3. Tipos de datos básicos

3.1. Tipo de datos entero

Hemos hablado de números enteros, de cómo realizar operaciones sencillas y de cómo usar variables para reservar espacio y así poder trabajar con datos cuyo valor no sabemos de antemano.

Empieza a ser el momento de refinar, de dar más detalles. El primer "matiz" importante que hemos esquivado hasta ahora es el tamaño de los números que podemos emplear, así como su signo (positivo o negativo). Por ejemplo, un dato de tipo "int" puede guardar números de hasta unas nueve cifras, tanto positivos como negativos, y ocupa 4 bytes en memoria.

Pero no es la única opción. Por ejemplo, si queremos guardar la edad de una persona, no necesitamos usar números negativos, y nos bastaría con 3 cifras, así que es de suponer que existirá algún tipo de dato más adecuado, que desperdicie menos memoria. También existe el caso contrario: un banco puede necesitar manejar números con más de 9 cifras, así que un dato "int" se les quedaría corto. Siendo estrictos, si hablamos de valores monetarios, necesitaríamos usar decimales, pero eso lo dejamos para el siguiente apartado.

3.1.1. Tipos de datos para números enteros

Los tipos de datos enteros que podemos usar en C#, junto con el espacio que ocupan en memoria y el rango de valores que nos permiten almacenar son:

Nombre Del Tipo	Tamaño (bytes)	Rango de valores
sbyte	1	-128 a 127
byte	1	0 a 255
short	2	-32768 a 32767
ushort	2	0 a 65535
int	4	-2147483648 a 2147483647
uint	4	0 a 4294967295
long	8	-9223372036854775808 a 9223372036854775807
ulong	8	0 a 18446744073709551615

Como se puede observar en esta tabla, el tipo de dato más razonable para guardar edades sería "byte", que permite valores entre 0 y 255, y ocupa 3 bytes menos que un "int".

```
// Ejemplo_03_01_01a.cs
// Tipos de números enteros
// Introducción a C#

using System;

public class Ejemplo_03_01_01a
{
    public static void Main()
    {
        byte edad = 74;
        ushort anyo = 2001;
        long resultado = 10000000000;
        Console.WriteLine("Los datos son {0}, {1} y {2}",
            edad, anyo, resultado);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 3.1.1.1: *Calcula el producto de 1.000.000 por 1.000.000, usando una variable llamada "producto", de tipo "long". Prueba también a calcularlo usando una variable de tipo "int".*

3.1.2. Conversiones de cadena a entero

Si queremos obtener estos datos a partir de una cadena de texto, no siempre nos servirá Convert.ToInt32, porque no todos los datos son enteros de 32 bits (4 bytes). Para datos de tipo "byte" usaríamos Convert.ToByte (sin signo) y ToSByte (con signo), para datos de 2 bytes tenemos ToInt16 (con signo) y ToUInt16 (sin signo), y para los de 8 bytes existen ToInt64 (con signo) y ToUInt64 (sin signo).

```
// Ejemplo_03_01_02a.cs
// Conversiones para otros tipos de números enteros
// Introducción a C#

using System;

public class Ejemplo_03_01_02a
{
    public static void Main()
    {
        string ejemplo1 = "74";
        string ejemplo2 = "2001";
        string ejemplo3 = "10000000000";

        byte edad = Convert.ToByte(ejemplo1);
        ushort anyo = Convert.ToUInt16(ejemplo2);
        long resultado = Convert.ToInt64(ejemplo3);
        Console.WriteLine("Los datos son {0}, {1} y {2}",
            edad, anyo, resultado);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 3.1.2.1: *Pregunta al usuario su edad, que se guardará en un "byte". A continuación, le deberás decir que no aparenta "esos" años (por ejemplo, "No aparentas 20 años").*

Ejercicio propuesto 3.1.2.2: *Pide al usuario dos números de dos cifras ("byte"), calcula su multiplicación, que se deberá guardar en un "ushort", y muestra el resultado en pantalla.*

Ejercicio propuesto 3.1.2.3: *Pide al usuario dos números enteros largos ("long") y muestra su suma, su resta y su producto.*

3.1.3. Incremento y decremento

Conocemos la forma de realizar las operaciones aritméticas más habituales. Pero también existe una operación que es muy frecuente cuando se crean programas, especialmente (como ya hemos visto) a la hora de controlar bucles: incrementar el valor de una variable en una unidad:

```
a = a + 1;
```

