

# Tablas

## Contenido

Descripción general	1
Introducción a las tablas	2
Creación de tablas	10
Uso de tablas	18

## Notas para el instructor

Este módulo explica cómo declarar y usar tablas (arrays) de distintos rangos en C#. El módulo se limita intencionadamente a tablas rectangulares y no incluye tablas dentadas.

```
int[ ] fila;           // Incluida
int[,] cuadrícula;    // Incluida
int[, ,] cubo;        // Incluida

int[ ][ ] cuadrícula; // No se incluyen porque no considera
                      // que cumplan las normas del CLS
int[ ][ ][ ] cubo1;   // No se incluyen porque no considera
                      // que cumplan las normas del CLS
int[ ][ ,] cubo2;     // No se incluyen porque no considera
                      // que cumplan las normas del CLS
```

El módulo comienza explicando los conceptos básicos de tablas, incluyendo notación, variables y rango. Muestra la sintaxis para declarar una variable de tabla, para acceder a un elemento de una tabla y para comprobar límites de tablas. La sintaxis para crear una tabla se tratará en la siguiente sección.

La segunda sección describe la creación e inicialización de tablas. En la tercera sección se explica cómo usar propiedades y métodos de tabla, cómo pasar tablas como parámetros, cómo devolver tablas desde métodos, cómo usar argumentos de línea de comandos para **Main** y cómo usar instrucciones **foreach** en tablas.

En el Ejercicio 1, los estudiantes escribirán un programa que espera el nombre de un archivo de texto como argumento para **Main** y luego lee los contenidos de ese archivo en una tabla de caracteres. A continuación recorre la tabla de caracteres, clasificando cada carácter como vocal o consonante. Finalmente, el programa resume los contenidos de la tabla, imprimiendo un breve informe en la consola.

En el Ejercicio 2, los estudiantes crearán y emplearán tablas de rango 2 y usarán el método **Array.GetLength(int dimension)**.

Al final de este módulo, los estudiantes serán capaces de:

- Crear, inicializar y usar tablas de distintos rangos.
- Usar argumentos de línea de comandos en un programa C#.
- Entender la relación entre una variable de tabla y una tabla.
- Usar tablas como parámetros de métodos.
- Devolver tablas desde métodos.

## ◆ Descripción general

**Objetivo del tema**

Ofrecer una introducción a los contenidos y objetivos del módulo.

**Explicación previa**

En este módulo repasará el concepto básico de tabla y aprenderá cómo crear, inicializar y usar tablas.

- Introducción a las tablas
- Creación de tablas
- Uso de tablas

Las tablas proporcionan una forma importante de agrupar datos. Para sacar el máximo partido a C# es fundamental entender como crear y usar tablas eficazmente.

Al final de este módulo, usted será capaz de:

- Crear, inicializar y usar tablas de distintos rangos.
- Usar argumentos de línea de comandos en un programa C#.
- Entender la relación entre una variable de tabla y una tabla.
- Usar tablas como parámetros de métodos.
- Devolver tablas desde métodos.

## ◆ Introducción a las tablas

**Objetivo del tema**

Ofrecer una introducción a los temas tratados en esta sección.

**Explicación previa**

Esta sección ofrece una introducción a los conceptos, características y sintaxis básica de las tablas.

- ¿Qué es una tabla?
- Notación para tablas en C#
- Rango de una tabla
- Acceso a los elementos de una tabla
- Comprobación de los límites de una tabla
- Comparación de tablas y colecciones

---

Esta sección ofrece una introducción a los conceptos generales de tablas, presenta la sintaxis que se utiliza in C# para declarar tablas y describe características básicas de tablas como rango y elementos. En la siguiente sección veremos cómo definir y usar tablas.

## ¿Qué es una tabla?

### Objetivo del tema

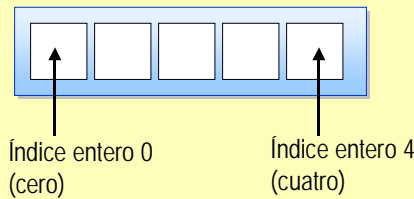
Explicar el concepto de tabla.

### Explicación previa

Los datos casi nunca están aislados, sino que suelen presentarse en grandes cantidades.

### ■ Una tabla es una secuencia de elementos

- Todos los elementos de una tabla son del mismo tipo
- Las estructuras pueden tener elementos de distintos tipos
- Se accede a elementos individuales usando índices enteros



Básicamente hay dos formas de agrupar datos relacionados: estructuras (**structs**) y tablas (**arrays**).

### Recomendación al profesor

Explique las diferencias entre una tabla y una estructura. Las estructuras pueden contener tipos diferentes y se accede a sus miembros por su nombre. Las tablas contienen el mismo tipo y se accede a sus elementos por un índice entero.

Destaque que el concepto de tabla es algo natural, ya que en cualquier situación suele ser necesario considerar muchos tipos de información. Por ejemplo, una tabla de casas forma una calle.

- Las estructuras son grupos de datos relacionados que tienen tipos diferentes.

Por ejemplo, un nombre (**string**), una edad (**int**), y un género (**enum**) se agrupan de forma natural en una **struct** que describe una persona. Se puede acceder a miembros individuales de una estructura utilizando sus nombres de campo.

- Las tablas son secuencias de datos del mismo tipo.

Por ejemplo, una serie de casas se agrupa de forma natural para formar una calle. Se puede acceder a un elemento individual de una tabla utilizando su posición entera, que recibe el nombre de índice.

Las tablas permiten el acceso aleatorio. Los elementos de una tabla ocupan posiciones de memoria contiguas, lo que significa que un programa puede acceder con la misma rapidez a todos los elementos de una tabla.

