TEMA 11

ORGANIZACIÓN LÓGICA DE LOS DATOS: ESTRUCTURAS ESTÁTICAS

INDICE:

Realizado por Cayetano Borja Carrillo

Tiempo de escritura: 2 horas

1. INTRODUCCIÓN

En informática, un dato es una unidad de información u objeto manipulable por un ordenador. Los datos pueden ser de tipo primitivo si almacenan un único valor simple (entero, flotante, carácter, etc.) o estructurado si se representan mediante una estructura de datos que puede contener varios valores.

Una estructura de datos es una forma de organizar una colección de datos con el fin de facilitar su acceso y manipulación. Las estructuras pueden ser estáticas o dinámicas, dependiendo de si el tamaño que ocupan en memoria es fijo o variable.

En este tema se desarrollan los principales conceptos de las estructuras estáticas, que son aquellas donde su tamaño se define durante el desarrollo del *software* y no cambia durante la ejecución del programa. Se trata de un tema de gran importancia dentro del campo de estudio de la programación ya que, conocer cómo son estas estructuras, permite al desarrollador elegir la que mejor se adapte a sus necesidades.

2. ARRAY

Un *array* es una estructura de datos compuesta por un conjunto determinado de elementos del mismo tipo, es decir, todos enteros, todos flotantes, etc. A continuación, se describen las principales características de los *arrays*.

2.1. Índices de un *array*

Cada elemento del *array* tiene asociado una dirección única llamada índice, que determina su posición en el *array*. Existen 3 tipos de indexación cuyo uso dependerá del lenguaje de programación que se esté utilizando:

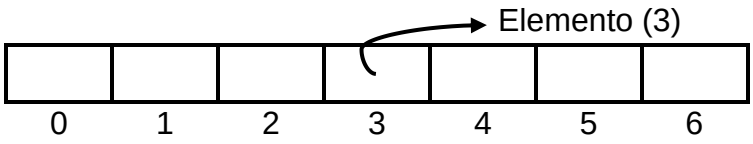
- Indexación base-cero: En un *array* de “n” elementos con indexación base-cero, el primer elemento tiene un índice de “0” y el último tiene un índice de “n-1“. Los lenguajes C, C++, PHP y Java utilizan este tipo de indexación.
- Indexación base-uno: En un *array* de “n” elementos con indexación base-uno, el primer elemento tiene un índice de “1” y el último de “n”. Los lenguajes Lua y Perl utilizan este tipo de indexación.
- Indexación base-n: En este caso, el índice del primer elemento puede ser elegido libremente. Algunos lenguajes permiten establecer números negativos o caracteres. Los lenguajes BASIC y JavaScript admiten esta indexación.

2.2. Dimensiones de un *array*

Las dimensiones de un *array* determinan el número de índices que se utilizan para acceder a cada elemento del *array*. Según sus dimensiones, un *array* puede ser unidimensional, bidimensional y multidimensional.

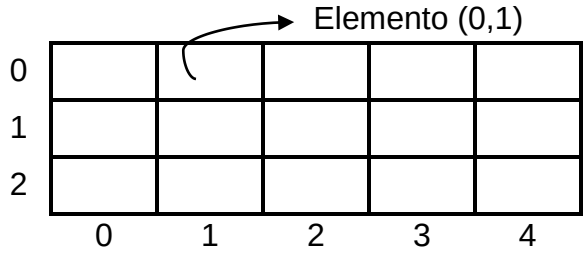
Array unidimensional

Un *array* unidimensional, también llamado vector, es aquel donde cada elemento del *array* es accedido por un único índice. En la siguiente ilustración se muestra un vector de 7 elementos con indexación base-cero.



