

4. Arrays, estructuras y cadenas de texto

4.1. Conceptos básicos sobre arrays o tablas

4.1.1. Definición de un array y acceso a los datos

Una tabla, vector, matriz o **array** (que algunos autores traducen por "arreglo") es un conjunto de elementos, todos los cuales son del mismo tipo, y a los que accederemos usando el mismo nombre.

Por ejemplo, si queremos definir un grupo de números enteros, el tipo de datos que usaremos para declararlo será "int []":

```
int[] ejemplo;
```

Cuando sepamos cuántos datos vamos a guardar (por ejemplo 4), podremos reservar espacio con la orden "new", así:

```
ejemplo = new int[4];
```

Si sabemos el tamaño desde el principio, podemos reservar espacio a la vez que declaramos la variable:

```
int[] ejemplo = new int[4];
```

Podemos acceder a cada uno de los valores individuales indicando su nombre (ejemplo) y el número de elemento que nos interesa, pero con una precaución: se empieza a numerar desde 0, así que en el caso anterior tendríamos 4 elementos, que serían ejemplo[0], ejemplo[1], ejemplo[2], ejemplo[3]. Por tanto, podríamos dar el valor 15 al primer elemento de nuestro array así:

```
ejemplo[0] = 15;
```

Habitualmente, como un array representa un conjunto de números, utilizaremos nombres en plural, como en

```
int[] datos = new int[100];
```

Como ejemplo, vamos a definir un grupo de 5 números enteros y hallar su suma:

```
// Ejemplo_04_01_01a.cs
// Primer ejemplo de tablas (arrays)
// Introducción a C#

using System;

public class Ejemplo_04_01_01a
{
    public static void Main()
    {
        int[] numeros = new int[5]; // Un array de 5 números enteros
        int suma; // Un entero que será la suma

        numeros[0] = 200; // Les damos valores
        numeros[1] = 150;
        numeros[2] = 100;
```

```

    numeros[3] = -50;
    numeros[4] = 300;
    suma = numeros[0] +          // Y calculamos la suma
           numeros[1] + numeros[2] + numeros[3] + numeros[4];
    Console.WriteLine("Su suma es {0}", suma);

    // Nota: esta es la forma más ineficiente e incómoda
    // Lo mejoraremos...
}
}

```

Ejercicios propuestos:

Ejercicio propuesto 4.1.1.1: Un programa que pida al usuario 4 números, los memorice (utilizando un array), calcule su media aritmética y después muestre en pantalla la media y los datos tecleados.

Ejercicio propuesto 4.1.1.2: Un programa que pida al usuario 5 números reales (pista: necesitarás un array de "float") y luego los muestre en el orden contrario al que se introdujeron.

4.1.2. Valor inicial de un array

Al igual que ocurría con las variables "normales", podemos dar valor inicial a los elementos de una tabla al principio del programa, si conocemos todos su valores. En este caso, los indicaremos todos entre llaves, separados por comas:

```

// Ejemplo_04_01_02a.cs
// Segundo ejemplo de tablas: valores iniciales con []
// Introducción a C#

using System;

public class Ejemplo_04_01_02a
{
    public static void Main()
    {
        int[] numeros =          // Un array de 5 números enteros
            {200, 150, 100, -50, 300};
        int suma;                // Un entero que será su suma

        suma = numeros[0] +      // Hallamos la suma
              numeros[1] + numeros[2] + numeros[3] + numeros[4];
        Console.WriteLine("Su suma es {0}", suma);
        // Nota: esta forma es algo menos engorrosa, pero todavía no
        // está bien hecho. Lo seguiremos mejorando.
    }
}

```

Nota: el formato completo para la declaración de un array con valores iniciales es éste, incluyendo tanto la orden "new" como los valores entre llaves:

```
int[] numeros = new int[5] {200, 150, 100, -50, 300};
```

Pero, como se ve en el ejemplo anterior, casi siempre se podrá abreviar, y no será necesario usar "new" junto con el tamaño, ya que el compilador lo puede deducir al analizar el contenido del array:

```
int[] numeros = {200, 150, 100, -50, 300};
```

Ejercicios propuestos:

Ejercicio propuesto 4.1.2.1: Un programa que almacene en una tabla el número de días que tiene cada mes (supondremos que es un año no bisiesto), pida al usuario que le indique un mes (1=enero, 12=diciembre) y muestre en pantalla el número de días que tiene ese mes.

4.1.3. Recorriendo los elementos de una tabla

Es de esperar que exista una forma más cómoda de acceder a varios elementos de un array, sin tener siempre que repetirlos todos, como hemos hecho en

```
suma = numeros[0] + numeros[1] + numeros[2] + numeros[3] + numeros[4];
```

El "truco" consistirá en emplear cualquiera de las estructuras repetitivas que ya hemos visto (while, do..while, for), por ejemplo así:

```
suma = 0; // Valor inicial de la suma: 0
for (int i=0; i<=4; i++) // Y calculamos la suma repetitiva
    suma += numeros[i];
```

El fuente completo podría ser así:

```
// Ejemplo_04_01_03a.cs
// Tercer ejemplo de tablas: valores iniciales con llaves
// y recorrido con "for"
// Introducción a C#

using System;

public class Ejemplo_04_01_03a
{
    public static void Main()
    {
        int[] numeros = // Un array de 5 números enteros
            {200, 150, 100, -50, 300};
        int suma; // Un entero que será su suma

        suma = 0; // Valor inicial de la suma: 0
        for (int i=0; i<=4; i++) // Y calculamos la suma repetitiva
            suma += numeros[i];

        Console.WriteLine("Su suma es {0}", suma);
    }
}
```

En este caso, que sólo sumábamos 5 números, no hemos escrito mucho menos, pero si trabajásemos con 100, 500 o 1000 números, la ganancia en comodidad sí que sería evidente.

Lógicamente, también podemos **pedir los datos** al usuario de forma repetitiva, usando un "for", "while" o "do..while":

```
// Ejemplo_04_01_03b.cs
// Cuarto ejemplo de tablas: introducir datos repetitivos
// Introducción a C#

using System;

public class Ejemplo_04_01_03b
{
    public static void Main()
    {
        int[] numeros = new int[5];
        int suma;

        for (int i=0; i<=4; i++)    // Pedimos los datos
        {
            Console.Write("Introduce el dato numero {0}: ", i+1);
            numeros[i] = Convert.ToInt32(Console.ReadLine());
        }

        suma = 0;                // Y calculamos la suma
        for (int i=0; i<=4; i++)
            suma += numeros[i];

        Console.WriteLine("Su suma es {0}", suma);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.1.3.1: Crea un programa que pida al usuario 6 números enteros cortos y luego los muestre en orden inverso (pista: usa un array para almacenarlos y "for" para mostrarlos).

Ejercicio propuesto 4.1.3.2: Crea un programa que pregunte al usuario cuántos números enteros va a introducir (por ejemplo, 10), le pida todos esos números, los guarde en un array y finalmente calcule y muestre la media de esos números.

Ejercicio propuesto 4.1.3.3: Un programa que pida al usuario 10 reales de doble precisión, calcule su media y luego muestre los que están por encima de la media.

Ejercicio propuesto 4.1.3.4: Un programa que almacene en una tabla el número de días que tiene cada mes (de un año no bisiesto), pida al usuario que le indique un mes (ej. 2 para febrero) y un día (ej. el día 15) y diga qué número de día es dentro del año (por ejemplo, el 15 de febrero sería el día número 46, el 31 de diciembre sería el día 365).

Ejercicio propuesto 4.1.3.5: A partir del ejercicio anterior, crea otro que pida al usuario que le indique la fecha, formada por día (1 al 31) y el mes (1=enero, 12=diciembre), y como respuesta muestre en pantalla el número de días que quedan hasta final de año.

Ejercicio propuesto 4.1.3.6: Un programa que pida 10 nombres y los memorice (pista: esta vez se trata de un array de "string"). Después deberá pedir que se teclee un nombre y dirá si se encuentra o no entre los 10 que se han tecleado antes. Volverá a pedir otro nombre y a decir si se encuentra entre ellos, y así sucesivamente hasta que se teclee "fin".

Ejercicio propuesto 4.1.3.7: Un programa que prepare espacio para guardar un máximo de 100 nombres. El usuario deberá ir introduciendo un nombre cada vez, hasta que se pulse Intro sin teclear nada, momento en

el que dejarán de pedirse más nombres y se mostrará en pantalla la lista de los nombres que se han introducido.

Ejercicio propuesto 4.1.3.8: Un programa que reserve espacio para un vector de 3 componentes, pida al usuario valores para dichas componentes (por ejemplo [2, -5, 7]) y muestre su módulo (raíz cuadrada de la suma de sus componentes al cuadrado).

Ejercicio propuesto 4.1.3.9: Un programa que reserve espacio para dos vectores de 3 componentes, pida al usuario sus valores y calcule la suma de ambos vectores (su primera componente será x_1+y_1 , la segunda será x_2+y_2 y así sucesivamente).

Ejercicio propuesto 4.1.3.10: Un programa que reserve espacio para dos vectores de 3 componentes, pida al usuario sus valores y calcule su producto escalar ($x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3$).

Ejercicio propuesto 4.1.3.11: Un programa que pida al usuario 4 números enteros y calcule (y muestre) cuál es el mayor de ellos. Nota: para calcular el mayor valor de un array, hay que comparar cada uno de los valores que tiene almacenados el array con el que hasta ese momento es el máximo provisional. El valor inicial de este máximo provisional no debería ser cero (porque el resultado sería incorrecto si todos los números son negativos), sino el primer elemento del array.

4.1.4. Operaciones habituales con arrays: buscar, añadir, insertar, borrar

Algunas operaciones con datos pertenecientes a un array son especialmente frecuentes: **buscar** si existe un cierto dato, **añadir** un dato al final de los existentes, **insertar** un dato entre dos que ya hay, **borrar** uno de los datos almacenados, etc. Por eso, vamos a ver las pautas básicas para realizar estas operaciones, y un “fuente” de ejemplo.

Para **ver si un dato existe**, habrá que recorrer todo el array, comparando el valor almacenado con el dato que se busca. Puede interesarnos simplemente saber si está o no (con lo que se podría interrumpir la búsqueda en cuanto aparezca una primera vez) o ver en qué posiciones se encuentra (para lo que habría que recorrer todo el array). Si el array estuviera ordenado, se podría buscar de una forma más rápida, pero la veremos más adelante.

Para encontrar **el máximo o el mínimo** de los datos, tomaremos el primero de los datos como valor provisional, y compararemos con cada uno de los demás, para ver si está por encima o debajo de ese máximo o mínimo provisional, y cambiarlo si fuera necesario.

Para poder **añadir** un dato al final de los ya existentes, necesitamos que el array no esté completamente lleno, y llevar un contador de cuántas posiciones hay ocupadas, de modo que seamos capaces de guardar el dato en la primera posición libre.

Para **insertar** un dato en una cierta posición, los que queden detrás deberán desplazarse "hacia la derecha" para dejarle hueco. Este movimiento debe empezar desde el final para que cada dato que se mueve no destruya el que estaba a continuación de él. También habrá que actualizar el contador, para indicar que queda una posición libre menos.

Si se quiere **borrar** el dato que hay en una cierta posición, los que estaban a continuación deberán desplazarse "hacia la izquierda" para que no queden huecos. Como en el caso anterior, habrá que actualizar el contador, pero ahora para indicar que queda una posición libre más.

Vamos a verlo con un ejemplo:

