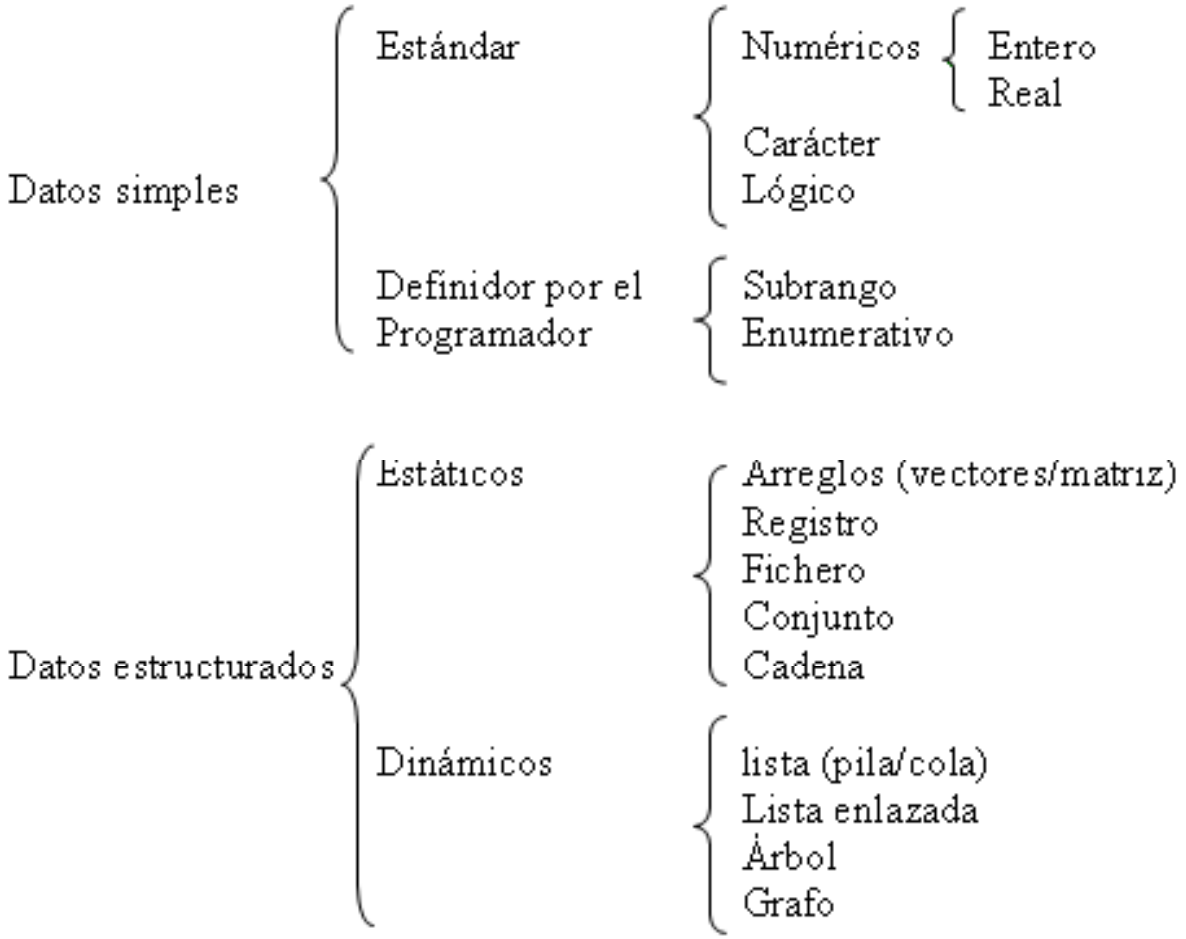
Unidad 5. Arreglos:
Vectores y Matrices

Prof. Eliana Guzmán U.
Semestre A-2015

Estructuras de datos

- Los arreglos son un tipo de estructura de datos.
- Una estructura de datos es una colección de datos que se caracteriza por su organización y las operaciones que se definen en ella.
- Las estructuras de datos son muy importantes en los sistemas de computadora.
- Los tipos de datos más frecuentes utilizados en los diferentes lenguajes de programación son:

Tipos de datos



Estructuras de datos

Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse durante la ejecución del programa. Estas estructuras están implementadas en casi todos los lenguajes de programación: arreglos (vector/matriz), registros, ficheros o archivos, conjuntos (Pascal).

Estructuras de datos

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 dato específico denominado puntero, es posible construir estructuras de datos dinámicas, que son soportadas por la mayoría de los lenguajes. Las estructuras de datos dinámicas por excelencia son: las listas (enlazadas, pilas y colas), árboles (binarios, árbol-b) y grafos.

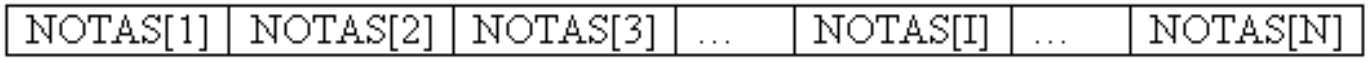
Arreglos

Un arreglo (matriz o vector) es un conjunto finito y ordenado de elementos homogéneos. La propiedad *ordenados* significa que el elemento primero, segundo, tercero,…, n-ésimo de un arreglo puede ser identificado.

Los elementos de un arreglo son homogéneos, es decir, todos son del mismo tipo de dato (cadena, carácter, lógico, entero o real).