## Notación para tablas en C#

### Objetivo del tema

Terminar la sintaxis y terminología que se utilizan en C# para declarar una tabla.

### Explicación previa

La mayor parte de los lenguajes de programación tienen una notación propia que permite declarar una entidad como tabla.

### ■ Una variable de tabla se declara especificando:

- El tipo de elementos de la tabla
- El rango de la tabla
- El nombre de la variable

`tipo[ ] nombre;`

↑  
Especifica el nombre de la variable de tabla

↑  
Especifica el rango de la tabla

↑  
Especifica el tipo de elementos de la tabla

### Recomendación al profesor

El rango se discute en la siguiente transparencia.

Esta transparencia muestra únicamente la sintaxis para una tabla de rango 1. La sintaxis se ampliará en la siguiente transparencia a tablas de rango 2.

Para declarar una tabla se utiliza la misma notación que para declarar una variable simple. Primero se especifica el tipo y a continuación el nombre de la variable seguido de un punto y coma. Para declarar el tipo de la variable como tabla se emplean corchetes. Muchos lenguajes de programación, como C y C++, también utilizan corchetes para declarar una tabla, mientras que otros, como Microsoft® Visual Basic®, usan paréntesis.

La notación para tablas en C# es muy similar a la empleada en C y C++, aunque con dos diferencias sutiles pero importantes:

- No se pueden poner corchetes a la derecha del nombre de la variable.
- Al declarar una variable de tabla no se especifica el tamaño de la tabla.

A continuación se ofrecen ejemplos de notaciones permitidas y no permitidas en C#:

```
tipo[ ]nombre;    // Permitida
tipo nombre[ ];   // No permitida en C#
tipo[4] nombre;   // Tampoco permitida en C#
```

## Rango de una tabla

### Objetivo del tema

Introducir el concepto de rango de tabla en C# y explicar cómo declarar tablas de rango 2.

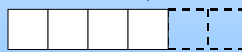
### Explicación previa

Distintos tipos de tablas requieren distintos números de índices para acceder a sus elementos. Por ejemplo, las celdas en una hoja de cálculo de Microsoft Excel requieren un índice de fila y un índice de columna.

- El rango se conoce también como **dimensión de la tabla**
- El número de índices asociados con cada elemento

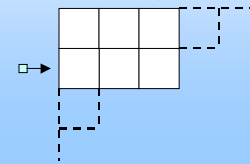
```
long[ ] fila;
```

Rango 1: Unidimensional  
Un solo índice asociado con cada elemento **long**



```
int[,] cuadrícula;
```

Rango 2: Bidimensional  
Dos índices asociados con cada elemento **int**



### Recomendación al profesor

Explique el término *rango* con respecto a tablas. La transparencia muestra la sintaxis para tablas de rango 1 y 2. Dé un ejemplo de rango 3 en una pizarra.

Nota: Los gráficos que ilustran las tablas de rango 1 y 2 no muestran longitudes de tablas porque todavía no se ha discutido su sintaxis.

Para declarar a una variable de tabla unidimensional se utilizan corchetes vacíos, como se ve en la transparencia. Esta tabla se llama también tabla de rango 1 porque hay un índice entero asociado con cada elemento de la tabla.

Para declarar a una tabla bidimensional se utiliza una sola coma dentro de los corchetes, como se ve en la transparencia. Esta tabla se llama también tabla de rango 2 porque hay dos índices enteros asociados con cada elemento de la tabla. La extensión de esta notación es obvia: cada coma adicional dentro de los corchetes aumenta el rango de la tabla en uno.

En la declaración de una variable de tabla no se incluye la longitud de las dimensiones.

## Acceso a los elementos de una tabla

### Objetivo del tema

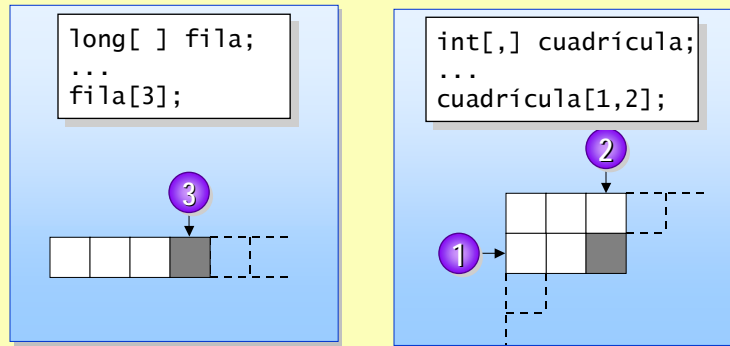
Describir la sintaxis para acceder a un elemento de una tabla y destacar que los índices de tablas se empiezan a contar desde cero.

### Explicación previa

Una vez declarada una variable de tabla, hay que saber cómo acceder a los elementos de la tabla.

### ■ Se indica un índice entero para cada rango

- Los índices se cuentan a partir de cero



### Recomendación al profesor

Esta transparencia es similar a la anterior, pero en este caso se accede a un solo elemento en cada ejemplo. El punto más importante es que los índices se cuentan a partir de cero. Aquí se muestra únicamente el acceso a elementos válidos, ya que las excepciones por fuera de límites se tratarán en la siguiente transparencia.

Técnicamente es posible crear una tabla con límites inferiores definidos por el usuario, como en `Array.CreateInstance` (Tipo, `int[ ]`, `int[ ]`), pero probablemente sea mejor no mencionarlo.

La sintaxis que se utiliza para acceder a los elementos de una tabla es similar a la empleada para declarar variables de tabla, ya que en ambas se utilizan corchetes. Este parecido visual (que es deliberado y sigue la tendencia marcada por C y C++) puede resultar confuso si no se está familiarizado con él. Por eso es importante ser capaz de distinguir entre la declaración de una variable de tabla y la expresión para acceder a los elementos de una tabla.

