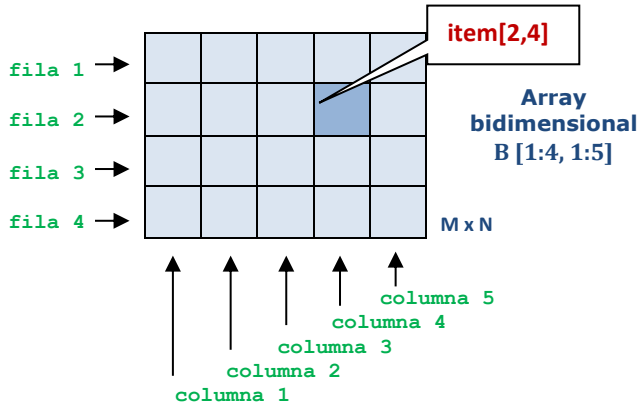
Estructuras de datos:

Arrays Multidimensionales: Tablas y Cubos

Las Matrices

1. 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. 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

3. 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

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

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, está dispuesto fundamentalmente en **secuencia contigua**.

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)

5. Operaciones con Arrays: Recorrido secuencial OFM /OCM.

Ejemplo 1.
Problema. Escribir un algoritmo que permita acumular (sumar) los saldos positivos y saldos negativos de una Tabla de valores reales.

Suponga una oficina contable que almacena saldos –positivos y negativos- de 40 clientes durante 12 meses por cliente.

meses / cliente	1	2	3	…	12
1	**10.50**	**-5.66**	**-9.99**		**-9.55**
2	**-5.55**	**10.22**	**10.05**		**10.00**
…					
40	**8.99**	**-7.44**	**-9.45**		**9.88**

Rango de saldos: **+**10.00 a **-**10.00

Análisis: Sea la tabla T de dimensiones M y N, que será leída desde un procedimiento, que toma como argumento a T[M,N] y la recorre secuencialmente para almacenar en ella una fuente de datos externa.

La fuente de datos debe ser simulada mediante generación de aleatorios según el rango de saldos: **+**10.00 a **-**10.00. Luego de asignar la fuente de datos externa, en el array T[M,N], procese este arreglo empleando almacenamiento en orden de fila mayor, verificando y sumando los valores positivo y negativos respectivamente.

Especificaciones de E/S

Entradas: lista de 40 clientes con 12 saldos cada uno
Salidas : Suma de saldos positivos anual, suma de saldos negativos anual.

Diseño de algoritmo: Pseudocodigo.

```
Algoritmo Acumulado de Saldos positivos y negativos.
const M = 40, N = 12    //dimensión de la tabla: 40 filas x 12 columnas
const U = 10.00, L = -10.00   //rango de valores aleatorios

tipo
      array[1..M, 1..N] de real: arrTabla
var
      arrTabla:  aT     //tabla de saldos positivos y negativos
  real:  sumaPos, sumaNeg     //acumuladores de saldos positivos y negativos
  entero: i, j    //contadores
inicio
      sumaPos ← 0
      sumaNeg ← 0

      llamar_a leerArray(aT, M, N)  //leer fuente de datos y almacenar en aT[M,N]

      //recorrido secuencial: OFM y acumulacion de saldos Positivos /Negativos
      desde( i ← 1 hasta M )hacer
            desde( j ← 1 hasta N )hacer
                  si( aT[i, j] >= 0 )entonces
                        sumaPos ← sumaPos + aT[i, j]
                  si_no
                        sumaNeg ← sumaNeg + aT[i, j]
                  fin_si
            fin_desde
      fin_desde
      escribir("Total saldos positivos: ", sumaPos, " negativos: ", sumaNeg)
fin


// array bi-dimensional como argumento.
procedimiento leerArray(E/S arrTabla: arrT; E entero: fila, col)
var
    entero: i, j
inicio
      //recorrido secuencial OFM
      desde( i ←1 hasta fila )hacer
            desde( j ←1 hasta col )hacer
                  arrT[i, j]← aleatorio() * (U - L) + L
            fin_desde
      fin_desde
fin_procedimiento
```

Ejemplo 2.
Problema. Diseñe un algoritmo que permita sumar los elementos una Tabla de valores que contiene un número constante de artículos producidos.

Análisis:
En una fabrica XYZ, ingresan al almacén 20 lotes de 400 productos c/u al día. Se necesita calcular el total de productos fabricados durante el mes (30 días).

constante: Lote — 400 artículos por c/lote.
Array Producion de dimension [M,N]: — 20 filas por 30 columnas.

Procesos:
Leer producción [M,N] en **orden de fila mayor** para almacenar la constante lote = 400
Recorrer, producción [M,N] en **orden de columna mayor** para calcular el total de artículos.

Especificaciones de E/S
Entradas: lista de 20 lotes x 30 días, cada lote contiene 400 artículos de fábrica.
Salidas : Suma total de artículos producidos.

