

# TEMA 11

## Organización lógica de los datos.

## Estructuras Estáticas

Por:

Tomás Saavedra Fernández

[informaticapreparacion@gmail.com](mailto:informaticapreparacion@gmail.com)

[tasaavedra@gmail.com](mailto:tasaavedra@gmail.com)

[www.informaticapreparacion.es](http://www.informaticapreparacion.es)

## ÍNDICE

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>3</b>
<b>2</b>	<b>DEFINICIONES Y CONCEPTOS .....</b>	<b>3</b>
2.1	DEFINICIÓN ESTRUCTURA DE DATOS .....	3
2.2	CARACTERÍSTICAS DESEABLES DE UNA ESTRUCTURA DE DATOS .....	3
2.3	ELECCIÓN DEL TIPO DE “ESTRUCTURA DE DATOS” .....	4
2.4	CLASIFICACIÓN DE LOS DATOS .....	4
<b>3</b>	<b>ESTRUCTURAS ESTÁTICAS INTERNAS .....</b>	<b>5</b>
3.1	TABLAS .....	5
3.2	CONJUNTOS .....	11
3.3	UNIONES .....	12
3.4	ENUMERACIONES .....	12
<b>4</b>	<b>ESTRUCTURAS ESTÁTICAS EXTERNAS .....</b>	<b>13</b>
4.1	REGISTROS .....	13
4.2	FICHEROS O ARCHIVOS .....	14
<b>5</b>	<b>PLANTEAMIENTO DIDÁCTICO .....</b>	<b>15</b>
<b>6</b>	<b>CONCLUSIÓN .....</b>	<b>16</b>
<b>7</b>	<b>BIBLIOGRAFÍA .....</b>	<b>16</b>

## 1 INTRODUCCIÓN

Con este tema comenzamos un bloque dedicado a la Programación, donde estudiaremos los distintos elementos que participan en la elaboración de un programa informático.

Los computadores son máquinas para el tratamiento automático de la información. Esta información no se almacena ni se representa al azar, sino que debe organizarse o estructurarse de forma adecuada para obtener un rendimiento razonable en su almacenamiento, tratamiento y recuperación. El tratamiento de la información se debe realizar de un modo sistemático. La resolución de cualquier problema conlleva el encontrar un método de resolución expresado con la suficiente precisión para poder ser descompuesto en acciones realizables por el computador, esto es lo que denominaremos un algoritmo, igualmente necesitamos una estructura de datos donde se puedan guardar los datos necesarios para el funcionamiento de este algoritmo.[5]

En este tema comenzamos estudiando el concepto de dato desde un enfoque lógico. Describimos los tipos de datos elementales más usuales en informática, analizando algunas estructuras de datos utilizadas en programación, en sistemas operativos, o en el diseño físico de computadores.

**Nota:** Marcamos [numero], las referencias a la Bibliografía.

## 2 DEFINICIONES Y CONCEPTOS

La eficiencia y la velocidad de un programa depende de: el tipo de computador, algoritmo, compilador y el sistema operativo, pero además existe otro factor que influye en ello y es la organización lógica de los datos.[2]

Para almacenar la información usamos “Tipos de Datos”, que se caracterizan por:

- Los valores que se pueden almacenar
- Las operaciones que con ellos podemos realizar

### 2.1 Definición estructura de datos

Una estructura de datos o tipo de dato estructurado es un tipo de dato construido a partir de otros tipos de datos. Así, un dato de tipo complejo, que representa al conjunto de los números complejos, es un par ordenado de datos reales, y por tanto un tipo de dato estructurado [1].

Un dato de tipo estructurado está compuesto por una serie de datos de tipos elementales y alguna relación existente entre ellos. Normalmente la relación suele ser de orden, aunque puede ser de otro tipo.

### 2.2 Características deseables de una estructura de datos

- Eficiencia en el uso de la memoria.
- Rapidez de Acceso a la memoria.
- El tipo de organización.
- Las operaciones que se va a poder realizar.

### 2.3 Elección del tipo de “Estructura de Datos”

Depende de:

- Del tipo de aplicación.
- Del Lenguaje de programación.

