

ESTRUCTURA DE DATOS (ARRAYS)

CONTENIDO

- 6.1. Introducción a la estructura de datos.
- 6.2. Arrays unidimensional: los vectores.
- 6.3. Operaciones con vectores.
- 6.4. Arrays varias dimensiones.
- 6.5. Arrays multidimensionales.
- 6.6. Almacenamiento de arrays en memoria.

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS. EJERCICIOS.

En los capítulos anteriores se ha introducido el concepto de datos de tipo simple que representan valores de tipo simple, como un número entero, real o un carácter. En muchas situaciones se necesita, sin embargo, procesar una colección de valores que están relacionados entre sí por algún método, por ejemplo, una lista de calificaciones, una serie de temperaturas medidas a lo largo de un mes, etc. el procesamiento de tales conjuntos de datos, utilizando datos simple, puede ser extremadamente difícil y por ello la mayoría de los lenguajes de programación incluyen características de estructuras de datos. La estructura de datos básicos que soportan la mayoría de los lenguajes de programación son los arrays conceptos matemáticos de “vector” y “matriz”.

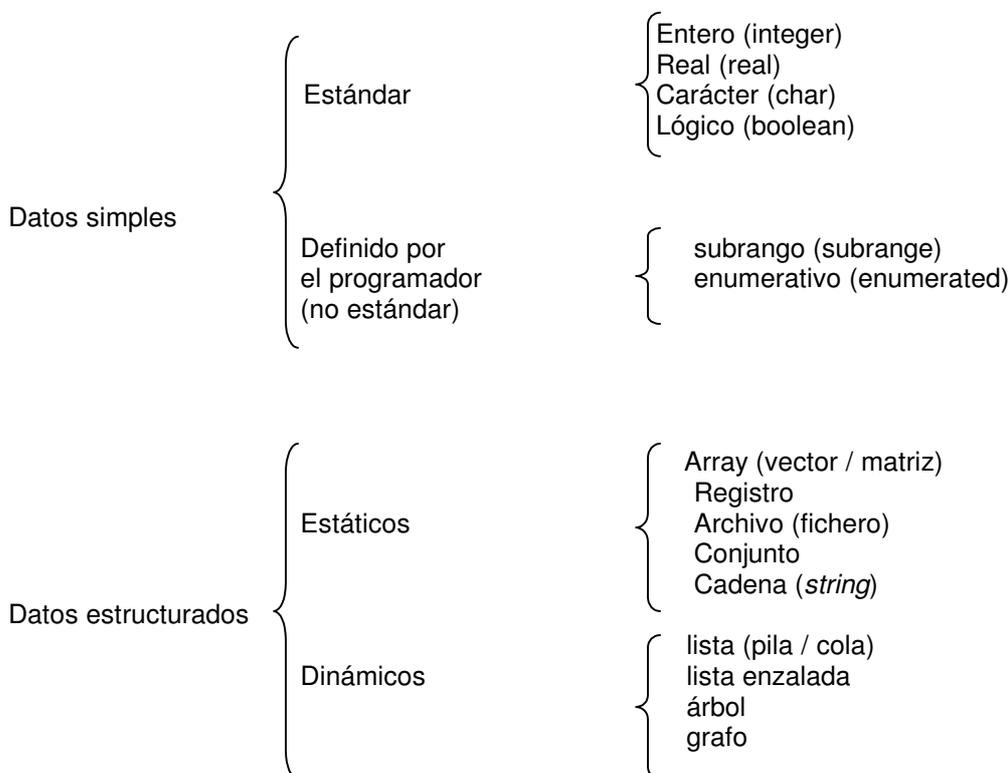
Un array (matriz, tabla, arreglo) es una secuencia de posiciones de memoria central a las que se puede acceder directamente, que contiene datos del mismo tipo y pueden ser seleccionados individualmente mediante el uso de subíndices. Este capítulo estudia el concepto de *arrays unidimensionales* y *multidimensionales*, así como el procesamiento de los mismos.

¹ En Latinoamérica, el término *array* se suele traducir por la palabra arreglo.

6.1. INTRODUCCIÓN A LAS ESTRUCTURAS DE DATOS

Una *estructura de datos* es una *colección* de datos que pueden ser caracterizados por su organización y las operaciones que definen en ella.

Las estructuras de datos son muy importante el los sistemas de computadora. Los tipos de datos más frecuentes utilizados en los diferentes lenguajes de programación son:



Los tipos de datos *simples* o *primitivos* significan que no están compuestos de otras estructuras de datos, los más frecuentes y utilizados por casi todos los lenguajes son: *enteros*, *reales* y *carácter*. (char), siendo los tipos *lógicos*, *subrango* y *enumerativos* propios del lenguaje estructurados como Pascal. Los tipos de datos compuestos estan construidos basados en tipos de datos primitivos; el ejemplo mas representativo es la *cadena* (string)de caracteres.

Los tipos de datos simples pueden ser organizados en diferentes estructuras de datos: *estáticas* y *dinámicas*. Las **estructuras estáticas** son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa. Estas estructuras están implementadas en casi todos los lenguajes; array (vectores / tablas-matrices),*registros*, *ficheros* (los conjuntos son específicos del lenguaje Pascal).

Las **estructuras de datos dinámicas** no tienen las limitaciones o restricciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas. Mediante el uso de un tipo de datos específico, denominado *puntero*, es posible consumir estructuras de datos dinámicas que son soportadas por la mayoría de los lenguajes, y en aquellos que sí tienen esas características ofrecen soluciones eficaces y efectivas en la solución de problemas complejos – Pascal es el lenguaje tipo por excelencia con posibilidad de estructuras de datos dinámicos - .La estructura dinámica por excelencia son las *listas* – enlazadas, pilas, colas - , árboles – binarios, árbol-b, de búsqueda binaria – y *grafos*.

La elección del tipo de estructura de datos idóneas a cada aplicación dependerá esencialmente del tipo de aplicación y, en menor medida, del lenguaje, ya que en aquellos en que no está implementada una estructura – por ejemplo, las listas y los árboles no los soporta

BASIC – deberá ser simulada con el algoritmo y las características del lenguaje su fácil o difícil solución.

Una característica importante que diferencia a los tipos de datos es la siguiente: los tipos de datos simples tienen como característica común que cada variable representa a un elemento; los tipos de los datos estructurados tienen como característica común que un *identificador* (nombre) puede representar múltiples datos individuales, pudiendo cada uno de éstos ser referenciado independientemente.

6.2. ARRAYS UNIDIMENSIONALES: LOS VECTORES

Un array (matriz o vector) es un conjunto finito y ordenado de elementos homogéneos. La propiedad “ordenado” significa que el elemento primero, segundo, tercero,..., enésimo de un array puede ser identificado. Los elementos de un array son homogéneos, es decir, del mismo tipo de datos. Un array puede ser compuesto de todos sus elementos de tipo entero, etc. los arrays se conocen también como matrices – en matemática – y tablas – en cálculos financieros.

El tipo más simple de array es el *array unidimensional o vector* (matriz de dimensión). Un vector de una dimensión denominado NOTAS que consta de n elementos se puede representar por la Figura 6.1.

El *subíndice o índice* de un elemento (1, 2, ..., i , n) designa su posición en la ordenación del vector. Otras posibles notaciones del vector son:

$$a_1, a_2, \dots, a_i, \dots, a_n$$

$$A(1), A(2), \dots, A(i), \dots, A(n)$$

Obsérvese que solo el vector global tiene nombre (NOTAS). Los elementos del vector se referencian por su *subíndice o índice* (subscript), es decir, posición relativa en el vector.

En algunos libros y tratados de programación, además de las notaciones anteriores se suele utilizar esta otra:

$$A(L:U) = (A(I))$$

Para $I = L, L + 1, \dots, U - 1, U$ donde cada elemento $A(I)$ es de tipo T

que significa: A , vector unidimensional con elementos de tipo T , cuyos subíndices varían en el rango de L a U , lo cual significa que el índice no tiene por que comenzar necesariamente en 0 o en 1.

como ejemplo de un vector o array unidimensional, se puede considerar el vector TEMPERATURA, que contiene las temperaturas horarias registradas en una ciudad durante los veinticuatro horas del día. Este vector constará de veinticuatro elementos de tipo real ya que las temperaturas normalmente no serán enteras siempre.

