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.
- Evitaremos mucho código repetitivo.
- 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 (aunque para proyectos realmente grandes, pronto veremos una alternativa que hace que el reparto y la integración sean más sencillos).

Esos "trozos" de programa son lo que suele llamar "subrutinas", "procedimientos" o "funciones". En el lenguaje C y sus derivados (entre los que está C#), 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.Write("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_05_02a.cs
// Funcion "Saludar"
// Introducción a C#

using System;

public class Ejemplo_05_02a
{

    public static void Saludar()
    {
        Console.Write("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();
    opcion = PedirOpcion();
    switch( opcion ) {
        case 1: BuscarDatos(); break;
        case 2: ModificarDatos(); break;
        case 3: AnadirDatos(); break;
        ...
```

Ejercicios propuestos:

Ejercicio propuesto 5.2.1: *Crea una función llamada "BorrarPantalla", que borre la pantalla dibujando 25 líneas en blanco. Crea también un "Main" que permita probarla.*

Ejercicio propuesto 5.2.2: *Crea una función llamada "DibujarCuadrado3x3", que dibuje un cuadrado formato por 3 filas con 3 asteriscos cada una. Incluye un "Main" para probarla.*

Ejercicio propuesto 5.2.3: *Descompón en funciones la base de datos de ficheros (ejemplo 04_06a), 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 indicarle a nuestra función ciertos datos con los que queremos que trabaje. Los llamaremos "parámetros" y los indicaremos dentro del paréntesis que sigue al nombre de la función, separados por comas. Para cada uno de ellos, deberemos indicar su tipo de datos (por ejemplo "int") y luego su nombre.

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 (que podría ser con exactamente 3 decimales). Lo podríamos hacer así:

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

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

```
  EscribirNumeroReal(2.3f);
```

(recordemos que el sufijo "f" sirve 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 pasando un dato "double" a una función que espera un "float").

El programa completo podría quedar así:

```
// Ejemplo_05_03a.cs
// Funcion "EscribirNumeroReal"

using System;

public class Ejemplo_05_03a
{

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

    public static void Main()
    {
        float x;

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

}
```

Como ya hemos anticipado, si hay más de un parámetro, deberemos indicar el tipo y el nombre para cada uno de ellos (incluso si todos son del mismo tipo), y separarlos por comas:

```
public static void EscribirSuma( int a, int b )
{
    ...
}
```

De modo que un programa completo de ejemplo para una función con dos parámetros podría ser:

```
// Ejemplo_05_03b.cs
// Funcion "EscribirSuma"

using System;

public class Ejemplo_05_03b
{

    public static void EscribirSuma( int a, int b )
    {
        Console.Write( a+b );
    }

    public static void Main()
    {
        Console.Write("La suma de 4 y 7 es: ");
        EscribirSuma(4, 7);
    }

}
```

Como se ve en estos ejemplos, se suelen seguir un par de convenios:

- Ya que las funciones expresan acciones, en general su nombre será un verbo.
- En C# se recomienda que los elementos públicos se escriban comenzando por una letra mayúscula (y recordemos que, hasta que conozcamos las alternativas y el motivo para usarlas, nuestras funciones comienzan con la palabra "public"). Este criterio depende del lenguaje. Por ejemplo, en lenguaje Java es habitual seguir el convenio de que los nombres de las funciones deban comenzar con una letra minúscula.

Ejercicios propuestos:

Ejercicio propuesto 5.3.1: *Crea una función "DibujarCuadrado" que dibuje en pantalla un cuadrado de asteriscos (*) del ancho (y alto) que se indique como parámetro. Completa el programa con un Main que permita probarla.*

Ejercicio propuesto 5.3.2: *Crea una función "DibujarRectangulo" que dibuje en pantalla un rectángulo de asteriscos (*) del ancho y alto que se indiquen como parámetros. Incluye un Main para probarla.*

Ejercicio propuesto 5.3.3: *Crea una función "DibujarRectanguloHueco" 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, que es el resultado de una operación (por ejemplo, la raíz cuadrada de un número tiene como resultado otro número).

De igual modo, para nosotros también será habitual crear funciones que realicen una serie de cálculos y nos "devuelvan" (**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;
}
```

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

```
resultado = Cuadrado( 5 );
```

En general, en las operaciones matemáticas, no será necesario que el nombre de la función sea un verbo. El programa debería ser suficientemente legible si el nombre expresa qué operación se va a realizar en la función.

