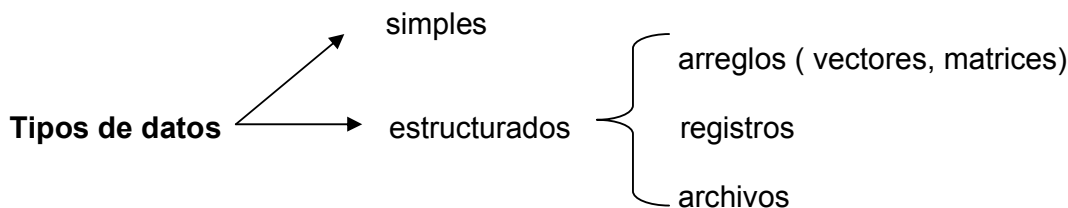


## Capítulo 5

1. Arreglos lineales, unidimensionales o Vectores. Declaración del tipo.
2. Lectura, escritura operaciones sobre arreglos. Ejemplos
3. Arreglos paralelos
4. Índice con significado. Acceso a una determinada componente del arreglo.
5. Búsqueda sobre arreglos. Secuencial. Binaria.
6. Eliminación e inserción de un elemento en un arreglo.
7. Intercalación de arreglos ordenados.
8. Métodos de Ordenación



Las variables de tipo simple almacenan un solo valor (entero, carácter, boolean, etc).  
 Los tipos de datos estructurados permiten declarar variables para almacenar un conjunto de valores y poder acceder en forma individual o total (en algunas operaciones).  
 Estos tipos deben ser declarados en la sección Type del programa.

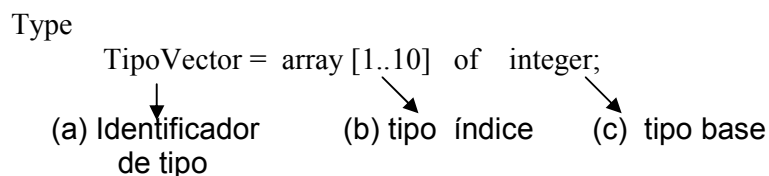
### 1. Arreglos lineales o Vectores

Agrupar un conjunto de valores del mismo tipo (simple o estructurado), en una única variable.  
 También se denominan arreglos unidimensionales, pues sus elementos se acceden en forma individual a través de su posición determinada por un índice (una dimensión).

#### 1.1.Declaración del tipo arreglo

Cuando se describe el tipo se establece:

- a. El identificador o nombre del tipo
- b. La cantidad máxima de elementos y el rango sus posiciones (para el acceso individual).
- c. El tipo de los valores que almacena



El tipo índice debe ser ordinal (entero, char), el tipo base no tiene restricciones (simple o estructurado)

La declaración del TipoVector asocia este identificador a una estructura de 10 elementos de tipo entero (base).

La cantidad de elementos se establece mediante un valor constante (10).

Cada uno de ellos se puede acceder mediante su ubicación : 1..10 (índice)

## 1.2. Declaración de variables del tipo arreglo. Operaciones

A partir de este tipo se declaran una o más variables del tipo TipoVector

```
Var
Vec1, Vec2 : TipoVector;
j, k : byte;
```

|      |    |   |   |   |   |   |   |   |   |
|------|----|---|---|---|---|---|---|---|---|
| Vec1 |    |   |   |   |   |   |   |   |   |
|      | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Vec2 |    |   |   |   |   |   |   |   |   |
|      | 10 |   |   |   |   |   |   |   |   |

Este espacio de memoria queda reservado, independientemente de que el programa requiera parte o la totalidad de las componentes de las variables Vec1 y Vec2.

Para acceder a una determinada componente de un vector, se indica la posición entre corchetes.

```
Vec1[2] := 5;
Vec1[1] := Vec1[2] - 3;
j := 3;
Vec1[j] := Vec1[j-1] * 2;
Vec2[1] := Vec1[3];
Vec1[j + 1] := Vec1[1] + Vec2[j - 2];
For j := 2 to 10 do
  Vec2[j] := 0;
For k := 1 to 4 do
  Write(Vec1[k] : 5);
```

|      |    |   |    |    |   |   |   |   |   |
|------|----|---|----|----|---|---|---|---|---|
| Vec1 | 2  | 5 | 10 | 12 |   |   |   |   |   |
|      | 1  | 2 | 3  | 4  | 5 | 6 | 7 | 8 | 9 |
| Vec2 | 10 | 0 | 0  | 0  | 0 | 0 | 0 | 0 | 0 |
|      | 10 |   |    |    |   |   |   |   |   |

El índice puede ser constante, variable o expresión. Debe tomar valores dentro del rango con el que se describió el tipo,

Como se observa en las líneas de código que operan sobre los vectores Vec1 y Vec2 es posible acceder a un elemento en particular o (utilizando un ciclo) a un conjunto de elementos.

La compatibilidad que debe tenerse cuenta, tanto en las operaciones como en las asignaciones, está condicionada al tipo Base.

Tanto la lectura desde teclado (o archivo de texto) como la escritura de un arreglo, debe hacerse componente a componente.

Es posible asignar una variable de tipo arreglo a otra del mismo tipo ( $\text{Vec1} := \text{Vec2}$ ), pero no admite otro tipo de operaciones como  $\text{Vec1} + \text{Vec2}$ . o  $\text{Vec1} * 5$ .