```
// Ejemplo_04_01_04a.cs
// Añadir y borrar en un array
// Introducción a C#

using System;

public class Ejemplo_04_01_04a
{
    public static void Main()
    {
        int[] datos = {10, 15, 12, 0, 0};

        int capacidad = 5;           // Capacidad maxima del array
        int cantidad = 3;             // Número real de datos guardados

        int i;                       // Para recorrer los elementos

        // Mostramos el array
        for (i=0; i<cantidad; i++)
            Console.Write("{0} ", datos[i]);
        Console.WriteLine();

        // Buscamos el dato "15"
        for (i=0; i<cantidad; i++)
            if (datos[i] == 15)
                Console.WriteLine("15 encontrado en la posición {0} ", i+1);

        // Buscamos el máximo
        int maximo = datos[0];
        for (i=1; i<cantidad; i++)
            if (datos[i] > maximo)
                maximo = datos[i];
        Console.WriteLine("El máximo es {0} ", maximo);

        // Añadimos un dato al final
        Console.WriteLine("Añadiendo 6 al final");
        if (cantidad < capacidad)
        {
            datos[cantidad] = 6;
            cantidad++;
        }

        // Y volvemos a mostrar el array
        for (i=0; i<cantidad; i++)
            Console.Write("{0} ", datos[i]);
        Console.WriteLine();

        // Borramos el segundo dato
        Console.WriteLine("Borrando el segundo dato");
        int posicionBorrar = 1;
        for (i=posicionBorrar; i<cantidad-1; i++)
            datos[i] = datos[i+1];
        cantidad--;

        // Y volvemos a mostrar el array
        for (i=0; i<cantidad; i++)
            Console.Write("{0} ", datos[i]);
        Console.WriteLine();

        // Insertamos 30 en la tercera posición
        if (cantidad < capacidad)
```

```

    {
        Console.WriteLine("Insertando 30 en la posición 3");
        int posicionInsertar = 2;
        for (i=cantidad; i>posicionInsertar; i--)
            datos[i] = datos[i-1];
        datos[posicionInsertar] = 30;
        cantidad++;
    }

    // Y volvemos a mostrar el array
    for (i=0; i<cantidad; i++)
        Console.Write("{0} ",datos[i]);
    Console.WriteLine();
}

```

que tendría como resultado:

```

10 15 12
15 encontrado en la posición 2
El máximo es 15
Añadiendo 6 al final
10 15 12 6
Borrando el segundo dato
10 12 6
Insertando 30 en la posición 3
10 12 30 6

```

Este programa "no dice nada" cuando no se encuentra el dato que se está buscando. Se puede mejorar usando una variable "booleana" que nos sirva de testigo, de forma que al final nos avise si el dato no existía (no sirve emplear un "else", porque en cada pasada del bucle "for" no sabemos si el dato no existe, sólo sabemos que no está en la posición actual).

Ejercicios propuestos:

Ejercicio propuesto 4.1.4.1: Crea una variante del ejemplo anterior (04_01_04a) que pida al usuario el dato a buscar, avise si ese dato no aparece, y que diga cuántas veces se ha encontrado en caso contrario.

Ejercicio propuesto 4.1.4.2: Crea una variante del ejemplo anterior (04_01_04a) que añada un dato introducido por el usuario al final de los datos existentes.

Ejercicio propuesto 4.1.4.3: Crea una variante del ejemplo anterior (04_01_04a) que inserte un dato introducido por el usuario en la posición que elija el usuario. Debe avisar si la posición escogida es incorrecta (porque esté más allá del final de los datos).

Ejercicio propuesto 4.1.4.4: Crea una variante del ejemplo anterior (04_01_04a) que borre el dato que se encuentre en la posición que elija el usuario. Debe avisar si la posición escogida no es válida.

Ejercicio propuesto 4.1.4.5: Crea un programa que prepare espacio para un máximo de 10 nombres. Deberá mostrar al usuario un menú que le permita realizar las siguientes operaciones:

- Añadir un dato al final de los ya existentes.
- Insertar un dato en una cierta posición (como ya se ha comentado, los que queden detrás deberán desplazarse "a la derecha" para dejarle hueco; por ejemplo, si el array contiene "hola", "adios" y se pide insertar "bien" en la segunda posición, el array pasará a contener "hola", "bien", "adios".

- *Borrar el dato que hay en una cierta posición (como se ha visto, lo que estaban detrás deberán desplazarse "a la izquierda" para que no haya huecos; por ejemplo, si el array contiene "hola", "bien", "adios" y se pide borrar el dato de la segunda posición, el array pasará a contener "hola", "adios"*
- *Mostrar los datos que contiene el array.*
- *Salir del programa.*

4.1.5. Constantes

En ocasiones, manejaremos valores que realmente no van a variar. Es el caso del tamaño máximo de un array. En esos casos, por legibilidad y por facilidad de mantenimiento del programa, puede ser preferible no usar el valor numérico, sino una variable. Dado que el valor de ésta no debería cambiar, podemos usar la palabra "const" para indicar que debe ser constante, y el compilador no permitirá que la modifiquemos por error. Además, por convenio, para que sea fácil distinguir una constante de una variable, se suele escribir su nombre totalmente en mayúsculas:

```
const int MAXIMO = 5;
```

Así, una nueva versión del fuente del apartado 4.1.3 (b), usando una constante para la capacidad del array podría ser:

```
// Ejemplo_04_01_05a.cs
// Quinto ejemplo de tablas: constantes
// Introducción a C#

using System;

public class Ejemplo_04_01_05a
{
    public static void Main()
    {
        const int MAXIMO = 5;           // Cantidad de datos
        int[] numeros = new int[MAXIMO];
        int suma;

        for (int i=0; i<MAXIMO; i++)    // Pedimos los datos
        {
            Console.Write("Introduce el dato numero {0}: ", i+1);
            numeros[i] = Convert.ToInt32(Console.ReadLine());
        }

        suma = 0;                       // Y calculamos la suma
        for (int i=0; i<MAXIMO; i++)
            suma += numeros[i];

        Console.WriteLine("Su suma es {0}", suma);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.1.5.1: *Crea un programa que contenga un array con los nombres de los meses del año. El usuario podrá elegir entre verlos en orden natural (de Enero a Diciembre) o en orden inverso (de Diciembre a Enero). Usa constantes para el valor máximo del array en ambos recorridos.*

4.2. Arrays bidimensionales

Podemos declarar tablas de **dos o más dimensiones**. Por ejemplo, si un profesor quiere guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 personas, tendría dos opciones:

- Se puede usar `int datosAlumnos[40]` y entonces debemos recordar que los 20 primeros datos corresponden realmente a un grupo de alumnos y los 20 siguientes a otro grupo. Es "demasiado artesanal", así que no daremos más detalles.
- O bien podemos emplear `int datosAlumnos[2,20]` y entonces sabemos que los datos de la forma `datosAlumnos[0,i]` son los del primer grupo, y los `datosAlumnos[1,i]` son los del segundo.
- Una alternativa, que puede sonar más familiar a quien ya haya programado en C es emplear `int datosAlumnos[2][20]` pero en C# esto no tiene exactamente el mismo significado que `[2,20]`, sino que se trata de dos arrays, cuyos elementos a su vez son arrays de 20 elementos. De hecho, podrían ser incluso dos arrays de distinto tamaño, como veremos en el segundo ejemplo.