### 2.4 Clasificación de los datos

según Joyanes [1]:

Los datos pueden ser:

- **Elementales**

*contienen 1 solo elemento ( Entero,Carácter,Lógico), por lo tanto no están formados por otros elementos formando estructuras*

- **Estructurados**

*Están compuestos o contruidos basados en tipos de datos elementales, según la cantidad de datos almacenados pueden ser:*

- **Internos**

*Están en la memoria principal.*

- **Estructuras Estáticas de Datos**

*El número de elementos que podemos almacenar es fijo.*

*Vectores,Tablas,Poliedros,Enumeraciones,Uniones*

- **Estructuras Dinámicas de Datos**

*El número de elementos puede varias durante la Ejecución del programa*

- **Lineales:** listas,pilas,colas

- **No lineales:** arboles,grafos

- **Externos**

*Están en la memoria secundaria o externa:Registros y Ficheros*

- **Derivados**

*Punteros*

### 3 ESTRUCTURAS ESTÁTICAS INTERNAS

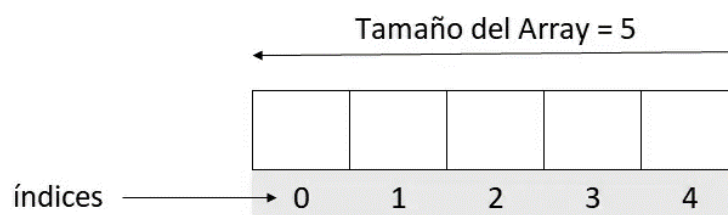
#### 3.1 Tablas

##### 3.1.1 Tablas unidimensionales ( Vectores, Arrays )

Una Tabla unidimensional, un vector o array es la estructura de datos más usual. Existe en la mayoría de los lenguajes de programación y en algunos es de las pocas estructuras de datos existentes (Basic y FORTRAN).

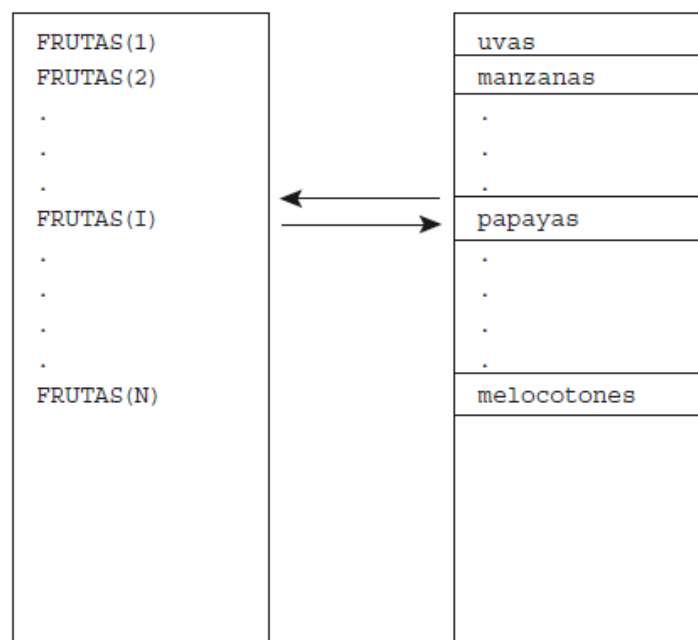
Un array es una estructura de datos formada por una cantidad fija de datos de un mismo tipo, cada uno de los cuales tiene asociado uno o más índices que determinan de forma unívoca la posición del dato en el array. Cada índice es un dato de tipo subrango. Para cada combinación posible de valores de índices existe un y sólo un dato del tipo constituyente, o elemento del array.

Podemos imaginar un array como una estructura de celdas donde se pueden almacenar valores.



Figura, ejemplo de la representación de un array de 5 elementos

Los vectores, pueden contener datos no numéricos, es decir, tipo "carácter". Por ejemplo, un vector que representa las frutas que se venden en un supermercado



Los vectores se almacenan en la memoria central de la computadora en un orden adyacente. Así, un vector de cincuenta números denominado NUMEROS se representa gráficamente por cincuenta posiciones de memoria sucesivas.