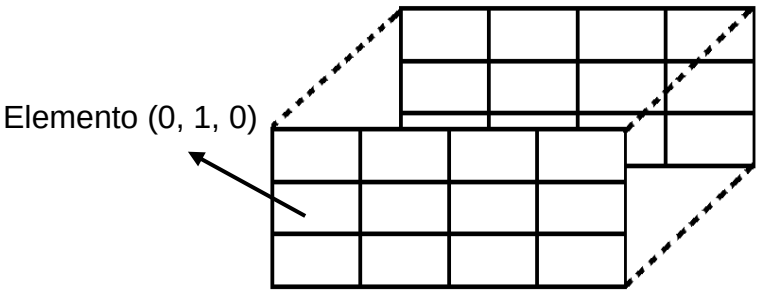
Array bidimensional

Un *array* bidimensional, también llamado matriz, es aquel donde cada elemento del *array* es accedido por un par de índices: uno para indicar la fila y otro para la columna. En la siguiente ilustración se muestra una matriz de 3x5 elementos.



Array multidimensional

Es aquel que tiene 3 o más dimensiones. Los elementos de un *array* de 3 dimensiones son accesibles mediante un conjunto de 3 índices; uno para indicar la fila, otro la columna y otro la página. En la siguiente ilustración se muestra un *array* de 3 dimensiones de 3x4x2 elementos.



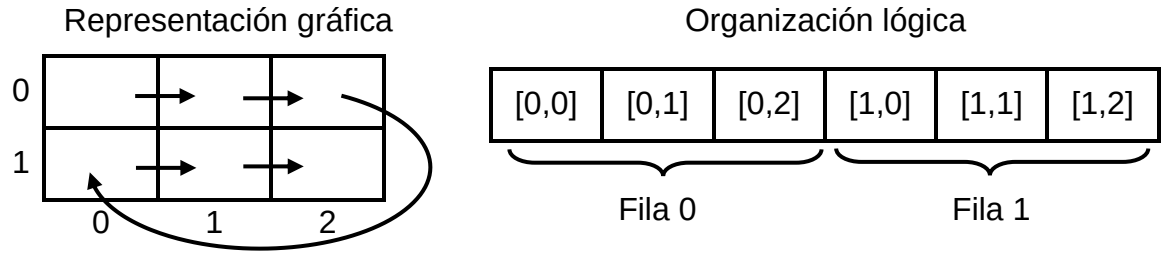
2.3. Organización lógica

Aunque de forma gráfica los *arrays* se representan mediante tablas, su representación interna es distinta, ya que los elementos de un *array* siempre se almacenan en memoria de forma lineal y secuencial, independientemente de sus dimensiones.

La organización lógica de un *array* puede realizarse de 2 maneras: por orden de fila mayor y por orden de columna mayor.

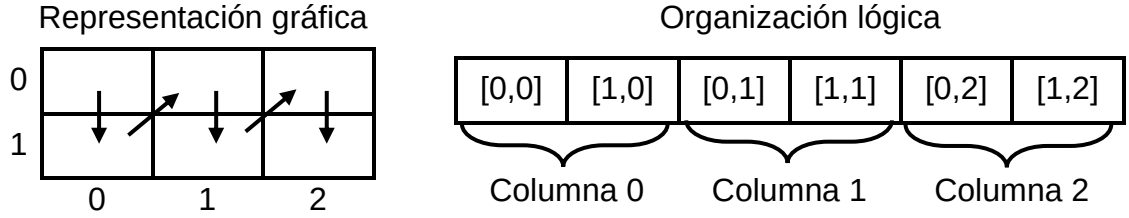
Por orden de fila mayor

Los elementos del *array* se almacenan por filas consecutivas donde, después del último elemento de una fila está el primer elemento de la siguiente fila. Ejemplo:



Por orden de columna mayor

Los elementos del *array* se almacenan por columnas consecutivas donde, después del último elemento de una columna está el primer elemento de la siguiente columna. Ejemplo:



Si el *array* tiene, por ejemplo, 3 dimensiones, primero se almacenan los elementos de una página usando una de las 2 ordenaciones anteriores y, a continuación, los elementos de la siguiente página.

2.4. Operaciones

A continuación, se describen las operaciones básicas que se pueden hacer con los *arrays*. Para todos los ejemplos se usa el lenguaje de programación C.

