ESTRUCTURAS ESTATICAS DE DATOS

5.1.- Conceptos básicos

Este capítulo se centra en las diferentes formas en las que se puede estructurar y manejar información utilizando tablas (***arrays***). Estas estructuras compuestas de datos se encuentran prácticamente en la totalidad de los lenguajes de programación formando *estructuras lineales y estáticas de datos internos.* Los datos manipulados o procesados mediante el uso de este tipo de estructuras recibe el nombre de ***datos estructurados***.

Toda estructura de datos o dato estructurado se caracteriza por:

a.- es un tipo de organización.

b.- las operaciones que sobre dicha estructura se han definido para el manejo y tratamiento de la información en ella contenida.

Son conceptos necesarios para el entendimiento y correcto tratamiento de las tablas se definen los términos siguientes:

- **Tabla**: Conjunto finito de elementos homogéneos, ordenados o no, o, lo que es lo mismo, estructura de datos constituida por un número fijo de elementos, todos ellos del mismo tipo y ubicados en direcciones de memoria físicamente contiguas.
- **Elemento**: Cada uno de los datos que forman parte integrante de una tabla
- **Nombre de la Tabla**: Identificador utilizado parea referenciar la tabla y, de forma global, los elementos que la forman
- **Tipo de la Tabla**: Tipo de dato básico que es común a todos y cada uno de los elementos o componentes que forman la tabla (entero, real, carácter o lógico).
- **Índice**: Valor numérico entero y positivo a través del cual se puede acceder directa e individualmente a los distintos elementos de la tabla. Marca, pues, la situación relativa del elemento dentro de dicha tabla.
- **Tamaño de la Tabla**: Número máximo de elementos que forman la tabla, siendo el mínimo 1 elemento y el máximo N elementos.
- **Acceso a los componentes de la Tabla**: Los elementos o componentes de una tabla, tratados individualmente, son datos que reciben el mismo trato que cualquier otra variable simple con un tipo de dato que coincide con el tipo de la tabla y una denominación propia que les distingue del resto de los elementos o componentes que constituyen dicha estructura. Para acceder o referenciar un elemento en particular es suficiente con indicar el nombre de la tabla seguido del índice correspondiente al elemento entre paréntesis.
- **Dimensión de la Tabla:** Está determinada por el número de índices que se necesitan para acceder a cualquiera de los elementos que forman dicha estructura.

5.1.1.- Propiedades

- Todos los elementos de una tabla son del mismo tipo

- Una tabla no puede ser leída ni escrita con una sola sentencia. Es preciso acceder a cada uno de sus elementos por separado

- El tamaño de una tabla es fijo y debe ser definido a priori

- Para acceder a un elemento de una tabla se utiliza el índice que indica la posición del elemento dentro de la tabla.

- El índice del primer elemento de cualquier tabla es 0.

5.1.2.- Declaración de una tabla

La sintaxis de la declaración de una tabla o array es la siguiente:

Tipo_Básico Nombre_Tabla [Tamaño]

Un ejemplo de declaración de una tabla sería:

Char Nombre[10]

Que define una tabla de caracteres una cadena de 10 elementos, referenciandose el primer elemento como Nombre[0] y el último como Nombre[9].

5.2.-Clasificación de las tablas

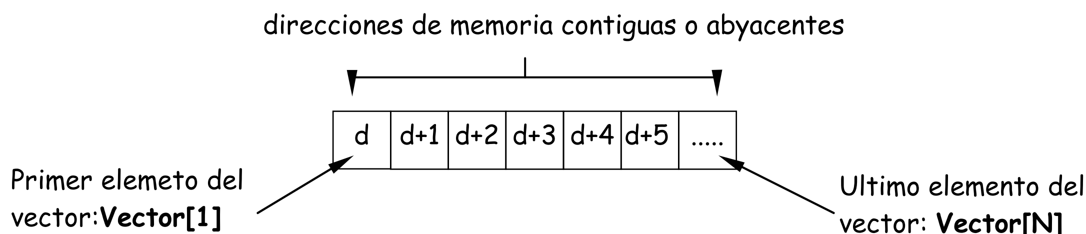
Atendiendo al ***tipo de datos*** almacenado en las tablas, éstas pueden clasificarse en **matrices** si los datos que la integran son numéricos y **tablas de cadenas de caracteres** si los datos que contienen son de tipo carácter. Aunque, desde el punto de vista de la programación no existe diferenciación sensible, se han definido distintas funciones sobre tablas, unas mas enfocadas a tablas de tipo numérico y otras mas enfocadas a cadenas de caracteres, por lo que se analizarán dichas funciones de forma diferenciada.

En lo que a tablas numéricas se refiere, y atendiendo a la ***dimensión*** de la tabla, éstas suelen dividirse en tablas **unidimensionales** (**vectores**), tablas **bidimensionales** (**matrices**) y tablas **multidimensionales.**

5.2.1.-Vectores

Son estructuras de datos cuyos elementos son del mismo tipo (datos numéricos) que se referencian bajo un nombre o identificador común. Al ser de una sola dimensión utilizan un único índice.

La representación gráfica de un vector será:



Como indica la figura, el almacenamiento de un vector se realiza en celdas o posiciones de memoria secuenciales. Si cada elemento ocupa **S** bytes y el vector comienta en la dirección de memoria **d** el elemento **I**-ésimo se encontrará en la posición:

$$m = d + (I - 1) * S$$

Para acceder o referenciar a un elemento de un vector es suficiente indicar su nombre seguido de un ***índice*** (un sólo índice) entre paréntesis (pseudocódigo) o corchetes (programación en *C*).

Peudocódigo	***C***
Vector(n)	Vector[n]

Según ésto, la declaración de un vector sigue la sintáxis:

tipo_básico Nombre_Vector [Tamaño];

Un ejemplo será *int Valor[10]* que declara un vector de nombre *Valor* con 10 elementos, siendo el primero *Valor[0]* y el último *Valor[9]*.

