Unidad 5
ESTRUCTURAS ESTÁTICAS I: Tablas
Desarrollo de la unidad : 36 h

Prácticas:
Ejercicios:
Conceptos:
Tablas unidimensionales, vectores, cadenas, modificación, búsqueda, ordenación, mezclas
Tablas bidimensionales, matrices, tridimensionales
Implementación de pilas y colas

MANEJO DE TABLAS

(También llamados: array, arreglos)

Introducción

Repaso: Tipos básicos

Entero, Real, Carácter, Lógico y Enumerado

Problemas que necesitan una estructura de datos diferente

- Ordenar 2, 3, N Elementos
- Leer X valores y mostrarlos en orden inverso
- Obtener los X valores más grandes
- Mostrar el valor que más veces se repite
- Problemas de tablas Loto, Quiniela, Tablero de Ajedrez, Matrices, Polinomios,
- Almacenar una frase o una palabra.

Definición de las tablas

Conjunto finito y fijo de elementos de un mismo tipo almacenados en posiciones consecutivas.

Existen algunos lenguajes que permiten tablas que no cumplen esta definición: de tamaño variable (Problema de asignación de memoria) y de distinto tipo (Pérdida de eficiencia)

Cada tabla tiene un nombre, tamaño y un tipo de elemento
Ej.-

<Tipo de elementos> <Nombre de la tabla> [<Número de elementos>]

Entero MiTabla [10]

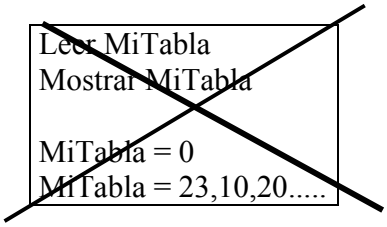
20	12	5	123	4	-5	0	32	23	10
1	2	3	4	5	6	7	8	9	10

Los valores de una tabla sólo se pueden modificar elemento a elemento, indicando la posición del elemento que quiero modificar.

Para indicar un elemento concreto de una tabla lo expresamos mediante un valor entero que debe estar entre 1 y N siendo N el tamaño máximo de la tabla.

MiTabla [3] = 100
MiTabla [Num/ 2 + 1] = 1000 * Dato

<No se permiten utilizar operaciones básicas que afecten a toda la tabla>



Sobre los elementos de una tabla se pueden hacer las mismas operaciones que sobre un elemento básico que forma la tabla.

MiTabla[2] = MiTabla[0] * 12 + Num
Mostrar MiTabla[1]
Leer MiTabla[3]

Para recorrer los distintos elementos de una tabla es necesario utilizar una variable entera que tome los valores 1 a N, esta variable se denomina **índice** de la tabla.

-- Hay que garantizar que el valor del índice no supera en ningún momento los límites de la tabla

Existen lenguajes donde se pueden definir el tipo de índices a utilizar: enteros, letras, tipos enumerados: Ej- 'A'-'Z' , VERDE a ROJO 4-8 y se detecta en ejecución si un índice está fuera de la tabla. El Lenguaje C es muy limitado en este aspecto: el índice sólo puede ser un valor entero Ej.- int Tabla [10] (0..N-1), y no comprueba en tiempo de ejecución, si se supera los límites de la tabla , lo que puede producir errores inesperados.

EJEMPLO - Inicializar todos los elementos a 0
Entorno
  Entero Mitabla[10]

1) Forma:
MiTabla[1] =0, MiTabla[2] = 0, ... MiTabla[10] = 0;

2) Forma
```
PARA i = 1 HASTA 10 HACER
  MiTabla[i] = 0
FIN-PARA
```

< El ciclo "PARA" es consustancial con el manejo de las tabla, al ser una estructura de datos de tamaño fijo>

Clasificación:

Según el número de índices o coordenadas que debemos utilizar para acceder a un elementos

- Unidimensionales o Vectores
  <Tipo de Elemento> <Nombre Tabla> [NumMax]

  Real edad[20]

18	19	21	..	20	18	18
edad[0]	edad[1]	edad[2]		edad[17]	edad[18]	edad[19]

- Bidimensionales o Matrices
  <Tipo de Elemento> <Nombre Tabla> [NumMax1, NumMax2]