### 1.3. Arreglos constantes

Es posible declarar una arreglo constante, describiendo en la sección Const, el nombre, el tipo asociado y todos los valores que requiere el rango de índice.

Type

TV= array[1..12] of byte;

Const

DiasMes : TV = ( 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);

Los arreglos constantes son útiles como tablas de consulta, accediendo directamente al elemento requerido (DiasMes[3]) o recorriéndolo secuencialmente.

### 1.4. Otros ejemplos de tipo arreglo

Const

MaxElem = 5;

Type

St10=string[10];

SubRango= -3..2;

TVecSt=array[SubRango] of St10;

TVecCh = array [1..MaxElem] of char;

TVecReal = array ['A'..'Z'] of real;

**Utilidad.** Los arreglos permiten mantener en memoria un conjunto de datos y acceder a ellos (en cualquier lugar del programa) sin necesidad de volver a leerlos o generarlos.

Por ejemplo si se quiere determinar de un conjunto de personas cuantas tienen edad por encima del promedio. Se debe ingresar las edades almacenándolas en un arreglo, sumarlas, dividir la suma por la cantidad de personas y luego comparar cada edad (almacenada en el arreglo) con el promedio.

### 1.5. Arreglos cuyas componentes son de tipo función o tipo procedimiento

Program VectorDeFunciones;

Type

Tfunc = function( m: integer): real ;

TVec = array[1..3] of Tfunc;

**function F1**( m: integer) : real;

begin

F1 := m\*(m - 0.5);

End;

**function F2**( m: integer) : real;

begin

F2 := m / (m - 0.5)

End;

**function F3**( m: integer) : real;

begin

F3 := (m - 1) / m

```

End;

Const
    Vec : Tvec = (@F1, @F2, @F3);
Var
    i, m: integer;
begin
    readln (m);
    For i := 1 to 3 do
        Writeln('F', i, '(', m, ')=' , Vec[ i ]( m ):0:2);
    End.

```

### Consideraciones

Las componentes del vector Vec son tipo función (direcciones de memoria del código de una función)

## **2.Lectura, escritura operaciones sobre arreglos. Ejemplos**

Ej1. A continuación se desarrolla un programa que utilizando funciones y procedimientos para cada ítem:

- a. Lee un arreglo de enteros
- b. Calcula la suma de sus elementos
- c. Cuenta la cantidad de componentes pares
- d. Imprime las componentes que se encuentran en ubicaciones pares
- e. Imprime el mínimo
- f. Genera un segundo arreglo con los múltiplos de K (dato entero) y lo escribe.
- g. Permite elegir por medio de un menú el/los proceso/s descriptos

```

Program Ej1;
TYPE
    TV=array[1..100] of integer;

VAR
    N, M, K : integer;
    A, B :TV;    Op : char;

Procedure LeeVector ( Var A : TV; Var N : integer)
Var
    i : integer;
Begin
    Write('Ingrese la cantidad de elementos del vector<=100');    Readln(N);
    For i:=1 to N do
        Begin
            Write('Ingrese el elemento ',i); Readln(A[i])
        End
    End;
End;

```

```
Function Suma (A : TV; N: integer) : integer;
Var
    i, Sum : integer;
Begin
    Sum:=0;
    for i:=1 to N do
        Sum:=Sum+ A[i];
    Suma := Sum;
End;
```

```
Function CuentaPares (A : TV; N: integer) : integer;
Var
    i, Cont : integer;
Begin
    Cont:=0;
    for i:=1 to N do
        If Not Odd (A[i] ) then
            Cont:=Cont+ 1;
    CuentaPares := Cont;
End;
```

```
Procedure EscPosPares ( A : TV; N : integer);
Var i : integer;
Begin
    i:= 2;
    While i<= N do
        Begin
            Write(A[i] : 5); i:=i +2;
        End
    End;
End;
```

```
Function Minimo (A : TV; N: integer) : integer;
Var
    i, Min : integer;
Begin
    Min:=A[1];
    for i:=2 to N do
        If A[i] < Min then
            Min:=A[i];
    Minimo := Min;
End;
```

```
Procedure GeneraOtro (A : TV; N, K : integer; Var B : TV; Var M : integer);
Var i : integer;
Begin
    M:= 0;
    For i:=1 to N do
        If A[i] Mod K = 0 Then
            Begin
                M:=M +1; B[M] := A[i];
            End
    End;
End;
```

```
Procedure EscVector ( V : TV; L : integer)
```

```
Var i : integer;
```

```
Begin
```