Memoria		
NUMEROS [1]		Dirección X
NUMEROS [2]		Dirección X+1
NUMEROS [3]		Dirección X+2
NUMEROS [50]		Dirección X+49

Para definir un vector se indica el nombre, el número máximo de elementos y el tipo de dato que contiene, por ejemplo en pseudocódigo y en C:

mivector: vector [1..10] de Enteros  
*int mivector[9];*

**inicializarlos**

*int mivector[9]={ 3,6,7,5,7,7,3,2,1,2};*

**Acceso**

*dato=mivector[6];*

**Recorrido**

```
for (i=0;i<10;i++)
{
    .....= mivector[i];
}
```

nota: en C y C++ , los vectores o arrays comienza en la posición 0.

### **Insertión.**

Supone la adición lógica, nunca física, de un elemento del vector. Si se produce al final del vector, debe comprobarse que el vector dispone de espacio de memoria suficiente.

En caso de añadir elementos en el interior del vector, es necesario reorganizar el resto de elementos del vector, de tal manera que todos los elementos situados a la derecha del punto de inserción debe ser desplazados una posición hacia arriba antes de la inserción.

### **Borrado.**

Supone la eliminación lógica, nunca física, de un elemento del vector. Si se produce al final del vector, debe comprobarse que el vector dispone de al menos un elemento.

En caso de eliminar elementos en el interior del vector, es necesario reorganizar el resto de elementos del vector, de tal manera que todos los elementos situados a la derecha del punto de eliminación debe ser desplazados una posición hacia abajo para evitar que queden huecos libres.

### **Búsqueda.**

Consiste en localizar la posición de un elemento con una determinada clave. La eficiencia de una búsqueda dependerá enormemente del grado de ordenación previa de la que dispongan los datos.

Los algoritmos de búsqueda pueden ser:

*Búsqueda lineal.* Se aplica a datos ordenados y no ordenados. Consiste en recorrer el vector desde el primer elemento al último hasta encontrar un elemento cuya clave coincida con la buscada o hasta que se acabe el vector. En este último caso, debe indicarse la no existencia de dicho valor en el vector.

*Búsqueda binaria.* Se aplica a datos ordenados. Consiste en comparar en primer lugar con el componente central del vector y se decide la mitad en la que debe encontrarse el valor buscado. El proceso se repite hasta que se encuentre el valor buscado o hasta que el tamaño del intervalo de búsqueda quede anulado.

Ejemplo en C, de búsqueda lineal:

```
for(i=0;i<N;i++)
{
    if(array[i]==elemento)
    {
        solucion=array[i];
        i++;
    }
}
```

### **Ordenación**

Consiste en organizar los elementos del vector por orden creciente o decreciente según una determinada clave. La eficiencia de un método de ordenación suele determinarse según el número de comparaciones realizadas y el número de movimientos de las mismas. Ambos son función del número de elementos que componen el vector a ordenar.

Los algoritmos de ordenación pueden ser:

*inserción directa.* Consiste en tomar los elementos del vector desde el segundo hasta el último y con cada uno de ellos repetir el siguiente conjunto de operaciones:

Se saca del vector el elemento *i*-ésimo utilizando una variable auxiliar.

Desde el anterior al que estamos tratando y hasta el primero, desplazamos un lugar a la derecha todos los que sean mayores para buscar su hueco.

Encontrado el hueco del elemento, lo insertamos en él.

*selección directa.* Consiste en tomar los elementos del vector desde el primero hasta el penúltimo y con cada uno de ellos repetir el siguiente conjunto de operaciones:

Se guarda la posición del elemento *i*-ésimo utilizando una variable auxiliar.

Desde el posterior al que estamos tratando y hasta el último, guardamos en la variable auxiliar la posición del elemento menor.

Se intercambian los elementos *i*-ésimo y el elemento menor encontrado.

*Intercambio directo o burbuja.* Consiste en recorrer sucesivamente de izquierda a derecha el vector y realizar pasadas sucesivas desde el primer elemento hasta el penúltimo, comparando cada uno de ellos con el siguiente e intercambiándolos cuando estén descolocados. Existe una mejora de los métodos de intercambio directo en la que se comprueba mediante un switch si el vector está totalmente ordenado después de cada pasada, terminando la ejecución en caso afirmativo.

