Tema 2:
Introducción a la programación

***Objetivos del tema**: Es el momento de comenzar a estudiar un lenguaje de programación que permita concretar una solución a un determinado en software. En este tema comenzaremos a ver el lenguaje de programación C#.*

CEU

1. Nuestro primer programa en C#

Un programa en C# es un fichero de texto, normalmente con extensión .cs, que contiene las sentencias del lenguaje. A continuación mostramos un programa muy simple que imprime en pantalla el texto "¡Hola mundo!":

```
using System;

public class Ejemplo1{
 public static void Main(string[] args){
      Console.WriteLine("¡Hola Mundo!");}
}
```

La ejecución del programa comienza por la primera línea de la función principal, la función *Main*. En este caso la única sentencia que tiene la función principal es un *Console.WriteLine*, que imprime por pantalla el texto que se le pasa como argumento e introduce un salto de linea. Si no quisiéramos introducir salto de linea valdría solo con *Console.Write*.

Nota: Pasar por argumento ocurre cuando se proporcionan valores a los parámetros de un método al llamarlo. Por ejemplo, en el caso del método *Console.WriteLine*, le hemos pasado el valor "¡Hola Mundo!". Cada método tendrá diferentes parámetros, los cuales se irán viendo a medida que avancemos.

2. Componentes de un programa en C#

Un programa en C# puede contener los siguientes elementos: directivas de uso (using), comentarios, palabras reservadas, identificadores, símbolos y literales. Veamos un ejemplo de un programa un poco más complejo que el anterior:

```
/* ejemplo2.cs */

using System;

public class Ejemplo2 { //declaración de clase

      public static void Main(string[] args){ //Main: método principal

            int numero = 5;

            Console.WriteLine("El número es : " + numero);

      }

}
```

En este programa:

- using System: es una directiva de uso
- /* este es ejercicio 1*/: es un comentario
- Public, static, void, int: son palabras reservadas.
- Main, Ejemplo2, numero: son identificadores
- ; () {} >, “” : son símbolos
- “El número es :”, 5: son literales, alfanumérico y numérico, respectivamente.

El formato en que se deben describir los distintos componentes de un programa en C# es bastante libre, y se rige por las siguientes reglas:

- No se pueden pegar palabras: No es correcto escribir intnumero; debe ser int numero.
- No se pueden partir palabras: No es correcto escribir Ma in; debe ser Main.
- Diferenciación entre mayúsculas y minúsculas: Se diferencian las mayúsculas de las minúsculas (numero es distinto de Numero).
- Espacios en blanco: El número de espacios en blanco entre palabras (1 o más) o símbolos (0 o más) no importa.
- Formato de las líneas: El formato de las líneas es libre; se puede escribir el mismo programa en una sola línea o en varias, siempre que no dividamos una palabra o un literal en dos partes. El siguiente ejemplo muestra dos formas equivalentes de escribir una sentencia en C#:

```
int numero; numero = 3; Console.WriteLine("El número es: " + numero);
```

Es lo mismo que:

```
int numero;

numero = 3;

Console.WriteLine("El número es: " + numero);
```

- Cada sentencia (no cada línea) finaliza con un punto y coma (generalmente).
- Cualquier identificador (variable, constante o función) que usemos en el programa debe ser definido antes de ser utilizado. El compilador traduce el programa leyéndolo de arriba a abajo y da un error si encuentra que se utiliza algo que no ha sido definido previamente.

A continuación veremos más en detalle cada uno de los componentes que pueden aparecer en un programa C#.

3. Directivas de C#

3.1. Directivas de uso

La directiva using en C# permite incluir y utilizar clases y métodos desde un archivo de código, sin necesidad de incluir todo el código cada vez. Esto simplifica el código y mejora su legibilidad. También se utiliza para importar que librerías se utilizarán, tanto internas de C#, como externas descargadas de internet. La directiva using siempre se colocan al principio del archivo. El formato de esta directiva es:

```
|   using System;
```

En este ejemplo, la directiva using está importando la librería System, esta contiene múltiples funciones relacionadas con el sistema. Por ejemplo, permite usar *Console.WriteLine()*, si no importaremos la librería deberíamos utilizar *System.Console.WriteLine()* cada vez que quisiéramos imprimir algo por pantalla.

