GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, COMERCIO Y TURISMO
EOi Escuela de organización industrial

CURSO DE PROGRAMACIÓN.NET

M.374.001.003

ARRAYS

Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

Ayuntamiento de Málaga

polo de contenidos digitales

Index

Arrays

Un **vector** o **array** es una estructura de datos capaz de almacenar una colección de valores del mismo tipo, a diferencia de una variable con un tipo de datos básico que sólo almacena un único valor.

Para declarar una variable de tipo array, simplemente tendremos que indicar el tipo de datos que almacenará dicho array (int, double, string, ...) seguido de dos corchetes vacíos.

```
int[] numeros;
```

Antes de empezar a guardar valores en un array, debemos inicializar sus posiciones (reservar memoria). Para ello usaremos la instrucción **new** seguida del tipo del array, pero esta vez indicaremos entre corchetes cuantas posiciones vamos a reservar.

```
int[] numeros = new int[5];
```

Aunque ya podemos utilizar el array y acceder a sus posiciones, no tenemos valores guardados. Para guardar o acceder a los valores almacenados, debemos indicar después del nombre de la variable, entre corchetes, la posición a la cual queremos acceder. Esta posición podrá ser un número del cero al número de posiciones del array menos una (0 → N-1). En caso de intentar acceder a una posición no válida el programa lanzará un error y terminará.

```
numeros[0] = 12;
numeros[1] = 23;
numeros[2] = 53;
numeros[3] = 5;
numeros[4] = 92;
```

Índice	0	1	2	3	4
Valor	12	23	53	5	92

Si conocemos de antemano los valores iniciales del array, podemos hacer una asignación de los mismos al declarar el array. Para ello indicamos los valores entre llaves separados por coma.

```
int[] numeros = {12, 23, 53, 5, 92};
```

En este caso estamos creando un array con 5 posiciones (0-4). Cuando indicamos los valores iniciales, se reservan únicamente las posiciones a las que hemos dado valor.

Recorrer arrays

Para recorrer un array, debemos saber la posición del último elemento (si el array no está lleno), o en el caso de querer recorrer el array completo, consultar su

propiedad **Length** (nos devuelve el tamaño del mismo). La primera posición siempre será la cero.

Una vez sepamos el rango a recorrer, se puede usar cualquier tipo de bucle para recorrerlo, aunque los más recomendados son **for** y **foreach**.

```
for (int i = 0; i < numeros.Length; i++)
{
    Console.WriteLine($"Índice {i}: {numeros[i]}");
}
```

```
Índice 0: 12
Índice 1: 23
Índice 2: 53
Índice 3: 5
Índice 4: 92
```

```
foreach (int num in numeros)
{
    Console.WriteLine(num);
}
```

```
12
23
53
5
92
```

Operaciones básicas con arrays

Buscar un elemento

Para comprobar si un elemento existe en un array, podemos recorrerlo e ir comprobando cada una de sus posiciones a ver si coincide con el valor que estamos buscando.

```
string[] nombres = { "Juan", "Ana", "Pedro", "Eva", "Paco" };
bool encontrado = false; // Nos indicará si hemos encontrado el valor
string buscar = "Pedro"; // Valor a buscar en el array


// La condición también incluye que no hayamos encontrado lo que buscamos

for (int i = 0; i < nombres.Length && !encontrado; i++)
{
    if (nombres[i] == buscar)
    {
        encontrado = true;
    }
}

if (encontrado)
{
    Console.WriteLine($"El nombre {buscar} está en el array");
}
else
{
    Console.WriteLine($"{buscar} no encontrado...");
```

```
}
```

En el futuro veremos opciones más óptimas como esta:

```
if (Array.Exists(nombres, n => n == buscar))
{
    Console.WriteLine($"El nombre {buscar} está en el array");
}
else
{
    Console.WriteLine($"{buscar} no encontrado...");
}
```

Redimensionar el array

Si necesitamos más posiciones en un array y necesitamos redimensionarlo, la única opción es crear un nuevo array auxiliar con las posiciones necesarias, copiar el contenido del array existente en el nuevo, y finalmente, reasignar el array auxiliar (con los valores copiados y más posiciones disponibles) a la variable original.

```
string[] nombres = { "Juan", "Ana", "Pedro", "Eva", "Paco" };
string[] nombresAux = new string[10];

for (int i = 0; i < nombres.Length; i++)
{
    nombresAux[i] = nombres[i];
}

nombres = nombresAux; // La variable nombres apunta al array redimensionado
nombres[5] = "Marta";
```

Existe ya una función de C# que hace lo mismo internamente, por lo que si la utilizamos nos queda un código más compacto.

```
string[] nombres = { "Juan", "Ana", "Pedro", "Eva", "Paco" };

Array.Resize(ref nombres, 10);

nombres[5] = "Marta";
System.Console.WriteLine(nombres.Length); // 10
```

Borrar un elemento

Para borrar un elemento del array y no dejar un hueco vacío, se suelen mover los elementos de su derecha una posición a la izquierda. Ahora bien, tenemos 2 posibles formas de gestionar el tamaño del array (posiciones que contienen un valor válido).