*Quick Sort.* Consiste en seleccionar un elemento especial llamado pivote, buscar un elemento situado a la izquierda del pivote mayor que él y un elemento situado a la derecha del pivote menor que él, e intercambiarlos. Debe repetirse el proceso hasta que las búsquedas por la izquierda y por la derecha se crucen. Después recursivamente se hace de nuevo la partición sobre cada una de las dos partes obtenidas siempre que tengan más de un elemento.

Ejemplo en C ,de ordenación por intercambio directo o burbuja. [6]

```
for (i=0; i<n-1; i++)
{
    for (j=i+1; j<n; j++)
    {
        if(V[i]>V[j])
        {
            aux = V[i];
            V[i] = V[j];
            V[j] = aux;
        }
    }
}
```

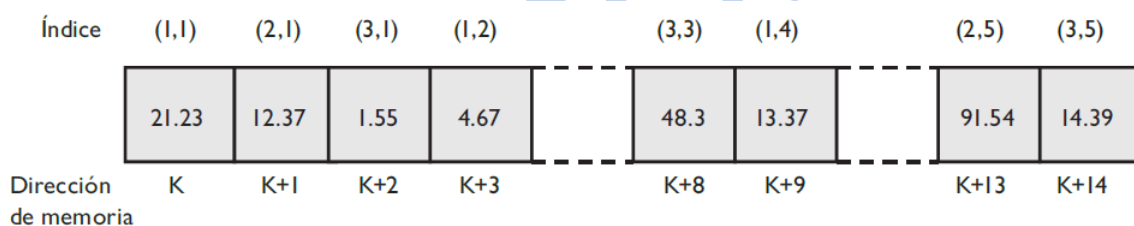


### 3.1.2 Tablas bidimensionales ( Matrices o Tablas )

También llamadas array bidimensionales, matrices o simplemente tablas, son estructuras que agrupan un número fijo y finito de vectores del mismo tamaño y tipo de datos en posiciones contiguas de memoria bajo un nombre común, como un vector de vectores.

#### Características

- Son estructuras de dos dimensiones.
- Cada uno de los elementos de la matriz se referencia mediante el nombre común, el índice del vector contenedor (fila) y el índice del elemento dentro del vector contenido (columna).
- El número máximo de elementos de una matriz se define en tiempo de compilación de los programas. No es posible modificarlo en tiempo de ejecución.
- Las matrices se almacenan en memoria ordenadamente por filas.
- La definición de una matriz es igual a la de un vector, indicando además el número de elementos de los vectores que forma parte del vector principal.
- Extendiendo este concepto es posible realizar matrices de más de dos dimensiones, aunque no se suelen utilizar más de tres dimensiones debido a su complejidad de manejo.



*Almacenamiento en memoria del array bidimensional (Suponiendo almacenamiento por columnas.)*

Para definir una tabla se indica el nombre, el número máximo de elementos y el tipo de dato que contiene, por ejemplo en pseudocódigo y en C:

mitabla: tabla [1..10][1..15] de Enteros

`int mitabla[9][14];`

	1	2	3	4	...	J	...	N
1								
2								
...								
I						B[I, J]		
...								
M								

Elemento B [I, J] del array B.

**Figura, Representación de una posición en un tabla**

**inicializarlos**

```
int mivector[9][14]= { { 3,6,7,5,7,7,3,2,1,2} , {5,6,6,7,7,8,9,4,4,4,4,5,6} };
```

**Acceso**

```
dato=mivector[6][5];
```

**Recorrido**

```
for (i=0;i<10;i++)  
{  
    for (j=0;j<14;j++)  
    {  
        .....= mivector[i][j];  
    }  
}
```

Tanto la inserción como el borrado, es aplicable lo anteriormente mencionado para los vectores.