Para leer una linea de texto desde consola se utiliza *Console.ReadLine()*, esta se debe de igualar a una variable string para almacenar la información introducida, de esta manera:

```
|       string leido= Console.ReadLine();
```

Tras la ejecución, leido pasará a tener el valor que haya introducido el usuario.

Ejercicio de clase: Pide al usuario que introduzca su nombre e imprímelo por pantalla con el formato: “Hola [nombre]!!!”.

3.2. Comentarios

Los comentarios en un programa sirven para documentarlo, de forma que sea comprensible tanto para el que lo ha escrito como para otras personas. Los comentarios en C# van entre los caracteres /* y */. El compilador no hace caso del contenido de los comentarios. Un comentario puede colocarse en cualquier parte del programa, salvo en medio de una palabra o un literal, y puede ocupar varias líneas.

```
/* este programa es un ejemplo */
int a,b, /* coeficientes */ err; /* error */
/* comentario
de 2 lineas  */
public class Ejemplo2 {  /* comienzo de la clase*/
 public static void Main(string[] args){ /* comienzan las sentencias*/
   /* hacer aqui la lectura de datos */
   a=2; /* la variable a vale 2*/
 }/* fin del main */
} /* fin de la clase */
```

Los comentarios también se pueden poner utilizando "//". Este comentario afecta todo el texto que se encuentre en la misma línea donde aparecen las dos barras y después de ellas. A continuación volvemos a presentar el mismo código del ejemplo anterior empleando este nuevo tipo de comentario en todas las situaciones en las que es posible:

```
// este programa es un ejemplo
int a,b, /* coeficientes*/ err; // error
/* comentario
de 2 lineas  */
public class Ejemplo2 {  // comienzo de la clase
 public static void Main(string[] args){ // comienzan las sentencias
   // hacer aqui la lectura de datos
   a=2; // la variable a vale 2
 }// fin del main
} // fin de la clase
```

3.3. Literales

Son caracteres cuyo valor es literalmente el que aparece escrito en el programa. Los literales pueden ser numéricos o alfanuméricos. Los números que aparecen en cualquier programa son literales numéricos. Los literales alfanuméricos son caracteres o cadenas de caracteres, que pueden contener letras, números y otros símbolos. Los caracteres individuales se encierran entre comillas simples ('a') y las cadenas de caracteres entre comillas dobles ("hola"), aunque sean cadenas de un sólo carácter (“e”).

Los literales numéricos pueden estar en base 10 (decimal) u otras bases como la base 16 (hexadecimal) o la base 2 (binario). Los números pueden ser enteros o reales. Para los reales se utiliza el punto decimal (.) y en ningún caso se utiliza la coma (,). Los números reales se pueden escribir en notación científica, como 1.34E-18.

3.4. Palabras reservadas

Son palabras de C# que tienen un uso establecido y no pueden ser utilizadas para ninguna otra función como, por ejemplo, para ser identificadores. El compilador de C# produce un error si estas palabras se utilizan de forma no permitida. Son palabras reservadas los nombres de tipos de datos, instrucciones y algunas más:

- **Palabras reservadas de tipos de datos:** int, bool, void, float, string, char…
- **Palabras reservadas relacionadas con modificadores (acceso, propiedades):** public, private, static, protected, const…
- **Palabras reservadas relacionadas con el control de flujo**: if-else, do-while, for, break, return, continue, switch, case, default…
- **Palabras reservadas relacionadas con clases**: class, this, struct…

3.5. Identificadores

Son palabras que sirven para dar nombre a variables, funciones y tipos de datos. Tienen que cumplir las siguientes reglas de formación:

- Su longitud debe estar entre 1 y 32 caracteres.
- Pueden contener letras del abecedario inglés, números y el símbolo _ (guión bajo).
  - No pueden contener espacios ni ningún símbolo y se distinguen las mayúsculas de las minúsculas.
- El primer carácter debe ser una letra o _.