Un programa más detallado de ejemplo podría ser:

```
// Ejemplo_05_04a.cs
// Funcion "Cuadrado"

using System;

public class Ejemplo_05_04a
{

    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 número {0} es {1}",
            numero, resultado);
        Console.WriteLine(" y el de 3 es {0}", Cuadrado(3));
    }
}
```

Podremos devolver cualquier tipo de datos, no sólo números enteros. Como segundo ejemplo, 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;
}
```

Como se ve en este ejemplo, una función puede tener más de un "return".

En cuanto se alcance un "return", se sale de la función por completo. Eso puede hacer que una función mal diseñada haga que el compilador nos dé un aviso de "código inalcanzable", como en el siguiente ejemplo:

```
public static string Inalcanzable()
{
    return "Aquí sí llegamos";

    string ejemplo = "Aquí no llegamos";
    return ejemplo;
}
```

En ese caso, el primer ***return*** hace que no se llegue pasar por las dos últimas líneas en ningún momento.

Ejercicios propuestos:

Ejercicio propuesto 5.4.1: *Crea una función "Cubo" que calcule el cubo de un número real (float) que se indique como parámetro. El resultado deberá ser otro número real. Prueba esta función para calcular el cubo de 3.2 y el de 5.*

Ejercicio propuesto 5.4.2: *Crea una función "Menor" que calcule el menor de dos números enteros que recibirá como parámetros. El resultado será otro número entero.*

Ejercicio propuesto 5.4.3: *Crea 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.*

Ejercicio propuesto 5.4.4: *Crea una función "Inicial", que devuelva la primera letra de una cadena de texto. Prueba esta función para calcular la primera letra de la frase "Hola".*

Ejercicio propuesto 5.4.5: *Crea una función "UltimaLetra", que devuelva la última letra de una cadena de texto. Prueba esta función para calcular la última letra de la frase "Hola".*

Ejercicio propuesto 5.4.6: *Crea una función "MostrarPerimSuperfCuadrado" que reciba un número entero 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 las variables se comportarán de forma distinta según donde las declaremos.

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. Por ahora, para nosotros, una variable global deberá llevar siempre la palabra "**static**" (dentro de poco veremos el motivo real y cuándo no será necesario).

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 no es a través de variables globales, sino usando los parámetros de cada función y los valores devueltos, como en el anterior ejemplo. Aun así, esta restricción es menos grave en lenguajes modernos, como C#, que en otros lenguajes más antiguos, como C, porque, como veremos en el próximo tema, el hecho de descomponer un programa en varias clases minimiza los efectos negativos de esas variables que se comparten entre varias funciones, además de que muchas veces tendremos datos compartidos, que no serán realmente "variables globales" sino datos específicos del problema, que llamaremos "atributos".

Vamos a ver el uso de variables locales 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_05_05a.cs
// Ejemplo de función con variables locales

using System;
public class Ejemplo_05_05a
{

    public static int Potencia(int nBase, int nExponente)
    {
        int temporal = 1;         // Valor inicial que voy incrementando

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

        return temporal; // Al final, 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". Este ejemplo no contiene ninguna variable global.

(Nota: el parámetro no se llama "base" sino "nBase" porque la palabra "base" es una palabra reservada en C#, que no podremos usar como nombre de variable; más adelante veremos para qué se usa "base").

Ejercicios propuestos:

Ejercicio propuesto 5.5.1: *Crea 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. Pruébalo con un programa que pida al usuario un año entre 1800 y 2100.*

Ejercicio propuesto 5.5.2: *Crea 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").*

Ejercicio propuesto 5.5.3: *Crea una función "EsPrimo", que reciba un número y devuelva el valor booleano "true" si es un número primo o "false" en caso contrario.*

Ejercicio propuesto 5.5.4: *Crea una función "ContarLetra", 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).*

Ejercicio propuesto 5.5.5: *Crea una función "SumaCifras" que reciba un número cualquiera y que devuelva como resultado la suma de sus dígitos. Por ejemplo, si el número fuera 123 la suma sería 6.*

Ejercicio propuesto 5.5.6: *Crea una función "Triángulo" 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 distintas? Vamos a comprobarlo con un ejemplo:

```
// Ejemplo_05_06a.cs
// Dos variables locales con el mismo nombre

using System;
public class Ejemplo_05_06a
{

    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 variables tengan el mismo nombre no afecta al funcionamiento del programa, siguen siendo distintas, porque cada una está en un bloque ("ámbito") distinto.

