
COLECCIONES
EN C#

Introducción

Muchas veces es necesario agrupar elementos de una misma clase para tratarlos como una única entidad.

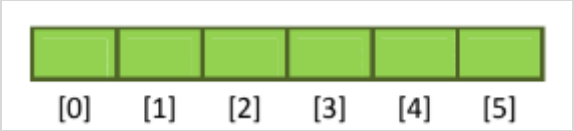
Para eso en C# tenemos las matrices o arrays.

Todos los elementos de un array serán del mismo tipo.

Los arrays son de naturaleza estática. No tendrán un número variable de posiciones una vez incializados.

Matrices unidimensionales - Vectores

- Son arrays organizados en una sola dimensión.




```
Ámbito tipo[] identificador;
identificador = new tipo[cantidad_elementos];
```

```
int[] vectorEnteros;
vectorEnteros = new int[25];

Producto[] productos;
productos = new Producto[25];
```

Arrays primitivos

Cuando usamos tipos de datos “primitivos”(int, bool, float, etc…) cada uno de los elementos se inicializan con el valor por defecto.

Para los enteros del ejemplo sería 0.

Arrays de objetos definidos por nosotros

- Para estos casos que no son primitivos no se pueden instanciar automáticamente.
- Cada uno de los elementos del array harán referencia a null.
- Es necesario inicializar elemento por elemento.

```
Producto[] productos;
productos = new Producto[25];
productos[0] = new Producto(1);
productos[1] = new Producto(2);
```

```
Producto[] productos;
productos = new Producto[25];
for(ulong i =0;i<25; i++)
{
    productos[i] = new Producto(i);
}
```

MATRICES MULTIDIMIENSIONALES

Matrices multidimensionales

- Acá podemos llegar a hablar de matrices en forma de tablas (2 dimensiones), cubos (3 dimensiones) o poliedros (4 o más).
- La forma general de delcararlas es:

```
Ámbito tipo[,,…] identificador;
identificador = new tipo[elem_1°Dimensión,elem_2°Dimensión,… ];
```

```
float [,,] cubo;
cubo = new float[10,3,2];
```

Instanciando una matriz

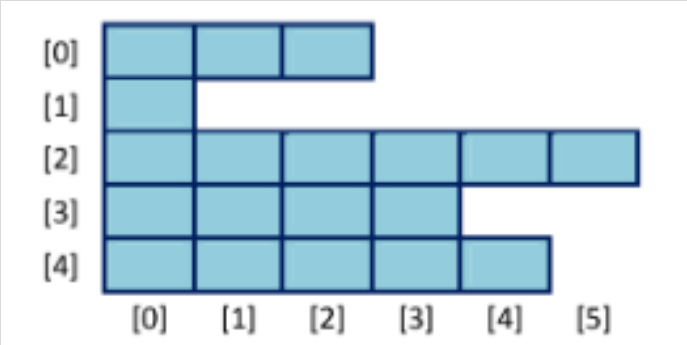
- Recordemos que al ser un Array de objetos no primitivos debemos instanciarlos uno por uno. Podemos hacerlo al momento de su creación

```
Producto[,] tabla;
tabla = new Producto[10, 5];
```

MATRICES IRREGULARES

Matrices irregulares

- Las matrices irregulares o dentadas son arrays que recorriéndolas en un sentido de una o más dimensiones encontraremos diferentes cantidades de elementos.
- Pueden verse con una matriz con más matrices distintas adentro



Instanciando matrices irregulares

- Vamos a probar con una matriz de enteros

```
int[][] jagged;
jagged = new int[5][];
jagged[0] = new int[3];
jagged[1] = new int[1];
jagged[2] = new int[5];
jagged[3] = new int[3];
jagged[4] = new int[4];
```

- De la misma forma podemos trabajar con elementos no primitivos

MÉTODOS DE LA CLASE ARRAY

Acceso a datos

- Para una instancia de la clase Array podemos emplear dos métodos para acceder:
  - vector.SetValue(valor,indices);
  - vector.GetValue(índices)

```
vectorEnteros.SetValue(34, 2);
vectorEnteros.GetValue(2);
```

Determinación de límites

- Para determinar los límites superior e inferior de un array podemos usar las funciones:
- arreglo.GetUpperBound (Dimensión) As Integer
- arreglo.GetLowerBound (Dimensión) As Integer
- (Si se trata de un vector, Dimensión=0)

```
int[] vectorEnteros;
vectorEnteros = new int[25];
vectorEnteros.GetUpperBound(0);
vectorEnteros.GetLowerBound(0);
```

Determinar cantidad de elementos

- Podemos saber la longitud (o cantidad de elementos) con la propiedad Length o el método GetLength(Dimensión).
- si se emplea la propiedad Length, paradeterminar la cantidad de elementos deun arreglo multidimensional, se obtendrá la cantidad total de elementos

```
int[] vectorEnteros;
vectorEnteros = new int[25];
int elementos = vectorEnteros.Length;
int longitud = vectorEnteros.GetLength(0);
```

Ordenamiento de arrays

- Nota: Podemos ordenar arrays de elementos **primitivos**, luego veremos como podemos ordenar clases.
- Usamos el método de clase Array.Sort(array,[posición inicial],[cantidad de elementos])
- Esto va a ordenar los elementos de mayor a menor

```
int[] vectorDesordenado = new int[25];
Random rd = new Random();
for(int i = 0; i < 25; i++)
{
    vectorDesordenado[i] = rd.Next(0, 500);
    Console.WriteLine(vectorDesordenado[i]);
}
Array.Sort(vectorDesordenado);
for (int i = 0; i < vectorDesordenado.Length; i++)
{
    Console.WriteLine(vectorDesordenado[i]);
}
```

Búsqueda binaria en arrays

- La búsqueda binaria es una forma muy eficiente de buscar elementos en un array.
- El problema es que solo se puede usar en arrays ordenados.
- Para usarla usamos el método de clase Array.BinarySearch(Array,[posición inicial],[cantidad de elementos],valor a buscar)
- Este método devuelve la posición del elemento a buscar o -1 si no se encontró nada

```
Array.Sort(vectorDesordenado);
int valorABuscar = 13;
int posicion = Array.BinarySearch(vectorDesordenado, valorABuscar);
if(posicion < 0)
{
    Console.WriteLine("No encontre nada bro");
}
else
{
    Console.WriteLine("Encontrado en la posicion {0}", posicion);
}
```

Búsqueda lineal

La búsqueda lineal es menos eficiente pero no necesitamos ordenar el vector.

Usamos el método Array.IndexOf(Array, elemento) que funciona de la misma manera que BinarySearch

Copiando un array

- Tenemos diferentes formas de copiar o duplicar un vector.
- Puede hacerse uso de los métodosCopyTo () (de instancia), Copy () (de clase) y Clone () (de instancia) de la claseArray.
- Array.Copy(origen, posic_inic_origen, destino, posic_inic_destino , cant_elementos)
- origen.CopyTo (destino, posición_inicial_destino)
- referenciaNoInstanciada = origen.Clone ( )

Recorriendo un vector – el foreach

- Para recorrer un vector podemos usar la estructura foreach () que hará una determinada acción para cada elemento de un vector.
- Esta secuencia es de SOLO LECTURA

```
int[] vectorNuevo = new int[23];
foreach(int elemento in vectorNuevo)
{
    Console.WriteLine(elemento);
}
```