

TEORIA 10

TEMAS de la CLASE

Estructura de datos ARREGLO

- 1 Definición de tipo de dato Arreglo
- 2 Declaración del tipo de dato Vector
- 3 Operaciones frecuentes en el tipo Vector
- 4 Ejercitación

Arreglos: Motivación



Supongamos que un supermercado dispone de la información de sus 100 productos y desea informar los nombres de los productos cuyo precio supera el promedio de precios del supermercado.



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Ala
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

- ¿Cómo organizaría la información?
- ¿Cómo procesaría los datos para tener el promedio de los precios y compararlo con el precio de cada producto?

Pueden existir diferentes soluciones con los tipos de datos que hemos visto hasta ahora en el curso:

Código
Marca
Nombre
Precio
Stock

Arreglos: Motivación



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Ala
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

a) Ingresar 2 veces el conjunto de datos.

Se ingresan los datos para calcular el promedio de precios y luego volvemos a ingresar los mismos datos, comparando el precio promedio con el precio de cada producto.

ATENCION!! Para los 100 productos, donde cada uno tiene 5 datos nos obliga a leer una vez $100 \times 5 = 500$ datos. Y luego otra vez $100 \times 5 = 500$ datos. En total se leen 1000 datos.

Código
Marca
Nombre
Precio
Stock

Arreglos: Motivación



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Ala
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

b) Usar tantas variables diferentes como productos existen.

Es decir en cada variable guardamos un producto distinto a medida que se lee (en este caso necesitamos 100 variables diferentes). Luego calculamos el promedio y comparamos cada precio con el mismo.

ATENCIÓN!! Esta solución resulta mas compleja a medida que aumenta el número de productos ¿Por qué?

Arreglos: Motivación

A partir de la solución (b) resulta claro que sería conveniente tener una estructura que reúna a todos los productos bajo un único nombre y que a la vez permita diferenciar (y acceder) a los datos de cada uno.

Para resolver estos problemas podemos usar una estructura de datos tipo **ARREGLO**.

Productos



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Ala
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

Productos [1]



Productos [2]



Productos [3]



Productos [4]



Arreglos: Concepto

Un arreglo (ARRAY) es una estructura de datos que permite acceder a cada componente a través de una variable denominada índice, que indica la posición de la componente dentro de la estructura de datos.

Productos



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Ala
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

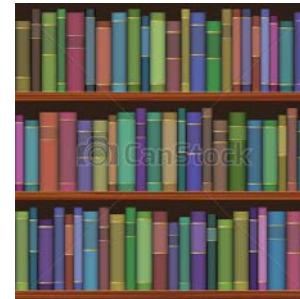
Indice

Arreglos: Ejemplos

Precisamente por ser estática, permite el acceso rápido a sus componentes a través de la variable índice (que tiene que ser de tipo ordinal) y que puede verse como el desplazamiento desde la posición inicial de comienzo de la estructura.