Declaración

Declarar una variable significa reservarle un espacio en memoria RAM. Para declarar un *array*, hay que especificar el nombre que se le quiera dar, la cantidad de elementos que contendrá y el tipo de datos. Ejemplo: Declarar un *array* llamado “vector” de 10 elementos de tipo entero.

```
int vector[10];
```

Asignación

Consiste en asignarle valores a los elementos del *array*. Ejemplo: Asignar el valor “73” al primer elemento del *array* anterior.

vector[0] = 73;

También es posible asignar los valores a la hora de declarar el *array*. Ejemplo:

int vector[ ] = {15, 23, 30, 7, 52};

Consulta

Los valores de un *array* pueden ser consultados en cualquier momento. Ejemplo: Mostrar por pantalla el valor que hay en el tercer elemento del *array* “vector”.

printf ("El tercer elemento es: %d", vector[2]);

2.5. Array asociativo

Un *array* asociativo, también llamado mapa, es un *array* cuyos índices pueden ser de cualquier tipo de datos como, por ejemplo, una cadena de texto. Este tipo de *array* especial no existe en C y solo es soportado por algunos lenguajes como PHP o Python.

A continuación, se muestran las operaciones básicas que se pueden hacer con un *array* asociativo. Los ejemplos están en el lenguaje PHP.

Declaración

Se declara la estructura. Ejemplo: Declarar el mapa “inventario” con los campos “tornillos”, “tuercas” y “clavos”.

```
$inventario = array(
        "tornillos" => 124,
        "tuercas" => 32,
        "clavos" => 21
);
```

Asignación

Asignar el valor “300” al elemento con índice “tornillos”.

$inventario["tornillos"] = 300;

Consulta

Mostrar por pantalla la cantidad de clavos que hay.

echo "Quedan " .$inventario["clavos"] . "clavos.";

3. CADENA

La estructura de datos cadena o *string* es un tipo de datos compuesto de una secuencia de caracteres alfanuméricos que se almacenan de forma contigua en memoria. En realidad, una cadena se trata de un *array* de caracteres.

3.1. Organización lógica

Las cadenas pueden representarse internamente de 2 maneras: de forma implícita y explícita. El uso de una forma u otra depende del lenguaje de programación utilizado.

- Organización implícita: Un carácter especial indica el fin de la cadena. En la mayoría de los lenguajes como C se usa el carácter nulo “\0”. Ejemplo de cómo se representa la cadena “Hola” en un *array* de 7 elementos en C.

'H'	'o'	'l'	'a'	\0	\0	\0

- Organización explícita: En este caso, el primer Byte de la cadena sirve para indicar la longitud del texto. Pascal usa este método. Ejemplo de cómo se representa la cadena “Hola” bajo esta organización:

4	'H'	'o'	'l'	'a'	''	''

3.2. Operaciones

A continuación, se describen las operaciones básicas que se pueden hacer con las cadenas. Para todos los ejemplos se utiliza el lenguaje de programación Java.

Declaración, asignación y consulta

Se realiza de forma similar a los *arrays*. Ejemplo: Declarar la cadena “saludo”, asígnale el valor “Hola” y mostrarlo por pantalla.

```
String saludo = "";
saludo = "Hola";
System.out.println(saludo);
```

Concatenación

Consiste en formar una cadena a partir de la unión de dos o más cadenas. Ejemplo: Declarar 2 cadenas llamadas “saludo” y “nombre” y concatenarlas en la cadena llamada “saludoCompleto”.

```
String saludo = "Hola";
String nombre = ", Fulano";
String saludoCompleto = saludo + nombre;
```

Extraer subcadena

Obtiene una parte de una cadena existente. Ejemplo: Extraer los 4 primeros caracteres de la cadena “saludoCompleto” y guardarlo en la cadena “auxiliar”.

String auxiliar = saludoCompleto.substring(0, 4);

Obtener longitud

Obtiene el número de caracteres que contiene la cadena. Ejemplo:

int longitud = cadena.length();

Comparación

Compara una cadena con otra cadena carácter a carácter. Ejemplo:

boolean esMenor = "abc" < "acb" //El valor es true

4. REGISTRO

Un registro es una estructura de datos que puede contener datos de distinto tipo. Por ejemplo, un registro puede estar compuesto de una cadena de texto de tamaño 20 y un entero.

cadena [20]	entero

4.1. Operaciones

A continuación, se indican las operaciones básicas que se pueden hacer con registros. En todos los ejemplos se usa el lenguaje de programación C.