Arreglos unidimensionales: vectores

- El tipo más simple de arreglo es el unidimensional o vector. Por ejemplo, un vector de una dimensión denominado NOTAS[i], que consta de n elementos se puede representar así:



- El subíndice o índice de un elemento [1,2,3,…,i,…,n] designa la posición que ocupa cada elemento del vector.
- Solo el vector global tiene nombre (NOTAS). Los elementos del vector se referencian por su subíndice o índice, es decir, su posición relativa en el vector.

Arreglos unidimensionales: vectores

Notación algorítmica para declarar vectores:

tipo
<nombre del tipo arreglo> = **arreglo** [dimensiones] **de** <tipo de dato>
var
identificador de la variable: <nombre del tipo arreglo>

Ejemplos:

```
tipo
 nombres= arreglo [1..10] de caracter
var
  N,M: nombres
```

se están declarando dos vectores N y M de 10 elementos cada uno de tipo carácter.

```
tipo
    número=arreglo [1..100] de entero
var
   NUM: número
```

se está declarando un vector NUM de 100 elementos de tipo entero.

Arreglos unidimensionales: Operaciones con Vectores

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

- **Recorrido (acceso secuencial).**
- Lectura/escritura.
- Asignación.
- Actualización (añadir, borrar, insertar).
- Ordenación.
- Búsqueda.

1. Recorrido (acceso secuencial)

- Se puede acceder a cada elemento de un vector para introducir datos en él (leer) ó bien para visualizar su contenido (escribir).
- A la operación de efectuar una acción general sobre todos los elementos de un vector, se le denomina recorrido secuencial del vector.

1. Recorrido (acceso secuencial)

- 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 acceso sucesivo a cada elemento del vector.

1. Recorrido (acceso secuencial)

Normalmente se utiliza la estructura de repetición desde, ya que se conoce de antemano la cantidad de veces que se desea repetir el bucle. Por ejemplo para un vector de 20 elementos:

desde i ← 1 hasta **20** hacer

  escribir(‘Introduzca el elemento ‘ ,i, ‘del vector F: ’)

  leer(F[i])

fin_desde

1. Recorrido (acceso secuencial)

También se pueden utilizar las estructuras de repetición mientras y repetir:

```
i ← 1
mientras (i <= 20) hacer
   escribir('Introduzca el elemento ' ,i, 'del
                  vector F: ')
   leer(F[i])
   i ← i + 1
fin_mientras
```

1. Recorrido (acceso secuencial)

```
i ← 1
repetir
  escribir(‘Introduzca el elemento ‘ ,i, ‘del
   vector F: ’)
  leer(F[i])
  i ← i + 1
hasta_que (i > 20)
```

2. **Lectura/escritura**

La lectura/escritura de datos en arreglos normalmente se realiza con estructuras repetitivas (usando un recorrido secuencial).

Las instrucciones simples de lectura/escritura se representarán como:

- **leer**(A[5]) lectura del elemento 5 del vector A
- **escribir**(A[8]) escribir el elemento 8 del vector A

2. **Lectura/escritura**

Generalmente se desea leer o escribir el vector completo, para lo cual se debe hacer un recorrido del vector:

```
desde i←1 hasta n hacer
    escribir(‘Introduzca el elemento ‘ ,i, ‘del
                vector F: ’)
    leer(F[i])
fin_desde
```

2. **Lectura/escritura**

Para escribir el vector F:

```
desde i←1 hasta n hacer
    escribir(F[i])
fin_desde
```

2. **Lectura/escritura**

Para facilitar futuras operaciones con el vector, se recomienda inicializar el vector antes de operar con él. Puede usarse cualquier valor que respete el tipo de dato del vector y que no sea una entrada válida de la variable involucrada:

```
desde i ← 1 hasta n hacer
  nombre[i] ← '*'
fin_desde
```

3. **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 29 del vector A
- Suma ← A[1] + A[3]
- A[3] ← A[3] + 10.8
- A[1] ← A[4] + A[5]

3. Asignación

- Si se desea asignar valores a todos los elementos de un vector, se debe recurrir a estructuras repetitivas e incluso selectivas.
- Ejemplo: si se desea dar el mismo valor a todos los elementos del vector A de tipo entero:

```
desde i ← 1 hasta 5 hacer
  A[i] ← 8
fin_desde
```

Ejemplos de vectores

1. Escribir un algoritmo que determine el mayor valor de una lista de 50 números reales e indique la posición que ocupa.

Ejemplos de vectores

2. Escribir un algoritmo que permita calcular la desviación estándar de una lista de N números. El valor de N no puede ser mayor a 15.

$$desviación = \sqrt{\frac{\sum_{i=1}^{n} (x_i - media)^2}{n-1}}$$

Ejemplos de vectores

3. Escribir un algoritmo que determine:
   a) el promedio de N números enteros e indique cuáles elementos son mayores a dicho promedio.
   b) la suma de los números pares e impares.
4. Escriba un algoritmo que determine si dos vectores de treinta caracteres son iguales.

4. Actualización

La operación de actualización de un vector consta a su vez de tres operaciones más elementales:

- Añadir elementos.
- Insertar elementos.
- Borrar elementos.

4. Actualización

Añadir elementos: es la operación de agregar un nuevo elemento al final del vector. La única condición necesaria para esta operación consistirá en la comprobación de que existe espacio suficiente para el nuevo elemento, dicho de otra manera, que el vector no contenga todos los elementos con que fue definido.