**Un Índice:
vector**

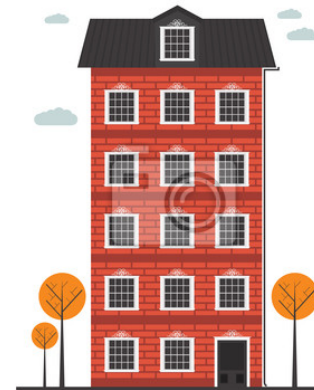


**Dos Índices:
matriz**

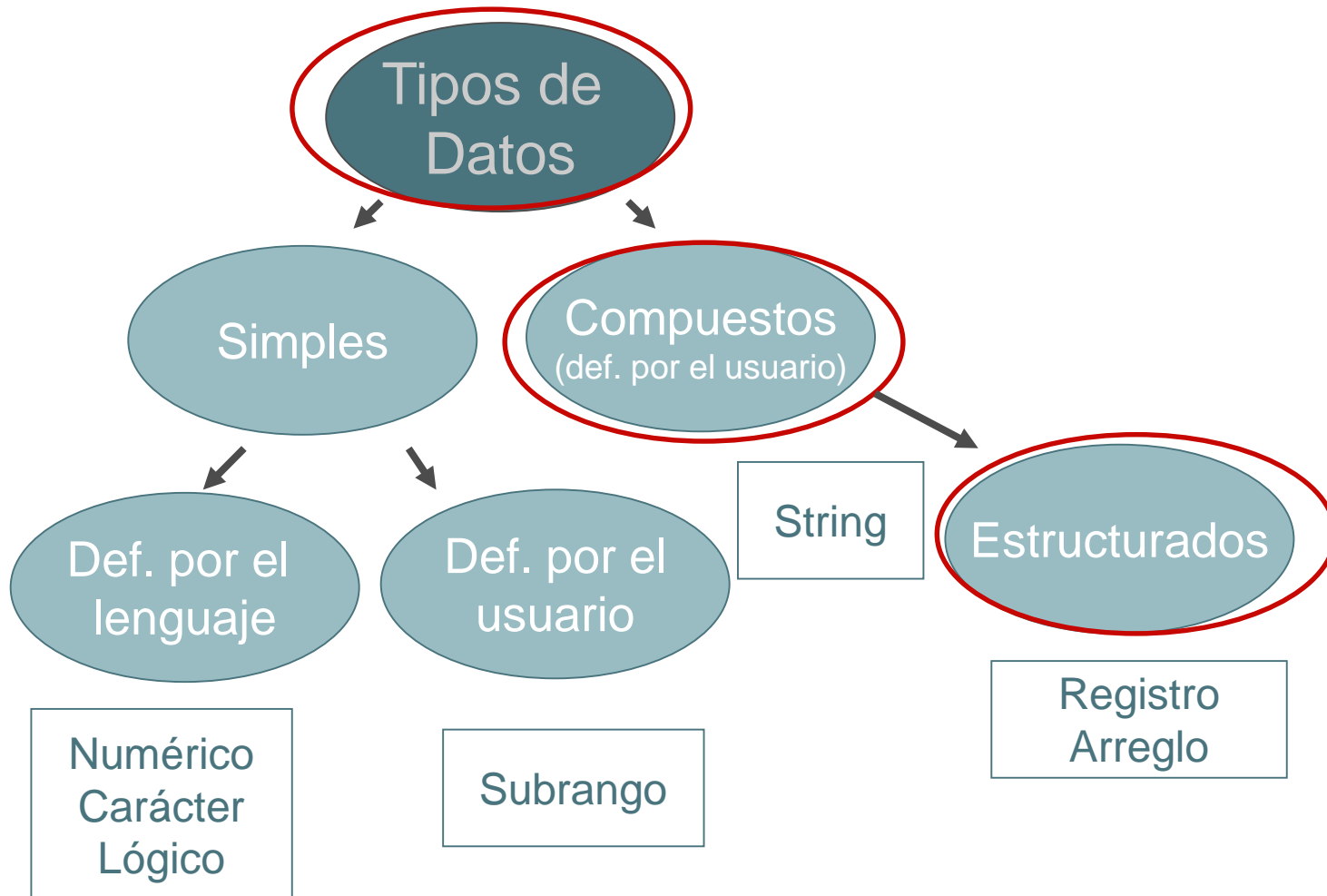
Horarios de Clase

Hora	Lunes	Martes	Miércoles	Jueves	Viernes

✂️ 📅 ✂️ ✂️ ✂️ ✂️ ✂️ ✂️ ✂️ ✂️



Arreglos: Recordemos clasificación...



Arreglos: Definición

Un tipo de dato **Arreglo** es una colección de elementos que se guardan consecutivamente en la memoria y se pueden referenciar a través de índices.

Esta estructura de datos reúne las siguientes características:

- ✓ Todos los elementos son del mismo tipo de datos, por eso es una estructura de datos **homogénea**.
- ✓ Los elementos o componentes pueden recuperarse en cualquier orden, indicando simplemente su posición, por eso es una estructura de datos de **acceso directo**. Como el acceso se hace a través del índice se la denomina también indexada.
- ✓ La memoria ocupada durante la ejecución del programa es fija, por eso se dice que es una estructura de datos **estática**.
- ✓ Dado que cada elemento tiene un elemento que le precede y uno que le sigue, esta estructura se denomina **lineal**.

Arreglos: Tipo Vector

Si tenemos los 100 productos del supermercado

Productos



1000
Leche
SanCor
100
20



1100
Yoghurt
Sancor
200
23



5055
Detergente
Aldo
0
55



2400
Fideos
Matarazzo
35
30



5250
Cerveza
Quilmes
100
50

y los almacenamos en un vector llamado PRODUCTOS de esta forma:

1000 Leche SanCor 100 20	1100 Yoghurt Sancor 200 23	5055 Detergente Aldo 0 55	2400 Fideos Matarazzo 35 30						5250 Cerveza Quilmes 100 50
--------------------------------------	--	---------------------------------------	---	--	--	--	--	--	---

Productos

En la memoria habrá 100 casilleros seguidos conteniendo cada uno los datos de los productos. La cantidad total de memoria depende de la cantidad de componentes y del tipo de dato de cada uno. En este caso serán 100 componentes de 40 bytes cada una → 4000 bytes.