Para acceder a un elemento de una tabla de rango 1 se utiliza un índice entero. Para acceder a un elemento de una tabla de rango 2 se utilizan dos índices enteros separados por una coma. La extensión de esta notación es igual que en la declaración de variables: Para acceder a un elemento de una tabla de rango  $n$  se utilizan  $n$  índices enteros separados por comas. Como puede verse, la sintaxis usada en una expresión para acceder a los elementos de una tabla es igual a la empleada para declarar variables.

Los índices de una tabla (de cualquier rango) se cuentan desde cero. Así, para acceder al primer elemento de una fila se usa la expresión:

`fila[0]`

en lugar de:

`fila[1]`



Algunos programadores prefieren decir “elemento inicial” en lugar de “primer elemento” para evitar la posibilidad de confusión. Contar los índices desde 0 significa que el último elemento de una tabla con *tamaño* elementos se encuentra en [*tamaño* - 1] y no en [*tamaño*]. Es muy habitual cometer el error de usar [*tamaño*], especialmente para programadores acostumbrados a lenguajes que cuentan los índices a partir de uno, como Visual Basic.

---

**Nota** Aunque es una técnica que apenas se utiliza, es posible crear tablas con límites inferiores de índice entero definido por el usuario. Para más información, busque “Array.CreateInstance” en los documentos de ayuda del SDK de Microsoft .NET Framework.

---

## Comprobación de los límites de una tabla

### Objetivo del tema

Describir el comportamiento del índice de una tabla cuando está fuera de límites y explicar cómo encontrar la longitud de cada rango de tabla.

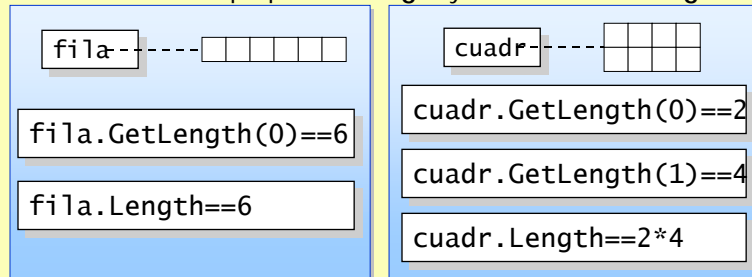
### Explicación previa

¿Qué ocurre si un índice está fuera de límites?

¿Cómo se puede saber la longitud de cada rango de tabla?

### ■ Se comprueban los límites cada vez que se intenta acceder a una tabla

- Un índice erróneo lanza la excepción `IndexOutOfRangeException`
- Se usan la propiedad **Length** y el método **GetLength**



### Recomendación al profesor

Esta transparencia muestra únicamente ejemplos de longitud fija para poder discutir tanto la propiedad **Length** como el método **GetLength**. Es importante estudiar **GetLength** porque se utilizará en el Ejercicio 2 de la práctica.

No dedique demasiado tiempo a la excepción `IndexOutOfRangeException`. Las excepciones no son importantes en este módulo y no se emplean en los ejercicios.

La expresión para acceder a un elemento de una tabla en C# se analiza automáticamente para comprobar que el índice es válido. Esta comprobación implícita de límites está siempre activada. Comprobar los límites es una manera de garantizar que C# es un lenguaje de especificaciones seguras.

Aunque se comprueben automáticamente, hay que asegurarse de que los índices de una tabla están dentro de los límites. Para ello es preciso comprobar manualmente los límites de los índices, para lo cual se suele emplear la condición de fin de una expresión **for**, como en este ejemplo:

```
for (int i = 0; i < fila.Length; i++) {
    Console.WriteLine(fila[i]);
}
```

La propiedad **Length** es la longitud total de la tabla, independientemente de su rango. Para determinar la longitud de una dimensión concreta se puede utilizar el método **GetLength** como se muestra a continuación:

```
for (int r = 0; r < cuadr.GetLength(0); r++) {
    for (int c = 0; c < cuadr.GetLength(1); c++) {
        Console.WriteLine(cuadr[r,c]);
    }
}
```

## Comparación de tablas y colecciones

### Objetivo del tema

Describir las ventajas e inconvenientes de tablas y colecciones.

### Explicación previa

¿Cuáles son las ventajas e inconvenientes de tablas y colecciones? ¿Cuándo se debe usar una tabla? ¿Cuándo hay que emplear una colección?

- **Una tabla no puede cambiar su tamaño cuando está llena**
  - Una clase de colección, como ArrayList, puede cambiar su tamaño
- **Una tabla contiene elementos de un solo tipo**
  - Una colección está diseñada para contener elementos de distintos tipos
- **Los elementos de una tabla no pueden ser de sólo lectura**
  - Una colección puede tener acceso de sólo lectura
- **En general, las tablas son más rápidas pero menos flexibles**
  - Las colecciones son algo más lentas pero más flexibles

El tamaño de una tabla y el tipo de sus elementos se fijan permanentemente en el momento de crearla. Para crear una tabla que siempre contenga exactamente 42 elementos de tipo **int** se usa la siguiente sintaxis:

```
int[ ] rigid = new int [ 42 ];
```

Una tabla no aumenta ni disminuye de tamaño y nunca puede contener nada que no sean **ints**. Las colecciones son más flexibles, ya que se pueden expandir o contraer a medida que se eliminan o añaden elementos. Las tablas contienen elementos de un solo tipo, mientras que las colecciones están diseñadas para albergar elementos de muchos tipos diferentes. Esta flexibilidad se puede conseguir utilizando boxing, como se muestra a continuación:

```
ArrayList flexible = new ArrayList( );
flexible.Add("one"); // Añadir string
...
flexible.Add(99);    // Añadir int
```

No es posible crear una tabla con elementos de sólo lectura. Este código, por ejemplo, no se compilará:

```
const int[ ] array = {0, 1, 2, 3};
readonly int[ ] array = {4,2};
```

Por el contrario, se puede crear una colección de sólo lectura:

```
ArrayList flexible = new ArrayList( );
...
ArrayList noWrite = ArrayList.ReadOnly(flexible);
noWrite [0] = 42;    // Excepción en tiempo de ejecución
```

## ◆ Creación de tablas

**Objetivo del tema**

Ofrecer una introducción a los temas tratados en esta sección.

