UNIVERSIDAD AUTÓNOMA DE SINALOA

Facultad de Informática Culiacán

Arreglos en C#

Instructor:

MC. Gerardo Gálvez Gámez

gerardo.galvez@uas.edu.mx

Septiembre de 2017

Arreglos

C#

Competencias

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

- Describir los distintos tipos de Arreglos.
- Crear arreglos.
- Usar arreglos.

Descripción general

Los arreglos proporcionan una forma importante de agrupar datos.

Para sacar el máximo partido a C#, es fundamental entender como:

- Creación de arreglos
- Uso de arreglos

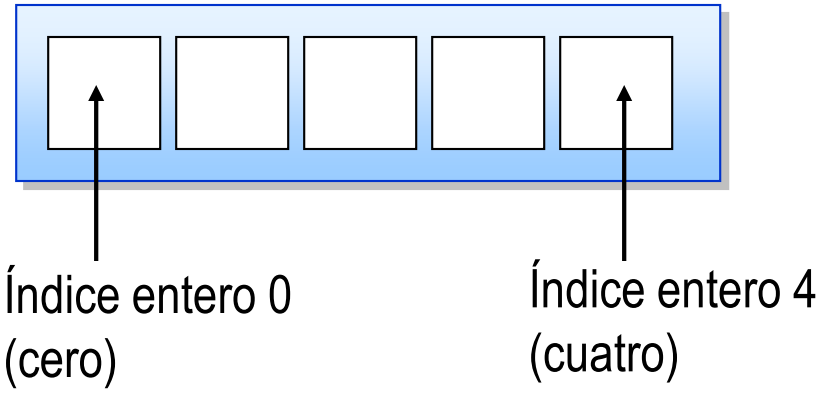
Introducción a los arreglos

- ¿Qué es un arreglo?
- Notación para arreglos en C#
- Rango de un arreglo
- Acceso a los elementos de un arreglo
- Comprobación de los límites de un arreglo
- Comparación de arreglos y colecciones

¿Qué es un arreglo?

Un arreglo es una secuencia de elementos

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



¿Qué es un arreglo?

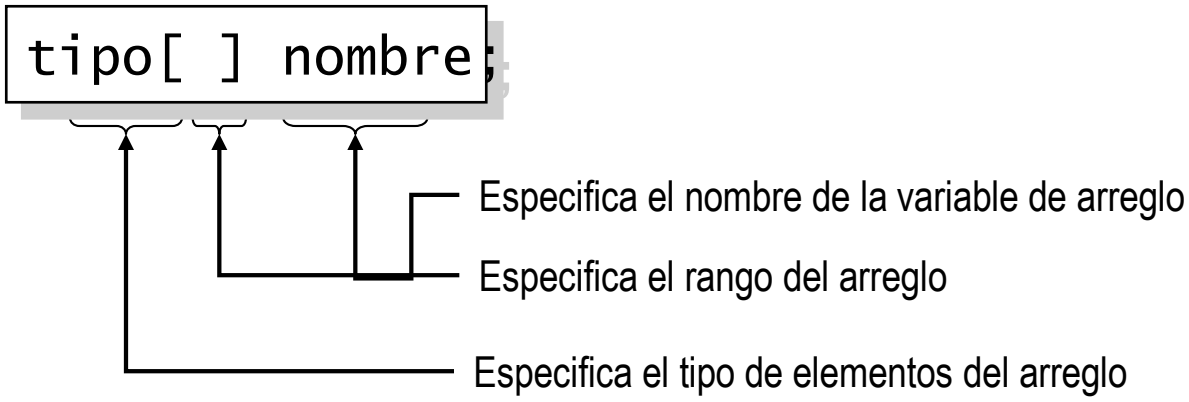
Los arreglos permiten el acceso aleatorio.

Los elementos de un arreglo ocupan posiciones de memoria contiguas, lo que significa que un programa puede acceder con la misma rapidez a todos los elementos de un arreglo.

