

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 `{ }` 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 se empleará 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.

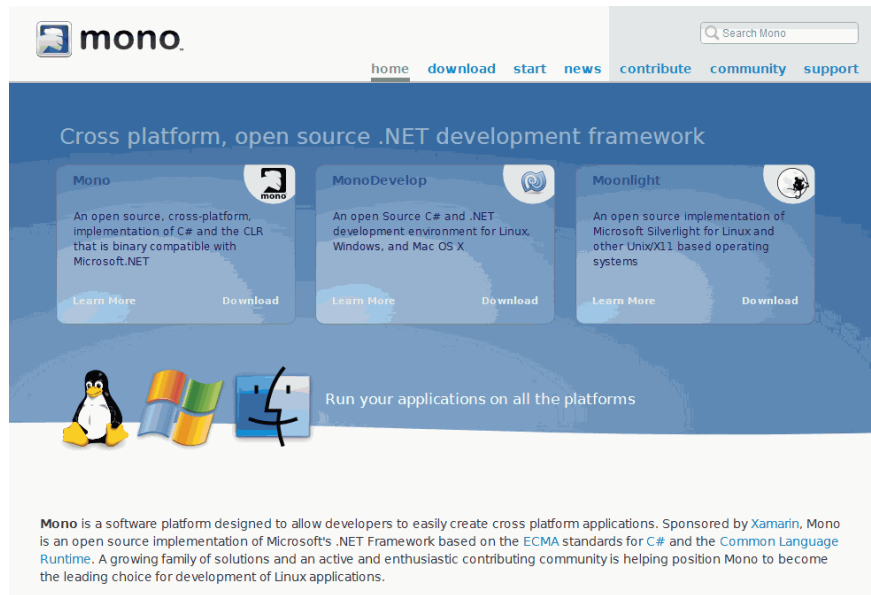
1.2. *Cómo probar este programa*

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/>



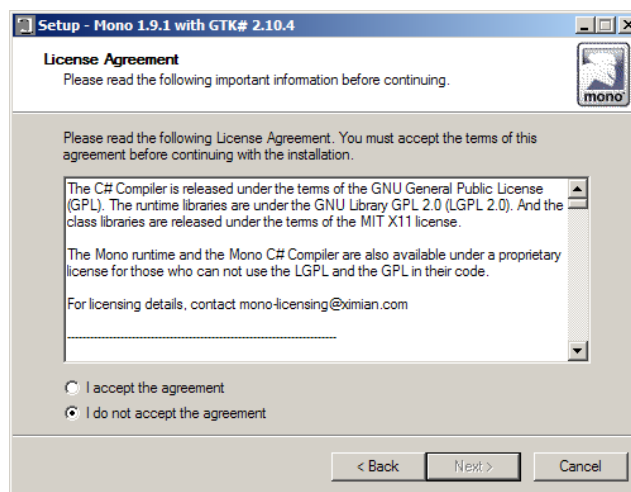
En la parte superior derecha aparece el enlace para descargar ("download now"), que nos lleva a una nueva página en la que debemos elegir la plataforma para la que queremos nuestro Mono. Nosotros descargaremos la versión más reciente para Windows (la 2.10.5 en el momento de escribir este texto).



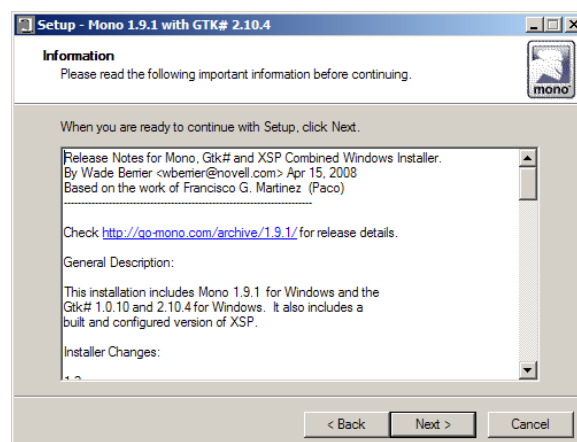
Se trata de un fichero de cerca de 90 Mb. Cuando termina la descarga, haremos doble clic en el fichero recibido, aceptaremos el aviso de seguridad que posiblemente nos mostrará Windows, y comenzará la instalación, en la que primero se nos muestra el mensaje de bienvenida:



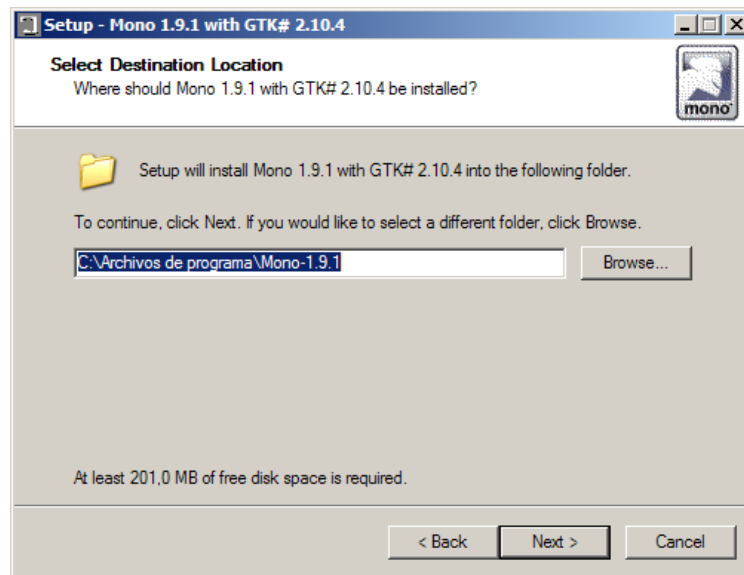
El siguiente paso será aceptar el acuerdo de licencia:



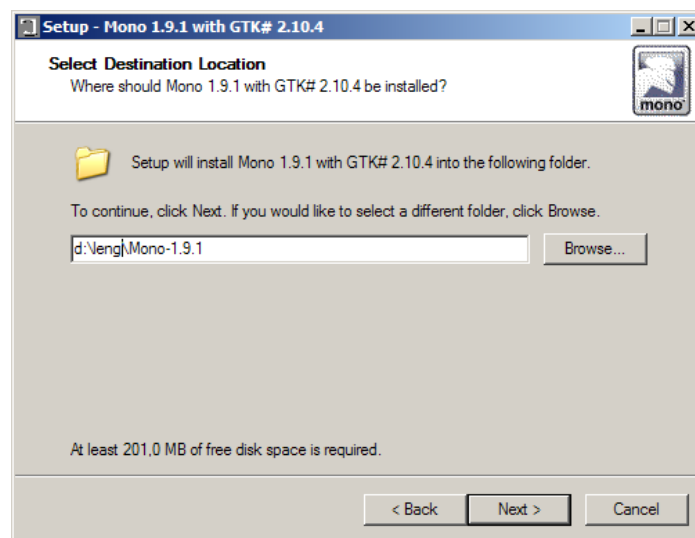
Después se nos muestra una ventana de información, en la que se nos avisa de que se va a instalar Mono x.x.x (donde x.x.x es la versión actual, por ejemplo 2.10.5), junto con las librerías Gtk# para creación de interfaces de usuario y XSP (eXtensible Server Pages, un servidor web).



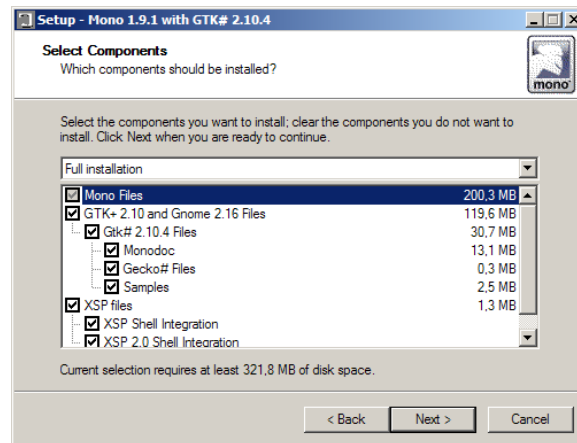
A continuación se nos pregunta en qué carpeta queremos instalar. Como es habitual, se nos propone que sea dentro de "Archivos de programa":



