

INTRODUCCION A LA PROGRAMACION C#

2 - MÓDULO 2 : ARREGLOS Y MATRICES

UNIDAD: 2

MODULO: 2

PRESENTACIÓN: En esta unidad se explicarán los conceptos de arreglos y matrices en el lenguaje C#.

OBJETIVOS

Que los participantes logren: Comprender los conceptos de esta unidad y aplicar en la practica dichos conceptos.

TEMARIO

Arreglos.....	4
Arreglos multidimensionales o Matrices	7
Matrices irregulares (Tablas dentadas o Jagged Arrays)	9

Arreglos

Un arreglo es un tipo especial de variable que es capaz de almacenar en su interior y de manera ordenada uno o varios datos de un determinado tipo.

Para declarar variables de este tipo especial se usa la siguiente sintaxis:

```
<tipoDatos>[] <nombreArray>;
```

Por ejemplo, una tabla que pueda almacenar objetos de tipo *int* se declara así:

```
int[] tabla;
```

Con esto la tabla creada no almacenaría ningún objeto, sino que valdría *null* . Si se desea que verdaderamente almacene objetos hay que indicar cuál es el número de objetos que podrá almacenar, lo que puede hacerse usando la siguiente sintaxis al declararla:

```
<tipoDatos>[] <nombreTabla> = new <tipoDatos>[<númeroDatos>];
```

Por ejemplo, una tabla que pueda almacenar 100 objetos de tipo *int* se declara así:

```
int[] tabla = new int[100];
```

Aunque también sería posible definir el tamaño de la tabla de forma separada a su declaración de este modo:

```
int[] tabla;  
tabla = new int[100];
```

Con esta última sintaxis es posible cambiar dinámicamente el número de elementos de una variable tabla sin más que irle asignando nuevas tablas.

Ello no significa que una tabla se pueda redimensionar conservando los elementos que tuviese antes del cambio de tamaño, sino que ocurre todo lo contrario: cuando a una variable tabla se le asigna una tabla de otro tamaño, sus elementos antiguos son **sobrescritos** por los nuevos.

Si se crea una tabla con la sintaxis hasta ahora explicada todos sus elementos tendrían el valor por defecto de su tipo de dato. Si queremos darles otros valores al declarar la tabla, hemos de indicarlo entre llaves usando esta sintaxis:

```
<tipoDatos>[] <nombreTabla> = new <tipoDatos>[] {<valores>;
```

Ha de especificarse tantos *<valores>* como número de elementos se desee que tenga la tabla, y si son más de uno se han de separar entre sí mediante comas (,) Nótese que ahora no es necesario indicar el número de elementos de la tabla (aunque puede hacerse) si se desea, sí mediante comas (,) Nótese que ahora no es necesario indicar el número de elementos de la tabla (aunque puede hacerse) si se desea).

Ejemplo:

```
int [] misEnteros = new int[] {10, 2, 45, 23, 29};
```

Nótese que ahora no es necesario indicar el número de elementos de la tabla (aunque puede hacerse) si se desea, pues el compilador puede deducirlo del número de valores especificados.

Incluso se puede compactar aún más la sintaxis declarando la tabla así:

```
int[] tabla = {10, 2, 45, 23, 29};
```

También podemos crear tablas cuyo tamaño se pueda establecer dinámicamente a partir del valor de cualquier expresión que produzca un valor de tipo entero. Por ejemplo, para crear una tabla cuyo tamaño sea el valor indicado por una variable de tipo *int* (luego su valor será de tipo entero) se haría:

```
int i = 5;  
int[] tablaDinámica = new int[i];
```

A la hora de acceder a los elementos almacenados en una tabla basta indicar entre corchetes, y a continuación de la referencia a la misma, la posición que ocupe en la tabla el elemento al que acceder.