En cualquier caso, si queremos indicar valores iniciales, lo haremos entre llaves, igual que si fuera una tabla de una única dimensión.

Vamos a ver un primer ejemplo de uso con **arrays "rectangulares"**, de la forma `[2,20]`, lo que podríamos llamar el "estilo Pascal". En este ejemplo usaremos tanto arrays con valores prefijados, como arrays para los que reservemos espacio con "new" y a los que demos valores en un segundo paso:

```
// Ejemplo_04_02a.cs
// Array de dos dimensiones "rectangulares" (estilo Pascal)
// Introducción a C#

using System;

public class Ejemplo_04_02a
{
    public static void Main()
    {
        int[, ] notas1 = new int[2,2]; // 2 bloques de 2 datos
        notas1[0,0] = 1;
        notas1[0,1] = 2;
        notas1[1,0] = 3;
        notas1[1,1] = 4;

        int[, ] notas2 = // 2 bloques de 10 datos, prefijados
        {
            {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
            {11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
        };

        Console.WriteLine("La nota1 del segundo alumno del grupo 1 es {0}",
            notas1[0,1]);
        Console.WriteLine("La nota2 del tercer alumno del grupo 1 es {0}",
            notas2[0,2]);
    }
}
```

Este tipo de tablas de varias dimensiones son las que se usan también para guardar matrices, cuando se trata de resolver problemas matemáticos más complejos que los que hemos visto hasta ahora. Si ya has estudiado la teoría de matrices, más adelante tienes algunos ejercicios propuestos para aplicar esos conocimientos al uso de arrays bidimensionales.

La otra forma de tener arrays multidimensionales son los "arrays de arrays", que, como ya hemos comentado, y como veremos en este ejemplo, pueden tener elementos de distinto tamaño. En ese caso nos puede interesar saber su **longitud**, para lo que podemos usar "a.Length":

```
// Ejemplo_04_02b.cs
// Array de arrays (array de dos dimensiones al estilo C)
// Introducción a C#

using System;

public class Ejemplo_04_02b
{
    public static void Main()
    {
        int[][] notas;           // Array de dos dimensiones
        notas = new int[3][];    // Seran 3 bloques de datos
        notas[0] = new int[10];  // 10 notas en un grupo
        notas[1] = new int[15];  // 15 notas en otro grupo
        notas[2] = new int[12];  // 12 notas en el ultimo

        // Damos valores de ejemplo
        for (int i=0;i<notas.Length;i++)
        {
            for (int j=0;j<notas[i].Length;j++)
            {
                notas[i][j] = i + j;
            }
        }

        // Y mostramos esos valores
        for (int i=0;i<notas.Length;i++)
        {
            for (int j=0;j<notas[i].Length;j++)
            {
                Console.Write(" {0}", notas[i][j]);
            }
            Console.WriteLine();
        }

    } // Fin de "Main"
}
```

La salida de este programa sería

```
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 3 4 5 6 7 8 9 10 11 12 13
```

Ejercicios propuestos:

Ejercicio propuesto 4.2.1: Un programa que pida al usuario dos bloques de 10 números enteros (usando un array de dos dimensiones). Después deberá mostrar el mayor dato que se ha introducido en cada uno de ellos.

Ejercicio propuesto 4.2.2: Un programa que pida al usuario dos bloques de 6 cadenas de texto. Después pedirá al usuario una nueva cadena y comprobará si aparece en alguno de los dos bloques de información anteriores.

Ejercicio propuesto 4.2.3: Un programa que pregunte al usuario el tamaño que tendrán dos bloques de números enteros (por ejemplo, uno de 10 elementos y otro de 12). Luego pedirá los datos para ambos bloques de datos. Finalmente deberá mostrar el mayor dato que se ha introducido en cada uno de ellos.

Si has estudiado **álgebra de matrices**:

Ejercicio propuesto 4.2.4: Un programa que calcule el determinante de una matriz de 2x2.

Ejercicio propuesto 4.2.5: Un programa que calcule el determinante de una matriz de 3x3.

Ejercicio propuesto 4.2.6: Un programa que calcule si las filas de una matriz son linealmente dependientes.

Ejercicio propuesto 4.2.7: Un programa que use matrices para resolver un sistema de ecuaciones lineales mediante el método de Gauss.

4.3. Estructuras o registros

4.3.1. Definición y acceso a los datos

Un **registro** es una agrupación de datos, llamados **campos**, los cuales no necesariamente son del mismo tipo. Se definen con la palabra "**struct**".

En C# (al contrario que en C), primero deberemos declarar cual va a ser la estructura de nuestro registro, lo que no se puede hacer dentro de "Main". Más adelante, ya dentro de "Main", podremos declarar variables de ese nuevo tipo.

Los datos que forman un "struct" pueden ser públicos o privados. Por ahora, a nosotros nos interesará que sean accesibles desde el resto de nuestro programa, por lo que siempre les añadiremos delante la palabra "public" para indicar que queremos que sean públicos.

Ya desde el cuerpo del programa, para acceder a cada uno de los datos que forman el registro, tanto si queremos leer su valor como si queremos cambiarlo, se debe indicar el nombre de la variable y el del dato (o campo) separados por un punto:

```
// Ejemplo_04_03_01a.cs
// Registros (struct)
// Introducción a C#

using System;

public class Ejemplo_04_03_01a
{
    struct tipoPersona
    {
        public string nombre;
        public char inicial;
        public int edad;
        public float nota;
    }

    public static void Main()
    {
        tipoPersona persona;

        persona.nombre = "Juan";
    }
}
```

```
        persona.inicial = 'J';
        persona.edad = 20;
        persona.nota = 7.5f;
        Console.WriteLine("La edad de {0} es {1}",
            persona.nombre, persona.edad);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.3.1.1: Crea un "struct" que almacene datos de una canción en formato MP3: Artista, Título, Duración (en segundos), Tamaño del fichero (en KB). Un programa debe pedir los datos de una canción al usuario, almacenarlos en dicho "struct" y después mostrarlos en pantalla.

4.3.2. Arrays de structs

Hemos guardado varios datos de una persona. Se pueden almacenar los de **varias personas** si combinamos el uso de los "struct" con las tablas (arrays) que vimos anteriormente. Por ejemplo, si queremos guardar los datos de 100 personas podríamos hacer:

```
// Ejemplo_04_03_02a.cs
// Array de struct
// Introducción a C#

using System;

public class Ejemplo_04_03_02a
{
    struct tipoPersona
    {
        public string nombre;
        public char inicial;
        public int edad;
        public float nota;
    }

    public static void Main()
    {
        tipoPersona[] personas = new tipoPersona[100];

        personas[0].nombre = "Juan";
        personas[0].inicial = 'J';
        personas[0].edad = 20;
        personas[0].nota = 7.5f;
        Console.WriteLine("La edad de {0} es {1}",
            personas[0].nombre, personas[0].edad);

        personas[1].nombre = "Pedro";
        Console.WriteLine("La edad de {0} es {1}",
            personas[1].nombre, personas[1].edad);
    }
}
```

La inicial de la primera persona sería "personas[0].inicial", y la edad del último sería "personas[99].edad".

Al probar este programa obtenemos