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

```
    Write(V[i] : 5)
```

```
End;
```

```
Procedure Menu ( Var Op : Char);
```

```
Begin
```

```
  Writeln(Menú de opciones');
```

```
  Writeln('b - Suma los elementos del arreglo');
```

```
  Writeln('c - Cuenta los elementos pares);
```

```
  Writeln('d - Imprime los elementos de las posiciones pares');
```

```
  Writeln('e - Calcula el mínimo');
```

```
  Writeln('f - Genera un arreglo con los elementos múltiplos de K');
```

```
  Writeln('g - Salir');
```

```
  Repeat
```

```
    Write(' Ingrese su opción'); Readln(Op);
```

```
  Until ( 'b' <= Op) and ( Op <= 'g');
```

```
End;
```

```
Begin {programa principal}
```

```
  LeeVec(A, N);
```

```
  Repeat
```

```
    Menu(Op);
```

```
  Case Op of
```

```
    'b': writeln('La suma de los elementos del arreglo es:', Suma(A, N));
```

```
    'c': writeln('La cantidad de elementos pares del arreglo es:', CuentaPares(A, N));
```

```
    'd': begin
```

```
      writeln('Elementos de las posiciones pares del arreglo'); EscPosPares ( A , N )
```

```
    end;
```

```
    'e': writeln('El mínimo del arreglo es:', Minimo(A, N));
```

```
    'f': begin
```

```
      Write ('Ingrese un valor de K para seleccionar los múltiplos de K'); Readln (K);
```

```
      GeneraOtro (A , N, K, B, M );
```

```
      writeln('Elementos de múltiplos de ', K ); EscVector ( B, M )
```

```
    end;
```

```
  End ; {case}
```

```
  Until Op = 'g';
```

```
End.
```

### 3. Arreglos paralelos

Dos o más arreglos son paralelos, si tienen la misma cantidad de componentes y están relacionados de forma tal que sus componentes en la i-esima posición forman una unidad de información.

A partir de las siguientes declaraciones

```
Type
```

```
  St25 = string[25];
```

```
  TVNom = array[1..50] of St25;
```

```
  TVEdad = array[1..50] of byte;
```

Var

VNom : TVNom;  
VEdad : TVEdad;

Para almacenar nombre y edad de un conjunto de personas, se utilizan los vectores VNom y VEdad en forma "paralela".

VNom[1] y VEdad[1] nombre y edad de la primera persona

VNom[2] y VEdad[2] nombre y edad de la segunda persona

.....

VNom[i] y VEdad[i] nombre y edad de la *i-esima* persona

Ej2. El siguiente programa lee el nombre y la edad de N personas , calcula e imprime:

- Listado de nombres de las personas de mas de X años.
- Porcentaje de las personas mayores de edad sobre el total.
- Obtener un arreglo que contenga los nombres de las personas con edad entre E1 y E2.

Program Paralelos;

Type

St25 = string[25];  
TVNom = array[1..50] of St25;  
TVEdad = array[1..50] of byte;

Var

VNom, Per : TVNom; VEdad : TVEdad;  
K, X, E1, E2, N : byte;

Procedure LeeParalelo( Var VNom:TVNom; Var VEdad: TVEdad; Var N:Byte);

Var

I : Byte

Begin

Readln ( N );

For I := 1 to N do

Begin

Readln ( VNom[ i ] ); Readln ( VEdad [ i ] );

End;

End;

Procedure Listado( VNom: TVNom; VEdad: TVEdad; N,X: Byte);

Var

I: Byte

Begin

For i:= 1 to N do

If VEdad[ i ] > X Then

Writeln ( VNom[ i ] );

End;

```
Function Porcen( VEdad : TVEdad; N: Byte): Real;
```

```
  Var
```

```
    I, Cont : Byte;
```

```
  Begin
```

```
    Cont:=0;
```

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

```
      If VEdad[ i ] >=21 Then
```

```
        Cont : Cont + 1;
```

```
    Porcen = Cont * 100 / N;
```

```
  End;
```

```
Procedure Personas (VNom: TVNom; VEdad:TVEdad; E1,E2, N:Byte; Var Per:TVNom; Var k: Byte);
```

```
  Var
```

```
    I : Byte;
```

```
  Begin
```

```
    K:= 0;
```

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

```
      If (VEdad[ i ] >= E1) and (VEdad[ i ] <= E2) Then
```

```
        Begin
```

```
          K:= K +1;
```

```
          Per[ K ] := VNom[ i ];
```

```
        End;
```

```
  End;
```

```
Procedure EscribeVec( V: TVNom; M: Byte);
```

```
  Var
```

```
    I: Byte
```

```
  Begin
```

```
    For i:= 1 toM do
```

```
      Writeln ( V[ i ] );
```

```
  End;
```

```
Begin
```

```
LeeParalelo( Vnom, VEdad, N);
```

```
Writeln("Ingrese edad mínima para listado de nombres"); readln (X);
```

```
Listado( Vnom, VEdad, N, X);
```

```
Writeln ( 'El porcentaje de personas mayores de edad es ', Porcen( VEdad, N));
```

```
Writeln("Ingrese intervalo para nuevo arreglo"); readln (E1, E2);
```

```
Personas (Vnom, VEdad; E1,E2, N, Per, K);
```

```
EscribeVec (Per, k);
```

```
End.
```



Ej3. Se tiene un archivo que contiene una cadena de caracteres en cada línea formada por letras y dígitos (ambos) finalizada con un punto.

Se pide leerlo y generar dos vectores paralelos, uno contendrá las cadenas formada por las letras y el otro las cadenas formadas por los números.