Tipo Vector: Posición y contenido

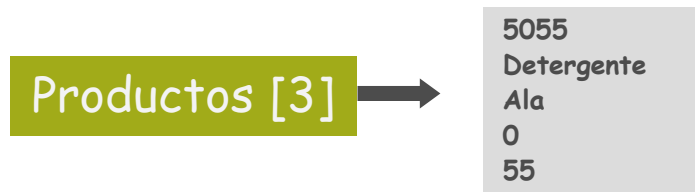
Cuando se trabaja con vectores hay que tener en cuenta que:

Productos

1000 Leche SanCor 100 20	1100 Yoghurt Sancor 200 23	5055 Detergente Ala 0 55	2400 Fideos Matarazzo 35 30						5250 Cerveza Quilmes 100 50
1	2	3							100

Contenido (pointing to the third column)

Posición (pointing to the number 3 in the second row)



- ➡ El valor 3 es la posición ó ubicación dentro de la estructura de datos (índice).
- ➡ Los datos del registro son el contenido de esa posición o ubicación.

Declaración del Tipo Vector en Pascal

Type

```
cadena15 = string [15];
```

```
producto= Record
```

```
    codigo: integer;
```

```
    nombre: cadena15;
```

```
    marca: cadena15;
```

```
    stock: integer;
```

```
    precio: real
```

```
End;
```

```
vectorProductos=array [1..100] of producto;
```

Var

```
    Productos : vectorProductos;
```

Se puede decir que:

✓ La variable **Productos** tiene asociada un área de memoria fija consecutiva que es el lugar donde se almacenará la información de los productos.

✓ La variable **Productos** ocupa 100 posiciones de memoria como lo indica su declaración.

Declaración del Tipo Vector en Pascal

Vector = Array [**índice**] of **tipo_elementos**;

Es posible indexar los elementos por un **índice** que corresponde a cualquier tipo ordinal:

- Entero
- Carácter
- Subrango

¿Por qué?

Los **elementos** de un arreglo pueden pertenecer a cualquier tipo de datos de asignación estática:

- Entero, Real, Lógico, Carácter
- String
- Registros
- Otro arreglo

¿Por qué?

Declaraciones en Pascal - Ejemplos

Const

```
limite=1000;
```

Type

```
periodo = 2000..2015;
```

```
AñosAutos = Array [periodo] of integer;
```

```
Cajas = Array [ 'A' .. 'D' ] of real;
```

```
numeros = array [1..limite] of integer;
```

```
cadena15 = string [15];
```

```
fecha = record
```

```
    día: 1..31;
```

```
    mes: 1..12;
```

```
    año: 1900..2100;
```

```
end;
```

```
articulo = Record
```

```
    codigo: integer;
```

```
    ident: cadena15;
```

```
    marca: cadena15;
```

```
    fechaVenc: fecha;
```

```
    Precio: real
```

```
End;
```

```
todoslosarticulos= Array [1..100] of articulo;
```

Analicemos para cada declaración de variable:

- ✓ ¿Memoria ocupada?
- ✓ ¿Tipo de índice de cada vector?
- ✓ ¿Tipo de los elementos de cada vector?

Var

```
N: numeros;    {1000 elem entero}
```

```
Autos: añosAutos;    {16 elem entero}
```

```
negocio: todoslosarticulos; {100 elem tipo articulo}
```

```
TotalporCaja: cajas; {4 elem real}
```

Arreglos: Operaciones del tipo vector

- Asignación de contenido a un elemento
- Lectura / Escritura
- Recorridos
- Cargar datos en un vector
- Agregar elementos al final
- Insertar elementos
- Borrar elementos
- Buscar un elemento

Operación: Asignación de contenido a un elemento del vector

Const

```
    limite=1000;
```

Type

```
    numeros = array [1..limite] of integer;
```

```
    cadena15 = string [15];
```

producto= Record

```
        codigo: integer;
        nombre: cadena15;
        marca: cadena15;
        stock; integer;
        precio: real
```

End;

```
    vectorProductos = array [1..100] of producto;
```

Var

```
    N: números; i:integer;
    Productos : vectorProductos;
```

Begin

```
    N[6]:= 'a';
    N['b'] := 12;
    i:= 1
    productos[120].precio := 25.5;
    productos[i].stock := 23.5;
```

.....

End.

N [6] := 'a'
¿Es válida? ¿Por qué?

N ['b'] := 12
¿Es válida? ¿Por qué?

productos[120].precio := 25.5
¿Es válida? ¿Por qué?

productos[i].stock := 23.5
¿Es válida? ¿Por qué?