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

```
// Ejemplo_05_06b.cs
// Una variable global

using System;

public class Ejemplo_05_06b
{

    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);
    }

}
```

Ahora, el resultado del programa será:

```
n vale 7
Ahora n vale 8
```

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_05_07a.cs
// Modificar una variable recibida como parámetro - acercamiento

using System;

public class Ejemplo_05_07a
{

    public static void Duplicar(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);
        Duplicar(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, **no se conservan los cambios** que hagamos a esa variable que se ha recibido como parámetro.

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 las modificaciones 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_05_07b.cs
// Modificar una variable recibida como parámetro - correcto

using System;

public class Ejemplo_05_07b
{

    public static void Duplicar(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);
        Duplicar(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**". No podemos llamar a una función que tenga parámetros por referencia si los parámetros no tienen valor inicial. Por ejemplo, una función que devuelva la

primera y segunda letra de una frase sería así:

```
// Ejemplo_05_07c.cs
// Parámetros "out"

using System;
public class Ejemplo_05_07c
{

    public static void DosPrimerasLetras(string cadena,
        out char l1, out char l2)
    {
        l1 = cadena[0];
        l2 = cadena[1];
    }

    public static void Main()
    {
        char letra1, letra2;
        DosPrimerasLetras("Nacho", out letra1, out letra2);
        Console.WriteLine("Las dos primeras letras son {0} y {1}",
            letra1, letra2);
    }

}
```

Si pruebas este ejemplo, verás que **no** compila si cambias "out" por "ref", a no ser que des valores iniciales a "letra1" y "letra2".

Ejercicios propuestos:

Ejercicio propuesto 5.7.1: *Crea una función "Intercambiar", que intercambie el valor de los dos números enteros que se le indiquen como parámetro. Crea también un programa que la pruebe.*

Ejercicio propuesto 5.7.2: *Crea una función "Iniciales", que reciba una cadena como "Pepe Saltos" y devuelva las letras P y S (primera letra, y letra situada tras el primer espacio), usando parámetros por referencia. Crea un "Main" que te permita comprobar que funciona correctamente.*

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 ejemplo 05_07b, de modo que "Main" aparezca en primer lugar y "Duplicar" aparezca después, y seguiría compilando y funcionando igual:

```
// Ejemplo_05_08a.cs
// Función tras Main

using System;
public class Ejemplo_05_08a
{

    public static void Main()
    {
        int n = 5;
```

```
        Console.WriteLine("n vale {0}", n);
        Duplicar(ref n);
        Console.WriteLine("Ahora n vale {0}", n);
    }

    public static void Duplicar(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" (una única vez), y luego llamaremos a "Next" cada vez que queramos 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 único 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, no habría más remedio que utilizar "Random" y llamar dos veces a "Next".

Vamos a ver un ejemplo, que muestre en pantalla dos números al azar:

```
// Ejemplo_05_09_01a.cs
// Números al azar

using System;

public class Ejemplo_05_09_01a
{
    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(10, 21);
        Console.WriteLine("Otro entre 10 y 20: {0}", aleatorio2);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 5.9.1.1: *Crea un programa que imite el lanzamiento de un dado, generando un número al azar entre 1 y 6.*

Ejercicio propuesto 5.9.1.2: *Crea un programa que genere un número al azar entre 1 y 100. El usuario tendrá 6 oportunidades para acertarlo.*

Ejercicio propuesto 5.9.1.3: *Mejora el programa del ahorcado (4.4.9.3), para que la palabra a adivinar no sea tecleada por un segundo usuario, sino que se escoja al azar de un "array" de palabras prefijadas (por ejemplo, nombres de ciudades).*

Ejercicio propuesto 5.9.1.4: *Crea un programa que genere un array relleno con 100 números reales al azar entre -1000 y 1000. Luego deberá calcular y mostrar su media.*

Ejercicio propuesto 5.9.1.5: *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"; en el caso de las funciones trigonométricas, el ángulo se debe indicar en radianes, no en grados)

y también tenemos 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, que casi cualquier programador pueda necesitar:

- 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);