**Explicación previa**

Una vez explicados los conceptos básicos de las tablas, vamos a ver la sintaxis que se utiliza para crear tablas en C#. Casi toda la sintaxis explicada en esta sección se empleará después en las prácticas.

- Creación de una tabla
- Inicialización de los elementos de una tabla
- Inicialización de los elementos de una tabla multidimensional
- Creación de una tabla de tamaño calculado
- Copia de variables de tabla

---

En esta sección veremos cómo crear tablas, cómo inicializar explícitamente los elementos de una tabla y cómo copiar variables de tabla.

## Creación de una tabla

### Objetivo del tema

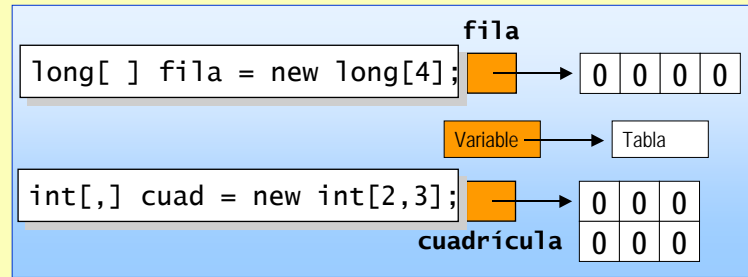
Presentar tablas y explicar la relación entre tablas y variables de tabla.

### Explicación previa

Una variable de tabla es distinta a todas las variables vistas hasta ahora.

### ■ ¡Declarar una variable de tabla no es lo mismo que crear una tabla!

- Para crear la tabla explícitamente hay que usar **new**
- El valor implícito por defecto de los elementos de una tabla es cero



### Recomendación al profesor

Recuerde que aún no se ha discutido la palabra reservada **new**. Mencione que un elemento de punto flotante se inicializa a 0.0 y que un elemento booleano se inicializa a **false**. Escriba en la pizarra un ejemplo que muestre cómo crear una tabla de estructuras.

La declaración de una variable de tabla no implica la creación de una tabla, ya que las tablas no son tipos de valor sino tipos de referencia. En la expresión para crear una tabla se emplea la palabra reservada **new** y en ella hay que especificar el tamaño de todas las longitudes del rango. El siguiente código producirá un error en tiempo de compilación:

```
long[ ] fila = new long[ ];    // No permitido
int[,] cuadr = new int[,];    // No permitido
```

El compilador de C# inicializa implícitamente todos los elementos de una tabla a un valor que depende de su tipo: los elementos enteros se inicializan implícitamente a 0, los elementos de punto flotante a 0.0 y los elementos booleanos a **false**. En otras palabras, el código C#:

```
long[ ] fila = new long[4];
```

ejecutará el siguiente código en tiempo de ejecución:

```
long[ ] fila = new long[4];
fila[0] = 0L;
fila[1] = 0L;
fila[2] = 0L;
fila[3] = 0L;
```

El compilador siempre asigna a una tabla posiciones de memoria contiguas, independientemente de cuál sea su tipo y número de dimensiones. Si se crea una tabla con una expresión como `new int[2, 3, 4]`, conceptualmente es 2 x 3 x 4, pero la asignación de memoria subyacente es un solo bloque de memoria lo suficientemente grande para contener  $2 \times 3 \times 4$  elementos.

## Inicialización de los elementos de una tabla

### Objetivo del tema

Describir la sintaxis que se emplea para inicializar los elementos de una tabla unidimensional.

### Explicación previa

En muchos casos puede ser necesario inicializar los elementos de una tabla a valores distintos de cero.

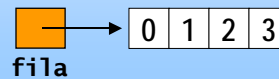
### ■ Es posible inicializar explícitamente los elementos de una tabla

- Se puede utilizar una expresión abreviada

```
long[ ] fila = new long[4] {0, 1, 2, 3};
```

```
long[ ] fila = {0, 1, 2, 3};
```

← Equivalentes



### Recomendación al profesor

La expresión abreviada se utiliza muy a menudo, pero los estudiantes deben ser conscientes de que la expresión de creación de una tabla sigue siendo implícita. Las notas para los estudiantes incluyen más información sobre sintaxis que debe explicar en la pizarra.

En el Ejercicio 1 de la práctica se empleará la sintaxis para inicializar una tabla de rango 1.