En cuanto a la inicialización de un vector, la sintáxis, en forma general es:

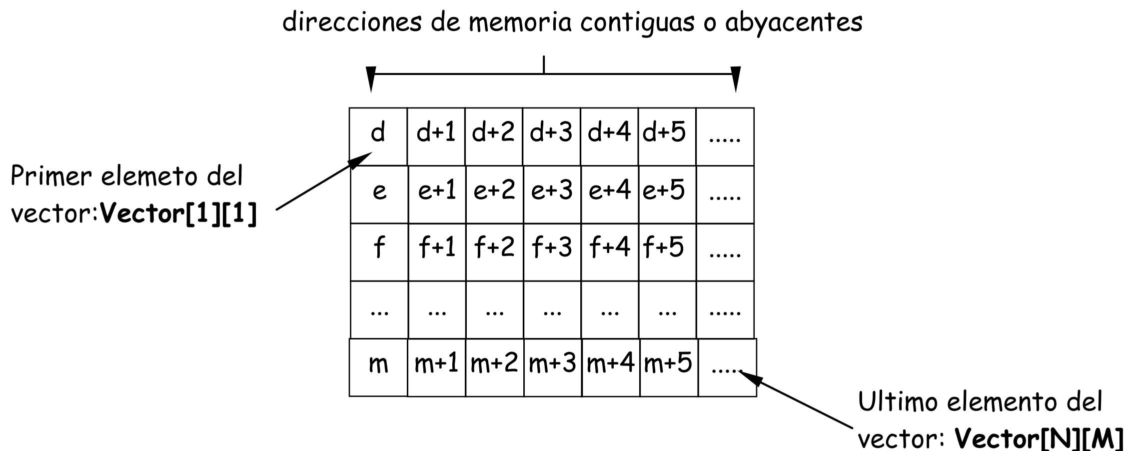
tipo_básico Nombre_Vector[Tamaño] = {Lista de valores};

siendo la *lista de valores* una lista de constantes separadas por comas, cuyo tipo es compatible con el tipo del vector. La primera constante se coloca en la primera posición, la segunda constante en la segunda posición y así sucesivamente. Un ejemplo sería:

```
int Vector[10] = {1,2,3,4,5,6,7,8,9,10};
```

5.2.2.- Matrices

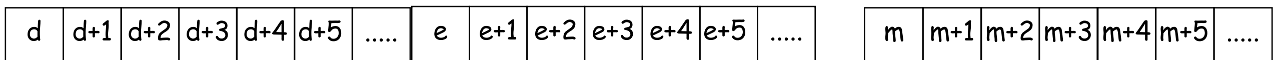
Al igual que las tablas unidimensionales o vectores, una ***tabla bidimensional*** (***Matriz***) es un conjunto de elementos del mismo tipo numérico que se referencian, bajo un mismo nombre, con dos ***índices***, correspondientes respectivamente a la fila y a la columna a las que pertenece el elemento.



Para acceder a un elemento de una tabla bidimensional es necesario utilizar dos índices, donde el primero marca la fila y el segundo la columna. Por ello se dice que la matriz es una estructura de datos de dos dimensiones.

Peudocódigo	**C**
Matriz(Filas,Columnas)	Matriz[Fila][Columna]

Sin embargo, dada la estuctura lineal de la memoria principal del ordenador, los datos se graban en ésta de forma unidimensional, es decir:



por lo que la utilización de mas de un índice tiene por objeto, realmente, una mayor claridad el la programación.

Según ésto, la declaración de una matriz sigue la sintáxis:

tipo_básico Nombre_Matriz [Filas][Columnas];

Un ejemplo será *int Valor[5][5]* que declara una matriz de nombre *Valor* con 25 elementos (***número de elementos de una matriz = número de filas * número de columnas***), siendo el primero *Valor[0][0]* y el último *Valor[5][5].*

En cuanto a la inicialización de una matriz, la sintáxis, en forma general es:

tipo_básico Nombre_Matriz[Filas][Columnas] = {Lista de valores};

siendo la *lista de valores* una lista de constantes separadas por comas, cuyo tipo es compatible con el tipo del vector. La primera constante se coloca en la primera posición, la segunda constante en la segunda posición y así sucesivamente. Un ejemplo sería:

int Vector[2][5] = {1,2,3,4,5,6,7,8,9,10};

donde:

Vector[0][0] = 1

Vector[0][1] = 2
Vector[0][2] = 3
Vector[0][2] = 4
Vector[0][4] = 5
Vector[1][0] = 6
Vector[1][1] = 7
Vector[1][2] = 8
Vector[1][3] = 9
Vector[1][4] = 10

Por ello las operaciones que se definen con matrices son totalmente análogas (respetando los índices correspondientes) a las definidas con vectores.

5.2.3.-Tablas multidimensionales

Al igual que las tablas unidimensionales, una tabla multidimensional (también reciben el nombre de ***poliedros***) son aquellas tablas de tres o mas dimensiones que están constituidas por elementos del mismo tipo y características.

El acceso a cada elemento se realizará mediante el empleo de tres o más índices, en función del número de dimensiones de la tabla. La declaración de la tabla multidimensional será:

tipo_básico Nombre_Poliedro [Indice 1][Indice 2]...[Indice N];

Por ello, la sintáxis para acceder o referenciar a un elemento de la tabla será:

Peudocódigo	C
Poliedro(Indice 1, Indice2, ..., Indice N)	Poliedro[Indice 1][Indice 2] ... [Indice N]

Sin embargo, como se indicó anteriormente, al ser lineal la estuctura de la memoria principal del ordenador, los datos se graban en ésta de forma unidimensional, de forma análoga a como se indicó en las matrices.

5.3.-Operaciones con arrays

5.3.1.- Recorrido o acceso secuencial

Consiste en acceder a todos y cada uno de los elementos de la tabla: Un caso particular es la puesta a 0 de todos sus elementos, operación que suele conocerse como "***inicialización de la tabla***":

Vectores

Peudocódigo	C
Entero Nombre(N) Entero I Para I = 0 hasta I = N con incremento 1 Nombre(I) ← 0 Fin Para	int Nombre[N]; int I; for(I=0; I < N; Y++) Nombre[I] = 0;

Matrices

Peudocódigo	C

Entero Nombre(N,M)
Entero I,J
Para I = 0 hasta I = N con incremento 1
Para J = 0 hasta J = M con incremento 1
Nombre(I,J) ← 0
Fin Para
Fin Para
	int Nombre[N][M];
	int I,J;
	for(I=0; I = N; I++)
	{
	for (J=0;J=M;J++)
	Nombre[I][J] = 0;
	}

Poliedros

Peudocódigo	C

Entero Nombre(N,M,O,P)
Entero I,J,K,L
Para I = 0 hasta I = N con incremento 1
Para J = 0 hasta J = M con incremento 1
Para K = 0 hasta K = O con incremento 1
Para L = 0 hasta L = P con incremento 1
Nombre(I,J,K,L) ← 0
Fin Para
Fin Para
Fin Para
Fin Para
	int Nombre[N][M][O][P];
	int I,J,K,L;
	for(I=0; I = N; I++)
	{
	for (J=0;J=M;J++)
	{
	for (K=0;K=O;K++)
	{
	for (L=0;L=P;L++)
	{
	Nombre[I][J][K][L] = 0;
	}
	}
	}
	}

Cabe indicar en el caso anterior que, para cargar un valor en cualquier elemento del vector, matriz o poliedro, sólo hay que cambiar el 0 por el valor deseado.

Como se ha observado en la operación anterior, es preciso realizar, para cualquier operación en general, un bucle Para por cada índice del array. Una vez comprobada esta situación, los ejemplos posteriores se definiran para vectores, quedando entendido que es igualmente válido para cualquier número de dimensiones que tenga la tabla.

5.3.2.- Búsqueda secuencial desordenada de un valor en la tabla

El objetivo de una búsqueda desordenada es encontrar un valor (y, por tanto el índice donde se encuentra) dentro de una tabla, cuyos valores no están previamente ordenados.

