Programación c#

Arrays

Índice

Arrays

Un array es un conjunto de elementos de un mismo tipo,agrupados bajo un único nombre,se accede a cada valor indicando su posición en el conjunto,empezando por el 0.

Declaración

```
int [] numeros = new int [tamaño del array];
```

Acceso a posiciones

```
int [] numeros = new int [tamaño del array];
numeros[0]=valor;
numeros[1]=valor;
numeros[2]=valor;
…
Console.WriteLine(numeros[5]);
```

Dar valores iniciales

Opción A

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

Opción B

```
Int [] numeros = new int [4] { 1, 2, 3, 4};
```

Recorrer los elementos

Opción A

Bucle hasta tamaño puesto a mano,su inconveniente es que si cambias el tamaño te toca cambiarlo en todas las partes del código donde estuviera.

```
Int [] numeros = new int [10];

for (int i = 0; i < 10; i++)
{
      Console.WriteLine(numeros[i]);
}
```

Opción B

Definir una constante con el tamaño.

```
Int TAM = 10;
Int [] numeros = new int [TAM];

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

Opción C

Utilizar la propiedad **Length** del array

```
Int [] numeros = new int [10];

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

Buscar

Para ver si un dato existe,debemos comprobar cada posición del array,una por una,si solo nos interesa saber si aparece o no,una vez lo encuentre acabará el programa,sin embargo,si queremos saber cuantas veces aparece debemos recorrer todo el array.

```
bool encontrado = false;
int i = 0;
while (!encontrado && i < numeros.Length)
{
    if (numeros[i] == 15)
        encontrado = true;
    else
        i++;
}
```

Buscar máximos y mínimos

Ponemos el primer valor del array como máximo/mínimo para evitar darle un valor arbitrario inicial,posteriormente vamos evaluando cada valor del valor en busca de un valor que lo sobrepase,en cuyo caso ocupará su lugar.

```
int maximo = numeros[0];
for (int i = 1; i < numeros.Length; i++)
    if (numeros[i] > maximo)
    {
        maximo = numeros[i];
    }
```

Arrays sobredimensionados

Son arrays definidos con más capacidad de la que realmente necesitan,permiten ir añadiendo o eliminando elementos del array de forma dinámica,usaremos una variable auxiliar para almacenar el número de valores almacenados en el array.

```
string[] textos = new string[100];
int cantidad = 0;
```

Añadir elementos al final

Colocamos el valor en la posición final (la misma que la auxiliar cantidad).

```
if (cantidad < textos.Length)
{
   textos[cantidad] = "Nuevo texto";
   cantidad++;
}
```

Añadir elementos en una posición

Buscamos la posición donde queremos añadir el valor,si hay huecos libres,desplazamos todos los valores posteriores a ese un hueco a la derecha y añadimos el valor en la posición deseada.

```
if (cantidad < textos.Length)
{
    for (int i = cantidad; i >= posicion; i--)
        textos[i] = textos[i-1];
    textos[posicion] = "Hola";
    cantidad++;
}
```

Borrar un elemento de una posición

Buscamos la posición donde queremos borrar(si hay elementos en dicha posición),desplazaremos desde esa posición hasta el final todos un valor hacia atrás para tapar el hueco y decrementamos el contador.

```
if (cantidad > 0 && posicion >= 0 && posicion < cantidad)
{
   for (int i = posicion; i < cantidad - 1; i++)
      numeros[i] = numeros[i+1];
   cantidad--;
```

Arrays bidimensionales

Permite estructurar la información en filas(primera dimensión) y columnas(segunda dimensión) para cada fila.

Alternativas:

Arrays rectangulares (matrices o tablas).

Arrays de arrays.

Arrays rectangulares

Definición

Definimos cuántas filas y columnas tiene el array, separadas por comas.

```
Int [,] tabla = new int [filas, columnas];
```

Acceso

Accedemos a cada posición indicando su fila y columna, separadas por comas.

```
tabla[1, 3] = 12;
Console.WriteLine(tabla[0, 4]);
```

recorrido

Necesitamos un doble bucle anidado (el primero para las filas y el segundo para las columnas, o viceversa).

Usamos la instrucción **GetLength(n)** del array para obtener el tamaño de la dimensión n, y así saber cuándo parar.

array.GetLength(0) para las filas.

array.GetLength(1) para las columnas.

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

Arrays de arrays

Definición

Definimos cuántas filas vamos a crear en una pareja de corchetes.

Para cada fila, indicamos qué tamaño va a tener en la segunda pareja de corchetes.

```
int[][] datos = new int[cantidad arrays][];
datos[0] = new int[tamaño array];
datos[1] = new int[tamaño array];
datos[2] = new int[tamaño array];
```

Acceso

Colocamos en los primeros corchetes la fila que nos interesa, y en los segundos corchetes la columna dentro de esa fila.

```
datos[fila][columna] = valor;
Console.WriteLine(datos[fila][columna]);
```

Recorrido

Usamos un doble bucle anidado, y la propiedad **Length** para saber el tamaño de cada dimensión.

```
// datos.Length = número de filas
for (int i = 0; i < datos.Length; i++)
{
// datos[i].Length = columnas de la fila i
   for (int j = 0; j < datos[i].Length; j++)
      {
            Console.Write(datos[i][j] + " ");
      }
   Console.WriteLine();
}
```

Operaciones avanzadas

Recorrido de arrays con foreach

Útil cuando queremos recorrer TODAS las posiciones del array SIN modificar ninguna (sólo examinando sus valores).