Si se necesita inicializar el stock en 0 de todos los elementos del vector de Productos o el total facturado por cada caja, se puede :

Type

```
rangoCajas = 'A'..'D';  
Cajas = Array [ rangoCajas ] of real;
```

```
cadena15 = string [15];
```

producto= **Record**

```
    codigo: integer;  
    nombre: cadena15;  
    marca: cadena15;  
    stock; integer;  
    precio: real
```

End;

```
vectorProductos = array [1..100] of producto;
```

Var

```
TotalporCaja: Cajas;  
Productos : vectorProductos;  
i : integer; j: rangoCajas;
```

Begin

```
.....  
for i:= 1 to 100 do  
    productos[i].stock:= 0;
```

```
for j:= 'A' to 'D' do  
    TotalporCaja[i]:= 0;
```

```
.....  
End.
```

*En productos, el valor 0 se asigna al campo stock de cada elemento.
En TotalporCaja, el valor 0 se asigna a cada uno de los elementos del vector*

Operación: Asignación de contenido a un elemento del vector

Type

```
cadena15 = string [15];
```

producto= Record

```
    codigo: integer;  
    nombre: cadena15;  
    marca: cadena15;  
    stock: integer;  
    precio: real
```

End;

```
vectorProductos = array [1..100] of producto;
```

Var

```
    Productos : vectorProductos;
```

```
    i : integer; suma: integer;
```

Begin

```
    ....
```

```
    productos[1].precio:= productos[1].precio + 10;
```

```
    productos[3].precio:= 100 + 10;
```

```
    productos[5].precio:= productos[1].precio + productos[3].precio;
```

```
    suma := 0;
```

```
    for i:= 1 to 100 do
```

```
        suma := suma + productos[i].stock;
```

```
    ..... 
```

End.

Operación: Lectura y escritura de los elementos de un vector

Const

```
limite=1000;
```

Type

```
índice = 1..limite;  
numeros = array [índice] of integer;  
cadena15 = string [15];  
producto= Record  
    codigo: integer;  
    nombre: cadena15;  
    marca: cadena15;  
    stock: integer;  
    precio: real  
End;  
vectorProductos=array [1..100] of producto;  
Procedure LeerProducto (Var prod: producto)  
begin  
    Readln (prod.codigo);  
    Readln (prod.nombre);  
    Readln (prod.marca);  
    Readln (prod.stock);  
    Readln (prod.precio);  
end;  
Procedure mostrarProducto (prod: producto);  
begin  
    Writeln (prod.codigo);  
    Writeln(prod.nombre);  
    Writeln (prod.marca);  
    Writeln (prod.stock);  
    Writeln (prod.precio);  
end;
```

Cada componente se trata individualmente.

Var

```
N:números;  
Productos : vectorProductos;  
i : integer; k:índice;
```

Begin

```
for k:= 1 to limite do  
    readln (N[k]);  
  
for k:= 1 to limite do  
    writeln (N[k]);  
  
for i:= 1 to 100 do  
    leerProducto (Productos[i]);  
  
    for i:= 1 to 100 do  
        MostrarProducto (Productos[i]);  
    .....  
End.
```

¿En qué variable queda el resultado de la lectura? ¿Qué muestra el write?

Vector : Operación de Recorrido

La operación de **Recorrido** en un vector consiste en recorrer el vector de manera **total** o **parcial**, para realizar algún proceso sobre sus elementos.

La operación de **Recorrido Total**, implica analizar **todos** los elementos del vector, lo que lleva a recorrer completamente la estructura.

La operación de **Recorrido Parcial**, implica analizar los elementos del vector, **hasta** encontrar aquel que cumple con lo pedido. Puede ocurrir que se recorra todo el vector.

¿Qué estructuras de control conviene utilizar en cada caso?

¿Ejemplos?

Vector : Ejemplo de Recorrido Total



Por ejemplo, si se necesita conocer la cantidad total de productos del supermercado, habrá que realizar un recorrido total que acumule los stocks de cada producto.

Estructura de control

Type

```
cadena15 = string [15];
```

```
producto= Record
```

```
    codigo: integer;
```

```
    nombre: cadena15;
```

```
    marca: cadena15;
```

```
    stock: integer;
```

```
    precio: real
```

```
End;
```

```
vectorProductos=array [1..100] of producto;
```

Var

```
productos: vectorProductos;
```

```
st_total:integer;
```

```
i: integer;
```

begin

```
{el vector contiene datos de 100 productos}
```

```
st_total:= 0;
```

```
{recorrido total del vector}
```

```
For i := 1 to 100 do
```

```
    st_total := st_total + productos[i].stock;
```

```
    writeln ( 'El stok total es: ', st_total);
```

```
end.
```

Vector : Ejemplo de Recorrido parcial



Por ejemplo, se quiere conocer el nombre del primer producto con stock en 0, (seguro existe). Tendremos que recorrer el vector de productos hasta encontrar el primer producto con stock en 0.

*Estructura
de control*

```
Program    stock0;
Type
  cadena15 = string [15];
  producto= Record
    codigo: integer;
    nombre: cadena15;
    marca: cadena15;
    stock: integer;
    precio: real
  End;
  vectorProductos = array [1..100] of producto;
Var
  productos: vectorProductos; i:integer;
begin
  {el vector contiene datos de 100 productos}
  i := 1;
  while (productos[i].stock <> 0) do
    i := i+1;
  writeln('Producto: ', productos[i].nombre);
end.
```

Vector : Ejemplo de Recorrido parcial



¿Qué ocurre si en el ejemplo anterior se cambia la precondición y el producto con stock en 0 podría no existir?