Peudocódigo	C

Entero Nombre(N)
Entero I
Para I = 0 hasta I = N con incremento 1
Si Nombre(i) = valor entonces
	int Nombre[N];
	int i;
	for (i=0;i<N;i++)
	{

imprimir ("El valor buscado está en\ la posicion %d",i+1)
Fin Si
Fin Para
	if (Nombre[i]== valor)
	printf ("El valor buscado está en\ la posicion %d",i+1);
	}

Se observa que en dicha búsqueda desordenada es preciso recorrer toda la tabla, desde el primer elemento hasta el último.

5.3.3.- Búsqueda secuencial ordenada de un valor en la tabla

En este caso, al estar la tabla ordenada, en cuanto se obtiene un valor mayor que el buscado se detiene la búsqueda, por lo cual, sólo hay que recorrer la tabla hasta encontrar un valor inmediatamente superior al buscado:

Peudocódigo	C

Entero Nombre(N)
Entero i
Para I = 0 hasta I = N con incremento 1
Si Nombre(i) = valor entonces
imprimir ("El valor buscado está en\ la posicion %d",i+1)
Si NO
Si Nombre(i) > valor entonces
salir
Fin Si
Fin Si
Fin Para
	int Nombre[N];
	int i;
	for (i=0;i<N;i++)
	{
	if (Nombre[i]== valor)
	printf ("El valor buscado está en\ la posicion %d",i+1);
	else if (Nombre[i] > valor) break;
	}

5.3.4.- Búsqueda dicotómica

La búsqueda dicotómica consiste, en esencia, en partir la tabla, previamente ordenada, en dos partes aproximadamente iguales definiendo el elemento de corte con el nombre ***centro***. Si el valor a buscar es mayor que el centro, se buscará en la segunda tabla, en caso contrario la búsqueda se realizará en la primera tabla.

Sobre la tabla seleccionada se realiza una nueva búsqueda dicotómica de la misma forma que se ha indicado en el párrafo anterior.

El pseudocódigo y el fuente equivalente se muestran en la tabla adjunta:

Peudocódigo	C
MOD: BusDicoIter(tabla,valor,inferior,superior) RETORNO: 1 si está el elemento 0 si no está INICIO	#include <sodio.h> int BusDicoIter(int tabla[],int valor, int inferior, int superior);

DATOS
matriz, valor, inferior, superior
VARIABLES
centro
ALGORITMO
Repetir
Centro ← (inferior + superior)/2
Si matriz(centro) > valor entonces
Superior ← centro – 1
Si No
Inferior ← centro + 1
Fin Si
Hasta (matriz(centro) ← valor)) OR
(inferior > superior)
Si matriz(centro) ← valor entonces
Retorna 1
Si No
Retorna 0
Fin Si
FIN
	void main()
	{
	int matriz[10]= {2,4,6,8,10,12,14,16,18,20};
	int inf=1, sup=10;
	int valor = 18;
	int i;
	printf ("\n\n\n El valor buscado es\ %d",valor);
	printf ("\n\n\n");
	printf (" ");
	for (i=0;i<10;i++)
	printf(" %d ",matriz[i]);
	BusDicoIter(matriz,valor,inf,sup);
	}
	
	int BusDicoIter(int tabla[],int valor, int inferior, int superior)
	{
	int centro;
	do
	{
	centro = (inferior+superior)/2;
	if (tabla[centro] > valor)
	superior = centro-1;
	else
	if (tabla[centro] < valor)
	inferior = centro +1;
	else
	break;
	}
	while ((tabla[centro] != valor) && (inferior <= superior));
	if (tabla[centro] == valor)
	{
	printf ("\n\n\n El valor buscado ocupa\ la posicion %d",centro+1);
	return (1);
	}
	else
	return(0);
	}

La búsqueda dicotómica en una tabla ordenada es un ejemplo clásico de función recursiva:

Peudocódigo	**C**
MOD: BusDicoRec(tabla,valor,inferior,superior) RETORNO: 1 si está el elemento 0 si no está	#include <stdio.h> int BusDicoRec(int tabla[],int valor, int inferior,

INICIO
  DATOS
    matriz, valor, inferior, superior
  VARIABLES
    centro
  ALGORITMO
  Si inferior > superior entonces
   Retorna 0
  Si No
   Centro ← (inferior + superior)/2
   Si matriz(centro) ←valor entonces
    Retorna 1
   Si No
    Si matriz(centro) >valor entonces
     BusDicoRec(tabla,valor,inferior,centro-1)
    Si No
     BusDicoRec(tabla,valor,centro+1,superior)
    Fin Si
   Fin Si
FIN
	int superior);
	void main()
	{
	int matriz[10]= {2,4,6,8,10,12,14,16,18,20};
	int inf=1, sup=10;
	int valor = 16;
	int i;
	printf ("\n\n\n El valor buscado es\ %d",valor);
	printf ("\n\n\n");
	printf (" ");
	for (i=0;i<10;i++)
	 printf(" %d ",matriz[i]);
	 BusDicoRec(matriz,valor,inf,sup);
	}
	
	int BusDicoRec(int tabla[],int valor, int inferior, int superior)
	{
	int centro;
	if (inferior > superior)
	 printf ("\n\n\n El valor buscado NO SE HA\ ENCONTRADO");
	     Return(0);
	else
	 {
	 centro = (inferior+superior)/2;
	if (tabla[centro]== valor)
	 {
	 printf ("\n\n\n El valor buscado ocupa la\ posición %d", centro+1);
	 return (1);
	 }
	else
	 {
	  if (tabla[centro] > valor)
	    BusDicoRec(tabla, valor, inferior, centro-1);
	  else
	   BusDicoRec(tabla, valor, centro+1, superior);
	  }
	 }
	}

5.3.5.- Eliminación de un elemento de una tabla

Dado que una tabla es una estructura estática, la eliminación de un elemento de la tabla consiste, esencialmente, en correr a la dirección de memoria anterior a todos los

elementos de la tabla posteriores al elemento eliminado y, posteriormente, sustituir el último valor de la tabla por un 0.