Ejemplo:

| Archivo          |
|------------------|
| 1JEF3UANTE1ZYH.  |
| MA5RIADAGTET.    |
| JUATTT1NRT13ZNG. |

| VecLetras    |
|--------------|
| JEUANTEZYH   |
| MARIADAGTET  |
| JUATTTNRTZNG |

| VecNros |
|---------|
| 131     |
| 5       |
| 113     |

Program Separa;

Type

ST10= string[10];

TV= array[1..10] of St10;

Procedure SeparaCadenas( var VLetras, VNros: TV; var N:byte);

Var

Arch: Text;

Car: char;

Begin

ASSIGN (Arch, 'Cadenas.txt'); RESET(Arch);

N:=0;

while not eof(Arch) do

begin

N:=N+1;

VLetras[N]:= '';

VNros[N]:= '';

Read(Arch, Car);

While Car <> '.' do

begin

If Car in ['0'..'9'] then

VNros[N]:= VNros[N]+ Car

else

VLetras[N]:=VLetras[N] + Car;

Read(Arch, Car) ;

End;

Readln(Arch);

End;

CLOSE(Arch);

End;

Procedure Muestra( VLetras, VNros: TV; N:byte);

Var

i:byte;

Begin

For i := 1 to N do

Writeln (VLetras[i]:15, VNros[i]:10);

End;

```

Var
  N:byte;
  VLetras, VNros: TV
Begin
  SeparaCadenas( VLetras, VNros);
  Muestra( VLetras, VNros, N);
End.

```

#### 4. Índice con significado. Acceso a una determinada componente del arreglo.

En los ejemplos anteriores se han tratado los arreglos con acceso secuencial, es decir recorriendo en forma progresiva sus elementos (sucesión del índice: 1, 2, 3, 4 ...)

La estructura del vector (y de arreglos en general) permite el acceso directo a una componente determinada, indicando una posición que en general está asociada a alguna característica o atributo de la información tratada, es decir tiene un **significado** dentro del ámbito del problema planteado. El concepto se ilustra en el siguiente ejemplo:

Ej3. En un edificio de N pisos, se desea evaluar el uso del ascensor en relación a la cantidad de personas y al piso al cual se dirigen. Para ello durante una día se ha grabado en cada línea del archivo cada parada del ascensor: Piso, Cantidad (de personas que bajan en dicho piso).

Se pide leer los datos del archivo para informar:

- Para cada piso el promedio de personas que bajaron en dicho piso.
- Cuál fue el piso con más paradas (no importa la cantidad de personas).

Es necesario utilizar un contador de paradas y un acumulador de personas para cada piso.

Se implementan dos arreglos enteros, un contador de paradas y un acumulador de personas.

El piso “trabaja” como índice seleccionando la componente que será incrementada y acumulada.

```

Program IndiceSignificativo;
Type
  TV=array[1..50] of word;
Procedure IniciaV(Var V :TV);
Var
  i: Byte;
Begin
  For i := 1 to 50 do
    V[i] :=0 ;
  End ;
Procedure Contabiliza(Var Personas, Paradas:TV);
Var
  Piso, Cant: Byte;
  Arch:text;
Begin
  IniciaV(Personas); IniciaV(Paradas);
  Assig(Arch, 'Movimientos.txt'); Reset(Arch);
  While Not Eof(Arch) do
    Begin
      Readln(Arch, Piso, Cant);
      Paradas[Piso]:= Paradas[Piso] + 1;
      Personas[Piso]:= Personas[Piso] + Cant;
    End;
  Close(Arch);
End;

```

El Piso es índice de los arreglos Paradas y Personas

```

Procedure Promedio( Personas, Paradas:TV ; N :byte);
Var
  i: Byte;
Begin
  Writeln('Piso   Promedio de personas por viaje' );
  For i := 1 to N do
    If Paradas[i] <> 0 then
      Writeln(i :3, Personas[i] div Paradas[i] :12)
    Else
      Writeln(i :3, 0 :12)
  End ;

Function MaxPiso(V:TV; N:byte):byte;
Var
  i, Pos:byte;
begin
  Pos:=1;
  for i:=2 to N do
    if V[i]>V[Pos] then
      Pos:=i;
  MaxPiso:= Pos;
End;

Var
  Personas, Paradas:TV ;
  N : byte ;
{programa principal}
Begin
  Writeln('Ingrese cantidad de pisos' ) ; Readln(N) ;
  Contabiliza(Personas, Paradas);
  Promedio(Personas, Paradas, N);
  writeln('El piso con mayor cantidad de viajes es', MaxPiso(Paradas,N))
end.

```

**5. Búsqueda**

Siendo V un arreglo de N elementos enteros, se requiere saber si un valor X se encuentra en el arreglo y cuál es su posición.

**5.1.** El proceso que conlleva dicho requerimiento depende de:

- El elemento puede no estar.
- Se sabe que el elemento está, solo se requiere su posición.
- El arreglo está ordenado.

A pesar que se recorre el arreglo y se conoce la cantidad de elementos del vector, se utiliza un ciclo While y no un For. Porque el avance sobre el arreglo depende de dos condiciones:

- ✓ Todavía quedan componentes por inspeccionar ( $i \leq N$ )
- ✓ No se encontró el valor buscado ( $V[i] \neq X$ )

El índice se inicia antes del ciclo y se incrementa mientras no se encuentre el valor buscado.

a.1.El ciclo se detiene cuando encuentra X o en la última componente, devolviendo un valor boolean.

```
Function Esta ( V:TV; N:byte; X:integer ):boolean;
Var
  i: byte;
Begin
  i:=1;
  While (i< N) and (X <> V[i]) do
    i:= i+1;
  Esta := V[i] = X;
End;
```

a.2. El ciclo se detiene cuando encuentra X o cuando ya examinó todos los elementos y no lo encontró (devuelve índice 0 )

```
Function EstaPos ( V:TV; N:byte; x:integer ):byte;
Var
  I: byte;
Begin
  i:=1;
  While (i<= N) and (x <> V[i]) do
    i:= i+1;
  If i<=N then
    EstaPos := i
  Else
    EstaPos:= 0;
End;
```

