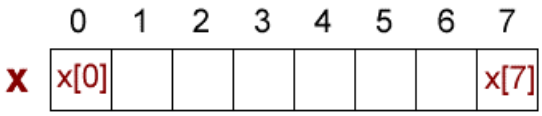
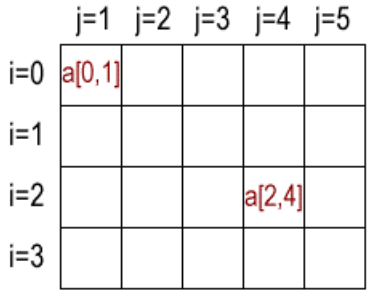
Capítulo V :
Um Tipo Estruturado de Dados: o “array”

Ex1: Vector, Variável com um índice, Tabela unidimensional.



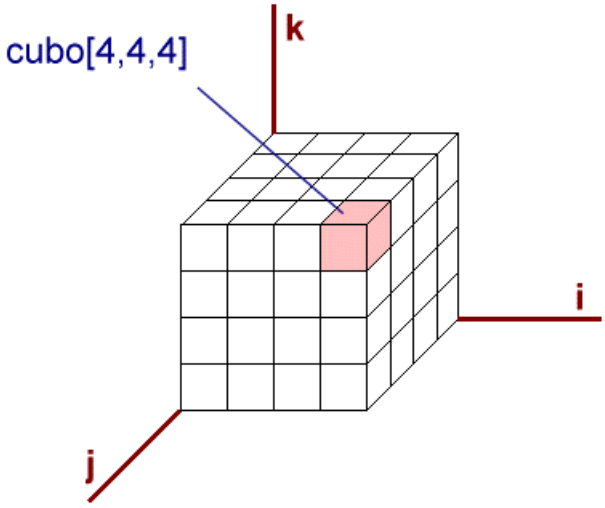
var x : **array** [0..7] **of** real;

Ex2: Matriz, Variável com dois índices, Tabela bidimensional.



var a : **array** [0..3, 1..5] **of** real;

Ex3: Variável com três índices, Tabela tridimensional.



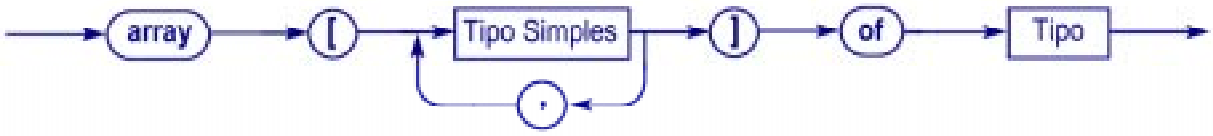
var cubo : **array** [1..4, 1..4, 1..4] **of** real;

...

Variável com n índices, Tabela n-dimensional.

(O número máximo de dimensões depende do compilador Pascal e do computador utilizado)

Tipo:



índice(s)

- **O índice é de tipo escalar, não real, nem todo o domínio dos inteiros, nem todo o domínio dos caracteres (só subdomínios).**

Exs:

```
type  cores = (branco, amarelo, verde, azul, vermelho, preto);
      vector = array [1..100] of real;
      matriz = array [1..10, 1..10] of integer;

var   x, y : vector;
      a, b, c : matriz;
      nome : array [1..50] of char;
      contaletras : array ['a'.. 'z'] of integer;
      bandeira : array [cores] of boolean;
      cubo = array [1..3, 1..3, 1..3] of cores;
```

- **O tipo dos elementos é qualquer (mesmo outro array).**

Ex:

```
var  tabuleiro : array [1..8] of  array [1..8] of figuras;
```

… o que é equivalente a :

```
var  tabuleiro : array [1..8,1..8] of figuras;
```

Exemplo: Operações elementares sobre vectores em $\mathbb{R}^n$.

```
const n = 3; (* Dimensão do espaço *)
type  vector = array [1..n] of real;

var   i : 1..n;
      k, norma, prodesc : real;
      x, y, z : vector;
```

- **Soma de vectores, z ← x + y:**

```
for  i:= 1 to n do
     z[i] := x[i] + y[i];
```