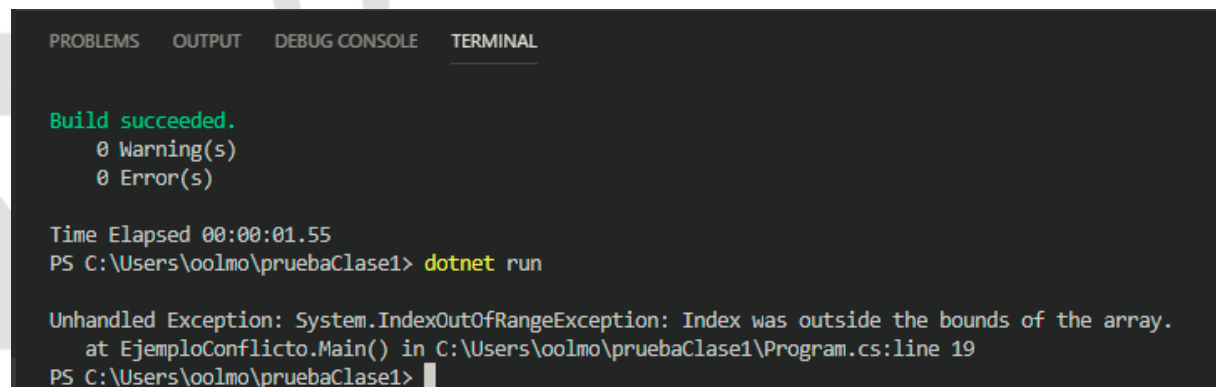
Cuando se haga hay que tener en cuenta que **en C# las tablas se indexan desde 0**, lo que significa que el primer elemento de la tabla ocupará su posición 0, el segundo ocupará la posición 1, y así sucesivamente para el resto de elementos. Por ejemplo, aunque es más ineficiente, la tabla declarada en el último fragmento de código de ejemplo también podría haberse definido así:

```
int[] tabla = new int[4];
tabla[0] = 5;
tabla[1]++; // Por defecto se inicializó a 0, luego ahora el valor de tabla[1]
pasa a ser 1
tabla[2] = tabla[0] - tabla[1] ; // tabla[2] pasa a valer 4, pues 5-4 = 1

// El contenido de la tabla será {5,1,4,0}, pues tabla[3] se inicializó por
defecto a 0.
```

Hay que tener cuidado a la hora de acceder a los elementos de una tabla ya que si se especifica una posición superior al número de elementos que pueda almacenar la tabla se producirá una excepción de tipo *System.OutOfBoundsException*.

```
System.Console.WriteLine(tabla[6]);
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Build succeeded.
0 Warning(s)
0 Error(s)

Time Elapsed 00:00:01.55
PS C:\Users\oolmo\pruebaClase1> dotnet run

Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of the array.
at EjemploConflicto.Main() in C:\Users\oolmo\pruebaClase1\Program.cs:line 19
PS C:\Users\oolmo\pruebaClase1> █

Por ahora basta considerar que **las excepciones son objetos que informan de situaciones no esperadas** (generalmente errores) producidas durante la ejecución de una aplicación.

Para evitar este tipo de excepciones puede consultar el valor del campo de sólo lectura *Length* que está asociado a toda tabla y contiene el número de elementos de la misma. Por ejemplo, para asignar un 7 al último elemento de la tabla anterior se haría:

```
tabla[tbl.Length - 1] = 7;
// Se resta 1 porque tabla.Length devuelve 4 pero el último elemento de la
tabla es tabla[3]
```

Arreglos multidimensionales o Matrices

Una matriz es una tabla cuyos elementos se encuentran organizando una estructura de varias dimensiones. Para definir este tipo de tablas se usa una sintaxis similar a la usada para declarar tablas unidimensionales pero separando las diferentes dimensiones mediante comas (,).

Por ejemplo, una matriz de elementos de tipo *int* que conste de 12 elementos puede tener sus elementos distribuidos en dos dimensiones formando una estructura 3x4 similar a una matriz de la forma:

1	2	3	4
5	6	7	8
9	10	11	12

Esta matriz se podría declarar así:

```
int[,] matriz = new int[3,4] {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

En realidad no es necesario indicar el número de elementos de cada dimensión de la tabla ya que pueden deducirse de los valores explícitamente indicados entre llaves, por lo que la definición anterior es similar a esta:

```
int[,] matriz = new int[,] {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

Incluso puede reducirse aún más la sintaxis necesaria quedando tan sólo:

```
int[,] matriz = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

Si no queremos indicar explícitamente los elementos de la tabla al declararla, podemos obviarlos pero aún así indicar el tamaño de cada dimensión de la tabla (a los elementos se les daría el valor por defecto de su tipo de dato) así:

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

También podemos no especificar ni siquiera el número de elementos de la tabla de esta forma (matriz contendría ahora *null*):

```
int[,] matriz;
```

Aunque los ejemplos de tablas multidimensionales hasta ahora mostrados son de tablas de dos dimensiones, en general también es posible crear tablas de cualquier número de dimensiones. Por ejemplo, una tabla que almacene 24 elementos de tipo *int* y valor 0 en una estructura tridimensional 3x4x2 se declararía así:

```
int[, ,] matriz = new int[3,4,2];
```

El acceso a los elementos de una tabla multidimensional es muy sencillo: sólo hay que indicar los índices de la posición que ocupe en la estructura multidimensional el elemento al que se desee acceder. Por ejemplo, para incrementar en una unidad el elemento que ocupe la posición (1,3,2) de la tabla anterior se haría (se indexa desde 0):

```
matriz [0,2,1]++;
```


Matrices irregulares (Tablas dentadas o Jagged Arrays)

Una tabla dentada no es más que una tabla cuyos elementos son a su vez tablas, pudiéndose así anidar cualquier número de tablas. Para declarar tablas de este tipo se usa una sintaxis muy similar a la explicada para las tablas unidimensionales solo que ahora se indican tantos corchetes como nivel de anidación se desee.