Evalúe el resultado de las funciones con los siguientes datos:

V = (2, 4, 5, -2, 3, 8, 15, 9) ; N = 8;  
X = -2            y    X = 1

b.El ciclo se detiene cuando encuentra X, no debe controlar el índice.

```
Function Busca ( V:TV; N:byte; x:integer ):byte;
Var
  i: byte;
Begin
  i:=1;
  While x <> V[ i ] do
    i:= i+1;
  Busca := i
End;
```

c. Si el arreglo esta ordenado no debe recorrerlo hasta el final para determinar que x no está en el arreglo, por lo tanto se avanza mientras  $X > V[i]$ .

```
Function BuscaOrd ( V:TV; N:byte; x:integer ):byte;
Var
  i: byte;
Begin
  i:=1;
  While (i< N) and (x > V[ i ]) do
    i:= i+1;
  If V[ i ] = x then
    BuscaOrd := i
  Else
    BuscaOrd:= 0;
End;
```

Evalúe el resultado de las funciones con los siguientes datos:

$V = (-2, 2, 3, 4, 5, 8, 9, 15)$ ;  $N = 8$ ;  
 $X = 3$                       y                       $X = 7$

## 5.2. Búsqueda binaria

Si el arreglo está ordenado no es eficiente recorrerlo secuencialmente, en el peor caso debe hacer N comparaciones.

La estrategia es “partir” a la mitad el arreglo y analizar si el elemento del medio es el valor buscado, de no serlo puede ser menor y se descarta la mitad derecha o mayor descartándose la mitad izquierda.

Este proceso implica redefinir el limite derecho o izquierdo de un subarreglo donde se aplicara el mismo criterio. El proceso finaliza cuando lo encuentra o cuando ya no es posible redefinir un subarreglo.

```
Function BusquedaBinaria ( V:TV; N:byte; x:integer ):boolean;
Var
  Medio, Pri, Ult: byte;
Begin
  Pri := 1;
  Ult := N;
  Medio:=(Pri + Ult) DIV 2;
  While (Pri <Ult) and (x <> V[ Medio ]) do
    Begin
      If x < V[ Medio ] then
        Ult := Medio - 1
      Else
        Pri:= Medio + 1;
      Medio:=(Pri + Ult) DIV 2;
    End;
  BusquedaBinaria:= x = V[ Medio]
End;
```

Evalúe el resultado de las funciones con los siguientes datos:

$V = (-2, 2, 3, 4, 5, 8, 9, 15)$ ;  $N = 8$ ;  
 $X = 2$                       y                       $X = 7$

Que se debería cambiar para que la función devuelva la posición donde encuentra X o cero en caso de no encontrarlo.

## 6. Eliminación e inserción de un elemento en un arreglo.

**6.1.** Eliminar en un arreglo V de N elementos el que está en la posición Pos. El proceso “compacta” el arreglo corriendo todos los elementos hacia la izquierda (desde Pos + 1 hasta N). Se decrementa la cantidad de elementos (N-1)

Procedure Elimina ( Var V: Tv; Var N : Byte; Pos:Byte);

Var

i: Byte;

Begin

For i:= Pos to N-1 do

V[ i ] := V[ i+1 ];

N:=N-1;

End;

Evalúe el resultado obtenido con los siguientes datos:

N = 5      V= ( 2, 4, 7, 1, 3 )

a. Pos = 5

b. Pos = 3

c. Pos = 1

V = ?

N = ?

**6.2** .Insertar en un arreglo V de N elementos el valor X en la posición Pos. El proceso “abre” el arreglo para hacer lugar al nuevo valor. Se corren todos los elementos hacia la derecha (desde N hasta Pos).

Se incrementa la cantidad de elementos (N+1).

Procedure Inserta( Var V:TV; Var N:Byte; Pos: Byte; X:Integer);

Var

i: Byte;

Begin

For i := N downto Pos do

V[ i+1 ] := V[ i ];

V[ Pos ]:= X;

N := N+1;

End;

Evalúe el resultado obtenido con los siguientes datos:

N = 5      V= ( 2, 4, 7, 1, 3 )

a. Pos = 6

b. Pos = 3

c. Pos = 1

V = ?

N = ?

**6.3.** Si la inserción es en un arreglo ordenado, se debe determinar la posición. El proceso busca de derecha a izquierda y realiza el corrimiento simultáneamente

Se incrementa la cantidad de elementos (N+1).

Procedure InsertaOrdenado ( Var V: TV; Var N: Byte; X: Real);

Var

J: Byte;

Begin

J:= N;

While ( J>0 ) and ( X < V[ J ] ) do

Begin

V[ J+1 ] := V[ j ] ; J := J-1;

End;

V[ J+1 ] := X ;

N := N+1;

End;

Evalúe el resultado obtenido con los siguientes datos:

N = 5      V= ( 2, 3, 5, 7, 9 )

a. X = 10

b. X = 6

c. X = 1

V = ?

N = ?

## 7. Intercalación de arreglos ordenados.

Como resultado de la intercalación (fusión o mezcla) de dos arreglos ordenados V1 y V2, de N y M elementos respectivamente, se obtiene un tercero ordenado V3. La cantidad de elementos de éste último es N +M, o N +M – Repeticiones, según el criterio elegido.