Se debe reemplazar por:

Analizar condición al salir:

```
Program    stock0;
```

```
Type
```

```
cadena15 = string [15];
```

```
producto= Record
```

```
    codigo: integer;
```

```
    nombre: cadena15;
```

```
    marca: cadena15;
```

```
    stock: integer;
```

```
    precio: real
```

```
End;
```

```
vectorProductos = array [1..100] of producto;
```

```
Var
```

```
    productos: vectorProductos; i:integer;
```

```
begin
```

```
    {el vector contiene datos de 100 productos}
```

```
    i := 1;
```

```
    While ( i <= 100) and ( productos[i].stock<>0) do
```

```
        i:= i+1;
```

```
    If i<=100 then writeln ('Producto:',  productos[i].nombre)
        else write ('No existe'
```

```
End:
```

Analicemos la condición del While...

Vector: dimensiones Física y lógica

Cuando se trabaja con vectores se deben hacer consideraciones importantes acerca de algunos aspectos:

➡ **Dimensión Física del vector**

se especifica en el momento de la declaración y determina su ocupación de memoria. La cantidad de memoria no variará durante la ejecución del programa.

¿Por qué?

➡ **Dimensión Lógica del vector**

se determina cuando se cargan contenidos a los elementos del arreglo. Indica la cantidad de posiciones de memoria ocupadas con contenidos cargados desde la posición inicial en forma consecutiva.

Vector: parámetros...

- ¿Qué parámetros deberían recibir y devolver los módulos que tratan con arreglos?
- ¿De qué tipo conviene que sean los parámetros relacionados con arreglos?

Arreglo

*Dimensión
lógica*

¿Parámetro por valor?

¿Parámetro por referencia?

Vector: Cargar datos en un vector

Consideraciones

- ➡ dimF : dimensión física del vector (tamaño especificado en la declaración del vector)
- ➡ dimL : cantidad de elementos en el vector (dimensión lógica).



Se pide cargar valores enteros en un vector de a lo sumo 1000 elementos. La lectura de los valores finaliza con el valor 99.

declaración

Const

```
DimF = 1000;
```


Type

```
vector = Array [ 1..DimF ] of integer;
```

Vector: Cargar datos en un vector

```
Procedure CARGAR ( var num: vector; var dimL: integer );  
  var dato : integer;  
  begin  
    dimL := 0;  
    read (dato);  
    while (dato <> 99) and ( dimL < dimF ) do begin  
      dimL := dimL + 1;  
      num [dimL] := dato;  
      read (dato)  
    end;  
  End;
```

¿Qué retorna el procedimiento?
Observar los parámetros...



dimL -> 6

Supongamos que se leen los valores: 5, 20, 13, 18, 100, 25 y 99

Vector: Agregar un elemento al final

La operación de Agregar un elemento en un vector consiste en incorporar el elemento a continuación del último ingresado, es decir, en la posición siguiente a la indicada en la dimensión lógica.

Esta operación debe verificar que haya lugar en la estructura, es decir que la dimensión lógica sea menor que la dimensión física.

CONSIDERACIONES

- ➡ **dimL** : cantidad de elementos en el vector (dimensión lógica).
- ➡ **dimF** : dimensión física del vector (tamaño especificado en la declaración del vector)
- ➡ **espacio suficiente**: Debe verificarse que $\text{dimL} < \text{dimF}$ (dimensión lógica < dimensión física)
- ➡ El elemento a **agregar** ocupa la **posición dimL+1**.
- ➡ Luego de la operación la **cantidad** de elementos es **dimL+1**.

Vector: Agregar un elemento al final

Observar los parámetros...

```
Procedure AGREGAR (var v: vector; var dimL:integer;
                  elemento: integer; var exito : boolean);

Begin
  {verificar espacio suficiente}
  If (dimL < dimF) then begin
    dimL:= dimL+1; {actualizar cantidad de elementos}
    v [dimL]:= elemento;
    exito := true
  end
  else exito := false;

end;
```

Vector: Insertar un elemento

La operación de Insertar un elemento en un vector consiste en incorporar el elemento en una posición determinada o de acuerdo a un orden impuesto a sus datos.