NOTAS (1)	NOTAS (2)	NOTAS (I)	NOTAS (N)
-----------	-----------	-------	-----------	-------	-----------

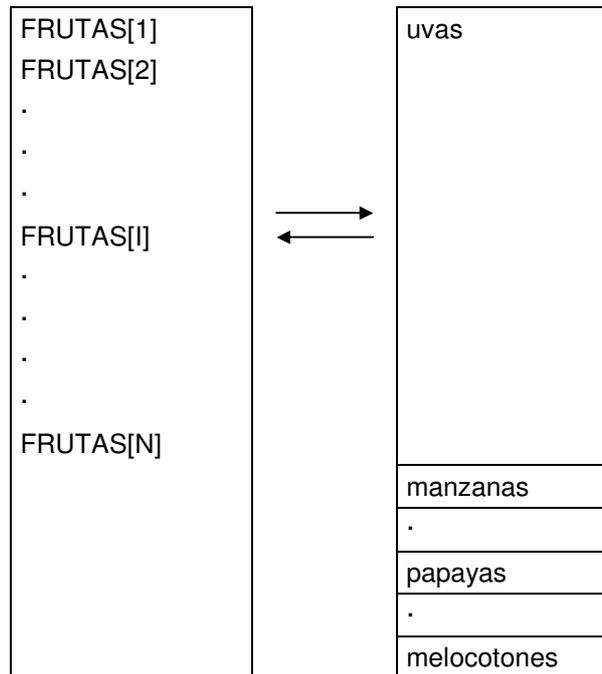
Figura 6.1. Vector.

El valor máximo permitido de un vector se denomina *límite inferior* del vector (L) y el valor máximo permitido se denomina *límite superior* (U). En el ejemplo del vector TEMPERATURAS el límite inferior es 1 y el superior 24.

$$\text{TEMPERATURA}(I) \quad \text{donde } 1 \leq I \leq 24$$

El número de elementos de un vector se denomina *rango del vector*. El rango del vector $A(L:U)$ es $U - L + 1$. el rango del vector $B(1:n)$ es n .

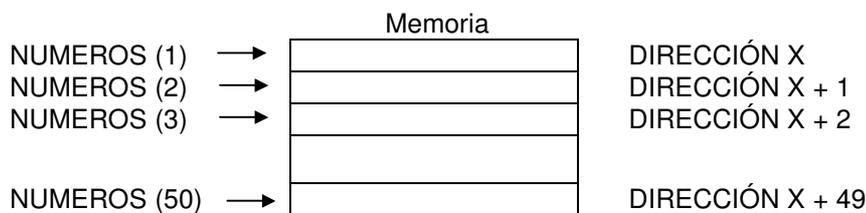
Los vectores, como ya se ha comentado, pueden contener datos no numérico, es decir, tipo "carácter". Por ejemplo, un vector que representa las frutas que se venden en un supermercado:



Otros ejemplos de un vector pueden ser los nombres de los alumnos de una clase. El vector se denomina ALUMNOS y tiene treinta elementos de rango:

ALUMNOS	
1	Luis Francisco
2	José
3	Victoria
.	
I	Martín
.	
30	Graciela

Los vectores se almacenan en memoria central del computador en un orden adyacente. Así, un vector de cincuenta números denominado NUMEROS se representa gráficamente por cincuenta posiciones de memoria sucesivas:



Cada elemento de un vector se puede procesar como si fuese una variable simple al ocupar una posición de memoria. Así,

NUMEROS [25] ! 72

almacena el valor entero real 72 en la posición 25ª del vector NUMEROS y la instrucción de salida

escribir (NUMERO [25])

visualiza el valor almacenado en la posición 25ª , en este caso 72.

Esta propiedad significa que cada elemento de un vector —y posteriormente una tabla o matriz —es accesible directamente. Ésta será una de las *ventajas* más importantes de usar un vector: *almacenar un conjunto de datos*.

Consideremos un vector x de ocho elementos:

x (1)	x (2)	x (3)	x (4)	x (5)	x (6)	x (7)	x (8)
14.0	12.0	8.0	7.0	6.41	5.23	6.15	7.25
Elemento 1º	Elemento 2º						Elemento 8º

Algunas instrucciones que manipulan este vector se representa en la Tabla 6.1.

Tabla 6.1. Operaciones básicas con vectores

Acciones	Resultados
escribir (X [1])	Visualiza el valor de X [1] o 14.0
X(4) ! 45	Almacena el valor 45 en X [4]
SUMA ! X [1] + [3]	Almacena la suma de X [1] y X [3];o bien 22.0 en la variable SUMA
SUMA ! SUMA + X [4]	Añade en la variable Suma el valor de X [4], es decir Suma = 67.0
X (5) ! X [5] + 3.5	Suma 3.5 a X [5]; EL NUEVO VALOR De X [5] será 9.91
X (6) ! X [1] + X [2]	Almacena la suma de X[1] y X[2] en X[6] el nuevo valor de X[6] será 26

Antes de pasar a tratar las diversas operaciones que se pueden efectuar con vectores, consideremos la notación de los diferentes elementos.

Supongamos un vector V de ocho elementos.

V [1]	V [2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]
12	5	-7	14.5	20	1.5	2.5	-10

Los subíndices de un vector pueden ser enteros, variables o expresiones enteras. Así por ejemplo, si

I ! 4

V [I + 1] representa el elemento V [5] de valor 20
 V [I + 2] representa el elemento V [6] de valor 1.5
 V [I - 2] representa el elemento V [2] de valor 5
 V [I + 3] representa el elemento V [7] de valor 2.5

Los arrays unidimensionales, al igual que posteriormente se verán los arrays multidimensionales, necesitan ser dimensionados previamente a su uso de un programa.

6.3. OPERACIONES CON VECTORES

Un vector como ya se ha mencionado, es una secuencia ordenada de elementos como

$$X [1], X [2], \dots, X [n]$$

El límite inferior no tiene por que empezar en uno. El vector L

$$L [0], L [1], L [3], L [4], L [5]$$

contiene seis elementos, en el que el primer elemento empieza en cero. El vector P, cuyo rango es 7 y sus límites inferior y superior son -3 y 3 , es:

$$P [-3], P [-2], P [-1], P [1], P [2], P [3]$$

Las operaciones, que se pueden realizar con vectores durante el proceso de resolución de un problema son:

- asignación,
- lectura / escritura,
- recorrido (acceso secuencial),
- actualizar (añadir, borrar, insertar),
- Ordenación,
- búsqueda.

En general, las operaciones con vectores implican el procesamiento o tratamiento de los elementos individuales del vector.

Las notaciones algorítmicas que utilizaremos en este libro son:

```
tipo
  array [dimensiones] de <tipo de dato>:<nombre_del_tipo_array>
```

```
tipo
  array [1..10] de caracter: nombres
var
  nombres: n
```

significa que nombres es un array (vector) unidimensional de diez elementos (1 a 10) de tipo carácter.

```
tipo
  array ['A' .. 'Z'] de real: lista
var
  lista: 1
```

representa un vector cuyos subíndices son A, B, . . . y cuyos elementos son de tipo real.

```
tipo
  array [0..100] de entero: numero
var
  numero: nu
```

numero es un vector cuyos subíndices van de 0 a 100 y de tipo entero.

Las operaciones que analizaremos en esta sección serán: *asignación*, *lectura/escritura*, *recorrido* y *actualización*. dejando por su especial relevancia como tema exclusivo de un capítulo la *ordenación o clasificación* y *búsqueda*.

6.3.1. Asignación

La asignación de valores a un elemento del vector se realizará con la instrucción de asignación:

`A [29] ! 5` asigna el valor 5 al elemento 20 del vector A

Si se desea asignar valores a todos los elementos de un vector, se debe recurrir a estructuras repetitivas (**desde, mientras o repetir**) e incluso selectivas (**si-entonces, según**).

`leer(A[i])`

Si se introducen los valores 5, 7, 8, 14 y 12 mediante asignaciones

```
A[1] ! 5
A[2] ! 7
A[3] ! 8
A[4] ! 14
A[5] ! 12
```

El ejemplo anterior ha asignado diferentes valores a cada elemento del vector A; si se desea dar el mismo valor a todos los elementos, la notación algorítmica se simplifica con el formato:

```
desde i = 1 hasta 5 hacer
  A[i] <- 8
fin_desde
```

donde A[i] tomará los valores numéricos A [1] = 8, A [2] = 8, ..., A [5] = 8. Se puede utilizar también la notación

`A ! 8`

para indicar la asignación de un mismo valor a cada elemento de un vector A. Esta notación se considerará con mucho cuidado para evitar confusión con posibles variables simples numéricas de igual nombre (A).

6.3.2. Lectura/escritura de datos

La lectura/escritura de datos en arrays u operaciones de entrada/salida normalmente se realizan con estructuras repetitivas, aunque puede también hacerse con estructuras selectivas. Las instrucciones simples de lectura/escritura se representarán como

Leer (A)	lectura del vector A
escribir (A)	escritura del vector A
leer (V[5])	leer el elemento V[5] del vector V

6.3.3. Acceso secuencial al vector (recorrido)

Se puede acceder a los elementos de un vector para introducir datos (leer) en él o bien para visualizar su contenido (escribir). A la operación de efectuar una acción general sobre todos los elementos de un vector se la denomina recorrido del vector. Estas operaciones se realizan utilizando estructuras repetitivas, cuyas variables de control (por ejemplo, I) se utilizan como subíndices del vector (por ejemplo, S [I]). El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos del vector,

Ejemplo 6.1

Lectura de veinte valores enteros de un vector denominado F.