Se deben comparar los elementos de ambos arreglos de a pares, comenzando con las primeras componentes de cada uno, copiando en el tercer arreglo la menor y avanzando sobre el arreglo del cual proviene dicha componente. En la siguiente comparación se enfrentan nuevamente dos componentes y se repite el proceso hasta que uno o ambos arreglos hayan sido procesados en su totalidad. En caso que uno de ellos tenga elementos sin copiar al tercer arreglo (por ser los mayores), se copian en un ciclo final.

Procedure Intercalacion (V1, V2 : TV ; N, M : byte; Var V3 : TV; Var K : byte);

Var

t, i, j : byte ;

Begin

i:= 1; j:=1; K := 0;

While (i<= N) and (j<=M)do

Begin

K:=K +1;

If V1[i] < V2[j] then

Begin

V3[K]:= V1[i]; i:= i + 1 ;

end

else

If V1[i] > V2[j] then

Begin

V3[K]:= V2[j]; j:= j +=1 ;

end

else

Begin

V3[K]:= V1[i]; i:= i + 1 ; j:= j +=1 ;

end

end;

for t := i to N do {completa la copia de los elementos de V1 en V3}

begin

K:=K +1;

V3[K]:= V1[t]

End;

for t:=j to M do {completa la copia de los elementos de V2 en V3}

begin

K:=K +1;

V3[K]:= V2 [t]

End;

End;

La intercalación de dos conjuntos de valores ordenados es un proceso usual en programación que se aplica sobre otras estructuras de datos como archivos y listas.

## 8. Métodos de Ordenación

Algunos procesos, como la búsqueda, o inserción de un elemento en un vector dependen de que los elementos se encuentren, o no, ordenados. Siempre es posible ordenarlo, existen numerosos métodos de ordenación que comparan y mueven los datos, su eficiencia (rapidez) es proporcional a la cantidad y al orden parcial, que presenten sus elementos.

Algunos métodos no aprovechan el orden inicial y repiten procesos de comparación e intercambio una cantidad fija de veces, otros detectan situaciones de orden parcial o total y evitan comparaciones y repeticiones inútiles

Si quiero ordenar en forma ascendente, V1 presenta sus elementos mejor dispuestos que V2

V1 = ( 2, 3, 12, 7, 8, 21, 9, 15 )      V2 = ( 21, 18, 10, 15, 6, 2, 3, 1)

Cuando se menciona “ordenado”, por defecto se entiende en forma ascendente.

A continuación se presentan distintos métodos de ordenación, implementados a través de procedimientos considerando que TV es un tipo vector de enteros.

### 8.1. Método de Selección

Es uno de los más simples, pero también más ineficiente, ya que la cantidad de comparaciones que realiza es fija en el orden de  $N^2$  (no aprovecha el orden inicial, N es la cantidad de elementos ordenar)

Se basa en las siguientes acciones:

- Selecciona el elemento menor (o mayor si es orden descendente)
- Lo coloca en la posición mas baja (o más alta) del arreglo (no ordenado aún)

Procedure Seleccion (Var V:TV; N:byte);

Var

Min, i, k: byte;

Aux: integer;

Begin

For k:=1 to N-1 do {N-1 pasadas}

Begin

Min:=k;

For i:=k+1 to N do {elementos de la pasada k-esima, se selecciona el indice del mínimo Min}

If V[Min]>V[i] then

Min:= i;

If Min <> k then

begin

Aux:=V[k]; V[k]:=V[Min]; V[Min]:=Aux;

End

End;

El siguiente link explica el método [https://www.youtube.com/watch?v=f8hXR\\_Hvybo](https://www.youtube.com/watch?v=f8hXR_Hvybo)

### 8.2. Método de Burbujeo o Intercambio de a pares

Se basa en recorrer el arreglo comparando dos elementos consecutivos V [i] y V[i + 1].

Si están desordenados (V[i] > V[i+1]) los intercambia. Este proceso provoca que al finalizar una “pasada” o “barrida” sobre el arreglo, el elemento mayor se desplace a la derecha, ocupando el sitio que le corresponde. Los elementos de la derecha van quedando ordenados en forma creciente (como mínimo a ese orden parcial se incorpora un elemento en cada pasada), por lo tanto en la próxima barrida la cantidad de elementos a comparar se reduce, al menos, en uno. Puede ocurrir (debido al orden inicial) que a los elementos ya ordenados de la derecha, se incorpore más de un elemento. Para detectar esta situación y no revisar inútilmente dichos



elementos, se guarda la posición del último cambio y en la próxima barrida no se avanza más allá de esa posición.

El método finaliza cuando en una pasada no se producen cambios, implica que está ordenado ya que todos los pares cumplen  $V[i] \leq V[i+1]$ .

Se utiliza una variable K para guardar la posición del último cambio, será el límite del próximo recorrido o indicara que el arreglo está ordenado cuando  $K \leq 1$ .