Notación para arreglos en C#

Una variable de arreglo se declara especificando:

- El tipo de elementos del arreglo
- El rango del arreglo
- El nombre de la variable



Notación para arreglos en C#

La notación para arreglos 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.

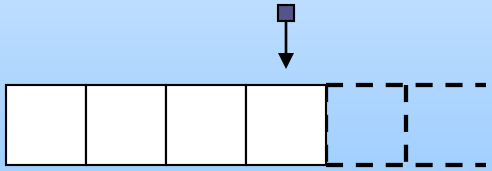
- Al declarar una variable de arreglo no se especifica el tamaño del arreglo.
- A continuación se ofrecen ejemplos de notaciones permitidas y no permitidas en C#:
  - *tipo*[ ]*nombre*; // Permitida
  - *tipo nombre*[ ]; // No permitida en C#
  - *tipo*[4] *nombre*; // Tampoco permitida en C#

Rango de un arreglo

- El rango se conoce también como dimensión del arreglo
- El número de índices asociados con cada elemento

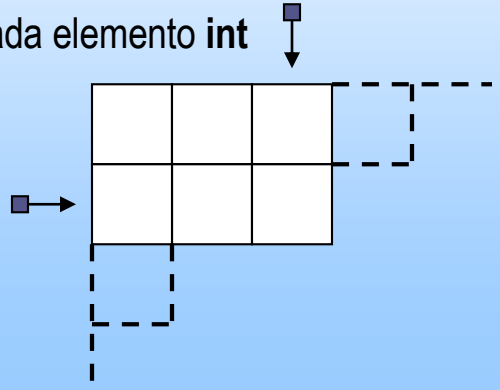
```
long[ ] Fila;
```

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



```
int[ , ] Cuadrícula;
```

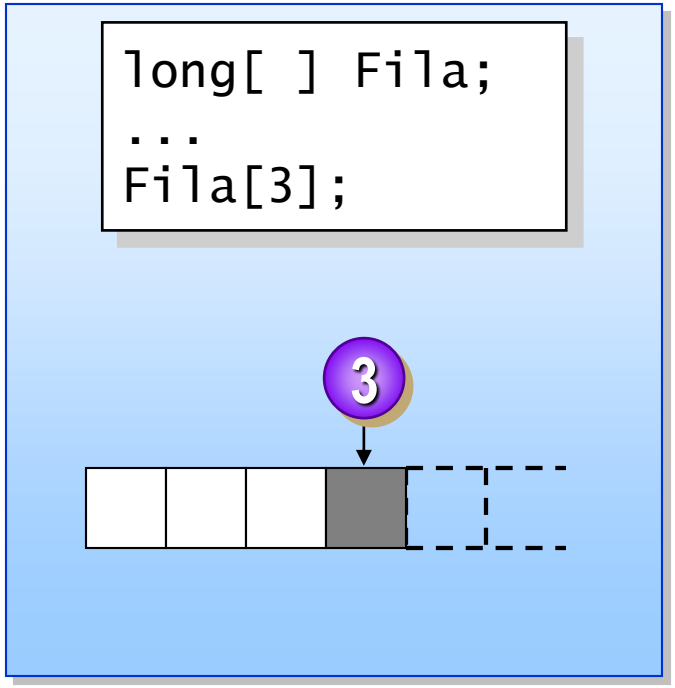
Rango 2: Bidimensional
Dos índices asociados con cada elemento **int**