4. Actualización

Ejemplo: se tiene un vector de edades definido para 7 elementos, pero ya tiene almacenado 5 elementos EDADES[1], EDADES[2], EDADES[3], EDADES[4] y EDADES[5]. Se podrán añadir dos elementos más al final del vector con una simple operación de asignación:

- EDADES[6] $\leftarrow$ 23
- EDADES[7] $\leftarrow$ 20

(Si conoce los espacio del vector que están libres.)

4. Actualización

Si no se sabe si el vector tiene espacios disponibles, primero debe determinarse esto antes de intentar añadir elementos al vector:

```
desde i ← 1 hasta n hacer
  si (edades[i]=-1) entonces
    escribir('Introduzca una edad:')
    leer(edades[i])
  si_no
    cont ← cont + 1
  fin_si
fin_desde
 si (cont=n) entonces
    escribir('El vector no tiene espacio para añadir más elementos')
  fin_si
```

4. Actualización

Insertar elementos: consiste en introducir un elemento en el interior de un vector ordenado. En este caso se necesita un desplazamiento previo hacia abajo, para colocar el nuevo elemento en su posición relativa.

Ejemplo: se tiene un vector de 8 elementos que contiene nombres ordenados alfabéticamente y se desea insertar dos nuevos nombres: Fernando y Luis.

4. Actualización

Como Fernando está entre Carlos y Gerardo se deben desplazar hacia abajo los elementos 3, 4 y 5 que pasarán a ocupar las posiciones relativas 4, 5 y 6.
Posteriormente debe realizarse la misma operación con el nombre Luis que ocupará la posición 6.

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

```
algoritmo insertar_elemento
const
  n=500
tipo
 vector=arreglo [1 .. n] de cadena[50]
var
  NOMBRES: vector
  nuevo: cadena[50]
  Pos, ocupada, cont: entero
inicio
desde i ← 1 hasta n hacer
    si (NOMBRES[i]<>'vacio') entonces
      ocupada ← ocupada + 1
    fin_si
  fin_desde
  si (ocupada=n) entonces
    escribir('No se pueden insertar elementos')
  si_no
    escribir('Introduzca el elemento:')
    leer(nuevo)
    desde i ← 1 hasta n hacer
      si (NOMBRES[i]<nuevo) entonces
        cont ← cont + 1
      fin_si
    fin_desde
    Pos ← cont + 1
    i ←ocupada
     mientras (i >= Pos) hacer
        NOMBRES[i+1] ← NOMBRES[i]
        i ← i - 1
     fin_mientras
     NOMBRES[Pos] ← nuevo
     ocupada ← ocupada + 1
    fin_si
fin
```

4. Actualización

Borrar elementos: la operación de borrar el último elemento de un vector no representa 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.

4. Actualización

1	Ana
2	Carlos
3	Gerardo
4	Lorena
5	Marcos
6	
7	
8	

1	Ana
2	Carlos
3	Lorena
4	Marcos
5	
6	
7	
8	

Si desea borrar elemento 3 (Gerardo), debe desplazar hacia arriba los elementos de las posiciones 4 (Lorena) y 5 (Marcos).

4. Actualización

Ejemplo: en el vector del ejemplo anterior NOMBRES, borrar el elemento que el usuario desee.

```
algoritmo borrar_elemento
const
  N=500
tipo
 vector = arreglo [1 .. N] de cadena[50]
var
  NOMBRES: vector
  j,ocupada: entero
  nom: cadena[50]
Inicio
  escribir('Introduzca el nombre a borrar:')
  leer(nom)
```

4. Actualización

```
  desde i ← 1 hasta N hacer
     si (NOMBRES[i]=nom) entonces
        j ← i
     fin_si
  fin_desde
  desde i ← j hasta N-1 hacer
     NOMBRES[i] ← NOMBRES[i+1]
  fin_desde
  ocupada ← ocupada -1
fin
```

5. Métodos de ordenamiento

Ordenación (clasificación)

- Es la operación de organizar un conjunto de datos en algún orden o secuencia específica, tal como creciente o decreciente para datos numéricos o alfabéticamente para datos de tipo carácter o cadena de caracteres.
- Operaciones típicas de ordenación son: lista de números, archivos de clientes de banco, nombres en una agenda telefónica, entre otras.

Ordenación (clasificación)

- En síntesis, la ordenación significa poner objetos en orden ascendente o descendente.
- El propósito final de la ordenación es facilitar la manipulación de datos en un vector.

Ordenación (clasificación)

Los métodos directos son los que se realizan en el espacio ocupado por el arreglo. Los que vamos a estudiar son:

- Método de intercambio o burbuja.
- Ordenación por Selección.
- Ordenación por Inserción.

Método de intercambio o de burbuja

Se basa en el principio de comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que estén todos ordenados.

Método de intercambio o de burbuja

El elemento cuyo valor es mayor sube posición a posición hacia el final de la lista, al igual que las burbujas de aire en un depósito (si se ordena de forma ascendente).

Tras realizar un recorrido completo por todo el vector, el elemento mencionado habrá subido en la lista y ocupará la última posición.

En el segundo recorrido, el segundo elemento llegará a la penúltima posición, y así sucesivamente.