Se puede utilizar un inicializador de tabla para inicializar los valores de los elementos de una tabla. Un inicializador de tabla es una secuencia de expresiones puestas entre llaves y separadas por comas. Los inicializadores de tablas se ejecutan de izquierda a derecha y pueden incluir llamadas a métodos y expresiones complejas, como en el siguiente ejemplo:

```
int[ ] data = new int[4]{a, b( ), c*d, e( )+f( )};
```

Los inicializadores de tablas también se pueden utilizar para inicializar tablas de estructuras:

```
struct Date { ... }
Date [ ] dates = new Date[2];
```

Esta notación abreviada sólo se puede emplear cuando se inicializa una tabla como parte de la declaración de una variable de tabla, y no como parte de una instrucción de asignación normal.

```
int[ ] data1 = new int[4]{0, 1, 2, 3}; // Permitido
int[ ] data2 = {0, 1, 2, 3}; // Permitido
data2 = new int[4]{0, 1, 2, 3}; // Permitido
data2 = {0, 1, 2, 4}; // No permitido
```

Cuando se inicializan tablas hay que inicializar explícitamente todos los elementos de la tabla. No es posible dejar que sólo algunos elementos adopten su valor cero predeterminado:

```
int[ ] data3 = new int[2]{}; // No permitido
int[ ] data4 = new int[2]{42}; // Tampoco permitido
int[ ] data5 = new int[2]{42,42}; // Permitido
```

## Inicialización de los elementos de una tabla multidimensional

### Objetivo del tema

Explicar cómo inicializar los elementos de una tabla unidimensional.

### Explicación previa


En muchos casos puede ser necesario inicializar los elementos de una tabla multidimensional a valores distintos de cero.

### ■ También se pueden inicializar los elementos de una tabla multidimensional

- Hay que especificar todos los elementos

```
int[,] cuadr = {
    {5, 4, 3},
    {2, 1, 0}
};
```

← Nueva tabla int[2,3] implícita



✓  
**cuadrícula**

5	4	3
2	1	0

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

✗

### Recomendación al profesor

Este tema es importante, ya que muestra la sintaxis para inicializar tablas de rango mayor que uno. Los estudiantes necesitarán saber cómo inicializar tablas de rango 2 para hacer el Ejercicio 2 de la práctica.

Todos los elementos de una tabla deben estar inicializados explícitamente, independientemente de la dimensión de la tabla:

```
int[,] data = new int[2,3] {           // Permitido
    {42, 42, 42},
    {42, 42, 42},
};
```

```
int[,] data = new int[2,3] {           // No permitido
    {42, 42},
    {42, 42, 42},
};
```

```
int[,] data = new int[2,3] {           // No permitido
    {42},
    {42, 42, 42},
};
```



## Creación de una tabla de tamaño calculado

### Objetivo del tema

Explicar que no es necesario que el tamaño de una tabla sea una constante de tiempo de compilación y que se accede a los elementos de una tabla con la misma rapidez independientemente de cómo se especifique el tamaño de la tabla.

### Explicación previa

¿Es posible especificar el tamaño de una tabla usando una expresión entera de tiempo de ejecución?  
¿Haría eso que el acceso a los elementos fuera más lento?

- No es necesario que el tamaño de una tabla sea una constante de tiempo de compilación
    - Se puede usar cualquier expresión entera válida
    - El acceso a los elementos es igualmente rápido en todos los casos
- Tamaño de tabla especificado por constante entera de tiempo de compilación:

```
long[ ] fila = new long[4];
```

Tamaño de tabla especificado por valor entero de tiempo de ejecución:

```
string s = Console.ReadLine();
int tamano = int.Parse(s);
long[ ] fila = new long[tamano];
```

### Recomendación al profesor

El código de la transparencia supone la existencia de una instrucción using. Explique que una tabla de rango mayor o igual que 2 sigue siendo una sola tabla y por tanto es muy eficiente.

En el Ejercicio 1 de la práctica se empleará la sintaxis para crear una tabla de tamaño calculado.

Las instrucciones para leer un valor desde la consola se utilizan en los dos ejercicios.

Es posible crear tablas multidimensionales utilizando expresiones de tiempo de ejecución para la longitud de cada dimensión, como en el siguiente código:

```
System.Console.WriteLine("Escriba el número de filas: ");
string s1 = System.Console.ReadLine( );
int filas = int.Parse(s1);
System.Console.WriteLine("Escriba el número de columnas: ");
string s2 = System.Console.ReadLine( );
int columnas = int.Parse(s2);
...
int[,] matriz = new int[filas,columnas];
```

También se puede emplear una mezcla de constantes de tiempo de compilación y expresiones de tiempo de ejecución:

```
System.Console.WriteLine("Escriba el número de filas: ");
string s1 = System.Console.ReadLine( );
int filas = int.Parse(s1);
...
int[,] matriz = new int[filas,4];
```

Hay una pequeña restricción, ya que no está permitido usar una expresión de tiempo de ejecución para especificar el tamaño de una tabla en combinación con inicializadores de tablas:

```
string s = System.Console.ReadLine( );
int tamano = int.Parse(s);
int[ ] data = new int[tamano]{0,1,2,3}; // No permitido
```

## Copia de variables de tablas

### Objetivo del tema

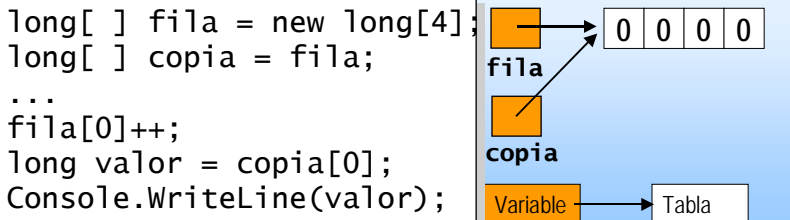
Explicar qué ocurre (y qué no ocurre) cuando se copia una variable de tabla.

### Explicación previa

¿Qué se copia cuando se copia una variable de tabla?  
¿Es la variable de tabla o es la tabla?

