

Estructura de datos ARREGLO

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

Arreglos: Motivación



Se leen 100 productos de un supermercado. Cada producto está caracterizado por código, nombre, marca, stock y precio. Informar los nombres y precios 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 podemos procesar los datos para obtener 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
Nombre
Marca
Stock
Precio

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.

ATENCIÓN!!! 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
Nombre
Marca
Stock
Precio

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.

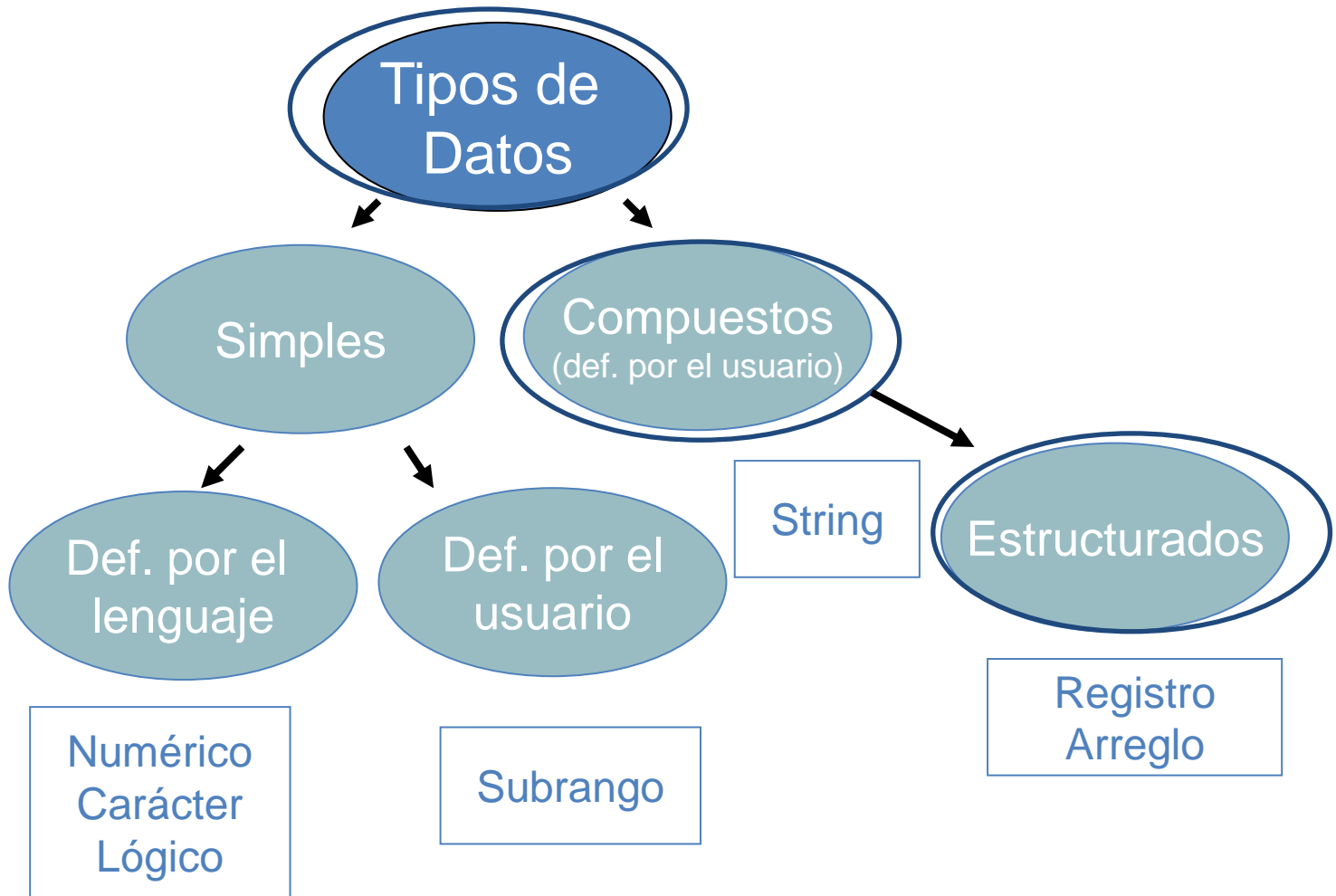
Es decir en cada variable guardamos un producto distinto a medida que se lee (en este caso necesitamos 100 variables diferentes). Luego calculamos el promedio y comparamos cada precio con el mismo.

