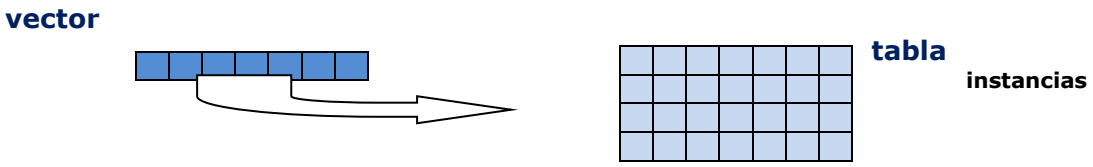
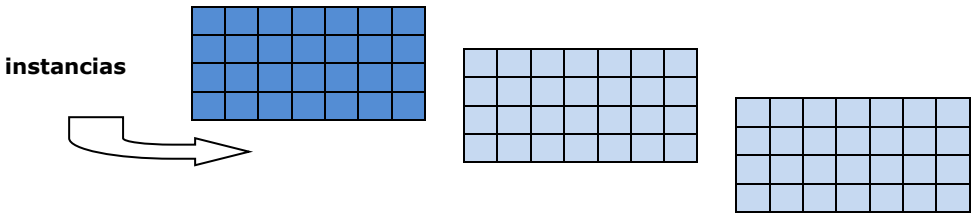
Estructuras de datos:
Arrays Multidimensionales: Tablas y Cubos

Arrays Multidimensionales: **Las Matrices**
1. Introducción.

Los arrays de varias dimensiones se pueden considerar como una extensión de los arreglos unidimensionales o vectores, es decir como instancias de n-filas. De esta manera se obtienen ***Tablas*** o estructuras de doble entrada (filas, columnas) para almacenar datos.



Así mismo, si estas tablas o planos Bi-*dimensionales*, a su vez se extienden a n-planos, obtenemos estructuras de n-tablas o arrays *tridimensionales (Cubos)*.

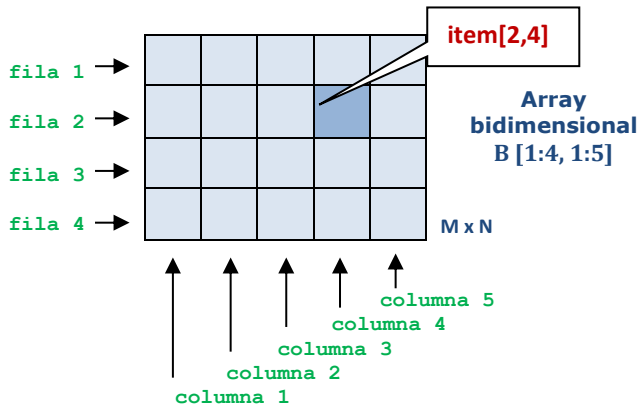


En general, se pueden definir arrays multidimensionales (n-dimensiones), cuyos elementos se pueden referenciar por dos, tres, o más sub-índices. Las arrays Multidimensionales o de varias dimensiones, más utilizados son:

arrays Bidimensionales	Tablas (2 dimensiones)
arrays Tridimensionales	Cubos (3 dimensiones)

2. Arrays bidimensionales: Tablas o matrices de segundo orden.

Considerados como vector de vectores. En consecuencia para identificar cada uno de sus elementos, es necesario especificar dos *subíndices*.



2.1. Items: Notación estándar.

Normalmente el *primer subíndice* se refiere a la ***fila*** del array, en tanto que el *segundo subíndice* se refiere a la **columna** del array.

