Prof. Navarro Gonzalo
Laboratorio de Algoritmos y Estructura de datos

Introducción a C#

Mostrar datos por pantalla

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.

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

Esto escribe "Hola" en la pantalla. Pero hay muchas "cosas raras" alrededor de ese "Hola". En este primer análisis, iremos desde 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");** : WriteLine siempre irá precedido de "Console." porque es una orden de manejo de la "consola" (la pantalla "negra" en modo texto del sistema operativo).
- **Las llaves { y }** se usan para delimitar un bloque de programa. En nuestro caso, se trata del bloque principal del programa (Main).
- **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".
- **public class Ejemplo:** De momento pensaremos que es el nombre de nuestro programa. Una línea como esa deberá existir también siempre en nuestros programas y eso de "public class" será obligatorio.
- **Console.ReadKey();** Hace que el programa solicite que se presione una tecla. Lo utilizaremos para evitar que el programa se cierre al terminar de ejecutarse.

Variables

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.

Necesitamos usar variables, zonas de memoria en las que guardemos los datos con los que vamos a trabajar y también los resultados temporales.

Declarar variables

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, 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".

Asignar valores a variables

Podemos darle un valor a esa variable durante el programa con la siguiente línea:

```
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;
```

Después ya podemos hacer operaciones con las variables:

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

Mostrar el valor de una variable en pantalla

Esta línea mostrar la el resultado de trabajar con constantes:

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

Esta línea mostrará la palabra suma:

```
Console.WriteLine(“suma”);
```

Si se trata de una variable es idéntico pero sin comillas, para que el compilador se refiera a la variable con dicho nombre:

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

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

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

Si queremos mostrar de más de una variable, detallaremos en el texto dónde debe aparecer cada una de ellas, usando {0}, {1} y tantos números sucesivos como sea necesario, y tras el texto incluiremos los nombres de cada una de esas variables, separados por comas:

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

Identificadores

Los nombres de variables (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 intermedios. También hay que recordar que las vocales acentuadas y la eñe son problemáticas, porque no son letras "estándar" en todos los idiomas, así que no se pueden utilizar como parte de un identificador en la mayoría de lenguajes de programación.

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.

Hay que recordar que en C# las mayúsculas y minúsculas se consideran diferentes, de modo que si intentamos hacer, el compilador protestará y nos dirá que no conoce esa variable, porque la habíamos declarado con el identificador **primerNumero;**

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

Operadores matemáticos

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

Así, podemos calcular el resto de la división entre dos números de la siguiente forma:

```
public class Ejemplo
{
   public static void Main()
   {
      Console.WriteLine("El resto de dividir 19 entre 5 es");
      Console.WriteLine(19 % 5);
   }
}
```

Orden de prioridad de los operadores

- 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.

Ingreso de datos por teclado

Hasta ahora hemos tenido datos prefijados, pero eso es poco frecuente en el mundo real. Es mucho más habitual que los datos los introduzca el usuario, o que se lean desde un fichero, o desde una base de datos, o se reciban de Internet o cualquier otra red. El primer caso que veremos será el de interaccionar directamente con el usuario.

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 Console.WriteLine, también existe Console.ReadLine ("leer línea"). Para leer textos, haríamos

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

pero eso ocurrirá un poco más adelante, cuando veamos cómo manejar textos. De momento, nosotros sólo sabemos manipular números enteros, así que deberemos convertir ese dato a un número entero, usando **Int.Parse**:

```
primerNumero = Int.Parse(Console.ReadLine());
```

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

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


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


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

Escribir sin avanzar de línea

Vimos cómo usar {0} para escribir en una misma línea datos calculados y textos prefijados. Pero hay otra alternativa, que además nos permite también escribir un texto y pedir un dato a continuación, en la misma línea de pantalla: utilizando "Write" en vez de "WriteLine", así:

```
using System;


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


      Console.Write("Introduce el primer número: ");
      primerNumero = int.Parse(Console.ReadLine());
      Console.Write("Introduce el segundo número: ");
      segundoNumero = int.Parse(Console.ReadLine());
      suma = primerNumero + segundoNumero;


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

Incluso el último "WriteLine" de varios datos se podría convertir en varios Write (aunque generalmente eso hará el programa más largo y no necesariamente más legible), así

```
      Console.Write("La suma de ");
      Console.Write(primerNumero);
      Console.Write(" y ");
      Console.Write(segundoNumero);
      Console.Write(" es ");
      Console.WriteLine(suma);
```

Comentarios

Podemos escribir comentarios, que el compilador ignorará, pero que pueden ser útiles para nosotros mismos, haciendo que sea más fácil recordar el cometido un fragmento del programa más adelante, cuando tengamos que ampliarlo o corregirlo.

Existen dos formas de indicar comentarios. En su forma más general, los escribiremos entre /* y */:

```
int suma;  /* 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 podría ser:

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


public class Ejemplo
{
   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 de una línea":

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

De modo que el programa anterior se podría reescribir usando comentarios de una línea:

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

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