1. Toma de contacto con C#

C# es un lenguaje de programación de ordenadores. Se trata de un lenguaje moderno, evolucionado a partir de C y C++, y con una sintaxis muy similar a la de Java. Los programas creados con C# no suelen ser tan rápidos como los creados con C, pero a cambio la productividad del programador es mucho mayor.

Se trata de un lenguaje creado por Microsoft para crear programas para su plataforma .NET, pero estandarizado posteriormente por ECMA y por ISO, y del que existe una implementación alternativa de "código abierto", el "proyecto Mono", que está disponible para Windows, Linux, Mac OS X y otros sistemas operativos.

Nosotros comenzaremos por usar Mono como plataforma de desarrollo durante los primeros temas. Cuando los conceptos básicos estén asentados, pasaremos a emplear Visual C#, de Microsoft, que requiere un ordenador más potente pero a cambio incluye un entorno de desarrollo muy avanzado, y está disponible también en una versión gratuita (Visual Studio Express Edition).

Los **pasos** que seguiremos para crear un programa en C# serán:

1. Escribir el programa en lenguaje C# (**fichero fuente**), con cualquier editor de textos.
2. Compilarlo con nuestro compilador. Esto creará un **"fichero ejecutable"**.
3. Lanzar el fichero ejecutable.

La mayoría de los compiladores actuales permiten dar todos estos pasos desde un único **entorno**, en el que escribimos nuestros programas, los compilamos, y los depuramos en caso de que exista algún fallo.

En el siguiente apartado veremos un ejemplo de uno de estos entornos, dónde localizarlo y cómo instalarlo.

1.1 Escribir un texto en C#

Vamos con un primer ejemplo de programa en C#, posiblemente el más sencillo de los que "hacen algo útil". Se trata de escribir un texto en pantalla. La apariencia de este programa la vimos en el tema anterior. Vamos a verlo ahora con más detalle:

```
public class Ejemplo01
{
    public static void Main()
    {
        System.Console.WriteLine("Hola");
    }
}
```

Esto escribe "Hola" en la pantalla. Pero hay mucho alrededor de ese "Hola", y vamos a comentarlo antes de proseguir, aunque muchos de los detalles se irán aclarando más adelante. En este primer análisis, iremos de dentro hacia fuera:

- `WriteLine("Hola");` - "Hola" es el texto que queremos escribir, y WriteLine es la orden encargada de escribir (Write) una línea (Line) de texto en pantalla.
- `Console.WriteLine("Hola");` porque WriteLine es una orden de manejo de la "consola" (la pantalla "negra" en modo texto del sistema operativo).
- `System.Console.WriteLine("Hola");` porque las órdenes relacionadas con el manejo de consola (Console) pertenecen a la categoría de sistema (System).
- Las llaves { y } se usan para delimitar un bloque de programa. En nuestro caso, se trata del bloque principal del programa.
- `public static void Main()` - Main indica cual es "el cuerpo del programa", la parte principal (un programa puede estar dividido en varios fragmentos, como veremos más adelante). Todos los programas tienen que tener un bloque "Main". Los detalles de por qué hay que poner delante "public static void" y de por qué se pone después un paréntesis vacío los iremos aclarando más tarde. De momento, deberemos memorizar que ésa será la forma correcta de escribir "Main".
- `public class Ejemplo01` - de momento pensaremos que "Ejemplo01" es el nombre de nuestro programa. Una línea como esa deberá existir también siempre en nuestros programas, y eso de "public class" será obligatorio. Nuevamente, aplazamos para más tarde los detalles sobre qué quiere decir "class" y por qué debe ser "public".

Como se puede ver, mucha parte de este programa todavía es casi un "acto de fe" para nosotros. Debemos creernos que "se debe hacer así". Poco a poco iremos detallando el por qué de "public", de "static", de "void", de "class"... Por ahora nos limitaremos a "rellenar" el cuerpo del programa para entender los conceptos básicos de programación.

Ejercicio propuesto (1.1.1): Crea un programa en C# que te salude por tu nombre (ej: "Hola, Nacho").

Sólo un par de cosas más antes de seguir adelante:

- Cada orden de C# debe terminar con un **punto y coma** (;)
- C# es un lenguaje de **formato libre**, de modo que puede haber varias órdenes en una misma línea, u órdenes separadas por varias líneas o espacios entre medias. Lo que realmente indica donde termina una orden y donde empieza la siguiente son los puntos y coma. Por ese motivo, el programa anterior se podría haber escrito también así (aunque no es aconsejable, porque puede resultar menos legible):

```
public class Ejemplo01 {
public
static
void Main() { System.Console.WriteLine("Hola"); }  }
```

- De hecho, hay dos formas especialmente frecuentes de colocar la llave de comienzo, y yo usaré ambas indistintamente. Una es como hemos hecho en el primer ejemplo: situar la llave de apertura en una línea, sola, y justo encima de la llave de cierre correspondiente. Esto es lo que muchos autores llaman el "estilo C". La segunda forma habitual es situándola a continuación del nombre del bloque que comienza (el "estilo Java"), así:

```
public class Ejemplo01 {
```

```
    public static void Main(){
        System.Console.WriteLine("Hola");
    }
}
```

(esta es la forma que yo emplearé preferentemente en este texto cuando estemos trabajando con fuentes de mayor tamaño, para que ocupe un poco menos de espacio).

- La gran mayoría de las órdenes que encontraremos en el lenguaje C# son palabras en inglés o abreviaturas de éstas. Pero hay que tener en cuenta que C# **distingue entre mayúsculas** y minúsculas, por lo que "WriteLine" es una palabra reconocida, pero "writeLine", "WRITELINE" o "Writeline" no lo son.

1.2. Cómo probar este programa

1.2.1 Cómo probarlo con Mono

Como ya hemos comentado, usaremos Mono como plataforma de desarrollo para nuestros primeros programas. Por eso, vamos a comenzar por ver dónde encontrar esta herramienta, cómo instalarla y cómo utilizarla.

Podemos descargar Mono desde su página oficial:

`http://www.mono-project.com/`

En la parte superior derecha aparece el enlace para descargar ("download now"), que nos lleva a una nueva página en la que debemos elegir la plataforma para la que queremos nuestro Mono. Nosotros descargaremos la versión más reciente para Windows (la 2.10.5 en el momento de escribir este texto).

```
public class Ejemplo01b
{
    public static void Main()
    {
        System.Console.WriteLine("Hola");
        System.Console.ReadLine();
    }
}
```

1.3. Mostrar números enteros en pantalla

Cuando queremos escribir un texto "tal cual", como en el ejemplo anterior, lo encerramos entre comillas. Pero no siempre querremos escribir textos prefijados. En muchos casos, se tratará de algo que habrá que calcular.

El ejemplo más sencillo es el de una operación matemática. La forma de realizarla es sencilla: no usar comillas en WriteLine. Entonces, C# intentará analizar el contenido para ver qué quiere decir. Por ejemplo, para sumar 3 y 4 bastaría hacer:

```
public class Ejemplo01suma
{
    public static void Main()
    {
        System.Console.WriteLine(3+4);
    }
}
```

Ejercicios propuestos:

- **(1.3.1)** Crea un programa que diga el resultado de sumar 118 y 56.
- **(1.3.2)** Crea un programa que diga el resultado de sumar 12345 y 67890.

1.4. Operaciones aritméticas básicas

1.4.1. Operadores

Está claro que el símbolo de la suma será un +, y podemos esperar cual será el de la resta, pero alguna de las operaciones matemáticas habituales tienen símbolos menos intuitivos. Veamos cuales son los más importantes:

Operador	Operación
+	Suma
-	Resta, negación
*	Multiplicación
/	División
%	Resto de la división ("módulo")

Ejercicios propuestos:

- **(1.4.1.1)** Hacer un programa que calcule el producto de los números 12 y 13.
- **(1.4.1.2)** Hacer un programa que calcule la diferencia (resta) entre 321 y 213.
- **(1.4.1.3)** Hacer un programa que calcule el resultado de dividir 301 entre 3.
- **(1.4.1.4)** Hacer un programa que calcule el resto de la división de 301 entre 3.

1.4.2. Orden de prioridad de los operadores

Sencillo:

- En primer lugar se realizarán las operaciones indicadas entre paréntesis.
- Luego la negación.
- Después las multiplicaciones, divisiones y el resto de la división.
- Finalmente, las sumas y las restas.
- En caso de tener igual prioridad, se analizan de izquierda a derecha.

Ejercicios propuestos: Calcular (a mano y después comprobar desde C#) el resultado de las siguientes operaciones:

- **(1.4.2.1)** Calcular el resultado de -2 + 3 * 5
- **(1.4.2.2)** Calcular el resultado de (20+5) % 6
- **(1.4.2.3)** Calcular el resultado de 15 + -5*6 / 10
- **(1.4.2.4)** Calcular el resultado de 2 + 10 / 5 * 2 - 7 % 1

1.4.3. Introducción a los problemas de desbordamiento

El espacio del que disponemos para almacenar los números es limitado. Si el resultado de una operación es un número "demasiado grande", obtendremos un mensaje de error o un resultado erróneo. Por eso en los primeros ejemplos usaremos números pequeños. Más adelante veremos a qué se debe realmente este problema y cómo evitarlo. Como anticipo, el siguiente programa ni siquiera compila, porque el compilador sabe que el resultado va a ser "demasiado grande":

```
public class Ejemplo01multiplic
{
    public static void Main()
    {
        System.Console.WriteLine(10000000*10000000);
    }
}
```

1.5. Introducción a las variables: int

Las **variables** son algo que no contiene un valor predeterminado, un espacio de memoria al que nosotros asignamos un nombre y en el que podremos almacenar datos.

El primer ejemplo nos permitía escribir "Hola". El segundo nos permitía sumar dos números que habíamos prefijado en nuestro programa. Pero esto tampoco es "lo habitual", sino que esos números dependerán de valores que haya tecleado el usuario o de cálculos anteriores.

Por eso necesitaremos usar variables, zonas de memoria en las que guardemos los datos con los que vamos a trabajar y también los resultados temporales. Como primer ejemplo, vamos a ver lo que haríamos para sumar dos números enteros que fijásemos en el programa.

1.5.1. Definición de variables: números enteros

Para usar una cierta variable primero hay que **declararla**: indicar su nombre y el tipo de datos que querremos guardar.

El primer tipo de datos que usaremos serán números enteros (sin decimales), que se indican con "int" (abreviatura del inglés "integer"). Después de esta palabra se indica el nombre que tendrá la variable:

```
int primerNumero;
```

Esa orden reserva espacio para almacenar un número entero, que podrá tomar distintos valores, y al que nos referiremos con el nombre "primerNumero".

1.5.2. Asignación de valores

Podemos darle un valor a esa variable durante el programa haciendo

```
primerNumero = 234;
```

O también podemos darles un valor inicial ("inicializarlas") antes de que empiece el programa, en el mismo momento en que las definimos:

```
int primerNumero = 234;
```

O incluso podemos definir e inicializar más de una variable a la vez

```
int primerNumero = 234, segundoNumero = 567;
```

(esta línea reserva espacio para dos variables, que usaremos para almacenar números enteros; una de ellas se llama primerNumero y tiene como valor inicial 234 y la otra se llama segundoNumero y tiene como valor inicial 567).