- **Produto do vector x por um escalar k:**

```
for  i:= 1 to n do
     x[i] := k * x[i];
```

- **Produto escalar: z ← x . y**

```
prodesc := 0;
for  i:= 1 to n do
     prodesc := prodesc + x[i] * y[i];
```

- **Norma do vector x:**

```
norma := 0;
for  i:= 1 to n do
     norma := norma + sqr(x[i]);
noma := sqrt(norma);
```

- …

Problema: (Ver Cap. IV, pág. 9) O ficheiro origem contem um texto cujas palavras estão separadas por, um ou mais, espaços. Pretende-se saber **qual é a maior palavra** que nele ocorre.

(Alerações à versão anterior)

```
...
type  palavra = array [1..50] of char;
var   esta, maior : palavra;
...

                                ...
                                (* Início de palavra *)
                                (* Contar e registar letras desta palavra *)
                                comp:= 0;
                                while  c <> ' '  do
                                        begin  comp:= comp + 1;
                                               esta[comp]:= c;
                                               read(origem, c)
                                        end;
                                (* Fim de palavra *)
                                (* Actualizações *)
                                if comp > compmax
                                then  begin  compmax:= comp;
                                             (* Guardar esta palavra *)
                                             for  n:=1 to comp do
                                                  maior[n]:= esta[n]
                                      end
                          end; (* Fim de linha *)
                    readln(origem)
            end; (* Fim do texto *)

    write('Maior palavra do texto, com', compmax, ' letras = ');
    for n:= 1 to compmax do
        write(maior[n]:1);
    writeln
end.
```

Problema: Cálculo do valor de um Polinómio

Dados: $x \in \mathbb{R}$; $\{a_0, a_1, a_2, \ldots, a_n\}$

Resultado: $P_n(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n$

Algoritmo trivial:

```
polinomio:= 0;
for i:=0 to n do
    begin termo:= a[i];
          for k:=1 to n-i do
              termo:= termo * x;
          polinomio:= polinomio + termo
    end;
```

Método de Horner:

Exemplo para **n=3**:

$$P_3(x) = a_0 x^3 + a_1 x^2 + a_2 x + a_3 \qquad \text{(6 mult.)}$$

$$= ((a_0 x + a_1) x + a_2) x + a_3 \qquad \text{(3 mult.)}$$

Basta portanto calcular:

$$\text{polinomio} \leftarrow a_0$$
$$\text{polinomio} \leftarrow a_0 x + a_1$$
$$\text{polinomio} \leftarrow (a_0 x + a_1) x + a_2$$
$$\text{polinomio} \leftarrow ((a_0 x + a_1) x + a_2) x + a_3$$

ou seja,

$$\textbf{polinomio} \leftarrow \textbf{polinomio} * x + a_i \quad (i = 0, 1, \ldots, n)$$

Programa:

```
program Horner (input, output);
const dimmax = 100;
type   indice = 0..dimmax;
var   i, n : indice;
      x, polinomio : real;
      a : array [indice] of real;

begin  writeln('Qual é a dimensão?');
       readln(n);
       writeln('Quais são os coeficientes polinomiais?');
       for  i:= 0 to n do
            readln(a[i]);
       writeln('Para que ponto deseja calcular o polinómio?');
       readln(x);

       polinomio:= a[0];
       for  i:= 1 to n do
            polinomio:= polinomio * x + a[i];

       writeln('Valor do polinomio nesse ponto = ', polinomio)
end.
```

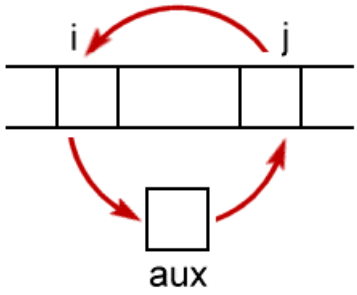
Nota: O problema pode ser resolvido utilizando apenas uma variável simples a, onde são registados os sucessivos valores dos coeficientes polinomiais.

```
       ...
       readln(a);
       polinomio:= a;

       for  i:= 1 to n do
            begin   readln(a);
                    polinomio:= polinomio * x + a
            end;
```