Procedure Burbujeo (Var V : TV ; N : byte);

Var

Aux : integer;

i, K, Tope : byte;

Begin

Tope := N ;

Repeat

K := 0;

For i := 1 to Tope - 1 do

*{el Tope no es estático, se redefine en cada pasada}*

If  $V[i] > V[i+1]$  then

Begin

Aux := V[i]; V[i] := V[i+1]; V[i+1] := Aux

K := i ;

*{guarda la posición del último cambio}*

End;;

Tope := K ;

*{fija el tope de la próxima pasada en la posición del último cambio}*

Until  $K \leq 1$ ;

*{hasta que no se produzcan cambios, o el último cambio es en el 1er para}*

End;

Es complejo calcular la cantidad de comparaciones que realiza pues dependen del orden inicial de los elementos, pero en iguales condiciones realiza más intercambios que el método de selección.

El siguiente link explica el método, aunque no considera el cambio del Tope (que lo hace más eficiente) <https://www.youtube.com/watch?v=8Kp-8OGwphY>

### 8.3. Método de Inserción

Es un método simple y eficiente para una cantidad pequeña de datos, sobre todo si ya presentan un orden parcial. Consiste en ir generando un orden creciente en la parte izquierda del arreglo, incorporando a ese orden cada valor de la derecha.

Procedure Insercion (Var V : TV ; N : byte);

Var

Aux : integer;

i, j : byte;

Begin

For i := 2 to N do *{el índice i determina que V[i] explora el subarreglo V[i-1]..V[1]}*

Begin *{buscando la posición correcta j donde insertarlo}*

j := i - 1;

Aux := V[i];

while (j > 0) and (Aux < V[j]) do *{busca la posición y realiza un corrimiento- pag 67, pto 5.3}*

begin

V[j+1] := V[j]; j := j - 1;

end;

V[j+1] := Aux;

End;

End;

La cantidad de comparaciones es del orden de  $N^2$  (puede ser menor pues aprovecha el orden inicial)

#### 8.4. Método Shell

Recibe este nombre en honor a su creador Donald Shell y es conveniente utilizarlo cuando la cantidad de elementos a ordenar es grande.

Es una mejora del método de inserción, la diferencia radica en implementar saltos de más de una posición. Si la cantidad de elementos a ordenar es  $N$ , el salto inicial es  $N \div 2$ . El tamaño del salto se va reduciendo hasta 1. Cada etapa con un valor de salto determinado concluye cuando no se detectan cambios.

Procedure Shell (Var  $V : TV$  ;  $N : \text{byte}$ );

Var

Cambio : Boolean;

Aux : integer;

i, Paso : byte;

Begin

Paso :=  $N \div 2$ ;

Repeat

Repeat

Cambio := False;

For  $i := 1$  to  $N - \text{Paso}$  do *{enfrenta pares con distancia Paso}*

If  $V[i] > V[i + \text{Paso}]$  then

Begin

Aux :=  $V[i]$ ;  $V[i] := V[i + \text{Paso}]$ ;  $V[i + \text{Paso}] := \text{Aux}$

Cambio := True ; *{registra cambio con el Paso actual}*

End;

Until Not cambio: *{recorre con el mismo Paso hasta que no se produzcan cambios}*

Paso :=  $\text{Paso} \div 2$ ; *{redefine el paso a la mitad}*

Until Paso = 0;

End;

El siguiente link explica el método, <https://www.youtube.com/watch?v=MHW-QNd6IUE>

#### 8.5. Método Quick Sort

El método consiste en dividir el arreglo en dos partes con respecto a un elemento (Pivote) que puede ser elegido con diferentes criterios. Se ubican en la parte derecha todos los elementos mayores al pivote y en la izquierda los menores. Este proceso se repite sucesivamente con ambas partes (lo que produce que sean cada vez mas pequeñas), y cuando estas se reducen a un elemento, el arreglo está ordenado.

Se puede implementar utilizando ciclos, pero la solución más clara es *recursiva*, por lo tanto se verá como una aplicación dentro del tema *recursividad*.

**Ejercitación** (abarca los siguientes temas)

- Estructura de datos : arreglos unidimensionales
- Arreglos paralelos
- Búsqueda – Inserción -- Supresión

1. Dado un arreglo A de N elementos reales, calcular y mostrar:

- a) La suma de sus elementos.
- b) Los elementos que se encuentran en posiciones pares.
- c) El máximo y el mínimo elemento.
- d) Generar un arreglo B con los elementos de A permutado (  $B[1]=A[N]$ ,  $B[2]=A[N-1]$  ...  $B[N]=A[1]$  ).

2. Dado un archivo de enteros, se pide cargar en un vector V sólo los elementos impares y en otro vector W, los pares, ignorando si hubiese algún cero.

3. Dado un arreglo y un número K, cambiar todas las ocurrencias de K por 0.

4. Se pide compactar un arreglo (eliminar los ceros que contiene) generando otro arreglo.

5. Dado un archivo de enteros, leerlos en un arreglo, obtener el promedio de todos los números divisores del máximo y con ellos armar otro arreglo. Mostrar ambos arreglos.

6. Escriba un programa que lea una frase (secuencia de caracteres separadas por espacio, coma, punto) y desarrolle subprogramas para informar:

- a) El número de palabras que contiene.
- b) La longitud de la palabra más larga.
- c) De todas las palabras, la que aparece antes en el diccionario.
- d) La frecuencia de cada letra (mayúscula o minúscula) que aparece en el texto.

7. Escriba un programa que realice la suma de dos números positivos muy grandes (de 10 cifras, por ejemplo).

8. Dado un arreglo V de N elementos ordenados en forma ascendente con componentes repetidas, generar W con la frecuencia de los elementos de V.