Después ya podemos hacer operaciones con las variables, igual que las hacíamos con los números:

```
suma = primerNumero + segundoNumero;
```

1.5.3. Mostrar el valor de una variable en pantalla

Una vez que sabemos cómo mostrar un número en pantalla, es sencillo mostrar el valor de una variable. Para un número hacíamos cosas como

```
System.Console.WriteLine(3+4);
```

pero si se trata de una variable es idéntico:

```
System.Console.WriteLine(suma);
```

O bien, si queremos mostrar un texto además del valor de la variable, podemos indicar el texto entre comillas, detallando con {0} en qué parte del texto queremos que aparezca el valor de la variable, de la siguiente forma:

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

Si se trata de más de una variable, indicaremos todas ellas tras el texto, y detallaremos dónde debe aparecer cada una de ellas, usando {0}, {1} y así sucesivamente:

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

Ya sabemos todo lo suficiente para crear nuestro programa que sume dos números usando variables:

```
public class Ejemplo02
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        primerNumero = 234;
        segundoNumero = 567;
        suma = primerNumero + segundoNumero;

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

Repasemos lo que hace:

- (Nos saltamos todavía los detalles de qué quieren decir "public", "class", "static" y "void").
- *Main()* indica donde comienza el cuerpo del programa, que se delimita entre llaves { y }
- *int primerNumero;* reserva espacio para guardar un número entero, al que llamaremos primerNumero.
- *int segundoNumero;* reserva espacio para guardar otro número entero, al que llamaremos segundoNumero.
- *int suma;* reserva espacio para guardar un tercer número entero, al que llamaremos suma.
- *primerNumero = 234;* da el valor del primer número que queremos sumar
- *segundoNumero = 567;* da el valor del segundo número que queremos sumar
- *suma = primerNumero + segundoNumero;* halla la suma de esos dos números y la guarda en otra variable, en vez de mostrarla directamente en pantalla.

- *System.Console.WriteLine("La suma de {0} y {1} es {2}", primerNumero, segundoNumero, suma);* muestra en pantalla el texto y los valores de las tres variables (los dos números iniciales y su suma).

Ejercicios propuestos:

- **(1.5.3.1)** Crea un programa que calcule el producto de los números 121 y 132, usando variables.
- **(1.5.3.2)** Crea un programa que calcule la suma de 285 y 1396, usando variables.
- **(1.5.3.3)** Crea un programa que calcule el resto de dividir 3784 entre 16, usando variables.

1.6. Identificadores

Estos nombres de variable (lo que se conoce como "**identificadores**") pueden estar formados por letras, números o el símbolo de subrayado (_) y deben comenzar por letra o subrayado. No deben tener espacios entre medias, y hay que recordar que las vocales acentuadas y la eñe son problemáticas, porque no son letras "estándar" en todos los idiomas.

Por eso, no son nombres de variable válidos:

```
1numero        (empieza por número)
un numero      (contiene un espacio)
Año1           (tiene una eñe)
MásDatos       (tiene una vocal acentuada)
```

Tampoco podremos usar como identificadores las **palabras reservadas** de C#. Por ejemplo, la palabra "int" se refiere a que cierta variable guardará un número entero, así que esa palabra "int" no la podremos usar tampoco como nombre de variable (pero no vamos a incluir ahora una lista de palabras reservadas de C#, ya nos iremos encontrando con ellas).

De momento, intentaremos usar nombres de variables que a nosotros nos resulten claros, y que no parezca que puedan ser alguna orden de C#.

Hay que recordar que en C# las **mayúsculas y minúsculas** se consideran diferentes, de modo que si intentamos hacer

```
PrimerNumero = 0;
primernumero = 0;
```

o cualquier variación similar, el compilador protestará y nos dirá que no conoce esa variable, porque la habíamos declarado como

```
int primerNumero;
```

1.7. Comentarios

Podemos escribir comentarios, que el compilador ignora, pero que pueden servir para aclararnos cosas a nosotros. Se escriben entre /* y */:

```
int suma;  /* Porque guardaré el valor para usarlo más tarde */
```

Es conveniente escribir comentarios que aclaren la misión de las partes de nuestros programas que puedan resultar menos claras a simple vista. Incluso suele ser aconsejable que el programa comience con un comentario, que nos recuerde qué hace el programa sin que necesitemos mirarlo de arriba a abajo. Un ejemplo casi exagerado:

```
/* ---- Ejemplo en C#: sumar dos números prefijados ---- */

public class Ejemplo02b
{
    public static void Main()
    {
        int primerNumero = 234;
        int segundoNumero = 567;
        int suma;  /* Guardaré el valor para usarlo más tarde */

        /* Primero calculo la suma */
        suma = primerNumero + segundoNumero;

        /* Y después muestro su valor */
        System.Console.WriteLine("La suma de {0} y {1} es {2}",
          primerNumero, segundoNumero, suma);
    }
}
```

Un comentario puede empezar en una línea y terminar en otra distinta, así:

```
/*  Esto
    es un comentario que
    ocupa más de una línea
*/
```

También es posible declarar otro tipo de comentarios, que comienzan con doble barra y terminan cuando se acaba la línea (estos comentarios, claramente, no podrán ocupar más de una línea). Son los "comentarios al estilo de C++":

```
//  Este es un comentario "al estilo C++"
```

1.8. Datos por el usuario: ReadLine

Si queremos que sea el usuario de nuestro programa quien teclee los valores, necesitamos una nueva orden, que nos permita leer desde teclado. Pues bien, al igual que tenemos System.Console.WriteLine ("escribir línea), también existe System.Console.ReadLine ("leer línea"). Para leer textos, haríamos

```
texto = System.Console.ReadLine();
```

pero eso ocurrirá en el próximo tema, cuando veamos cómo manejar textos. De momento, nosotros sólo sabemos manipular números enteros, así que deberemos convertir ese dato a un número entero, usando Convert.ToInt32:

```
primerNumero = System.Convert.ToInt32( System.Console.ReadLine() );
```

Un ejemplo de programa que sume dos números tecleados por el usuario sería:

```
public class Ejemplo03
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        System.Console.WriteLine("Introduce el primer número");
        primerNumero = System.Convert.ToInt32(
          System.Console.ReadLine());
        System.Console.WriteLine("Introduce el segundo número");
        segundoNumero = System.Convert.ToInt32(
          System.Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Ejercicios propuestos:

- **(1.8.1)** Crea un programa que calcule el producto de dos números introducidos por el usuario.
- **(1.8.2)** Crea un programa que calcule la división de dos números introducidos por el usuario, así como el resto de esa división.

1.9. Pequeñas mejoras

Va siendo hora de hacer una pequeña mejora: no es necesario repetir "System." al principio de la mayoría de las órdenes que tienen que ver con el sistema (por ahora, las de consola y las de conversión), si al principio del programa utilizamos "using System":

```
using System;

public class Ejemplo04
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        Console.WriteLine("Introduce el primer número");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Introduce el segundo número");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

```
}
```

Podemos declarar varias variables a la vez, si van a almacenar datos del mismo tipo. Para hacerlo, tras el tipo de datos indicaríamos todos sus nombres, separados por comas:

```
using System;

public class Ejemplo04b
{
    public static void Main()
    {
        int primerNumero, segundoNumero, suma;

        Console.WriteLine("Introduce el primer número");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Introduce el segundo número");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Y podemos escribir sin avanzar a la línea siguiente de pantalla, si usamos "Write" en vez de "WriteLine":

```
using System;

public class Ejemplo04c
{
    public static void Main()
    {
        int primerNumero, segundoNumero, suma;

        Console.Write("Introduce el primer número");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.Write("Introduce el segundo número");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Y ahora que conocemos los fundamentos, puede ser el momento de pasar a un editor de texto un poco más avanzado. Por ejemplo, en Windows podemos usar Notepad++, que es gratuito, destaca la sintaxis en colores, muestra la línea y columna en la que nos encontramos, ayuda a encontrar las llaves emparejadas, realza la línea en la que nos encontramos, tiene soporte para múltiples ventanas, etc.:

2. Estructuras de control

2.1. Estructuras alternativas

2.1.1. if

Vamos a ver cómo podemos comprobar si se cumplen condiciones. La primera construcción que usaremos será **"si ... entonces ..**.". El formato es

```
if (condición) sentencia;
```

Vamos a verlo con un ejemplo:

```
/*---------------------------*/
/*  Ejemplo en C# nº 5:      */
/*  ejemplo05.cs             */
/*                           */
/*  Condiciones con if       */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo05
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero>0) Console.WriteLine("El número es positivo.");
    }
}
```

Este programa pide un número al usuario. Si es positivo (mayor que 0), escribe en pantalla "El número es positivo."; si es negativo o cero, no hace nada.

Como se ve en el ejemplo, para comprobar si un valor numérico es mayor que otro, usamos el símbolo ">". Para ver si dos valores son iguales, usaremos dos símbolos de "igual": `if (numero==0)`. Las demás posibilidades las veremos algo más adelante. En todos los casos, la condición que queremos comprobar deberá indicarse entre paréntesis.

Este programa comienza por un comentario que nos recuerda de qué se trata. Como nuestros fuentes irán siendo cada vez más complejos, a partir de ahora incluiremos comentarios que nos permitan recordar de un vistazo qué pretendíamos hacer.

Si la orden "if" es larga, se puede partir en dos líneas para que resulte más legible:

```
        if (numero>0)
            Console.WriteLine("El número es positivo.");
```

Ejercicios propuestos:

- **(2.1.1.1)** Crear un programa que pida al usuario un número entero y diga si es par (pista: habrá que comprobar si el resto que se obtiene al dividir entre dos es cero: if (x % 2 == 0) ...).
- **(2.1.1.2)** Crear un programa que pida al usuario dos números enteros y diga cuál es el mayor de ellos.
- **(2.1.1.3)** Crear un programa que pida al usuario dos números enteros y diga si el primero es múltiplo del segundo (pista: igual que antes, habrá que ver si el resto de la división es cero: a % b == 0).

2.1.2. if y sentencias compuestas

Habíamos dicho que el formato básico de "if" es `if (condición) sentencia;` Esa "sentencia" que se ejecuta si se cumple la condición puede ser una sentencia simple o una compuesta. Las sentencias **compuestas** se forman agrupando varias sentencias simples entre llaves (**{** y **}**), como en este ejemplo:

```
/*---------------------------*/
/*  Ejemplo en C# nº 6:      */
/*  ejemplo06.cs             */
/*                           */
/*  Condiciones con if (2)   */
/*  Sentencias compuestas    */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo06
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());
    if (numero > 0)
    {
      Console.WriteLine("El número es positivo.");
      Console.WriteLine("Recuerde que también puede usar negativos.");
    }  /* Aquí acaba el "if" */
  }    /* Aquí acaba "Main" */
}      /* Aquí acaba "Ejemplo06" */
```

En este caso, si el número es positivo, se hacen dos cosas: escribir un texto y luego... ¡escribir otro! (En este ejemplo, esos dos "WriteLine" podrían ser uno solo, en el que los dos textos estuvieran separados por un carácter especial, el símbolo de "salto de línea"; más adelante

iremos encontrando casos en lo que necesitemos hacer cosas "más serias" dentro de una sentencia compuesta).

Como se ve en este ejemplo, cada nuevo "bloque" se suele escribir un poco más a la derecha que los anteriores, para que sea fácil ver dónde comienza y termina cada sección de un programa. Por ejemplo, el contenido de "Ejemplo06" está un poco más a la derecha que la cabecera "public class Ejemplo06", y el contenido de "Main" algo más a la derecha, y la sentencia compuesta que se debe realizar si se cumple la condición del "if" está algo más a la derecha que la orden "if". Este "**sangrado**" del texto se suele llamar "**escritura indentada**". Un tamaño habitual para el sangrado es de 4 espacios, aunque en este texto muchas veces usaremos sólo dos espacios, para no llegar al margen derecho del papel con demasiada facilidad.

Ejercicios propuestos:

- **(2.1.2.1)** Crear un programa que pida al usuario un número entero. Si es múltiplo de 10, se lo avisará al usuario y pedirá un segundo número, para decir a continuación si este segundo número también es múltiplo de 10.

2.1.3. Operadores relacionales: <, <=, >, >=, ==, !=

Hemos visto que el símbolo ">" es el que se usa para comprobar si un número es mayor que otro. El símbolo de "menor que" también es sencillo, pero los demás son un poco menos evidentes, así que vamos a verlos:

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (distinto de)

Así, un ejemplo, que diga si un número NO ES cero sería:

```
/*----------------------------*/
/*  Ejemplo en C# nº 7:       */
/*  ejemplo07.cs              */
/*                            */
/*  Condiciones con if (3)    */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo07
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());
```

```
    if (numero != 0)
      Console.WriteLine("El número no es cero.");
  }
}
```

Ejercicios propuestos:

- **(2.1.3.1)** Crear un programa que multiplique dos números enteros de la siguiente forma: pedirá al usuario un primer número entero. Si el número que se que teclee es 0, escribirá en pantalla "El producto de 0 por cualquier número es 0". Si se ha tecleado un número distinto de cero, se pedirá al usuario un segundo número y se mostrará el producto de ambos.
- **(2.1.3.2)** Crear un programa que pida al usuario dos números enteros. Si el segundo no es cero, mostrará el resultado de dividir entre el primero y el segundo. Por el contrario, si el segundo número es cero, escribirá "Error: No se puede dividir entre cero".

2.1.4. if-else

Podemos indicar lo que queremos que ocurra en caso de que no se cumpla la condición, usando la orden "else" (en caso contrario), así:

```
/*----------------------------*/
/*  Ejemplo en C# nº 8:       */
/*  ejemplo08.cs              */
/*                            */
/*  Condiciones con if (4)    */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo08
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());
    if (numero > 0)
      Console.WriteLine("El número es positivo.");
    else
      Console.WriteLine("El número es cero o negativo.");
  }
}
```

Podríamos intentar evitar el uso de "else" si utilizamos un "if" a continuación de otro, así:

```
/*----------------------------*/
/*  Ejemplo en C# nº 9:       */
/*  ejemplo09.cs              */
```

```
/*                           */
/*  Condiciones con if (5)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo09
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());

    if (numero > 0)
      Console.WriteLine("El número es positivo.");

    if (numero <= 0)
      Console.WriteLine("El número es cero o negativo.");
  }
}
```

Pero el comportamiento **no es el mismo**: en el primer caso (ejemplo 8) se mira si el valor es positivo; si no lo es, se pasa a la segunda orden, pero si lo es, el programa ya ha terminado. En el segundo caso (ejemplo 9), aunque el número sea positivo, se vuelve a realizar la segunda comprobación para ver si es negativo o cero, por lo que el programa es algo más lento.

Podemos enlazar varios "if" usando "else", para decir "si no se cumple esta condición, mira a ver si se cumple esta otra":

```
/*---------------------------*/
/*  Ejemplo en C# nº 10:     */
/*  ejemplo10.cs             */
/*                           */
/*  Condiciones con if (6)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo10
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());

    if (numero > 0)
      Console.WriteLine("El número es positivo.");
```

```
    else
      if (numero < 0)
        Console.WriteLine("El número es negativo.");
      else
        Console.WriteLine("El número es cero.");
  }
}
```

Ejercicio propuesto:

- **(2.1.4.1)** Mejorar la solución al ejercicio 2.1.3.1, usando "else".
- **(2.1.4.2)** Mejorar la solución al ejercicio 2.1.3.2, usando "else".

2.1.5. Operadores lógicos: &&, ||, !

Estas condiciones se puede **encadenar** con "y", "o", etc., que se indican de la siguiente forma

Operador	Significado
&&	Y
||	O
!	No

De modo que podremos escribir cosas como

```
if ((opcion==1) && (usuario==2))  ...
if ((opcion==1) || (opcion==3))   ...
if ((!(opcion==opcCorrecta)) || (tecla==ESC))   ...
```

Ejercicios propuestos:

- **(2.1.5.1)** Crear un programa que pida al usuario un número enteros y diga si es múltiplo de 2 o de 3.
- **(2.1.5.2)** Crear un programa que pida al usuario dos números enteros y diga "Uno de los números es positivo", "Los dos números son positivos" o bien "Ninguno de los números es positivo", según corresponda.
- **(2.1.5.3)** Crear un programa que pida al usuario tres números reales y muestre cuál es el mayor de los tres.
- **(2.1.5.4)** Crear un programa que pida al usuario dos números enteros cortos y diga si son iguales o, en caso contrario, cuál es el mayor de ellos.

2.1.6. El peligro de la asignación en un "if"

Cuidado con el operador de **igualdad**: hay que recordar que el formato es if (a==b) ... Si no nos acordamos y escribimos if (a=b), estamos intentando asignar a "a" el valor de "b".

En algunos compiladores de lenguaje C, esto podría ser un problema serio, porque se considera válido hacer una asignación dentro de un "if" (aunque la mayoría de compiladores modernos nos avisarían de que quizá estemos asignando un valor sin pretenderlo, pero no es un "error" sino un "aviso", lo que permite que se genere un ejecutable, y podríamos pasar por alto el aviso, dando lugar a un funcionamiento incorrecto de nuestro programa).

En el caso del lenguaje C#, este riesgo no existe, porque la "condición" debe ser algo cuyo resultado "verdadero" o "falso" (un dato de tipo "bool"), de modo que obtendríamos un error de

compilación "Cannot implicitly convert type 'int' to 'bool'" (*no puedo convertir un "int" a "bool"*). Es el caso del siguiente programa:

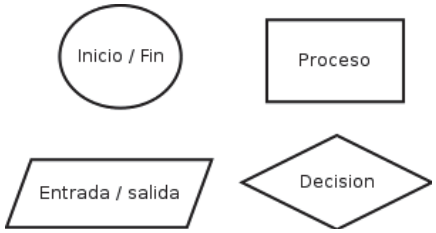
```
/*---------------------------*/
/*  Ejemplo en C# nº 11:     */
/*  ejemplo11.cs             */
/*                           */
/*  Condiciones con if (7)   */
/*    comparacion incorrecta */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo11
{
  public static void Main()
  {
    int numero;

    Console.WriteLine("Introduce un número");
    numero = Convert.ToInt32(Console.ReadLine());
    if (numero = 0)
      Console.WriteLine("El número es cero.");
    else
      if (numero < 0)
        Console.WriteLine("El número es negativo.");
      else
        Console.WriteLine("El número es positivo.");
  }
}
```

2.1.7. Introducción a los diagramas de flujo

A veces puede resultar difícil ver claro donde usar un "else" o qué instrucciones de las que siguen a un "if" deben ir entre llaves y cuales no. Generalmente la dificultad está en el hecho de intentar teclear directamente un programa en C#, en vez de pensar en el problema que se pretende resolver.

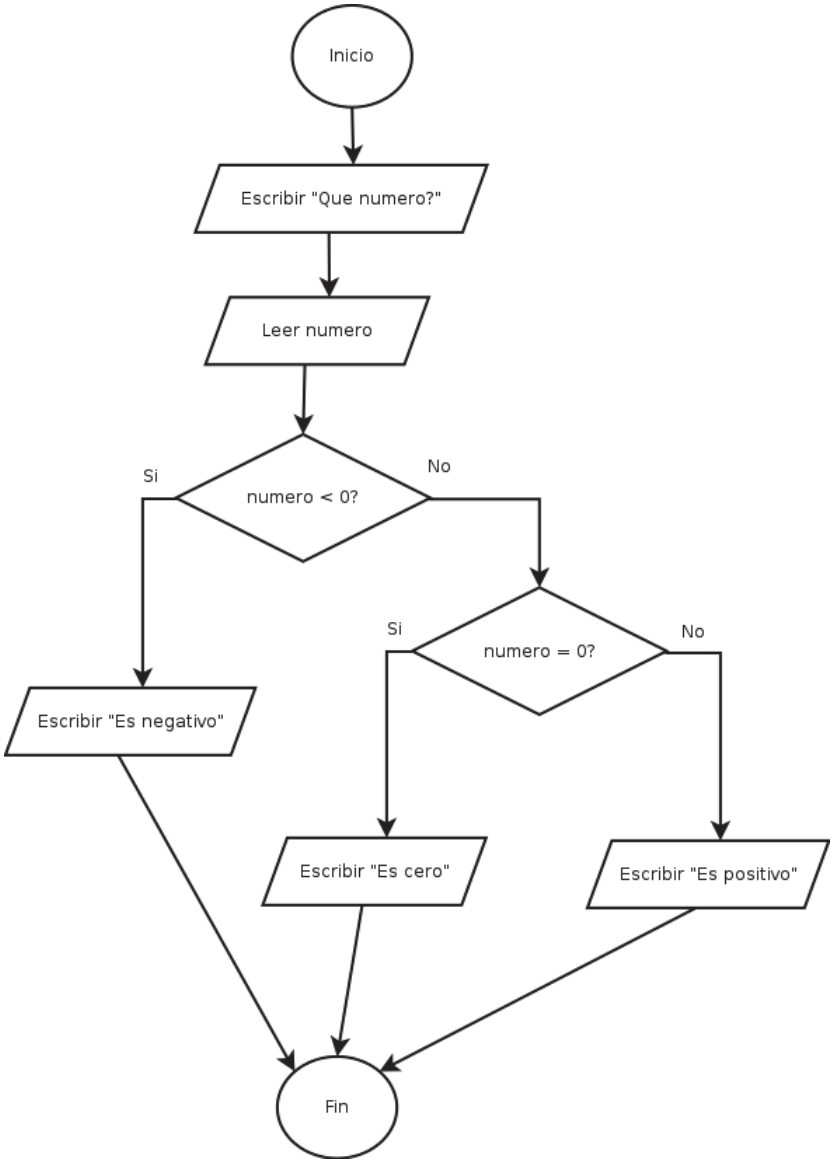
Para ayudarnos a centrarnos en el problema, existen notaciones gráficas, como los diagramas de flujo, que nos permiten ver mejor qué se debe hacer y cuando.

En primer lugar, vamos a ver los 4 elementos básicos de un diagrama de flujo, y luego los aplicaremos a un caso concreto.



El inicio o el final del programa se indica dentro de un círculo. Los procesos internos, como realizar operaciones, se encuadran en un rectángulo. Las entradas y salidas (escrituras en pantalla y lecturas de teclado) se indican con un paralelogramo que tenga su lados superior e inferior horizontales, pero no tenga verticales los otros dos. Las decisiones se indican dentro de un rombo.

Vamos a aplicarlo al ejemplo de un programa que pida un número al usuario y diga si es positivo, negativo o cero:



El paso de aquí al correspondiente programa en lenguaje C# (el que vimos en el ejemplo 11) debe ser casi inmediato: sabemos como leer de teclado, como escribir en pantalla, y las decisiones serán un "if", que si se cumple ejecutará la sentencia que aparece en su salida "si" y si no se cumple ("else") ejecutará lo que aparezca en su salida "no".

Ejercicios propuestos:

- **(2.1.7.1)** Crear el diagrama de flujo para el programa que pide al usuario dos números y dice si uno de ellos es positivo, si lo son los dos o si no lo es ninguno.
- **(2.1.7.2)** Crear el diagrama de flujo para el programa que pide tres números al usuario y dice cuál es el mayor de los tres.

2.1.8. Operador condicional: ?

En C# hay otra forma de asignar un valor según se cumpla una condición o no. Es el "**operador condicional** **?** : que se usa