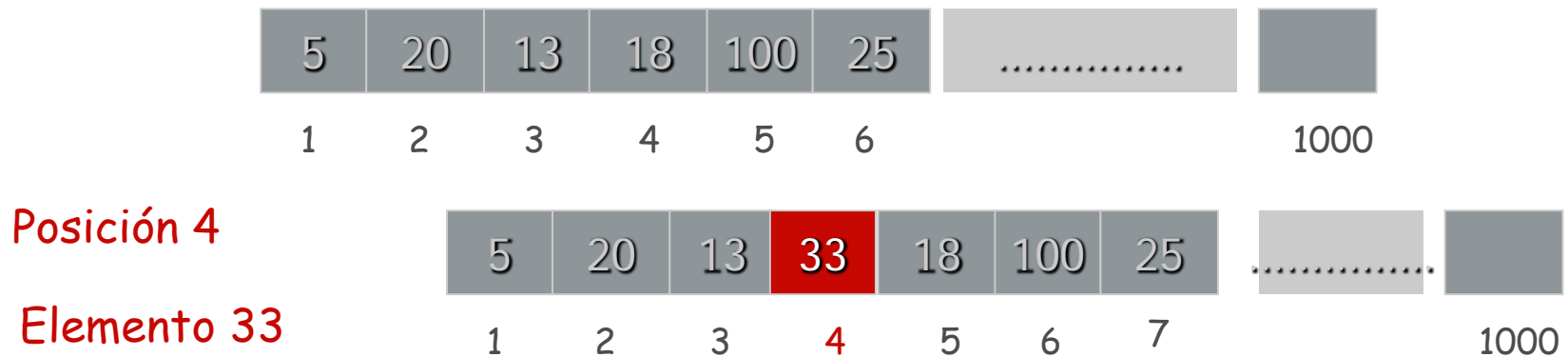
Esta operación también tiene que verificar que haya lugar en la estructura, es decir que la dimensión lógica sea menor que la dimensión física.

CONSIDERACIONES

- ➡ **dimF** : dimensión física del vector (tamaño especificado en la declaración del vector).
- ➡ **dimL**: cantidad de elementos en el vector. De ser posible la operación, la cantidad de elementos será $\text{dimL}+1$.

Vector: Insertar un elemento

1 Insertar un elemento en una posición determinada



2 Insertar un elemento manteniendo un orden interno determinado

Lo veremos mas adelante

Insertar un elemento en una posición determinada

➔ Esta operación también tiene que verificar que la posición sea válida.



Supongamos que se quiere insertar el valor 26 en la posición 2 de un vector de valores enteros...

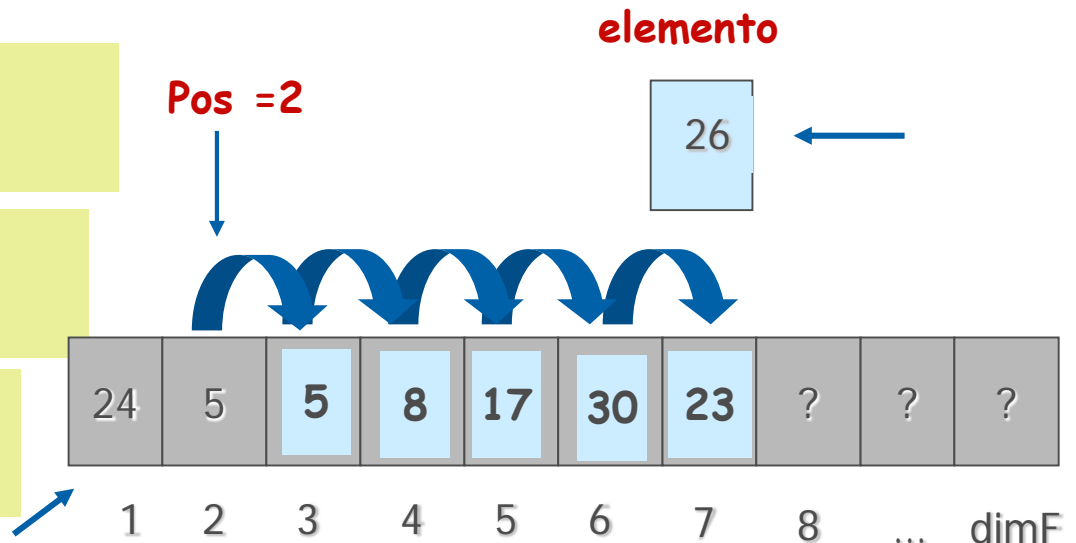
1. Verificar la posición a insertar

2. Verificar espacio en el vector

3. Abrir el vector (a partir de la dimensión lógica)

4. Asignar el valor

5. Aumentar la dimensión lógica



Insertar un elemento en una posición determinada

1. Verificar la posición a insertar
2. Verificar espacio en el vector
3. Abrir el vector (a partir de la dimensión lógica)
4. Asignar el valor
5. Aumentar la dimensión lógica

Parámetros de la operación:

- ✓ v: arreglo a trabajar
- ✓ dimL: cantidad de elementos
- ✓ Elemento: dato a insertar
- ✓ Pos: posición donde insertar
- ✓ éxito: resultado operación

```
Procedure INSERTARPOS(var v: vector;  
var dimL: integer; elemento: integer; pos: integer;  
var éxito: boolean );
```

```
var i : integer;
```

```
Begin
```

```
if (dimF > dimL) and ((pos>=1) and (pos<= dimL))
```

```
then begin
```

```
    éxito := true;
```

```
    for i := dimL downto pos do
```

```
        v [ i + 1 ] := v [ i ];
```

```
        v [pos] := elemento;
```

```
        dimL := dimL + 1;
```

```
    end
```

```
else éxito := false;
```

```
end;
```

Verificar espacio y posición válida

Abrir el arreglo

Asignar el valor

Aumentar la dimensión

Vector: Borrar un elemento

La operación de Borrar un elemento en un vector consiste en eliminar un elemento determinado o bien eliminar un elemento que ocupa una posición determinada.