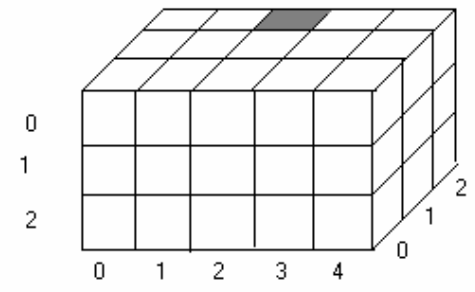
  Real notas[25,3]

	NOTAS DEL MÓDULO SIMM	NOTAS DEL MÓDULO PLE	NOTAS DEL MÓDULO ANÁLISIS
Notas del primer alumno	5	7	8
Notas del segundo alumno	8	7	6
Notas del tercer alumno	6	5	7
Notas del cuarto alumno	5	**5**	5
.....	...	..	...
Notas del 25º alumno	5	9	8

notas[4,2]

- Multidimensionales o poliedros
  <Tipo de Elemento> <Nombre Tabla> [NumMax, NumMax2,..,NumMaxN]

Entero Posición [3,5,2]



TABLAS UNIDIMENSIONALES O VECTORES

Almacenamiento de tabla en memoria

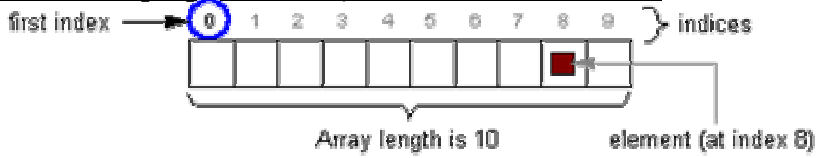
Dir(I) = DirBase + (I -1) * Tamaño Elemento

Algoritmos básicos sobre tablas unidimensionales

1. - Recorrer una tabla

```
Suma = 0
PARA I = 1 HASTA N HACER
   Suma = Suma + Tabla [ I ]
FIN-PARA
```

NOTA MUY IMPORTANTE:
Las tablas en lenguaje C, comienzan con la posición 0, en una tabla definida como *int tabla[10]*, los valores posibles son desde el *tabla[0]* hasta *tabla[9],*
el valor *tabla[10]* no existe, está fuera de la tabla.



2.- Rellenar con valores en la entrada

```
Mostrar " Introduce los valores : "
PARA I = 1  HASTA N  HACER
   Mostrar " Valor para el elemento Nº ", I
   Leer Tabla [ I ]
FIN-PARA
```

3.- Mostrar tabla

```
Mostrar " Contenido de la tabla: "
PARA I = 1  HASTA N  HACER
   Mostrar " Valor para el elemento Nº ", I, "=", Tabla [ I ]
FIN-PARA
```

4.- Invertir una tabla

```
PARA I = 1 HASTA  N/2
       aux =  Tabla [ I ]
       Tabla [ I ]  =  Tabla [ N – ( I - 1 ) ]
       Tabla [ N – ( I – 1) ] = aux
FIN-PARA
```

5.- Calcular la Moda : El valor que más veces se repite en una tabla

```
FModa = 0
PARA  I = 1 HASTA   N  HACER
   Fmodatmp = 0
   Valor = Tabla [ I ]
   PARA J = 1  HASTA   N HACER
      SI (  Tabla [ J ]  =  Valor )
          ENTONCES
             FModaTmp = FmodaTmp  +1
      FIN-SI
  FIN-PARA
  SI  (  FmodaTmp   >  FModa  )
     FModa = FmodaTmp
     Moda = Valor
  FIN-SI
FIN-PARA
```

Mostrar Moda

MÉTODOS DE BÚSQUEDA DE UN ELEMENTO EN UNA TABLA

1. Búsqueda secuencial: Número medio de elementos recorridos **(N / 2)**

```
A) I=1
Encontrado = NO
MIENTRAS ( NOT Encontrado  AND  ( I < = N ) )
    SI (  Tabla [ i ] = Valor )
        Encontrado = SI
   SINO
        I = I + 1
   FIN-SI
FIN-MIENTRAS

SI Encontrado
    Mostrar " El Elemento está en la tabla en la posición",I
SINO
   Mostrar  "El Elemento no está en la tabla "
FINSI

B)
Encontrado = NO
PARA I = 1 HASTA N HACER
   SI ( Tabla [ i ] =  Valor )
        Encontrado = SI
        Salir PARA
  FIN-SI
FIN- PARA
```