Diseño de algoritmo: Pseudocodigo.

```
Algoritmo Acumulado de productos por mes.
const M = 20, N = 30    //dimensiones del array
tipo
      array[1..M, 1..N] de entero: arrProduccion
var
      arrProduccion:  arrPro  // tabla de registro de artículos por mes

inicio
      llamar_a leerArray(arrPro, M, N)//almacenar const 400 en arrP[M,N]
      escribir("Total Producción (mes): ", totalProductos(arrPro, M, N))
fin



//Definir prototipo: array bi-dimensional como argumento.
procedimiento leerArray(E/S produccion: arrP; E entero; fila, col)
var
      entero: i, j
      const LOTE = 400        //productos por lote
inicio
      // recorrido secuencial: OFM, y asignación de contante LOTE
      desde( i← 1 hasta fila )hacer
            desde( j ←1 hasta col )hacer
                  arrP[i, j] ← LOTE
            fin_desde
      fin_desde
fin_procedimiento
```

```
//Definir prototipo: array bi-dimensional como argumento.
real funcion totalProductos(E/S produccion: arrP; E entero: fila, col)
var
   real suma      //acumulador de artículos
   entero: i, j
inicio
      suma ← 0
      // recorrido secuencial: OCM y calculo del total de productos
      desde( j ← 1 hasta col )hacer
            desde( i ← 1 hasta fila )hacer
                  suma← suma + arrP[i, j]
            fin_desde
      fin_desde

      devolver (suma)
fin_funcion
```

Ejemplo 3.

Problema. Se necesita calcular los promedios semestrales de un aula de 20 alumnos, cuyas notas están en escala 5 a 20. Los tipos de notas correspondientes a cada alumno son: Promedio de prácticas(PP), Examen Parcial(EP), y Examen Final(EF). Se requiere:

a) Calcular el promedio semestral
b) Extraer el promedio Máximo
c) y Mínimo del aula.

Análisis.

El promedio para cada alumno es: media = (PP + EP + EF) / 3
A partir de la lista de promedios, extraer el máximo y mínimo valor, mediante recorrido secuencial.

Especificaciones de E/S

Entradas: array de notas[1:10, 1:3] de tipo entero para 10 alumnos con 3 tipos de notas c/u
Salida: vector de promedios[1:10] de tipo real, máximo y minimo

Diseño de algoritmo: Pseudocodigo.

```
Algoritmo. Evaluacion Semestral
const FILAS = 10, COLUMNAS = 3
tipo
      array[1:FILAS, 1:COLUMNAS] de entero: arrNotas
      array[1:FILAS] de real: arrMedia
var
      arrNotas: notas
      arrMedia: media
      real: máximo, minimo
      entero: i
inicio
      //Simulacion: lectura de datos masivos desde dispositivos externos
      llamar_a leerArray(notas, M, N)
      //procesar datos
      llamar_a calculaPromedios(notas, media, FILAS, COLUMNAS)

      //salida en pantalla de promedios
      desde( i ← 1 hasta FILAS )hacer
            escribir("Alumno: ", i, "Promedio: ", promedios[i])
      fin_desde
```

```
      maximo ← calculaMaximo( media, FILAS)
      minimo ← calculaMaximo( media, FILAS)

      escribir("Promedio maximo: ", maximo, "Minimo: ", minimo)
fin



//prototipo sub-rutina: leerArray
procedimiento leerArray(S arrNotas: arrN, fila, col)
var
      entero: f, c      //contadores de fila y columna
inicio
      desde( f ← 1 hasta fila )hacer
            desde( c ← 1 hasta col )hacer
                  arrN[f, c] ← aleatorio() * (20-5)+5
            fin_desde
      fin_desde
fin_procedimiento



//prototipo sub-rutina: calcular promedio
procedimiento calculaPromedios(E/S arrNotas: arrN; S arrMedia: arrM; E entero: fila, col)
var
      entero: f, c
      real: suma
inicio
      desde( f ← 1 hasta fila )hacer
            suma ← 0
            desde( c ← 1 hasta col )hacer
                  suma ← suma + arrN[f, c]
            fin_desde
            arrMedia[f] ← suma / (col)
      fin_desde
fin_procedimiento



//prototipo de sub-rutina: calcularMaximo
real funcion calculaMaximo(E/S arrMedia: arrM; E entero: rango)
var
      entero: i, max
inicio
            max ← arrM[1]  //asignar primer valor del array
            desde( i ← 1 hasta rango)hacer
                  si( arrM[i] > max )entonces
                        max ← arrM[i]
                  fin_si
            fin_desde
            devolver (max)
fin_funcion
```

```
//prototipo de sub-rutina: calcularMaximo
real funcion calculaMinimo(E/S arrMedia: arrM; E entero: rango)
var
      entero: i, min
inicio
            min ← arrM[rango] //asignar ultimo valor del array
            desde( i ← 1 hasta rango)hacer
                  si( arrM[i] < min )entonces
                        min ← arrM[i]
                  fin_si
            fin_desde
            devolver (min)
fin_funcion
```