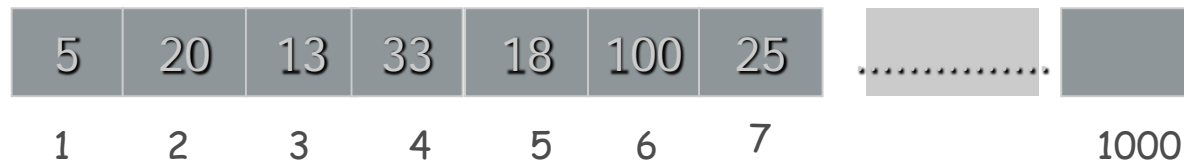
En el caso de borrar un elemento en una posición determinada, se debe verificar que la posición sea válida.

Consideración

➡ **dimL** : cantidad de elementos en el vector.

Vector: Borrar un elemento

1 Borrar un elemento en una posición determinada



Posición 3



¿Dimensión lógica?

2 Borrar un elemento determinado del vector

*Lo veremos
mas adelante*

Vector: Borrar un elemento en una posición

1. Validar la posición
borrar

2. Desplazar elementos
(a partir de la posición
siguiente)

3. Disminuir la dimensión
lógica

```
Procedure BORRARPOS (var v: vector;  
                    var dimL: integer; pos: posicion;  
                    var exito: boolean );  
  
var j: integer;  
begin  
    Validar la posición a borrar  
    if (pos >= 1 and pos <= dimL)  
    then begin  
        Desplazar elementos  
        exito := true  
        for j:= pos+1 to dimL do  
            v [ j-1 ] := v [ j ] ;  
        dimL := dimL - 1 ; Disminuir la dimensión lógica  
    end  
    else exito := false;  
  
End;
```

- ✓ Parámetros de la operación:
- ✓ Num: vector a trabajar
- ✓ dimL: cantidad de elementos
- ✓ Pos: posición a borrar
- ✓ exito: resultado operación

Ejercitación



1. Se lee una sucesión de 100 nombres y notas de alumnos. Informar los nombres y notas de los alumnos que superen el promedio del grupo

- Leer, Guardar y Sumar todas las notas
- Calcular promedio
- Recorrer y Comparar cada nota con el promedio

Leer, Guardar y Sumar todas las notas

Inicializar suma de notas

Repetir 100

 leer datos de alumno

 guardar datos del alumno

 actualizar suma de notas

Recorrer y Comparar cada nota con el promedio

Repetir 100

 acceder a los datos del alumno

 si $\text{nota} > \text{promedio}$ entonces

 informar nombre



1. Se lee una sucesión de 100 nombres y notas de alumnos. Informar los nombres y notas de los alumnos que superan el promedio del grupo.

Analicemos la declaración de tipos y variables

¿Dimensión Física?

¿Dimensión Lógica?

Program ejemplo1;

const total= 100; {dimensión física}

type

str20 := string [20];

alumno = record

nombre : str20;

nota : real;

end;

ListaDatos = array [1..total] of alumno;

{implementación LeerGuardarSumar}

{implementación RecorreryComparar}

var

datos: ListaDatos;

suma, promedio : Real;

Begin

LeerGuardarSumar(datos, suma);

promedio := suma/total;

RecorreryComparar (datos, promedio);

End.

Leer, Guardar y Sumar todas las notas

Inicializar suma de notas

Repetir 100

leer datos de alumno

guardar datos del alumno

actualizar suma de notas

```
const
    total= 100; {dimensión física}
type
    str20 = string [20];
    alumno = record
        nombre : str20;
        nota : real;
    end;
    ListaDatos=array [1..total] of alumno;
```

```
procedure LeerGuardarSumar (var dat:
    ListaDatos; var sum : Real);
```

```
Procedure leerAlumno (var alu:alumno);
begin
    read (alu.nombre);
    read (alu.nota);
end;
```

```
var
    j : integer; a:alumno;
begin
    sum := 0;
    for j := 1 to total do begin
        leerAlumno (a);
        dat[j]:= a;
        sum := sum + dat[j].nota;
    end
end;
```