Procedimiento 1:

```

algoritmo Leer_vector
tipo
  array[1..20] de entero: final
var
  final: f
inicio
  desde i ! 1 hasta 20 hacer
    leer(F[i])
  fin_desde
fin

```

La lectura de veinte valores sucesivos desde el teclado rellenará de valores el vector F, comenzando con el elemento F (1) Y terminando en F (20). Si se cambian los límites inferior y superior (por ejemplo, 5 y 10), el bucle de lectura sería:

```

desde i ! 5 hasta 10 hacer
  leer(F[i] )
fin_desde

```

Procedimiento 2:

Los elementos del vector se pueden leer también con bucles mientras o repetir.

```

i ! 1
mientras i ! 20 hacer
  leer(F[i] )
  i ! i + 1
fin_mientras

```

o bien:

```

i ! 1
repetir
  leer(F[i] )
  i ! i + 1
hasta_que i > 20

```

La salida o escritura de vectores se representa de un modo similar. La estructura

```

desde i ! 1 hasta 20 hacer
  escribir(F[i] )
fin_desde

```

visualiza todo el vector completo (un elemento en cada línea independiente).

Ejemplo 6.2

Este ejemplo procesa un array PUNTOS, realizando las siguientes operaciones; a) lectura del array, b) cálculo de la suma de los valores del array, c) cálculo de la media de los valores.

El *array* lo denominaremos PUNTOS; el límite superior del rango se fijará mediante la constante LIMITE, a la que se asigne el valor 40, y el límite inferior lo consideraremos 1.

```

algoritmo Media_puntos

```

```

const
  LIMITE = 40
tipo
  array[1..LIMITE] de real: PUNTUACIÓN
var
  PUNTUACION: PUNTOS c
  real: suma, media
  entero: i

inicio
  suma ! 0
  escribir('Datos del array')
  desde i ! 1 hasta LIMITE hacer
    leer(PUNTOS[i])
    suma ! suma + PUNTOS [i]
  fin_desde
  media ! suma/LIMITE
  escribir('La media es', media)
fin

```

Se podría ampliar el ejemplo, en el sentido de visualizar los elementos del array, cuyo valor es superior a la media. Mediante una estructura desde se podría realizar la operación, añadiéndole al algoritmo anterior.

```

escribir ('Elementos del array superior a la media')
desde i ! 1 hasta LIMITE hacer
  si PUNTOS[i] > media entonces
    escribir(PUNTOS[i] )
  fin_si
fin_desde

```

Ejemplo 6.3

Calcular la media de las estaturas de una clase. Deducir cuántos son más altos que la media y cuántos son más bajos que dicha media (Figura 6.2).

Tabla de variables

n	número de estudiantes de la clase	: entera
R[1]...H[n]	estatura de los n alumnos	: real
i	contador de alumnos	: entera
MEDIA	media de estaturas	: real
ALTOS	alumnos de estatura mayor que la media	: entera
BAJOS	alumnos de estatura menor que la media	: entera
SUMA	totalizador de estaturas	: real

6.3.4. Actualización de un vector

La operación de actualizar un vector puede constar a su vez de tres operaciones elementales:

```

añadir   elementos,
insertar elementos,
borrar   elementos,

```

Se denomina añadir datos a un vector la operación de añadir un nuevo elemento al final del vector. La única condición necesaria para esta operación consistirá en la comprobación de espacio de memoria suficiente para el nuevo elemento; dicho de otro modo, que el vector no contenga todos los elementos con que fue definido al principio del programa.

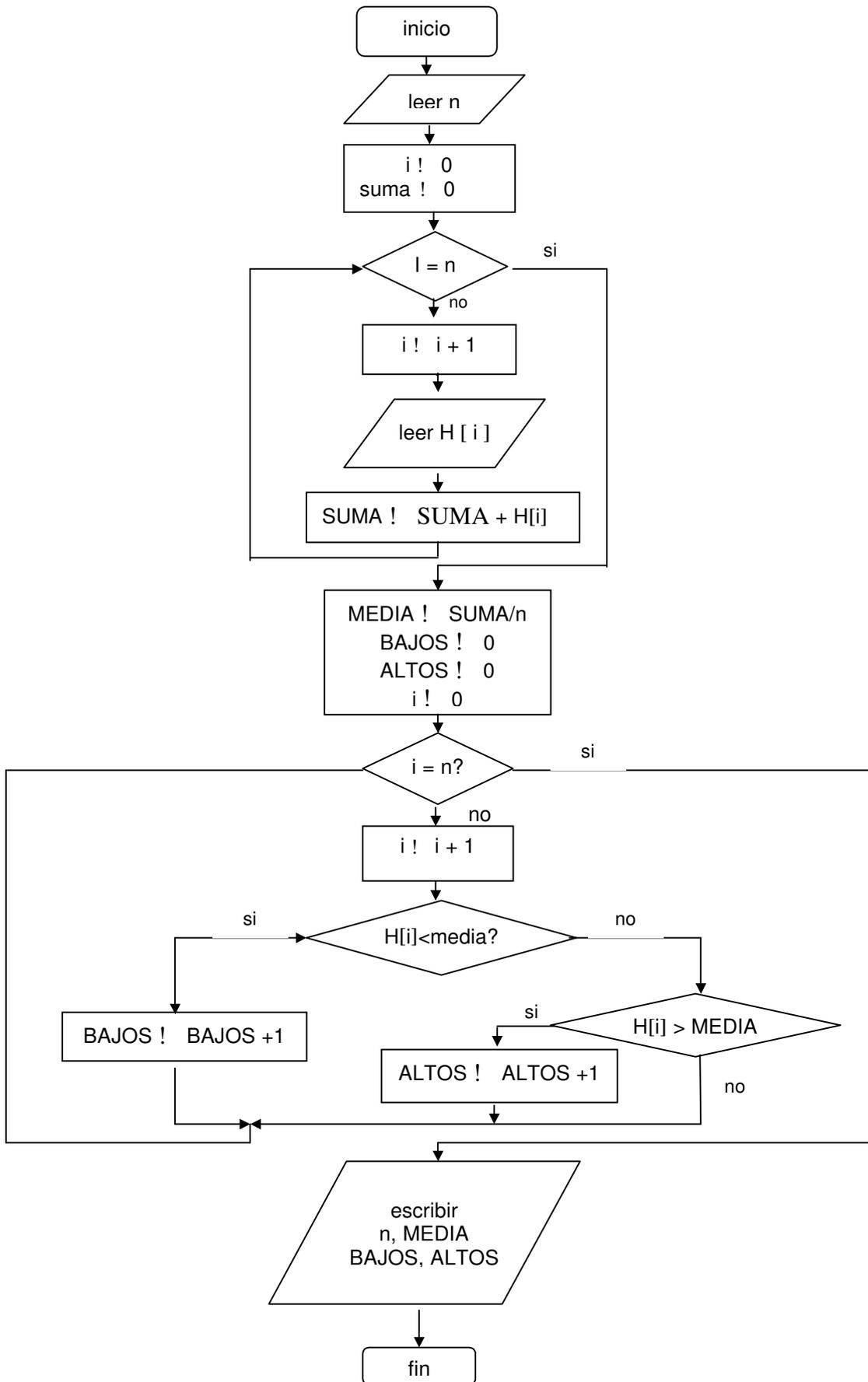


Figura 6.2 Diagrama de flujo para el cálculo de la estructura media de una clase

Ejemplos 6.4

Un array TOTAL se ha dimensionado a seis elementos pero sólo se le han asignado cuatro valores a los elementos TOTAL [1], TOTAL [2], TOTAL [3] Y TOTAL [4]. Se podrán añadir dos elementos más con una simple acción de asignación.

```
TOTAL [5] ! 14
TOTAL [6] ! 12
```

La operación de insertar un elemento consiste en introducir dicho elemento en el interior del vector. En este caso se necesita un desplazamiento previo hacia abajo para colocar el elemento nuevo en su notación relativa.

Ejemplo 6.5.

Se tiene un array COCHES2 de nueve elementos que contiene siete marcas de automóviles en orden alfabético y se desea insertar dos nuevas marcas: OPEL y CITROËN.

Como Opel está comprendido entre Lancia y Renault, se deberán desplazar hacia abajo los elementos 5 y 6, que pasarán a ocupar la posición relativa 6 y 7. Posteriormente debe realizarse la operación con Citroën, que ocupará la posición 2.

El algoritmo que realiza esta operación para un vector de n elementos es el siguiente, suponiendo que haya espacio suficiente en el vector.

1. //Calcular la posición del elemento a insertar (por ejemplo, P)
2. //Inicializar contador de inserciones i ! n
3. **mientras** i >= P **hacer**
 (transferir el elemento actual i – esimo hacia abajo, a la posición i + 1)
 COCHES [i, 1] ! COCHES [i]
 //decrementar contador
 i ! i – 1
Fin_mientras
4. //insertar el elemento en la posición P
 COCHES [P] ! ‘nuevo elemento’
5. //actualizar el contador de elementos del vector
6. n ! n + 1
7. **fin**

a) COCHES		b) Insertar OPEL		c) Insertar CITROËN	
1	Alfa Romeo	1	Alfa Romeo	1	Alfa Romeo
2	Fiat	2	Fiat	2	Citroën ←
3	Ford	3	Ford	3	Fiat
4	Lancia	4	Lancia	4	Ford
5	Renault	5	Opel ←	5	Lancia
6	Seat	6	Renault	6	Opel
7		7	Seat	7	Renault
8		8		8	Seat
9		9		9	

² En Latinoamérica, el término español COCHE no se suele utilizar para determinar un automóvil, en su lugar se suele emplear el término CARRO.