### ■ Al copiar una variable de tabla se copia sólo la variable de tabla

- No se copia la tabla
- Dos variables de tabla pueden apuntar a la misma tabla



### Recomendación al profesor

Este tema es importante. Compare la copia de una variable **struct** con la copia de una variable de tabla. Compare el siguiente fragmento de código con el de la transparencia:

```

struct Value {
    public long
    a,b,c,d;
}
Value fila;
fila.a = 0L;
fila.b = 0L;
fila.c = 0L;
fila.d = 0L;
Value copia =
fila;
fila.a++;
long valor =
copia.a;
Console.WriteLine(
value);

```

Cuando se copia una variable de tabla lo que se obtiene no es una copia completa de la tabla. Un análisis del código de la transparencia revela qué es lo que ocurre al copiar una variable de tabla.

Las instrucciones siguientes dos declaran variables de tabla llamadas *copia* y *fila* que apuntan a la misma tabla (de cuatro enteros **long**).

```

long[ ] fila = new long[4];
long[ ] copia = fila;

```

Esta instrucción incrementa de 0 a 1 el elemento inicial de esta tabla. Ambas variables de tabla siguen apuntando a la misma tabla, cuyo elemento inicial es ahora 1.

```

fila[0]++;

```

La siguiente instrucción inicializa un entero **long** llamado *valor* a *copia*[0], que es el elemento inicial de la tabla a la que apunta *copia*.

```

long valor = copia[0];

```

Puesto que *copia* y *fila* apuntan ambos a la misma tabla, inicializar a *fila*[0] tiene exactamente el mismo efecto.

La instrucción final escribe *valor* (que es 1) en la consola:

```

Console.WriteLine(value);

```

## ◆ Uso de tablas

### Objetivo del tema

Ofrecer una introducción a los temas tratados en esta sección.

### Explicación previa

Ahora que ya sabemos cómo crear e inicializar tablas, vamos a explicar la sintaxis del uso de tablas. Casi toda la sintaxis explicada en esta sección se empleará después en las prácticas.

- Propiedades de tablas
- Métodos de tablas
- Devolución de tablas desde métodos
- Paso de tablas como parámetros
- Argumentos de línea de comandos
- Demostración: Argumentos para Main
- Uso de tablas con foreach
- Problema: ¿Dónde está el error?

En esta sección estudiaremos cómo usar tablas y cómo pasar tablas como parámetros a métodos.

Veremos las reglas que rigen los valores predeterminados de los elementos de una tabla. Las tablas heredan implícitamente de la clase **System.Array**, que contiene muchas propiedades y métodos de los cuales discutiremos los empleados más frecuentemente. También estudiaremos cómo utilizar la instrucción **foreach** para recorrer tablas. Finalmente, discutiremos cómo evitar algunos errores comunes.

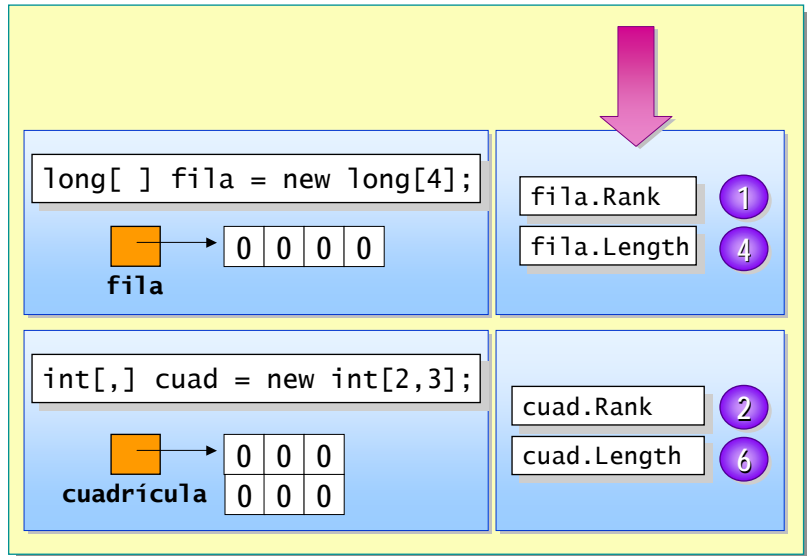
## Propiedades de tablas

### Objetivo del tema

Presentar algunas propiedades útiles que las tablas heredan de **System.Array**.

### Explicación previa

La clase **System.Array** contiene varias propiedades útiles que utilizan todas las tablas.



### Recomendación al profesor

Recuerde que la propiedad **Length** da la longitud total de todas las dimensiones multiplicadas entre sí. La propiedad **Length** se empleará en el Ejercicio 1 de la práctica. Trate la sintaxis de las propiedades como si fueran una caja negra, ya que no se verán en detalle hasta un módulo posterior.

La propiedad **Rank** es un valor entero de sólo lectura que indica la dimensión de una tabla. Por ejemplo, en el código

```
int[ ] uno = new int[a];  
int[,] dos = new int[a,b];  
int[, ,] tres = new int[a,b,c];
```

los valores resultantes del rango son los siguientes:

```
uno.Rank == 1  
dos.Rank == 2  
tres.Rank == 3
```

La propiedad **Length** es un valor entero de sólo lectura que indica la longitud total de una tabla. Por ejemplo, dadas las tres declaraciones anteriores de tablas, los valores resultantes de la longitud son:

```
uno.Length == a  
dos.Length == a * b  
tres.Length == a * b * c
```

## Métodos de tablas

### Objetivo del tema

Presentar algunos métodos útiles que las tablas heredan de **System.Array**.

### Explicación previa

La clase **System.Array** contiene varios métodos útiles que utilizan todas las tablas.