Recorrer y Comparar cada nota con el promedio

Repetir 100

acceder a los datos del alumno

si nota > promedio entonces

informar nombre

```
const
    total= 100; {dimensión física}
type
    str20 := string [20];
    alumno = record
        nombre : str20;
        nota : real;
    end;
    ListaDatos=array [1..total] of alumno;
```

```
Procedure RecorreryComparar (dat: ListaDatos; prom: real);
var i: integer;
begin
    for i := 1 to total do
        if (dat[i].nota > prom) then
            Writeln (dat[i].nombre, ' ', dat[i].nota )
    End.
```


Ejercitación



2. Se lee una sucesión nom
"Fulano". Informar los nom
promedio del grupo.

Nota: como máximo se pue

Leer, Guardar y Sumar todas las notas

Inicializar suma de notas

Leer datos de alumno

Mientras (haya lugar) y (no ingrese "Fulano")

guardar en el vector

actualizar suma de notas

leer datos de alumno

- Leer, Guardar y Sumar todas las notas

- Calcular promedio

- Recorrer y Comparar

cada nota con el promedio

Recorrer y Comparar cada nota con el promedio

Repetir cantidad alumnos guardada

acceder a los datos del alumno

si nota > promedio entonces

informar nombre



¿Dimensión Física?

¿Dimensión Lógica?

2. Se lee una sucesión de nombres y notas de alumnos que finaliza con nombre "Fulano". Informar los nombres y notas de los alumnos que superan el promedio del grupo.

Nota: como máximo pueden ingresar 100 alumnos.

- Leer, Guardar y Sumar todas las notas
- Calcular promedio
- Recorrer y Comparar cada nota con el promedio

Program ejemplo2;

const total= 100; {dimensión física}

type

str20 := string [20];

alumno = record

nombre : str20;

nota : real;

end;

rango = 0.. total;

ListaDatos = array [1..total] of alumno;

{implementación LeerGuardarSumar}

{implementación Recorrer y Comparar}

var

datos: ListaDatos; dim: rango;

suma, promedio : Real;

Begin

LeerGuardarSumar(datos, dim, suma);

if dim <> 0 then begin

promedio := suma/dim;

Recorrer y Comparar(datos, dim, promedio)

end;

End.

Leer, Guardar y Sumar notas

Inicializar suma de notas

Leer datos de alumno

Mientras (haya lugar) y (no ingrese)

guardar en el vector

actualizar suma de notas

leer datos de alumno

```
const total= 100; {dimensión física}
type
  str20 := string [20];
  alumno = record
      nombre : str20;
      nota : real;
  end;
  rango = 0.. total;
  ListaDatos=array [1..total] of alumno;
```

```
procedure LeerGuardarSumar
(var dat:ListaDatos; var dim: rango;
var sum : Real);
```

```
Procedure leerAlumno (var alu:alumnos);
begin
  read (alu.nombre);
  if (alu.nombre <> 'Fulano')
    then read (alu.nota);
  end;
```

```
var
  a : alumno;
```

```
begin
  sum := 0;
  dim := 0; {dimensión lógica}
  leerAlumno (a);
  while (dim < 100) and (a.nombre <> 'Fulano')
    do begin
      dim := dim + 1;
      dat[dim]:= a;
      sum := sum + dat[dim].nota;
      leerAlumno (a);
    end
  end;
```

Recorrer y Comparar cada nota con el promedio

*Repetir cantidad alumnos guardada
acceder a los datos del alumno
si nota > promedio entonces
informar nombre*

```
const total= 100; {dimensión física}  
type  
  str20 := string [20];  
  alumno = record  
    nombre : str20;  
    nota : real;  
  end;  
rango = 0.. total;  
ListaDatos = array [1..total] of alumno;
```

Procedure RecorreryComparar (dat: ListaDatos; dim:rango; prom: real);

var i: integer;

begin

for i := 1 **to** dim **do**

if (dat[i].nota > prom) **then**

Writeln (dat[i].nombre, ' ', dat[i].nota)

End.

Ejercitación



3. Se lee una secuencia de letras minúsculas que termina en punto. Informar la cantidad de veces que aparece cada letra.



4. Se desea simular la venta de pasajes de micro para Mar del Plata, el día 25 de mayo de 2015 a las 12 Hs. Se sabe que el micro dispone de 45 asientos (ventanilla/pasillo). El pasajero solicita un asiento determinado y se registrará la venta solamente si está libre y en ese caso también se guardará el DNI del pasajero. De lo contrario indicar que la venta no se pudo realizar.



5. Se desea simular la atención de la Oficina de Tránsito de la Municipalidad que diariamente entrega hasta 50 números para obtener la Licencia de Conductor de 7 a 8Hs. Para ello, se requiere guardar el DNI y Nombre y Apellido de la persona a la que se le entrega el número.

Analicemos la declaración de tipos y variables

¿Dimensión Física?

¿Dimensión Lógica?