    5(3)
```

Programa:

```
program factoresprimos500 (output);
(* Decomposicao dos numeros ate 500 em factores primos *)
const n = 500;
      lim = 56; (* Aproximacao de Legendre para pi(500/2) *)

var  primos : array[1..lim] of integer; (* Lista de numeros primos *)
     i, indice, conta, numero, num : integer;
     novoprimo : boolean;

begin for  i:= 1 to lim do
           primos[i]:= maxint;

      (* O primeiro numero primo *)
      indice:= 1;
      primos[1]:= 2;
      writeln(2 : 10, ':', 'Primo' : 9);
```

```
for  numero:= 3 to n do
     begin (* Analisar numero *)
           write(numero : 10, ':');
     →     novoprimo:= true;
           num:= numero; (* cópia *)
           i:= 1;
           while  primos[i] <= (numero div 2) do
                  begin (* Tentar dividir por este primo *)
                        conta:= 0;
                        while  (num mod primos[i]) = 0 do
                               begin (* Ir dividindo por este primo *)
                                     conta:= conta+1;
                                     num:= num div primos[i]
                               end;
                        (* Dividiu conta vezes por este primo *)
                        if conta >= 1
                        then begin  write(primos[i] : 5);
                                    if conta > 1
                                    then  write('(', conta : 1, ')')
                                    else  write(' ' : 3)
                             end;
                        (*(conta=0) ⇒ este primo não divide o número*)
                  →     novoprimo:= novoprimo and (conta=0);
                        i:= i + 1
                  end;
     →     if novoprimo
           then  begin write('Primo' : 9);
                       if numero <= (n div 2)
                       then  begin indice:= indice+1;
                                   primos[indice]:= numero
                             end
                 end;
           writeln
     end (* Analisar numero *)
end (* factoresprimos500*) .
```

Nota:

- **Utilização da variável lógica novoprimo:**

  O numero é um novoprimo sse (conta = 0) para todas as divisões por primos[i] tentadas.

Exercícios:

- **Legendre** estabeleceu uma aproximação para o **número de primos existentes entre 2 e n**:

  $$\pi(n) \approx n / (\ln n - 1.08366)$$

  Compare o valor aproximado assim obtido com o valor exacto de $\pi(n)$, Para n = 100, 200, 300, …, 4900, 5000.

- **Goldbach** conjecturou que **todo o número par (≥4) é a soma de dois números primos**. Verifique esta conjectura até um dado valor de n.

- Diz-se que dois números primos são **gémeos** quando diferem de 2 unidades. (p.ex.: 11 e 13, 17 e 19). Conjectura-se que a existência de **uma infinidade de pares de primos gémeos**. Procure os pares de primos gémeos até um dado n.

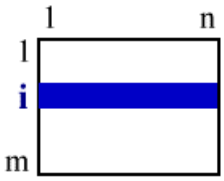
Variáveis com dois índices, Tabelas Bidimensionais:

Operações básicas em matrizes:

```
const max = 20;
type   matriz = array [1 .. max, 1 .. max] of real;
var   a, b, c : matriz;
      m, n, p, i, j, k : 1 .. max;
```

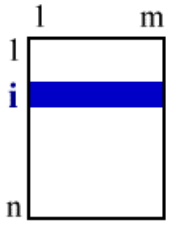
- **Escrever a matriz a[1..m, 1..n] (na sua forma habitual)**

```
for i:= 1 to m do
    begin (* Escrever linha a[i, 1..n] *)
          for j:= 1 to n do
              write( a[i, j] );
          writeln
    end;
```



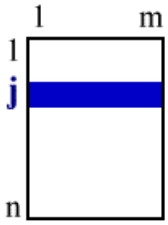
- **Escrever a matriz transposta de a[1..m, 1..n]**

```
for i:= 1 to n do
    begin (* Escrever linha a[i, 1..m] *)
          for j:= 1 to m do
              write( a[j, i] );
          writeln
    end;
```



ou:

```
for j:= 1 to n do
    begin (* Escrever linha a[j, 1..m] *)
          for i:= 1 to m do
              write( a[i, j] );
          writeln
    end;
```



- **Calcular o Traço da matriz quadrada a[1..n, 1..n]**

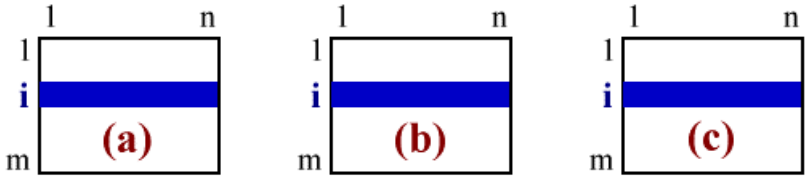
```
traco:= 0;
for i:= 1 to n do
    traco:= traco + a[i,i];
```

- **Somar as matrizes a[1..m, 1..n] e b[1..m, 1..n]:**

c[1..m, 1..n] ← a[1..m, 1..n] + b[1..m, 1..n]