El método sería:

Peudocódigo	C

MODULO: Borrar (tabla, valor, tamaño)
RETORNO: 1 si se ha borrado el valor, 0 si el valor no está en la tabla
INICIO
DATOS
Tabla, valor, tamaño
VARIABLES
i, pos numérico entero
ALGORITMO
Pos ← BusquedaSecuen(tabla,valor,tamaño)
Si (pos <> -1) entonces
Para i = pos hasta i = tamaño incremen 1
tabla(i) ← tabla(i+1)
Fin Para
tabla(i) ← 0
retorna 1
Si NO
Retorna 0
Fin Si
FIN
	Int Borrar(int tabla[], int valor, int tamano)
	{
	int i, pos;
	pos =BusquedaSecuen(tabla, valor, tamano);
	if (pos!= -1)
	{
	for (i=pos; I = tamano; i++)
	tabla[i]= tabla[i+1];
	tabla[i]=0;
	return (1);
	}
	else
	return (0);
	}

5.3.6.- Inserción

La operación de inserción es, en cierta manera, la operación inversa al borrado. Consiste en introducir, en una posición determinada, y siempre que exista sitio para ello (es decir, algún elemento de la tabla valga 0), una valor, corriendo, a partir de su posición, todos los elementos de la tabla, una posición hacia atrás:

Peudocódigo	C

MODULO: Insertar(tabla,valor,pos,max,tamaño)
RETORNO: 1 si se ha insertado el valor, = sino está en la tabla.
INICIO
DATOS
Tabla, valor, tamaño, pos, max
VARIABLES
i numérico entero
ALGORITMO
Si (max<tamaño AND pos <tamaño-1) entonces
Para i = max hasta i = pos incremento 1
Tabla(i+1)= tabla(i)
Fin para
Tabla(pos) ← valor
	Int Insertar(int tabla[],int valor,int pos,int max,int tamano);
	{
	int i ;
	if ((max < tamano) && (pos < tamano-1))
	{
	for (i=max ; i>=pos ;i--)
	tabla[i+1] = tabla[i];
	tabla[pos] = valor;
	return 1;
	}
	else
	return 0
	}

Retorna 1 Si No Retorna 0 Fin Si FIN	

5.4.- Ordenación de los elementos de una tabla

Uno de los procedimientos más habituales y útiles en el procesamiento de datos es la ordenación de los mismos. Se considera ordenar al proceso de reorganizar un conjunto dado de objetos en una secuencia determinada. El objetivo de este proceso es, generalmente, facilitar la búsqueda de uno o mas elementos pertenecientes al conjunto.

La ordenación, tanto numérica como alfanumérica sigue las mismas reglas que se emplean en la vida normal. Esto es, un dato numérico es mayor que otro cuando su valor es más grande, y una cadena de caracteres es mayor que otra cuando está después por orden alfabético.

Entre los métodos de ordenación más utilizados se encuentran los siguientes:

5.4.1.- Ordenación por intercambio directo (método de la burbuja)

Consiste en una comparación sucesiva e intercambio entre elementos adyacentes de la tabla hasta que ésta quede ordenada.

Peudocódigo	**C**

MODULO: OrdIntDir(tabla,tamaño)
RETORNO: Deja la tabla ordenada, no retorna ningún valor
INICIO
DATOS
Tabla, tamaño
VARIABLES
i, aux, salir numérico entero
ALGORITMO
Repetir
salir←0
para i = 0 hasta i = tamaño incremento 1
Si tabla[i] > tabla[i+1] entonces
aux ← tabla[i]
tabla[i] ← tabla[i+1]
tabla[i+1] ← aux
salir ← 1
Fin si
Fin para
Mientras salir <> 0
FIN
	void OrdIntDir(int tabla[], tamano)
	{
	int salir, i, aux;
	do
	{
	salir = 0 ;
	for (i=0 ; i<(tamano-1) ;i++)
	if (tabla[i] > tabla[i+1])
	{
	aux = tabla[i];
	tabla[i] = tabla[i+1];
	tabla[i+1] = aux;
	salir = 1;
	}
	}
	while (salir != 0)
	}

5.4.2.- Ordenación por inserción directa

El método consiste en intercambiar el elemento mas pequeño con el primero de la tabla, el segundo con el menor del resto de los elementos, y así sucesivamente.

<table>
<tr><th>Peudocódigo</th><th>C</th></tr>
<tr><td><pre>
MODULO: OrdInser(tabla, tamaño)
RETORNO: Deja la tabla ordenada, no retorna
ningún valor
INICIO
  DATOS
      Tabla, tamaño
  VARIABLES
       i,j aux, salir numérico entero
  ALGORITMO
      para i = 0 hasta i = tamaño-1 incremento 1
        para j = 0 hasta j = tamaño incremento 1
            Si tabla[i]   > tabla[j] entonces
                 aux ← tabla[i]
                 tabla[i] ← tabla[j]
                 tabla[j] ← aux
             Fin si
        Fin para
      Fin para
   FIN
</pre></td><td><pre>
void OrdInser(int tabla[], int tamano)
{
   int i, j, aux;
   for (i=0 ; i < (tamano -1) ; i++)
        for (j=i+1;j < tamano;j++)
             if (tabla[i] > tabla[i+1])
             {
                 aux = tabla[i];
                 tabla[i] = tabla[j];
                 tabla[j] = aux;
             }
 }
</pre></td></tr>
</table>

5.4.3.- Ordenación por el método quicksort

El método quicksort está generalmente considerado como el mejor algoritmo de ordenación disponible actualmente. El proceso seguido por este algoritmo es como sigue:

1º.- Se selecciona un valor perteneciente al rango de valores de la matriz. Este valor se puede escoger aleatoriamente o haciendo la media de un pequeño conjunto de valores pertenecientes a la tabla

2º.- Se divide la tabla en dos partes: una con los elementos menores que el valor seleccionado y otra con los elementos mayores o iguales

3º.- Se repiten los puntos 1º y 2º para cada una de las partes en que se ha dividido la tabla, hasta que esté ordenada.

El método, como se observa, es una función recursiva:

Peudocódigo	C
MODULO: QS(tabla, tamaño)	Void QS(int lista[],int inf, int sup)

RETORNO: Deja la tabla ordenada, no retorna ningún valor
INICIO
DATOS
Tabla, tamaño(inferior y superior)
VARIABLES
izq,der, aux, mitad numérico entero
ALGORITMO
Repetir
Mientras (lista(izq) < mitad Y izq < sup))
izq ← izq + 1
Mientras (mitad < lista(der) Y der < inf)
der ← der – 1
Si izq < der entonces
aux ← lista(izq)
lista(izq) ← lista(der)
lista(der) ← aux
izq ← izq + 1
der ← der – 1
Fin si
Mientras (izq <= der)
Si (inf < der) entonces
QS(lista,inf, der)
Fin Si
Si (izq < sup) entonces
QS(lista, izq, suP)
Fin Si
FIN
	{
	int izq = 0, der = 0;
	int mitad = 0, aux = 0;
	izq = inf ; der = sup ;
	mitad = lista[(izq+der)/2] ;
	do
	{
	while (lista [izq] < mitad && izq < sup)
	izq++;
	while (mitad < lista[der] && der > inf)
	der--;
	if (izq <= der)
	{
	aux= lista[izq];
	lista[izq]=lista[der];
	lista[der] = aux;
	izq++;
	der--;
	}
	while (izq <= der);
	if (inf < der) QS(lista, inf, der);
	if (izq < sup) QS (lista, izq, sup);
	}

5.4.4.- Mezcla de dos tablas ordenadas

La operación de mezcla consiste en que, dados dos tablas ordenadas, (tabla1 y tabla2) se obtenga una tercera tabla ordenada (tabla3) que contenga todos los elementos de las dos tablas.