Diseño

A diferencia del resto de estructuras estáticas, los registros no tienen una forma establecida y deben de ser diseñados por el programador. Ejemplo: Diseñar la estructura “Persona” cuyos campos sean una cadena de texto de longitud 20 llamada “nombre” y un entero llamado “edad”.

```
typedef struct Persona{
        char nombre[20];
        int edad;
}TipoPersona;
```

Declaración

Una vez se ha diseñado el registro, se pueden declarar variables del tipo de datos definido. Ejemplo: Declarar la variable “ciudadano” del tipo de datos “TipoPersona”.

TipoPersona ciudadano;

Asignación

Se asignan valores a los campos del registro. Ejemplo: Asignar los valores “Fulano” y “25” a los campos “nombre” y “edad” de la variable “ciudadano”.

```
ciudadano.nombre = “Fulano”;
ciudadano.edad = 25;
```

Consulta

La lectura se realiza de la misma manera que con cualquier tipo de datos. Ejemplo: Mostrar por pantalla el valor del campo “nombre” del registro “ciudadano”.

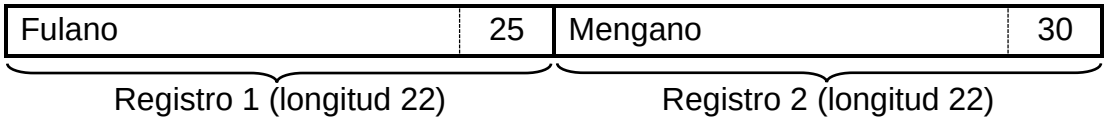
```
printf (“El nombre es: %s”, ciudadano.nombre);
```

4.2. Organización lógica y almacenamiento

Durante la ejecución de un programa, los registros creados a partir de estructuras estáticas se representan internamente en la memoria RAM de forma secuencial y con un tamaño fijo. Sin embargo, a la hora de guardarlos en archivos esto no tiene por qué ser así, pudiendo ser almacenados de las siguientes maneras:

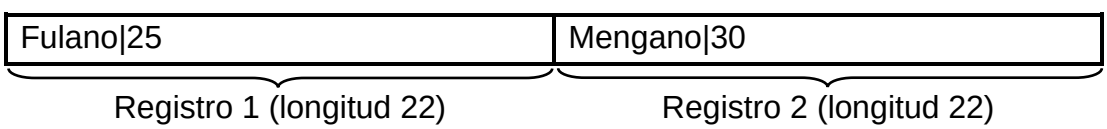
Registros de longitud fija con idéntico tamaño en cada campo

Los registros se almacenan en el archivo tal cual se han diseñado, es decir, si un campo de tipo cadena tiene asignado un tamaño de 20, ese campo ocupará en el archivo 20 caracteres, aunque se necesiten menos. Esto provoca desperdicio de espacio por fragmentación interna. Ejemplo de cómo quedarían almacenados en un archivo 2 registros del tipo “TipoPersona” de esta forma:



Registros de longitud fija con distinto tamaño en cada campo

En este caso, cada campo ocupará lo que necesite, pero la suma de todos los campos no puede sobrepasar la longitud del registro. Para saber cuándo termina un campo y empieza otro, se usa un carácter especial como “|”, que actuará de separador. Este carácter separador ocupa espacio adicional, pero permite jugar con el tamaño de los campos. Ejemplo de cómo quedarían almacenados en un archivo los 2 registros anteriores:



Registros de longitud variable

En este caso, el tamaño de cada registro puede variar y será la suma del tamaño de sus campos más los separadores.

```
Fulano|25|Mengano|30
```

5. ALGORITMOS SOBRE ESTRUCTURAS ESTÁTICAS

Las operaciones más comunes que se pueden realizar con las estructuras estáticas son las de búsqueda y ordenación. Para realizar estas operaciones existen diferentes algoritmos como los siguientes:

5.1. Algoritmos de búsqueda

Un algoritmo de búsqueda es aquel que está diseñado para localizar un elemento dentro de una secuencia de elementos. Los algoritmos de búsqueda que se utilizan son los siguientes:

- Búsqueda secuencial: Consiste en comprobar secuencialmente todos los elementos de la estructura desde el principio hasta encontrar el elemento deseado o hasta llegar al final en caso de que no exista.