ATENCION!!! Esta solución resulta más 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.
- Un arreglo es una estructura de datos que permite acceder a cada componente a través de índices, que indican la posición de cada componente dentro de la estructura de datos.

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**.

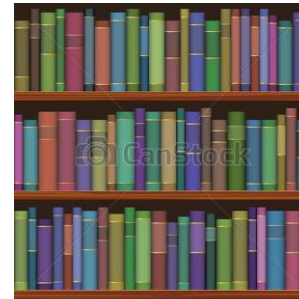
Tipos de arreglos

Existen distintos tipos de arreglos, alguno de ellos son:

- Arreglos unidimensionales: vectores (los usaremos en este curso)
- Arreglos bidimensionales: matrices



Un Indice:
vector



© Can Stock Photo - csp12009223

Dos Indices:
matriz

Horarios de Clase

Hora	Lunes	Martes	Miércoles	Jueves	Viernes

✂️ 📅 ✂️ ✂️ ✂️ ✂️ ✂️ ✂️ ✂️ ✂️



© | N° 495526 | www.photaki.com



Tipo Vector

Volviendo al problema inicial, 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

los podemos almacenar 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
1	2	3	4		...				100

Productos

Tipo Vector: Posición y contenido

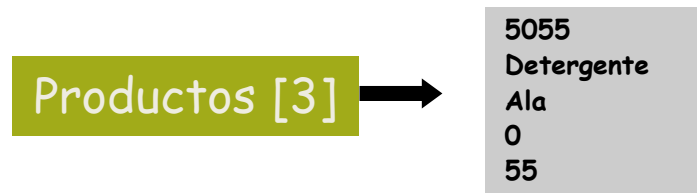
Cuando se trabaja con vectores hay que tener en cuenta que:

Contenido

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

Posición



- ➡ 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.

Tipo Vector: Declaración 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

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

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

Tipo Vector: Declaración 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.

Tipo Vector: Declaración en Pascal

```
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];
```

```
  cadena7 = string [7];
```

```
  auto = Record
```

```
    patente: cadena7;
```

```
    marca: cadena15;
```

```
    modelo: cadena15;
```

```
    precio: real;
```

```
  End;
```

```
  vectorAutos= array [1..50] of auto;
```

```
Var
```

```
  N: numeros;      {1000 elementos enteros}
```

```
  Autos: añosAutos; {16 elementos enteros}
```

```
  concesionaria: vectorAutos; {50 elementos tipo auto}
```

```
  TotalporCaja: cajas; {4 elementos reales}
```

Analicemos para cada variable:

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

Tipo vector: operaciones

- 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;
```

```
    cajas = array [ 'A' .. 'D' ] of real;
```

Var

```
    productos : vectorProductos; N: números;
```

```
    i : integer; totalPorCaja: cajas;
```

Begin

```
    totalPorCaja[ 'C' ] := 2500,50;
```

```
    N [5] := 500;
```

```
    N [1] := N [5] * 2;
```

```
    productos[1].precio:= 100;
```

```
    productos[3].precio:= productos[1].precio + 10;
```

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

```
    i:= 10;
```

```
    productos[i].stock:= 50;
```

```
    . . .
```

End.

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?*

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 del vector se trabaja individualmente.

Var

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

Begin

```
readln (N[1]);  
  
writeln (N[1]);  
  
For i:= 1 to 100 do  
    LeerProducto (Productos[i]);  
  
For i:= 1 to 100 do  
    MostrarProducto (Productos[i]);  
    . . .  
End.
```

Tipo 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?

Tipo Vector : Ejemplo de Recorrido Total



Por ejemplo, si se necesita conocer la cantidad total del stock 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.
```

Tipo 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 precondition 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 writeln ('No existe');
```

```
End:
```

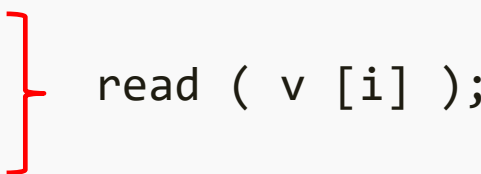
Analicemos la condición del While...

Tipo Vector : Operación de Carga Completa

La operación de **Carga Completa** en un vector consiste en guardar un elemento en cada posición del vector.

Por ejemplo, implementar un programa que cargue un vector con 50 números enteros que se leen.