Si se deseara realizar más inserciones, habría que incluir una estructura de decisión sientonces para preguntar si se van a realizar más inserciones.

La operación de borrar un elemento al final del vector no presenta ningún problema; el borrado de un elemento del interior del vector provoca el movimiento hacia arriba de los elementos inferiores a él para reorganizar el vector.

El algoritmo de borrado del elemento j -ésimo del vector COCHES es el siguiente:

```

algoritmo Borrado
inicio
//se utilizara una variable auxiliar -AUX- que contendra el valor del
elemento que se desea borrar
AUX ! COCHES [j]
desde i ! j hasta N - 1 hacer
//llevar elemento j + 1 hacia arriba
COCHES [i] ! COCHES [i + 1]
fin_desde
//actualizar contador de elementos
//ahora tendrá un elemento menos, N - 1
N ! N - 1
fin

```

6.4. ARRAYS DE VARIAS DIMENSIONES

Los vectores examinados hasta ahora se denominan arrays unidimensionales y en ellos cada elemento se define o referencia por un índice o subíndice. Estos vectores son elementos de datos escritos en una secuencia. Sin embargo, existen grupos de datos que son representados mejor en forma de tabla o matriz con dos o más subíndices. Ejemplos típicos de tablas o matrices son: tablas de distancias kilométricas entre ciudades, cuadros horarios de trenes o aviones, informes de ventas periódicas (mes/unidades vendidas o bien mes/ventas totales), etc. Se pueden definir tablas o matrices como arrays multidimensionales, cuyos elementos se pueden referenciar por dos, tres o más subíndices. Los arrays no unidimensionales los dividiremos en dos grandes grupos:

Arrays bidimensionales	(2 dimensiones)
Arrays multidimensionales	(3 o más dimensiones)

6.4.1. Arrays bidimensionales (tablas/matrices)

El array bidimensional se puede considerar como un vector de vectores. Es, por consiguiente, un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo y en el que se necesita especificar dos subíndices para poder identificar cada elemento del array.

Si se visualiza un array unidimensional, se puede considerar como una columna de datos: un array bidimensional es un grupo de columnas, como se ilustra en la Figura 6.3.

El diagrama representa una tabla o matriz de treinta elementos (5 x 6) con 5 filas y 6 columnas.

Como en un vector de treinta elementos, cada uno de ellos tiene el mismo nombre. Sin embargo, un subíndice no es suficiente para especificar un elemento de un array bidimensional; por ejemplo, si el nombre del array es M, no se puede indicar M [3], ya que no sabemos si es el tercer elemento de la primera fila o de la primera columna. Para evitar la ambigüedad, los elementos de un array bidimensional se referencian con dos subíndices: el primer subíndice se refiere a la fila y el segundo subíndice se refiere a la columna. Por consiguiente, M[2, 3] se refiere al elemento de la segunda fila, tercera columna. En nuestra tabla ejemplo M [2, 3] contiene el valor 18.

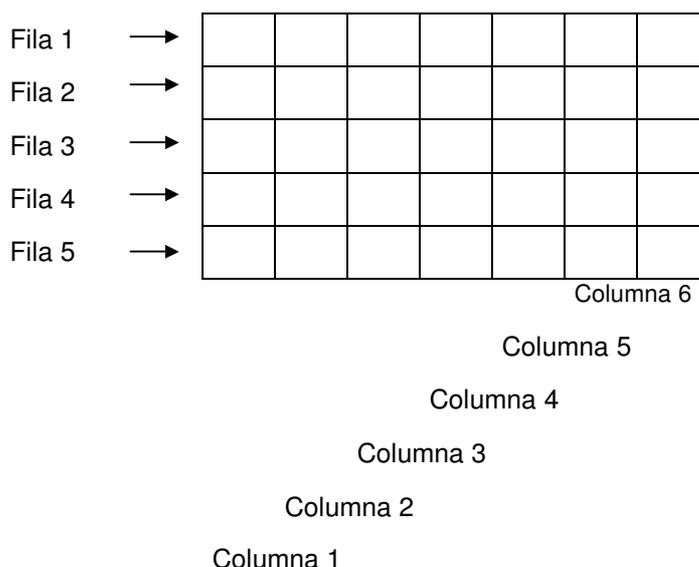


Figura 6.3. Array Bidimensional.

Un array bidimensional M , también denominado matriz (términos matemáticos) o tabla (términos financieros), se considera que tiene dos dimensiones (una dimensión por cada subíndice) y necesita un valor para cada subíndice se refiere a la fila del array, mientras que el segundo subíndice se refiere a la columna del array. Es decir, $B [I, J]$ es el elemento de B que ocupa la I^{a} fila y la J^{a} columna, como se indica en la figura 6.4.

El elemento $B [I, J]$ también se puede representar por $B_{i, j}$. Más formalmente en notación algorítmica, el array B con elementos del tipo T (numéricos, alfanuméricos, etc.) con subíndices fila que varían en el rango de 1 a M y subíndices columna en el rango de 1 a N es:

$$B [1:M, 1:N] = \{B[I, J]\}$$

	1	2	3	4	...	J	...	N
1								
2								
...								
I						B[I,J]		
...								
M								

Figura 6.4. Elemento $B[I, J]$ del array B .

donde $I = 1, \dots, M$ o bien: $1 \leq I \leq M$
 $J = 1, \dots, N$ $1 \leq J \leq N$

Cada elemento $B[I, J]$ es de tipo T .

El array B se dice que tiene M por N elementos. Existen N elementos en cada fila y M elementos en cada columna ($M * N$).

Los arrays de dos dimensiones son muy frecuentes: las calificaciones de los estudiantes de una clase se almacenan en una tabla NOTAS de dimensiones NOTAS [20, 5], donde 20 es el número de alumnos y 5 el número de asignaturas. El valor del subíndice I debe estar entre 1 y 20, y el de J entre 1 y 5. Los subíndices pueden ser variables o expresiones numéricas, NOTAS ($M, 4$) y en ellos el subíndice fila irá de 1 a M y el de columnas de 1 a N .

En general, se considera que un array bidimensional comienza sus subíndices en 0 o en 1 (según el lenguaje de programación, 0 en BASIC, 1 en FORTRAN), pero pueden tener límites seleccionados por el usuario durante la codificación del algoritmo. En general, el array

bidimensional B con su primer subíndice, variando desde un límite inferior L (inferior, low) a un límite superior U (superior, up). En notación algorítmica:

$$B [L1:U1, L2:U2] = \{B[I, J]\}$$

Donde $L1 \leq I \leq U1$
 $L2 \leq J \leq U2$

cada elemento B [I, J] s del tipo T.

El número de elementos de una fila de B es $U1 - L1 + 1$ y el número de elementos en una columna de B es $U2 - L2 + 1$. Por consiguiente, el número total de elementos del array B es $(U2 - L2 + 1) * (U1 - L1 + 1)$.

Ejemplo 6.6

La matriz T representa una tabla de notaciones de saltos de altura (primer salto), donde las filas representan el nombre del atleta y las columnas las diferentes alturas saltadas por el atleta. Los símbolos almacenados en la tabla son: x, salto válido; 0, salto nulo o no intentado.

Fila \ Columna	T					
	2.00	2.10	2.20	2.30	2.35	2.40
García	x	0	x	x	x	0
Pérez	0	x	x	0	x	0
Gil	0	0	0	0	0	0
Mortimer	0	0	0	x	x	x

Ejemplo 6.7.

Un ejemplo típico de un array bidimensional es un tablero de ajedrez. Se puede representar cada posición o casilla del tablero mediante un array, en el que cada elemento es una casilla y en el que su valor será un código representativo de cada figura del juego.

Los diferentes elementos serán:

elemento [i, j] = 0 si no hay nada en la casilla [I, j]
 elemento [i, j] = 1 si el cuadro (casilla) contiene un peón blanco
 elemento [i, j] = 2 un caballo blanco
 elemento [i, j] = 3 un alfil blanco
 elemento [i, j] = 4 una torre blanca
 elemento [i, j] = 5 una reina blanca
 elemento [i, j] = 6 un rey blanco

y los correspondientes números negativos para las piezas negras.

Ejemplo 6.8.

Supongamos que se dispone de un mapa de ferrocarriles y los nombres de las estaciones (ciudades) están en un vector denominado 'ciudad'. El diagrama N-S siguiente lee un nombre de una ciudad e imprime los nombres de las ciudades con las que están enlazada directamente. El array f puede tener los siguientes valores:

f [i, j] = 1 si existe enlace entre las ciudades I y j
 f [i, j] = 0 no existe enlace

NOTA: El array f resume la información de la estructura de la red de enlaces.