2. Búsqueda secuencial en una tabla ordenada

A) CON MIENTRAS

```
I = 1
Terminar = NO
MIENTRAS ( NOT Terminar  AND  ( I < = N ) )
    SI (  Tabla [ i ]  >= Valor )
        Terminar  = SI
   SINO
        I = I + 1
   FIN-SI
FIN-MIENTRAS

SI  ( Terminar  )
   SI  ( Tabla [ I ]  = Valor )
      Mostrar " Esta en la posición ",I
   FIN-SI
SINO
      Mostar " No está en la tabla "
FINSI
```

B) CON REPETIR

```
I = 0
Terminar  =  NO
REPETIR
  I = I + 1
  SI  (Tabla [ I ] > = Valor )
         Terminar = SI
 FIN-SI
HASTA ( Terminar  = SI ) OR ( I = N)

SI ( Terminar )
   SI ( Tabla [ I  ] = valor )
      Mostrar " Esta en la posición ",I
  FIN-SI
SINO
      Mostar " No está en la tabla "
FIN-SI
```

3.- Búsqueda <u>dicotómica o binaria</u> Máximo número de posiciones recorridas**: $\mathbf{Log_2 N + 1}$**

```
Izda = 1
Dcha = N
Encontrado = NO
MIENTRAS  ( Izda <= Dcha ) and ( NOT Encontrado )
    Medio = ( Izda + Dcha )  /  2
    SI ( Tabla [ Medio ] = Valor )
        Encontrado = SI
   SINO
         SI  ( Tabla [Medio] < Valor )
            Izda = Medio + 1
        SINO
            Dcha = Medio – 1
        FIN-SI
   FIN-SI
FIN-MIENTRAS
-----
SI ( Encontrado )
   Mostrar " El elemento se encuentra en la posición ", Medio
SINO
   Mostrar " El elemento no aparece en la tabla."
FIN-SI
```

Siempre que tengamos que realizar una búsqueda en una tabla ordenada debemos utilizar este método al ser mucho más eficiente que la búsqueda secuencial.

Ejemplo

Búsqueda secuencial sobre una tabla de 1024 Posiciones
Número medio de posiciones recorridas 1024 / 2 -> 512 posiciones

Búsqueda dicotómica sobre una tabla de 1024 Posiciones
Número máximo de posiciones recorridas Log_2 1024 + 1 = 11 posiciones

LA TABLA GESTIONADA COMO ALMACÉN :

La tabla es una estructura de datos muy utilizada en multitud de casos para guardar datos que se van generando periódicamente y que posteriormente tenemos que recuperar:

Operaciones habituales sobre un almacén de datos: Poner, Sacar, Buscar, Vaciar, Modificar, Ordenar

Tipos de implementaciones:

1. Tabla con huecos, donde existe un valor que indica que la posición está libre.

x		x	x		x	x		x	

2. Con un contador que indica cual es la última posición libre, o el número de elementos almacenados. Podemos añadir siempre al final o insertar para mantener siempre la tabla ordenada

x	x	x	x	x	x				

   NumElementos = 7

3. La Tabla gestionada como una Cola.

   El primer elemento que se guarda es el primero que se debe eliminar

4. La Tabla gestionada como una Pila.

   El último elemento que se guarda es el primero que se debe eliminar.

Ejemplos Implementación

Codificar las siguiente funciones y procedimientos, suponiendo que trabajamos con la variable global entero TAlmacen [N]

A) Suponer que trabajamos con una tabla de huecos, donde la posición libre se indica con el valor 0
B) Suponer que trabajamos con una tabla ordenada, donde guardamos en una variable NumElementos, con el número de datos almacenados en la tabla

PROCEDIMIENTO Inicializar ()	Modifica la tabla para indicar que todas la posiciones están libre
PROCEDIMIENTO Mostrar ()	Muestra todos los valores almacenados en las posiciones ocupadas de la tabla,
FUNCIÓN BuscarDato (entero Num) devuelve Entero	Devuelve la posición donde aparece Num o un valor 0
FUNCIÓN PonerDato (entero Num) devuelve LÓGICO	Devuelve Cierto si ha podido guardar el elemento
FUNCIÓN BorrarDato (entero Num) devuelve LÓGICO	Devuelve Cierto si ha podido eliminar el elemento
FUNCIÓN ContarDatos () devuelve Entero	Devuelve el número de elementos almacenados en la tabla

Para el primer caso codificar el procedimiento **Empaquetar**, que elimine todos los huecos intermedios situándolos al final de la tabla.