```
nombreVariable = condicion ?  valor1 : valor2;
```

y equivale a decir "si se cumple la condición, toma el valor *valor1*; si no, toma el valor *valor2*".
Un ejemplo de cómo podríamos usarlo sería para calcular el mayor de dos números:

```
numeroMayor = a>b ?  a : b;
```

esto equivale a la siguiente orden "if":

```
if ( a > b )
    numeroMayor = a;
else
    numeroMayor = b;
```

Aplicado a un programa sencillo, podría ser

```
/*---------------------------*/
/*  Ejemplo en C# nº 12:     */
/*  ejemplo12.cs             */
/*                           */
/*  El operador condicional  */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo12
{
  public static void Main()
  {
    int a, b, mayor;

    Console.Write("Escriba un número: ");
    a = Convert.ToInt32(Console.ReadLine());
```

```
    Console.Write("Escriba otro: ");
    b = Convert.ToInt32(Console.ReadLine());

    mayor = (a>b) ?  a : b;

    Console.WriteLine("El mayor de los números es {0}.", mayor);
  }
}
```

(La orden Console.Write, empleada en el ejemplo anterior, escribe un texto sin avanzar a la línea siguiente, de modo que el próximo texto que escribamos –o introduzcamos- quedará a continuación de éste).

Un segundo ejemplo, que sume o reste dos números según la opción que se escoja, sería:

```
/*---------------------------*/
/*  Ejemplo en C# nº 13:     */
/*  ejemplo13.cs             */
/*                           */
/*  Operador condicional - 2 */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo13
{
  public static void Main()
  {
    int a, b, operacion, resultado;

    Console.Write("Escriba un número: ");
    a = Convert.ToInt32(Console.ReadLine());

    Console.Write("Escriba otro: ");
    b = Convert.ToInt32(Console.ReadLine());

    Console.Write("Escriba una operación (1 = resta; otro = suma): ");
    operacion = Convert.ToInt32(Console.ReadLine());

    resultado = (operacion == 1) ?  a-b : a+b;
    Console.WriteLine("El resultado es {0}.", resultado);
  }
}
```

Ejercicios propuestos:

- **(2.1.8.1)** Crear un programa que use el operador condicional para mostrar un el valor absoluto de un número de la siguiente forma: si el número es positivo, se mostrará tal cual; si es negativo, se mostrará cambiado de signo.
- **(2.1.8.2)** Usar el operador condicional para calcular el menor de dos números.

2.1.10. switch

Si queremos ver **varios posibles valores**, sería muy pesado tener que hacerlo con muchos "if" seguidos o encadenados. La alternativa es la orden "switch", cuya sintaxis es

```
switch (expresión)
{
   case valor1: sentencia1;
          break;
   case valor2: sentencia2;
          sentencia2b;
          break;
    ...
   case valorN: sentenciaN;
          break;
  default:
          otraSentencia;
          break;
}
```

Es decir, se escribe tras "**switch**" la expresión a analizar, entre paréntesis. Después, tras varias órdenes "**case**" se indica cada uno de los valores posibles. Los pasos (porque pueden ser varios) que se deben dar si se trata de ese valor se indican a continuación, terminando con "**break**". Si hay que hacer algo en caso de que no se cumpla ninguna de las condiciones, se detalla después de la palabra "**default**". Si dos casos tienen que hacer lo mismo, se añade "**goto case**" a uno de ellos para indicarlo.

Vamos con un ejemplo, que diga si el símbolo que introduce el usuario es una cifra numérica, un espacio u otro símbolo. Para ello usaremos un dato de tipo "**char**" (carácter), que veremos con más detalle en el próximo tema. De momento nos basta que deberemos usar Convert.ToChar si lo leemos desde teclado con ReadLine, y que le podemos dar un valor (o compararlo) usando comillas simples:

```
/*---------------------------*/
/*  Ejemplo en C# nº 14:     */
/*  ejemplo14.cs             */
/*                           */
/*  La orden "switch"  (1)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo14
{
  public static void Main()
  {
    char letra;

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

    switch (letra)
    {
      case ' ': Console.WriteLine("Espacio.");
```

```
                break;
      case '1': goto case '0';
      case '2': goto case '0';
      case '3': goto case '0';
      case '4': goto case '0';
      case '5': goto case '0';
      case '6': goto case '0';
      case '7': goto case '0';
      case '8': goto case '0';
      case '9': goto case '0';
      case '0': Console.WriteLine("Dígito.");
                break;
      default:  Console.WriteLine("Ni espacio ni dígito.");
                break;
    }
  }
}
```

Cuidado quien venga del lenguaje C: en C se puede dejar que un caso sea manejado por el siguiente, lo que se consigue si no se usa "break", mientras que C# siempre obliga a usar "break" o "goto" al final de cada cada caso, con la **única excepción** de que un caso no haga absolutamente nada que no sea dejar pasar el control al siguiente caso, y en ese caso se puede dejar totalmente vacío:

```
/*---------------------------*/
/*  Ejemplo en C# nº 14b:    */
/*  ejemplo14b.cs            */
/*                           */
/*  La orden "switch"  (1b)  */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo14b
{
  public static void Main()
  {
    char letra;

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

    switch (letra)
    {
      case ' ': Console.WriteLine("Espacio.");
                break;
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
```

```
      case '8':
      case '9':
      case '0': Console.WriteLine("Dígito.");
                break;
      default:  Console.WriteLine("Ni espacio ni dígito.");
                break;
    }
  }
}
```

En el lenguaje C, que es más antiguo, sólo se podía usar "switch" para comprobar valores de variables "simples" (numéricas y caracteres); en C#, que es un lenguaje más evolucionado, se puede usar también para comprobar valores de cadenas de texto ("strings").

Una cadena de texto, como veremos con más detalle en el próximo tema, se declara con la palabra "**string**", se puede leer de teclado con ReadLine (sin necesidad de convertir) y se le puede dar un valor desde programa si se indica entre comillas dobles. Por ejemplo, un programa que nos salude de forma personalizada si somos "Juan" o "Pedro" podría ser:

```
/*----------------------------*/
/*  Ejemplo en C# nº 15:      */
/*  ejemplo15.cs              */
/*                            */
/*  La orden "switch"  (2)    */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo15
{
  public static void Main()
  {
    string nombre;

    Console.WriteLine("Introduce tu nombre");
    nombre = Console.ReadLine();

    switch (nombre)
    {
      case "Juan": Console.WriteLine("Bienvenido, Juan.");
                   break;
      case "Pedro": Console.WriteLine("Que tal estas, Pedro.");
                   break;
      default:  Console.WriteLine("Procede con cautela, desconocido.");
                   break;
    }
  }
}
```

Ejercicios propuestos:

- **(2.1.9.1)** Crear un programa que lea una letra tecleada por el usuario y diga si se trata de una vocal, una cifra numérica o una consonante (pista: habrá que usar un dato de tipo "char").
- **(2.1.9.2)** Crear un programa que lea una letra tecleada por el usuario y diga si se trata de un signo de puntuación, una cifra numérica o algún otro carácter.
- **(2.1.9.3)** Repetir el ejercicio 2.1.9.1, empleando "if" en lugar de "switch".
- **(2.1.9.4)** Repetir el ejercicio 2.1.9.2, empleando "if" en lugar de "switch".

2.2. Estructuras repetitivas

Hemos visto cómo comprobar condiciones, pero no cómo hacer que una cierta parte de un programa se repita un cierto número de veces o mientras se cumpla una condición (lo que llamaremos un "**bucle**"). En C# tenemos varias formas de conseguirlo.

2.2.1. while

Si queremos hacer que una sección de nuestro programa se repita mientras se cumpla una cierta condición, usaremos la orden "while". Esta orden tiene dos formatos distintos, según comprobemos la condición al principio o al final.

En el primer caso, su sintaxis es

```
while (condición)
  sentencia;
```

Es decir, la sentencia se repetirá **mientras** la condición sea cierta. Si la condición es falsa ya desde un principio, la sentencia no se ejecuta nunca. Si queremos que se repita más de una sentencia, basta agruparlas entre { y }.

Un ejemplo que nos diga si cada número que tecleemos es positivo o negativo, y que pare cuando tecleemos el número 0, podría ser:

```
/*----------------------------*/
/*  Ejemplo en C# nº 16:      */
/*  ejemplo16.cs              */
/*                            */
/*  La orden "while"          */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo16
{
  public static void Main()
  {
    int numero;

    Console.Write("Teclea un número (0 para salir): ");
    numero = Convert.ToInt32(Console.ReadLine());
```

```
    while (numero != 0)
    {

      if (numero > 0) Console.WriteLine("Es positivo");
        else Console.WriteLine("Es negativo");

      Console.WriteLine("Teclea otro número (0 para salir): ");
      numero = Convert.ToInt32(Console.ReadLine());

    }

  }
}
```

En este ejemplo, si se introduce 0 la primera vez, la condición es falsa y ni siquiera se entra al bloque del "while", terminando el programa inmediatamente.

Ahora que sabemos "repetir" cosas, podemos utilizarlo también para **contar**. Por ejemplo, si queremos contar del 1 al 5, nuestra variable empezaría en 1, aumentaría una unidad en cada repetición y se repetiría hasta llegar al valor 5, así:

```
/*----------------------------*/
/*  Ejemplo en C# nº 16b:     */
/*  ejemplo16b.cs             */
/*                            */
/*  Contar con "while"        */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo16b
{
  public static void Main()
  {
    int n = 1;

    while (n < 6)
    {
      Console.WriteLine(n);
      n = n + 1;
    }
  }
}
```

Ejercicios propuestos:

- **(2.2.1.1)** Crear un programa que pida al usuario su contraseña (numérica). Deberá terminar cuando introduzca como contraseña el número 1111, pero volvérsela a pedir tantas veces como sea necesario.
- **(2.2.1.2)** Crea un programa que escriba en pantalla los números del 1 al 10, usando "while".

- **(2.2.1.3)** Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "while".
- **(2.2.1.4)** Crear un programa calcule cuantas cifras tiene un número entero positivo (pista: se puede hacer dividiendo varias veces entre 10).
- **(2.2.1.5)** Crear el diagrama de flujo y la versión en C# de un programa que dé al usuario tres oportunidades para adivinar un número del 1 al 10.

2.2.2. do ... while

Este es el otro formato que puede tener la orden "while": la condición se comprueba **al final** (equivale a "repetir...mientras"). El punto en que comienza a repetirse se indica con la orden "do", así:

```
do
  sentencia;
while (condición)
```

Al igual que en el caso anterior, si queremos que se repitan varias órdenes (es lo habitual), deberemos encerrarlas entre llaves.

Como ejemplo, vamos a ver cómo sería el típico programa que nos pide una clave de acceso y no nos deja entrar hasta que tecleemos la clave correcta:

```
/*----------------------------*/
/*  Ejemplo en C# nº 17:      */
/*  ejemplo17.cs              */
/*                            */
/*  La orden "do..while"      */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo17
{
  public static void Main()
  {

    int valida = 711;
    int clave;

    do
    {

      Console.Write("Introduzca su clave numérica: ");
      clave = Convert.ToInt32(Console.ReadLine());

      if (clave != valida)
        Console.WriteLine("No válida!");

    }
    while (clave != valida);
```

```
    Console.WriteLine("Aceptada.");

  }
}
```

En este caso, se comprueba la condición al final, de modo que se nos preguntará la clave al menos una vez. Mientras que la respuesta que demos no sea la correcta, se nos vuelve a preguntar. Finalmente, cuando tecleamos la clave correcta, el ordenador escribe "Aceptada" y termina el programa.

Como veremos un poco más adelante, si preferimos que la clave sea un texto en vez de un número, los cambios al programa son mínimos, basta con usar "string":

```
/*---------------------------*/
/*  Ejemplo en C# nº 18:     */
/*  ejemplo18.cs             */
/*                           */
/*  La orden "do..while" (2) */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo18
{
  public static void Main()
  {

    string valida = "secreto";
    string clave;

    do
    {
      Console.Write("Introduzca su clave: ");
      clave = Console.ReadLine();

      if (clave != valida)
          Console.WriteLine("No válida!");

    }
    while (clave != valida);

    Console.WriteLine("Aceptada.");

  }
}
```

Ejercicios propuestos:

- **(2.2.2.1)** Crear un programa que pida números positivos al usuario, y vaya calculando la suma de todos ellos (terminará cuando se teclea un número negativo o cero).
- **(2.2.2.2)** Crea un programa que escriba en pantalla los números del 1 al 10, usando "do..while".

- **(2.2.2.3)** Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "do..while".
- **(2.2.2.4)** Crea un programa que pida al usuario su identificador y su contraseña (ambos numéricos), y no le permita seguir hasta que introduzca como identificador "1234" y como contraseña "1111".
- **(2.2.2.5)** Crea un programa que pida al usuario su identificador y su contraseña, y no le permita seguir hasta que introduzca como nombre "Pedro" y como contraseña "Peter".

2.2.3. for

Ésta es la orden que usaremos habitualmente para crear partes del programa que **se repitan** un cierto número de veces. El formato de "for" es

```
for (valorInicial; CondiciónRepetición; Incremento)
  Sentencia;
```

Así, para **contar del 1 al 10**, tendríamos 1 como valor inicial, <=10 como condición de repetición, y el incremento sería de 1 en 1. Es muy habitual usar la letra "i" como contador, cuando se trata de tareas muy sencillas, así que el valor inicial sería "i=1", la condición de repetición sería "i<=10" y el incremento sería "i=i+1":

```
for (i=1; i<=10; i=i+1)
    ...
```

La orden para incrementar el valor de una variable ("i = i+1") se puede escribir de la forma abreviada "i++", como veremos con más detalle en el próximo tema.

En general, será preferible usar nombres de variable más descriptivos que "i". Así, un programa que escribiera los números del 1 al 10 podría ser:

```
/*----------------------------*/
/*  Ejemplo en C# nº 19:      */
/*  ejemplo19.cs              */
/*                            */
/*  Uso básico de "for"       */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo19
{
  public static void Main()
  {

    int contador;

    for (contador=1; contador<=10; contador++)
      Console.Write("{0} ", contador);
```

```
  }
}
```

Ejercicios propuestos:

- **(2.2.3.1)** Crear un programa que muestre los números del 15 al 5, descendiendo (pista: en cada pasada habrá que descontar 1, por ejemplo haciendo `i=i-1`, que se puede abreviar `i--`).
- **(2.2.3.2)** Crear un programa que muestre los primeros ocho números pares (pista: en cada pasada habrá que aumentar de 2 en 2, o bien mostrar el doble del valor que hace de contador).

En un "for", realmente, la parte que hemos llamado "Incremento" no tiene por qué incrementar la variable, aunque ése es su uso más habitual. Es simplemente una orden que se ejecuta cuando se termine la "Sentencia" y antes de volver a comprobar si todavía se cumple la condición de repetición.

Por eso, si escribimos la siguiente línea:

```
for (contador=1; contador<=10; )
```

la variable "contador" no se incrementa nunca, por lo que nunca se cumplirá la condición de salida: nos quedamos encerrados dando vueltas dentro de la orden que siga al "for". El programa no termina nunca. Se trata de un "bucle sin fin".

Un caso todavía más exagerado de algo a lo que se entra y de lo que no se sale sería la siguiente orden:

```
for ( ; ; )
```

Los bucles "for" se pueden **anidar** (incluir uno dentro de otro), de modo que podríamos escribir las tablas de multiplicar del 1 al 5 con:

```
/*----------------------------*/
/*  Ejemplo en C# nº 20:      */
/*  ejemplo20.cs              */
/*                            */
/*  "for" anidados            */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo20
{
  public static void Main()
  {

    int tabla, numero;

    for (tabla=1; tabla<=5; tabla++)
```

```
      for (numero=1; numero<=10; numero++)

        Console.WriteLine("{0} por {1} es {2}", tabla, numero,
          tabla*numero);

  }
}
```

En estos ejemplos que hemos visto, después de "for" había una única sentencia. Si queremos que se hagan varias cosas, basta definirlas como un **bloque** (una sentencia compuesta) encerrándolas entre llaves. Por ejemplo, si queremos mejorar el ejemplo anterior haciendo que deje una línea en blanco entre tabla y tabla, sería:

```
/*--------------------------*/
/*  Ejemplo en C# nº 21:    */
/*  ejemplo21.cs            */
/*                          */
/*  "for" anidados (2)      */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo21
{
  public static void Main()
  {

    int tabla, numero;

    for (tabla=1; tabla<=5; tabla++)
    {

      for (numero=1; numero<=10; numero++)

        Console.WriteLine("{0} por {1} es {2}", tabla, numero,
          tabla*numero);

      Console.WriteLine();
    }

  }
}
```

Para "contar" no necesariamente hay que usar **números**. Por ejemplo, podemos contar con letras así:

```
/*--------------------------*/
/*  Ejemplo en C# nº 22:    */
/*  ejemplo22.cs            */
```

```
/*                           */
/*  "for" que usa "char"     */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo22
{
  public static void Main()
  {

    char letra;

    for (letra='a'; letra<='z'; letra++)
      Console.Write("{0} ", letra);

  }
}
```

En este caso, empezamos en la "a" y terminamos en la "z", aumentando de uno en uno.

Si queremos contar de forma **decreciente**, o de dos en dos, o como nos interese, basta indicarlo en la condición de finalización del "for" y en la parte que lo incrementa:

```
/*---------------------------*/
/*  Ejemplo en C# nº 23:     */
/*  ejemplo23.cs             */
/*                           */
/*  "for" que descuenta      */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo23
{
  public static void Main()
  {

    char letra;

    for (letra='z'; letra>='a'; letra--)
      Console.Write("{0} ", letra);

  }
}
```

Ejercicios propuestos:

- **(2.2.3.3)** Crear un programa que muestre las letras de la Z (mayúscula) a la A (mayúscula, descendiendo).
- **(2.2.3.4)** Crear un programa que escriba en pantalla la tabla de multiplicar del 5.
- **(2.2.3.5)** Crear un programa que escriba en pantalla los números del 1 al 50 que sean múltiplos de 3 (pista: habrá que recorrer todos esos números y ver si el resto de la división entre 3 resulta 0).

Nota: Se puede incluso declarar una nueva variable en el interior de "for", y esa variable desaparecerá cuando el "for" acabe:

```
for (int i=1; i<=10; i++) …
```

2.3. Sentencia break: termina el bucle

Podemos salir de un bucle "for" antes de tiempo con la orden "**break**":

```
/*----------------------------*/
/*  Ejemplo en C# nº 24:      */
/*  ejemplo24.cs              */
/*                            */
/*  "for" interrumpido con    */
/*  "break"                   */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo24
{
  public static void Main()
  {

    int contador;

    for (contador=1; contador<=10; contador++)
    {
      if (contador==5)
        break;

      Console.Write("{0} ", contador);
    }

  }
}
```

El resultado de este programa es:

```
1 2 3 4
```

(en cuanto se llega al valor 5, se interrumpe el "for", por lo que no se alcanza el valor 10).

2.4. Sentencia continue: fuerza la siguiente iteración

Podemos saltar alguna repetición de un bucle con la orden "**continue**":

```
/*----------------------------*/
/*  Ejemplo en C# nº 25:      */
/*  ejemplo25.cs              */
/*                            */
/*  "for" interrumpido con    */
/*  "continue"                */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo25
{
  public static void Main()
  {

    int contador;

    for (contador=1; contador<=10; contador++)
    {
      if (contador==5)
        continue;

      Console.Write("{0} ", contador);
    }

  }
}
```

El resultado de este programa es:

```
1 2 3 4 6 7 8 9 10
```

En él podemos observar que no aparece el valor 5.

Ejercicios resueltos:

- ¿Qué escribiría en pantalla este fragmento de código?