Además del tamaño del array (que no cambiará), debemos llevar la cuenta (en otra variable) del número de elementos almacenados en el array. Esta última variable será la que se use para saber hasta dónde hay que recorrer el array.

```
const int MAX_ITEMS = 10;
int[] nums = new int[MAX_ITEMS];
int guardados = 0;
```

```
nums[guardados++] = 15;
nums[guardados++] = 6;
nums[guardados++] = 9;
nums[guardados++] = 12;
nums[guardados++] = 20;
Console.WriteLine($"Números guardados: {guardados}"); // Números guardados: 5


for (int i = 0; i < guardados; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 15 6 9 12 20


// Ahora vamos a borrar la posición 2 (número 9)

for (int  i = 2; i < guardados - 1; i++)
{
    nums[i] = nums[i + 1]; // Desplazamos los números a la izquierda

}
guardados--; // Decrementamos el número de elementos guardados


for (int i = 0; i < guardados; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 15 6 12 20 (posición borrada con éxito)
```

Redimensionar el array cada vez que hagamos una operación de borrado o inserción. Así siempre podremos consultar la propiedad Length para saber el número de elementos almacenados.

```
int[] nums = {15, 6, 9, 12, 20};

Console.WriteLine($"Números guardados: {nums.Length}"); // Números guardados: 5


// Ahora vamos a borrar la posición 2 (número 9)

for (int  i = 2; i < nums.Length - 1; i++)
{
    nums[i] = nums[i + 1]; // Desplazamos los números a la izquierda

}
Array.Resize(ref nums, nums.Length - 1); // Redimensionamos una posición menos


for (int i = 0; i < nums.Length; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 15 6 12 20 (posición borrada con éxito)
```

Añadir un elemento en cualquier posición

Si añadimos cualquier elemento en un array que no esté después de la última posición asignada, debemos hacer la operación contraria al borrado. Es decir, desplazar los elementos a partir de la posición donde se realizará la inserción, una posición más a la derecha.

Al igual que antes, podríamos utilizar un array de longitud fija y usar una variable que nos indique el número de elementos guardados o ir modificando la longitud del array en función de los elementos a almacenar.

```
const int MAX_ITEMS = 10;
int[] nums = new int[MAX_ITEMS];
int guardados = 0;

nums[guardados++] = 15;
nums[guardados++] = 6;
nums[guardados++] = 9;
nums[guardados++] = 12;
nums[guardados++] = 20;
Console.WriteLine($"Números guardados: {guardados}"); // Números guardados: 5


// Vamos a insertar el número 2 en la posición 3 (donde está el 12)
if ( guardados < MAX_ITEMS ) // Comprobamos que cabe

{
    for (int i = guardados - 1; i >= 3; i--) // Recorremos al revés
    {
        nums[i + 1] = nums[i]; // Desplazamos los números a la derecha

    }
    nums[3] = 2; // Insertamos el número 2 en la posición 3
    guardados++;
}

for (int i = 0; i < guardados; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 15 6 9 2 12 20 (posición insertada con éxito)
```

Ahora adaptando la longitud del array al número de elementos.

```
int[] nums = {15, 6, 9, 12, 20};

// Ahora vamos a insertar el número 2 en la posición 3
Array.Resize(ref nums, nums.Length + 1); // Redimensionamos array

for (int  i = nums.Length - 2; i >= 3; i--)
{
    nums[i + 1] = nums[i]; // Desplazamos los números a la derecha

}
nums[3] = 2; // Añadimos el 2 en la posición 3


for (int i = 0; i < nums.Length; i++)
{
    Console.Write(nums[i] + " ");
```

```
}
Console.WriteLine(); // 15 6 9 2 12 20 (posición insertada con éxito)
```

Ordenar arrays

Hay varios algoritmos para ordenar un array. Uno de los más simples aunque no es el más eficiente en cuanto a rendimiento es el algoritmo de la burbuja. Básicamente consiste en los siguientes pasos:

- Recorremos todas las posiciones del array (índice **i**).
- En cada iteración, a partir de la siguiente posición en la que estamos ahora (i+1), recorremos lo que queda del array (índice j). Cada vez que nos encontremos con un número menor que el de la posición i, los intercambiamos.
- El objetivo es asegurarse que en el índice i (bucle exterior) siempre almacenamos el menor de todos los elementos que hay entre los restantes.

```
int[] nums = { 15, 6, 9, 12, 20 };

for (int i = 0; i < nums.Length; i++)
{
    for (int j = i + 1; j < nums.Length; j++)
    {
        if (nums[j] < nums[i]) // Intercambiamos
        {
            int aux = nums[i];
            nums[i] = nums[j];
            nums[j] = aux;
        }
    }
}

for (int i = 0; i < nums.Length; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 6 9 12 15 20 (Ordenados)
```

El lenguaje C# tiene funciones integradas para ordenar arrays. En este caso podríamos usar la función Array.Sort.

```
int[] nums = { 15, 6, 9, 12, 20 };

Array.Sort(nums);

for (int i = 0; i < nums.Length; i++)
{
    Console.Write(nums[i] + " ");
}
Console.WriteLine(); // 6 9 12 15 20 (Ordenados)
```

Arrays multidimensionales

Un array puede tener más de un índice. En este caso, podemos definirlo como un array multidimensional o un array escalonado. La principal diferencia es que en el primero se definen los tamaños de la dimensiones al principio, mientras que el

segundo puede tener un número variable de elementos en la segunda, tercera,... (o posteriores) dimensión/es en función de la posición a la que accedamos.

Array multidimensional

Al declarar la variable debemos indicar cuantas dimensiones queremos añadiendo comas entre los corchetes (0 comas → 1 dimensión, 1 coma → 2 dimensiones, ...). Al inicializarlo indicaremos cuantas posiciones debe reservar en cada dimensión.

```
int[,] array2D = new int[4,3];
```

Si conocemos los valores iniciales se pueden usar en la inicialización:

```
// array de 4 x 3  (4 filas x 3 columnas)
int[,] array2D = {{12, 3, 4}, {22, 36, 34}, {23, 65, 75}, {1, 4, 7}};
```

fila/col	1 (pos 0)	2 (pos 1)	3 (pos 2)
1 (pos 0)	12	3	4
2 (pos1)	22	36	34
3 (pos 2)	23	65	75
4 (pos 3)	1	4	7

Azul: dimensión 1, verde: dimensión 2 pos: posición

Para acceder a un valor para leerlo o modificarlo, ahora se deben indicar ambas dimensiones (empezando por cero siempre):

```
Console.WriteLine(array2D[2,1]); // 65
```

Existe una propiedad llamada **Rank** que nos indica cuantas dimensiones tiene un array:

```
Console.WriteLine($"Dimensiones: {array2D.Rank}"); // Dimensiones: 2
```

Es complicado recorrer este tipo de arrays usando directamente la propiedad Length, ya que nos devolverá el número de elementos total. Es decir, el tamaño de la primera dimensión multiplicado por el tamaño de la segunda (y así sucesivamente). Sin embargo, existe la función GetLength(dimensión) que nos puede decir cuantos elementos tiene cada dimensión por separado.

```
for (int i = 0; i < array2D.GetLength(0); i++)
{
    for (int j = 0; j < array2D.GetLength(1); j++)
    {
        Console.Write(array2D[i, j] + " ");
    }
    Console.WriteLine();
}
```

```
12 3 4
22 36 34
23 65 75
1 4 7
```

Array de arrays (array escalonado)

Otro tipo de arrays multidimensionales son los arrays escalonados. En este caso se puede considerar que estamos ante **un array de arrays**. La forma de indicar las dimensiones en este caso sería poner tantos corchetes vacíos como dimensiones queramos.

```
int[][] array2D;
```

En este tipo de arrays no podemos inicializar todas sus dimensiones al mismo tiempo, sino que tenemos que inicializar la primera dimensión y posteriormente en cada posición de dicha dimensión, inicializar otro array (2ª dimensión).

```
int[][] array2D = new int[4][]; // Inicializamos la primera dimensión

for(int i = 0; i < array2D.Length; i++)
{
    array2D[i] = new int[3]; // Inicializamos la segunda dimensión

}
```