Un ejemplo más avanzado, usando funciones trigonométricas, que calculase el "coseno de 45 grados" podría ser:

```
// Ejemplo_05_09_02a.cs
// Ejemplo de funciones trigonométricas

using System;
public class Ejemplo_05_09_02a
{
    public static void Main()
    {
        double anguloGrados = 45;
        double anguloRadianes = anguloGrados * Math.PI / 180.0;

        Console.WriteLine("El coseno de 45 grados es: {0}",
            Math.Cos(anguloRadianes));
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 5.9.2.1: *Crea un programa que halle (y muestre) la raíz cuadrada del número que introduzca el usuario. Se repetirá hasta que introduzca 0.*

Ejercicio propuesto 5.9.2.2: *Crea un programa que halle cualquier raíz (de cualquier orden) 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.*

Ejercicio propuesto 5.9.2.3: *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 "CalcularRaicesSegundoGrado", que recibirá como parámetros los coeficientes a, b y c (por valor), 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 $x = -b \pm \sqrt{(b^2 - 4 \cdot a \cdot c)} / (2 \cdot a)$*

Ejercicio propuesto 5.9.2.4: *Haz un programa que pida al usuario 5 datos numéricos enteros, los guarde en un array, pida un nuevo dato y muestre el valor del array que se encuentra más cerca de ese dato, siendo mayor que él, o el texto "Ninguno es mayor" si ninguno lo es.*

Ejercicio propuesto 5.9.2.5: *Crea un programa que pida al usuario 5 datos numéricos reales, los guarde en un array, pida un nuevo dato y muestre el valor del array que se encuentra más cerca de ese dato en valor absoluto (es decir, el más próximo, sea mayor que él o menor que él).*

Ejercicio propuesto 5.9.2.6: *Crea una función "Distancia", que calcule la distancia entre dos puntos (x1,y1) y (x2,y2), usando la expresión d = raíz [$(x1-x2)^2 + (y1-y2)^2$].*

Ejercicio propuesto 5.9.2.7: *Crea un programa que pida al usuario un ángulo (en grados) y muestre su seno, coseno y tangente. Recuerda que las funciones trigonométricas esperan que el ángulo se indique en radianes, no en grados. La equivalencia es que 360 grados son 2*PI radianes.*

Ejercicio propuesto 5.9.2.8: *Crea un programa que muestre los valores de la función y = 10 * seno(x*5), para valores de x entre 0 y 72 grados.*

Ejercicio propuesto 5.9.2.9: *Crea un programa que "dibuje" la gráfica de la función y = 10 * seno(x*5), para valores de x entre 0 y 72 grados. Para ayudarte para escribir en cualquier coordenada, puedes usar un array de dos dimensiones, que primero rellenes y luego dibujes en pantalla (mira el ejercicio 5.9.1.5).*

Ejercicio propuesto 5.9.2.10: *Crea un programa que "dibuje" un círculo dentro de un array de dos dimensiones, usando las ecuaciones x = xCentro + radio * coseno(ángulo), y = yCentro + radio * seno(ángulo). Si tu array es de 24x79, las coordenadas del centro serían (12,40). Recuerda que el ángulo se debe indicar en radianes (mira el ejercicio 5.9.1.5 y el 5.9.2.9).*

5.9.3. Pero hay muchas más funciones…

En C# hay muchas más funciones de lo que parece. De hecho, salvo algunas palabras reservadas (int, float, string, if, switch, for, do, while...), gran parte de lo que hasta ahora hemos llamado "órdenes", son realmente "funciones", como Console.ReadLine (que devuelve la cadena que se ha introducido por teclado) o Console.WriteLine (que es "void", no devuelve nada). Nos iremos encontrando con otras muchas funciones a medida que avancemos.

5.10. Recursividad

La recursividad consiste en resolver un problema a partir de casos más simples del mismo problema. Una función recursiva es aquella que se "llama a ella misma", reduciendo la complejidad paso a paso hasta llegar a un caso trivial.

Dentro de las matemáticas tenemos varios ejemplos de funciones recursivas. Uno clásico es el "factorial de un número":

El factorial de 1 es 1:

```
1! = 1
```

Y el factorial de un número arbitrario es el producto de ese número por los que le siguen, hasta llegar a uno:

```
n! = n · (n-1) · (n-2) · ... · 3 · 2 · 1
```

(por ejemplo, el factorial de 4 es 4 • 3 • 2 • 1 = 24)

Si pensamos que el factorial de n-1 es

$$(n-1)! = (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Entonces podemos escribir el factorial de un número a partir del factorial del siguiente número:

$$n! = n \cdot (n-1)!$$

Esta es la definición recursiva del factorial, ni más ni menos. Esto se podría programar así:

```
// Ejemplo_05_10a.cs
// Funciones recursivas: factorial

using System;
public class Ejemplo_05_10a
{

    public static long Factorial(int n)
    {
        if (n==1)                // Aseguramos que termine (caso base)
            return 1;
        return n * Factorial(n-1);  // Si no es 1, sigue la recursión
    }

    public static void Main()
    {
        int num;
        Console.WriteLine("Introduzca un número entero: ");
        num = Convert.ToInt32(System.Console.ReadLine());
        Console.WriteLine("Su factorial es: {0}", Factorial(num));
    }