```
Program cargarVector;  
Const  
    fin = 50;  
Type  
    vectorNúmeros = array [1..fin] of integer;  
Var  
    i, num: integer;  
    v: vectorNúmeros;  
begin  
    For i:= 1 to fin do  
        begin  
            read (num);  
            v [i] := num;  
        end;  
    end.  
end.
```



Tipo Vector: Ejercitación



1. Se leen 100 productos de un supermercado. Cada producto está caracterizado por código, nombre, marca, stock y precio. Informar los nombres y precios de los productos cuyo precio supera el promedio de precios del supermercado.

- Leer, Guardar y Sumar todos los precios
- Calcular promedio
- Recorrer y Comparar cada precio con el promedio

Leer, Guardar y Sumar todos los precios

Inicializar suma de precios

Repetir 100

leer datos del producto

guardar datos del producto

actualizar suma de precios

Recorrer y Comparar cada nota con el promedio

Repetir 100

acceder a los datos del producto

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

informar nombre y precio



Se leen 100 productos de un supermercado. Cada producto está caracterizado por código, nombre, marca, stock y precio. Informar los nombres y precios de los productos cuyo precio supera el promedio de precios del supermercado.

Program ejercitación;

const total= 100;

type

cadena15 = string [15];

producto= **Record**

codigo: integer;

nombre: cadena15;

marca: cadena15;

stock: integer;

precio: real;

End;

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

{implementación LeerGuardarSumar}

{implementación RecorreryComparar}

var

super: vectorProductos;

suma, promedio : Real;

Begin

LeerGuardarSumar (super, suma);

promedio := suma/total;

RecorreryComparar (super, promedio);

End.

Leer, Guardar y Sumar todos los precios

Inicializar suma de precios

Repetir 100

leer datos de producto

guardar datos del producto

actualizar suma de precios

```
const
    total= 100;
type
    cadena15 = string [15];
    producto= Record
        codigo: integer;
        nombre: cadena15;
        marca: cadena15;
        stock: integer;
        precio: real;
    End;
    vectorProductos=array [1..100]
        of producto;
```

```
Procedure LeerGuardarSumar(var v:vectorProductos;
                            var sum: Real);
Procedure leerProducto (var p:producto);
begin
    read (p.codigo);
    read (p.nombre);
    read (p.marca);
    read (p.stock);
    read (p.precio);
end;

var
    j : integer; prod:producto;
begin
    sum := 0;
    for j := 1 to total do begin
        leerProducto (prod);
        v [j]:= prod;
        sum := sum + v [j].precio;
    end
end;
```

*Recorrer y Comparar cada precio
con el promedio*

Repetir 100

acceder a los datos del producto

si precio > promedio entonces

informar nombre y precio

```
const
    total= 100;
type
    cadena15 = string [15];
    producto= Record
        codigo: integer;
        nombre: cadena15;
        marca: cadena15;
        stock: integer;
        precio: real;
    End;
vectorProductos=array [1..100]of producto;
```

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

ESTRUCTURA DE DATOS VECTOR – Para resolver en clase



2. Implementar un programa que lea 200 artículos de una juguetería. De cada artículo se lee: código, descripción, fecha de fabricación, edad recomendada y precio. Informar la cantidad de artículos que superan el promedio de precios.

3. Implementar un programa que lea una secuencia de caracteres que termina en punto. Se debe informar la cantidad de veces que aparece cada letra minúscula en la secuencia.

ESTRUCTURA DE DATOS VECTOR – Para resolver en clase



•Un centro de deportes quiere almacenar la información de sus clientes y de los 4 tipos de actividades que ofrece: *1) Musculación, 2) Spinning, 3) Cross Fit, 4) Libre*. Para ello, se debe leer el precio mensual de cada actividad y almacenarlo en un vector. Además, se debe leer y almacenar la información de los 1000 clientes del centro de deportes. De cada uno se conoce: código de cliente, DNI, apellido, nombre, edad y el número de actividad elegida (1..4). **Al finalizar la lectura y el almacenamiento de los clientes, se pide:**

- a) Calcular e informar el monto total a recaudar por cuotas.
- b) Informar el DNI de los clientes que superen el promedio de edad entre todos los clientes.
- c) Informar el código de actividad más elegido.

Se sabe que cada cliente elige una sola actividad. Modularizar su solución.