Operações básicas em vectores:

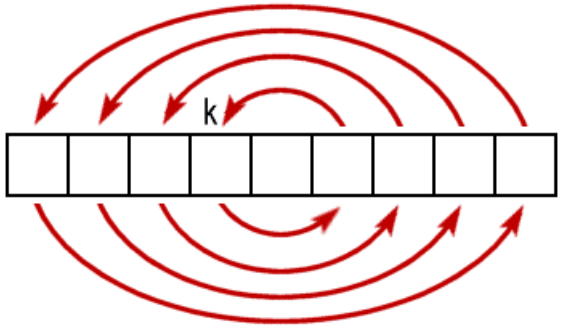
```
const n = 100;
type   vector = array [1 .. n] of integer;
var   a : vector;
```

• Troca dos valores de dois elementos



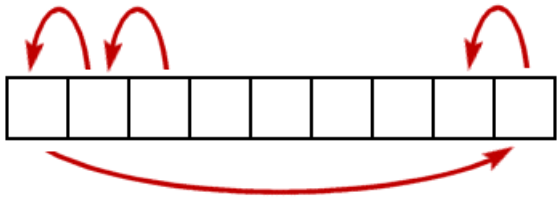

```
aux := a[i];
a[i] := a[j];
a[j] := aux;
```

• Inversão (ou Rotação)



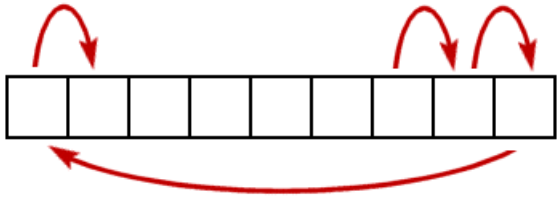

```
k:= n div 2;
for  i:=1 to k do
     begin  aux := a[i];
            a[i] := a[n-i+1];
            a[n-i+1] := aux
     end;
```

- **Permutação Circular à Esquerda**



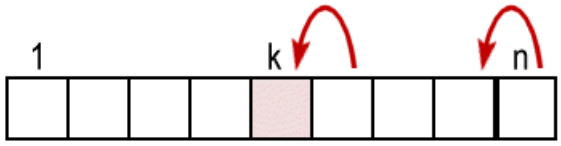
```
aux := a[1];
for i:=1 to n-1 do
    a[i] := a[i+1];
a[n] := aux;
```

- **Permutação Circular à Direita**



```
aux := a[n];
for i:=n downto 2 do
    a[i] := a[i-1];
a[1] := aux;
```

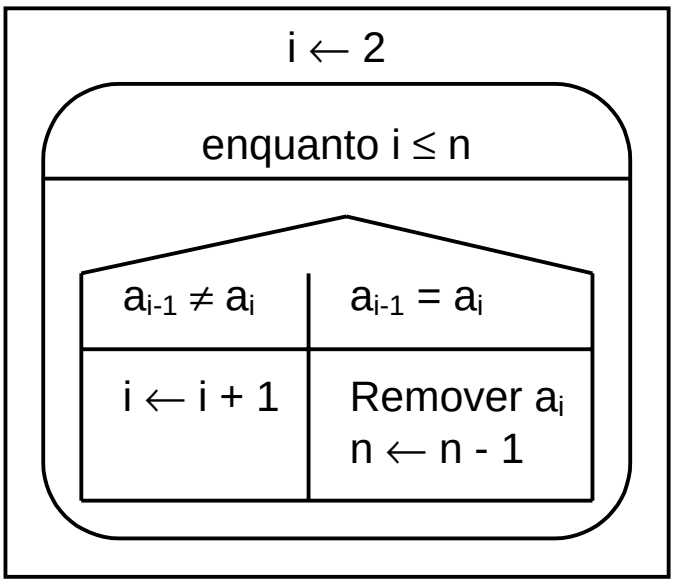
- **Remoção do elemento de ordem k (1 ≤ k ≤ n)**