}
```

Dos consideraciones importantes:

- Atención a la primera parte de la función recursiva: es **muy importante** comprobar que hay **salida** de la función, para que nuestro programa no se quede dando vueltas todo el tiempo y deje el ordenador (o la tarea actual) "colgado". Debemos encontrar un "caso trivial" que alcanzar, y un modo de disminuir la complejidad del problema acercándolo a ese caso.
- Los factoriales **crecen rápidamente**, así que no conviene poner números grandes: el factorial de 16 es 2.004.189.184, luego a partir de 17 podemos obtener resultados erróneos, si usamos números enteros "normales".

¿Qué utilidad tiene esto? Más de la que parece: muchos problemas complicados se pueden expresar a partir de otro más sencillo. En muchos de esos casos, ese problema se podrá expresar de forma recursiva. Los ejercicios propuestos te ayudarán a descubrir otros ejemplos de situaciones en las que se puede aplicar la recursividad.

Ejercicios propuestos:

Ejercicio propuesto 5.10.1: *Crea una función que calcule el valor de elevar un número entero a otro número entero (por ejemplo, 5 elevado a 3 = 53 = 5 • 5 • 5 = 125). Esta función se debe crear de forma recursiva. Piensa cuál será el caso base (qué potencia se puede calcular de forma trivial) y cómo pasar del caso "n-1" al caso "n" (por ejemplo, si sabes el valor de 5^4, cómo hallarías el de 5^5 a partir de él).*

Ejercicio propuesto 5.10.2: *Como alternativa, crea una función que calcule el valor de elevar un número entero a otro número entero de forma NO recursiva (lo que llamaremos "de forma iterativa"), usando la orden "for".*

Ejercicio propuesto 5.10.3: *Crea un programa que emplee recursividad para calcular un número de la serie Fibonacci (en la que los dos primeros elementos valen 1, y para los restantes, cada elemento es la suma de los dos anteriores).*

Ejercicio propuesto 5.10.4: *Crea un programa que emplee recursividad para calcular la suma de los elementos de un vector de números enteros, desde su posición inicial a la final, usando una función recursiva que tendrá la apariencia: SumaVector(v, desde, hasta). Nuevamente, piensa cuál será el caso base (cuántos elementos podrías sumar para que dicha suma sea trivial) y cómo pasar del caso "n-1" al caso "n" (por ejemplo, si conoces la suma de los 6 primeros elementos y el valor del séptimo elemento, cómo podrías emplear esta información para conocer la suma de los 7 primeros).*

Ejercicio propuesto 5.10.5: *Crea un programa que emplee recursividad para calcular el mayor de los elementos de un vector. El planteamiento será muy similar al del ejercicio anterior.*

Ejercicio propuesto 5.10.6: *Crea un programa que emplee recursividad para dar la vuelta a una cadena de caracteres (por ejemplo, a partir de "Hola" devolvería "aloH"). La función recursiva se llamará "Invertir(cadena)". Como siempre, analiza cuál será el caso base (qué longitud debería tener una cadena para que sea trivial darle la vuelta) y cómo pasar del caso "n-1" al caso "n" (por ejemplo, si ya has invertido las 5 primeras letras, que ocurriría con la de la sexta posición).*

Ejercicio propuesto 5.10.7: *Crea, tanto de forma recursiva como de forma iterativa, una función diga si una cadena de caracteres es simétrica (un palíndromo). Por ejemplo, "DABALEARROZALAZORRAELABAD" es un palíndromo.*

Ejercicio propuesto 5.10.8: *Crear un programa que encuentre el máximo común divisor de dos números usando el algoritmo de Euclides: Dados dos números enteros positivos m y n, tal que m > n, para encontrar su máximo común divisor, es decir, el mayor entero positivo que divide a ambos: - Dividir m por n para obtener el resto r ($0 \leq r < n$) ; - Si r = 0, el MCD es n.; - Si no, el máximo común divisor es MCD(n,r).*

Ejercicio propuesto 5.10.9: *Crea dos funciones que sirvan para saber si un cierto texto es subcadena de una cadena. No puedes usar "Contains" ni "IndexOf", sino que debes analizar letra a letra. Una función debe ser iterativa y la otra debe ser recursiva.*

Ejercicio propuesto 5.10.10: *Crea una función que reciba una cadena de texto, y una subcadena, y devuelva cuántas veces aparece la subcadena en la cadena, como subsecuencia formada a partir de sus letras en orden. Por ejemplo, si recibes la palabra "Hhoola" y la subcadena "hola", la respuesta sería 4, porque se podría tomar la primera H con la primera O (y con la L y con la A), la primera H con la segunda O, la segunda H con la primera O, o bien la segunda H con la segunda O. Si recibes "hobla", la respuesta sería 1. Si recibes "ohla", la respuesta sería 0, porque tras la H no hay ninguna O que permita completar la secuencia en orden.*

5.11. Parámetros y valor de retorno de Main

Es muy frecuente que un programa llamado desde la "línea de comandos" tenga ciertas opciones que le indicamos como argumentos. Por ejemplo, bajo Linux o cualquier otro sistema operativo de la familia Unix, podemos ver la lista detallada de ficheros que terminan en .cs haciendo