La inicialización no es tan directa como en un array multidimensional, ya que seguimos necesitando inicializar los arrays de la segunda dimensión uno a uno. Podemos inicializar el array con todos su valores de esta forma (como hacíamos con los arrays unidimensionales (simples):

```
int[][] array2D = {
    new int[] {12, 3, 4},
    new int[] {22, 36, 34},
    new int[] {23, 65, 75},
    new int[] {1, 4, 7}
};
```

Como en este caso, lo que almacenamos dentro de la variable es un array que contiene otros arrays dentro, podríamos tener tamaños diferentes en cada posición:

```
int[][] array2D = {
    new int[] {12, 3},
    new int[] {22, 36, 34, 23},
    new int[] {23},
    new int[] {1, 4, 7, 12, 3}
};
```

Para recorrer un array escalonado, recorremos la primera dimensión y después cada array interno con otro índice:

```
int[][] array2D = {
    new int[] {12, 3, 4},
    new int[] {22, 36, 34},
    new int[] {23, 65, 75},
    new int[] {1, 4, 7},
};

for (int i = 0; i < array2D.Length; i++)        // array2D.Length (longitud de la 1ª dimensión)
{
    for (int j = 0; j < array2D[i].Length; j++)// array2D[i].Length (longitud de la 2ªdimensión)
```

```
    {
        Console.Write(array2D[i][j] + " ");
    }
    Console.WriteLine();
}
```

```
12 3 4
22 36 34
23 65 75
1 4 7
```

Se pueden hacer combinaciones más complejas como arrays escalonados multidimensionales pero no es muy habitual utilizarlos, además de que aumenta mucho la complejidad al gestionarlos.

Recorrer los caracteres de una cadena

Las cadenas pueden funcionar como un array de caracteres, por lo que podemos usar los corchetes para acceder a cada carácter por separado (no se pueden modificar, solo leer las posiciones).

```
string cadena = "Hola que tal";
Console.WriteLine($"Cuarta letra: {cadena[3]}"); // Cuarta letra: a

for (int i = 0; i < cadena.Length; i++) {
    Console.WriteLine($"[{i}]: {cadena[i]}");
}
```

```
[0]: H
[1]: o
[2]: l
[3]: a
[4]:
[5]: q
[6]: u
[7]: e
[8]:
[9]: t
[10]: a
[11]: l
```

Parámetros de entrada a un programa

A estas alturas te habrás dado cuenta que en el punto de entrada del programa (método Main) hay un array llamado **args**. Todavía no hemos visto el concepto de función pero digamos que en ese array podríamos encontrar valores que se envían justo en el momento de ejecutar el programa.

Para enviar **valores de entrada** a un programa podemos hacerlo de 3 formas:

- Si ejecutamos el programa con el comando **dotnet run**, justo después, **separados por espacios**, podemos enviar una lista de valores.
    - Si necesitamos pasar un valor que contenga espacios, agruparíamos dicho valor entre comillas dobles.

```
dotnet run 23 43 65
dotnet run "Alberto Perez" "Juana Monllor"
```

- Para ejecutar con **F5** o **Ctrl + F5** (sin depurar), si abrimos el archivo .vscode/launch.json (o desde el menú Depurar → Abrir configuraciones), editamos la propiedad "args". Entre los corchetes pondremos los valores separados por comas y entre comillas (es un array en formato JSON).

```
"args": ["Pedro", "Juan", "Marta"],
```

- Si tenemos la tarea de ejecución (run) creada y ejecutamos el programa usando **Terminal → Ejecutar tarea**, o su correspondiente atajo de teclado, habrá que editar el archivo **.vscode/tasks.json**. En la tarea run, añadimos al comando los parámetros a continuación (separados por espacio).

```
{
   "label": "run",
   "type": "shell",
   "command": "dotnet run Pepe Paco Ana",
   "problemMatcher": "$tsc",
},
```

    - Si tenemos algún parámetro que contenga espacios, podemos poner la contrabarra '\' delante del espacio o encerrar el parámetro entre comillas. Como el comando va a su vez encerrado entre comillas dobles, deberemos *escaparlas* (poner la contrabarra delante).

```
"command": "dotnet run \"Pepe Pérez\" \"Paco Martínez\" \"Ana García\"",
```

Recorrer los parámetros de entrada

Los parámetros de entrada se recorren como un array normal. Por ejemplo, si ejecutamos el comando **dotnet run manzana pera plátano**:

```
static void Main(string[] args)
{
    Console.WriteLine($"He recibido {args.Length} parámetros: ");
    foreach (string param in args)
    {
        Console.WriteLine(param);
    }
}

He recibido 3 parámetros:
manzana
pera
plátano
```