A) **IMPLEMENTACIÓN CON UNA TABLA DE HUECOS**: Las posiciones libre se indican con el valor 0

Entorno del programa principal

```
Constantes
  Entero N =  100
Variables
  Entero Almacen [ N ]
```

```
PROCEDIMIENTO  Inicializar ()
Entorno
  Entero i
Inicio
/* Todas las posiciones libres */
Para i = 1 hasta N
     Almacen [ i ] = 0
Fin-para
Fin-procedimiento
```

```
PROCEDIMIENTO  Mostrar ()
Entorno
  Entero i
Inicio
/* Muestro sólo las posiciones ocupadas */
Para i = 1 hasta N
  Si (  Almacen [ i] <> 0 )
    entonces
       Mostrar Almacen [i]
  fin-si
Fin-para
Fin-procedimiento
```

```
FUNCIÓN BuscarDato ( entero Num) devuelve Entero
Entorno
  Entero i, posición;
Inicio
posición = 0
i = 1
Mientras ( i <= N ) and ( posición = 0 )
  Si ( Almacen [ i ] =  Num )
    entonces
       posición = i
  fin-si
  i = i + 1
fin-mientras
RETURN  posición
Fin-Función
```

```
FUNCIÓN PonerDato  ( entero Num) devuelve LÓGICO
Entorno
  Entero i
   Lógico colocado
Inicio
  colocado = falso
  i = 1
  Mientras ( colocado = falso) and  ( i <= N )
     Si  ( Almacen [ i ] = 0 )
        Entonces
          Almacen [ i ] = Num
          colocado = cierto
     fin-si
     i = i + 1
   fin-mientras
   RETURN colocado
fin-función
```

```
FUNCIÓN BorrarDato  ( entero Num) devuelve LÓGICO
Entorno
  Entero i
   Lógico borrado
Inicio
  borrado = falso
  i = 1
  Mientras ( borrado = falso) and  ( i <= N )
     Si  ( Almacen [ i ] = Num )
        Entonces
          Almacen [ i ] = 0
          borrado = cierto
     fin-si
     i = i + 1
   fin-mientras
   RETURN borrado
fin-función
```

```
FUNCIÓN ContarDatos () devuelve Entero
Entorno
  Entero i
  Entero contador
Inicio
  Contador = 0
  Para i = 1 hasta N
    Si (  Almacen [ i] <> 0 )
       contador = contador + 1
    fin-si
   Fin-para
   RETURN contador
Fin-función
```

B) **IMPLEMENTACIÓN CON UNA TABLA ORDENADA**, con un contador de posiciones ocupadas

Entorno del programa principal

```
Constantes
  Entero N =  100
Variables
  Entero Almacén [ N ]
  Entero NumElementos
```

```
PROCEDIMIENTO  Inicializar ()
Inicio
  NumElementos = 0
Fin-procedimiento
```

```
PROCEDIMIENTO  Mostrar ()
Entorno
  Entero i
Inicio
/* Muestro sólo las posiciones ocupadas */
Para i = 1 hasta NumElementos
   Mostrar Almacen [i]
  Fin-para
Fin-procedimiento
```

```
/* Al estar ordenada realizo una búsqueda dicotómica */
/* Devuelve 0 si no encuentra el elemento (Ojo en C) */
FUNCIÓN BuscarDato ( entero Num) devuelve Entero
Entorno
  Entero Izda,Dcha, Centro
  Entero posición
Inicio
  Posición = 0
  Izda = 1
  Dcha = NumElementos
  MIENTRAS  ( Izda <= Dcha ) and ( Posición = 0 )
    Medio = ( Izda + Dcha )  /  2
    SI ( Almacen [ Medio ] = Valor )
         Posición = Medio
    SINO
         SI  ( Almacen [Medio] < Valor )
            Izda = Medio + 1
        SINO
            Dcha =  Dcha – 1
        FIN-SI
    FIN-SI
  FIN-MIENTRAS
  RETURN Posición
Fin-función
```

```
FUNCIÓN PonerDato  ( entero Num) devuelve LÓGICO
Entorno
  Entero i, j
   Lógico colocado
Inicio
  colocado = falso
  /* Si hay sitio en el Almacén */
  Si ( NumElementos < N )
     Entonces
       /* Busco el lugar para dejar la tabla ordenada */
      i = 1
      Mientras ( i <= NumElementos ) and ( Almacen [ i ] < Num )
         i = i +1
      fin-mientras
      /* Desplazo los elementos a la derecha */
      Para j = Nelementos hasta i decremento
         Almacen [ j + 1 ] = Almacen [ j ]
      fin-para
     /* Coloco el elemento e incremento el contador */
     Almacen [ i ] = Num
      colocado = cierto
     NumElementos = NumElementos  + 1
  fin-si
  RETURN colocado
fin-función
```

```
FUNCIÓN Borrar dato  ( entero Num) devuelve LÓGICO
Entorno
   Entero i, j
   Entero posición
   Lógico borrado
Inicio
  borrado = falso
   /* Obtengo la posición con la función BuscarDato */
   posición = BuscarDato ( Num )
   Si ( posición <> 0 )
     Entonces
         /* Desplazo los elementos a la Izquierda */
         Para  i = posición+1 hasta NumElementos
            Almacen [ i – 1 ] = Almacen [ i ]
         Fin-para
         NumElementos = NumElementos – 1
        borrado = cierto
    fin-si
    RETURN borrado
Fin-función
```

```
FUNCIÓN ContarDatos () devuelve Entero
Inicio
   RETURN NumElementos
fin-función
```