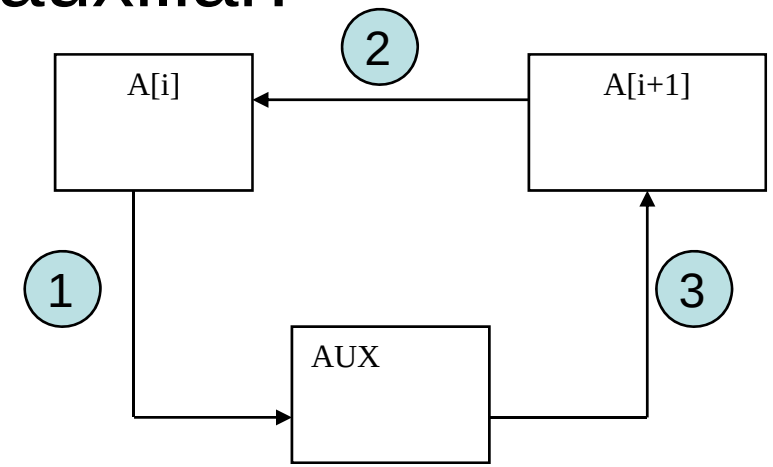
Método de intercambio o de burbuja

Los pasos a dar son:

1. Comparar A[1] y A[2], si están en orden, se mantienen como están, en caso contrario se intercambian entre si.
2. A continuación se comparan los elementos 2 y 3, de nuevo se intercambian si es necesario.
3. El proceso continúa hasta que cada elemento del vector ha sido comparado con sus elementos adyacentes y se han realizado los intercambios necesarios.

Método de intercambio o de burbuja

La acción de intercambiar entre sí los valores de dos elementos A[i], A[i+1] es una acción compuesta que contiene las siguientes acciones, utilizando una variable auxiliar:



Método de intercambio o de burbuja

En pseudocódigo:

AUX ← A[i]

A[i] ← A[i+1]

A[i+1] ← AUX

Método de intercambio o de burbuja

```
algoritmo burbuja1
const
  N=200
tipo
 vector=arreglo [1..N] de entero
Var
  X: vector
  i, j, aux: entero
inicio
  desde i ← 1 hasta N hacer
    leer(X[i])
  fin_desde
  desde i ← 1 hasta N-1 hacer
     desde j ← 1 hasta N-1 hacer
       si (X[j] > X[j+1]) entonces
          AUX ← X[j]
          X[j] ← X[j+1]
          X[j+1] ← AUX
       fin_si
      fin_desde
  fin_desde
  desde i ← 1 hasta N hacer
     escribir(X[i])
  fin_desde
fin
```

```
algoritmo burbuja2
const
  N=200
tipo
 vector =arreglo [1..N] de entero
var
  X: vector
  i, j,aux: entero
inicio
  desde i ← 1 hasta N hacer
    leer(X[i])
  fin_desde
  desde i ← 1 hasta N-1 hacer
     desde j ← 1 hasta N-i hacer
       si (X[j] > X[j+1]) entonces
          AUX ← X[j]
          X[j] ← X[j+1]
          X[j+1] ← AUX
       fin_si
      fin_desde
  fin_desde
  desde i ← 1 hasta N hacer
     escribir(X[i])
  fin_desde
fin
```

Método de ordenación por selección

Este método se basa en buscar el menor elemento del vector y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño y se coloca en la segunda posición, y así sucesivamente.

Método de ordenación por selección

Los pasos sucesivos a dar son:

1. Seleccionar el menor elemento del vector de n elementos.
2. Intercambiar dicho elemento con el primero.
3. Repetir estas operaciones con los n-1 elementos restantes, seleccionando el segundo elemento, continuar con los n-2 elementos restantes hasta que sólo quede el mayor.

Método de ordenación por selección

Pseudocódigo con estructura **desde**

```
inicio
  desde i  hasta N-1 hacer
     AUX ← X[i]
     K ← i
     desde j ← i+1 hasta N hacer
        si (X[j] < AUX) entonces
          AUX ← X[j]
          K ← j
        fin_si
      fin_desde
      X[K] ← X[i]
      X[i] ← AUX
   fin_desde
fin
```

Método de ordenación por inserción

El método se basa en comparaciones y desplazamientos sucesivos. El algoritmo de ordenación de un vector X de N elementos, se realiza con un recorrido de todo el vector y la selección e inserción del elemento correspondiente en el lugar adecuado.

Método de ordenación por inserción

Por usar la misma lógica con las que se ordenan las cartas, también se conoce con el nombre de método de la baraja.

Método de ordenación por inserción

```
algoritmo método_inserción
tipo
 vector = arreglo [1..10] de entero
var
   X: vector
   i, j, k, aux: entero
  sw: lógico
inicio
  desde i ← 2 hasta 10 hacer
      AUX ← X[i]
      K ← i – 1
      sw ← falso
      mientras no(sw) y (K>=1) hacer
        si (aux<x[k]) entonces
           X[K+1] ← X[K]
           K ← K – 1
       si_no
           sw ←verdadero
       fin_si
      fin_mientras
      X[K+1] ← AUX
  fin_desde
fin.
```

Métodos de búsqueda en vectores

Métodos de Búsqueda

- La recuperación de información, como ya se ha comentado, es una de las aplicaciones más importantes de las computadoras.
- La búsqueda se refiere a la operación de encontrar la posición de un elemento entre un conjunto de elementos dados: lista, tabla o fichero.
- Existen diferentes algoritmos de búsqueda. El algoritmo elegido depende de la forma en que se encuentren organizados los datos.