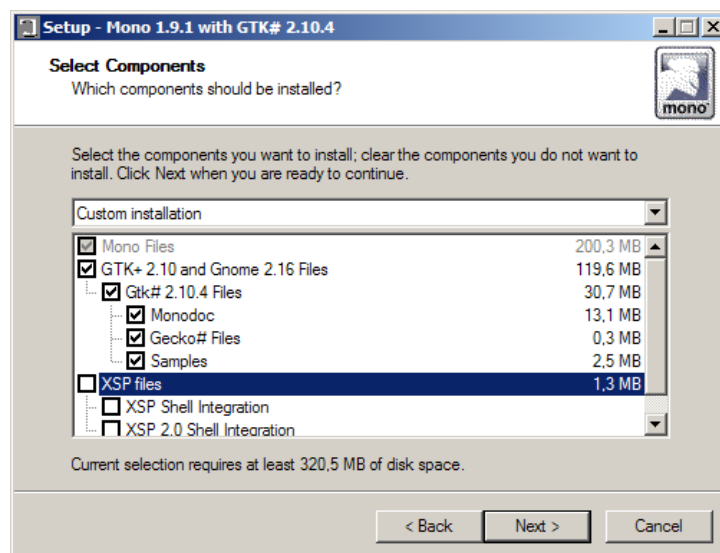
Yo no soy partidario de instalar todo en "Archivos de Programa". Mis herramientas de programación suelen estar en otra unidad de disco (D:), así que prefiero cambiar esa opción por defecto:



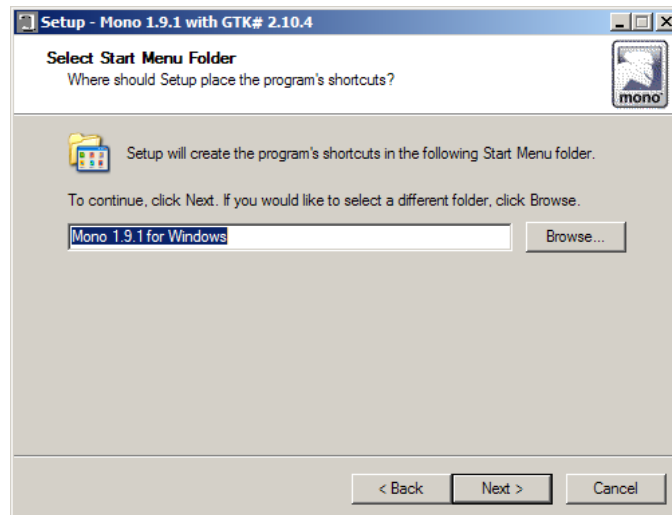
El siguiente paso es elegir qué componentes queremos instalar (Mono, Gtk#, XSP):



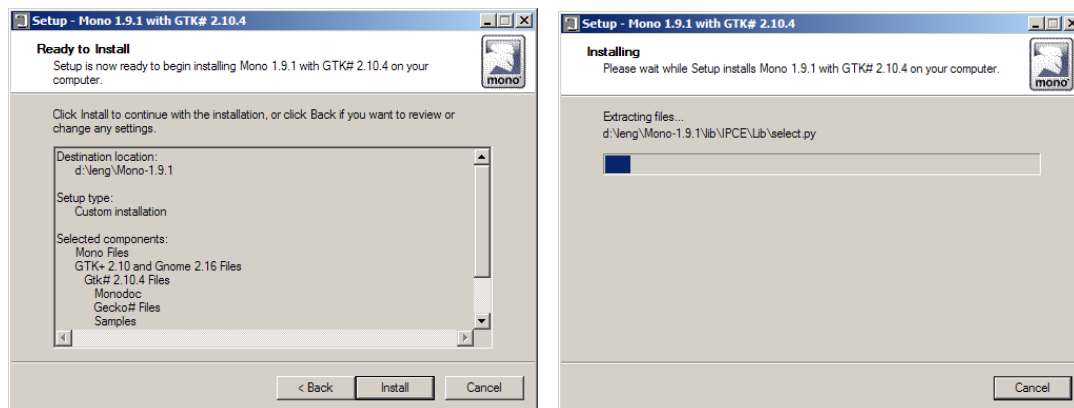
Nuevamente, soy partidario de no instalar todo. Mono es imprescindible. La creación de interfaces de usuario con Gtk# queda fuera del alcance que se pretende con este texto, pero aun así puede ser interesante para quien quiera profundizar. El servidor web XSP es algo claramente innecesario por ahora, y que además instalaría un "listener" que ralentizaría ligeramente el ordenador, así que puede ser razonable no instalarlo por ahora:



El siguiente paso es indicar en qué carpeta del menú de Inicio queremos que quede accesible:



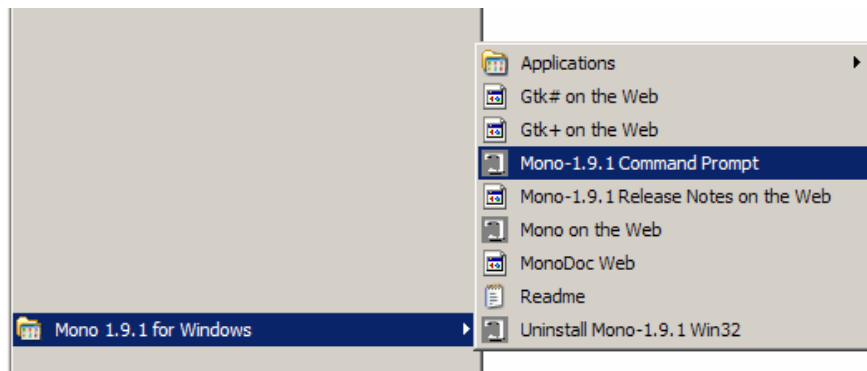
A continuación se nos muestra el resumen de lo que se va a instalar. Si confirmamos que todo nos parece correcto, comienza la copia de ficheros:



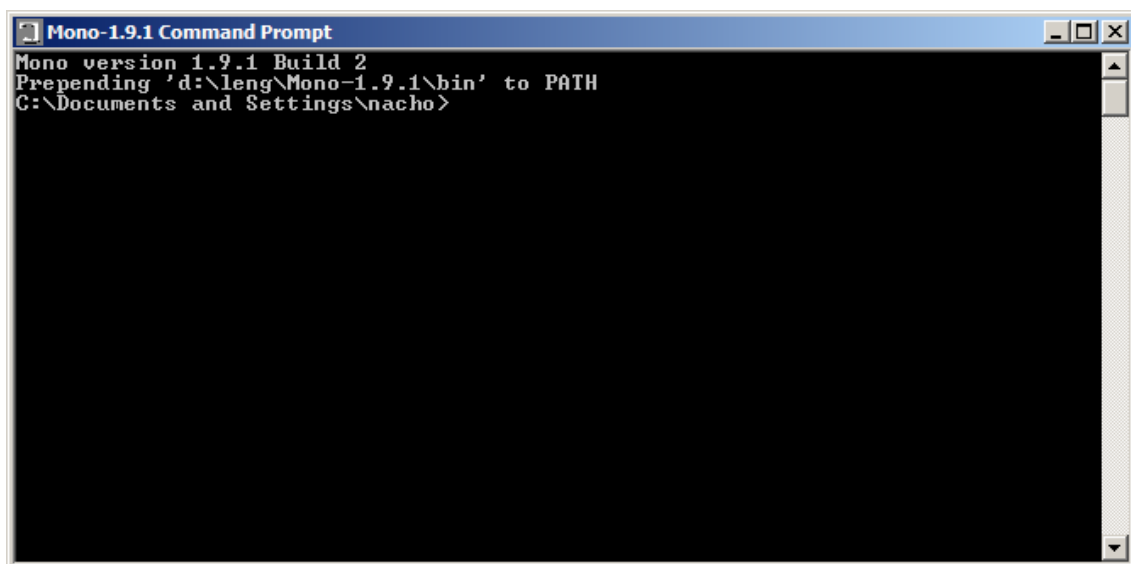
Si todo es correcto, al cabo de un instante tendremos el mensaje de confirmación de que la instalación se ha completado:



Mono está listo para usar. En nuestro menú de Inicio deberíamos tener una nueva carpeta llamada "Mono x.x.x for Windows", y dentro de ella un acceso a "Mono-x.x.x Command Prompt":



Si hacemos clic en esa opción, accedemos al símbolo de sistema ("command prompt"), la pantalla negra del sistema operativo, pero con el "path" (la ruta de búsqueda) preparada para que podamos acceder al compilador desde ella:

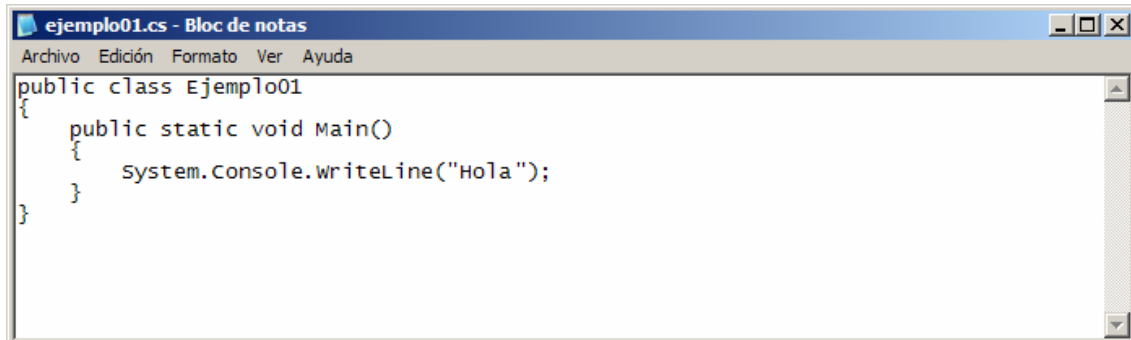


Quizá se nos lleve a una carpeta que esté dentro de "Documents and settings" o quizá incluso a alguna en la que no tengamos permiso para escribir, como "Windows\System32". Si queremos cambiar la carpeta de arranque de Mono, lo podemos hacer pulsando el botón derecho sobre la opción "Mono-x.x.x Command Prompt" del menú de inicio y escogiendo "Propiedades".

Para crear un programa, el primero paso será teclear el "fuente". Para ello podemos usar cualquier editor de texto. En este primer fuente, usaremos simplemente el "Bloc de notas" de Windows. Para ello tecleamos:

```
notepad ejemplo01.cs
```

Aparecerá la pantalla del "Bloc de notas", junto con un aviso que nos indica que no existe ese fichero, y que nos pregunta si deseamos crearlo. Respondemos que sí y podemos empezar a teclear el ejemplo que habíamos visto anteriormente:

A screenshot of a Notepad window titled "ejemplo01.cs - Bloc de notas". The window has a menu bar with "Archivo", "Edición", "Formato", "Ver", and "Ayuda". The text area contains the following C# code:

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

Guardamos los cambios, salimos del "Bloc de notas" y nos volvemos a encontrar en la pantalla negra del símbolo del sistema. Nuestro fuente ya está escrito. El siguiente paso es compilarlo. Para eso, tecleamos

```
gmcs ejemplo01.cs
```

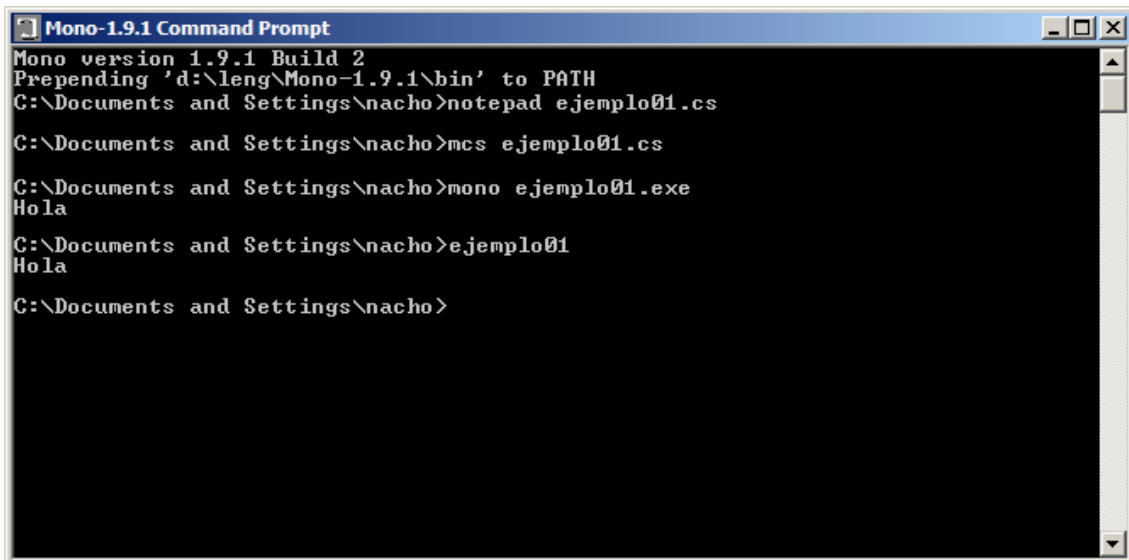
Si no se nos responde nada, quiere decir que no ha habido errores. Si todo va bien, se acaba de crear un fichero "ejemplo01.exe". En ese caso, podríamos lanzar el programa tecleando

```
mono ejemplo01.exe
```