Ejemplos de programa principal de Manejo del Almacén:

```
PROGRAMA  GestionAlmacen
/*  Gestión básica un almacén de número enteros */
Entorno
  Constantes N = 100
Variables
  Entero Almacen [ N ]
  Entero NumElementos;
  Entero Valor, Numopcion

/* Definición de Funciones */
PROCEDIMIENTO  Inicializar ()
....
PROCEDIMIENTO  Mostrar ()
....

/* Programa Principal */
INICIO
  Inicializar()
   REPETIR
     Mostrar “  --- MENÚ DE  CONTROL DEL ALMACÉN -----“
     Mostrar “          1.Mostrar contenido del Almacén “
     Mostrar “          2. Poner un Número        “
     Mostrar “          3. Buscar un Número “
     Mostrar “          4. Borrar un  Número  “
     Mostrar “          5. Contar Datos “
     Mostrar “          0. Terminar. “
     Mostrar ”  Introduzca una opción (0-5): “
     Leer Numopción
     SEGÚN (opción)
        1 :  Mostrar ()
        2 :  Mostrar “ Valor a almacenar: “
             Leer Valor
            SI (  PonerDato ( Valor )  = Verdadero )
                 Mostrar “ Valor almacenado.”
              SINO
                 Mostrar “ El valor no cabe, el almacén está lleno “
            FIN-SI
        3:
        ..............
        5:
     FIN-SEGÚN
  HASTA (Numopción = 0 )
FIN
```

MÉTODOS DE ORDENACIÓN

Una de la operaciones más comunes que realiza un sistema informático es la ordenación de datos, (de ahí proviene la palabra de origen francés ordenador), la ordenación de tablas es fundamental en la mayoría de las ocasiones para realizar un manejo eficiente de las mismas. Existen varios algoritmos de ordenación siendo los métodos más conocidos los siguientes:

A) **Intercambio directo (Burbuja)**, compara cada dos elementos y si no están ordenados los intercambia, repite tantas veces como elementos.

```
PARA  I = 1 HASTA  N - 1
      PARA J =  1  HASTA  N -  I
        SI TABLA [ J ]  > TABLA [ J + 1 ]
            AUX = TABLA [J]
            TABLA [J] = TABLA[J +1]
            TABLA [J +1] = AUX
        FIN-SI
     FIN-PARA
FIN-PARA
```

7	5	1	2	4
5	1	2	4	7
1	2	4	5	7
1	2	4	5	7
1	2	4	5	7

Mejoras:
- No seguir si no ha habido ningún intercambio
- Hacer una pasada en un sentido y otro en otro

Versión mejorada que termina cuando no se produce ningún intercambio:

```
Cambio = SI
I = 1
MIENTRAS ( Cambio = SI )
       Cambio = NO
       PARA J =  1  HASTA  N -  I
          SI TABLA [ J ]  > TABLA [ J + 1 ]
              AUX = TABLA [J]
              TABLA [J] = TABLA[J +1]
              TABLA [J +1] = AUX
              Cambio = SI
          FIN-SI
       FIN-PARA
       I = I + 1
FIN-MIENTRAS
```

A pesar de está mejora el método de intercambio directo es uno de los métodos de ordenación más lentos, debido a que desplaza los elementos desordenados una posición cada vez.