Por ejemplo, para crear una tabla de tablas de elementos de tipo *int* formada por dos elementos, uno de los cuales fuese una tabla de elementos de tipo *int* formada por los elementos de valores 1,2 y el otro fuese una tabla de elementos de tipo *int* y valores 3,4,5, se puede hacer:

```
int[][] tablaDentada = new int[2][] {new int[] {1,2}, new int[] {3,4,5}};
```

Como se indica explícitamente cuáles son los elementos de la tabla declarada no hace falta indicar el tamaño de la tabla, por lo que la declaración anterior es equivalente a:

```
int[][] tablaDentada = new int[][] {new int[] {1,2}, new int[] {3,4,5}};
```

Es más, igual que como se vio con las tablas unidimensionales también es válido hacer:

```
int[][] tablaDentada = {new int[] {1,2}, new int[] {3,4,5}};
```

Si no quisiésemos indicar cuáles son los elementos de las tablas componentes, entonces tendríamos que indicar al menos cuál es el número de elementos que podrán almacenar (se inicializarán con valores por defecto) quedando:

```
int[][] tablaDentada = {new int[2], new int[3]};
```

Si no queremos crear las tablas componentes en el momento de crear la tabla dentada, entonces tendremos que indicar por lo menos cuál es el número de tablas componentes posibles (cada una valdría *null*), con lo que quedaría:

```
int[][] tablaDentada = {new int[2], new int[3]};
```

Es importante señalar que no es posible especificar todas las dimensiones de una tabla dentada en su definición si no se indica explícitamente el valor inicial de éstas entre llaves. Es decir, esta declaración es incorrecta:

```
int[][] tablaDentada = new int[2][5];
```

Esto se debe a que el tamaño de cada tabla componente puede ser distinto y con la sintaxis anterior no se puede decir cuál es el tamaño de cada una.

Una opción hubiese sido considerar que es 5 para todas como se hace en Java, pero ello no se ha implementado en C# y habría que declarar la tabla de, por ejemplo, esta manera:

```
int[][] tablaDentada = {new int[5], new int[5]};
```

Finalmente, si sólo queremos declarar una variable tabla dentada pero no queremos indicar su número de el

```
int[][] tablaDentada = {new int[5], new int[5]};
```

Finalmente, si sólo queremos declarar una variable tabla dentada pero no queremos indicar su número de elementos, (luego la variable valdría null), entonces basta poner:

```
int[][] tablaDentada;
```

Hay que precisar que aunque en los ejemplos hasta ahora presentes se han escrito ejemplos basados en tablas dentadas de sólo dos niveles de anidación, también es posible crear tablas dentadas de cualquier número de niveles de anidación. Por ejemplo, para una tabla de tablas de tablas de enteros de 2 elementos en la que el primero fuese una tabla dentada formada por dos tablas de 5 enteros y el segundo elemento fuese una tabla dentada formada por una tabla de 4 enteros y otra de 3 se podría definir así:

```
int[][][] tablaDentada = new int[][][] { new int[][] {new int[5], new int[5]}, new int[][] {new int[4], new int[3]}};
```

A la hora de acceder a los elementos de una tabla dentada lo único que hay que hacer es indicar entre corchetes cuál es el elemento exacto de las tablas componentes al que se desea acceder, indicándose un elemento de cada nivel de anidación entre unos corchetes diferentes pero colocándose todas las parejas de corchetes juntas y ordenadas de la tabla más externa a la más interna.

Por ejemplo, para asignar el valor 10 al elemento cuarto de la tabla que es elemento primero de la tabla que es elemento segundo de la tabla dentada declarada en último lugar se haría:

```
tablaDentada[1][0][3] = 10;
```

BIBLIOGRAFÍA RECOMENDADA:

<https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/types/index>

Head First C# 3rd Edition Jennifer Greene Andrew Stellman



Fecha	Versión	Observaciones
09/07/2018	1.0	Versión Original
11/07/2018	1.1	Se corrige typo en pagina 23