Métodos de Búsqueda

La operación de búsqueda de un elemento X en un conjunto de elementos consiste en:

1. Determinar si X pertenece al conjunto y, en ese caso, indicar su posición en él.
2. Determinar si X no pertenece al conjunto.

Métodos de Búsqueda

Los métodos más usuales de búsqueda son:

- Búsqueda secuencial o lineal.
- Búsqueda binaria.
- Búsqueda por transformación de claves (hash).

Búsqueda secuencial o lineal

El método más sencillo de buscar un elemento en un vector es explorar secuencialmente el vector (recorrer el vector), desde el primer elemento hasta el último. Si se encuentra el elemento buscado visualizar un mensaje similar a ‘Elemento encontrado en la posición x’, en caso contrario visualizar un mensaje similar a ‘Elemento no existe en el vector’.

Búsqueda secuencial o lineal

- En otras palabras, la búsqueda secuencial compara cada elemento del vector con el valor deseado, hasta que se encuentra y termina de recorrer el vector completo.
- La búsqueda secuencial no requiere ningún registro por parte del vector por consiguiente no requiere que el vector esté ordenado.

Búsqueda secuencial o lineal

Este método tiene el inconveniente del consumo excesivo de tiempo en la localización del elemento buscado.

Cuando el elemento buscado no se encuentra en el vector, se verifican o comprueban sus n elementos. Por esto no es el método más adecuado para vectores con un gran número de elementos.

Búsqueda secuencial o lineal

```
algoritmo búsqueda_secuencial
const
   N=1000
tipo
 vector=arreglo [1..N] de entero
var
  X: vector
  i, t, cont: entero
inicio
  desde i ← 1 hasta N hacer
     leer(X[i])
  fin_desde
  escribir('Introduzca el elemento a buscar: ')
  leer(t)
  desde i ← 1 hasta N hacer
      si (X[i] = t) entonces
         escribir('Elemento encontrado en la posición ',i)
      si_no
         cont ← cont + 1
      fin_si
  fin_desde
si (cont=N) entonces
   escribir('El elemento ',t,' no se
            encuentra en este vector')
  fin_si
fin.
```

```
algoritmo búsqueda_secuencial2
tipo
 vector =arreglo [1..20] de entero
var
  X: vector
  i,j,t: entero
  encontrado: lógica
inicio
  desde i ← 1 hasta N hacer
     leer(X[i])
  fin_desde
  escribir('Introduzca el elemento a buscar: ')
  leer(t)
  encontrado ← falso
  desde i ← 1 hasta N hacer
      si (X[i] = t) entonces
         encontrado ← verdadero
         j ← i
      fin_si
  fin_desde
  si encontrado entonces
     escribir('Elemento encontrado en la posición ',j)
  si_no
     escribir('Elemento no encontrado')
  fin_si
fin.
```

Búsqueda binaria

Presupone una ordenación previa de los elementos del vector.

Este método se basa en la división sucesiva del vector en dos partes, y seguir dividiendo cada mitad hasta encontrar el elemento buscado.

Búsqueda binaria

Utiliza un método de divide y vencerás para localizar el valor deseado.

Con este método se examina primero el elemento central del vector, si este es el elemento buscado, entonces la búsqueda ha terminado.

En caso contrario se determina si el elemento buscado está en la primera o segunda mitad del vector, y a continuación se repite este proceso, utilizando el elemento central de esa parte del vector.

Búsqueda binaria

Es un método eficiente siempre que el vector esté ordenado.

En la práctica esto suele suceder, pero no siempre es así. Por esta razón la búsqueda binaria exige una ordenación previa del vector.

```
algoritmo búsqueda_binaria
tipo
 vector=arreglo [1..500] de entero
var
  X: vector
  i,j,t,k,primero,ultimo,central: entero
  encontrado: lógica
Inicio
  escribir('Introduzca el valor a buscar: ')
  leer(k)
  primero ← 1
  último ← N
  central ← trunc((primero+último)/2)
  mientras (primero <= último) y (X[central] <> K) hacer
      si  (K < X[central]) entonces
        último ← central - 1
       si_no
        primero ← central + 1
      fin_si
      central ← trunc((primero+último)/2)
  fin_mientras
  si K = X[central] entonces
    escribir('Elemento encontrado en la posición ',central)
  si_no
    escribir('Elemento no encontrado')
  fin_si
fin.
```

Ejemplo 5: Un viajero conoce todos los gastos que hizo en su último viaje con la tarjeta de crédito, fueron 18 en total, los cuales se pueden clasificar en cuatro tipos: comida, hospedaje, transporte y ropa. Escriba un algoritmo, que haga uso de vectores, y le permita al viajero determinar:

- Gastos totales discriminados por tipo.
- Cantidad de gastos que realizó en comida.
- Tipo de gasto en el que más invirtió dinero en el viaje.

Ejemplo 6: Escriba un algoritmo que haciendo uso de un arreglo unidimensional de 100 elementos denominado CEDULA, permita:

- Leer los 100 elementos por teclado.
- Ordenar los números de cédula de forma ascendente.
- Buscar si el número de cédula 19144473 se encuentra en el arreglo CEDULA e indicar la posición que ocupa.
- Determinar cuántos números de cédula son mayores a 19145954.