- El nombre de un identificador no puede coincidir con el nombre de una palabra reservada.

A diferencia de las palabras reservadas, cuyo significado no puede ser modificado por el usuario, los identificadores incluidos en las librerías del compilador pueden ser redefinidos (es decir, definidos con un nuevo significado), aunque no es conveniente, ya que en este caso pierden su significado original.

3.6. Símbolos

En C# se usan los símbolos siguientes:

- ; para indicar el final de una sentencia.
- {} para indicar el principio y el fin de un bloque de sentencias o de una función.
- () para indicar los argumentos de una función, para encerrar una expresión lógica en las sentencias de control de flujo y para indicar la precedencia en expresiones aritméticas.
- /* */ para indicar el principio y el fin de un comentario.
- , para separar varias variables en una declaración, varios parámetros de una función. También puede separar varias sentencias en una secuencia de sentencias (algo poco usado).
- . para separar el nombre de una variable unión o estructura del nombre de sus campos.
- -> para separar el nombre de un puntero del de un campo.
- ? : en la sentencia-expresión condicional.

Además se utilizan los siguientes símbolos de operadores aritméticos, relacionales, lógicos y de bits.

Operadores Aritméticos:

- = Asignación.
- + Suma.
- - Resta, signo negativo.
- * Multiplicación
- / División (cociente); aplicado sobre datos enteros en la división de números enteros (9/4=2).

- % División en módulo (resto: 9%4=1).
- ++ Incremento en 1.
- - - Decremento en 1.
- += Suma y asignación sobre una misma variable; int x = 0; x+=2; //x ahora vale 2
- -= Resta y asignación sobre una misma variable; int x = 0; x-=2; //x ahora vale -2
- *= Multiplicación y asignación sobre una misma variable; int x = 1; x*=2; //x ahora vale 2
- /= División y asignación sobre una misma variable; int x = 4; x/=2; //x ahora vale 2

En C# a diferencia de en otros lenguajes no existe el operador exponenciación. Tampoco existen los operadores Seno o Coseno. Si se desea realizar una operación de exponenciación se deberá invocar el método correspondiente de la clase Math (incluido en la librería System). Estos métodos son estáticos, por lo que no es necesario crear un objeto de dicha clase. Su sintaxis general es:

```
|  Math.metodo(argumentos)
```

Algunos ejemplos son:

```
|  Math.Pow( Base, Exponente) -> Ej:  Math.Pow(2,4)= 24
|  Math.Sqrt(número) -> Ej: Math.Sqrt(16)=4
|  Math.Sin(águalo en radiantes) -> Ej: (int) Math.Sin(Math.PI)= 0
```

Operadores Relacionales:

Permiten comprobar relaciones entre un par de datos:

- == Igual que
- != Distinto que
- > Mayor que
- >= Mayor o igual que
- < Menor que
- <= Menor o igual que

Operadores de bits:

Operan directamente sobre los bits de un dato, no sobre valores lógicos.

- & AND
- | OR
- ~ NOT
- ^ XOR (OR Exclusivo)

A continuación mostramos un código de ejemplo que muestra cómo funcionan los operadores de incremento y decremento, así como los operadores que permiten realizar una operación aritmética sobre una variable y asignar el resultado a esa misma variable:

```
/*ejemplo3.cs*/

using System;

public class Ejemplo3{

    public static void Main(string[] args) {

        int entero = 1;

        float real = 10.0F;

        Console.WriteLine("entero: " + entero);

        //imprimirá 1 ya que el entero se muestran antes de incrementarse

        Console.WriteLine("entero++: "+ entero++);

        //pero después de ejecutar la sentencia entero vale 2

        Console.WriteLine("entero: "+ entero);

        //imprimirá 3 ya que se incrementa antes de mostrarse

        Console.WriteLine("++entero: "+ ++entero);

        //imprimirá 3 ya que el entero se muestra antes debe incrementarse

        Console.WriteLine("entero--: "+ entero--);

        //pero tras ejecutar la sentencia entero vale 2

        Console.WriteLine("entero: "+ entero);

        //imprimirá 1 ya que se decrementa antes de mostrarse
```

```
        Console.WriteLine("--entero: "+ --entero);
        Console.WriteLine("real: "+ real);
        real += 2; //equivalente a real = real + 2;
        Console.WriteLine("real += 2: "+ real);
        real -= 6; //equivalente a real = real - 6;
        Console.WriteLine("real -= 6: "+ real);
        real *= 3; //equivalente a real = real * 3;
        Console.WriteLine("real *= 3: "+ real);
        real /= 9; //equivalente a real = real / 9;
        Console.WriteLine("real /= 9: "+ real);
    } }
```