Pues bien, en C# (y en otros lenguajes que derivan de C, como C++, Java y PHP), existe una notación más compacta para esta operación, y para la opuesta (el decremento):

```
a++;          es lo mismo que    a = a+1;
a--;          es lo mismo que    a = a-1;
```

```
// Ejemplo_03_01_03a.cs
// Incremento y decremento
// Introducción a C#

using System;

public class Ejemplo_03_01_03a
{
    public static void Main()
    {
        int n = 10;
        Console.WriteLine("n vale {0}", n);
        n++;
        Console.WriteLine("Tras incrementar vale {0}", n);
        n--;
        n--;
        Console.WriteLine("Tras decrementar dos veces, vale {0}", n);
    }
}
```

Pero esto tiene algo más de dificultad de la que puede parecer en un primer vistazo: podemos distinguir entre "preincremento" y "postincremento". En C# es posible hacer asignaciones como

```
b = a++;
```

Así, si "a" valía 2, lo que esta instrucción hace es dar a "b" el valor de "a" y aumentar el valor de "a". Por tanto, al final tenemos que b=2 y a=3 (postincremento: se incrementa "a" tras asignar su valor).

En cambio, si escribimos

```
b = ++a;
```

y "a" valía 2, primero aumentamos "a" y luego los asignamos a "b" (preincremento), de modo que a=3 y b=3.

Por supuesto, también podemos distinguir postdecremento (a--) y predecremento (--a).

Ejercicios propuestos:

Ejercicio propuesto 3.1.3.1: *Crea un programa que use tres variables x,y,z. Sus valores iniciales serán 15, -10, 2.147.483.647. Se deberá incrementar el valor de estas variables. ¿Qué valores esperas que se obtengan? Contrástalo con el resultado obtenido por el programa.*

Ejercicio propuesto 3.1.3.2: *¿Cuál sería el valor de a y b al finalizar las siguientes operaciones? a=5; b=++a; c=a++; b=b*5; a=a*2; Calcúlalo a mano y luego crea un programa que lo resuelva, para ver si habías hallado la solución correcta.*

3.1.4. Operaciones abreviadas: +=

Aún hay más. Tenemos incluso formas reducidas de escribir cosas como "a = a+5". Allá van

```
a += b ;          es lo mismo que    a = a+b;
a -= b ;          es lo mismo que    a = a-b;
a *= b ;          es lo mismo que    a = a*b;
a /= b ;          es lo mismo que    a = a/b;
a %= b ;          es lo mismo que    a = a%b;
```

```
// Ejemplo_03_01_04a.cs
// Operaciones abreviadas
// Introducción a C#

using System;

public class Ejemplo_03_01_04a
{
    public static void Main()
    {
        int n = 10;
        Console.WriteLine("n vale {0}", n);
        n *= 2;
        Console.WriteLine("Tras duplicarlo, vale {0}", n);
        n /= 3;
        Console.WriteLine("Tras dividirlo entre tres, vale {0}", n);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 3.1.4.1: *Crea un programa que use tres variables x,y,z. Sus valores iniciales serán 15, -10, 214. Deberás incrementar el valor de estas variables en 12, usando el formato abreviado. ¿Qué valores esperas que se obtengan? Contrástalo con el resultado obtenido por el programa.*

Ejercicio propuesto 3.1.4.2: *¿Cuál sería el resultado de las siguientes operaciones? a=5; b=a+2; b-=3; c=-3; c*=2; ++c; a*=b; Crea un programa que te lo muestre.*

3.1.5. Asignaciones múltiples

Ya que estamos hablando de las asignaciones, es interesante comentar que en C# es posible hacer asignaciones múltiples:

```
   a = b = c = 1;
```

```
// Ejemplo_03_01_05a.cs
// Asignaciones múltiples
// Introducción a C#

using System;

public class Ejemplo_03_01_05a
{
    public static void Main()
    {
        int a=5, b=2, c=-3;
        Console.WriteLine("a={0}, b={1}, c={2}", a, b, c);
        a = b = c = 4;
        Console.WriteLine("Ahora a={0}, b={1}, c={2}", a, b, c);
        a++; b--; c*=2;
        Console.WriteLine("Y finalmente a={0}, b={1}, c={2}", a, b, c);
    }
}
```

3.2. Tipo de datos real

Cuando queremos almacenar datos con decimales, no nos sirve el tipo de datos "int". Necesitamos otro tipo de datos que sí esté preparado para guardar números "reales" (con decimales). Al igual que ocurría con los números enteros, tendremos más de un tipo de número real para elegir.

3.2.1. Coma fija y coma flotante

En el mundo de la informática hay dos formas de trabajar con números reales:

- **Coma fija**: el número máximo de cifras decimales está fijado de antemano, y el número de cifras enteras también. Por ejemplo, con un formato de 3 cifras enteras y 4 cifras decimales, el número 3,75 se almacenaría correctamente (como 003,7500), el número 970,4361 también se guardaría sin problemas, pero el 5,678642 se guardaría como 5,6786 (se perdería a partir de la cuarta cifra decimal) y el 1020 no se podría guardar (tiene más de 3 cifras enteras).

- **Coma flotante**: el número de decimales y de cifras enteras permitido es variable, lo que importa es el número de cifras significativas (a partir del último 0). Por ejemplo, con 5 cifras significativas se podrían almacenar números como el 13405000000 o como el 0,0000007349 pero no se guardaría correctamente el 12,0000034, que se redondearía a un número cercano.

Casi cualquier lenguaje de programación actual va a emplear números de coma flotante. En C# corresponden al tipo de datos llamado "float".