Ejemplo 7: Se han registrado las notas definitivas de 58 estudiantes de Programación Digital en un arreglo unidimensional llamado NOTAS. Cada calificación es un número entero entre 1 y 20. Construya un algoritmo que le permita al profesor de esta asignatura obtener la siguiente información:

- La nota promedio de la clase.
- Las notas ordenadas de menor a mayor.
- Nota más alta y más baja obtenida en esta sección de Programación Digital, e indicar los nombres de los estudiantes que las obtuvieron.
- Cuántos estudiantes obtuvieron notas de 18, 19 ó 20.
- Las notas que fueron mayores al promedio.
- El porcentaje de estudiantes aprobados y el porcentaje de los reprobados.

Arreglos Multidimensionales

Los arreglos de varias dimensiones se dividen en dos grandes grupos:

- Arreglos bidimensionales: tablas o matrices.
- Arreglos multidimensionales.

Arreglos multidimensionales

Un arreglo multidimensional, se puede definir de tres, cuatro y hasta n dimensiones.

Se manejan los mismos conceptos para los subíndices que en los vectores.

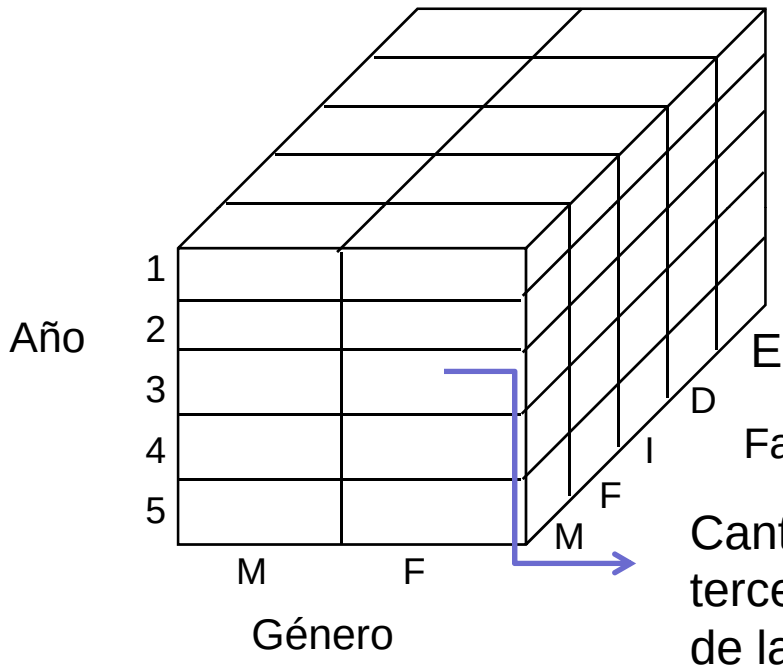
Cada elemento del arreglo se puede identificar usando la cantidad de subíndices necesarios, por ejemplo en un arreglo de n dimensiones se escribirá: $A[I_1, I_2, I_3, \ldots, I_n]$

Arreglos multidimensionales

Ejemplo: Un arreglo de tres dimensiones puede ser uno que contenga los datos relativos a la cantidad de estudiantes de una universidad de acuerdo a los siguientes criterios:

- año (primero a quinto).
- género (femenino/masculino).
- facultad (medicina, farmacia, ingeniería, derecho y educación).

Arreglos multidimensionales:



Cantidad de estudiantes que cursan el tercer año, de género femenino (F) de la facultad de medicina.

Arreglos Bidimensionales: Matrices

Es un tipo de arreglo, cuyos elementos se pueden referenciar por dos subíndices.

Existen grupos de datos que se representan mejor en forma de tabla o matriz con dos subíndices. Ejemplos típicos de tablas o matrices son:

- Distancias entre ciudades.
- Horarios.
- Informes de ventas periódicas.

Arreglos Bidimensionales: matrices

Un arreglo bidimensional se puede considerar como un vector de vectores.

Es un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo, y en el que se necesitan especificar dos subíndices para poder identificar cada elemento del arreglo.

Arreglos Bidimensionales: matrices

Un arreglo bidimensional almacena la información que relaciona **dos variables, características o factores**.