### ■ Métodos utilizados frecuentemente

- **Sort**: Ordena los elementos en una tabla de rango 1
- **Clear**: Asigna el valor cero o **null** a un rango de elementos
- **Clone**: Crea una copia de la tabla
- **GetLength**: Devuelve la longitud de una dimensión dada
- **IndexOf**: Devuelve el índice de la primera vez que aparece un valor

### Recomendación al profesor

Estos métodos no se usan en ninguno de los ejercicios. Es posible ordenar una tabla de una estructura o clase definida por el usuario, siempre y cuando la estructura o clases admita la interfaz **IComparable**. **Sort**, **Clear** e **IndexOf** son métodos static.

La clase **System.Array** (que utilizan implícitamente todas las tablas) contiene muchos métodos que se pueden utilizar para trabajar con tablas. Este tema describe los métodos empleados más frecuentemente

#### • Método **Sort**

Este método efectúa una ordenación en contexto de la tabla pasada como argumento. Se puede utilizar para ordenar tablas de estructuras y clases que admitan la interfaz **IComparable**.

```
int[ ] data = {4,6,3,8,9,3}; // Sin ordenar
System.Array.Sort(data);    // Ordenada
```

#### • Método **Clear**

Este método cambia el valor de un rango de elementos de la tabla a cero (para tipos de valor) o **null** (para tipos de referencia), como en este ejemplo:

```
int[ ] data = {4,6,3,8,9,3};
System.Array.Clear(data, 0, data.Length);
```

#### • Método **Clone**

Este método crea una nueva tabla cuyos elementos son copias de los elementos de la tabla original. Se puede utilizar para clonar tablas de estructuras y clases definidas por el usuario. El siguiente es un ejemplo:

```
int[ ] data = {4,6,3,8,9,3};
int[ ] clone = (int[ ])data.Clone();
```

---

**Precaución** El método **Clone** realiza una copia superficial. Si la tabla copiada contiene referencias a objetos, se copiarán las referencias pero no lo objetos; las dos tablas apuntarán a los mismos objetos.

---

- **Método GetLength**

Este método devuelve la longitud de una dimensión pasada como un argumento entero. Se puede utilizar para comprobar los límites de tablas multidimensionales. El siguiente es un ejemplo:

```
int[,] data = { {0, 1, 2, 3}, {4, 5, 6, 7} };  
int dim0 = data.GetLength(0); // == 2  
int dim1 = data.GetLength(1); // == 4
```

- **Método IndexOf**

Este método devuelve el índice entero de la primera vez que aparece un valor pasado como argumento, o -1 si el valor no está presente. Sólo se puede utilizar en tablas unidimensionales. El siguiente es un ejemplo:

```
int[] data = {4,6,3,8,9,3};  
int donde = System.Array.IndexOf(data, 9); // == 4
```

---

**Nota** Dependiendo del tipo de los elementos en la tabla, el método **IndexOf** puede obligar a omitir el método **Equals** para el tipo de los elementos. Esto se estudiará en un módulo posterior.

---

## Devolución de tablas desde métodos

### Objetivo del tema

Describir la sintaxis que se utiliza para declarar un método que devuelve una tabla, y mostrar un ejemplo de devolución de una tabla usando una instrucción **return**.

### Explicación previa

¿Cómo se declara un método que devuelve una tabla?

- Es posible declarar métodos para que devuelvan tablas

```
class Example {
    static void Main( ) {
        int[ ] array = CreateArray(42);
        ...
    }
    static int[ ] CreateArray(int tamano) {
        int[ ] creada = new int[tamano];
        return creada;
    }
}
```

### Recomendación al profesor

Comente para declarar una variable de tabla es similar a la que se usa para declarar un método que devuelve una tabla.

En el Ejercicio 2 de la práctica, los estudiantes tendrán que devolver una tabla de rango 2 desde un método.

El método **CreateArray** en la transparencia contiene dos instrucciones. Es posible combinar estas dos instrucciones en una sola instrucción **return** como se indica a continuación:

```
static int[ ] CreateArray(int tamano) {
    return new int[tamano];
}
```

Los programadores en C++ notarán que en ninguno de los dos casos se especifica el tamaño de la tabla que se devuelve. Si se indica el tamaño de la tabla se obtendrá un error en tiempo de ejecución, como en este ejemplo:

```
static int[4] CreateArray( ) // Error de compilación

{
    return new int[4];
}
```

También es posible devolver tablas de rango mayor que uno, como se ve en el siguiente ejemplo:

```
static int[,] CreateArray( ) {
    string s1 = System.Console.ReadLine( );
    int filas = int.Parse(s1);
    string s2 = System.Console.ReadLine( );
    int columnas = int.Parse(s2);
    return new int[filas,columnas];
}
```

## Paso de tablas como parámetros

**Objetivo del tema**

Explicar cómo se puede pasar una tabla como parámetro y qué ocurre cuando se hace.

**Explicación previa**

Ya hemos visto cómo devolver una tabla desde un método. Veamos ahora cómo pasar una tabla a un método como parámetro.

- Un parámetro de tabla es una copia de la variable de tabla
  - No es una copia de la tabla

```
class Example2 {  
    static void Main( ) {  
        int[ ] arg = {10, 9, 8, 7};  
        Method(arg);  
        System.Console.WriteLine(arg[0]);  
    }  
    static void Metodo(int[ ] parametro) {  
        parametro[0]++;  
    }  
}
```

Este método modificará la tabla original creada en Main



**Recomendación al profesor**

Es importante que los estudiantes sepan pasar tablas como parámetros, ya que tendrán que hacerlo en los dos ejercicios de la práctica. Puede empezar con el siguiente código para ver hasta qué punto los estudiantes entienden este proceso:

```
static void Main(
) {
    int arg = 42;
    m(arg);
}
void m(int param)
{
    param++;
}
```

Luego se convierte **arg** en una tabla que contiene un solo **int**:

```
static void Main(
) {
    int[ ] arg = {
42 };
    m(arg);
}
void m(int[ ]
param) {
    param[0]++;
}
```

Cuando se pasa una variable de tabla como argumento para un método, el parámetro del método pasa a ser una copia del argumento de la variable de tabla. En otras palabras, el parámetro de tabla se inicializa desde el argumento. Para inicializar el parámetro de tabla se usa la misma sintaxis que para inicializar una variable de tabla, como se explicó anteriormente en el tema “Copia de variables de tablas”. Tanto el argumento de tabla como el parámetro de tabla apuntan a la misma tabla.