Es decir: **b[i, j]** es el elemento del array **b** de tipo T,
que ocupa la **i-esima** fila y la **j-esima** columna

2.2. Matrices: Notación Algorítmica.

tipo
array[L1:U1, L2:U2] ***de*** <*tipo_dato*> : <*identificador_tipo_array*>

Formalmente, 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, se denota asi:

B[1:M, 1:N] = {b[i, j]}

Donde: 1 <= i <= M es el rango valido para las filas
1 <= j <= N es el rango valido para las columnas

Y cada elemento b[i, j] es de tipo T

2.3. Rangos y elementos de un array Bi-dimensional.

Si definimos el siguiente array de dos-dimensiones:

tipo
array[L1:U1, L2:U2] **de** <T>: arrTabla
var
arrTabla: tabla

Entonces el conjunto de elementos del array **tabla** es:

M[L1:U1, L2:U2] = {m[i, j]}

En donde los limites definen:
L1 <= i <= U1 elementos fila
L2 <= j <= U2 elementos columna

Y cada elemento tabla[i, j] es de tipo T (entero, real, caracter).

El número de *filas* o primera dimensión es:

M filas = (U1 – L1) + 1

y el número de *columnas* o segunda dimensión es:

N columnas = (U2 – L2) + 1

Por consiguiente, el número *total* de elementos del array **tabla** es:

(U1 –L1 + 1) * (U2 – L2 + 1)

Los tipos de datos estructurados como los *arrays* son conjuntos de variables que representan mediante un identificador o nombre a múltiples datos o elementos, cada uno de estos elementos, es referenciado independientemente por subíndices de orden *n-dimensiones.*

El **almacenamiento de** los elementos de un **array en la memoria** de una computadora, esta dispuesto fundamentalmente en **secuencia contigua**.

Ejemplo de Definición. Tabla de registro para temperaturas medias del mes de Noviembre del 2006 (7 dias x 4 semanas).

```
tipo
      array[1:4, 1:7] de real: arrTemperatura
var
      arrTemperatura:  tempNov
```

tempNov[i,j]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[1]	22.4	22.7	22.3	21.8	21.9	22.6	23.0
[2]	21.3	21.9	23.0	22.5	22.7	23.0	23.1
[3]	22.8	23.0	22.5	21.8	21.9	22.0	22.5
[4]	22.5	22.8	22.9	23.0	23.3	23.2	23.0

Significa que, **tempNov** es un array de 2-dimensiones (orden 2) con elementos de tipo real y subíndices fila de 1 a 4 (cuatro filas) y subíndices columnas de 1 hasta 7 (siete columnas), con un total de: ((4-1) + 1) * ((7-1) + 1) = 4 * 7, o 28 elementos.

Dónde: **tempNov[i, j]** está en el rango `1 <= i <= 4` para las filas
`1 <= j <= 7` para las columnas
y cada elemento **tempNov[i, j]** es de tipo real.

3. Arrays Multi-dimensionales. (orden n-dimensiones)

Los arreglos, pueden ser definidos de una, dos hasta n-dimensiones. En general, un arreglo de orden más alto, requiere que los valores de n-subíndice sean especificados con el propósito de identificar cada elemento de manera única en el array.

*"Si cada elemento de un array tiene **n-subíndices**, se dice entonces que el array es de **n-dimensiones**".*

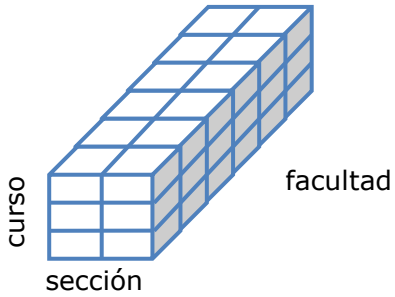


Fig. Array de 3-dimensiones

3.1. Notaciones.

El arreglo A de n-dimensiones se define asi:

tipo
array[L1:U1, L2:U2, … Ln:Un] ***de*** *<tipo_dato> : <identificador_tipo_array>*

y un elemento individual del arreglo queda identificado por: **A[i, j, …, k_n]**
Donde cada subíndice I_k esta dentro de los limites:

$$L_k <= I_k <= U_k \quad \text{siendo k = 1, 2, 3, …, n dimensiones}$$

El numero total de elementos del array A es: $\prod (U_k - L_k + 1)$ siedo $\prod$ símbolo de producto, y que se puede extender a la siguiente expresión:

$$\text{rango} = (U_1 - L_1 + 1) * (U_2 - L_2 + 1) * \ldots * (U_n - L_n + 1)$$

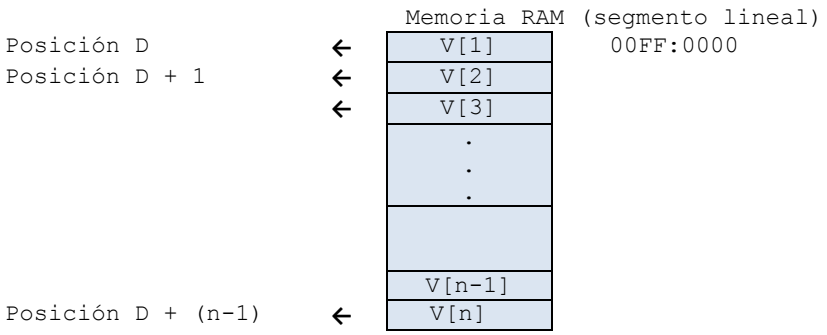
4. Almacenamiento de Arrays en memoria.

En general, los lenguajes de programación y en especial los compiladores, proporcionan medios eficaces para almacenar y acceder a los elementos de un array, abstrayendo para el programador los detalles específicos de almacenamiento y administración de la memoria RAM de la PC.

Sin embargo, es necesario recordar que el almacenamiento de los arrays, esta dispuesto fundamentalmente en secuencia inmediata, de modo que la tarea de acceso a los elementos de un arreglo por parte de la maquina, se realiza mediante la sincronización de la ***posición indexada del array*** y una posición de memoria perteneciente a un ***segmento lineal de memoria*** (*direcciones de memoria en secuencia contigua*).

4.1. Almacenamiento de un vector.

El los vectores se ubican en posiciones secuenciales. Así, si el vector V con subíndices de rango 1 a n se representan en memoria:



Así por ejemplo, si cada elemento del array ocupa B bytes (1 byte = 8 bits), y D es la dirección inicial en la memoria RAM de la computadora (representado como *segmento:offset* –dirección base o posición en hexadecimal-), la posición o dirección del elemento i-esimo seria:

$$D + (i - 1) * B$$

En general, el elemento m[i] de un array definido como M[L:U], tiene la dirección o posición de inicio:

$$\mathbf{D + (i - L) * B}$$

4.2 Almacenamiento de arrays multidimensionales.

Considerando que la memoria RAM de la computadora es lineal, un arreglo multidimensional debe ser ***linealizado*** para su almacenamiento.

TECNICAS: Dependiendo de los lenguajes de programación de alto nivel, los arreglos pueden ser almacenados en memoria de dos formas:

- OFM: Orden de fila mayor, y
- OCM: Orden de columna mayor.

4.2.1. Orden de fila mayor.- El medio predeterminado de almacenamiento en la mayoría de los compiladores es el denominado: orden de fila mayor.
Asi, sea el array A[2, 3], el orden de los elementos en memoria RAM, es:

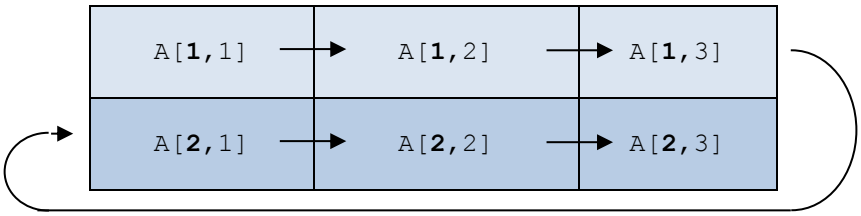


Fig. OFM: Orden de fila mayor

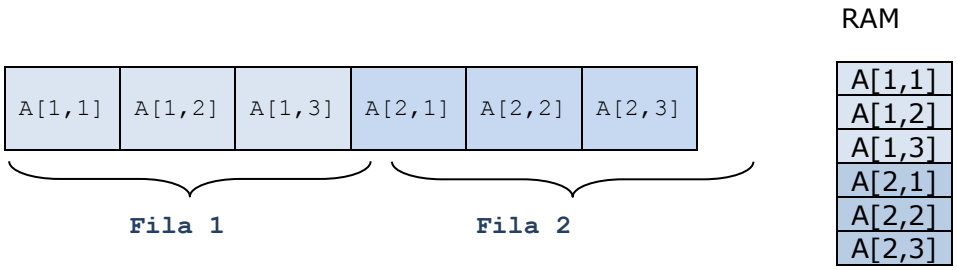


Fig. Almacenamiento secuencial contiguo por filas.

Técnica de recorrido secuencial: OFM.

1. **Algoritmo OFM.** Almacenamiento Array por filas (**orden de fila mayor**).
   Ejemplo. Cálculo del valor medio de los elementos de un array de 40 por 200 items

```
tipo
     array[1:40, 1:200] de entero: arrTabla
var
   arrTabla: tabla
   entero:  suma, i, j
   real:    media
inicio
      suma ← 0
      //recorrido secuencial: OFM
      desde( i ← 1 hasta 40 )hacer
            desde( j ← 1 hasta 200 )hacer
                  suma ← suma + tabla[i, j]
            fin_desde
      fin_desde
      media ← suma / (200 * 40)
fin
```

Posición de Elementos. De modo general, el compilador del lenguaje de alto nivel tiene la capacidad de calcular con una referencia a un índice [i, j] la posición del elemento correspondiente. 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 a la posición del primer elemento es:

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

4.2.2. Orden de columna mayor.- Excepcionalmente, algunos compiladores emplean el almacenamiento en orden de columna mayor.

Asi, tomando el array A[2, 3], el orden de los elementos en memoria RAM, es:

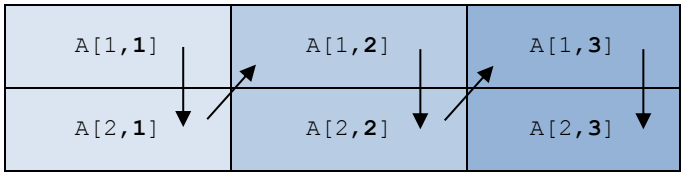


Fig. OCM: Orden de columna mayor

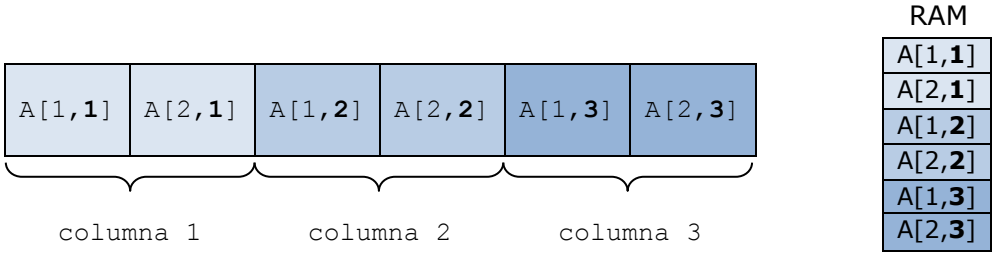


Fig. Almacenamiento secuencial contiguo por columnas.

Técnica de recorrido secuencial: OCM.

2. **Algoritmo OCM.** Almacenamiento Array por columnas (**orden de columna mayor**).
   Ejemplo. Cálculo del valor medio de los elementos de un array de 40 por 200 ítems

```
tipo
     array[1:40, 1:200] de entero: arrTabla
vart
   arrTabla:      abla
   entero:  suma, i, j
   real:    media
inicio
      total ← 0
      //recorrido secuencial: OCM
      desde( j ← 1 hasta 200 )hacer
            desde( i ←1 hasta 40 )hacer
                  suma ←suma + tabla[i, j]
            fin_desde
      fin_desde
      media ←suma / (200 * 40)
fin
```