- Búsqueda binaria: Consiste en comparar el elemento a buscar con el elemento central. Si no es el deseado, se comprueba si el elemento central es mayor o menor. Si es menor, se descarta la segunda mitad y se hará la misma operación sobre la primera mitad y, si es mayor, al contrario. Esta operación se repite recursivamente hasta llegar al elemento deseado o hasta que la secuencia no se pueda dividir más en caso de que el elemento no exista.

El algoritmo de búsqueda binaria es más veloz que el de búsqueda secuencial, pero sólo se puede aplicar si los elementos de la estructura son de longitud fija y están ordenados de menor a mayor.

5.2. Algoritmos de ordenación.

Un algoritmo de ordenación es aquel que permite ordenar los elementos de una estructura para poder realizar búsquedas binarias. Algunos algoritmos de ordenación son los siguientes:

- Ordenación por selección: Consiste en recorrer secuencialmente la estructura de principio a fin en busca del elemento de menor valor. Una vez localizado, se intercambia por el elemento que se encuentra en la primera posición. La operación se repite recursivamente a partir del siguiente elemento.

- Ordenación por inserción: Consiste en comparar cada elemento de la estructura con los elementos anteriores y realizar un intercambio en caso de

estar desordenada. Ejemplo: Se compara el segundo elemento con el primero y se intercambian si están desordenados. Después se compara el tercer elemento con los anteriores (el primero y el segundo) y se intercambian si es necesario. La operación se repite hasta revisar todos los elementos.

- Ordenación por intercambio o de burbuja: Consiste en comparar cada elemento con su colindante, intercambiándose mutuamente la posición si están en orden equivocado. Una vez se ha revisado la estructura completa, se repite la operación hasta que no se produzcan más intercambios.

6. CONCLUSIÓN

Durante el desarrollo de un *software*, el programador va declarando variables y diseñado estructuras capaces de almacenar información según los requerimientos del programa que está en desarrollo.

En este tema se han abordado las estructuras de datos estáticas, que son aquellas donde el tamaño que ocupan en memoria se define a la hora de declararlas y permanece fijo durante la ejecución del programa.

Existen varios tipos de estructuras estáticas como los *arrays*, que contienen elementos del mismo tipo, las cadenas que son *arrays* de caracteres y los registros que permiten almacenar datos de distinto tipo.

Conocer la organización lógica de cada una de estas estructuras y sus operaciones es fundamental para desarrollar un *software* de calidad, eficiente y sin errores.

7. BIBLIOGRAFÍA

- López Ureña, L. A. et al. (1997). *Fundamentos de Informática (1ª ed.)*. Ra-ma.
- Prieto Espinosa, A. et al. (2006). *Introducción a la informática (4ª ed.)*. McGraw-Hill.
- Brookshear, J. G. (2012). *Introducción a la computación (11ª ed.)*. Pearson Educación.
- Joyanes Aguilar, L. (2020). *Fundamentos de programación (5ª ed.)*. McGraw-Hill

8. NORMATIVA

Para el desarrollo de este tema, se ha tenido en cuenta la siguiente normativa, donde se especifican los contenidos, competencias y criterios de evaluación de los Ciclos Formativos y Bachillerato en Andalucía:

- Orden 16 de junio de 2011 (DAW/DAM). La parte correspondiente a los módulos “Entornos de Desarrollo” y “Programación”.
- Instrucción 13/2022 (Bachillerato). La parte correspondiente a la asignatura “Tecnologías de la Información y Comunicación”