```
// Ejemplo_03_02_01a.cs
// Números reales (1: float)
// Introducción a C#

using System;

public class Ejemplo_03_02_01a
{
    public static void Main()
    {
        int i1 = 2, i2 = 3;
        float divisionI;

        Console.WriteLine("Vamos a dividir 2 entre 3 usando enteros");
        divisionI = i1/i2;
        Console.WriteLine("El resultado es {0}", divisionI);

        float f1 = 2, f2 = 3;
        float divisionF;

        Console.WriteLine("Vamos a dividir 2 entre 3 usando reales");
        divisionF = f1/f2;
        Console.WriteLine("El resultado es {0}", divisionF);
    }
}
```

Usando "float" sí hemos podido dividir con decimales. El resultado de este programa es:

```
Vamos a dividir 2 entre 3 usando enteros
El resultado es 0
Vamos a dividir 2 entre 3 usando reales
El resultado es 0,6666667
```

Ejercicios propuestos:

Ejercicio propuesto 3.2.1.1: *Crea un programa que muestre el resultado de dividir 3 entre 4 usando números enteros y luego usando números de coma flotante.*

Ejercicio propuesto 3.2.1.2: *¿Cuál sería el resultado de las siguientes operaciones, usando números reales? a=5; a/=2; a+=1; a*=3; --a;*

3.2.2. Simple y doble precisión

En la mayoría de lenguajes de programación, contamos con dos tamaños de números reales para elegir, según si queremos guardar números con mayor cantidad de cifras o con menos. Para números

con pocas cifras significativas (un máximo de 7, lo que se conoce como "un dato real de simple precisión") usaremos el tipo "float" y para números que necesiten más precisión (unas 15 cifras, "doble precisión") tenemos el tipo "double". En C# existe un tercer tipo de números reales, con mayor precisión todavía, el tipo "decimal", que se acerca a las 30 cifras significativas:

	float	double	decimal
Tamaño en bits	32	64	128
Valor más pequeño	$-1{,}5 \cdot 10^{-45}$	$5{,}0 \cdot 10^{-324}$	$1{,}0 \cdot 10^{-28}$
Valor más grande	$3{,}4 \cdot 10^{38}$	$1{,}7 \cdot 10^{308}$	$7{,}9 \cdot 10^{28}$
Cifras significativas	7	15-16	28-29

Así, podríamos plantear el ejemplo anterior con un "double" para obtener un resultado más preciso:

```
// Ejemplo_03_02_02a.cs
// Números reales (2: double)
// Introducción a C#

using System;

public class Ejemplo_03_02_02a
{
    public static void Main()
    {
        double n1 = 2, n2 = 3;
        double division;

        Console.WriteLine("Vamos a dividir 2 entre 3");
        division = n1/n2;
        Console.WriteLine("El resultado es {0}", division);
    }
}
```

Ahora su resultado sería:

```
Vamos a dividir 2 entre 3
El resultado es 0,666666666666667
```

Si queremos **dar un valor** inicial a un dato "float", debemos llevar cuidado. Es válido darle un valor entero, como hemos hecho en el ejemplo anterior:

```
float x = 2;
```

pero no podremos dar un valor que contenga cifras decimales, porque el compilador mostrará un mensaje de error diciendo que lo que aparece a la derecha es para él un dato "double" y el tipo de la variable es "float", así que se perderá precisión:

```
float pi = 3.14; // No válido
```

Hay un par de formas de solucionarlo. La más sencilla es añadir el sufijo "f" al número, para indicar al compilar que debe ser tratado como un "float":