6.5. ARRAYS MULTIDIMENSIONALES

Un array puede ser definido de tres dimensiones, cuatro dimensiones, hasta de n-dimensiones. Los conceptos de rango de subíndices y número de elementos se pueden ampliar directamente desde arrays de una y dos dimensiones a estos arrays de orden más alto. En general, un array de n-dimensiones requiere que los valores de los n subíndices puedan ser especificados a fin de identificar un elemento individual del array. Si cada componente de un array tiene a subíndices, el array se dice que es sólo de n-dimensiones.

inicio	
leer nombreciudad	
i! 0	
repetir	i! i + 1 Leer ciudad[i]
hasta_que ciudad[i] = nombreciudad	
escribir nombreciudad 'está enlazada directamente a las siguientes ciudades'	
j! 0	
repetir	j! j + 1
	escribir ciudad[j]
hasta_que j = n	

El array A de n-dimensiones se puede identificar como

$A [L_1 : U_1, L_2 : U_2, \dots, L_n : U_n]$

Y un elemento individual del array se puede especificar por

$A [I_1, I_2; \dots, I_n]$

Donde cada subíndice I_k está dentro de los límites adecuados

$L_k \leq I_k \leq U_k$

Donde

$K = 1, 2, \dots, n$

El número total de elementos de un array A es:

$\prod (U_k - L_k + 1)$ \prod (símbolo del producto)

que se puede escribir alternativamente como

$(U_1 - L_1 + 1) * (U_2 - L_2 + 1) * \dots * (U_n - L_n + 1)$

Si los límites inferiores comenzasen en 1, el array se representaría por

$A [S_1, S_2, \dots, S_n]$ y un elemento individual A_{A_1, A_2, \dots, A_n}

donde

$$\begin{aligned} 1 &\leq K_1 \leq S_1 \\ 1 &\leq K_2 \leq S_2 \\ &\vdots \\ 1 &\leq K_n \leq S_n \end{aligned}$$

Ejemplo 6.9.

Un array de tres dimensiones puede ser uno que contenga los datos relativos al número de estudiantes de la universidad ALFA de acuerdo a los siguientes criterios:

- cursos (primero a quinto),
- sexo (varón/hembra),
- diez facultades.

El array ALFA puede ser de dimensiones 5 por 2 por 10 (alternativamente 10 x 5 x 2 o 10 x 2 x 5, 2 x 5 x 10, etc.). La Figura 6.6 representa el array ALFA.

El valor de elementos ALFA [I, J, K] es el número de estudiantes del curso I de sexo J de la facultad K. Para ser válido I, debe ser 1, 2, 3, 4 o 5; J debe ser 1 o 2; k debe estar comprendida entre 1 y 10 inclusive.

Ejemplo 6.10.

Otro array de tres dimensiones puede ser PASAJE que representa el estado actual del sistema de reserva de una línea aérea, donde:

$i = 1, 2, \dots, 10$	representa el número de vuelo,
$j = 1, 2, \dots, 60$	representa la fila del avión.
$k = 1, 2, \dots, 4$	representa el asiento dentro de la fila.

Entonces:

$\text{pasaje}[i, j, k] = 0$	asiento libre
$\text{pasaje}[i, j, k] = 1$	asiento ocupado

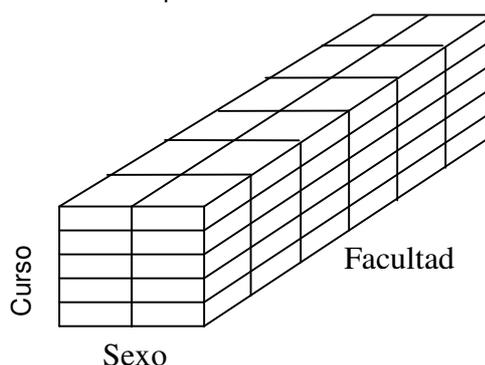


Figura 6.6 Array de tres dimensiones

6.6. ALMACENAMIENTO DE ARRAYS EN MEMORIA

Las representaciones gráficas de los diferentes arrays de una o dos dimensiones se recogen en la Figura 6.7.

Debido a la importancia de los arrays, casi todos los lenguajes de programación de alto nivel proporcionan medios eficaces para almacenar y acceder a los elementos de los arrays, de

modo que el programador no tenga que preocuparse sobre los detalles específicos de almacenamiento. Sin embargo, el almacenamiento en la computadora está dispuesto fundamentalmente en secuencia contigua, de modo que cada acceso a una matriz o tabla la máquina debe realizar la tarea de convertir la posición dentro del array en una posición perteneciente a una línea.

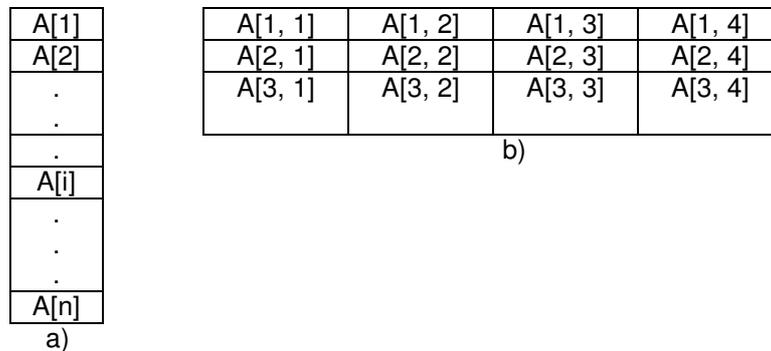
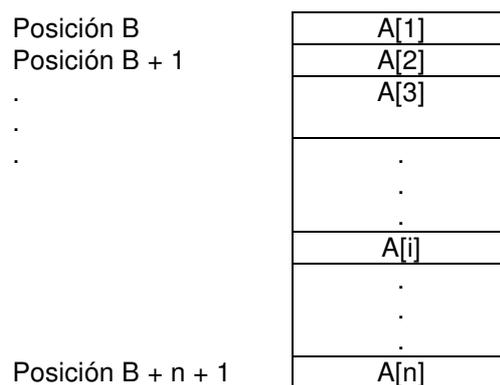


Figura 6.7. Array de una y dos dimensiones.

6.6.1. Almacenamiento de un vector

El almacenamiento de un vector en memoria se realiza en celdas o posiciones secuenciales. Así, en el caso de un vector A con un subíndice de rango 1 a n,



Si cada elemento del array ocupa S bytes (1 byte = 8 bits), y B es la dirección inicial de la memoria central de la computadora – posición o dirección base-, la dirección inicial del elemento l-ésimo sería:

$$B + (l - 1) * S$$

Nota: Si el límite inferior no es igual a 1, considérese el array declarado como N [4: 10]; la dirección inicial de N [6] es:

$$B + (l - L) * S$$

6.6.2. Almacenamiento de arrays multidimensionales.

Debido a que la memoria de la computadora es lineal, un array multidimensional debe estar linealizado para su disposición en el almacenamiento.

Los lenguajes de programación pueden almacenar los arrays en memoria de dos formas: orden de fila mayor y orden de columna mayor.

El medio más natural en que se leen y almacenan los arrays en la mayoría de los compiladores es el denominado orden de fila mayor (véase Figura 6.8.). Por ejemplo, si un array es B [2, 3], el orden de los elementos en la memoria es:

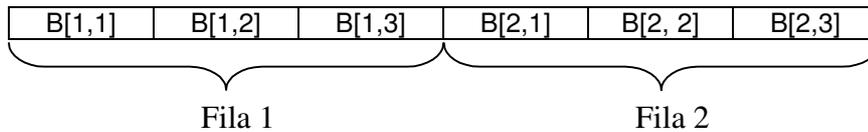


Figura 6.8. Orden de fila mayor.

C, BASIC, COBOL, PL/1 y Pascal almacenan los elementos por filas. FORTRAN emplea el orden de columna mayor, en el que las entradas de la primera fila vienen primero.

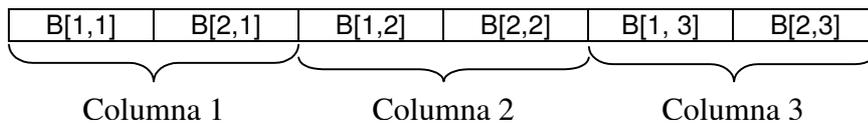


Figura 6.9. Orden de columna mayor.

De modo general, el compilador del lenguaje de alto nivel debe ser capaz de calcular con un índice [i, J] la posición del elemento correspondiente.