y el mensaje "Hola" debería aparecer en pantalla.

Si en nuestro ordenador está instalado el "Dot Net Framework" (algo que debería ser cierto en las últimas versiones de Windows, y que no ocurrirá en Linux ni Mac OSx), no debería hacer falta decir que queremos que sea Mono quien lance nuestro programa, y podremos ejecutarlo directamente con su nombre:

```
ejemplo01
```



```
Mono-1.9.1 Command Prompt
Mono version 1.9.1 Build 2
Prepending 'd:\leng\Mono-1.9.1\bin' to PATH
C:\Documents and Settings\nacho>notepad ejemplo01.cs
C:\Documents and Settings\nacho>mcs ejemplo01.cs
C:\Documents and Settings\nacho>mono ejemplo01.exe
Hola
C:\Documents and Settings\nacho>ejemplo01
Hola
C:\Documents and Settings\nacho>
```

1.2.2 Otros editores más avanzados

Si quieres un editor más potente que el Bloc de notas de Windows, puedes probar Notepad++, que es gratuito (realmente más que eso: es de "código abierto") y podrás localizar fácilmente en Internet. Geany también es una alternativa muy interesante, disponible para muchos sistemas operativos.

Si prefieres un entorno desde el que puedas teclear, compilar y probar tus programas, incluso los de gran tamaño que estén formados por varios ficheros, en el apartado 6.13 hablaremos de SharpDevelop (para Windows), y de MonoDevelop (para Windows, Linux y Mac). Si quieres saber cosas sobre el entorno "oficial" de desarrollo, llamado Visual Studio, lo tienes en el Apartado 6.8.

Hay un **posible problema** que se debe tener en cuenta: algunos de estos entornos de desarrollo muestran el resultado de nuestro programa y luego regresan al editor tan rápido que no da tiempo a ver los resultados. Una solución provisional puede ser añadir "System.Console.ReadLine()" al final del programa, de modo que se quede parado hasta que pulsemos Intro:

```
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 queremos 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
*	Multipliación
/	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 queremos 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. Existe dos formas de indicar comentarios. En su forma más general, los escribiremos 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 indicaremos 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.:

```

1 public class Ejercicio001
2 {
3     public static void Main( )
4     {
5         System.Console.WriteLine("Hola");
6     }
7 }

```

Ejercicios propuestos:

- **(1.9.1)** Multiplicar dos números tecleados por usuario. El programa deberá contener un comentario al principio, que recuerde cual es su objetivo.
- **(1.9.2)** El usuario tecleará dos números (x e y), y el programa deberá calcular cuál es el resultado de su división y el resto de esa división. Deberás usar "Write" en vez de "WriteLine" para pedir los datos, e incluir un comentario con tu nombre y la fecha en que has realizado el programa.
- **(1.9.3)** El usuario tecleará dos números (a y b), y el programa mostrará el resultado de la operación $(a+b)*(a-b)$ y el resultado de la operación a^2-b^2 .
- **(1.9.4)** Sumar tres números tecleados por usuario.
- **(1.9.5)** Pedir al usuario un número y mostrar su tabla de multiplicar. Por ejemplo, si el número es el 3, debería escribirse algo como

```

3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
...
3 x 10 = 30

```

- **(1.9.6)** Crea un programa que convierta de grados Celsius (centígrados) a Kelvin y a Fahrenheit: pedirá al usuario la cantidad de grados centígrados y usará las siguientes tablas de conversión: $kelvin = celsius + 273$; $fahrenheit = celsius \times 1.8 + 32$
- **(1.9.7)** Pide al usuario una cantidad de "millas" y muestra la equivalencia en metros, usando: 1 milla = 1609.344 metros.