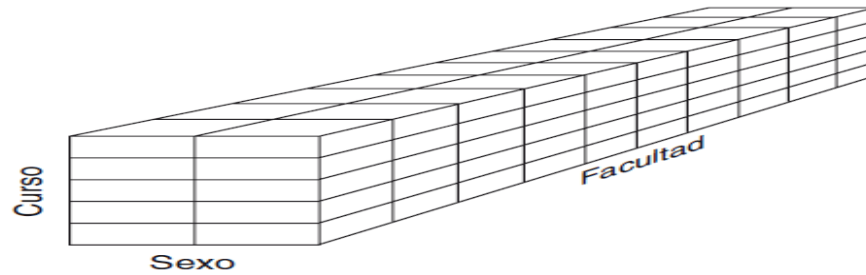
**Búsqueda**

Igualmente lo explicado en vectores, es aplicable aquí, tan solo , existe un bucle por cada dimensión del array

```
for(i=0;i<N;i++)  
{  
    for(j=0;j<M;j++)  
    {  
        if(array[i][j]==elemento)  
        {  
            solucion=array[i][j];  
            i++;  
        }  
    }  
}
```

**3.1.3 Tablas multidimensionales ( Poliedros )**

Definimos un poliedro como un vector o array de mas 2 dimensiones, todo lo estudiado a vectores y tablas, es perfectamente aplicable a los poliedros, pero dada la complejidad de uso de mas de 2 dimensiones , su uso esta muy limitado como estructura de datos.



Array de tres dimensiones.

### 3.2 Conjuntos

Es una Estructura de datos Estática donde todos los elementos que la componen son del mismo tipo.

A diferencia de los vectores en los conjuntos no existe orden entre sus componentes.

Por ejemplo, Conjunto llamando Vocales

Vocales= {A,E,I,O,U}

El Orden o Cardinalidad: Cantidad de elementos que contiene.

Esta Estructura Estática existe en determinados Leng. de programación, pero otros no lo permiten, pero lo simulan a través de vectores.

#### 3.2.1 Operaciones sobre 1 Conjunto

##### Adición

Insertar un Nuevo elfo al conjunto, Insertar (elemento.conjunto)

##### Eliminación

Eliminar un elemento del conjunto, Borrar (elemento, conjunto)

##### Cardinalidad

Cantidad de elementos que pertenecen al conjunto, Tamaño (conjunto)

##### Pertenencia

Si un elemento pertenece al conjunto o no, Miembro(elemento,conjunto)

Antes de realizar la operación correspondiente tendremos que realizar las declaraciones del conjunto y de las variables a leer o escribir.

#### 3.2.2 Operaciones sobre 2 Conjuntos

Estas operaciones dan como resultado un nuevo conjunto:

##### Unión

Da como resultado un nuevo conjunto que contiene los elementos comunes y no comunes a ambos conjuntos.

Crearemos un conjunto auxiliar donde iremos insertando los elementos de ambos'

##### Intersección

Da como resultado otro conjunto que contiene los elementos comunes a ambos, es decir, los elementos que se encuentran en los dos conjuntos.

Crearemos un conjunto auxiliar donde insertaremos los elementos que coinciden.

### Resta

Da como resultado otro conjunto con los elementos del primer conjunto que no se encuentran en el segundo conjunto.

Crearemos un conjunto auxiliar donde estarán dichos elementos

### 3.3 Uniones

Una unión es un tipo de dato derivado (estructurado) que contiene sólo uno de sus miembros a la vez durante la ejecución del programa. Estos miembros comparten el mismo espacio de almacenamiento; es decir, una unión comparte el espacio en lugar de desperdiciar espacio en variables que no se están utilizando. Los miembros de una unión pueden ser de cualquier tipo, y pueden contener dos o más tipos de datos. La sintaxis en C para declarar un tipo union es idéntica a la utilizada para definir un tipo estructura, excepto que la palabra union sustituye a Struct

Una unión es similar a una estructura con la diferencia de que sólo se puede almacenar en memoria de modo simultáneo un único miembro o campo, al contrario que la estructura que almacena espacio de memoria para todos sus miembros.

Ejemplo en C:

```
union frases
{
    char mensajes[50];
    char ayudas[50];
    char lineas[50];
}
```

### 3.4 Enumeraciones

Una de las características importantes de la mayoría de los lenguajes de programación modernos es la posibilidad de definir nuevos tipos de datos. Entre estos tipos definidos por el usuario se encuentran los tipos enumerados o enumeraciones.

Un tipo enumerado o de enumeración es un tipo cuyos valores están definidos por una lista de constantes de tipo entero. En un tipo de enumeración las constantes se representan por identificadores separados por comas y encerrados entre llaves.