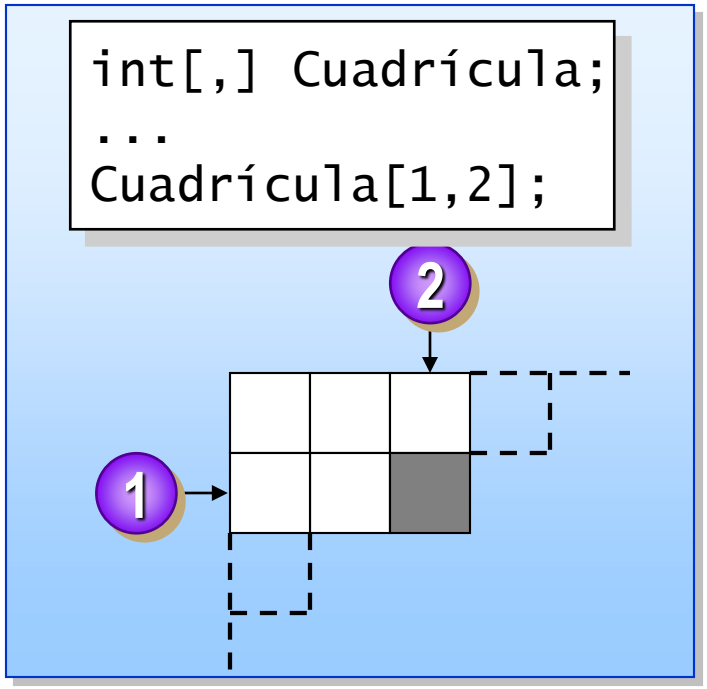


Acceso a los elementos de un arreglo

Se indica un índice entero para cada rango

- Los índices se cuentan a partir de cero

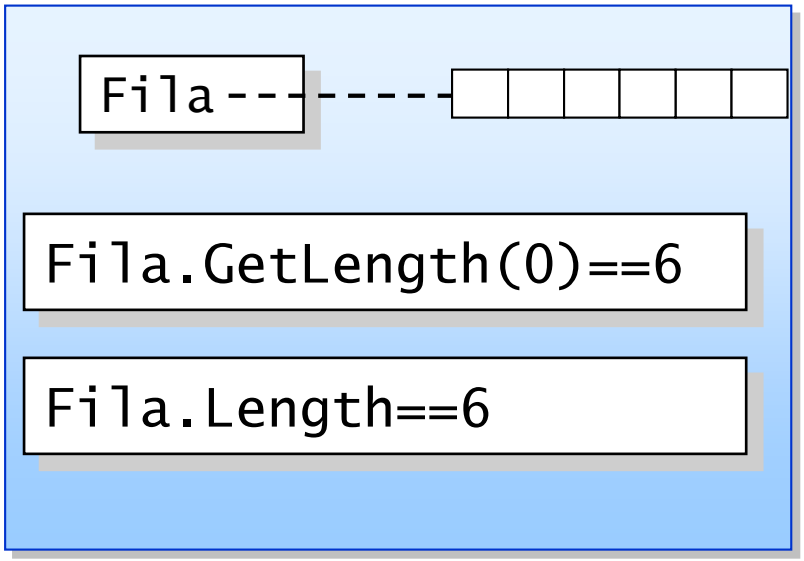


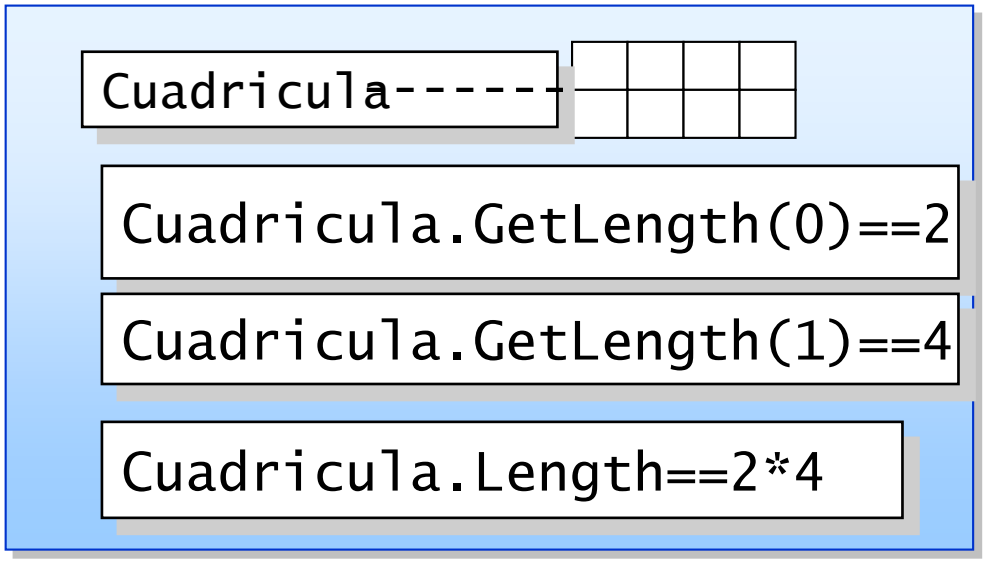


Comprobación de los límites de un arreglo

Se comprueban los límites cada vez que se intenta acceder a un arreglo:

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





Comprobación de los límites de un arreglo

La propiedad **Length,** es la longitud total del arreglo, independientemente de su rango. Para determinar la longitud de una dimensión concreta se puede utilizar el método **GetLength**