```
for i:= 1 to m do
    for j:= 1 to n do
        c[i, j]:= a[i, j] + b[i, j];
```

- **Escrever as três matrizes, na forma:**



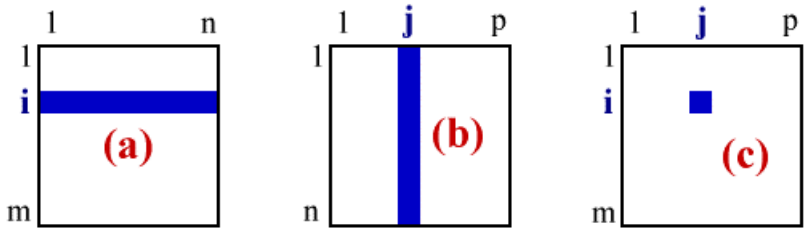

```
for i:= 1 to m do
    begin for j:= 1 to n do
              write(a[i,j] :8:2);
          write(' ' :5);
          for j:= 1 to n do
              write(b[i,j] :8:2);
          write(' ' :5);
          for j:= 1 to n do
              write(c[i,j] :8:2);
          writeln
    end;
```

- **Multiplicar as matrizes a[1..m, 1..n] e b[1..n, 1..p]:**

c[1..m, 1..p] ← a[1..m, 1..n] × b[1..n, 1..p]

$$\forall i \in [1..m], \forall j \in [1..p] : C_{ij} \leftarrow \sum_{k=1}^{n} (A_{ik} \cdot B_{kj})$$

(Cada um dos m×p elementos c[i,j] da matriz produto é o **produto escalar** da linha a[i, 1..n] pela coluna b[1..n, j])




```
for i:= 1 to m do
    for j:= 1 to p do
        begin soma:= 0;
              for k:= 1 to n do
                  soma:= soma + a[i, k] * b[k, j];
              c[i, j]:= soma
        end;
```

Problema: Construir o Triângulo de Pascal de grau n.

Ex.: n = 5

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10  10  5   1
```

Nota:

$(a+b)^0 = 1$

$(a+b)^1 = 1\ a + 1\ b$

$(a+b)^2 = 1\ a^2 + 2\ a\ b + 1\ b^2$

$(a+b)^3 = 1\ a^3 + 3\ a^2\ b + 3\ a\ b^2 + 1\ b^3$

$(a+b)^4 = 1\ a^4 + 4\ a^3\ b + 6\ a^2\ b^2 + 4\ a\ b^3 + 1\ b^4$

...

Estratégia: Utilizar uma matriz auxiliar T[0..n, 0..n],

1	0	0	0	0	0
1	1	0	0	0	0
1	2	1	0	0	0
1	3	3	1	0	0
1	4	6	4	1	0
1	5	10	10	5	1

Onde $T[i,j] \leftarrow T[i-1,j-1] + T[i-1,j]$, $\forall\ i \in [1..n]$, $\forall\ j \in [1..n]$

Programa:

```
program TriangulodePascal(input,output);

const  max = 10;
var   T : array[0..max, 0..max] of integer;
      i, j, n : integer;

begin  writeln('Qual a dimensão pretendida?');
       readln(n);

       (* Anular todos os elementos da matriz *)
       for  i:= 0 to n do
            for  j:= 0 to n do
                 T[i,j]:= 0;

       (* Colocar 1's ao longo da primeira coluna da matriz *)
       for  i:= 0 to n do
            T[i,0]:= 1;

       (* Constriur o resto da matriz *)
       for  i:= 1 to n do
            for  j:= 1 to n do
                 T[i,j]:= T[i-1,j-1] + T[i-1,j];

       (* Escrever a matriz na forma de um Triângulo de Pascal *)
       page(output);
       writeln;
       for  i:= 0 to n do
            begin  (* Assumindo um ecran de 80 colunas *)
                   (* Deixar espaços de modo a centrar *)
                   write(' ': 40 - 3*i);
                   for  j:= 0 to i do
                        write(T[i,j]: 3, ' ': 3);
                   writeln;writeln
            end

end (* TriangulodePascal *).
```