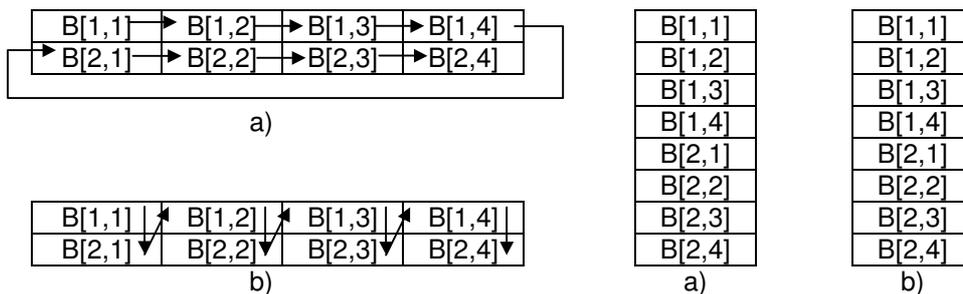


Figura 6.10. Almacenamiento de una matriz: a) por filas, b) por columnas.

En un array en orden de fila mayor, cuyos subíndices máximos sean m y n (m, filas; n, columnas), la posición p del elemento [i, j] con relación al primer elemento es:

$$P = n(i - 1) + j$$

Para calcular la dirección real del elemento [i, j] se añade p a la posición del primer elemento y se resta 1. La representación gráfica del almacenamiento de una tabla o matriz B [2, 4] y C [2, 4].

En el caso de un array de tres dimensiones, supongamos un array tridimensional A [2, 4, 3]. La Figura 6.11 representa el array A y su almacenamiento en memoria.

En orden a determinar si es más ventajoso almacenar un array en orden de columnas mayor o en orden de fila mayor, es necesario conocer en qué orden se referencian los elementos del array. De hecho, los lenguajes de programación no le dan opción al programador para que elija una técnica de almacenamiento.

Consideremos un ejemplo del cálculo del valor medio de los elementos de un array A de 50 por 300 elementos, A [50, 300]. Los algoritmos de almacenamiento respectivos serán:

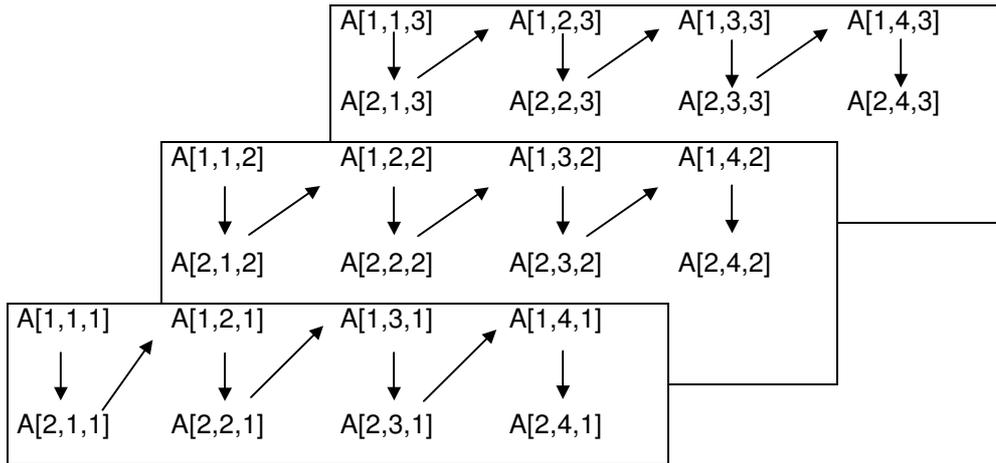


Figura 6.11. Almacenamiento de una matriz A [2, 4, 3] por columnas.

Almacenamiento por columna mayor:

```
total ! 0
desde j ! 1 hasta 300 hacer
  desde i ! 1 hasta 50 hacer
    total ! total + a [i, j]
  fin_desde
fin_desde
media ! total / (300 * 50)
```

Almacenamiento por fila mayor:

```
total ! 0
desde i ! 1 hasta 50 hacer
  desde j ! 1 hasta 300 hacer
    total ! total + a [i, j]
  fin_desde
fin_desde
media ! total / (300 * 50)
```

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS.

6.1. Escribir un algoritmo que permita calcular el cuadrado de los cien primeros números enteros y a continuación escribir una tabla que contenga dichos cien números cuadrados.

El problema consta de dos partes:

1. Cálculo de los cuadrados y almacenamiento en la tabla.
2. Escritura de la tabla T, T[1], T[2], ..., T[100] que contiene los siguientes valores:

$$\begin{aligned} T[1] &= 1 * 1 = 1 \\ T[2] &= 2 * 2 = 4 \\ T[3] &= 3 * 3 = 9 \end{aligned}$$

El algoritmo se puede construir con estructuras de decisión o alternativas, o bien con estructuras repetitivas. En nuestro caso utilizaremos una estructura repetitiva desde.

Algoritmo Cuadrados

```

tipo
  array [1..100] de entero: tabla
var
  tabla: T
  entero: I, C

inicio
  desde I ! 1 hasta 100 hacer
    C ! I * I
    T [I] ! C
  fin_desde
  desde I ! 1 hasta 100 hacer
    escribir (T[I])
  fin_desde

```

6.2. Se tienen N temperaturas. Se desea calcular su media y determinar entre todas ellas cuáles son superiores o iguales a esa media.

Análisis

En un primer momento se leen los datos y se almacenan en un vector (array unidimensional) TEMP [1:N].

A continuación se van realizando las sumas sucesivas a fin de obtener la media.

Por último, con un bucle de lectura de la tabla se va comparando cada elemento de la misma con la media y luego, mediante un contador, se calcula el número de temperaturas igual superior a la media.

Tabla de variables

N	Número de elementos del vector o tabla
TEMP	Vector o tabla de temperatura
SUMA	Sumas sucesivas de las temperaturas
MEDIA	Media de a tabla
C	Contador de temperatura >= MEDIA

Pseudocódigo

algoritmo Temperaturas

```

const
  N = 100
tipo
  array [1:N] de real; temperatura
var
  temperatura; Temp.
  entero; I, C
  real; suma, media

inicio
  suma ! 0
  media ! 0
  C ! 0

  desde I ! 1 hasta N hacer
    leer (Temp. [I])
    suma ! suma + Temp. [I]
  fin_desde
  media ! suma/N

```

```

desde I ! 1 hasta N hacer
  si Temp. [I] >= media entonces
    C ! C + 1
    escribir (Temp. [I])
  fin_si
fin_desde
escribir: La media es: (media)
escribir: El total de temperaturas >= , media, es; (C)
fin

```

6.3. Escribir el algoritmo que permita sumar el número de elementos positivos y el de negativos de una tabla T.

Sea una tabla de dimensiones M, N leídas desde el teclado.

Tabla de variables

I, J, M, N; entero
 SP; real
 SN; real

Pseudocódigo

algoritmo Suma_resta

```

const
  N = 50
  N = 20
tipo
  array [1..M, 1..N] de real; tabla
var
  tabla: T
  entero: I, J
  real: SP, Sn
inicio
  SP ! 0
  SN ! 0
  desde I ! 1 hasta N hacer
    desde J ! 1 hasta N hacer
      si T[I, J] >= 0 entonces
        SP ! SP + T [I, J]
      si_no
        SN ! SN + T [I, J]
      fin_si
    fin_desde
  fin_desde
  escribir (Suma de positivos, Sp, de negativos, SN)
fin

```

6.4. Inicializar una matriz de dos dimensiones con un valor constante dado K

Análisis

El algoritmo debe tratar de asignar la constante K a todos los elementos de la matriz A

$$\begin{array}{l}
 A[1, 1] = K \quad A[1, 2] = K \quad \dots \quad A[1, K] = k \\
 \cdot \\
 \cdot \\
 A[N, 1] = K \quad A[N, 2] = K \quad \dots \quad A[M, N] = K
 \end{array}$$

Dado que es una matriz de dos dimensiones, se necesitan dos bucles anidados para recorrer todos sus elementos.

Pseudocódigo

```

algoritmo inicializa_matriz
inicio
  k ! 0
  desde I ! 1 hasta M hacer
    desde J ! 1 hasta N hacer
      A [J, 2] ! K
    fin_desde
  fin_desde
fin

```

6.5. Realizar la suma de dos matrices bidimensionales.

Análisis

Las matrices $A[I, J]$, $B[I, J]$ para que se puedan sumar deben tener las mismas dimensiones. La matriz suma $S[I, J]$ tendrá iguales dimensiones y cada elemento será la suma de las correspondientes matrices A y B. Es decir.

$$S[I, J] = A[I, J] + B[I, J]$$

Dado que se trata de matrices de dos dimensiones, el proceso se realizará con dos bucles anidados.

Pseudocódigo

```

algoritmo Suma_matrices
inicio
  desde I ! 1 hasta N hacer
    desde J ! 1 hasta M hacer
      S[I, J] ! A[I, J] + B[I, J]
    fin_desde
  fin_desde
fin

```

6.6. Se tiene una tabla T de dos dimensiones. Calcular la suma de sus elementos. Supongamos sean M y N las dimensiones de T, y que sus componentes son números reales.