```
float pi = 3.14f; // Dato correcto
```

Otra forma alternativa es forzar una "conversión de tipos", como veremos dentro de muy poco.

Así, podemos crear un programa que pida al usuario el radio de una circunferencia (que será un número entero) para mostrar la longitud de la circunferencia (cuyo valor será 2 * PI * radio) podría ser:

```
// Ejemplo_03_02_02a.cs
// Números reales (3: float)
// Introducción a C#

using System;

public class Ejemplo_03_02_02a
{
    public static void Main()
    {
        float radio, longitud , pi = 3.14159f;

        Console.WriteLine("Introduce el radio");
        radio = Convert.ToSingle(Console.ReadLine()); // Simple precisión
        longitud = 2*pi*radio;
        Console.WriteLine("La longitud de la circunferencia de radio {0} es
      {1}", radio, longitud);
    }
}
```

*¿Qué ocurre al introducir el radio desde consola con **,** (coma) o con **.** (punto) decimal?*

Ejercicios propuestos:

Ejercicio propuesto 3.2.2.1: *Crea un programa que muestre el resultado de dividir 13 entre 6 usando números enteros, luego usando números de coma flotante de simple precisión y luego con números de doble precisión.*

Ejercicio propuesto 3.2.2.2: *Calcula el área de un círculo, dado su radio, que será un número entero (área = pi * radio al cuadrado)*

3.2.3. Pedir números reales al usuario

Al igual que hacíamos con los enteros, podemos leer como cadena de texto, y convertir cuando vayamos a realizar operaciones aritméticas. Ahora usaremos Convert.ToDouble cuando se trate de un dato de doble precisión, Convert.ToSingle cuando sea un dato de simple precisión (float) y Convert.ToDecimal para un dato de precisión extra (decimal):

```
// Ejemplo_03_02_03a.cs
// Números reales: pedir al usuario
// Introducción a C#

using System;

public class Ejemplo_03_02_03a
{
    public static void Main()
    {
        float primerNumero;
        float segundoNumero;
        float suma;

        Console.WriteLine("Introduce el primer número");
        primerNumero = Convert.ToSingle(Console.ReadLine());
        Console.WriteLine("Introduce el segundo número");
        segundoNumero = Convert.ToSingle(Console.ReadLine());
        suma = primerNumero + segundoNumero;

        Console.WriteLine("La suma de {0} y {1} es {2}",
          primerNumero, segundoNumero, suma);
    }
}
```

Cuidado al probar este programa: aunque en el fuente debemos escribir los decimales usando **un punto**, como 123.456, al poner el ejecutable en marcha, cuando se pidan datos al usuario, parte del trabajo se le encarga al sistema operativo, de modo que si éste sabe que en nuestro país se usa la "coma" para separar los decimales, considerará que la coma es el separador correcto y no el punto, que será ignorado. Por ejemplo, ocurre si introducimos los datos 23,6 y 34.2 en la versión española de Windows 8 obtendremos como respuesta:

```
Introduce el primer número
23,6
Introduce el segundo número
34.2
La suma de 23,6 y 342 es 365,6
```

Ejercicios propuestos:

Ejercicio propuesto 3.2.3.1: *Calcula el volumen de una esfera, dado su radio, que será un número de doble precisión (volumen = pi * radio al cubo * 4/3)*

Ejercicio propuesto 3.2.3.2: *Crea un programa que pida al usuario a una distancia (en metros) y el tiempo necesario para recorrerla (como tres números: horas, minutos, segundos), y muestre la velocidad, en metros por segundo, en kilómetros por hora y en millas por hora (pista: 1 milla = 1.609 metros).*

Ejercicio propuesto 3.2.3.3: *Halla las soluciones de una ecuación de segundo grado del tipo* $y = Ax^2 + Bx + C$. *Pista: la raíz cuadrada de un número x se calcula con Math.Sqrt(x)*

Ejercicio propuesto 3.2.3.4: *Si se ingresan **E** euros en el banco a un cierto interés **I** durante **N** años, el dinero obtenido viene dado por la fórmula del interés compuesto: Resultado = e (1+ i)n Aplicarlo para calcular en cuanto se convierten 1.000 euros al cabo de 10 años al 3% de interés anual.*

Ejercicio propuesto 3.2.3.5: *Crea un programa que muestre los primeros 20 valores de la función* $y = x^2 - 1$

Ejercicio propuesto 3.2.3.6: *Crea un programa que "dibuje" la gráfica de $y = (x-5)^2$ para valores de x entre 1 y 10. Deberá hacerlo dibujando varios espacios en pantalla y luego un asterisco. La cantidad de espacios dependerá del valor obtenido para "y".*

Ejercicio propuesto 3.2.3.7: *Escribe un programa que calcule una aproximación de PI mediante la expresión: pi/4 = 1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 ... El usuario deberá indicar la cantidad de términos a utilizar, y el programa mostrará todos los resultados hasta esa cantidad de términos. Debes hacer todas las operaciones con "double".*

3.2.4. Conversión de tipos (typecast)

Cuando queremos convertir de un tipo de número a otro (por ejemplo, para quedarnos con la parte entera de un número real), tenemos dos alternativas:

- Usar Convert, como en x = Convert.ToInt32(y);
- Hacer un **forzado de tipos**, que es más rápido pero no siempre es posible, sólo cuando el tipo de origen y el de destino se parecen lo suficiente. Para ello, se precede el valor de la variable con el nuevo tipo de datos entre paréntesis, así: x = (int) y;

Por ejemplo, podríamos retocar el programa que calculaba la longitud de la circunferencia, de modo que su resultado sea un "double", que luego convertiremos a "float" y a "int" forzando el nuevo tipo de datos:

```
// Ejemplo_03_02_04a.cs
// Números reales: typecast
// Introducción a C#

using System;

public class Ejemplo_03_02_04a
{
    public static void Main()
    {
        double radio;
        float pi = (float) 3.141592654;
        double longitud;
        float longitudSimplePrec;
        int longitudEntera;

        Console.WriteLine("Introduce el radio");
        radio = Convert.ToDouble(Console.ReadLine());

        longitud = 2 * pi * radio;
        Console.WriteLine("La longitud de la circunferencia es");
        Console.WriteLine(longitud);

        longitudSimplePrec = (float) longitud;
        Console.WriteLine("Y con simple precisión");
        Console.WriteLine(longitudSimplePrec);

        longitudEntera = (int) longitud;
        Console.WriteLine("Y como número entero");
        Console.WriteLine(longitudEntera);
    }
}
```

Su resultado sería:

```
Introduce el radio
2,3456789
La longitud de la circunferencia es
14,7383356099727
Y con simple precisión
14,73834
Y como número entero
14
```

Ejercicio propuesto 3.2.4.1: *Crea un programa que calcule la raíz cuadrada del número que introduzca el usuario. La raíz se deberá calcular como "double", pero el resultado se mostrará como "float"*

Ejercicio propuesto 3.2.4.2: *Crea una nueva versión del un programa que calcula una aproximación de PI mediante la expresión: pi/4 = 1/1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 (...) con tantos términos como indique el usuario. Debes hacer todas las operaciones con "double", pero mostrar el resultado como "float".*

3.2.5. Formatear números

En más de una ocasión nos interesará afinar la apariencia de los números en pantalla, para mostrar sólo una cierta cantidad de decimales: por ejemplo, nos puede interesar que una cifra que corresponde a dinero se muestre siempre con dos cifras decimales, o que una nota se muestre redondeada, sin decimales, o con sólo un decimal.

Una forma de conseguirlo es crear una cadena de texto a partir del número, usando "ToString". A esta orden se le puede indicar un dato adicional, que es el formato numérico que queremos usar, por ejemplo: suma.ToString("0.00")

Algunos de los códigos de formato que se pueden usar son:

- Un cero (0) indica una posición en la que debe aparecer un número, y se mostrará un 0 si no hay ninguno.
- Una almohadilla (#) indica una posición en la que puede aparecer un número, y no se escribirá nada si no hay número.
- Un punto (.) indica la posición en la que deberá aparecer la coma decimal.
- Alternativamente, se pueden usar otros formatos abreviados: por ejemplo, N2 quiere decir "con dos cifras decimales" y N5 es "con cinco cifras decimales".

Vamos a probarlos en un ejemplo:

```
// Ejemplo_03_02_05a.cs
// Formato de números reales
// Introducción a C#

using System;

public class Ejemplo_03_02_05a
{
    public static void Main()
    {
        double numero = 12.34;
```

```
        Console.WriteLine( numero.ToString("N1") );
        Console.WriteLine( numero.ToString("N3") );
        Console.WriteLine( numero.ToString("0.0") );
        Console.WriteLine( numero.ToString("0.000") );
        Console.WriteLine( numero.ToString("#.#") );
        Console.WriteLine( numero.ToString("#.###") );
    }
}
```

El resultado de este ejemplo sería:

```
12,3
12,340
12,3
12,340
12,3
12,34
```

Ejercicio propuesto 3.2.5.1: *El usuario de nuestro programa podrá teclear dos números de hasta 12 cifras significativas. El programa deberá mostrar el resultado de dividir el primer número entre el segundo, utilizando tres cifras decimales.*

Ejercicio propuesto 3.2.5.2: *Crea un programa que use tres variables x,y,z. Las tres serán números reales, y nos bastará con dos cifras decimales. Se deberá pedir al usuario los valores para las tres variables y mostrar en pantalla el valor de $x^2 + y - z$ (con exactamente dos cifras decimales).*

Ejercicio propuesto 3.2.5.3: *Calcula el perímetro, área y diagonal de un rectángulo, a partir de su ancho y alto (perímetro = suma de los cuatro lados; área = base x altura; diagonal, usando el teorema de Pitágoras). Muestra todos ellos con una cifra decimal.*

Ejercicio propuesto 3.2.5.4: *Calcula la superficie y el volumen de una esfera, a partir de su radio (superficie = 4 * pi * radio al cuadrado; volumen = 4/3 * pi * radio al cubo). Usa datos "doble" y muestra los resultados con 5 cifras decimales.*

3.2.6. Cambios de base

Un uso alternativo de ToString es el de cambiar un número de base. Por ejemplo, habitualmente trabajamos con números decimales (en base 10), pero en informática son también muy frecuentes la base 2 (el sistema binario) y la base 16 (el sistema hexadecimal). Podemos convertir un número a binario o hexadecimal (o a base octal, menos frecuente) usando Convert.ToString e indicando la base, como en este ejemplo:

```
// Ejemplo_03_02_06a.cs
// De decimal a hexadecimal y binario
// Introducción a C#

using System;

public class Ejemplo_03_02_06a
{
    public static void Main()
    {
        int numero = 247;
```

```
        Console.WriteLine( Convert.ToString(numero, 16) );
        Console.WriteLine( Convert.ToString(numero, 2) );
    }
}
```

Su resultado sería:

```
f7
11110111
```

Ejercicios propuestos:

Ejercicio propuesto 3.2.6.1: *Crea un programa que pida números (en base 10) al usuario y muestre su equivalente en sistema binario y en hexadecimal. Debe repetirse hasta que el usuario introduzca el número 0.*

Ejercicio propuesto 3.2.6.2: *Crea un programa que pida al usuario la cantidad de rojo (por ejemplo, 255), verde (por ejemplo, 160) y azul (por ejemplo, 0) que tiene un color, y que muestre ese color RGB en notación hexadecimal (por ejemplo, FFA000).*

Ejercicio propuesto 3.2.6.3: *Crea un programa para mostrar los números del 0 a 255 en hexadecimal, en 16 filas de 16 columnas cada una (la primera fila contendrá los números del 0 al 15 –decimal-, la segunda del 16 al 31 –decimal- y así sucesivamente).*

Para convertir en sentido contrario, de **hexadecimal o binario a decimal**, podemos usar Convert.ToInt32, como se ve en el siguiente ejemplo. Es importante destacar que una constante hexadecimal se puede expresar precedida por "0x", como en "int n1 = 0x13;" (donde n1 tendría el valor 16+3=19, expresado en base 10). En los lenguajes C y C++, un valor precedido por "0" se considera octal, de modo que para "int n2 = 013;" el valor decimal de n2 sería 8+3=11, pero en C# no es así: un número que empiece por 0 (no por 0x) se considera que está escrito en base 10, como se ve en este ejemplo:

```
// Ejemplo_03_02_06b.cs
// De hexadecimal y binario a decimal
// Introducción a C#

using System;

public class Ejemplo_03_02_06b
{
    public static void Main()
    {
        int n1 = 0x13;
        int n2 = Convert.ToInt32("1a", 16);

        int n3 = 013; // No es octal, al contrario que en C y C++
        int n4 = Convert.ToInt32("14", 8);

        int n5 = Convert.ToInt32("11001001", 2);

        Console.WriteLine( "{0} {1} {2} {3} {4}",
            n1, n2, n3, n4, n5);
    }
}
```

```
Que mostraría:
19 26 13 12 201
```

Ejercicios propuestos:

Ejercicio propuesto 3.2.6.4: *Crea un programa que pida números binarios al usuario y muestre su equivalente en sistema hexadecimal y en decimal. Debe repetirse hasta que el usuario introduzca la palabra "fin".*

3.3. Tipo de datos carácter

3.3.1. Leer y mostrar caracteres

Como ya vimos brevemente, en C# también tenemos un tipo de datos que nos permite almacenar una única letra, el tipo "char":

```
char letra;
```

Asignar valores es sencillo: el valor se indica entre comillas simples