También la podemos aplicar a tablas si no nos interesa saber la posición de los elementos.

```
int[] numeros = new int[10];
int[,] tabla = new int[4, 10];
...
foreach(int numero in numeros)
    Console.WriteLine(numero);

foreach(int numero in tabla)
    Console.WriteLine(numero);
```

Ordenación

Existen varias estrategias para ordenar (ascendente o descendentemente) los datos de un array.

Casi todas se basan en un doble bucle anidado que recorre cada posición y la compara con otras para ver si los datos están desordenados, intercambiándolos en ese caso.

La complejidad de estas operaciones es cuadrática (n2), lo que significa que para ordenar un array de N elementos se hacen en torno a N * N repeticiones en total.

Método de la burbuja

Compara cada elemento de una posición i con la adyacente i+1 y los intercambia si están desordenados.

Al final de este proceso, tendremos ordenado el último elemento del array.

Repetir el proceso deteniéndose cada vez una posición antes.

Se le llama "burbuja" porque los números fluyen por el array de forma similar a las burbujas en una botella.

Cada paso se muestra verticalmente

```
for (int i = 0; i < numeros.Length; i++)
{
    for (int j = 0; j < numeros.Length - i - 1; j++)
    {
        if (numeros[j] > numeros[j+1])
        {
            int auxiliar = numeros[j];
            numeros[j] = numeros[j+1];
            numeros[j+1] = auxiliar;
        }
    }
}
```

Método de intercambio directo

A veces se le confunde con el algoritmo de burbuja.

Para cada posición i del array, se recorre el mismo desde esa posición hasta el final buscando el menor elemento, y se inserta en esa posición i.

Buscamos el menor elemento y lo colocamos en 1ª posición.Buscamos el segundo menor elemento y lo colocamos en 2ª posición.

...

Cada elemento que se encuentre que sea menor que el que actualmente esté en la posición i, se intercambia.

Cada paso se muestra horizontalmente

```
for (int i = 0; i < numeros.Length - 1; i++)
{
    for (int j = i+1; j < numeros.Length; j++)
    {
        if (numeros[i] > numeros[j])
        {
            int auxiliar = numeros[i];
            numeros[i] = numeros[j];
            numeros[j] = auxiliar;
        }
    }
}
```

Ordenación automática

La instrucción **Array.Sort** ordena automáticamente el array que le indiquemos de menor a mayor

Sólo funciona para arrays de tipos simples (enteros, reales, strings...), no para structs, por ejemplo

Búsqueda binaria

La búsqueda secuencial tiene una complejidad lineal O(n)

Significa que, en el peor de los casos, tendremos que hacer n repeticiones para encontrar el dato (si está al final del array)

La búsqueda binaria tiene una complejidad O(log(n)), menor que la lineal

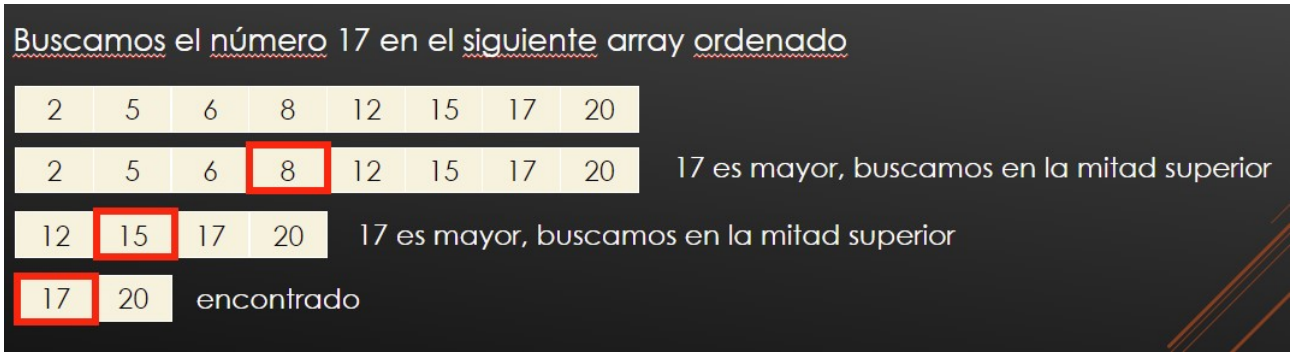
Es adecuada si el array está previamente ordenado (ascendente o descendente)

1-Tomamos el elemento del centro del array.

2-Si es el que estamos buscando, hemos terminado.

3-Si es menor, sabemos que tenemos que buscar en la mitad superior del array.

4-Si es mayor, sabemos que tenemos que buscar en la mitad inferior del array.



Suponemos que buscamos el 15 en un array

```
int[] numeros = new int[50];
// Suponemos un array ya relleno y ordenado
ascendentemente
int mitad, limiteInferior = 0, limiteSuperior =
numeros.Length - 1;
bool encontrado = false;
while (!encontrado && limiteInferior <= limiteSuperior)
{
    mitad = (limiteInferior + limiteSuperior) / 2;
    if (numeros[mitad] == 15)
        encontrado = true;
    else if (numeros[mitad] < 15)
    {
        // Buscamos en mitad superior
        limiteInferior = mitad + 1;
    }
    else
    {
        // Buscamos en mitad inferior
        limiteSuperior = mitad - 1;
    }
}
```