```
La edad de Juan es 20
La edad de Pedro es 0
```

porque cuando reservamos espacio para los elementos de un "array" usando "new", sus valores se dejan "vacíos" (0 para los números, cadenas vacías para las cadenas de texto).

Ejercicios propuestos:

Ejercicio propuesto 4.3.2.1: Amplia el programa del ejercicio 4.3.1.1, para que almacene datos de hasta 100 canciones. Deberá tener un menú que permita las opciones: añadir una nueva canción, mostrar el título de todas las canciones, buscar la canción que contenga un cierto texto (en el artista o en el título).

Ejercicio propuesto 4.3.2.2: Crea un programa que permita guardar datos de "imágenes" (ficheros de ordenador que contengan fotografías o cualquier otro tipo de información gráfica). De cada imagen se debe guardar: nombre (texto), ancho en píxeles (por ejemplo 2000), alto en píxeles (por ejemplo, 3000), tamaño en Kb (por ejemplo 145,6). El programa debe ser capaz de almacenar hasta 700 imágenes (deberá avisar cuando su capacidad esté llena). Debe permitir las opciones: añadir una ficha nueva, ver todas las fichas (número y nombre de cada imagen), buscar la ficha que tenga un cierto nombre.

4.3.3. structs anidados

Podemos encontrarnos con un registro que tenga varios datos, y que a su vez ocurra que uno de esos datos esté formado por varios datos más sencillos. Por ejemplo, una fecha de nacimiento podría estar formada por día, mes y año. Para hacerlo desde C#, incluiríamos un "struct" dentro de otro, así:

```
// Ejemplo_04_03_03a.cs
// Registros anidados
// Introducción a C#

using System;

public class Ejemplo_04_03_03a
{
    struct fechaNacimiento
    {
        public int dia;
        public int mes;
        public int anyo;
    }

    struct tipoPersona
    {
        public string nombre;
        public char inicial;
        public fechaNacimiento diaDeNacimiento;
        public float nota;
    }

    public static void Main()
    {
        tipoPersona persona;

        persona.nombre = "Juan";
```

```

        persona.inicial = 'J';
        persona.diaDeNacimiento.dia = 15;
        persona.diaDeNacimiento.mes = 9;
        persona.nota = 7.5f;
        Console.WriteLine("{0} nació en el mes {1}",
            persona.nombre, persona.diaDeNacimiento.mes);
    }
}

```

Ejercicios propuestos:

Ejercicio propuesto 4.3.3.1: Amplia el programa 4.3.2.1, para que el campo "duración" se almacene como minutos y segundos, usando un "struct" anidado que contenga a su vez estos dos campos.

4.4. Cadenas de caracteres

4.4.1. Definición. Lectura desde teclado

Hemos visto cómo leer cadenas de caracteres (Console.ReadLine) y cómo mostrarlas en pantalla (Console.Write), así como la forma de darles un valor (=) y de comparar cuál es su valor (==).

Vamos a repasar todas esas posibilidades, junto con la de formar una cadena a partir de otras si las unimos con el símbolo de la suma (+), lo que llamaremos "**concatenar**" cadenas. Un ejemplo que nos pidiese nuestro nombre y nos saludase usando todo ello podría ser:

```

// Ejemplo_04_04_01a.cs
// Cadenas de texto (1: manejo básico)
// Introducción a C#

using System;

public class Ejemplo_04_04_01a
{
    public static void Main()
    {
        string saludo = "Hola";
        string segundoSaludo;
        string nombre, despedida;

        segundoSaludo = "Que tal?";
        Console.WriteLine("Dime tu nombre... ");
        nombre = Console.ReadLine();

        Console.WriteLine("{0} {1}", saludo, nombre);
        Console.WriteLine(segundoSaludo);

        if (nombre == "Alberto")
            Console.WriteLine("Dices que eres Alberto?");
        else
            Console.WriteLine("Así que no eres Alberto?");

        despedida = "Adios " + nombre + "!";
        Console.WriteLine(despedida);
    }
}

```

Ejercicios propuestos:

Ejercicio propuesto 4.4.1.1: Crea un programa que te pida tu nombre y lo escriba 5 veces.

Ejercicio propuesto 4.4.1.2: Crea un programa que pida al usuario su nombre. Si se llama como tú (por ejemplo, "Pepe"), responderá "Bienvenido, jefe". En caso contrario, le saludará por su nombre.

Ejercicio propuesto 4.4.1.3: Un programa que pida tu nombre, tu día de nacimiento y tu mes de nacimiento y lo junte todo en una cadena, separando el nombre de la fecha por una coma y el día del mes por una barra inclinada, así: "Juan, nacido el 31/12".

Ejercicio propuesto 4.4.1.4: Crea un programa que pida al usuario dos números enteros y después una operación que realizar con ellos. La operación podrá ser "suma", "resta", "multiplicación" y "división", que también se podrán escribir de forma abreviado con los operadores matemáticos "+", "-", "*" y "/". Para multiplicar también se podrá usar una "x", minúscula o mayúscula. A continuación se mostrará el resultado de esa operación (por ejemplo, si los números son 3 y 6 y la operación es "suma", el resultado sería 9). La operación debes tomarla como una cadena de texto y analizarla con un "switch".

4.4.2. Cómo acceder a las letras que forman una cadena

Podemos leer una de las letras de una cadena, de igual forma que leemos los elementos de cualquier array: si la cadena se llama "texto", el primer elemento será texto[0], el segundo será texto[1] y así sucesivamente.

Eso sí, las cadenas en C# no se pueden modificar letra a letra: no podemos hacer texto[0]='a'. Para eso habrá que usar una construcción auxiliar, que veremos más adelante.

```
// Ejemplo_04_04_02a.cs
// Cadenas de texto (2: acceder a una letra)
// Introducción a C#

using System;

public class Ejemplo_04_04_02a
{
    public static void Main()
    {
        string saludo = "Hola";
        Console.WriteLine("La tercera letra de {0} es {1}",
            saludo, saludo[2]);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.2.1: Crea un programa que pregunte al usuario su nombre y le responda cuál es su inicial.

4.4.3. Longitud de la cadena

Podemos saber cuántas letras forman una cadena con "cadena.Length". Esto permite que podamos recorrer la cadena letra por letra, usando construcciones como "for".

```
// Ejemplo_04_04_03a.cs
// Cadenas de texto (3: longitud)
// Introducción a C#

using System;

public class Ejemplo_04_04_03a
{
    public static void Main()
    {
        string saludo = "Hola";
        int longitud = saludo.Length;
        Console.WriteLine("La longitud de {0} es {1}", saludo, longitud);
        for (int i=0; i<longitud; i++)
        {
            Console.WriteLine("La letra {0} es {1}", i, saludo[i]);
        }
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.3.1: Un programa que te pida tu nombre y lo muestre en pantalla separando cada letra de la siguiente con un espacio. Por ejemplo, si tu nombre es "Juan", debería aparecer en pantalla "J u a n".

Ejercicio propuesto 4.4.3.2: Un programa que pida una frase al usuario y la muestre en orden inverso (de la última letra a la primera).

Ejercicio propuesto 4.4.3.3: Un programa que pida al usuario una frase, después una letra y finalmente diga si aparece esa letra como parte de esa frase o no.

Ejercicio propuesto 4.4.3.4: Un programa capaz de sumar dos números enteros muy grandes (por ejemplo, de 30 cifras), que se deberán pedir como cadena de texto y analizar letra a letra.

Ejercicio propuesto 4.4.3.5: Un programa capaz de multiplicar dos números enteros muy grandes (por ejemplo, de 30 cifras), que se deberán pedir como cadena de texto y analizar letra a letra.

4.4.4. Extraer una subcadena

Podemos extraer parte del contenido de una cadena con "Substring", que recibe dos parámetros: la posición a partir de la que queremos empezar y la cantidad de caracteres que queremos obtener. El resultado será otra cadena:

```
saludo = frase.Substring(0,4);
```

Podemos omitir el segundo número, y entonces se extraerá desde la posición indicada hasta el final de la cadena.

```
// Ejemplo_04_04_04a.cs
// Cadenas de texto (4: substring)
// Introducción a C#

using System;

public class Ejemplo_04_04_04a
{

```



```
public static void Main()
{
    string saludo = "Hola";
    string subcadena = saludo.Substring(1,3);
    Console.WriteLine("Una subcadena de {0} es {1}",
        saludo, subcadena);
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.4.1: Un programa que te pida tu nombre y lo muestre en pantalla separando cada letra de la siguiente con un espacio, similar al 4.4.3.1, pero esta vez usando "Substring". Por ejemplo, si tu nombre es "Juan", debería aparecer en pantalla "J u a n".

Ejercicio propuesto 4.4.4.2: Un programa que te pida tu nombre y lo muestre en pantalla como un triángulo creciente. Por ejemplo, si tu nombre es "Juan", debería aparecer en pantalla:

```
J
Ju
Jua
Juan
```

4.4.5. Buscar en una cadena

Para ver si una cadena contiene un cierto texto, podemos usar IndexOf ("posición de"), que nos dice en qué posición se encuentra, siendo 0 la primera posición (o devuelve el valor -1, si no aparece):

```
if (nombre.IndexOf("Juan") >= 0) Console.Write("Bienvenido, Juan");
```

Podemos añadir un segundo parámetro opcional, que es la posición a partir de la que queremos buscar:

```
if (nombre.IndexOf("Juan", 5) >= 0) ...
```

La búsqueda termina al final de la cadena, salvo que indiquemos que termine antes con un tercer parámetro opcional:

```
if (nombre.IndexOf("Juan", 5, 15) >= 0) ...
```

De forma similar, LastIndexOf ("última posición de") indica la última aparición (es decir, busca de derecha a izquierda).