  ```
  for (i=1; i<4; i++) Console.Write("{0} ",i);
  ```

  Respuesta: los números del 1 al 3 (se empieza en 1 y se repite mientras sea menor que 4).

- ¿Qué escribiría en pantalla este fragmento de código?

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 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 nos hemos saltado 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.

(**Nota:** si no sabes lo que es un byte, deberías mirar el Apéndice 1 de este texto).

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 datos 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 os 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".

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).

Ejercicios propuestos:

- **(3.1.2.1)** Preguntar al usuario su edad, que se guardará en un "byte". A continuación, se deberá le deberá decir que no aparenta tantos años (por ejemplo, "No aparentas 20 años").
- **(3.1.2.2)** Pedir al usuario dos números de dos cifras ("byte"), calcular su multiplicación, que se deberá guardar en un "ushort", y mostrar el resultado en pantalla.
- **(3.1.2.3)** Pedir al usuario dos números enteros largos ("long") y mostrar 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, y que no tiene un símbolo específico para representarla en matemáticas: incrementar el valor de una variable en una unidad:

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

Pues bien, en C#, 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;
```

Pero esto tiene más misterio todavía del 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:

- **(3.1.3.1)** Crear 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.
- **(3.1.3.2)** ¿Cuál sería el resultado de las siguientes operaciones? a=5; b=++a; c=a++; b=b*5; a=a*2;

Y ya que estamos hablando de las asignaciones, hay que comentar que en C# es posible hacer **asignaciones múltiples**:

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

3.1.4. Operaciones abreviadas: +=

Pero 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;
```

Ejercicios propuestos:

- **(3.1.4.1)** Crear un programa que use tres variables x,y,z. Sus valores iniciales serán 15, -10, 214. Se deberá 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.
- **(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;

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). 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 1010 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.

3.2.1. Simple y doble precisión

Tenemos tres tamaños 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") existe 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, que es el tipo "decimal":

	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

Para definirlos, se hace igual que en el caso de los números enteros:

```
float x;
```

o bien, si queremos dar un valor inicial en el momento de definirlos (recordando que para las cifras decimales no debemos usar una coma, sino un punto):

```
float x = 12.56;
```

3.2.2. Pedir y mostrar números reales

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 en C# nº 27:      */
/*  ejemplo27.cs              */
/*                            */
/*  Números reales (1)        */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo27
{
    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 parte del trabajo se le encarga al sistema operativo, de modo que si éste sabe que en nuestro país se usa la coma para los decimales, considere la coma el separador correcto y no el punto, como ocurre si introducimos estos datos en la versión española de Windows XP:

```
ejemplo05
Introduce el primer número
23,6
Introduce el segundo número
34.2
La suma de 23,6 y 342 es 365,6
```

Ejercicios propuestos:

- **(3.2.2.1)** Calcular el área de un círculo, dado su radio (pi * radio al cuadrado)
- **(3.2.2.2)** Crear 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).
- **(3.2.2.3)** Hallar 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)
- **(3.2.2.4)** 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.
- **(3.2.2.5)** Crea un programa que muestre los primeros 20 valores de la función $y = x^2 - 1$
- **(3.2.2.6)** 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".
- **(3.2.2.7)** 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.

3.2.3. Formatear números

En más de una ocasión nos interesará formatear los números para mostrar 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.

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")

Algunas 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 en C# nº 28:      */
/*  ejemplo28.cs              */
/*                            */
/*  Formato de núms. reales   */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo28
{
    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
```

Como se puede ver, ocurre lo siguiente:

- Si indicamos menos decimales de los que tiene el número, se redondea.
- Si indicamos más decimales de los que tiene el número, se mostrarán ceros si usamos como formato Nx o 0.000, y no se mostrará nada si usamos #.###
- Si indicamos menos cifras antes de la coma decimal de las que realmente tiene el número, aun así se muestran todas ellas.

Ejercicios propuestos:

- **(3.2.3.1)** 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.
- **(3.2.3.2)** Crear un programa que use tres variables x,y,z. Las tres serán números reales, y nos bastará con dos cifras decimales. 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).
- **(3.2.3.3)** Calcular 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). Mostrar todos ellos con una cifra decimal.
- **(3.2.3.4)** Calcular 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). Mostrar los resultados con 3 cifras decimales.

3.2.4. 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 en C# nº 28b:    */
/*  ejemplo28b.cs            */
/*                           */
/*  Hexadecimal y binario    */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo28b
```

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

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

Su resultado sería:

```
f7
11110111
```

(Si quieres saber más sobre el sistema hexadecimal, mira los apéndices al final de este texto)

Ejercicios propuestos:

- **(3.2.4.1)** 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.
- **(3.2.4.2)** 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)
- **(3.2.4.3)** Crea un programa para mostar 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 de hexadecimal a binario o 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 = 0x12a3;" pero un valor precedido por "0" no se considera octal sino decimal, al contrario de lo que ocurre en los lenguajes C y C++:

```
/*----------------------------*/
/*  Ejemplo en C# nº 28c:     */
/*  ejemplo28c.cs             */
/*                            */
/*  Hexadecimal y binario     */
/* (2)                        */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Example28c
{
    public static void Main()
    {
        int n1 = 0x12a3;
```

```
        int n2 = Convert.ToInt32("12a4", 16);

        int n3 = 0123; // No es octal, al contrario que en C y C++
        int n4 = Convert.ToInt32("124", 8);

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

        double d1 = 5.7;
        float f1 = 5.7f;

        Console.WriteLine( "{0} {1} {2} {3} {4}",
            n1, n2, n3, n4, n5);
        Console.WriteLine( "{0} {1}",
            d1, f1);
    }
}
```

Que mostraría:

```
4771 4772 123 84 201
5,7 5,7
```

Nota: La notación "float f1 = 5.7f;" se usa para detallar que se trata de un número real de simple precisión (un "float"), porque de lo contrario, en " float f1 = 5.7;" se consideraría un número de doble precisión, y al tratar de compilar obtendríamos un mensaje de error, diciendo que no se puede convertir de "double" a "float" sin pérdida de precisión. Al añadir la "f" al final, estamos diciendo "quiero que éste número se tome como un float; sé que habrá una pérdida de precisión pero es aceptable para mí".

Ejercicios propuestos:

- **(3.2.4.4)** 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

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 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 en C# nº 29:      */
/*  ejemplo29.cs              */
/*                            */
/*  Tipo de datos "char"      */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo29
{
    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);
    }
}
```

Ejercicio propuesto

- **(3.3.1.1)** Crear un programa que pida una letra al usuario y diga si se trata de una vocal.

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, y con \' unas comillas simples, o con \n se avanzará a la línea siguiente de pantalla.

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 en C# nº 30:      */
/*  ejemplo30.cs              */
/*                            */
/*  Secuencias de escape      */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo30
{
    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 \\");
    }
}
```

Su resultado sería este:

```
Esta es una frase


y esta es otra, separada dos lineas


Juguemos mas:

otro salto
```

```
Comillas dobles: " y simples ', y barra \
```

En algunas ocasiones puede ser incómodo manipular estas secuencias de escape. Por ejemplo, cuando usemos estructuras de directorios: 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:"
```

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

Las cadenas de texto son tan fáciles de manejar como los demás tipos de datos que hemos visto, con apenas tres diferencias:

- 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 "==" o "!=".

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 en C# nº 31:    */
/*  ejemplo31.cs            */
/*                          */
/*  Uso basico de "string"  */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo31
{
    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:

- **(3.4.1)** Crear 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.
- **(3.4.2)** Crear 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:

```
partidaTerminada = false;
partidaTerminada = (enemigos ==0) && (nivel == ultimoNivel);
if (vidas == 0) partidaTerminada = true;
```

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

```
/*--------------------------*/
/*  Ejemplo en C# nº 32:    */
/*  ejemplo32.cs            */
```

```
/*                          */
/*  Condiciones con if (8)  */
/*    Variables bool        */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo32
{
  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:

- **(3.5.1)** Crear 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.
- **(3.5.2)** Crea una versión alternativa del ejercicio 3.5.1, que use "if" en vez del operador condicional.
- **(3.5.3)** Crear 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.
- **(3.5.4)** Crea una versión alternativa del ejercicio 3.5.3, que use "if" en vez del operador condicional.

5. Introducción a las funciones

5.1. Diseño modular de programas: Descomposición modular

Hasta ahora hemos estado pensando los pasos que deberíamos dar para resolver un cierto problema, y hemos creado programas a partir de cada uno de esos pasos. Esto es razonable cuando los problemas son sencillos, pero puede no ser la mejor forma de actuar cuando se trata de algo más complicado.

A partir de ahora vamos a empezar a intentar descomponer los problemas en trozos más pequeños, que sean más fáciles de resolver. Esto nos puede suponer varias ventajas:

- Cada "trozo de programa" independiente será más fácil de programar, al realizar una función breve y concreta.
- El "programa principal" será más fácil de leer, porque no necesitará contener todos los detalles de cómo se hace cada cosa.
- Podremos repartir el trabajo, para que cada persona se encargue de realizar un "trozo de programa", y finalmente se integrará el trabajo individual de cada persona.

Esos "trozos" de programa son lo que suele llamar "subrutinas", "procedimientos" o "funciones". En el lenguaje C y sus derivados, el nombre que más se usa es el de **funciones**.

5.2. Conceptos básicos sobre funciones

En C#, al igual que en C y los demás lenguajes derivados de él, todos los "trozos de programa" son funciones, incluyendo el propio cuerpo de programa, **Main**. De hecho, la forma básica de **definir** una función será indicando su nombre seguido de unos paréntesis vacíos, como hacíamos con "Main", y precediéndolo por ciertas palabras reservadas, como "public static **void**", cuyo significado iremos viendo muy pronto. Después, entre llaves indicaremos todos los pasos que queremos que dé ese "trozo de programa".

Por ejemplo, podríamos crear una función llamada "saludar", que escribiera varios mensajes en la pantalla:

```
public static void saludar()
{
  Console.WriteLine("Bienvenido al programa");
  Console.WriteLine(" de ejemplo");
  Console.WriteLine("Espero que estés bien");
}
```

Ahora desde dentro del cuerpo de nuestro programa, podríamos "**llamar**" a esa función:

```
public static void Main()
{
  saludar();
  …
}
```

Así conseguimos que nuestro programa principal sea más fácil de leer.

Un detalle importante: tanto la función habitual "Main" como la nueva función "Saludar" serían parte de nuestra "class", es decir, el fuente completo sería así:

```
/*---------------------------*/
/*  Ejemplo en C# nº 47:     */
/*  ejemplo47.cs             */
/*                           */
/*  Funcion "saludar"        */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo47
{

  public static void Saludar()
  {
    Console.WriteLine("Bienvenido al programa");
    Console.WriteLine(" de ejemplo");
    Console.WriteLine("Espero que estés bien");
  }

  public static void Main()
  {
    Saludar();
    Console.WriteLine("Nada más por hoy...");
  }

}
```

Como ejemplo más detallado, la parte principal de una agenda o de una base de datos simple como las que hicimos en el tema anterior, podría ser simplemente:

```
leerDatosDeFichero();
do {
  mostrarMenu();
  pedirOpcion();
  switch( opcion ) {
    case 1: buscarDatos(); break;
    case 2: modificarDatos(); break;
    case 3: anadirDatos(); break;
    …
```

Ejercicios propuestos:

- **(5.2.1)** Crea una función llamada "BorrarPantalla", que borre la pantalla dibujando 25 líneas en blanco. No debe devolver ningún valor. Crea también un "Main" que permita probarla.
- **(5.2.2)** Crea una función llamada "DibujarCuadrado3x3", que dibuje un cuadrado formato por 3 filas con 3 asteriscos cada una. Crea también un "Main" que permita probarla.

- **(5.2.3)** Descompón en funciones la base de datos de ficheros (ejemplo 46), de modo que el "Main" sea breve y más legible (Pista: las variables que se compartan entre varias funciones deberán estar fuera de todas ellas, y deberán estar precedidas por la palabra "static").

5.3. Parámetros de una función

Es muy frecuente que nos interese además indicarle a nuestra función ciertos datos especiales con los que queremos que trabaje. Por ejemplo, si escribimos en pantalla números reales con frecuencia, nos puede resultar útil crear una función auxiliar que nos los muestre con el formato que nos interese. Lo podríamos hacer así:

```
public static void escribeNumeroReal( float n )
{
  Console.WriteLine( n.ToString("#.###") );
}
```

Y esta función se podría usar desde el cuerpo de nuestro programa así:

```
escribeNumeroReal(2.3f);
```

(recordemos que el sufijo "f" es para indicar al compilador que trate ese número como un "float", porque de lo contrario, al ver que tiene cifras decimales, lo tomaría como "double", que permite mayor precisión... pero a cambio nosotros tendríamos un mensaje de error en nuestro programa, diciendo que estamos dando un dato "double" a una función que espera un "float").

El programa completo podría quedar así:

```
/*---------------------------*/
/*  Ejemplo en C# nº 48:     */
/*  ejemplo48.cs             */
/*                           */
/*  Funcion                  */
/*    "escribeNumeroReal"    */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo48
{

  public static void escribeNumeroReal( float n )
  {
    Console.WriteLine( n.ToString("#.###") );
  }

  public static void Main()
  {
    float x;
```

```
    x= 5.1f;
    Console.WriteLine("El primer numero real es: ");
    escribeNumeroReal(x);
    Console.WriteLine(" y otro distinto es: ");
    escribeNumeroReal(2.3f);
  }

}
```

Estos datos adicionales que indicamos a la función es lo que llamaremos sus "parámetros". Como se ve en el ejemplo, tenemos que indicar un nombre para cada parámetro (puede haber varios) y el tipo de datos que corresponde a ese parámetro. Si hay más de un parámetro, deberemos indicar el tipo y el nombre para cada uno de ellos, y separarlos entre comas:

```
public static void escribirSuma ( int x, int y ) {
  ...
}
```

Ejercicios propuestos:

- **(5.3.1)** Crea una función que dibuje en pantalla un cuadrado del ancho (y alto) que se indique como parámetro. Completa el programa con un Main que permita probarla.
- **(5.3.2)** Crea una función que dibuje en pantalla un rectángulo del ancho y alto que se indiquen como parámetros. Completa el programa con un Main que permita probarla.
- **(5.3.3)** Crea una función que dibuje en pantalla un rectángulo hueco del ancho y alto que se indiquen como parámetros, formado por una letra que también se indique como parámetro. Completa el programa con un Main que pida esos datos al usuario y dibuje el rectángulo.

5.4. Valor devuelto por una función. El valor "void".

Cuando queremos dejar claro que una función no tiene que devolver ningún valor, podemos hacerlo indicando al principio que el tipo de datos va a ser "void" (nulo), como hacíamos hasta ahora con "Main" y como hicimos con nuestra función "saludar".

Pero eso no es lo que ocurre con las funciones matemáticas que estamos acostumbrados a manejar: sí devuelven un valor, el resultado de una operación.

De igual modo, para nosotros también será habitual que queramos que nuestra función realice una serie de cálculos y nos "devuelva" (**return**, en inglés) el resultado de esos cálculos, para poderlo usar desde cualquier otra parte de nuestro programa. Por ejemplo, podríamos crear una función para elevar un número entero al cuadrado así:

```
public static int cuadrado ( int n )
{
  return n*n;
}
```

y podríamos usar el resultado de esa función como si se tratara de un número o de una variable, así:

```
resultado = cuadrado( 5 );
```

Un programa más detallado de ejemplo podría ser:

```
/*--------------------------*/
/*  Ejemplo en C# nº 49:    */
/*  ejemplo49.cs            */
/*                          */
/*  Funcion "cuadrado"      */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo49
{

  public static int cuadrado ( int n )
  {
    return n*n;
  }


  public static void Main()
  {
    int numero;
    int resultado;

    numero= 5;
    resultado = cuadrado(numero);
    Console.WriteLine("El cuadrado del numero {0} es {1}",
      numero, resultado);
    Console.WriteLine(" y el de 3 es {0}", cuadrado(3));
  }

}
```

Podemos hacer una función que nos diga cuál es el mayor de dos números reales así:

```
  public static float mayor ( float n1, float n2 )
  {
    if (n1 > n2)
      return n1;
    else
      return n2;
  }
```

Ejercicios propuestos:

- **(5.4.1)** Crear una función que calcule el cubo de un número real (float) que se indique como parámetro. El resultado deberá ser otro número real. Probar esta función para calcular el cubo de 3.2 y el de 5.
- **(5.4.2)** Crear una función que calcule el menor de dos números enteros que recibirá como parámetros. El resultado será otro número entero.

- **(5.4.3)** Crear una función llamada "signo", que reciba un número real, y devuelva un número entero con el valor: -1 si el número es negativo, 1 si es positivo o 0 si es cero.
- **(5.4.4)** Crear una función que devuelva la primera letra de una cadena de texto. Probar esta función para calcular la primera letra de la frase "Hola".
- **(5.4.5)** Crear una función que devuelva la última letra de una cadena de texto. Probar esta función para calcular la última letra de la frase "Hola".
- **(5.4.6)** Crear una función que reciba un número y calcule y muestre en pantalla el valor del perímetro y de la superficie de un cuadrado que tenga como lado el número que se ha indicado como parámetro.

5.5. Variables locales y variables globales

Hasta ahora, hemos declarado las variables dentro de "Main". Ahora nuestros programas tienen varios "bloques", así que se comportarán de forma distinta según donde declaremos las variables.

Las variables se pueden declarar dentro de un bloque (una función), y entonces sólo ese bloque las conocerá, no se podrán usar desde ningún otro bloque del programa. Es lo que llamaremos "variables **locales**".

Por el contrario, si declaramos una variable al comienzo del programa, fuera de todos los "bloques" de programa, será una "**variable global**", a la que se podrá acceder desde cualquier parte.

Vamos a verlo con un ejemplo. Crearemos una función que calcule la potencia de un número entero (un número elevado a otro), y el cuerpo del programa que la use.

La forma de conseguir elevar un número a otro será a base de multiplicaciones, es decir:

```
3 elevado a 5   =   3 · 3 · 3 · 3 · 3
```

(multiplicamos 5 veces el 3 por sí mismo). En general, como nos pueden pedir cosas como "6 elevado a 100" (o en general números que pueden ser grandes), usaremos la orden "for" para multiplicar tantas veces como haga falta:

```
/*---------------------------*/
/*  Ejemplo en C# nº 50:     */
/*  ejemplo50.cs             */
/*                           */
/*  Ejemplo de función con   */
/*  variables locales        */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo50
{
```

```
  public static int potencia(int nBase, int nExponente)
  {
    int temporal = 1;         /* Valor que voy hallando */
    int i;                    /* Para bucles */

    for(i=1; i<=nExponente; i++)    /* Multiplico "n" veces */
      temporal *= nBase;            /* Y calculo el valor temporal */

    return temporal;               /* Tras las multiplicaciones, */
  }                                /*    obtengo el valor que buscaba */


  public static void Main()
  {
    int num1, num2;

    Console.WriteLine("Introduzca la base: ");
    num1 = Convert.ToInt32( Console.ReadLine() );

    Console.WriteLine("Introduzca el exponente: ");
    num2 = Convert.ToInt32( Console.ReadLine() );

    Console.WriteLine("{0} elevado a {1} vale {2}",
      num1, num2, potencia(num1,num2));
  }

}
```

En este caso, las variables "temporal" e "i" son locales a la función "potencia": para "Main" no existen. Si en "Main" intentáramos hacer *i=5;* obtendríamos un mensaje de error.

De igual modo, "num1" y "num2" son locales para "main": desde la función "potencia" no podemos acceder a su valor (ni para leerlo ni para modificarlo), sólo desde "main".

En general, deberemos intentar que la mayor cantidad de variables posible sean locales (lo ideal sería que todas lo fueran). Así hacemos que cada parte del programa trabaje con sus propios datos, y ayudamos a evitar que un error en un trozo de programa pueda afectar al resto. La forma correcta de pasar datos entre distintos trozos de programa es usando los parámetros de cada función, como en el anterior ejemplo.

Ejercicios propuestos:

- **(5.5.1)** Crear una función "pedirEntero", que reciba como parámetros el texto que se debe mostrar en pantalla, el valor mínimo aceptable y el valor máximo aceptable. Deberá pedir al usuario que introduzca el valor tantas veces como sea necesario, volvérselo a pedir en caso de error, y devolver un valor correcto. Probarlo con un programa que pida al usuario un año entre 1800 y 2100.
- **(5.5.2)** Crear una función "escribirTablaMultiplicar", que reciba como parámetro un número entero, y escriba la tabla de multiplicar de ese número (por ejemplo, para el 3 deberá llegar desde "3x0=0" hasta "3x10=30").
- **(5.5.3)** Crear una función "esPrimo", que reciba un número y devuelva el valor booleano "true" si es un número primo o "false" en caso contrario.

- **(5.5.4)** Crear una función que reciba una cadena y una letra, y devuelva la cantidad de veces que dicha letra aparece en la cadena. Por ejemplo, si la cadena es "Barcelona" y la letra es 'a', debería devolver 2 (porque la "a" aparece 2 veces).
- **(5.5.5)** Crear una función que reciba un numero cualquiera y que devuelva como resultado la suma de sus dígitos. Por ejemplo, si el número fuera 123 la suma sería 6.
- **(5.5.6)** Crear una función que reciba una letra y un número, y escriba un "triángulo" formado por esa letra, que tenga como anchura inicial la que se ha indicado. Por ejemplo, si la letra es * y la anchura es 4, debería escribir

  ****
  ***
  **
  *

5.6. Los conflictos de nombres en las variables

¿Qué ocurre si damos el mismo nombre a dos variables locales? Vamos a comprobarlo con un ejemplo:

```
/*---------------------------*/
/*  Ejemplo en C# nº 51:     */
/*  ejemplo51.cs             */
/*                           */
/*  Dos variables locales    */
/*  con el mismo nombre      */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo51
{

  public static void cambiaN() {
    int n = 7;
    n ++;
  }


  public static void Main()
  {
    int n = 5;
    Console.WriteLine("n vale {0}", n);
    cambiaN();
    Console.WriteLine("Ahora n vale {0}", n);
  }

}
```

El resultado de este programa es:

```
n vale 5
```

```
Ahora n vale 5
```

¿Por qué? Sencillo: tenemos una variable local dentro de "cambiaN" y otra dentro de "main". El hecho de que las dos tengan el mismo nombre no afecta al funcionamiento del programa, siguen siendo distintas.

Si la variable es "global", declarada fuera de estas funciones, sí será accesible por todas ellas:

```
/*----------------------------*/
/*  Ejemplo en C# nº 52:      */
/*  ejemplo52.cs              */
/*                            */
/*  Una variable global       */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo52
{

  static int n = 7;

  public static void cambiaN() {
    n ++;
  }


  public static void Main()
  {
    Console.WriteLine("n vale {0}", n);
    cambiaN();
    Console.WriteLine("Ahora n vale {0}", n);
  }

}
```

Dentro de poco, hablaremos de por qué cada uno de los bloques de nuestro programa, e incluso las "variables globales", tienen delante la palabra "static". Será cuando tratemos la "Programación Orientada a Objetos", en el próximo tema.

5.7. Modificando parámetros

Podemos modificar el valor de un dato que recibamos como parámetro, pero posiblemente el resultado no será el que esperamos. Vamos a verlo con un ejemplo:

```
/*----------------------------*/
/*  Ejemplo en C# nº 53:      */
/*  ejemplo53.cs              */
/*                            */
/*  Modificar una variable    */
/*  recibida como parámetro   */
/*                            */
```

```
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo53
{

  public static void duplica(int x) {
    Console.WriteLine("  El valor recibido vale {0}", x);
    x = x * 2;
    Console.WriteLine("  y ahora vale {0}", x);
  }

  public static void Main()
  {
    int n = 5;
    Console.WriteLine("n vale {0}", n);
    duplica(n);
    Console.WriteLine("Ahora n vale {0}", n);
  }