B) **Selección directa**, Buscar el más pequeño de la parte desordenada y lo sitúa al final de la parte ordenada, Intercambiando su posición

```
PARA  I = 1   HASTA N – 1
     PosM= I
     Mínimo = Tabla [ I ]
     PARA  J  = I +1 HASTA  N
        SI  ( Mínimo >  Tabla [ j ] )
           Mínimo = Tabla [ j ]
           PosM= J
        FIN-SI
    FIN-PARA
    /* Sólo si no esta ordenado lo intercambio */
    SI  (  PosM ≠ I  )
          Tabla [ PosM ] = Tabla [ I ]
          Tabla [ I ] = Mínimo
   FINSI
FIN-PARA
```

7	5	1	2	4
1	5	7	2	4
1	**2**	7	5	4
1	**2**	**4**	5	7
1	**2**	**4**	**5**	7

C) **Inserción directa,** Coge el primer elemento de la parte desordenada y lo sitúa en la posición correcta de la parte ordenada. (Método de la baraja)

```
PARA I = 2 HASTA N
     aux =  Tabla [ I ]
     J =   I - 1
    MIENTRAS (  Tabla [ J ]  > aux )  and  ( J  > 1 )
          Tabla [ J  + 1 ]  =  Tabla [ J ]
          J = J –1
    FIN-MIENTRAS
    SI  (  Tabla  [ J ] > aux )
          Tabla [ J + 1 ]  =  Tabla  [ J ]
          Tabla [ J ] = aux
   SINO
          Tabla [ J + 1 ]    =  aux
FINPARA
```

7	5	1	2	4
5	**7**	1	2	4
1	**5**	**7**	2	4
1	**2**	**5**	**7**	4
1	**2**	**4**	**5**	**7**

Mejora: Inserción Binaria, busca la posición a insertar mediante la búsqueda dicotómica o binaria al estar esa parte de la tabla ordenada.

D) **Rápido (Quicksort):** Nos situarnos en la mitad de la parte desordenada y intercambiar mayores a un lado y menores al otro, volvemos hacer lo mismo con cada una de las mitades mediante llamadas recursivas.

```
Entero  Tabla [ N ]    /* Variable Global */

PROCEDIMIENTO Quicksort  ( entero Inicial,  entero Final )
ENTORNO
Entero Izda, Dcha
Entero  Aux, Maux
Izda  =  Inicial
Dcha =  Final
Maux =  Tabla [  ( Izda + Dcha) /  2 ]
REPETIR
      MIENTRAS  (Tabla[Izda]  <  Maux)
          Izda = Izda +1
      FIN-MIENTRAS
      MIENTRAS  ( Tabla[Dcha]  >  Maux )
          Dcha = Dcha + 1
      FIN-MIENTRAS
     SI ( Izda <= Dcha )
        aux = Tabla [Izda]
       Tabla[Izda] = Tabla [Dcha]
       Tabla[Dcha] = aux
       Izda = Izda  + 1
       Dcha = Dcha  -1
   FINSI
HASTA  ( Izda > Dcha )
SI ( Inicio <  Dcha )
      Quicksort ( Inicio, Dcha )
FINSI
SI  ( Final >  Izda )
    Quicksort ( Izda, Final)
FINSI
FIN-PROCEDIMIENTO
```