Si solamente queremos ver si aparece, pero no nos importa en qué posición está, nos bastará con usar "Contains":

```
if (nombre.Contains("Juan")) ...
```

Un ejemplo de la utilización de IndexOf podría ser:

```
// Ejemplo_04_04_05a.cs
// Cadenas de texto (5: substring)
// Introducción a C#
```

```
using System;

public class Ejemplo_04_04_05a
{
    public static void Main()
    {
        string saludo = "Hola";
        string subcadena = "ola";
        Console.WriteLine("{0} aparece dentro de {1} en la posición {2}",
            subcadena, saludo, saludo.IndexOf(subcadena) );
    }
}
```

4.4.6. Otras manipulaciones de cadenas

Ya hemos comentado que las cadenas en C# son inmutables, no se pueden modificar. Pero sí podemos realizar ciertas operaciones sobre ellas para obtener una nueva cadena. Por ejemplo:

- ToUpper() convierte a mayúsculas: nombreCorrecto = nombre.ToUpper();
- ToLower() convierte a minúsculas: password2 = password.ToLower();
- Insert(int posición, string subcadena): Insertar una subcadena en una cierta posición de la cadena inicial: nombreFormal = nombre.Insert(0,"Don");
- Remove(int posición, int cantidad): Elimina una cantidad de caracteres en cierta posición: apellidos = nombreCompleto.Remove(0,6);
- Replace(string textoASustituir, string cadenaSustituta): Sustituye una cadena (todas las veces que aparezca) por otra: nombreCorregido = nombre.Replace("Pepe", "Jose");

Un programa que probara todas estas posibilidades podría ser así:

```
// Ejemplo_04_04_06a.cs
// Cadenas de texto (6: manipulaciones diversas)
// Introducción a C#

using System;

public class Ejemplo_04_04_06a
{
    public static void Main()
    {
        string ejemplo = "Hola, que tal estas";

        Console.WriteLine("El texto es: {0}",
            ejemplo);

        Console.WriteLine("La primera letra es: {0}",
            ejemplo[0]);

        Console.WriteLine("Las tres primeras letras son: {0}",
            ejemplo.Substring(0,3));

        Console.WriteLine("La longitud del texto es: {0}",
            ejemplo.Length);

        Console.WriteLine("La posicion de \"que\" es: {0}",
            ejemplo.IndexOf("que"));

        Console.WriteLine("La ultima A esta en la posicion: {0}",
```

```
ejemplo.LastIndexOf("a"));

Console.WriteLine("En mayúsculas: {0}",
    ejemplo.ToUpper());

Console.WriteLine("En minúsculas: {0}",
    ejemplo.ToLower());

Console.WriteLine("Si insertamos \", tio\": {0}",
    ejemplo.Insert(4, ", tio"));

Console.WriteLine("Si borramos las 6 primeras letras: {0}",
    ejemplo.Remove(0, 6));

Console.WriteLine("Si cambiamos ESTAS por ESTAMOS: {0}",
    ejemplo.Replace("estas", "estamos"));
}
```

Y su resultado sería

```
El texto es: Hola, que tal estas
La primera letra es: H
Las tres primeras letras son: Hol
La longitud del texto es: 19
La posicion de "que" es: 6
La ultima A esta en la posicion: 17
En mayúsculas: HOLA, QUE TAL ESTAS
En minúsculas: hola, que tal estas
Si insertamos ", tio": Hola, tio, que tal estas
Si borramos las 6 primeras letras: que tal estas
Si cambiamos ESTAS por ESTAMOS: Hola, que tal estamos
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.6.1: Una variante del ejercicio 4.4.4.2, que no distinga entre mayúsculas y minúsculas a la hora de buscar.

Ejercicio propuesto 4.4.6.2: Un programa que pida al usuario una frase y elimine todos los espacios redundantes que contenga (debe quedar sólo un espacio entre cada palabra y la siguiente).

4.4.7. Descomponer una cadena en fragmentos

Una operación relativamente frecuente, pero trabajosa, es descomponer una cadena en varios fragmentos que estén delimitados por ciertos separadores. Por ejemplo, podríamos descomponer una frase en varias palabras que estaban separadas por espacios en blanco.

Si lo queremos hacer "de forma artesanal", podemos recorrer la cadena buscando y contando los espacios (o los separadores que nos interesen). Así podremos saber el tamaño del array que deberá almacenar las palabras (por ejemplo, si hay dos espacios, tendremos tres palabras). En una segunda pasada, obtendremos las subcadenas que hay entre cada dos espacios y las guardaríamos en el array. No es especialmente sencillo.

Afortunadamente, C# nos permite hacerlo con **Split**, que crea un array a partir de los fragmentos de la cadena usando el separador que le indiquemos, así:

```
// Ejemplo_04_04_07a.cs
// Cadenas de texto: partir con "Split"
// Introducción a C#

using System;

public class Ejemplo_04_04_07a
{
    public static void Main()
    {
        string ejemplo = "uno dos tres cuatro";
        char delimitador = ' ';
        int i;

        string [] ejemploPartido = ejemplo.Split(delimitador);

        for (i=0; i<ejemploPartido.Length; i++)
            Console.WriteLine("Fragmento {0} = {1}",
                              i, ejemploPartido[i]);
    }
}
```

Que mostraría en pantalla lo siguiente:

```
Fragmento 0 = uno
Fragmento 1 = dos
Fragmento 2 = tres
Fragmento 3 = cuatro
```

Pero también podemos usar como delimitador un conjunto (un array de caracteres). Por ejemplo, podríamos considerar un separador válido el espacio, pero también la coma, el punto y cualquier otro. Los cambios en el programa serían mínimos:

```
// Ejemplo_04_04_07b.cs
// Cadenas de texto: partir con "Split" - 2
// Introducción a C#

using System;

public class Ejemplo_04_04_07b
{
    public static void Main()
    {
        string ejemplo = "uno,dos.tres,cuatro";
        char [] delimitadores = {',','.', ' '};
        int i;

        string [] ejemploPartido = ejemplo.Split(delimitadores);

        for (i=0; i<ejemploPartido.Length; i++)
            Console.WriteLine("Fragmento {0} = {1}",
                              i, ejemploPartido[i]);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.7.1: Un programa que pida al usuario una frase y muestre sus palabras en orden inverso.

Ejercicio propuesto 4.4.7.2: Un programa que pida al usuario varios números separados por espacios y muestre su suma.

4.4.8. Comparación de cadenas

Sabemos comprobar si una cadena tiene exactamente un cierto valor, con el operador de igualdad (==), pero no sabemos comparar qué cadena es "mayor" que otra, y eso algo que es necesario si queremos ordenar textos. El operador "mayor que" (>) que usamos con los números no se puede aplicar directamente a las cadenas. En su lugar, debemos usar "CompareTo", que devolverá un número mayor que 0 si la nuestra cadena es mayor que la que indicamos como parámetro (o un número negativo si nuestra cadena es menor, o 0 si son iguales):

```
if (frase.CompareTo("hola") > 0)
    Console.WriteLine("La frase es mayor que hola");
```

Esto tiene una limitación: si lo usamos de esa manera, las mayúsculas y minúsculas se consideran diferentes. Es más habitual que deseemos comparar sin distinguir entre mayúsculas y minúsculas, y eso se puede conseguir convirtiendo ambas cadenas a mayúsculas antes de convertir, o bien empleando String.Compare, al que indicamos las dos cadenas y un tercer dato booleano, que será "true" cuando queramos ignorar esa distinción:

```
if (String.Compare(frase, "hola", true) > 0)
    Console.WriteLine("Es mayor que hola (mays o mins)");
```

Un programa completo de prueba podría ser así:

```
// Ejemplo_04_04_08a.cs
// Cadenas de texto y comparación alfabética
// Introducción a C#

using System;

public class Ejemplo_04_04_08a
{
    public static void Main()
    {
        string frase;

        Console.WriteLine("Escriba una palabra");
        frase = Console.ReadLine();

        // Compruebo si es exactamente hola
        if (frase == "hola")
            Console.WriteLine("Ha escrito hola");

        // Compruebo si es mayor o menor
        if (frase.CompareTo("hola") > 0)
            Console.WriteLine("Es mayor que hola");
        else if (frase.CompareTo("hola") < 0)
            Console.WriteLine("Es menor que hola");
```

```
// Comparo sin distinguir mayúsculas ni minúsculas
bool ignorarMays = true;
if (String.Compare(frase, "hola", ignorarMays) > 0)
    Console.WriteLine("Es mayor que hola (mays o mins)");
else if (String.Compare(frase, "hola", ignorarMays) < 0)
    Console.WriteLine("Es menor que hola (mays o mins)");
else
    Console.WriteLine("Es hola (mays o mins)");
}
```

Si tecleamos una palabra como "gol", que comienza por G, que alfabéticamente está antes de la H de "hola", se nos dirá que esa palabra es menor:

```
Escriba una palabra
gol
Es menor que hola
Es menor que hola (mays o mins)
```

Si escribimos "hOLa", que coincide con "hola" salvo por las mayúsculas, una comparación normal nos dirá que es mayor (las mayúsculas se consideran "mayores" que las minúsculas), y una comparación sin considerar mayúsculas o minúsculas nos dirá que coinciden:

```
Escriba una palabra
hOLa
Es mayor que hola
Es hola (mays o mins)
```

Ejercicios propuestos:

Ejercicio propuesto 4.4.8.1: Un programa que pida al usuario dos frases y diga cuál sería la "mayor" de ellas (la que aparecería en último lugar en un diccionario).