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

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

Comparación de arreglo y colecciones

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

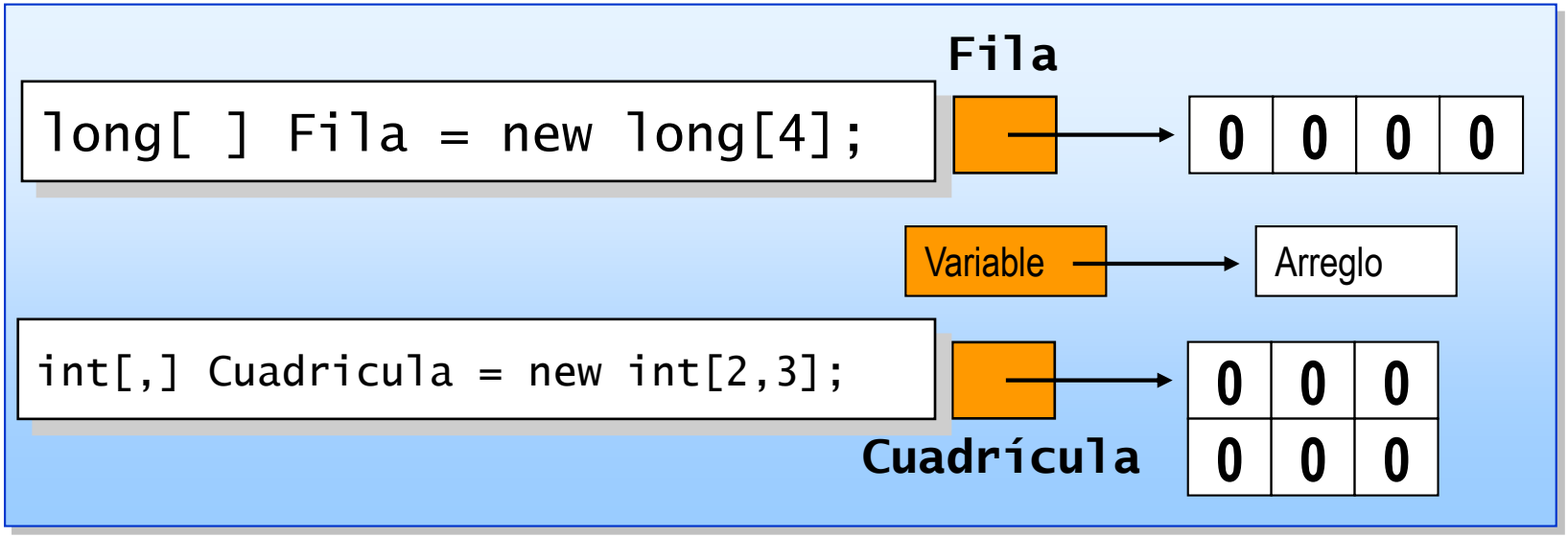
Creación de arreglos

- Creación de un arreglo
- Inicialización de los elementos de un arreglo
- Inicialización de los elementos de un arreglo multidimensional
- Creación de un arreglo de tamaño calculado
- Copia de variables de arreglo

Creación de un arreglo

¡Declarar una variable de arreglo no es lo mismo que crear un arreglo!