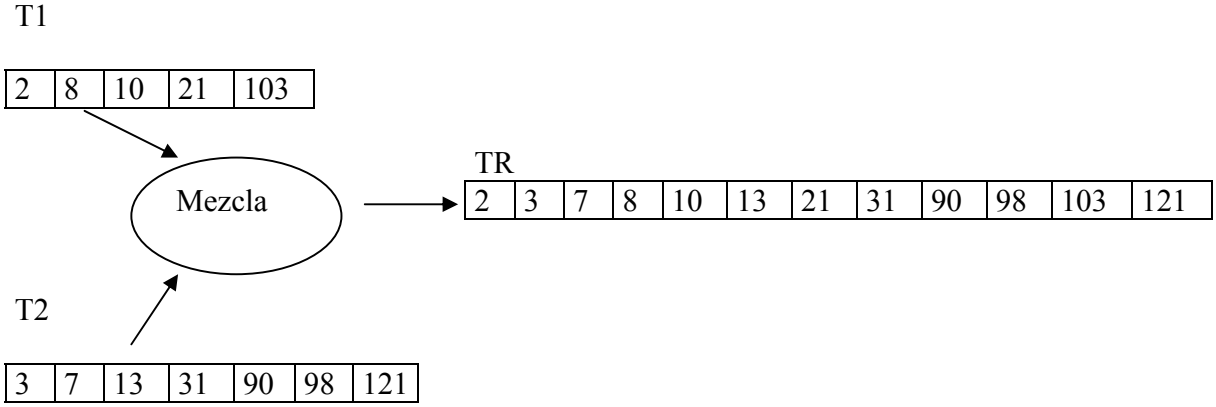
El quicksort es con diferencia el mejor método de ordenación. Existe en la librería estándar de C la función ***qsort*** *que permite ordenar cualquier tipo de tabla utilizando este algoritmo.*

Llamada desde el programa principal :

```
...
Quicksort ( 1, N )
...
```

ALGORITMOS SOBRE VARIAS TABLAS

Algoritmo de mezcla: Copiar en un tabla TR, el contenido de dos tablas ordenadas T1, T2




```
PROCEDIMIENTO Mezclar ( Entero t1[](E), Entero t2[] (E), Entero tr[](S), int lim1(E), int lim2(E))

ENTORNO
   Entero i, j ,k
INICIO
   i = 1
   j = 1
   k = 1
   /* Mezclo ámbas tablas */
   MIENTRAS (  ( i <= lim1) AND (  j <= lim2) )
     SI ( t1[ i ] < t2 [ j ] )
        tr [ k ] = t1 [ i ]
        i = i + 1
     SINO
       tr [ k ] = t2 [ j ];
       j = j + 1
     FIN-SI
     k = k + 1
   FIN-MIENTRAS

  /* Copio el resto */
   MIENTRAS ( i <= lim1 )
      tr [k] = t1[i];
      i = i +1
      k = k + 1
   FIN-MIENTRAS

  /* Copio el resto */
   MIENTRAS ( j <= lim2 )
      tr [k] = t2[j];
      j = j + 1
      k = k + 1
  FIN-MIENTRAS
FIN
```

Cadenas de caracteres (String)

Es muy habitual tener que utilizar tablas que almacenan caracteres, para poder guardar palabras, el nombre de una persona, una oración, etc. Estas tablas que almacenan caracteres se les suele denomina **String o cadenas.**

Para facilitar su manejo la mayoría de los lenguajes tiene funciones y definiciones especiales que facilitan su programación. En C los string son simplemente una tabla de caracteres donde existe el carácter especial '\0' que indica final de la cadena.

Ejemplo de manejo de String:
Algorimos de sobre tablas de caracteres :Copiar, Concatenar, Buscar, Cortar, Subcadena, Longitud, quitarblancos

ALMACENAMIENTO EN MEMORIA

Las tabla se almacenan en la memoria del ordenador siguiendo posiciones consecutivas a partir de una posición inicial donde comienza la tabla. El tamaño ocupado por una tabla estará en función de tamaño de los elementos que almacena y del número de ellos, cuyo valor dependerá de la dimensiones de la tabla.

Unidimensionales

Dirección de Tabla [i] = DirBase + Tamaño del Elemento * (I –1)

Ejemplo: Si tenemos definida una tabla de 10 números reales que comienza en al dirección de memoria 3104, y utilizamos 4 bytes. ¿ Cual seria el tamaño ocupado por la tabla expresado en bytes? ¿ Cúal seria la posición de memoria del elemento Tabla[5]?, ¿Cuál seria la dirección del último byte utilizado ocupado por la tabla?

Solución:
4 * 10 = 40 bytes,
3104 + 4 *(5-1) = 3120,
3104 – 40 –1 = 3143

Bidimensionales

Dir (I , J) = DirBase + Tamaño del Elemento * (Ncolumnas * (I –1) + (J –1))

Dir (I , J) = DirBase + Tamaño del Elemento * (NFilas * (J –1) + (I –1))

ALGORITMOS SOBRE MATRICES (TABLAS BIDIMENSIONALES)

Normalmente cuando trabajamos con una tabla de N dimensiones debemos utilizar N ciclos anidados (PARA o MIENTRAS) para buscar o recorrer la misma.