Ejercicio propuesto 4.4.8.2: Un programa que pida al usuario cinco frases, las guarde en un array y muestre la "mayor" de ellas.

4.4.9. Una cadena modificable: StringBuilder

Si tenemos la necesidad de modificar una cadena letra a letra, no podemos usar un "string" convencional, porque no es válido hacer cosas como `texto[1]='h'`; Deberíamos formar una nueva cadena en la que modificásemos esa letra a base de unir varios substring o de borrar un fragmento con `Remove` y añadir otro con `Insert`.

Como alternativa, podemos recurrir a un "StringBuilder", que sí lo permiten pero son algo más complejos de manejar: hay de reservarles espacio con "new" (igual que hacíamos en ciertas ocasiones con los Arrays), y se pueden convertir a una cadena "convencional" usando "ToString":

```
// Ejemplo_04_04_09a.cs
// Cadenas modificables con "StringBuilder"
// Introducción a C#
```

```

using System;
using System.Text; // Usaremos un System.Text.StringBuilder

public class Ejemplo_04_04_09a
{
    public static void Main()
    {
        StringBuilder cadenaModificable = new StringBuilder("Hola");
        cadenaModificable[0] = 'M';
        Console.WriteLine("Cadena modificada: {0}",
            cadenaModificable);

        string cadenaNormal;
        cadenaNormal = cadenaModificable.ToString();
        Console.WriteLine("Cadena normal a partir de ella: {0}",
            cadenaNormal);
    }
}

```

Ejercicios propuestos:

Ejercicio propuesto 4.4.9.1: Un programa que pida una cadena al usuario y la modifique, de modo que todas las vocales se conviertan en "o".

Ejercicio propuesto 4.4.9.2: Un programa que pida una cadena al usuario y la modifique, de modo que las letras de las posiciones impares (primera, tercera, etc.) estén en minúsculas y las de las posiciones pares estén en mayúsculas, mostrando el resultado en pantalla. Por ejemplo, a partir de un nombre como "Nacho", la cadena resultante sería "nAcHo".

Ejercicio propuesto 4.4.9.3: Crea un juego del ahorcado, en el que un primer usuario introduzca la palabra a adivinar, se muestre ésta oculta con guiones (----) y el programa acepte las letras que introduzca el segundo usuario, cambiando los guiones por letras correctas cada vez que acierte (por ejemplo, a---a-t-). La partida terminará cuando se acierte la palabra por completo o el usuario agote sus 8 intentos.

4.5. Recorriendo arrays y cadenas con foreach

Existe una construcción parecida a "for", pensada para recorrer ciertas estructuras de datos, como los arrays y las cadenas de texto (y otras que veremos más adelante).

Se usa con el formato "foreach (variable in ConjuntoDeValores)":

```

// Ejemplo_04_05a.cs
// Ejemplo de "foreach"
// Introducción a C#

using System;

public class Ejemplo_04_05a
{
    public static void Main()
    {
        int[] diasMes = {31, 28, 21};
        foreach(int dias in diasMes) {
            Console.WriteLine("Dias del mes: {0}", dias);
        }
    }
}

```

```
string[] nombres = {"Alberto", "Andrés", "Antonio"};
foreach(string nombre in nombres) {
    Console.Write(" {0}", nombre);
}
Console.WriteLine();

string saludo = "Hola";
foreach(char letra in saludo) {
    Console.Write("{0}-", letra);
}
Console.WriteLine();
}
```

Ejercicios propuestos:

Ejercicio propuesto 4.5.1: Un programa que pida tu nombre y lo muestre con un espacio entre cada par de letras, usando "foreach".

Ejercicio propuesto 4.5.2: Un programa que pida al usuario una frase y la descomponga en subcadenas separadas por espacios, usando "Split". Luego debe mostrar cada subcadena en una línea nueva, usando "foreach".

Ejercicio propuesto 4.5.3: Un programa que pida al usuario varios números separados por espacios y muestre su suma (como el del ejercicio 4.4.7.2), pero empleando "foreach".

4.6. Ejemplo completo

Vamos a hacer un ejemplo completo que use tablas ("arrays"), registros ("struct") y que además manipule cadenas.

La idea va a ser la siguiente: Crearemos un programa que pueda almacenar datos de hasta 1000 ficheros (archivos de ordenador). Para cada fichero, debe guardar los siguientes datos: Nombre del fichero, Tamaño (en KB, un número de 0 a 8.000.000.000). El programa mostrará un menú que permita al usuario las siguientes operaciones:

1. Añadir datos de un nuevo fichero
2. Mostrar los nombres de todos los ficheros almacenados
3. Mostrar ficheros que sean de más de un cierto tamaño (por ejemplo, 2000 KB).
4. Ver todos los datos de un cierto fichero (a partir de su nombre)
5. Salir de la aplicación (como no usamos ficheros, los datos se perderán).

No debería resultar difícil. Vamos a ver directamente una de las formas en que se podría plantear y luego comentaremos alguna de las mejoras que se podría (incluso se debería) hacer.

La única complicación real es que el array no estará completamente lleno: habrá espacio para 1000 datos, pero iremos añadiendo de uno en uno. Basta con contar el número de fichas que tenemos almacenadas, y así sabremos en qué posición iría la siguiente: si tenemos 0 fichas, deberemos almacenar la siguiente (la primera) en la posición 0; si tenemos dos fichas, serán la 0 y la 1, luego añadiremos en la posición 2; en general, si tenemos "n" fichas, añadiremos cada nueva ficha en la posición "n".

Por otra parte, para revisar todas las fichas existentes, recorreremos desde la posición 0 hasta la n-1, haciendo algo como

```
for (i=0; i<=n-1; i++) { /* ... más órdenes ... */ }
```

o bien algo como

```
for (i=0; i<n; i++) { /* ... más órdenes ... */ }
```

(esta segunda alternativa es la que se suele considerar "más natural" para un programador en un lenguaje como C#).

El resto del programa no es difícil: sabemos leer y comparar textos y números, comprobar varias opciones con "switch", etc. Aun así, haremos una última consideración: hemos limitado el número de fichas a 1000, así que, si nos piden añadir, deberíamos asegurarnos antes de que todavía tenemos hueco disponible.