}
```

El resultado de este programa será:

```
n vale 5
  El valor recibido vale 5
  y ahora vale 10
Ahora n vale 5
```

Vemos que al salir de la función, los cambios que hagamos a esa variable que se ha recibido como parámetro no se conservan.

Esto se debe a que, si no indicamos otra cosa, los parámetros "**se pasan por valor**", es decir, la función no recibe los datos originales, sino una copia de ellos. Si modificamos algo, estamos cambiando una copia de los datos originales, no dichos datos.

Si queremos que los cambios se conserven, basta con hacer un pequeño cambio: indicar que la variable se va a pasar "**por referencia**", lo que se indica usando la palabra "ref", tanto en la declaración de la función como en la llamada, así:

```
/*---------------------------*/
/*  Ejemplo en C# nº 54:     */
/*  ejemplo54.cs             */
/*                           */
/*  Modificar una variable   */
/*  recibida como parámetro  */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/
```

```
using System;

public class Ejemplo54
{

  public static void duplica(ref int x) {
    Console.WriteLine("  El valor recibido vale {0}", x);
    x = x * 2;
    Console.WriteLine("  y ahora vale {0}", x);
  }

  public static void Main()
  {
    int n = 5;
    Console.WriteLine("n vale {0}", n);
    duplica(ref n);
    Console.WriteLine("Ahora n vale {0}", n);
  }

}
```

En este caso sí se modifica la variable n:

```
n vale 5
  El valor recibido vale 5
  y ahora vale 10
Ahora n vale 10
```

El hecho de poder modificar valores que se reciban como parámetros abre una posibilidad que no se podría conseguir de otra forma: con "return" sólo se puede devolver un valor de una función, pero con parámetros pasados por referencia podríamos **devolver más de un dato**. Por ejemplo, podríamos crear una función que intercambiara los valores de dos variables:

```
public static void intercambia(ref int x, ref int y)
```

La posibilidad de pasar parámetros por valor y por referencia existe en la mayoría de lenguajes de programación. En el caso de C# existe alguna posibilidad adicional que no existe en otros lenguajes, como los "parámetros de salida". Las veremos más adelante.

Ejercicios propuestos:

- **(5.7.1)** Crear una función "intercambia", que intercambie el valor de los dos números enteros que se le indiquen como parámetro.
- **(5.7.2)** Crear una función "iniciales", que reciba una cadena como "Nacho Cabanes" y devuelva las letras N y C (primera letra, y letra situada tras el primer espacio), usando parámetros por referencia.

5.8. El orden no importa

En algunos lenguajes, una función debe estar declarada antes de usarse. Esto no es necesario en C#. Por ejemplo, podríamos rescribir el fuente anterior, de modo que "Main" aparezca en primer lugar y "duplica" aparezca después, y seguiría compilando y funcionando igual:

```
/*--------------------------*/
/*  Ejemplo en C# nº 55:    */
/*  ejemplo55.cs            */
/*                          */
/*  Función tras Main       */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo55
{

  public static void Main()
  {
    int n = 5;
    Console.WriteLine("n vale {0}", n);
    duplica(ref n);
    Console.WriteLine("Ahora n vale {0}", n);
  }

  public static void duplica(ref int x) {
    Console.WriteLine("  El valor recibido vale {0}", x);
    x = x * 2;
    Console.WriteLine("  y ahora vale {0}", x);
  }

}
```

5.9. Algunas funciones útiles

5.9.1. Números aleatorios

En un programa de gestión o una utilidad que nos ayuda a administrar un sistema, no es habitual que podamos permitir que las cosas ocurran al azar. Pero los juegos se encuentran muchas veces entre los ejercicios de programación más completos, y para un juego sí suele ser conveniente que haya algo de azar, para que una partida no sea exactamente igual a la anterior.

Generar números al azar ("números aleatorios") usando C# no es difícil: debemos crear un objeto de tipo "Random", y luego llamaremos a "Next" para obtener valores entre dos extremos:

```
// Creamos un objeto random
Random generador = new Random();

// Generamos un número entre dos valores dados
```

```
// (el segundo límite no está incluido)
int aleatorio = generador.Next(1, 101);
```

También, una forma simple de obtener un número "casi al azar" entre 0 y 999 es tomar las milésimas de segundo de la hora actual:

```
int falsoAleatorio = DateTime.Now.Millisecond;
```

Pero esta forma simplificada no sirve si necesitamos obtener dos números aleatorios a la vez, porque los dos se obtendrían en el mismo milisegundo y tendrían el mismo valor; en ese caso, usaríamos "Random" y llamaríamos dos veces a "Next".

Vamos a ver un ejemplo, que muestre en pantalla dos números al azar entre 1 y 10:

```
/*---------------------------*/
/*  Ejemplo en C# nº 56:     */
/*  ejemplo56.cs             */
/*                           */
/*  Números al azar          */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo56
{

  public static void Main()
  {

    Random r = new Random();
    int aleatorio = r.Next(1, 11);
    Console.WriteLine("Un número entre 1 y 10: {0}", aleatorio);
    int aleatorio2 = r.Next(1, 11);
    Console.WriteLine("Otro: {0}", aleatorio2);

  }

}
```

Ejercicios propuestos:

- **(5.9.1.1)** Crear un programa que genere un número al azar entre 1 y 100. El usuario tendrá 6 oportunidades para acertarlo.
- **(5.9.1.2)** Mejorar el programa del ahorcado (4.4.8.3), para que la palabra a adivinar no sea tecleado por un segundo usuario, sino que se escoja al azar de un "array" de palabras prefijadas (por ejemplo, nombres de ciudades).
- **(5.9.1.3)** Crea un programa que "dibuje" asteriscos en 100 posiciones al azar de la pantalla . Para ayudarte para escribir en cualquier coordenada, puedes usar un array de dos dimensiones (con tamaños 24 para el alto y 79 para el ancho), que primero rellenes y luego dibujes en pantalla.

5.9.2. Funciones matemáticas

En C# tenemos muchas funciones matemáticas predefinidas, como:

- Abs(x): Valor absoluto
- Acos(x): Arco coseno
- Asin(x): Arco seno
- Atan(x): Arco tangente
- Atan2(y,x): Arco tangente de y/x (por si x o y son 0)
- Ceiling(x): El valor entero superior a x y más cercano a él
- Cos(x): Coseno
- Cosh(x): Coseno hiperbólico
- Exp(x): Exponencial de x (e elevado a x)
- Floor(x): El mayor valor entero que es menor que x
- Log(x): Logaritmo natural (o neperiano, en base "e")
- Log10(x): Logaritmo en base 10
- Pow(x,y): x elevado a y
- Round(x, cifras): Redondea un número
- Sin(x): Seno
- Sinh(x): Seno hiperbólico
- Sqrt(x): Raíz cuadrada
- Tan(x): Tangente
- Tanh(x): Tangente hiperbólica

(casi todos ellos usan parámetros X e Y de tipo "double")

y una serie de constantes como

E, el número "e", con un valor de 2.71828...
PI, el número "Pi", 3.14159...

Todas ellas se usan precedidas por "Math."

La mayoría de ellas son específicas para ciertos problemas matemáticos, especialmente si interviene la trigonometría o si hay que usar logaritmos o exponenciales. Pero vamos a destacar las que sí pueden resultar útiles en situaciones más variadas:

La raíz cuadrada de 4 se calcularía haciendo `x = Math.Sqrt(4);`
La potencia: para elevar 2 al cubo haríamos `y = Math.Pow(2, 3);`
El valor absoluto: para trabajar sólo con números positivos usaríamos `n = Math.Abs(x);`

Ejercicios propuestos:

- **(5.9.2.1)** Crea un programa que halle cualquier raíz de un número. El usuario deberá indicar el número (por ejemplo, 2) y el índice de la raíz (por ejemplo, 3 para la raíz cúbica). Pista: hallar la raíz cúbica de 2 es lo mismo que elevar 2 a 1/3.
- **(5.9.2.2)** Haz un programa que resuelva ecuaciones de segundo grado, del tipo $ax^2 + bx + c = 0$. El usuario deberá introducir los valores de a, b y c. Se deberá crear una función "raicesSegundoGrado", que recibirá como parámetros los coeficientes a, b y c, así como las soluciones x1 y x2 (por referencia). Deberá devolver los valores de las dos soluciones x1 y x2. Si alguna solución no existe, se devolverá como valor 100.000 para esa solución. Pista: la solución se calcula con

13. Otras características avanzadas de C#

13.1. Espacios de nombres

Desde nuestros primeros programas hemos estado usando cosas como "System.Console" o bien "using System". Esa palabra "System" indica que las funciones que estamos usando pertenecen a la estructura básica de C# y de la plataforma .Net.

La idea detrás de ese "using" es que puede ocurrir que distintos programadores en distintos puntos del mundo creen funciones o clases que se llamen igual, y, si se mezclan fuentes de distintas procedencias, esto podría dar lugar a programas que no compilaran correctamente, o, peor aún, que compilaran pero no funcionaran de la forma esperada.

Por eso, se recomienda usar "espacios de nombres", que permitan distinguir unos de otros. Por ejemplo, si yo quisiera crear mi propia clase "Console" para el acceso a la consola, o mi propia clase "Random" para manejo de números aleatorios, lo razonable es crear un nuevo espacio de nombres.

De hecho, con entornos como SharpDevelop o Visual Studio, cuando creamos un nuevo proyecto, el fuente "casi vacío" que se nos propone contendrá un espacio de nombres que se llamará igual que el proyecto. Esta es apariencia del fuente si usamos VisualStudio 2008:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Y esta es apariencia del fuente si usamos SharpDevelop 3:

```
using System;

namespace PruebaDeNamespaces
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

            // TODO: Implement Functionality Here

            Console.Write("Press any key to continue . . . ");
```

```
            Console.ReadKey(true);
        }
    }
}
```

Vamos a un ejemplo algo más avanzado, que contenga un espacio de nombres, que cree una nueva clase Console y que utilice las dos (la nuestra y la original, de System):

```
/*---------------------------*/
/*  Ejemplo en C#            */
/*  namespaces.cs            */
/*                           */
/*  Ejemplo de espacios de   */
/*    nombres                */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

namespace ConsolaDeNacho {
  public class Console
  {
     public static void WriteLine(string texto)
    {
      System.Console.ForegroundColor = ConsoleColor.Blue;
      System.Console.WriteLine("Mensaje: "+texto);
    }
  }
}

public class PruebaDeNamespaces
{

  public static void Main()
  {
    System.Console.WriteLine("Hola");
    ConsolaDeNacho.Console.WriteLine("Hola otra vez");
  }
}
```

Como se puede ver, este ejemplo tiene dos clases Console, y ambas tienen un método WriteLine. Una es la original de C#, que invocaríamos con "System.Console". Otra es la que hemos creado para el ejemplo, que escribe un texto modifica y en color (ayudándose de System.Console), y que llamaríamos mediante "ConsolaDeNacho.Console". El resultado es que podemos tener dos clases Console accesibles desde el mismo programa, sin que existan conflictos entre ellas. El resultado del programa sería:

```
Hola
Mensaje: Hola otra vez
```