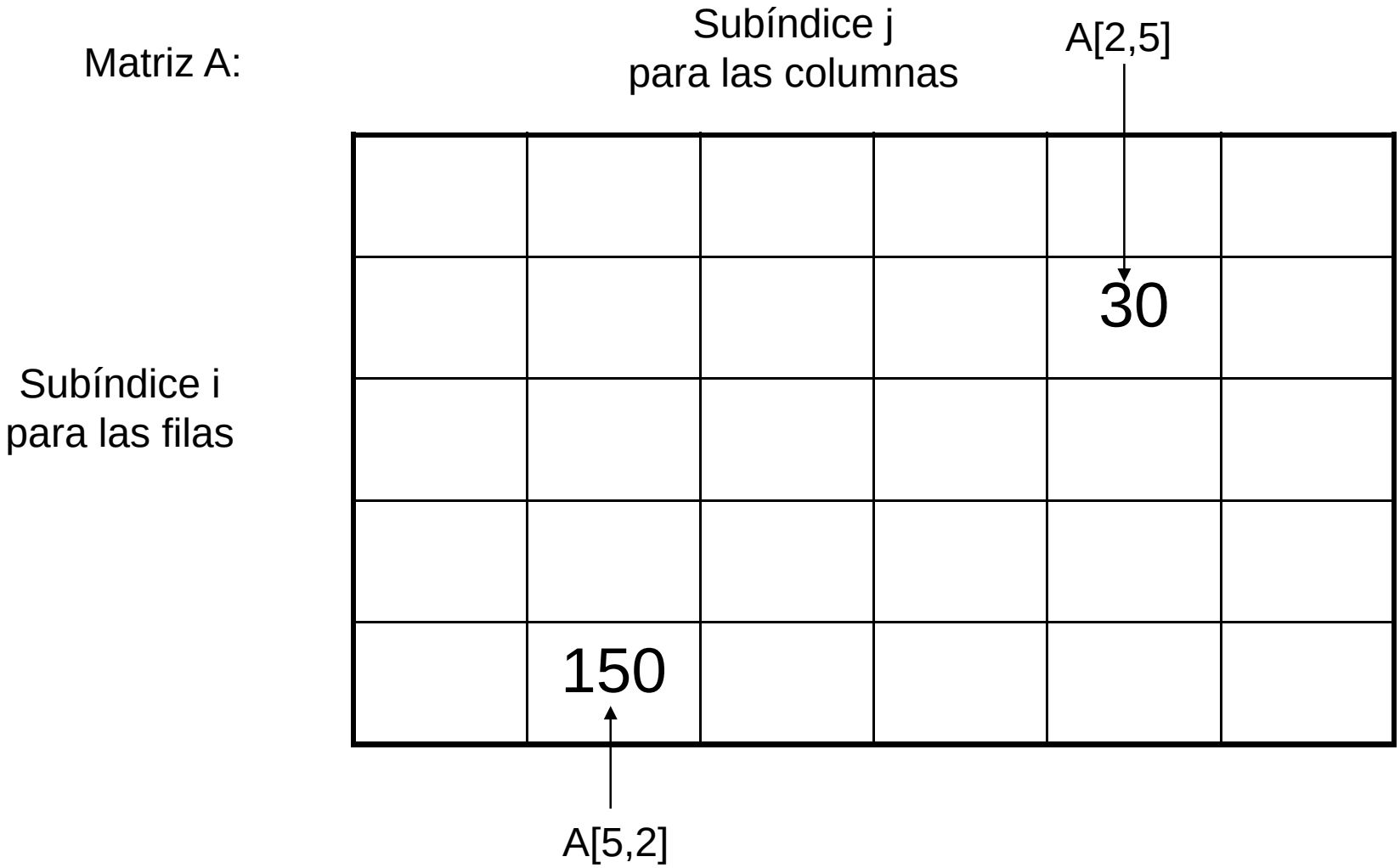
Las filas representan una de las variables y las columnas la otra variable.

Arreglos bidimensionales: matrices

Matriz A:

Fila 1	fila 1, columna 1 (1,1)					
Fila 2					30	
Fila 3						
Fila 4						
Fila 5		150				
	Columna 1	Columna 2				Columna 6

Arreglos bidimensionales: matrices



Arreglos bidimensionales: matrices

Notación algorítmica para declarar matrices:

tipo
 <nombre del tipo arreglo> =**arreglo** [1..F,1..C] **de** <tipo de dato>
var
  identificador de la variable de este tipo: <nombre del tipo arreglo>:

Ejemplo:

tipo
      notas = **arreglo** [1..5, 1..6] **de** entero
var
   A, B: notas (*Se están declarando dos matrices de números enteros
                    que tiene 5 filas y 6 columnas*)

Arreglos bidimensionales: matrices

Un arreglo bidimensional se dice que tiene F*C elementos, donde F es el número de filas y C el número de columnas.

Arreglos bidimensionales: matrices

Operaciones con matrices:

1. Asignación.
2. Lectura/escritura.
3. Recorrido secuencial:
   - Por fila
   - Por columna

Arreglos bidimensionales: matrices

1. Asignación: consiste en asignar directamente un valor a cualquier elemento de la matriz. Ejemplo: A[1,1] $\leftarrow$ 3

   Normalmente se requiere asignar valores a varios o todos los elementos de una matriz, para lo cual se usa un recorrido secuencial.

2. Lectura/escritura: normalmente se realiza usando un recorrido secuencial. Pero una instrucción simple de lectura/escritura podría ser:

   leer(A[2,3]) lectura del elemento en la fila 2 columna 3 de la matriz A.

Arreglos bidimensionales: matrices

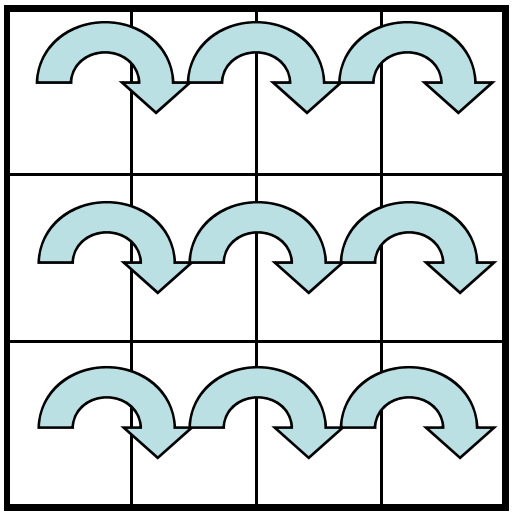
3. Recorrido secuencial: Se puede acceder a los elementos de una matriz para introducir datos (leer) en ella, o bien para visualizar su contenido (escribir), realizar comparaciones, búsquedas de elementos o cualquier otro tipo de operación.