Ejemplo en C:

```
enum dias_semana {LUNES=1, MARTES=2, MIERCOLES=3, JUEVES=4, VIERNES=5, SÁBADO=6, DOMINGO=7};
```

## 4 ESTRUCTURAS ESTÁTICAS EXTERNAS

Según [6], podemos englobar , aun residiendo en memoria externa, los Registros y Ficheros, como estructura estáticas, extendiendo así de una manera mas amplia la clasificación “clásica” de [2] de estructura estática de datos.

### 4.1 Registros

Agrupan un número fijo y finito de datos de distinto tipo y tamaño bajo un nombre común.

Pueden ser datos de tipo estándar o definidos por el usuario.[1]

#### 4.1.1 Características

Cada uno de los elementos del registro se denomina campo y se representa mediante un identificador propio. Los campos suelen mantener una relación lógica entre sí.

La declaración de un registro crea un nuevo tipo de dato que puede asignarse a una variable. Cada uno de los campos de un registro se referencia mediante el nombre de la variable a la que se ha asignado el registro, un operador (punto) y el nombre del campo.

Los campos de un registro pueden ser de tipo simple o de tipo estructurado estático.

Los registros ocupan posiciones consecutivas de memoria. Cuando se declara una variable del tipo del registro, el compilador reserva automáticamente el espacio de memoria necesario para cada uno de sus campos, manteniendo una copia de cada campo de la misma.

La información contenida en un registro se puede asignar a otro del mismo tipo mediante una única instrucción de asignación. No es necesario asignar valor a valor cada campo.

La definición de un registro indica al compilador sus características, e incluye el identificador del registro y los tipos de dato e identificadores de cada uno de los campos.

En C un registro se denomina estructura, y se expresa de la siguiente manera:

```
struct agenda {  
    char nombre[30];  
    char calle[40];  
    char ciudad[20];  
    char provincia[15];  
    char codigo[5];  
    char telefono[9];  
    int importe;  
};
```

Para acceder a cada uno de los elementos de un array , se declara una variable del tipo registro y se accede a cada elemento ( campo ) a través del operador .( punto)

```
struct agenda miregistro;  
dato=miregistro.nombre;
```

### 4.1.2 Clasificación de los registros

Según el numero de campos por registro y la longitud de los campos:

#### Registro de longitud fija

Todos los registros tienen el mismo número de campos.

#### Registro de longitud variable

No todos los registros tienen el mismo número de campos.

### 4.1.3 Operaciones con registros

#### Altas

Inserción de un registro en un fichero

#### Bajas

Eliminar un registro de un fichero

#### Modificaciones

Cambio total o parcial de uno o varios campos de un registro en un fichero

#### Consultas

Acceder a un registro de un fichero

#### Lectura

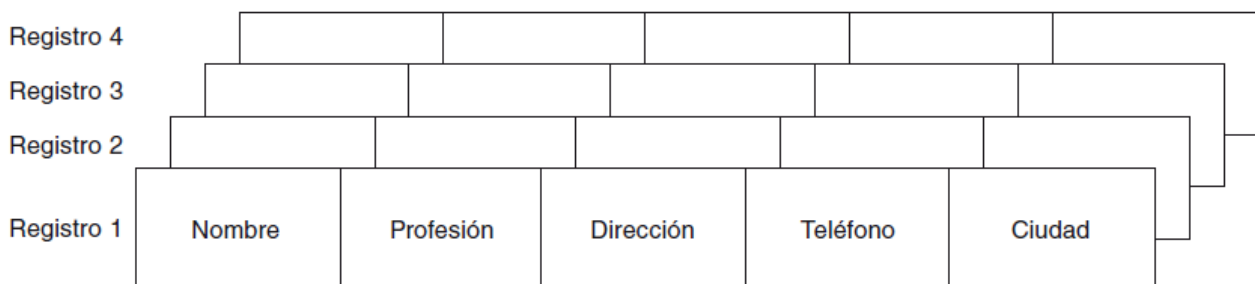
Leer un registro

#### Escritura

Escribir un registro

## 4.2 Ficheros o Archivos

Podemos definir un Fichero o Archivo como una colección de elementos denominados Registros ,que son del mismo tipo y están almacenados en memoria externa.[1]



Estructuras de un archivo "suscriptores".