```
letra = 'a';
```

Para leer valores desde teclado, lo podemos hacer de forma similar a los casos anteriores: leemos toda una frase (que debería tener sólo una letra) con ReadLine y convertimos a tipo "char" usando Convert.ToChar:

```
letra = Convert.ToChar(Console.ReadLine());
```

Así, un programa que de un valor inicial a una letra, la muestre, lea una nueva letra tecleada por el usuario, y la muestre, podría ser:

```
// Ejemplo_03_03_01a.cs
// Tipo de datos "char"
// Introducción a C#

using System;

public class Ejemplo_03_03_01a
{
    public static void Main()
    {
        char letra;

        letra = 'a';
        Console.WriteLine("La letra es {0}", letra);

        Console.WriteLine("Introduce una nueva letra");
        letra = Convert.ToChar(Console.ReadLine());
        Console.WriteLine("Ahora la letra es {0}", letra);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 3.3.1.1: *Crea un programa que pida una letra al usuario y diga si se trata de una vocal.*

Ejercicio propuesto 3.3.1.2: *Crea un programa que muestre una de cada dos letras entre la que teclee el usuario y la "z". Por ejemplo, si el usuario introduce una "a", se escribirá "aceg...".*

Ejercicio propuesto 3.3.1.3: *Crea un programa que pida al usuario el ancho (por ejemplo, 4) y el alto (por ejemplo, 3) y una letra (por ejemplo, X) y escriba un rectángulo formado por esa cantidad de letras:*
XXXX
XXXX
XXXX

3.3.2. Secuencias de escape: \n y otras

Como hemos visto, los textos que aparecen en pantalla se escriben con WriteLine, indicados entre paréntesis y entre comillas dobles. Entonces surge una dificultad: ¿cómo escribimos una comilla doble en pantalla? La forma de conseguirlo es usando ciertos caracteres especiales, lo que se conoce como "secuencias de escape". Existen ciertos caracteres especiales que se pueden escribir después de una barra invertida (\) y que nos permiten conseguir escribir esas comillas dobles y algún otro carácter poco habitual. Por ejemplo, con \" se escribirán unas comillas dobles, con \' unas comillas simples, con \\ se escribe una barra invertida y con \n se avanzará a la línea siguiente de pantalla (es preferible evitar este último, porque puede no funcionar correctamente en todos los sistemas operativos; más adelante veremos una alternativa más segura).

Estas secuencias especiales son las siguientes:

Secuencia	**Significado**
\a	Emite un pitido
\b	Retroceso (permite borrar el último carácter)
\f	Avance de página (expulsa una hoja en la impresora)
\n	Avanza de línea (salta a la línea siguiente)
\r	Retorno de carro (va al principio de la línea)
\t	Salto de tabulación horizontal
\v	Salto de tabulación vertical
\'	Muestra una comilla simple
\"	Muestra una comilla doble
\\	Muestra una barra invertida
\0	Carácter nulo (NULL)

Vamos a ver un ejemplo que use los más habituales:

```
// Ejemplo_03_03_02a.cs
// Secuencias de escape
// Introducción a C#

using System;

public class Ejemplo_03_03_02a
{
    public static void Main()
    {
        Console.WriteLine("Esta es una frase");
```

```
        Console.WriteLine();
        Console.WriteLine();
        Console.WriteLine("y esta es otra, separada dos lineas");

        Console.WriteLine("\n\nJuguemos mas:\n\notro salto");
        Console.WriteLine("Comillas dobles: \" y simples \', y barra \\");
    }
}
```

En algunas ocasiones puede ser incómodo manipular estas secuencias de escape. Por ejemplo, cuando usemos estructuras de directorios al estilo de MsDos y Windows, deberíamos duplicar todas las barras invertidas: c:\\datos\\ejemplos\\curso\\ejemplo1. En este caso, se puede usar una arroba (@) antes del texto, en vez de usar las barras invertidas:

```
ruta = @"c:\datos\ejemplos\curso\ejemplo1"
```

En este caso, el problema está si aparecen comillas en medio de la cadena. Para solucionarlo, se duplican las comillas, así:

```
orden = @"copy ""documento de ejemplo"" f:"
```

Ejercicios propuestos:

Ejercicio propuesto 3.3.2.1: *Crea un programa que pida al usuario que teclee cuatro letras y las muestre en pantalla juntas, pero en orden inverso, y entre comillas dobles. Por ejemplo si las letras que se teclean son a, l, o, h, escribiría "hola".*

3.4. Toma de contacto con las cadenas de texto

Al contrario que en lenguajes más antiguos (como C), las cadenas de texto en C# son tan fáciles de manejar como los demás tipos de datos que hemos visto. Los detalles que hay que tener en cuenta en un primer acercamiento son:

- Se declaran con "string".
- Si queremos dar un valor inicial, éste se indica entre comillas dobles.
- Cuando leemos con ReadLine, no hace falta convertir el valor obtenido.
- Podemos comparar su valor usando "==" (igualdad) o "!=" (desigualdad).

