Arrays

1. Introducción

Hasta el momento los tipos de datos que hemos visto ya sean **int, double, bool o char** nos permiten definir variables en las que solamente puede almacenarse un valor en un momento dado. Determinados problemas requieren grandes grupos de datos que formen por sí mismos una colección o estructura. Para poder definir estas estructuras los lenguajes de programación nos ofrecen unos tipos de datos más complejos basados en los tipos de datos simples que existen en el lenguaje.

En este tema estudiaremos los arrays, tipo compuesto (tanto unidimensionales como bidimensionales).

2. Arrays unidimensionales o vectores

Los arrays surgen de la necesidad de tratar conjuntos de datos del mismo tipo relacionados. Tendremos una lista de elementos del mismo tipo. Accederemos a estos datos mediante un nombre común y un índice que indicará qué elemento del array o vector utilizamos.

Un **array** es una estructura homogénea de datos, de tamaño constante y accedemos a sus elementos mediante un identificador común y uno o varios índices.

Así, las características de los arrays son:

- **Homogéneo**, todos sus datos son del mismo tipo.
- **Tamaño constante**: su número de elementos es el mismo a lo largo de toda la ejecución del programa.
- Accedemos a sus elementos mediante el identificador o nombre del array y mediante un índice (debe ser un entero).
- La **dimensión** indica el número de índices necesarios para acceder a un elemento.

Así mediante un array podremos definir: un conjunto de números enteros, o un conjunto de reales, o una cadena de caracteres o una matriz (dos dimensiones) o una lista de registros...

- **Declaración del Vector o Array Unidimensional.**

Para declarar un vector 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.

Al declarar una variable de tabla no se especifica el tamaño de la tabla. Se le dará tamaño con la instrucción **new**.

La sintaxis para declarar un array (de momento vamos a definir un vector de una única dimensión) en Visual C# es la siguiente:

tipo [ ] nombreVector;

- **Tamaño de la tabla.**

El tamaño del vector se da mediante la instrucción new. Se puede hacer en el momento de la declaración o en otro punto del código, pero siempre antes de acceder a los elementos del vector.

tipo [] nombreVector = new tipo [TAMAÑO];

- **Acceso a los componentes del vector.**

Nosotros podemos acceder a los elementos individuales de un array y utilizarlos como si fueran una variable normal, es decir podemos asignarles un valor o recoger su valor, operar con ellos ...

Cada componente del ARRAY se designa utilizando el nombre de la variable del tipo ARRAY seguida de la posición o índice escrito entre corchetes.

nombreVector[índice]

Importante tener en cuenta que los elementos de un vector en Visual C# van desde 0 al número de elementos - 1. Es decir, un vector de tamaño 10 tiene los elementos 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

Existe una función **Length** que nos devuelve el número de elementos que tiene el vector. En el caso anterior nos devolvería 10.

Ejemplo:

Realizar en un botón la lectura de un vector de 5 números enteros y una vez leídos que se impriman por pantalla.

```
            const int TAM = 5;

            // Declaramos y damos tamaño al vector.
            int[] vector = new int[TAM];
            int i;

            // Leemos todos los elementos del vector.
            for (i = 0; i < TAM; i++)
                vector[i] = int.Parse(InputBox("Introduzca el elemento: " + i));

            // Mostramos los elementos del vector.
            for (i = 0; i < TAM; i++)
                MessageBox.Show("Vector[" + i + "] = " + vector[i]);
```

Tener en cuenta que a la hora de recorrer los elementos del vector podemos utilizar **TAM** que es la constante que nos indica el tamaño del vector o bien **vector.Length()**.

Ejemplo de lectura y escritura de un vector utilizando **programación modular**.

```
        const int TAM = 5;

        // Declaramos y damos tamaño al vector.
        int[] vector = new int[TAM];
```

```
        void leerVector(int[] vector)
        {
            int i;

            // Leemos todos los elementos del vector.
            for (i = 0; i < vector.Length; i++)
                vector[i] = int.Parse(InputBox("Introduzca el elemento: " + i));
        }
```

```
        string mostrarVector(int[] vector)
        {
            string texto;
            int i;

            texto = "Elementos del vector: \n";
            for (i = 0; i < vector.Length; i++)
                texto = texto + vector[i] + ", ";

            return texto;
        }
```

```
    private void btnLeer_Click(object sender, EventArgs e)
        {
            leerVector(vector);
        }


        private void btnMostrar_Click(object sender, EventArgs e)
        {
            string texto;
```

```
        texto = mostrarVector(vector);

        MessageBox.Show(texto);
    }
```

Nota: Daros cuenta de que en este y ejemplos posteriores **el vector se declara fuera de los botones**. Esto es así porque leemos el vector en un botón y lo mostramos en otro. Si lo declaráramos dentro del botón su ámbito de utilización sería ese botón y por tanto no podríamos acceder a él desde el botón de escritura.

Sin embargo, en las funciones de leerVector y mostrarVector es importante que **pasemos por parámetros el vector** que queremos utilizar.