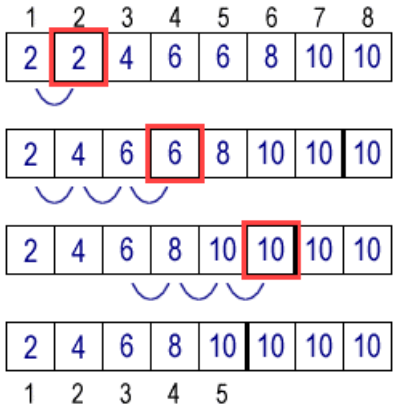



```
for i:=k to n-1 do
    a[i] := a[i+1];
n := n-1;
```

Problema: **Dado um vector de n elementos inteiros, dispostos por ordem crescente, remover os elementos repetidos.**

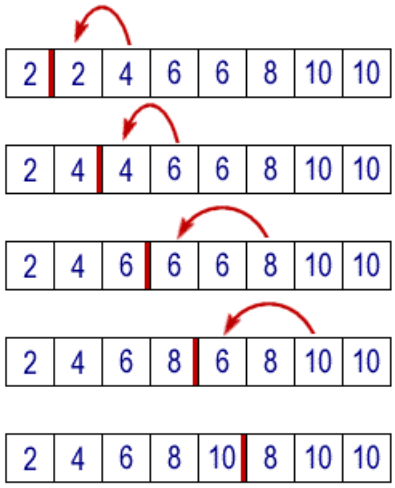
1ª Versão:






```
...
i := 2;
while  i<= n do
        if a[i-1] <> a[i]
        then i := i+1
        else  begin  (* Remover a[i] *)
                     for k:=i to n-1 do
                         a[k] := a[k+1];
                     n := n-1
              end;
...
```

2ª Versão :
(Com um único ciclo e o mínimo número de deslocamentos)




```
...
fim := 1; (* Número de elementos distintos *)
k := 1;
while  k < n do
        begin  k := k+1;
                if a[fim] <> a[k]
                then  begin  (* Mais um elemento distinto *)
                                fim := fim+1;
                                a[fim] := a[k]
                        end
        end;
(* Aqui k = n *)

n:= fim;
...
```

- **Podia ter sido usado um ciclo for.**

Pesquisa de um elemento num vector:

Verificar se um dado valor pertence a um vector e, caso pertença, qual a sua posição.

1ª Versão:

```
…
indice:= 0; (* Fora do domínio dos índices *)
for  i:=1 to n do
     if a[i] = valor
     then indice:= i;

if indice = 0
then  writeln(valor, 'Não pertence ao vector.')
else  writeln(valor, 'Está na posição ', indice, ' do vector.');
…
```

- **Caso ocorram vários elementos iguais ao valor pretendido, será identificado o último.**
- **Se se pretender identificar apenas a primeira ocorrência, ou se os elementos forem todos distintos, a pesquisa deverá parar logo que detecte a primeira igualdade.**

2ª Versão (utilização de uma variável lógica):

```
var achou : boolean;     (* achou ≡ (achou o valor no vector) *)
…
    …
    indice:= 0;
    achou:= false;
    while  not achou and (indice<n) do
           begin  indice:= indice+1;
                  achou:= a[indice] = valor
           end;
    if  achou
    then  writeln(valor, 'Está na posição ', indice, ' do vector.')
    else  writeln(valor, 'Não pertence ao vector.');
    …
```

Pesquisa num vector ordenado:

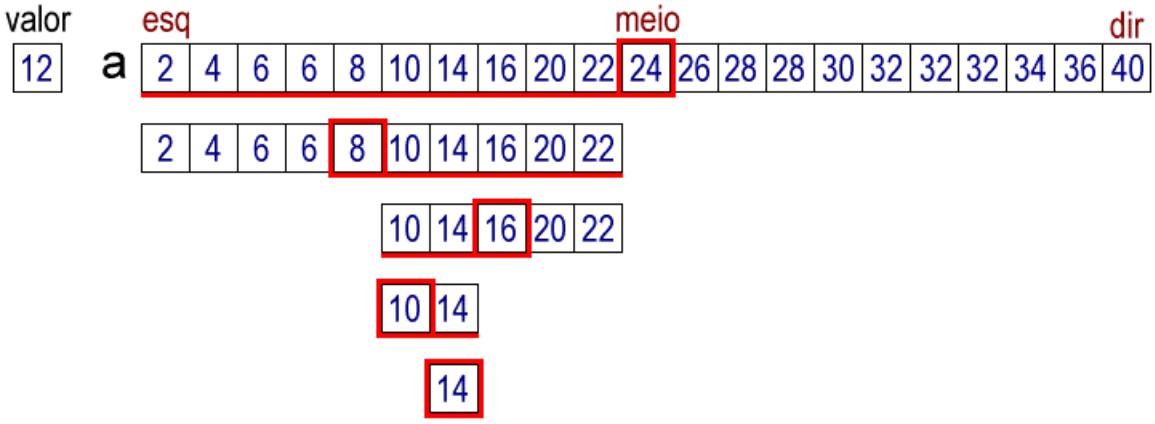
Pesquisa Sequencial ou Linear

- **Se os elementos do vector estiverem dispostos por ordem crescente, a pesquisa deverá parar quando encontre o elemento ou a posição que ocuparia, caso lá estivesse.**


```
var achou : boolean;
(* achou ≡ (achou o valor ou a sua posição, se lá estivesse) *)
…

  …
  indice:= 0;
  achou:= false;
  while  not achou and (indice<n) do
           begin  indice:= indice+1;
                  achou:= a[indice] >= valor
           end;
  (* Aqui achou ou indice = n *)

  (* achou = ( a[indice] >= valor ) *)

  if  achou
  then  if a[indice] = valor
          then writeln(valor, 'Está na posição ', indice, ' do vector.')
          else  writeln(valor, 'Não pertence ao vector.
                                        Deveria estar na posição ', indice)
  else  writeln(valor, 'É superior a qualquer dos', n,
                                                ' elementos do vector.');
  …
```

Pesquisa Binária

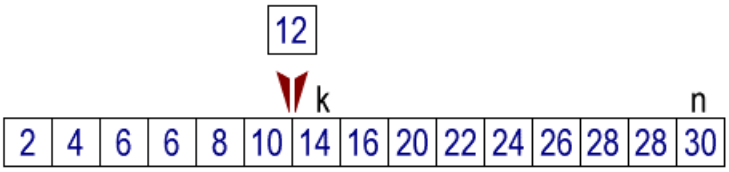
- **Tirando partido do ordenamento do vector, a pesquisa de um valor pode ser feita de um modo muito mais eficiente:**




```
…
var achou : boolean;     (* achou ≡ (achou o valor no vector) *)
…
   …
   esq:= 1; dir:= n;
   achou:= false;
   while  not achou and (esq <= dir) do
            begin  meio:= (esq+dir) div 2;
                   if valor = a[meio]
                   then achou:= true
                   else   if valor < a[meio]
                          then  dir:= meio-1
                          else   esq:= meio+1
            end;
   (* Aqui achou ou esq>dir *)

   (* achou = ( a[meio] = valor ) *)

   if  achou
   then  writeln(valor, 'Está na posição ', meio, ' do vector.')
   else   writeln(valor, 'Não pertence ao vector.');
   …
```

Inserção num vector ordenado:

Dado um vector ordenado, inserir um novo elemento.



1ª Versão:

Procurar o menor k tal que novo< a[k];
Deslocar o sub-vector a[k..n] para a direita;
Colocar o novo elemento em a[k].

```
…
(* Procurar o menor k tal que novo< a[k] *)
k:= 1;
while  (novo >= a[k]) and (k < n) do
          k:= k+1;
(* Aqui  novo<a[k]  ou  k=n *)

if novo<a[k]
then begin  (* Deslocar o sub-vector a[k..n] para a direita *)
                for  i:= n downto k do
                     a[i+1]:= a[i];
                (* Colocar o novo elemento em a[k] *)
                a[k]:= novo
        end
else  (* novo >= a[k]  e  k=n *)
        (* O novo elemento era maior que todos os outros *)
        a[n+1]:= novo;

n:= n+1;
…
```

2ª Versão:

Começando pelo último elemento, ir deslocado para a direita cada elemento, enquanto forem superiores ao novo;
Colocar o novo elemento.

```
…

if novo >= a[n]
then  (* Novo elemento maior que todos os outros *)
      a[n+1]:= novo

else  begin  (* Começando pelo último elemento *)
             k:= n;
             (* Enquanto forem superiores ao novo *)
             while  (k>=1) and (a[k] > novo) do
                    begin  (* Deslocar a[k] para a direita*)
                           a[k+1]:= a[k];
                           (* Analisar outro *)
                           k:= k-1
                    end;
             (* Aqui  k=0  ou  a[k] <= novo *)

             (* Colocar o novo elemento em a[k+1] *)
             a[k+1]:= novo
      end;

n:= n+1;
…
```

Exercício:

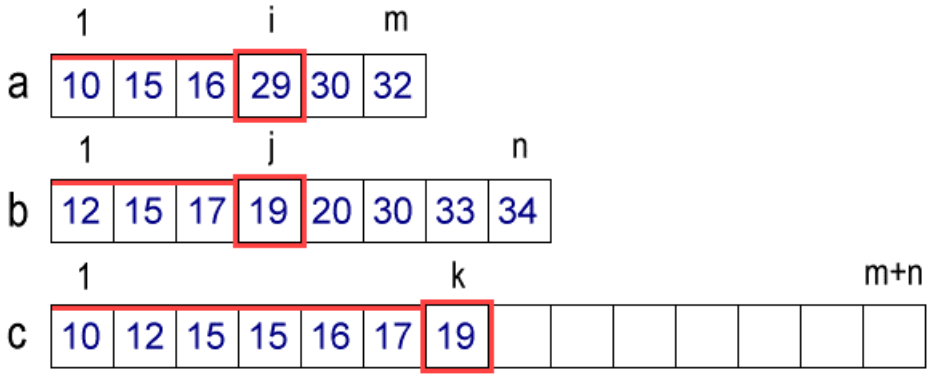
Pesquisar um elemento num vector ordenado. No caso de o elemento não se encontre no vector, inseri-lo ordenadamente.

Problema: **O ficheiro numeros contem números inteiros.**
Ler esses inteiros, colocando-os por ordem crescente num vector, à medida que forem sendo lidos.
Guardar o resultado no ficheiro ordenados.

```
program OrdenarFicheiro(numeros, ordenados);
const max=1000;
var  numeros, ordenados : text;
     a : array[1..max] of integer;
     k, n, num : integer;

begin reset(numeros);
      (* Ler o primeiro e construir um vector com um só elemento *)
      read(numeros, num);
      n:= 1; a[1]:= num;

      while not eof(numeros) do
            begin (* Ler outro número *)
                  read(numeros, num);
                  (* Inserir o número no vector de n elementos *)
                  if num >= a[n]
                  then a[n+1]:= num
                  else begin k:= n;
                             while (k>=1) and (a[k] > num) do
                                   begin a[k+1]:= a[k];
                                         k:= k-1
                                   end;
                             a[k+1]:= num
                       end;
                  n:= n+1
            end;

      (* Guardar a lista dos n números já ordenados *)
      rewrite(ordenados);
      for k:= 1 to n do
          writeln(ordenados, a[k])
end.
```

Fusão de vectores ordenados:

Dados dois vectores a[1..m] e b[1..n] de elementos interios, dispostos por ordem crescente, contruir o vector c[1..m+n] com a totalidade dos elementos de a e de b ordenados.




```
i:= 1; j:= 1; k:= 1;
while  (i<=m) and (j<=n) do
        begin (* Copiar o menor para c *)
                if a[i] < b[j]
                then  begin  c[k]:= a[i];
                             i:= i + 1
                      end
                else  begin  c[k]:= b[j];
                             j:= j + 1
                      end;
                k:= k+1
        end;
(* Acabar de copiar o resto de a, se existir *)
for  i:= i to m do
     begin  c[k]:= a[i];
            k:= k+1
     end;
(* Acabar de copiar o resto de b, se existir *)
for  j:= j to n do
     begin  c[k]:= b[j];
            k:= k+1
     end;
```