A continuación presentamos un código de ejemplo que muestra cómo emplear variables booleanas para representar valores lógicos y como los operadores lógicos actúan sobre ellas:

```
/*ejemplo4.cs*/
using System;
public class Ejemplo4{
    public static void Main(string[] args) {
        bool variableLogica = true, variableLogica2;
        variableLogica2 = !variableLogica; //false
        Console.WriteLine("\n\nTabla de verdad del operador &&:\n");
        Console.WriteLine("false && false: " + (false && false));
        Console.WriteLine("false && true: " + (false && true));
        Console.WriteLine("true && false: " + (true && false));
        Console.WriteLine("true && true: " + (true && true));
        Console.WriteLine("\n\nTabla de verdad del operador ||:\n");
        Console.WriteLine("false || false: " + (false || false));
        Console.WriteLine("false || true: " + (false || true));
```

```
        Console.WriteLine("true || false: " + (true || false));

        Console.WriteLine("true || true: " + (true || true));

    } }
```

4. Conversiones entre tipos de datos

A menudo en una misma operación estarán involucrados datos cuyo tipo difiere; por ejemplo podemos tener que sumar el valor de un número entero al valor de un número real. En este apartado explicaremos cómo se realizan en C# las conversiones entre los distintos tipos de datos que están involucrados en una operación, así como la forma de cambiar el tipo de dato que devuelve la operación si éste no es el deseado.

4.1. Conversiones implicitas de tipos de datos

Cuando se aplica un operador aritmético sobre dos operandos de distinto tipo el operando de menor precisión se convierten al tipo de dato que permite una mayor precisión y rango de representación numérica. Esto quiere decir que:

1. Si cualquier operando es long todos se convertirán en long.
2. Si cualquier operando es double todos se convertirán en double.
3. Si cualquier operando es float y no hay ningún double todos se convertirán a float.

Si no estamos en ninguna de las situaciones anteriores en la operación sólo intervienen operandos enteros. En este caso se aplica a las siguientes reglas:

1. Si ambos operandos tienen signo (signed) o ambos operandos son sin signo (unsigned) el operando resultado será el del tipo de dato de mayor rango de representación que esté involucrado en operación. Así, si un operando es float y no hay datos double todos se convertirán en float.
2. En caso contrario, si uno de los operandos tiene signo y el otro no, si el operando sin signo tiene un rango mayor o igual de valores que los demás operandos entonces el operando con signo se convierte al tipo del operando sin signo.
3. En caso contrario, si el operando con signo puede representar todos los valores del tipo del operando sin signo, entonces el operando sin signo se convierte al tipo del operando con signo.

4. Si no se da ninguna de las situaciones anteriores ambos operandos se convierten a entero sin signo correspondiente al tipo del operando con signo.

4.2. Conversiones explicitas (cast)

Puede darse el caso de estar operando valores enteros con valores reales y, a pesar de que el resultado puede tener una parte decimal, estar interesado sólo en la parte entera del resultado. También puede suceder que, aunque el resultado de la operación teórico deba de ser de un determinado tipo para que no se produzcan errores, nosotros sepamos que dada la naturaleza de los operandos es posible almacenar el resultado en otro tipo de dato con menos precisión o menor rango de representación. En estos casos se puede indicar de modo explícito al compilador que se desea realizar una conversión del tipo de datos. A esta operación se le denomina cast o molde.