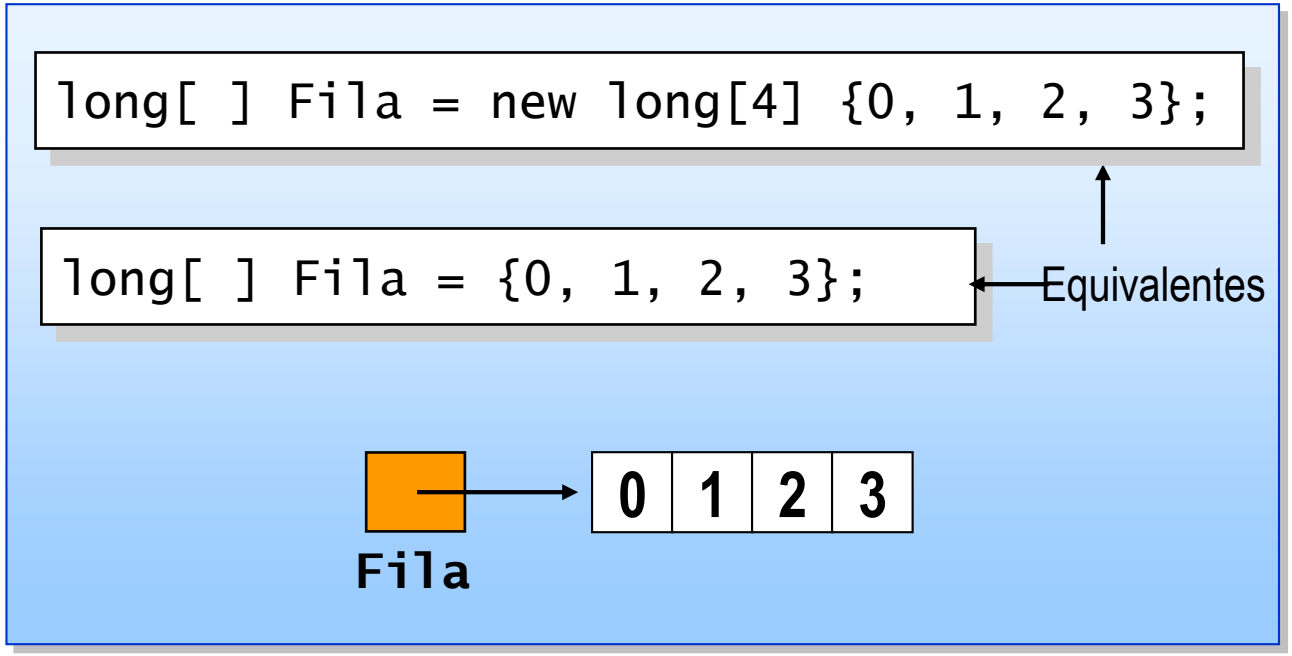
- Para crear el arreglo explícitamente hay que usar **new**
- El valor implícito por defecto de los elementos de un arreglo es cero



Inicialización de los elementos de un arreglo

Es posible inicializar explícitamente los elementos de un arreglo:

- Se puede utilizar una expresión abreviada



Ejemplos

- 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

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

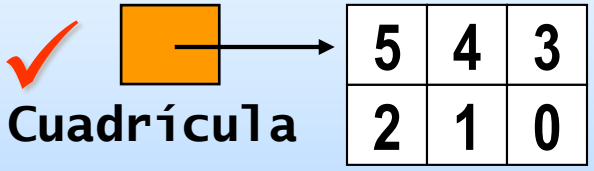
Inicialización de los elementos de un arreglo multidimensional

También se pueden inicializar los elementos de un arreglo multidimensional

- Hay que especificar todos los elementos

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

Nueva tabla int[2,3] implícita




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

Creación de un arreglo de tamaño calculado

No es necesario que el tamaño de un arreglo 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 arreglo especificado por constante entera de tiempo de compilación:

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

  - Tamaño de arreglo especificado por valor entero de tiempo de ejecución:

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

Creación de un arreglo de tamaño calculado

Hay una pequeña restricción, ya que no está permitido usar una expresión en tiempo de ejecución para especificar el tamaño de un arreglo en combinación con inicializador de arreglos:

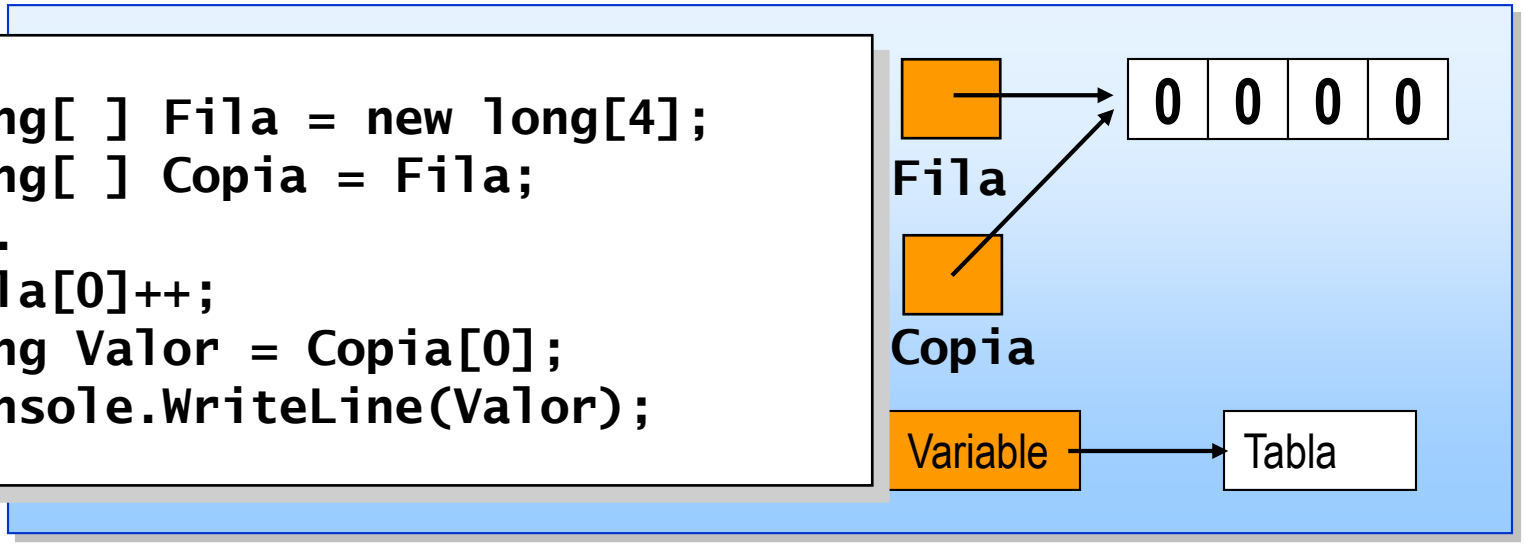
- string s = System.Console.ReadLine( );
- int Tamaño = int.Parse(s);
- int[ ] data = new int[Tamaño]{0,1,2,3}; // No permitido

Copia de variables de arreglos

Al copiar una variable de arreglo se copia sólo la variable de arreglo

- No se copia el arreglo
- Dos variables de arreglo pueden apuntar a el mismo arreglo