Con todo esto, nuestro "fuente" quedaría así:

```
// Ejemplo_04_06a.cs
// Tabla con muchos struct y menú para manejarla
// Introducción a C#

using System;

public class Ejemplo_04_06a {

    struct tipoFicha {
        public string nombreFich;    // Nombre del fichero
        public long tamanyo;         // El tamaño en KB
    };

    public static void Main() {
        tipoFicha[] fichas    // Los datos en si
            = new tipoFicha[1000];
        int numeroFichas=0;    // Número de fichas que ya tenemos
        int i;                // Para bucles
        int opcion;           // La opcion del menu que elija el usuario
        string textoBuscar;   // Para cuando preguntemos al usuario
        long tamanyoBuscar;   // Para buscar por tamaño

        do {
            // Menu principal, repetitivo
            Console.WriteLine();
            Console.WriteLine("Escoja una opción:");
            Console.WriteLine("1.- Añadir datos de un nuevo fichero");
            Console.WriteLine("2.- Mostrar los nombres de todos los ficheros");
            Console.WriteLine("3.- Mostrar ficheros por encima de un cierto tamaño");
            Console.WriteLine("4.- Ver datos de un fichero");
            Console.WriteLine("5.- Salir");

            opcion = Convert.ToInt32( Console.ReadLine() );

            // Hacemos una cosa u otra según la opción escogida
            switch(opcion) {

                case 1: // Añadir un dato nuevo
                    if (numeroFichas < 1000) { // Si queda hueco
                        Console.WriteLine("Introduce el nombre del fichero: ");
```

```

        fichas[numeroFichas].nombreFich = Console.ReadLine();
        Console.WriteLine("Introduce el tamaño en KB: ");
        fichas[numeroFichas].tamanyo = Convert.ToInt32(
            Console.ReadLine() );
        // Y ya tenemos una ficha más
        numeroFichas++;
    } else // Si no hay hueco para más fichas, avisamos
        Console.WriteLine("Máximo de fichas alcanzado (1000)!");
    break;

case 2: // Mostrar todos
    for (i=0; i<numeroFichas; i++)
        Console.WriteLine("Nombre: {0}; Tamaño: {1} KB",
            fichas[i].nombreFich, fichas[i].tamanyo);
    break;

case 3: // Mostrar según el tamaño
    Console.WriteLine("¿A partir de que tamaño quieres ver?");
    tamanyoBuscar = Convert.ToInt64( Console.ReadLine() );
    for (i=0; i<numeroFichas; i++)
        if (fichas[i].tamanyo >= tamanyoBuscar)
            Console.WriteLine("Nombre: {0}; Tamaño: {1} KB",
                fichas[i].nombreFich, fichas[i].tamanyo);
    break;

case 4: // Ver todos los datos (pocos) de un fichero
    Console.WriteLine("¿De qué fichero quieres ver todos los datos?");
    textoBuscar = Console.ReadLine();
    for (i=0; i<numeroFichas; i++)
        if ( fichas[i].nombreFich == textoBuscar )
            Console.WriteLine("Nombre: {0}; Tamaño: {1} KB",
                fichas[i].nombreFich, fichas[i].tamanyo);
    break;

case 5: // Salir: avisamos de que salimos */
    Console.WriteLine("Fin del programa");
    break;

default: // Otra opcion: no válida
    Console.WriteLine("Opción desconocida!");
    break;
}
} while (opcion != 5); // Si la opcion es 5, terminamos
}
}

```

(Como quizá hayas notado, este fuente, que es un poco más largo que los anteriores, abre las llaves al final de cada línea -estilo Java- y usa tabulaciones de 2 espacios, en vez de 4, para ocupar menos espacio en papel y caber en el ancho de una página; recuerda que puedes usar estilo Java si lo prefieres, pero en general el fuente será más legible con tabulaciones de 4 espacios en vez de 2).

Este programa funciona, y hace todo lo que tiene que hacer, pero es mejorable. Por supuesto, en un caso real es habitual que cada ficha tenga que guardar más información que sólo esos dos apartados de ejemplo que hemos previsto esta vez. Si nos muestra todos los datos en pantalla y se trata de muchos datos, puede ocurrir que aparezcan en pantalla tan rápido que no nos dé tiempo a leerlos, así que sería deseable que parase cuando se llenase la pantalla de información (por ejemplo, una pausa tras mostrar cada 25 datos). Por descontado, se nos pueden ocurrir muchas más preguntas que hacerle sobre nuestros datos. Y además, cuando salgamos del programa se borrarán todos los datos que habíamos tecleado, pero eso es lo único "casi inevitable", porque aún no sabemos manejar ficheros.

Ejercicios propuestos:

Ejercicio propuesto 4.6.1: Un programa que pida el nombre, el apellido y la edad de una persona, los almacene en un "struct" y luego muestre los tres datos en una misma línea, separados por comas.

Ejercicio propuesto 4.6.2: Un programa que pida datos de 8 personas: nombre, día de nacimiento, mes de nacimiento, y año de nacimiento (que se deben almacenar en un array de structs). Después deberá repetir lo siguiente: preguntar un número de mes y mostrar en pantalla los datos de las personas que cumplan los años durante ese mes. Terminará de repetirse cuando se teclee 0 como número de mes.

Ejercicio propuesto 4.6.3: Un programa que sea capaz de almacenar los datos de 50 personas: nombre, dirección, teléfono, edad (usando una tabla de structs). Deberá ir pidiendo los datos uno por uno, hasta que un nombre se introduzca vacío (se pulse Intro sin teclear nada). Entonces deberá aparecer un menú que permita:

- Mostrar la lista de todos los nombres
- Mostrar las personas de una cierta edad
- Mostrar las personas cuya inicial sea la que el usuario indique
- Salir del programa

(lógicamente, este menú debe repetirse hasta que se escoja la opción de "salir").

Ejercicio propuesto 4.6.4: Mejorar la base de datos de ficheros (ejemplo 04_06a) para que no permita introducir tamaños incorrectos (números negativos) ni nombres de fichero vacíos.

Ejercicio propuesto 4.6.5: Ampliar la base de datos de ficheros (ejemplo 04_06a) para que incluya una opción de búsqueda parcial, en la que el usuario indique parte del nombre y se muestre todos los ficheros que contienen ese fragmento (usando "Contains" o "IndexOf"). Esta búsqueda no debe distinguir mayúsculas y minúsculas (con la ayuda de ToUpper o ToLower).

Ejercicio propuesto 4.6.6: Ampliar el ejercicio anterior (4.6.5) para que la búsqueda sea incremental: el usuario irá indicando letra a letra el texto que quiere buscar, y se mostrará todos los datos que lo contienen (por ejemplo, primero los que contienen "j", luego "ju", después "jua" y finalmente "juan").

Ejercicio propuesto 4.6.7: Ampliar la base de datos de ficheros (ejemplo 04_06a) para que se pueda borrar un cierto dato (habrá que "mover hacia atrás" todos los datos que había después de ese, y disminuir el contador de la cantidad de datos que tenemos).

Ejercicio propuesto 4.6.8: Mejorar la base de datos de ficheros (ejemplo 04_06a) para que se pueda modificar un cierto dato a partir de su número (por ejemplo, el dato número 3). En esa modificación, se deberá permitir al usuario pulsar Intro sin teclear nada, para indicar que no desea modificar un cierto dato, en vez de reemplazarlo por una cadena vacía.

Ejercicio propuesto 4.6.9: Ampliar la base de datos de ficheros (ejemplo 04_06a) para que se permita ordenar los datos por nombre. Para ello, deberás buscar información sobre algún método de ordenación sencillo, como el "método de burbuja" (en el siguiente apartado tienes algunos), y aplicarlo a este caso concreto.

4.7. Ordenaciones simples

Es muy frecuente querer ordenar datos que tenemos en un array. Para conseguirlo, existen varios algoritmos sencillos, que no son especialmente eficientes, pero son fáciles de programar. La falta de eficiencia se refiere a que la mayoría de ellos se basan en dos bucles "for" anidados, de modo que en cada pasada quede ordenado un único dato, y habrá que dar tantas pasadas como datos existen. Por tanto, para un array con 1.000 datos, podrían llegar a ser necesarias un millón de comparaciones (1.000 x 1.000).

Existen ligeras mejoras (por ejemplo, cambiar uno de los "for" por un "while", para no repasar todos los datos si ya estaban parcialmente ordenados), así como métodos claramente más efectivos, pero más difíciles de programar, alguno de los cuales comentaremos más adelante.

Veremos tres de estos métodos simples de ordenación, primero mirando la apariencia que tiene el algoritmo, y luego juntando los tres métodos en un ejemplo que los pruebe:

- *Método de la burbuja* -

(Intercambiar cada pareja consecutiva que no esté ordenada)

```
Para i=1 hasta n-1
    Para j=i+1 hasta n
        Si A[i] > A[j]
            Intercambiar ( A[i], A[j])
```

(Nota: algunos autores hacen el bucle exterior creciente y otros decreciente, así:)

```
Para i=n descendiendo hasta 2
    Para j=2 hasta i
        Si A[j-1] > A[j]
            Intercambiar ( A[j-1], A[j])
```

- *Selección directa* -

(En cada pasada busca el menor, y lo intercambia al final de la pasada)

```
Para i=1 hasta n-1
    menor = i
    Para j=i+1 hasta n
        Si A[j] < A[menor]
            menor = j
    Si menor <> i
        Intercambiar ( A[i], A[menor])
```

Nota: el símbolo "<>" se suele usar en pseudocódigo para indicar que un dato es distinto de otro, de modo que equivale al "!=" de C#. La penúltima línea en C# saldría a ser algo como "if (menor != i)"

- *Inserción directa* -

(Comparar cada elemento con los anteriores -que ya están ordenados- y desplazarlo hasta su posición correcta).

```
Para i=2 hasta n
    j=i-1
    mientras (j>=1) y (A[j] > A[j+1])
        Intercambiar ( A[j], A[j+1])
        j = j - 1
```

(Es mejorable, no intercambiando el dato que se mueve con cada elemento, sino sólo al final de cada pasada, pero no entraremos en más detalles).