Ejemplo:  $N=10$ ;  $V=(2,2,3,3,3,3,7,9,9,9) \rightarrow W=(2,4,1,3)$

9. Dado un archivo de números enteros no nulos, almacenar en un arreglo A aquellos que sean ascendentes. A partir de A generar B con la misma cantidad de elementos de A pero poniendo ceros en aquellas componentes simétricas donde la simétrica derecha no sea múltiplo de la izquierda. Escribir ambos arreglos.

Ejemplo: Archivo : 5, 7, 1, 12, 15, -10, 10, 24, -25, 26, 50, 13

$A = (5, 7, 12, 15, 24, 26, 50) \rightarrow B = (5, 0, 12, 15, 24, 0, 50)$

10. Dado un arreglo A, generar dos arreglos paralelos B y C donde B contiene la suma de los elementos de A que están entre dos negativos y C la cantidad que se sumó. Mostrarlos.

Ejemplo:

$A=(3, 5, -1, 3, 8, 2, -5, 2, 10, -4, -8, 4, 6, 7, -3, 11, 2) \rightarrow B=(13, 12, 0, 17) \ C=(3, 2, 0, 3)$

11. De una competencia de atletismo se tienen los siguientes datos del lanzamiento de jabalina:

- ✓ Nombre del competidor (ordenado ascendentemente por este dato)
- ✓ 1º lanzamiento
- ✓ 2º lanzamiento

Se pide:

- a) Cuántos superaron con el 2º lanzamiento el 1º
- b) Nombre del que registró mejor marca.
- c) Dado el Nombre de un atleta, informar la marca que realizó en ambos lanzamientos.
- d) Generar un nuevo arreglo con los participantes que pasaron a la semifinal (aquellos que superaron una marca X establecida en el 1º ó 2º lanzamiento). Mostrar ambos arreglos.

12. En un archivo de texto se ha grabado por cada línea, una cadena (es una clave encriptada que puede contener letras, números y caracteres espaciales como &, \$, #, etc, siempre termina en letra), luego uno o más blancos y tres números separados por blancos, por último uno o más blancos y una letra. Ejemplo:

mU&1#hT34\*A 80 40 30 P

Se pide cargar tres arreglos paralelos :

- VClave contendrá las claves sin los caracteres especiales, que registren en su encriptación hasta tres caracteres especiales (las claves que tengan más de tres invalidan la información de la línea, que no se almacena)
- VProm almacena el promedio de los tres números
- VCar almacena el carácter que está al final de la línea, siempre que no coincida con la última letra de la clave, en cuyo caso invalida la información de la línea, que no se almacena)

Mostrar los arreglos generados. y luego para una letra y una clave que ingresan por teclado::

- a) Para la letra ingresada, calcular e informar la clave de máximo promedio
- b) Para la clave ingresada, informar el promedio y la letra asociada (puede no existir)

Resultados: si es archivo fuera

|               |          |   |     |
|---------------|----------|---|-----|
| mU&1#hT34*A   | 80 40 30 | P |     |
| AsD4=H23**9+T | 4 78 56  | L | (1) |
| jYC%MkS6M     | 10 70 3  | M | (2) |
| MnDe3!ku#z    | 6 8 19   | A |     |



| VClave   | VProm | VCar |
|----------|-------|------|
| mU1hT34A | 50    | P    |
| MnDe3kuz | 11    | A    |

- (1) se descarta la línea porque la clave tiene más de 3 caracteres especiales
- (2) se descarta la línea porque la última letra de la clave coincide con la última de la línea

13. Leer un conjunto de números enteros, determinar cuántos terminan con 0, 1, 2,.....9

14. En una estación de peaje hay N cabinas por donde pasan varios vehículos. Desde un archivo se lee la siguiente información:

- Número de cabina
- Tipo de vehículo (1=auto, 2=moto, 3=camioneta, 4=camión, 5=colectivo)
- Hora (0..23)

Las tarifas son: moto= \$ 15, auto y camioneta=\$ 30, camión=\$70, colectivo=\$90.

Si la hora está entre 10 y 18 sufre un incremento del %10.

Calcular e informar

- a) cuántos vehículos atendió cada una de las N cabinas
- b)Cuál fue la que más importe recaudó.

15. Se pide compactar un arreglo, como en el ejercicio 4, sin usar estructura auxiliar.

16. Una empresa de autos de alquiler grabó en cada línea del archivo la patente del vehículo y el total recaudado en el día. Se pide leer los datos en dos arreglos paralelos e Informar:

- a) El promedio recaudado por auto.
- b) Las patentes de los vehículos que superaron en la recaudación en X pesos (X es dato).
- c) La patente del que recaudó menos.

17. Una planta de silos registra el ingreso de camiones de sus clientes con cereal, por cada uno se tiene:

- Cliente
- Patente del camión
- Tipo de cereal ( 1-trigo, 2-girasol, 3-soja, 4-maiz )
- Toneladas

Se pide que lea los datos desde un archivo y mediante un menú de opciones iterativo calcule e informe:

- a)Porcentaje de toneladas de cada cereal respecto del total.
- b)Dada una patente, si es que existe, decir qué tipo de cereal transporta, las toneladas y a qué cliente pertenece.
- c)Generar un arreglo con los clientes que transportaron más de X peso ordenado alfabéticamente.