Tabla de variables

I	Contador de filas
J	Contador de columnas
M	Número de filas de la tabla T
N	Número de columnas de la tabla T
T	Tabla
S	Suma de los elementos de la tabla
I, J, M, N	Enteros
T, S	Reales

Pseudocódigo

algoritmo Suma_elementos

Const

M = 50

N = 20

tipo

array[1..M, 1..N] de real; tabla

var

entero: I, J

tabla: T

real: S

inicio

desde I ! 1 **hasta** M **hacer**

desde J ! 1 **hasta** N **hacer**

leer(T[I, J])

fin_desde

fin_desde

S ! 0 //inicialización de la suma S

desde I ! 1 **a** M **hacer**

desde J ! 1 **a** N **hacer**

 S ! S + T[I, J]

fin_desde

fin_desde

escribir('La suma de los elementos de la matriz = ', S)

fin

- 6.7. Realizar la búsqueda de un determinado nombre en una lista de nombres, de modo que el algoritmo imprima los siguientes mensajes según el resultado:

'Nombre encontrado' si el nombre está en la lista

'Nombre no existe' si el nombre no está en la lista

Se recurrirá en este ejercicio a utilizar un interruptor SW, de modo que si SW = falso, el nombre no existe en la lista, y si SW = cierto, el nombre existe en la lista (o bien, caso de no existir la posibilidad de variables lógicas, definir SW como SW = 0, si es falso, y SW = 1, si es verdadero o cierto).

Método1:

algoritmo Búsqueda

const

N = 50

tipo

array[1..N] de cadena: listas

var

listas: 1

lógico: SW

cadena: nombre

entero: y

inicio

SW ! falso

leer(nombre)

desde I ! 1 **hasta** N **hacer**

si l[I] = nombre **entonces**

```

        SW ! verdadero
    fin_si
fin_desde
si SW entonces
    escribir('Encontrado')
si_no
    escribir('No existe', nombre)
fin_si
fin

```

Método 2:

```

algoritmo Búsqueda
const
    N = 50
tipo
    array[1..N] de cadena: listas
var
    listas: 1
    entero: SW
    cadena: nombre
    entero: y

inicio
    SW ! 0
    leer(nombre)
    desde I ! 1 hasta N hacer
        si l[I] = nombre entonces
            SW ! 1
        fin_si
    fin_desde
    si SW = 1 entonces
        escribir('Encontrado')
    si_no
        escribir('No existe', nombre)
    fin_si
fin

```

- 6.8. Se desea permutar las filas I y J de una matriz (array) de dos dimensiones ($M * N$): M filas, N columnas.

Análisis

La tabla T ($M * N$) se puede representar por:

```

T[1, 1]   T[1, 2]   T[1, 3]   ... T[1, N]
T[2, 1]   T[2, 2]   T[2, 3]   ... T[2, N]
.
.
T[M, 1] T[M, 2] T[M, 3] ... T[M, N]

```

El sistema para permutar globalmente toda la fila I con la fila J se debe realizar permutando uno a uno el contenido de los elementos $T[I, K]$ y $T[J, K]$.

Para intercambiar entre sí los valores de dos variables, recordemos que se necesitaba una variable auxiliar AUXI.

Así, para el caso de las variables A y B:

```

AUXI ! A
A ! B
B ! AUXI

```

En el caso de nuestro ejercicio, para intercambiar los valores T[I, K] y T[J, K] se debe utilizar el algoritmo:

```
AUXI  ! T[I, K]
T[I, K] ! T[J, K]
T[J, K] ! AUXI
```

Tabla de variables

```
I, J, K, M, N  entero
AUXI           entero
T              tabla
```

Pseudocódigo

algoritmo intercambio

const

M = 50

N = 30

tipo

array[1..M, 1..N] de entero: tabla

var

tabla: T

entero: AUXI, I, J, K

inicio

//En este ejercicio y dado que ya se han realizado muchos ejemplos de lectura de

//arrays con dos bucles desde, la operación de lectura completa del array se

//representará con la construcción de leertabla (T)

leertabla (T)

//Deducir I, J a intercambiar

leer(I, J)

desde K ! 1 **hasta** N **hacer**

Auxi ! T[I, K]

T[I, K] ! T[J, K]

T[J, K] ! Auxi

fin_desde

//Escritura del nuevo array

escribirtabla (T)

fin

EJERCICIOS

6.1. Algoritmo que nos permita calcular la desviación estándar (SIGMA) de una lista de N números (N <= 15). Sabiendo que:

$$\text{Desviación} = \sqrt{\frac{\sum_{i=1}^n (x_i - m)^2}{n - 1}}$$

```

algoritmo Calcular_desviación
  tipo
    array [1..15] de real: arr
  var
    arr      : x
    entero   : n

  inicio
    llamar_a leer_array (x, n)
    escribir ('La desviación estandar es', desviación (x, n))
  fin

procedimiento leer_array (S arr:x; S entero:n)
  var
    entero : i
  inicio
    repetir
      escribir ('Diga numero de elementos de la lista')
      leer (n)
    hasta_que n <= 15
    escribir ('Deme los elementos')
    desde i ! 1 hasta n hacer
      leer (x [i])
    fin_desde
  fin_procedimiento

real función desviación (E arr : x; E entero : n)
  var
    real      : suma, xm, sigma
    entero    : i
  inicio
    suma ! 0
    desde i ! 1 hasta n hacer
      suma ! suma + x [i]
    fin_desde
    xm ! suma / n
    sigma ! 0
    desde i ! 1 hasta n hacer
      sigma ! sigma + cuadrado (x [i] - xm)
    fin_desde
    devolver (raiz2 (sigma / (n - 1)))
  fin_funcion

```

6.2. Utilizando arrays, escribir un algoritmo que visualice un cuadrado mágico de orden impar n, comprendido entre 3 y 11. El usuario debe elegir el valor de n.

Un cuadrado mágico se compone de números enteros comprendidos entre 1 y n. La suma de los números que figuran en cada fila, columna y diagonal son iguales.

Ejemplo: 8 1 6
 3 5 7
 4 9 2

Un método de generación consiste en situar el número 1 en el centro de la primera fila, el número siguiente en la casilla situada por encima y a la derecha y así sucesivamente. El cuadrado es cíclico, la línea encima de la primera es de hecho la última y la columna a la derecha de la última es la primera. En el caso de que el número generado caiga en una casilla ocupada, se elige la casilla que se encuentre debajo del número que acaba de ser situado.

```

Algoritmo Cuadrado_mágico
  var entero : n
  inicio
  repetir
    escribir ('Deme las dimensiones del cuadrado (3 a 11)')
    leer (n)
    hasta_que (n mod 2 <> 11) y (n >= 3)
    dibujarcuadrado (n)
  fin

procedimiento dibujarcuadrado (E entero : n)
  var array [1..11,1..11] de entero : a
      entero : i, j, c
  inicio
  i ! 2
  j ! n div 2

  desde c ! 1 hasta n * n hacer
    i ! i - 1
    j ! j + 1
    si j > n entonces
      j ! 1
    fin_si
    si i < 1 entonces
      i ! n
    fin_si
    a [i, j] ! c
    si c mod n = 0 entonces
      j ! j - 1
      i ! i + 2
    fin_si
  fin_desde

  desde i ! 1 hasta n hacer
    desde j ! 1 hasta n hacer
      escribir (a [i,j])
      //al codificar esta instrucción en un lenguaje sera
      conveniente utilizar el
      //parámetro correspondiente de "no avance de línea" en la
      salida en
      //pantalla o impresora
    fin_desde
    escribir (NL)
    //NL, representa Nueva Linea; es decir, avance de línea
  fin_desde

fin_procedimiento

```

6.3. Obtener un algoritmo que efectúe la multiplicación de dos matrices A, B:

A ! m * p elementos
 B ! p * n elementos

Matriz producto C ! m * n elementos, tal que:

$$C_{i,j} = \sum_{k=1}^p a_{i,k} * b_{k,j}$$

```

algoritmo Multiplicar_matrices

  tipo array [1..10, 1..10] de real : arr
  var entero : m, n, p
      arr    : a, b, c

inicio
  repetir
    escribir ('Dimensiones de la 1ª matriz (nº filas nº columnas)')
    leer (m, p)
    escribir ('Columnas de la 2ª matriz')
    leer (n)
  hasta_que (n <= 10) y (m < > 10) y (p <= 10)
  escribir ('Deme elementos de la 1ª matriz')
  llamar_a leer_matriz (b, p, n)
  llamar_a calcescrproducto (a, b, c, m, p, n)
fin

procedimiento leer_matriz (S arr:matriz; E/entero:filas, columnas)
var entero: i, j

inicio
  desde i ! 1 hasta filas hacer
    escribir ('Fila ', i, ':')
    desde j ! 1 hasta columnas hacer
      leer (matriz [i, j])
    fin_desde
  fin_desde
fin_procedimiento

procedimiento calcescrproducto (E arr: a, b, c; E entero: m, p, n)
var entero : i, j, k

inicio
  desde i ! 1 hasta m hacer
    desde j ! 1 hasta n hacer
      c [i,j] ! c [i,j] + a[i,j] * b [k, j]
    fin_desde
    escribir (c[I,j]) //no avanzarlinea
  fin_desde
  escribir (NL) //avanzar linea, nueva linea
fin_desde
fin_procedimiento

```

6.4. Algoritmo que triángule una matriz cuadrada y halle su determinante. En las matrices cuadradas el valor del determinante coincide con el producto de los elementos de la diagonal de la matriz triangulada, multiplicado por -1 tantas veces como hayamos intercambiado filas al triangular la matriz.