Una clave (key) o indicativo es un campo de datos que identifica el registro y lo diferencia de otros registros. Esta clave debe ser diferente para cada registro. Claves típicas son nombres o números de identificación.

#### 4.2.1 Registro físico o bloque

Un registro físico o bloque es la cantidad más pequeña de datos que pueden transferirse en una operación de entrada/salida entre la memoria central y los dispositivos periféricos o viceversa. Ejemplos de registros físicos son: una tarjeta perforada, una línea de impresión, un sector de un disco magnético, etc.

Un bloque puede contener uno o más registros lógicos.

Un registro lógico puede ocupar menos de un registro físico, un registro físico o más de un registro físico.

#### 4.2.2 Factor de bloqueo

Otra característica que es importante en relación con los archivos es el concepto de factor de bloqueo o bloqueaje. El número de registros lógicos que puede contener un registro físico se denomina factor de bloqueo.

Se pueden dar las siguientes situaciones:

- Registro lógico > Registro físico. En un bloque se contienen varios registros físicos por bloque; se denominan registros expandidos.
- Registro lógico = Registro físico. El factor de bloqueo es 1 y se dice que los registros no están bloqueados.
- Registro lógico < Registro físico. El factor de bloqueo es mayor que 1 y los registros están bloqueados.

Son estudiados en profundidad en los temas 13 y 14 de este temario.

## 5 PLANTEAMIENTO DIDÁCTICO

Un tema como este y los siguiente de este temario oficial, son ideales para el módulos formativos donde los contenidos están basados en programación, tanto orientados a un lenguaje concreto o a pseudocódigo, tanto en programación estructurada o orientada a objetos, módulo como Programación de los ciclos formativos de Desarrollo de aplicaciones multiplataforma o Desarrollo de aplicaciones web; igualmente la materia de estudio de este tema, tiene perfecta cabida en la secundaria tanto la educación secundario obligatoria y en bachillerato, donde igualmente iniciemos a los alumnos y alumnas en el mundo de la Programación, en especial se puede pensar en la asignatura de Tecnologías de la Información y Comunicación II, de segundo de bachillerato.

## 6 CONCLUSIÓN

Los tipos de datos simples o primitivos significan que no están compuestos de otras estructuras de datos. Los más frecuentes y utilizados por casi todos los lenguajes son: enteros, reales, y caracteres, siendo los tipos lógicos o booleanos, propios de lenguajes estructurados. Los tipos de datos estructurados están constituidos por tipos de datos primitivos.

Los tipos de datos primitivos pueden ser organizados en diferentes estructuras de datos: estáticas y dinámicas. 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 dicho tamaño durante la ejecución del programa. Estas estructuras están implementadas en casi todos los lenguajes: arrays (vectores, tablas o matrices), registros, y archivos o ficheros. Las estructuras de datos dinámicas no tienen las limitaciones o restricciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas.

Una característica importante que diferencia a los tipos de datos es la siguiente: los tipos de datos primitivos tienen como característica común que cada variable representa a un elemento; los tipos de datos estructurados llenan como característica común que un identificador (nombre) puede representar a múltiples datos individuales, pudiendo cada uno de éstos ser referenciados independientemente.

## 7 BIBLIOGRAFÍA

- [1] *Fundamentos de Programación* ; Luis Joyanes ; Editorial McGraw Hill.
- [2] *Estructura de Datos y Algoritmos* ; Aho, Hopcroft ; Editorial Alhambra.
- [3] *Algoritmos+Estructuras de datos =Programa*; Niklaus Wirth ; Ediciones del Castillo.
- [4] *Estructura de Datos* ; O.Cairo ; Editorial McGraw Hill.
- [5] *Introducción a la Informática*; A. Prieto y Otros ; Editorial McGraw Hill.
- [6] *Lenguaje C, Manual de Referencia*, Herbert Schildt ; Editorial McGraw Hill.

Internet:

[www.xataka.es](http://www.xataka.es)

[www.gizmodo.es](http://www.gizmodo.es)

[www.microsiervos.es](http://www.microsiervos.es)