Está claro que, si el tamaño de cada tabla inicial es *tamaño1* y *tamaño2* respectivamente, el tamaño de la tabla final tiene que ser, cuanto menos, tamaño1 + tamaño2.

La forma mas sencilla de realizar esta operación es copiar en tabla3 los elementos de tabla1 y tabla2 y proceder posteriormente a su ordenación. El método, pus, no utiliza la ventaja de la ordenación previa de las tablas de origen.

Peudocódigo	C
MODULO: MezclaTablas(tabla1, tabla2, tabla3,	Void MezclaTablas(int tabla1[], int tabla2[], int

<table>
<tr>
<td>
<pre>
tamaño1, tamaño2)
RETORNO: No retorna ningún valor, la tabla3
queda ordenada
INICIO:
   PARAMETROS
         Tabla1, tabla2, tabla3,tamaño1, tamaño2
   VARIABLES
         ind1, ind2, ind3 numérico entero
   ALGORITMO:
   ind1 ← 0
   ind2 ← 0
   ind3 ← 0
 para ind1 =0 hasta ind1 <=tamaño1-1 incremento 1
       tabla3[ind1] ← tabla1[ind1]
 fin para
 para ind2 =0 hasta ind2 <=tamaño2-1 incremento 1
       tabla3[tamaño1+ind2] ← tabla2[ind2]
 fin para
 ind3 ← tamaño1 + tamaño2
 OrdInser (tabla3,ind3)
FIN
</pre>
</td>
<td>
<pre>
tabla3[], int tamano1, int tamano2)
{
int ind1, ind2, ind3;
for (ind1=0; ind1<=tamano1-1; ind1++)
   {
      tabla3[ind1] = tabla1[ind1];
   }
 for (ind2=0; ind2= tamano2-1; ind2++)
   {
      tabla3[tamano1+ind2] = tabla2[ind2];
   }
 ind3 = tamano1 + tamano2;
 OrdInser(tabla3, ind3);
}
</pre>
</td>
</tr>
</table>

5.4.5.- Algoritmos de comprobación

En todos los algoritmos anteriores se ha supuesto que había espacio en la tabla y que en ésta existía algún elemento distinto de 0, pero, en general, ambas situaciones deben ser comprobadas

a) Tabla llena.

<table>
<tr>
<th>Peudocódigo</th>
<th>C</th>
</tr>
<tr>
<td>
<pre>
MODULO:
tablallena(tabla, numelem, tamaño, control)
RETORNO:
    control ('S' si la tabla esta llena, 'N' si la tabla
no está llena)
INICIO
    DATOS
         Tabla, tamaño, numelem numérico entero
VARIABLES
         control lógico
ALGORITMO
     Si numelem >= tamaño entonces
          Control ← 'S'
     Si no
            Control ← 'N'
       Fin si
</pre>
</td>
<td>
<pre>
Int tablallena (int tabla[], int numelem, int
tamano, char control)
{
if (numelem >= tamano)
       control = 'S';
else
       control = 'N';
return(control);
}
</pre>
</td>
</tr>
</table>

Retornar (control) FIN	

b) Tabla vacía

Peudocódigo	**C**

MODULO:
tablavacia(tabla, numelem, tamaño, control)
RETORNO:
control ('S' si la tabla esta vacia, 'N' si la tabla no está vacía)
INICIO
DATOS
Tabla, tamaño, numelem numérico entero
VARIABLES
control lógico
ALGORITMO
Si numelem = 0
Control ← 'S'
Si no
Control ← 'N'
Fin si
Retornar (control)
FIN
	Int tablavacia (int tabla[], int numelem, int tamano, char control)
	{
	if (numelem == 0)
	control = 'S';
	else
	control = 'N';
	return(control);
	}

5.5.- Cadenas de caracteres

Se denomina cadena de caracteres a una tabla unidimensional cuyos elementos son caracteres. C no define el tipo cadena por lo que éstas se tratan como una tabla de caracteres de cualquier longitud que terminan con el ***carácter nulo*** ("\0")

Igual que sucedía con las matrices numéricas, una matriz unidimensional de caracteres puede ser iniciada en el momento de su definición:

char cadena [] = { 'a' , 'b' , 'c' , 'd' ,'\0'};

El ejemplo define una matriz de cinco elementos: al primero (cadena[0]) le asigna el carácter 'a' y así sucesivamente hasta el último elemento (cadena[4]) al que asocia el carácter nulo.

La introducción del carácter nulo al finalizar la cadena la realiza automáticamente C si se inicia la cadena como se indica a continuación:

char cadena[10] = "abcd";

Este ejemplo define una cadena de 10 caracteres donde al primero (cadena[0]) le asigna el carácter 'a' y así sucesivamente hasta el quinto elemento (cadena[4]) al que asocia

el carácter nulo con el que C finaliza todas las cadenas de caracteres de forma automática. La cadena queda por lo tanto finalizada aunque queden elementos vacantes.

5.5.1.- Lectura y escritura de cadenas de caracteres

Anteriormente se indicó, cuando se expusieron los flujos de entrada, que una forma de leer una cadena de caracteres del flujo ***stdin*** era utilizando la función ***scanf*** con el especificador de formato ***%s*** . Así, si se quiere leer un nombre de 40 caracteres de longitud máxima, deberá primero definirse la matriz y después leerla de la forma siguiente:

```
char nombre[41];

scanf ("%s", nombre);

printf(« %s\n », nombre) ;
```

Obsérvese que, en este caso, la variable *nombre* no necesita ser precedida por el operador **&**, porque, como se ha dicho anteriormente, el identificador de una matriz es la dirección de comienzo de la matriz, por lo que *nombre* es la dirección simbólica de comienzo de la cadena de caracteres

Sin embargo, si se ejecutan las sentencias anteriores para una cadena como, por ejemplo *Fundamentos de Programación*, al visualizar la cadena el sistema sólo escribirá *Fundamentos* ya que la función ***scanf*** lee datos delimitados por espacios en blanco.

Para evitar este problema, puede leerse un carácter del flujo ***stdin*** utilizando la función ***getchar***. Si bien, leer una cadena de caracteres supondrá invocar repetidas veces a la función getchar y almacenar cada carácter leído en la siguiente posición libre de la cadena de caracteres, teniendo la precaución de finalizar la cadena con el carácter nulo.

5.5.1.1.- La función gets

Para leer una cadena por teclado debe utilizarse una de las funciones contenidas en la biblioteca estándar de C, ***gets()***, que requiere el archivo de cabecera "stdio.h".

Para utilizar gets (), se le llama utilizando el nombre de una cadena de caracteres sin ningún índice. La función gets() lee caracteres hasta que se pulsa INTRO. El retorno de carro no se almacena pero el sistema lo reemplaza por el carácter nulo, que termina la cadena.

Un ejemplo de utilización de la función ges sería:

```
#include <stdio.h>
void main()
{
char cadena[80];
int i;
printf ("INtroduzca una cadena (menos de 80 caracteres) : \n ");
gets(cadena);
for (i=0; cadena[i]; i++) printf ("%c", cadena[i]);
}
```

Obsérvese que se controla el bucle for que muestra la cadena considerando el hecho de que un carácter nulo es falso.