Ejemplo 1: Inicializar todo los elemento de una matriz a cero.

```
Entorno
  entero  Matriz  [10,4 ]
  entero I, J   /* El indice I indica la fila y el índice J indica la columna */
INICIO
 PARA  I= 1  HASTA 10
    PARA J= 1 HASTA 4
      Matriz[I,J] = 0
   FIN-PARA
 FIN-PARA
FIN
```

Ejemplo 2: Buscar e indicar en que posición está un valor dentro de una tabla bidimensional

```
ENTORNO
  entero  Matriz  [10,4 ]
  entero I, J   /* El indice I indica la fila y el índice J indica la columna */
  entero Valor  /* Valor a buscar en la Matriz */
  Lógico  encontrado
INICIO
 Encontrado = NO
 Mostrar " Introduccir el valor a buscar "
 Leer Valor
  J = 1
 I = 1
 MIENTRAS ( Encontrado = NO ) and ( I <= 10 )
    MIENTRAS ( Encontrado = NO ) and ( J <= 4 )
       Si (  Valor = Matriz [I,J] )
              Mostrar " El elemento se encuentra en la Fila ", I, " Columna" , J
              Encontrado = SI
       SINO
             J= J +1
       FIN-SI
    FIN-MIENTRAS
    I = I + 1
  FIN-MIENTRAS
  SI ( Encontrado = NO )
       Mostrar  " El valor no se encuentra en la Matriz "
  FIN-SI
FIN
```

Ejemplo 3: Acumular en una tabla unidimensional la suma de todas las columnas de una tabla bidimensional

Matriz [5, 4]

10	3	4	5
2	4	6	7
10	5	6	7
8	9	10	3
3	1	0	3

Tabla Suma [4]

33	22	26	25

```
Entero I,J,K
INICIO
 /* Recorro por columnas */
 PARA J = 1 HASTA 4
    Suma [ J ] = 0
    PARA  I = 1 HASTA 5
      Suma [ J] = Suma [ J ] +  Matriz [ I, J ]
    FIN-PARA
FIN-PARA
```

Ejemplo 4: Indicar cuando una Matriz de N x N es simétrica. Suponemos que una matriz es simétrica cuando cualquier elemento en una posición I,J es igual al de la posición J,I.

1	3	8	7
3	**4**	5	9
8	**5**	**6**	10
7	**9**	**10**	**3**

```
FUNCION EsSimetrica ( Entero Matriz [ , ], Entero N ) Devuelve
Logico
Entorno
  Lógico Resultado
Inicio
  Resultado = Cierto
  /* Sólo pregundo si los elemento debajo de la diagonal son      iguale a
los de arriba */
  PARA I = 2 HASTA N
     PARA J = 1  HASTA I – 1
        SI ( Matriz [I,J ] ≠ Matriz [ J, I ] )
           Resultado = Falso
        FIN-SI
    FIN-PARA
  FIN-PARA
  Return Resultado
FIN
```

Ejemplo 5: Tenemos una tabla de 3 x 3 con los valores enumerados Rojo, Azul y Libre, que representan el estado de una partida *de tres en raya*. Indicar si las fichas Rojas ha ganado la partida

ROJO	ROJO	AZUL
ROJO	AZUL	LIBRE
ROJO	LIBRE	LIBRE

```
Ganador = Falso
/* Compruebo primero filas y columnas */
PARA  I = 1 HASTA 3
  SI  (  (T [ I, 1 ] = ROJO ) AND ( T [ I, 2 ] = ROJO ) AND ( T[ I, 3 ] = ROJO ) ) OR
        (T [ 1, I ] = ROJO ) AND ( T [ 2, I ] = ROJO ) AND ( T[ 3, I ] = ROJO ) )
          Ganador = Cierto
  FIN-SI
FIN-PARA
/* Compruebo diagonales */
SI (   (T [ 1, 1 ] = ROJO ) AND ( T [ 2, 2 ] = ROJO ) AND ( T[ 3, 3 ] = ROJO ) )  OR
        (T [ 1, 3 ] = ROJO ) AND ( T [ 2, 2 ] = ROJO ) AND ( T[ 3, 1 ] = ROJO ) )
          Ganador = Cierto
  FIN-SI
```