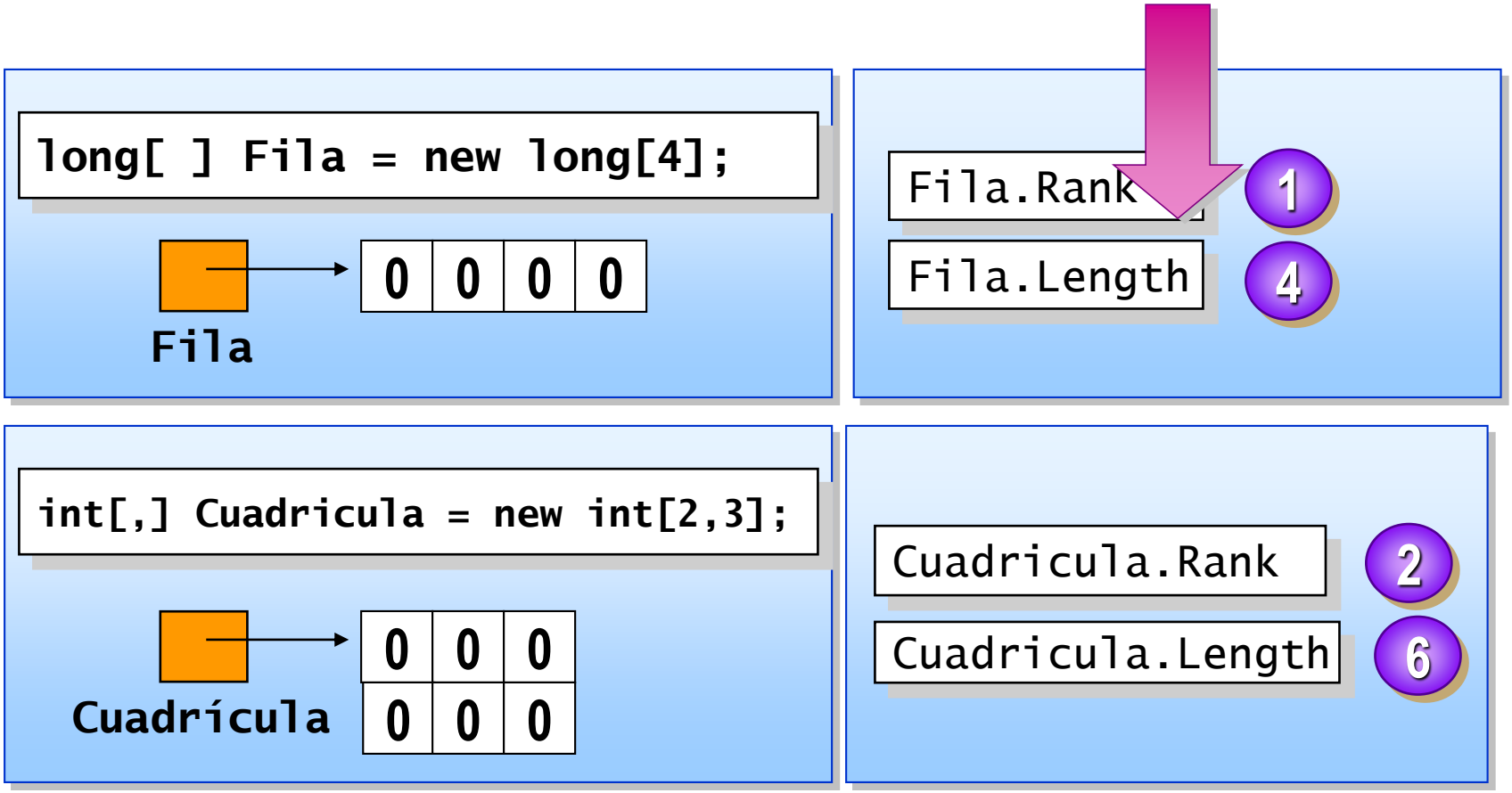
```
long[ ] Fila = new long[4];
long[ ] Copia = Fila;
...
Fila[0]++;
long Valor = Copia[0];
Console.WriteLine(Valor);
```



Uso de arreglos

- Propiedades de arreglos
- Métodos de arreglos
- Devolución de arreglos desde métodos
- Paso de arreglos como parámetros
- Argumentos de línea de comandos
- Demostración: Argumentos para Main
- Uso de arreglos con foreach

Propiedades de arreglos



La propiedad **Rank** es un valor entero de sólo lectura que indica la dimensión de un arreglo.
La propiedad **Length** es un valor entero de sólo lectura que indica la longitud total de un arreglo.