- Esta operación se realiza usando estructuras de repetición, cuyas variables de control se utilizan como subíndices de la matriz (por ejemplo i, j).
- El incremento del contador del bucle producirá el tratamiento sucesivo de los elementos de la matriz.

Arreglos bidimensionales: matrices

El recorrido secuencial se puede hacer por filas o por columnas.

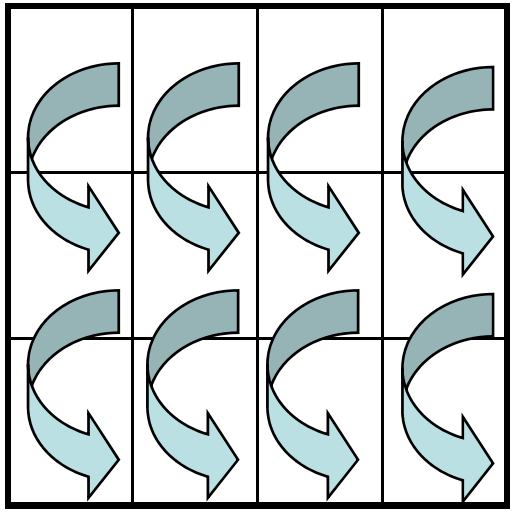
Recorrido secuencial por filas:



```
desde i ← 1 hasta 3 hacer
    desde j ← 1 hasta 4 hacer
        leer(A[i,j])
    fin_desde
fin_desde
```

Arreglos bidimensionales: matrices

Recorrido secuencial por columnas:



```
desde j ← 1 hasta 4 hacer
    desde i ← 1 hasta 3 hacer
        leer(A[i,j])
    fin_desde
fin_desde
```

Ejemplo 1: Inicializar la matriz A de 10 filas y 4 columnas con un valor constante dado por el usuario.

```
algoritmo inicialización_matriz
const
  F=10
  C=4
tipo
 matriz =arreglo [1..F, 1..C] de entero
var
  A:matriz
  i, j, k:entero
Inicio
  escribir('Introduzca el valor con el que desea inicializar la matriz: ')
  leer(k)
  desde i ← 1 hasta F hacer
     desde j ← 1 hasta C hacer
        A[i,j] ← k
     fin_desde
  fin_desde
fin.
```

Ejemplos de matrices

Ejemplo 2: Escribir un algoritmo que permita obtener la suma de los elementos positivos y la suma de los elementos negativos de una matriz T, que tiene 2 filas y 10 columnas.

Ejemplo 3: Escribir un algoritmo que obtenga la suma de los elementos de cada una de las filas y de cada una de las columnas de una matriz de 3 filas y 2 columnas.

Ejemplos de matrices

Ejemplo 4: Escriba un algoritmo que realice la suma de todos los elementos de una matriz B de 5 filas y 5 columnas.

Ejemplo 5: El jefe de recursos humanos de una tienda de 8 departamentos, desea registrar la asistencia de los trabajadores cada día de la semana en cada departamento, para obtener la siguiente información:

Ejemplos de matrices

a) La cantidad de trabajadores que laboraron cada día de la semana.
b) El departamento al que más asistieron sus trabajadores durante la semana.
c) La cantidad de trabajadores que asistieron el día sábado y el día domingo.
d) A cuál departamento asistieron la menor cantidad de trabajadores durante la semana.

Ejemplo 6: En Mérida existen 4 estaciones metereológicas, cada una de ellas registra la temperatura promedio mensual (temperatura mínima medida: 8°C y temperatura máxima medida: 32°C). Si a Ud. le proporcionan dicha información para el año 2011. Escriba un algoritmo que determine:

- La temperatura promedio en el año 2011, registrada por las 4 estaciones.
- La temperatura promedio en el año 2011, registrada por las estaciones 2 y 4.
- Los meses del año que tuvieron una temperatura promedio superior a la temperatura promedio en el año 2011, registrada por las 4 estaciones.

Debe Validar las entradas y dibujar las estructuras de datos que emplee en la solución del problema.

Ejemplo 7: Se van a registrar los votos de 50 personas para las elecciones de alcalde de la ciudad de Mérida, en la cual participan cuatro candidatos. Cada persona puede votar por un solo candidato el cual se registra con un uno, y por los candidatos que no votó con un cero. Haciendo uso de una matriz para registrar los votos, escriba un algoritmo que permita determinar:

a) La cantidad de votos que obtuvo cada uno de los cuatro candidatos.
b) La posición final ocupada por cada candidato.
c) El candidato ganador.
d) El candidato por el que menos votaron estas 50 personas.
e) Por cuál candidato votó el votante 5.

Ejemplo 8: En una zapatería se conocen las ventas mensuales de diez modelos de zapatos desde enero hasta junio del 2012, cuyo inventario inicial fue de 30 pares de cada modelo. Escriba un algoritmo que proporcione la información que necesita el dueño para realizar los pedidos del segundo semestre del año:

a) ¿Cuáles modelos de zapatos debe pedir porque ya no tiene existencia?
b) ¿Cuál modelo fue el menos vendido?
c) ¿En qué mes se vendieron más zapatos?
d) ¿Cuál es el mes en que más se vendieron zapatos de los modelos 3 y 6?
e) ¿Total de pares de zapatos vendidos en estos 6 meses, en dicha zapatería?