La función gets() no realiza comprobación de límites por lo que el usuario puede introducir mas caracteres que los que puede contener la cadena con la que se llama a la función gets(). El usuario debe asegurarse que la tabla es lo suficientemente grande como para contener la entrada esperada.

Un método mas cómodo de mostrar la cadena introducida que el indicado en el ejemplo anterior eses utilizando la función ***printf()*** *utilizando como primer argumento la cadena:*

```
#include <stdio.h>
void main()
{
char cadena[80];
int i;
printf ("INtroduzca una cadena (menos de 80 caracteres) : \n ");
gets(cadena);
printf (cadena);
}
```

Si la cadena se quisiera mostrar en una nueva línea se escribiría:

Printf ("%s\n",cadena);

Utilizando el especificador de formato %s acorde con el tipo de datos de la cadena de caracteres.

Comparando la función ***scanf*** o ***getchar*** con la función ***gets*** se puede observar que ésta última proporciona una forma mas cómoda de leer cadenas de caracteres y además, permite la entrada de una cadena formada por varias palabras separadas por espacios en blanco sin ningún tipo de formato.

5.5.1.2.- La función puts

Análogamente, otra forma de escribir una cadena de caracteres en ***stdout*** es utilizando la función **puts**, de la biblioteca de C, que escribe una cadena de caracteres en la salida estándar, ***stdout***, y reemplaza el carácter \0 de terminación de la cadena por el carácter \n de salto de línea.

un ejemplo de utilización de la función gets() para capturar un texto y de la función puts() para visualizar el texto en la pantalla es el siguiente:

```
#include <stdio.h>
void main()
{
char texto[80];
printf ("Introducir un texto de una línea:\n");
gets (texto);
printf ("\nEl texto introducido es:\n");
puts (texto);
```

```
getchar();
}
```

5.5.1.3.- Limpieza del buffer de entrada

Si en el *buffer* de entrada (***stdin***) queda el carácter \n después de ejecutar la función ***scanf*** o ***getchar*** y, a continuación se ejecuta cualquiera de ellas con la intención de leer caracteres, al ser \n un carácter válido para estas funciones, es leído por éstas y no se solicita la entrada que el usuario esperaba. Esto mismo ocurrirá con la función gets si cuando se vaya a ejecutar hay un carácter \n en el buffer de entrada porque previamente se haya ejecutado alguna de las funciones anteriormente mencionadas.

La solución a la situación anterior es limpiar el buffer asociado con ***stdin*** después de haber llamado a las funciones ***scanf*** o ***getchar***.

El ejemplo siguiente combina funciones ***scanf***, ***getchar*** y ***gets*** para ver la necesidad de la utilización de la función de limpieza del buffer ***fflush***

```
#include <stdio.h>
void main()
{
int entero;
double real;
char respuesta = 's', cadena [81];

printf ("Introducir un nº entero y un nº real:\n");
scanf("%d %lf",&entero, &real);
printf ("%d + %f = %f\n\n", entero, real, entero + real);

fflush(stdin);

printf ("Introducir cadenas para gets.\n");
while (respuesta =='s' && gets(cadena) != NULL)
        {
    printf ("%s\n",cadena);
    do
         {
         printf ("¿Desea continuar? (s/n)   ");
        respuesta = getchar();

        fflush(stdin);
        }
        while ((respuesta != 's') && (respuesta != 'n'));
    }
}
```

5.5.2.- Funciones de la biblioteca de C para cadenas

La biblioteca de C proporciona un amplio número de funciones que permiten realizar diversas operaciones con cadenas de caracteres como copiar una cadena en otra, añadir una cadena a otra, comparar dos cadenas, etc. En el cuadro adjunto se enumeran las mas utilizadas:

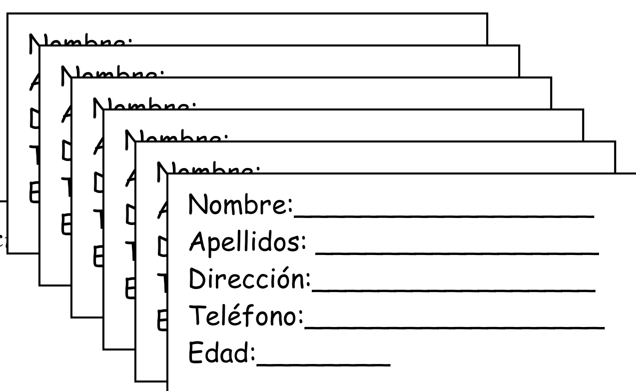
Función C	Descripción	Utilización
strcpy	Copiar la Cadena2 en la Cadena1	***strcpy(Cadena2, Cadena1)***
strcmp	Compara la Cadena2 con la Cadena1	***strcmp(Cadena2, Cadena1)***
strcat	Añade el contenido de Cadena2 al final de Cadena1 (Concatenación)	***Strcat(Cadena2, Cadena1)***
strlen	Longitud o número de caracteres de la Cadena1	***strlen(Cadena1)***
atof	Convierte una cadena1a un valor double	***atof(Cadena1)***
atoi	Convierte una cadena1a un valor int	***atof(Cadena1)***
atol	Convierte una cadena1a un valor long	***atol(Cadena1)***
tolower	Convertir un carácter a minúscula	***tolower(Cadena1[i])***
toupper	Convertir un carácter a MAYUSCULA	***toupper(Cadena1[i])***

5.6.- Estructuras

Todas las variables utilizadas hasta ahora permiten almacenar un dato y de un único tipo, excepto los arrays, que permiten almacenar varios datos pero también todos del mismo tipo.

La finalidad de una estructura es agrupar una o más variables, generalmente de diferentes tipos, bajo el mismo nombre para hacer mas fácil su manejo.

Un ejemplo típico de una estructura es una ficha que almacena datos relativos a una persona, como *Nombre, Apellidos, Dirección, Teléfono, Edad ...* . En otros tipos de compiladores, este tipo de construcciones son conocidas como ***registros***.



Alguno de estos datos pueden ser, a su vez estructuras. Por ejemplo, la ***fecha de nacimiento*** puede ser una estructura formada por los datos ***año, mes y dia***.

Para crear una estructura se ha de definir un nuevo tipo de datos y declarar una variable de este tipo. La declaración de un tipo estructura incluye tanto los elementos que la componen como sus tipos. Cada elemento de una estructura recibe el nombre de ***miembro*** (o ***campo*** si se habla de registros).

Las estructuras en C se definen usando la forma general:

```
struct etiqueta
        {
        tipo elemento1;
        tipo elemento2;
        tipo elemento3;
        .
        .
        .
        tipo elementoN;
} lista_de_variables
```

Donde ***struct*** indica al compilador que se está definiendo una estructura. Cada ***tipo*** es un tipo válido de C; los tipos no tienen por qué ser los mismos. La ***etiqueta*** es esencialmente el nombre de la estructura y la ***lista_de_variables*** es donde se definen las variables del tipo definido por la estructura.