Ejemplo en el que tendríamos un vector con las notas de los alumnos de DAW y les aumentaríamos a todos un punto.

```
        const int NUMALUMNOS = 35;
        int[] notasDAW = new int[NUMALUMNOS];

                ...

        for(i = 0; i < NUMALUMNOS; i++)
            notasDAW[i] = notasDAW[i] + 1;
```

Ejemplo. Realizar un programa que lea 10 números enteros y una vez leídos divida la suma total por el número que se introdujo en la posición **pos** siendo pos un número introducido por el usuario.

```
    const int TAM = 10;

    // Declaramos y damos tamaño al vector.
    int[] vector = new int[TAM];
```

```
    void leerVector(int[] vector)
    {
        int i;

        // Leemos todos los elementos del vector.
        for (i = 0; i < vector.Length; i++)
            vector[i] = int.Parse(InputBox("Introduzca el elemento: " + i));
    }
```

```
    string mostrarVector(int[] vector)
    {
        string texto;
        int i;

        texto = "Elementos del vector: \n";
        for (i = 0; i < vector.Length; i++)
            texto = texto + vector[i] + ", ";

        return texto;
    }
```

```
        // Subprograma que devuelve la suma de todos los elementos del vector.
        int sumaVector(int[] vector)
        {
            int suma, i;

            suma = 0;
            for (i = 0; i < vector.Length; i++)
                suma = suma + vector[i];

            return suma;
        }
```

```
        private void btnLeer_Click(object sender, EventArgs e)
        {
            leerVector(vector);
        }
```

```
        private void btnMostrar_Click(object sender, EventArgs e)
        {
            string texto;

            texto = mostrarVector(vector);

            MessageBox.Show(texto);
        }
```

```
        private void btnDividirPos_Click(object sender, EventArgs e)
        {
            int pos, suma;

            pos = int.Parse(InputBox("Introduzca la posición a dividir."));

            if (pos >= 0 && pos < vector.Length)
            {
                suma = sumaVector(vector);
                MessageBox.Show("Resultado: " + suma / vector[pos]);

            }
            else
                MessageBox.Show("La posición sale de los límites del vector.");
        }
```

Es muy importante que, al trabajar con arrays, siempre controlemos **no salirnos de los límites** del vector. Por ejemplo, si al leer el vector en el ejemplo anterior hubiéramos hecho:

```
            for (i = 0; i <= vector.Length; i++)
                vector[i] = int.Parse(InputBox("Introduzca el elemento: " + i));
```

Estaríamos recorriendo desde el elemento 0 hasta el 10, cuando solo tenemos desde el 0 hasta el 9 y nos daría un error de ejecución.

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/arrays/single-dimensional-arrays

3. Métodos de búsqueda y ordenación

En este apartado vamos a estudiar cómo buscar en y cómo ordenar un vector.

Aquí estamos buscando y ordenando un vector de enteros.

- **Búsqueda en un vector.**

La búsqueda de un elemento en un vector consiste en buscar un determinado valor dentro del vector, de forma que si dicho elemento existe devolveremos la posición donde se encuentra.

```
// Función que busca un valor en el vector. Devolverá la posición
// en el vector o bien -1 si no encuentra el valor en el mismo.
int busquedaVector(int[] vector, int valor)
{
    int i, res;
    bool encontrado;

    encontrado = false;
    i = 0;
    while (!encontrado && i < vector.Length)
    {
        if (vector[i] == valor)
            encontrado = true;
        else
            i++;
    }
    if (encontrado)
        res = i;
    else
        res = -1;

    return res;
}
```

- **Ordenación de un vector.**

El algoritmo que vamos a ver a continuación nos permite ordenar un vector de menor a mayor.

```
void ordenarVector(int[] vector)
{
    int i, j;
    int aux, tam;

    tam = vector.Length;
    for(i = 0; i < tam - 1; i ++)
        for (j = i + 1; j < tam; j++)
            if (vector[j] < vector[i])
            {
                aux = vector[i];
                vector[i] = vector[j];
                vector[j] = aux;
            }
}
```

4. Cadenas de caracteres o strings

El manejo de cadenas de caracteres es una tarea frecuente en la implementación de un programa.

Visual C# incluye un tipo de datos específico para tratar cadenas de caracteres denominado tipo string.

Un string puede considerarse un vector de caracteres, pero con varias peculiaridades.

Su cardinalidad es dinámica, es decir, el número de caracteres de una cadena puede variar con la ejecución del programa. Debemos distinguir entre la longitud máxima, y la longitud actual, es decir, el número de caracteres que contiene en un momento dado.

Ya sabemos que TextBox.Text es de tipo STRING.

Utilizamos la palabra reservada **string** para declarar una variable de este tipo.

- **Operaciones con Strings.**

Se puede asignar un string con el operador =.

Recordar que una cadena de caracteres se representa entre comillas dobles: “David González”

Podemos utilizar el operador + para **concatenar** cadenas:

st = “Me llamo “

st = st + “David González”

También podemos **acceder a los caracteres individuales** de un String como si fuera un vector. st[5] valdrá: “a”

cadena.Length nos devuelve el número de caracteres de cadena.

Si queremos comparar alfabéticamente dos cadenas de caracteres podemos utilizar la función string.Compare.

Así: string.Compare(cad1, cad2) devolverá un entero:

- 0 si las dos cadenas son iguales.
- Menor que 0 si cad1 es menor alfabéticamente que cad2
- Mayor que 0 si cad1 es mayor alfabéticamente que cad2.

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/strings/