La forma de realizar esta operación es:

```
|  variable_nueva= (tipo_nuevo) expression;
```

En este esquema, *tipo_nuevo* representa al nombre reservado del tipo de dato al que quieres convertir el valor que representa “expression”. Así, por ejemplo, si quiero realizar la división 13/7 y quiero que su resultado sea un real puedo escribir:

```
| float numero = (float) 13/7;
```

también puede usarse para convertir a un tipo de menor precisión o menor rango de representación, aunque en esta ocasión es posible que se pierde información. En el caso en el que la conversión se realice de un número real a un número entero el número real siempre se truncará, no se redondea. Así, si hacemos:

```
|  int i;
|  float f= 4.77F;
|  i= (int) f;
```

el resultado que se almacena en la variable i será 4.

En el siguiente código se hace uso tanto de conversiones implícitas como explícitas de tipo:

```
public class Ejemplo5 {

    public static void Main(string[] args)  {

        int unInt = 9, otroInt;

        float unReal = 47.9F;
```

```
        Console.WriteLine("unInt: "+ unInt + " unReal: "+ unReal);
        otroInt = (int)unReal; //empleo de un cast
        Console.WriteLine("unReal: " + unReal + " otroInt: " + otroInt);
        unReal = otroInt;
        Console.WriteLine("unReal: " + unReal + " otroInt: " + otroInt);
}}
```

5. *Operador sizeof*

Este operador devuelve el tamaño en bytes que ocupa un determinado tipo de variable. Para emplear este operador simplemente lo invocamos y a continuación, entre paréntesis, indicamos el tipo de dato del cual queremos conocer el tamaño: sizeof(int). A continuación mostramos un programa que muestra el tamaño que ocupa cada uno de los tipos de datos básicos en C#.

```
using System;
public class Ejemplo4{
    public static void Main(string[] args)
    {
        Console.WriteLine("Un char ocupa " + sizeof(char)+ " bytes");
        Console.WriteLine("Un signed int ocupa "+ sizeof(int) + " bytes");
        Console.WriteLine("Un unsigned int ocupa "+ sizeof(uint) +" bytes");
       Console.WriteLine("Un unsigned short ocupa "+ sizeof(ushort) + " bytes");
        Console.WriteLine("Un signed short ocupa "+ sizeof(short) + " bytes");
        Console.WriteLine("Un long ocupa " + sizeof(ulong) + " bytes");
        Console.WriteLine("Un signed long ocupa "+ sizeof(long) + " bytes");
    }}
```

6. *Ejercicios*

1. Escribir un programa que defina una variable que almacene 3 meses, otra que almacene 12 días, y otra que almacene 2 horas. Con esas variables, calcula cuántos segundos son los meses, días y horas almacenados en esas variables.

2. Escribe un programa que calcule el número de segundos que hay en una semana y el número de horas que hay en un año de 365 días.

3. Escribir un programa que defina variables que representen el número de días de un año, el número de horas que tiene un día, el número de minutos que tiene una hora y el número de segundos que tiene un minuto. Emplear las variables que ocupen el mínimo espacio de memoria posible. A continuación, calcular el número de segundos que tiene un año y almacenar el valor del cálculo en otra variable.

4. Escribir un programa que almacene los coeficientes de una función de segundo grado (ax +bx+c) en variables reales. Inicializar con cualquier valor dichas variables y a continuación calcular el valor de la función definida en los puntos x= 42.32 y x=27.35.

5. Escribir un programa que defina 4 variables de tipo carácter que contengan, espectivamente, los caracteres 'J', 'n', '8' y '?'. Suma, almacena y muestra la variable con el valor de la suma de las cuatro variables tipo carácter.

6. Escribir un programa que muestre por consola los mayores números enteros que se pueden representar mediante un char, short e int. Para ello utiliza la función sizeof, y lo estudiado sobre bits en el tema anterior.

7. Escribir un programa que muestre por consola todos los posibles valores para a||b||c, siendo a, b y c tres variables lógicas.

8. Escribir un programa que muestre por consola todos los posibles valores para a&&b&&c, siendo a, b y c tres variables lógicas.

9. Escribir un programa que convierta los números -0.9, -0.1, 0.1, y 0.9 a entero. Usa casts. Observa cómo se realiza la conversión.