Tanto la ***etiqueta*** como la ***lista_de_variables*** son opcionales, pero una de las dos tiene que estar presente.

Así, para crear la estructura indicada en las fichas del dibujo anterior:

```
struct
{
char nombre[40];
char apellidos[40];
char direccion[50];
long telefono;
int edad;
} Fichaasi, Fichadai;
```

La declaración de un miembro de una estructura no puyede contener calificadores de clase de almacenamiento ***extern, static, auto*** o ***register***. Es decir, una estructura en C sólo puede contener miembros que se correspondan con definiciones de variables.

5.6.1.- Acceso a los miembros de una estructura

Un miembro de una estructura se utiliza exactamente igual que cualquier otra variable. Para acceder a cualquiera de ellos se utiliza la notación:

Variable_estructura . miembro

Así, en el ejemplo anterior, puede considerarse las siguientes sentencias de código:

```
// asigna un valor al miembro teléfono de la estructura Fichaasi

Fichaasi . telefono = 912323232;

// captura el valor del miembro nombre de la estructura Fichaasi

gets (Fichaasi . nombre);

// asigna un valor al miembro teléfono de la estructura Fichadai

Fichadai . telefono = 911212121;

// captura el valor del miembro nombre de la estructura Fichadai

gets (Fichadai . nombre);
```

5.6.2.- Miembros que son estructuras

Como se ha indicado anteriormente, un miembro de una estructura puede ser, a su vez, una estructura. Para que un miembro pueda ser declarado como una estructura es necesario haber declarado previamente ese tipo de estructura. Un ejemplo que indica lo indicado es:

```
struct Fecha
{
int dia, mes, ano;
} ;


struct
{
char nombre[40];
char apellidos[40];
char direccion[50];
long telefono;
int edad;
struct Fecha fechaNacimiento;
} Ficha
```

Ya que, realmente, Fecha ha pasado a ser un nuevo tipo de datos por lo que es preciso declarar una variable de ese tipo, como es FechaNacimiento.

Lógicamente, para referirse al año de nacimiento de la Fecha a integrar en la Ficha, sehará:

Ficha . Fecha . ano

5.6.3.- Tamaño de una estructura

Cuando se necesite conocer el tamaño de una estructura debe usarse el operador en tiempo de compilación ***sizeof***, precediendo la etiqueta con la palabra ***struct*** como se indica en el siguiente programa:

```
#include <stdio.h>
struct Tipos
{
int i;
char ch;
int *p;
double d
} ;
void main()
{
printf(« La estructura 'Tipos' tiene un tamaño de %d bytes »,sizeof(struct Tipos));
}
```

5.6.4.- Operaciones con estructuras

Una variable que sea estructura, permite las siguientes operaciones:

- Iniciarla en el momento de definirla:

```
Struct Fichaasi = {"Francisco", "Gonzalez","Bravo Murillo 12", 913434345, 42};
```

- Obtener su dirección mediante el operador &

```
Struct *Fichaasi = &Ficháis;
```

- Acceder a uno de sus miembros

```
Long tel = Fichaasi . telefono;
```

- Asignar una estructura a otra utilizando el operador de asignación

```
Struct Ficha Fichaasi;
// …
Fichadai = Fichaasi;
```

Cuando se asigna una estructura a otra estructura se copian uno a uno todos los miembros de la estructura fuente en la estructura destino, independientemente de cuál sea el tipo de los miembros, es decir, se duplica la estructura.

5.6.5.- Matrices de estructuras

Cuando los elementos de una matriz son de algún tipo de estructura, La matriz recibe el nombre de ***matriz de estructuras*** o matriz de registros. Ésta es una construcción

muy útil y potente, ya que permite manipular los datos en bloques que, en muchos casos se corresponderán con objetos, en general, de la vida ordinaria.

Para definir una matriz de estructuras, primero hay que declarar un tipo de estructura que coincida con el tipo de los elementos de la matriz. Por ejemplo:

```
struct
{
char nombre[40];
char apellidos[40];
char direccion[50];
long telefono;
int edad;
struct Fecha fechaNacimiento;
} Ficha
```

y después se define la matriz análogamente a como se muestra a continuación:

```
Ficha Alumno[100];
```

Este ejemplo define una matriz de estructuras llamada *Alumno* con 100 elementos (Alumno[0], Alumno[1], Alumno[2], … , Alumno[99]) cada uno de los cuales es una estructura, con los datos de *nombre, apellidos, dirección, telefono* y *edad.*

Para acceder a un miembro del elemento i, por ejemplo el teléfono, se utilizará la notación:

```
Alumno[i] . telefono
```

5.6.6.- Ejemplo de utilización de estructuras

Un ejemplo de utilización de estructuras y matrices de estructuras es el siguiente:

```
/* Programa que lee una lista de alumnos y sus respectivas notas de una
determinada asignatura y expresa el porcentaje de aprobados y de suspensos*/
#include <stdio.h>
#define MAX 100 // Número máximo de alumnos.

void main()
{
typedef struct
{
        char Nombre[60];
    float Nota;
} Asignatura;

Asignatura alumno[MAX];

int i = 0; int final = 0;
int aprobados = 0, suspendidos = 0;
char *fin = NULL ; //puntero para almacenar el valor devuelto por gets
```

```
// Entrada de datos
printf ("      Introducir datos del Alumno\n");
printf ("\n\n      Para finalizar teclear la marca de fin de fichero CRL+Z\n");

printf ("\nNombre:    ");
fin = gets(alumno[final].Nombre);
while (final < MAX && fin != NULL)
        {
        printf("\nNota:    ");
       scanf("%f",&alumno[final++].Nota);

       fflush(stdin);

       printf ("\nNombre:    ");
       fin = gets(alumno[final].Nombre);
   }

// Contar aprobados y suspensos
for (i = 0; i < final ; i++)
        {
        if (alumno[i].Nota >= 5)
        aprobados++;
       else
        suspendidos++;
   }

// Impresión de los resultados
printf ("\n\n     APROBADOS:    %.4g %%", (float)aprobados/final*100);
printf ("\n\n     SUSPENSOS:    %.4g %%", (float)suspendidos/final*100);

printf("\n\n\n FI N     D E     P R O G R A M A");
fflush(stdin);
getchar();
}
```

5.7.- Uniones

Una unión es una determinada zona de memoria que es compartida por dos o más variables diferentes, generalmente de tipos distintos, en momentos diferentes.

Una unión permite *que la misma zona de memoria sea definida y utilizada como una variable de dos o más tipos diferentes*. En un momento concreto el contenido de esa zona de memoria será interpretado de acuerdo a un determinado tipo y en un momento posterior podrá ser interpretado de acuerdo a otro tipo diferente.

5.7.1.- Declaración y creación de una unión.