Proceso de triangulación de una matriz

para i **desde** 1 **a** n - 1 **hacer**:

a) Si el elemento de lugar (i,i) es nulo, intercambiar filas hasta que dicho elemento sea no nulo o agotar los posibles intercambios.

b) A continuación se busca el primer elemento no nulo de la fila i -ésima y, en el caso de existir, se usa para hacer ceros en la columna de abajo.

Sea dicho elemento matriz $[i,r]$

Multiplicar fila i por matriz $[i + 1,r] / \text{matriz } [i,r]$ y restarlo a la $i + 1$

Multiplicar fila i por matriz $[i + 2,r] / \text{matriz } [i,r]$ y restarlo a la $i + 2$

.....
Multiplicar fila i por matriz $[m,r] / \text{matriz } [i,r]$ y restarlo a m

algoritmo Triangulacion_matriz

const $m = \langle \text{expresión} \rangle$

$n = \langle \text{expresión} \rangle$

tipo array $[1..m, 1..n]$ **de** real : arr

var arr : matriz

real : dt

inicio

llamar_a leer_matriz (matriz)

llamar_a triangula (matriz, dt)

escribir ('Determinante= ', dt)

fin

procedimiento leer_matriz (S arr : matriz)

var entero: i, j

inicio

escribir ('Deme los valores para la matriz')

desde i ! 1 **hasta** n **hacer**

desde j ! 1 **hasta** n **hacer**

leer (matriz [i, j])

fin_desde

fin_desde

fin_procedimiento

procedimiento escribir_matriz (E arr : matriz)

var entero : i, j

 carácter : c

inicio

escribir ('matriz')

desde i ! 1 **hasta** m **hacer**

desde j ! 1 **hasta** n **hacer**

escribir (matriz [i, j]) ' //Al codificar, usar el parámetro
 //de 'no avance de linea''

fin_desde

escribir (NL)

fin_desde

escribir ('Pulse tecla para continuar')

leer (c)

fin_procedimiento

procedimiento interc (E/S real ; a,b]

var real : auxi

inicio

 auxi ! a

 a ! b

 b ! auxi

fin_procedimiento

```

procedimiento triangula (E arr: matriz; S real: dt)
var entero : signo
      entero : t, r, i, j
      real   : cs
inicio
  signo ! 1
  desde i ! 1 hasta m - 1 hacer
    t ! 1
    si matriz [i, i] = 0 entonces
      repetir
        si matriz [i + t, i] <> 0 entonces
          signo ! signo * [-1]
          desde j ! 1 hasta n hacer
            llamar_a interc (matriz [i, j], matriz [i + t, j])
          fin_desde
          llamar_a escribir_matriz (matriz)
        fin_si
      t ! t + 1
      hasta_que (matriz [i, i] <> 0) o (t = m - i + 1)
    fin_si
    r ! i - 1
    repetir
      r ! r + 1
      hasta_que (matriz [i, r] <> 0) o (r = n)
      si matriz [i, r] <> 0 entonces
        cs ! matriz [t, r]
        desde j ! r hasta n hacer
          matriz [t, j] ! matriz [t, j] - matriz [i, j] * (cs /
            matriz [i, r])
        fin_desde
        llamar_a escribir_matriz (matriz)
      fin_si
    fin_desde
  fin_si
  dt ! signo
  desde i ! 1 hasta m hacer
    dt ! dt * matriz [i, i]
  fin_desde
fin_procedimiento

```

6.5. Determinar los valores de I, J, después de la ejecución de las instrucciones siguientes:

```

var
  I, J: entero
  A: array [1..10] de entero
Inicio
  I ! 1
  J ! 2
  A [I] ! J
  A [J] ! I
  A [J + I] ! I + J
  I ! A [I] + A [J]
  A [J] ! 5
  J ! A [I] - A [J]
fin

```

- 6.6. Escribir el algoritmo que permita obtener el número de elementos positivos de una tabla.
- 6.7. Rellenar una matriz identidad de 4 por 4.
- 6.8. Leer una matriz de 3 por 3 elementos y calcular la suma de cada una de sus filas y columnas, dejando dichos resultados en dos vectores, uno de la suma de las filas y otro de las columnas.
- 6.9. Cálculo de la suma de todos los elementos de un vector, así como la media aritmética.
- 6.10. Calcular el número de elementos negativos, cero y positivos de un vector dado de sesenta elementos.
- 6.11. Calcular la suma de los elementos de la diagonal principal de una matriz cuatro por cuatro (4 x 4).
- 6.12. Se dispone de una tabla T de cincuenta números reales distintos de cero. Crear una nueva tabla en la que todos sus elementos resulten de dividir los elementos de la tabla T por el elemento T [K], siendo K un valor dado.
- 6.13. Se dispone de una lista (vector) de N elementos. Se desea diseñar un algoritmo que permita insertar el valor x en el lugar k-ésimo de la mencionada lista.
- 6.14. Se desea realizar un algoritmo que permita controlar las reservas de plazas de un vuelo MADRID-CARACAS, de acuerdo con las siguientes normas de la compañía aérea:
 Número de plazas del avión: 300.
 Plazas numeradas de 1 a 100: fumadores.
 Plazas numeradas de 101 a 300: no fumadores.

Se debe realizar la reserva a petición del pasajero y cerrar la reserva cuando no haya plazas libres o el avión esté próximo a despegar. Como ampliación de este algoritmo, considere la opción de anulaciones imprevistas de reservas.

- 6.15. Cada alumno de una clase de licenciatura en Ciencias de la Computación tiene notas correspondientes a ocho asignaturas diferentes, pudiendo no tener calificación en alguna asignatura. A cada asignatura le corresponde un determinado coeficiente.
1. Escribir un algoritmo que permita calcular la media de cada alumno.
 2. Modificar el algoritmo para obtener las siguientes medias:
 - general de la clase
 - de la clase en cada asignatura,
 - porcentaje de faltas (no presentado a examen).
- 6.16. Se dispone de las notas de cuarenta alumnos. Cada uno de ellos puede tener una o varias notas. Escribir un algoritmo que permita obtener la media de cada alumno y la media de la clase a partir de la entrada de las notas desde el terminal.
- 6.17. Una empresa tiene diez almacenes y necesita crear un algoritmo que lea las ventas mensuales de los diez almacenes, calcule la media de ventas y obtenga un listado de los almacenes cuyas ventas mensuales son superiores a la media.
- 6.18. Se dispone de una lista de cien números enteros. Calcular su valor máximo y el orden que ocupa en la tabla.
- 6.19. Un avión dispone de ciento ochenta plazas, de las cuales sesenta son de 'no fumador' y numeradas de 1 a 60 y ciento ochenta plazas numeradas de 61 a 180. Diseñar un algoritmo que permita hacer la reserva de plazas del avión y se detenga media hora antes de la salida del avión, en cuyo momento se abrirá la lista de espera.
- 6.20. Calcular las medias de las estaturas de la clase. Deducir cuántos son más altos que la media y cuántos más bajos que dicha media.
- 6.21. Las notas de un colegio se tienen en una matriz de 30 x 5 elementos (30, número de alumnos; 5, número de asignaturas). Se desea listar las notas de cada alumno y su media. Cada alumno tiene como mínimo dos asignaturas y máximo cinco, aunque los alumnos no necesariamente todos tienen que tener cinco materias.
- 6.22. Dado el nombre de una serie de estudiantes y las calificaciones obtenidas en un examen, calcular e imprimir la calificación media, así como cada calificación y la diferencia con la media.
- 6.23. Se introducen una serie de valores numéricos desde el teclado, siendo el valor final de entrada de datos o centinela -99. Se desea calcular e imprimir el número de valores leídos, la suma y media de los valores y una tabla que muestre cada valor leído y sus desviaciones de la media.
- 6.24. Se tiene una lista de N nombres de alumnos. Escribir un algoritmo que solicite el nombre de un alumno, busque en la lista (array) si el nombre está en la lista.

UNIVERSIDAD CATÓLICA SANTO TORIBIO DE
MOGROVEJO

TEMA :

**CAPÍTULO VI DE FUNDAMENTOS DE
PROGRAMACIÓN**

(Luis joyanes Aguilar)

CURSO :

ALGORITMO

INTEGRANTES :

- Cabrejos Vilela Cesar.
- Castellanos Gonzalez Gustavo.
- Custodio Saavedra Anibal.
- Espinoza Dávila Lisseth.
- Primo Bonilla Roberto.
- Quevedo Willis Carlos Daniel.

PROFESOR :

Luis Barrueto Chunga.

Chiclayo, mayo de 2003