Así, un ejemplo que diera un valor a un "string", lo mostrara (entre comillas, para practicar las secuencias de escape que hemos visto en el apartado anterior) y leyera un valor tecleado por el usuario podría ser:

```
// Ejemplo_03_04a.cs
// Uso basico de "string"
// Introducción a C#

using System;

public class Ejemplo_03_04a
{
    public static void Main()
```

```
    {
        string frase;

        frase = "Hola, como estas?";
        Console.WriteLine("La frase es \"{0}\"", frase);

        Console.WriteLine("Introduce una nueva frase");
        frase = Console.ReadLine();
        Console.WriteLine("Ahora la frase es \"{0}\"", frase);

        if (frase == "Hola!")
            Console.WriteLine("Hola a ti también! ");
    }
}
```

Se pueden hacer muchas más operaciones sobre cadenas de texto: convertir a mayúsculas o a minúsculas, eliminar espacios, cambiar una subcadena por otra, dividir en trozos, etc. Pero ya volveremos a las cadenas más adelante, en el próximo tema.

Ejercicios propuestos:

Ejercicio propuesto 3.4.1: *Crea un programa que pida al usuario su nombre, y le diga "Hola" si se llama "Juan", o bien le diga "No te conozco" si teclea otro nombre.*

Ejercicio propuesto 3.4.2: *Crea un programa que pida al usuario un nombre y una contraseña. La contraseña se debe introducir dos veces. Si las dos contraseñas no son iguales, se avisará al usuario y se le volverán a pedir las dos contraseñas.*

3.5. Los valores "booleanos"

En C# tenemos también un tipo de datos llamado "booleano" ("bool"), que puede tomar dos valores: verdadero ("true") o falso ("false"):

```
bool encontrado;
encontrado = true;
```

Este tipo de datos hará que podamos escribir de forma sencilla algunas condiciones que podrían resultar complejas. Así podemos hacer que ciertos fragmentos de nuestro programa no sean "if ((vidas == 0) || (tiempo == 0) || ((enemigos == 0) && (nivel == ultimoNivel)))" sino simplemente "if (partidaTerminada) ..."

A las variables "bool" también se le puede dar como valor el resultado de una comparación:

```
// Ejemplo básico
partidaTerminada = false;
if (vidas == 0) partidaTerminada = true;
// Notación alternativa, sin usar "if"
partidaTerminada = vidas == 0;

// Ejemplo más desarrollado
if (enemigos ==0) && (nivel == ultimoNivel)
    partidaTerminada = true;
```

```
else
    partidaTerminada = true;
// Notación alternativa, sin usar "if"
partidaTerminada = (enemigos ==0) && (nivel == ultimoNivel);
```

Lo emplearemos a partir de ahora en los fuentes que usen condiciones un poco complejas (es la alternativa más natural a los "break"). Un ejemplo que pida una letra y diga si es una vocal, una cifra numérica u otro símbolo, usando variables "bool" podría ser:

```
// Ejemplo_03_05a.cs
// Variables bool
// Introducción a C#

using System;

public class Ejemplo_03_05a
{
    public static void Main()
    {
        char letra;
        bool esVocal, esCifra;

        Console.WriteLine("Introduce una letra");
        letra = Convert.ToChar(Console.ReadLine());

        esCifra = (letra >= '0') && (letra <='9');

        esVocal = (letra == 'a') || (letra == 'e') || (letra == 'i') ||
            (letra == 'o') || (letra == 'u');

        if (esCifra)
            Console.WriteLine("Es una cifra numérica.");
        else if (esVocal)
            Console.WriteLine("Es una vocal.");
        else
            Console.WriteLine("Es una consonante u otro símbolo.");
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 3.5.1: *Crea un programa que use el operador condicional para dar a una variable llamada "iguales" (booleana) el valor "true" si los dos números que ha tecleado el usuario son iguales, o "false" si son distintos.*

Ejercicio propuesto 3.5.2: *Crea una versión alternativa del ejercicio 3.5.1, que use "if" en vez del operador condicional.*

Ejercicio propuesto 3.5.3: *Crea un programa que use el operador condicional para dar a una variable llamada "ambosPares" (booleana) el valor "true" si dos números introducidos por el usuario son pares, o "false" si alguno es impar.*