Un programa de prueba podría ser:

```
// Ejemplo_04_07a.cs
// Ordenaciones simples
// Introducción a C#

using System;

public class Ejemplo_04_07a
{
    public static void Main()
    {
        int[] datos = {5, 3, 14, 20, 8, 9, 1};
        int i,j,datoTemporal;
        int n=7; // Numero de datos

        // BURBUJA
        // (Intercambiar cada pareja consecutiva que no esté ordenada)
        // Para i=1 hasta n-1
        //     Para j=i+1 hasta n
        //         Si A[i] > A[j]
        //             Intercambiar ( A[i], A[j])
        Console.WriteLine("Ordenando mediante burbuja... ");
        for(i=0; i < n-1; i++)
        {
            foreach (int dato in datos) // Muestro datos
                Console.Write("{0} ",dato);
            Console.WriteLine();

            for(j=i+1; j < n; j++)
            {
                if (datos[i] > datos[j])
                {
                    datoTemporal = datos[i];
                    datos[i] = datos[j];
                    datos[j] = datoTemporal;
                }
            }
        }
        Console.WriteLine("Ordenado:");

        foreach (int dato in datos) // Muestro datos finales
            Console.Write("{0} ",dato);
        Console.WriteLine();
    }
}
```

```
// SELECCIÓN DIRECTA:
// (En cada pasada busca el menor, y lo intercambia al final de la
pasada)
// Para i=1 hasta n-1
//     menor = i
//     Para j=i+1 hasta n
//         Si A[j] < A[menor]
//             menor = j
//     Si menor <> i
//         Intercambiar ( A[i], A[menor])
Console.WriteLine("\nOrdenando mediante selección directa... ");
int[] datos2 = {5, 3, 14, 20, 8, 9, 1};
for(i=0; i < n-1; i++)
{
    foreach (int dato in datos2) // Muestro datos
        Console.Write("{0} ",dato);
    Console.WriteLine();

    int menor = i;
    for(j=i+1; j < n; j++)
        if (datos2[j] < datos2[menor])
            menor = j;

    if (i != menor)
    {
        datoTemporal = datos2[i];
        datos2[i] = datos2[menor];
        datos2[menor] = datoTemporal;
    }
}
Console.Write("Ordenado:");

foreach (int dato in datos2) // Muestro datos finales
    Console.Write("{0} ",dato);
Console.WriteLine();

// INSERCIÓN DIRECTA:
// (Comparar cada elemento con los anteriores -que ya están ordenados-
// y desplazarlo hasta su posición correcta).
// Para i=2 hasta n
//     j=i-1
//     mientras (j>=1) y (A[j] > A[j+1])
//         Intercambiar ( A[j], A[j+1])
//     j = j - 1
Console.WriteLine("\nOrdenando mediante inserción directa... ");
int[] datos3 = {5, 3, 14, 20, 8, 9, 1};
for(i=1; i < n; i++)
{
    foreach (int dato in datos3) // Muestro datos
        Console.Write("{0} ",dato);
    Console.WriteLine();

    j = i-1;
    while ((j>=0) && (datos3[j] > datos3[j+1]))
    {
        datoTemporal = datos3[j];
        datos3[j] = datos3[j+1];
        datos3[j+1] = datoTemporal;
        j--;
    }
}
Console.Write("Ordenado:");
```

```
foreach (int dato in datos3) // Muestro datos finales
    Console.Write("{0} ",dato);
Console.WriteLine();

}
```

Y su resultado sería:

Ordenando mediante burbuja...

```
5 3 14 20 8 9 1
1 5 14 20 8 9 3
1 3 14 20 8 9 5
1 3 5 20 14 9 8
1 3 5 8 20 14 9
1 3 5 8 9 20 14
Ordenado:1 3 5 8 9 14 20
```

Ordenando mediante selección directa...

```
5 3 14 20 8 9 1
1 3 14 20 8 9 5
1 3 14 20 8 9 5
1 3 5 20 8 9 14
1 3 5 8 20 9 14
1 3 5 8 9 20 14
Ordenado:1 3 5 8 9 14 20
```

Ordenando mediante inserción directa...

```
5 3 14 20 8 9 1
3 5 14 20 8 9 1
3 5 14 20 8 9 1
3 5 14 20 8 9 1
3 5 8 14 20 9 1
3 5 8 9 14 20 1
Ordenado:1 3 5 8 9 14 20
```

Ejercicios propuestos:

Ejercicio propuesto 4.7.1: Un programa que pida al usuario 6 números en coma flotante y los muestre ordenados de menor a mayor. Escoge el método de ordenación que prefieras.

Ejercicio propuesto 4.7.2: Un programa que pida al usuario 5 nombres y los muestre ordenados alfabéticamente (recuerda que para comparar cadenas no podrás usar el símbolo ">", sino "CompareTo").

Ejercicio propuesto 4.7.3: Un programa que pida al usuario varios números, los vaya añadiendo a un array, mantenga el array ordenado continuamente y muestre el resultado tras añadir cada nuevo dato (todos los datos se mostrarán en la misma línea, separados por espacios en blanco). Terminará cuando el usuario teclee "fin".

Ejercicio propuesto 4.7.4: Amplia el ejercicio anterior, para añadir una segunda fase en la que el usuario pueda "preguntar" si un cierto valor está en el array. Como el array está ordenado, la búsqueda no se hará hasta el final de los datos, sino hasta que se encuentre el dato buscado o un dato mayor que él.

Una vez que los datos están ordenados, podemos buscar uno concreto dentro de ellos empleando la **búsqueda binaria**: se comienza por el punto central; si el valor buscado es mayor que el del punto central, se busca esta vez sólo en la mitad superior (o en la inferior), y así sucesivamente, de modo

que cada vez se busca entre un conjunto de datos que tiene la mitad de tamaño que el anterior. Esto puede suponer una enorme ganancia en velocidad: si tenemos 1.000 datos, una búsqueda lineal hará 500 comparaciones como media, mientras que una búsqueda lineal hará 10 comparaciones o menos.

```
// Ejemplo_04_07b.cs
// Búsqueda binaria
// Introducción a C#

using System;

public class Ejemplo_04_07b
{
    public static void Main()
    {
        const int n = 1000;
        int[] datos = new int[n];
        int i, j, datoTemporal;

        // Primero generamos datos al azar
        Console.Write("Generando... ");
        Random r = new Random();
        for(i=0; i < n; i++)
            datos[i] = r.Next(1, n*2);

        // Luego los ordenamos mediante burbuja
        Console.Write("Ordenando... ");
        for(i=0; i < n-1; i++)
        {
            for(j=i+1; j < n; j++)
            {
                if (datos[i] > datos[j])
                {
                    datoTemporal = datos[i];
                    datos[i] = datos[j];
                    datos[j] = datoTemporal;
                }
            }
        }

        foreach (int dato in datos)
            Console.Write("{0} ", dato);
        Console.WriteLine();

        int valorBuscado = 1001;
        Console.WriteLine("Buscando si aparece {0}...", valorBuscado);

        int limiteInferior = 0;
        int limiteSuperior = 999;
        bool terminado = false;

        while(! terminado)
        {
            int puntoMedio = limiteInferior+(limiteSuperior-limiteInferior) / 2;
            // Aviso de dónde buscamos
            Console.WriteLine("Buscando entre pos {0} y {1}, valores {2} y {3}, "+" centro {4}:{5}",
                limiteInferior, limiteSuperior,
                datos[limiteInferior], datos[limiteSuperior],
                puntoMedio, datos[puntoMedio]);
            // Compruebo si hemos acertado
            if (datos[puntoMedio] == valorBuscado)
            {

```



```
        Console.WriteLine("Encontrado!");
        terminado = true;
    }
    else if (limiteInferior == limiteSuperior-1)
    {
        Console.WriteLine("No encontrado");
        terminado = true;
    }
    // Si no hemos terminado, debemos seguir buscando en una mitad
    if ( datos[puntoMedio] < valorBuscado )
        limiteInferior = puntoMedio;
    else
        limiteSuperior = puntoMedio;
}

}
```

Ejercicios propuestos:

Ejercicio propuesto 4.7.5: Realiza una variante del ejercicio 4.7.4, que en vez de hacer una búsqueda lineal (desde el principio), use "búsqueda binaria": se comenzará a comparar con el punto medio del array; si nuestro dato es menor, se vuelve a probar en el punto medio de la mitad inferior del array, y así sucesivamente.

¿Y no se puede **ordenar de forma más sencilla**? Sí, existe un "Array.Sort" que hace todo por nosotros... pero recuerda que no (sólo) se trata de que conozcas la forma más corta posible de hacer un ejercicio, sino de que aprendas a resolver problemas...

Ejercicios propuestos:

Ejercicio propuesto 4.7.6: Crea una variante del ejercicio 4.7.3, pero usando esta vez Array.Sort (investígalo) para ordenar: un programa que pida al usuario varios números, los vaya añadiendo a un array, mantenga el array ordenado continuamente y muestre el resultado tras añadir cada nuevo dato (todos los datos se mostrarán en la misma línea, separados por espacios en blanco). Terminará cuando el usuario teclee "fin".