```
ls -l *.cs
```

En este caso, la orden sería "ls", y las dos opciones (argumentos o parámetros) que le indicamos son "-l" y "*.cs".

La orden equivalente en MsDos y en el intérprete de comandos de Windows sería

```
dir *.cs
```

Ahora la orden sería "dir", y el parámetro es "*.cs".

Pues bien, estas opciones que se le pasan al programa se pueden leer desde C#. Se hace indicando un parámetro especial en Main, un array de strings:

```
static void Main(string[] args)
```

Para conocer esos parámetros lo haríamos de la misma forma que se recorre habitualmente un array cuyo tamaño no conocemos: con un "for" que termine en la longitud ("Length") del array:

```
for (int i = 0; i < args.Length; i++)
{
    System.Console.WriteLine("El parametro {0} es: {1}",
      i, args[i]);
}
```

Por otra parte, si queremos que nuestro programa **se interrumpa** en un cierto punto, podemos usar la orden "Environment.Exit". Su manejo habitual es algo como

```
Environment.Exit(1);
```

Es decir, entre paréntesis indicamos un cierto código, que suele ser (por convenio) un 0 si no ha habido ningún error, u otro código distinto en caso de que sí exista algún error.

Este valor se podría comprobar desde el sistema operativo. Por ejemplo, en MsDos y Windows se puede leer desde un fichero BAT o CMD usando "IF ERRORLEVEL", así:

```
IF ERRORLEVEL 1 ECHO Ha habido un error en el programa
```

Una forma alternativa de que "Main" indique errores al sistema operativo es no declarándolo como "void", sino como "int", y empleando entonces la orden "return" cuando nos interese (igual que antes, por convenio, devolviendo 0 si todo ha funcionado correctamente u otro código en caso contrario):

```
public static int Main(string[] args)
{
  ...
  return 0;
}
```

Un ejemplo que pusiera todo esto a prueba podría ser:

```
// Ejemplo_05_11a.cs
// Parámetros y valor de retorno de "Main"
// Introducción a C#

using System;

public class Ejemplo_05_11a
{
    public static int Main(string[] args)
    {
        Console.WriteLine("Parámetros: {0}", args.Length);

        for (int i = 0; i < args.Length; i++)
        {
            Console.WriteLine("El parámetro {0} es: {1}",
                i, args[i]);
        }

        if (args.Length == 0)
        {
            Console.WriteLine("No ha indicado ningún parámetro!");
            Environment.Exit(1);
        }

        return 0;
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 5.11.1: *Crea un programa llamado "suma", que calcule (y muestre) la suma de dos números que se le indiquen como parámetros en línea de comandos. Por ejemplo, si se teclea "suma 2 3" deberá responder "5", si se teclea "suma 2" responderá "2" y si se teclea únicamente "suma" deberá responder "no hay suficientes datos" y devolver un código de error 1.*

Ejercicio propuesto 5.11.2: *Crea una calculadora básica, llamada "calcula", que deberá sumar, restar, multiplicar o dividir los dos números que se le indiquen como parámetros. Ejemplos de su uso sería "calcula 2 + 3" o "calcula 5 * 60".*

Ejercicio propuesto 5.11.3: *Crea una variante del ejercicio 5.11.2, en la que Main devuelva el código 1 si la operación indicada no es válida o 0 cuando sí sea una operación aceptable.*

Ejercicio propuesto 5.11.4: *Crea una variante del ejercicio 5.11.3, en la que Main devuelva también el código 2 si alguno de los dos números con los que se quiere operar no tiene un valor numérico válido.*