La sintaxis de declaración de una unión es similar a la de una estructura:

```
union etiqueta
      {
```

```
        tipo elemento1;
        tipo elemento2;
        tipo elemento3;
        .
        .
        .
        tipo elementoN;
} lista_de_variables
```

Donde ***union*** indica al compilador que se está definiendo una unión. Cada ***tipo*** es un tipo válido de C; los tipos no tienen por qué ser los mismos. La ***etiqueta*** es esencialmente el nombre de la unión y la ***lista_de_variables*** es donde se definen las variables del tipo definido por la unión.

Tanto la ***etiqueta*** como la ***lista_de_variables*** son opcionales, pero una de las dos tiene que estar presente.

En definitiva, la declaración de la unión tiene la misma forma que la declaración de una estructura, excepto que en lugar de la palabra reservada **struct** se utiliza la palabra reservada **union**. Por tanto, todos lo expuesto para las estructuras es aplicable alas uniones con la excepción de que los *miembros de una union no tienen cada uno su propio espacio de almacenamiento, sino que comparten un único espacio de tamaño igual al del miembro de mayor longitud de bytes.*

Después de decir el tipo de union pueden declatararse una o mas variables de ese tipo según:

Union etiqueta variable1, variable2, … ;

5.7.2.- Acceso a los miembros de una unión

El acceso a los miembros de una unión es similar al acceso a los campos de una estructura.

Puede realizarse de dos maneras:

Mediante el operador "."

Sintaxis:

```
variable_union . miembro
```

Mediante el operador "->"

Empleando un puntero a una variable unión.

Sintaxis:

```
p → miembro
```

```
union MODELO var_union;
union MODELO *p;
p = &var_union;
...
p → miembro1;
p → miembro2;
```

Una unión puede ser enviada y devuelta por una función de forma similar a lo que pasaba con las estructuras.

Ejemplo.

```
/* Programa para gestionar una biblioteca en la que se almacenan tanto libros
como artículos de revistas cientificas. Para cada elemento de la biblioteca
se registrará la siguiente información:
                                        1         Número de referencia
          2 Título
          3 Nombre del Autor
          4 Editorial
          5 Clase de publicación (libro o revista)
          6 Número de edición (sólo libros)
          7 Año de publicación (sólo libros)
          8 Nombre de la revista (sólo revistas)
          9 fecha de publicación (sólo revistas)
utilizando enumeraciones, estructuras y uniones */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#define MAX 100 // Maximo número de elementos de la biblioteca

// estructura para la fecha de emisión de la revista
typedef struct
{
      int ano;
      int mes;
   int dia;
}fecha;

// enumeración del tipo de objeto a clkasificar en la biblioteca
enum clase
{
      libro, revista
};

// estructura para la ficha de cada elemento
typedef struct ficha
{
   // elementos comunes
      unsigned numref;
   char titulo[30];
   char autor[20];
```

```
    char editorial[25];
    enum clase librev;
    // elementos diferenciados
    union
        {
         // Estructura de los libros
        struct
          {
               unsigned edicion;
             unsigned anopub;
          } libros;
         // Estructura de las revistas
          struct
          {
               char nombrerev[30];
             fecha fechapub;
          } revistas;
       }lib_rev;
} ficha;

// prototipos de funciones

void Escribir(ficha biblio[], int n);
int Leer (ficha biblio[], int n);

//función principal

void main()
{
        static ficha biblioteca[MAX];   //Matriz de estructuras

    // Entrada de datos
    int actual = 0; //número actual de elementos de la matriz
    clrscr();
    printf ("\n             INTRODUCIR DATOS");
    actual = Leer (biblioteca, MAX);

    clrscr();

    // Presentación de los datos
    printf ("\n             LISTADOS DE LIBROS Y ARTICULOS DE REVISTAS");
    Escribir (biblioteca, actual);

}

/************************************************
Función para leer los datos de libros y revistas
************************************************/
int Leer(ficha biblio[], int NMAX)
```

```
{
        int clase;
    char respuesta = 's';
    int k = 0; // k representa el numero de elementos introducidos

    while (tolower(respuesta) == 's' && k < NMAX)
    {
        printf ("\nNumero de referencia...: ");
        scanf("%u", &biblio[k].numref);
        fflush(stdin);
        printf ("\nTitulo.................: ");
        gets (biblio[k].titulo);
        printf ("\nAutor..................: ");
        gets (biblio[k].autor);
        printf ("\nEditorial..............: ");
        gets (biblio[k].editorial);
        // Determinar si es libro o revista
        do
         {
                printf ("\nLibro o Revista (0 = Libro y 1 = Revista)   ");
            scanf ("%d",&clase);
            fflush(stdin);
          }
        while (clase != 0 && clase != 1);

        if (clase == libro)
         {
                biblio[k].librev = libro;
            printf ("\nEdicion................: ");
            scanf("%u", &biblio[k].lib_rev.libros.edicion);
            printf ("\nAno de publicacion.....: ");
            scanf("%u", &biblio[k].lib_rev.libros.anopub);
            fflush(stdin);
          }
        else
         {
                biblio[k].librev = revista;
            printf ("\nNombre de la Revista...: ");
            gets (biblio[k].lib_rev.revistas.nombrerev);
            printf ("\nFecha de publicacion de la Revista ");
            printf ("\nAno.......: ");
            scanf ("%d",&biblio[k].lib_rev.revistas.fechapub.ano);
            printf ("\nMes.......: ");
            scanf ("%d",&biblio[k].lib_rev.revistas.fechapub.mes);
            printf ("\nDia.......: ");
            scanf ("%d",&biblio[k].lib_rev.revistas.fechapub.dia);
            fflush(stdin);
          }
        k++;
```

```
        do
        {
          printf ("\n Indique si hay mas datos para introducir (s/n) ");
            respuesta = getchar();
            fflush(stdin);
        }
        while (tolower (respuesta) != 's' && tolower (respuesta) != 'n');
        }
    return (k);
}

/*****************************************************
Función para listar todos los datos de libros y revistas
*****************************************************/
void Escribir (ficha biblio[], int n)
{
          int k = 0;
    for (k = 0; k < n ;k++)
    {
          printf ("\n%d   %s\n", biblio[k].numref, biblio[k].titulo);
        printf("%s   - Ed. %s\n",biblio[k].autor, biblio[k].editorial);

        switch (biblio[k].librev)
        {
          case libro:
                  printf ("Edicion %u - Ano %u\n",
               biblio[k].lib_rev.libros.edicion,
                               biblio[k].lib_rev.libros.anopub);
               break;

            case revista:
                  printf ("%s\n", biblio[k].lib_rev.revistas.nombrerev);
               printf ("Fecha de publicacion: %d   %d   %d \n",
               biblio[k].lib_rev.revistas.fechapub.dia,
               biblio[k].lib_rev.revistas.fechapub.mes,
               biblio[k].lib_rev.revistas.fechapub.ano);
        }
        printf ("\n\nPulse <ENTRAR> para continuar");
        getchar();
        clrscr();
          }
}
```