En el código mostrado en la transparencia, **arg** se inicializa con una tabla de longitud 4 que contiene los enteros 10, 9, 8 y 7. A continuación se pasa **arg** como argumento para **Method**. **Method** acepta **arg** como parámetro, lo que quiere decir que tanto **arg** como el parámetro apuntan a la misma tabla (la empleada para inicializar **arg**). La expresión **parametro[0]++** en **Method** hace que se incremente el elemento inicial de la misma tabla de 10 a 11 (el elemento inicial de una tabla también recibe el nombre de elemento “cero”, ya que se accede a él con el valor de índice 0, no 1). **Method** devuelve el valor de **arg[0]** y **Main** lo escribe en la consola. El parámetro **arg** continúa apuntando a la misma tabla cuyo elemento cero ha sido incrementado, por lo que en la consola se escribe 11.

Cuando se pasa una variable de tabla no se crea una copia profunda de la tabla, lo que hace que pasar una tabla como parámetro sea muy rápido. Esta copia superficial es más que suficiente si se quiere que un método tenga acceso de escritura a la tabla del argumento.

El método **Array.Copy** resulta útil para asegurarse de que el método llamado no puede modificar la tabla, aunque esta seguridad se paga con un tiempo de ejecución más largo. También es posible pasar como parámetro de tabla una tabla de nueva creación de la siguiente manera:

```
Method(new int[4]{10, 9, 8, 7});
```

## Argumentos de línea de comandos

### Objetivo del tema

Explicar cómo acceder a argumentos de línea de comandos en **Main**.

### Explicación previa

Cuando se ejecutan programas de consola es frecuente añadir información en la línea de comandos.

¿Cómo accede el programa a esa información?

### ■ El runtime pasa argumentos de línea de comandos a **Main**

- **Main** puede aceptar como parámetro una tabla de cadenas de caracteres
- El nombre del programa no es un miembro de la tabla

```
class Example3 {  
    static void Main(string[ ] args) {  
        for (int i = 0; i < args.Length; i++) {  
            System.Console.WriteLine(args[i]);  
        }  
    }  
}
```

### Recomendación al profesor

En el Ejercicio 1 de la Práctica se emplea el paso de argumentos a **Main** en la línea de comandos. El código mostrado en la transparencia utiliza una instrucción **for**; en la siguiente transparencia se mostrará la instrucción **foreach** equivalente.

Cuando se ejecutan programas de consola es muy frecuente pasar nuevos argumentos en la línea de comandos. Por ejemplo, al añadir el programa **pkzip** desde el símbolo del sistema es posible añadir argumentos para controlar la creación de archivos .zip. El siguiente comando añade recursivamente a code.zip todos los archivos de código \*.cs:

```
C:\> pkzip -add -rec -path=relative c:\code *.cs
```

Si el programa **pkzip** estuviera escrito en C#, estos argumentos de línea de comandos se capturarían como una tabla de cadenas de caracteres que el runtime pasaría a **Main**:

```
class PKZip {  
    static void Main(string[ ] args) {  
        ...  
    }  
}
```

Cuando se ejecuta el programa **pkzip** con este ejemplo, el runtime ejecuta realmente el siguiente código:

```
string[ ] args = {  
    "-add",  
    "-rec",  
    "-path=relative",  
    "c:\\code",  
    "*.cs"  
};  
PKZip.Main(args);
```

---

**Nota** Al contrario de lo que ocurre en C y C++, in C# no se pasa el nombre del programa como args[0].

---

## Uso de tablas con foreach

### Objetivo del tema

Explicar cómo usar una tabla en una instrucción **foreach**.

### Explicación previa

Es muy habitual escribir código que recorre una tabla y ejecuta las mismas instrucciones para cada elemento. El uso de **foreach** simplifica este proceso.

- La instrucción **foreach** simplifica enormemente la manipulación de tablas

```
class Example4 {  
    static void Main(string[] args) {  
        foreach (string arg in args) {  
            System.Console.WriteLine(arg);  
        }  
    }  
}
```

### Recomendación al profesor

La instrucción **foreach** se empleará en los dos ejercicios de la práctica, ya que puede recorrer tablas multidimensionales.

Cuando se puede emplear, la instrucción **foreach** resulta útil porque simplifica el proceso de recorrer de uno en uno todos los elementos de una tabla. Sin **foreach** se podría escribir:

```
for (int i = 0; i < args.Length; i++) {  
    System.Console.WriteLine(args[i]);  
}
```

Con **foreach** se convertiría en:

```
foreach (string arg in args) {  
    System.Console.WriteLine(arg);  
}
```

Con la instrucción **foreach** no se necesita ni se utiliza:

- Un índice entero (`int i`)
- Una comprobación de los límites de la tabla (`i < args.Length`)
- Una expresión de acceso a la tabla (`args[i]`)

También se puede usar la instrucción **foreach** para recorrer de uno en uno todos los elementos de una tabla de rango 2 o superior. Por ejemplo, la siguiente instrucción **foreach** escribirá los valores 0, 1, 2, 3, 4 y 5:

```
int[,] numbers = { {0,1,2}, {3,4,5} };  
foreach (int number in numbers) {  
    System.Console.WriteLine(number);  
}
```