Métodos de arreglos

Métodos utilizados frecuentemente

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

Devolución de arreglos desde métodos

Es posible declarar métodos para que devuelvan arreglos

```
class Example {
    static void Main( ) {
        int[ ] Array = CreateArray(42);
        ...
    }
    static int[ ] CreateArray(int Tamano) {
        int[ ] Arreglo = new int[Tamano];
        return Arreglo;
    }
}
```

Paso de arreglos como parámetros

Un parámetro de arreglo es una copia de la variable de arreglo

- No es una copia del arreglo

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

Este método modificará el arreglo original creada en Main

Argumentos de línea de comandos

El runtime pasa argumentos de línea de comandos a Main

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

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

Uso de arreglos con foreach

La instrucción foreach simplifica enormemente la manipulación de arreglos

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


Demo

Problema #1

- Codificar el siguiente pseudocódigo, que permite:
  - Leer 10 números enteros proporcionados por el usuario, para:
    - Imprimir aquellos números que son superiores a la media de los números proporcionados.

Propuesta Algoritmo Modular

//Objetivo: *Leer valores numéricos, guardarlos en un arreglo e imprimir aquellos que son superiores a la media.*
//Programador: *MC. Gálvez*
//Fecha: ___de Noviembre de 2015

INICIO
//Definición de Constantes y Variables Globales

PRINCIPAL ()
INICIO
//Definición de Constantes y Variables Locales
CONST ENTERO Tamaño=10
ENTERO Numeros[Tamaño]
REAL Media=0
LecturaDatos(Numeros,Tamaño)
Media= CalcularMedia(Numeros,Tamaño)
ImprimirNumerosSuperiores(Numeros,Tamaño,Media)
FIN

Propuesta Algoritmo Modular

```
SINVALOR  LecturaDatos(ENTERO Numeros[], ENTERO Tamaño)
INICIO
     ENTERO Indice
      DESDE(Indice=0;Indice<Tamaño,Indice=Indice+1)
            IMPRIMIR "Proporciona el valor de la celda [", Indice+1, "]:"
            LEER Numeros[Indice]
     FIN_DESDE
     IMPRIMIR "Fin de lectura…"
FIN

REAL  CalcularMedia (ENTERO Numeros[], ENTERO Tamaño)
INICIO
     ENTERO Indice,Suma=0,Media
      DESDE(Indice=0;Indice<Tamaño,Indice=Indice+1)
             Suma=Suma+ Numeros[Indice]
     FIN_DESDE
     Media=Suma/Tamaño
FIN
```

Propuesta Algoritmo Modular

```
SINVALOR ImprimirNumerosSuperiores (ENTERO Numeros[],
      ENTERO Tamaño, REAL Media)
INICIO
    ENTERO Indice
     DESDE(Indice=0;Indice<Tamaño,Indice=Indice+1)
          SI(Numero[Indice]>Media)
               IMPRIMIR  Numeros[Indice]
          FIN_SI
    FIN_DESDE
    IMPRIMIR "Fin de Impresión…"
FIN
```

Problema #2

- Codificar al lenguaje el pseudocódigo Obtenido en la materia de algoritmia, que:
  - Permitía leer 10 valores enteros y guardarlos en un arreglo, luego
    a) Imprimirlos en forma inversa a como fueron proporcionados.
    b) Imprimir el mayor
    c) Imprimir el menor
    d) Determinar si un elemento X se encuentra en el arreglo.

Problema #3

- Codificar el pseudocódigo Obtenido en la materia de algoritmia que permitía:
  - Leer y guardar datos numéricos en un arreglo bidimensional de 2 x 3, y determinar:
    - Cual es el número mayor
    - Cual es el promedio
    - Cuantos números son mayores al promedio
    - Imprimir los valores por renglón
    - Imprimir los valores por columna

Preguntas?