

2. Tipos e instrucciones básicas

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

1/59

Nuestro segundo programa en C#

Problema: calcular el área de un triángulo dada su base y su altura.

Especificación y análisis:

- ▶ input: *base* y *altura* de un triángulo ($base, altura \geq 0$)
- ▶ output: área del triángulo, $base * altura / 2$

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

2/59

Diseño del algoritmo

Algoritmo (muy sencillo):

- solicitar (por pantalla) al usuario y recoger de teclado los datos *base* y *altura*
- calcular $area = base * altura / 2$
- escribir el resultado *area* en pantalla

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

3/59

Codificación del programa

```
using System;

class AreaTriangulo
{
    static void Main ()
    {
        double baseT, alturaT, areaT;
        string baseString, alturaString;

        Console.Write ("Base del triángulo: ");
        baseString = Console.ReadLine();
        baseT = double.Parse(baseString);

        Console.Write ("Altura del triángulo: ");
        alturaString = Console.ReadLine();
        alturaT = double.Parse(alturaString);

        areaT = baseT*alturaT/2;

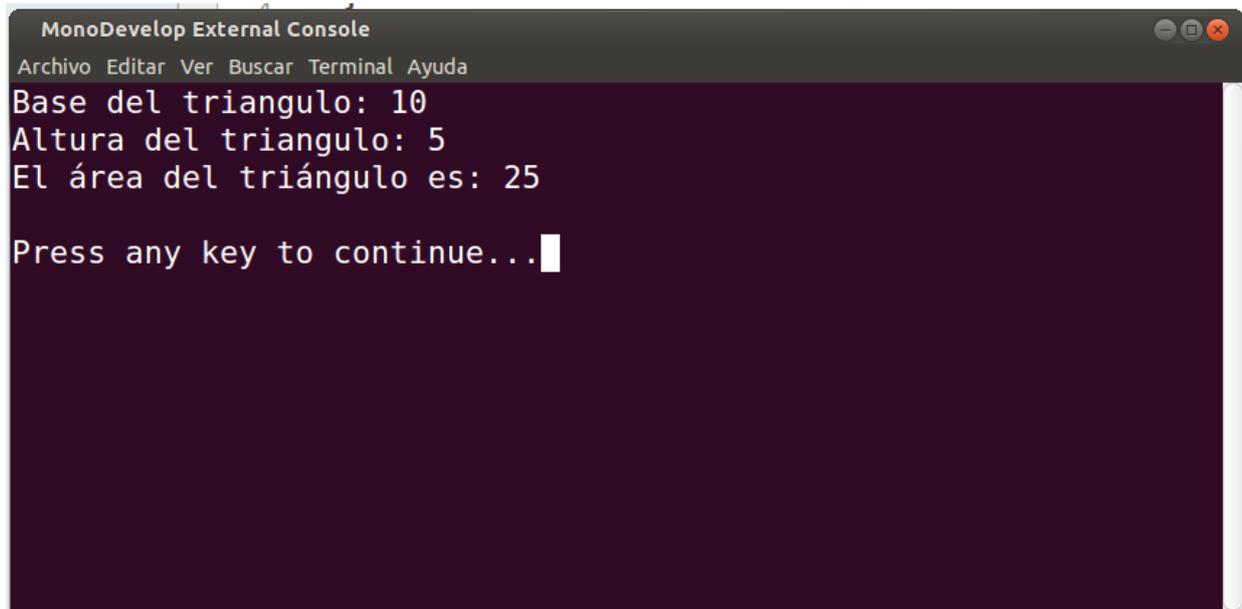
        Console.WriteLine ("El área del triángulo es: " + areaT);
    }
}
```

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

4/59

Funcionará?

Damos al *play*:

A screenshot of a terminal window titled "MonoDevelop External Console". The window has a dark background and a light-colored text. The text inside the terminal reads: "Base del triángulo: 10", "Altura del triángulo: 5", "El área del triángulo es: 25", and "Press any key to continue...". The terminal window has standard window controls (minimize, maximize, close) in the top right corner and a menu bar with "Archivo", "Editar", "Ver", "Buscar", "Terminal", and "Ayuda" in the top left corner.

```
MonoDevelop External Console
Archivo Editar Ver Buscar Terminal Ayuda
Base del triángulo: 10
Altura del triángulo: 5
El área del triángulo es: 25
Press any key to continue...
```

Parece que sí...

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

5/59

Entendiendo el programa

El fragmento principal del programa es el **bloque** de código de la función *main*:

```
double baseT, alturaT, areaT;
string baseString, alturaString;

Console.Write ("Base del triángulo: ");
baseString = Console.ReadLine();
baseT = double.Parse(baseString);

Console.Write ("Altura del triángulo: ");
alturaString = Console.ReadLine();
alturaT = double.Parse(alturaString);

areaT = baseT*alturaT/2;

Console.WriteLine ("El área del triángulo es: " + areaT);
```

Veamos paso a paso...

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

6/59

Entendiendo el programa

```
double baseT, alturaT, areaT;
```

Declara los **identificadores** `baseT`, `alturaT` y `areaT` como **variables** de **tipo** `double`

- ▶ **Declarar** es informar al compilador de la existencia de un *elemento* (entidad) en nuestro programa.
- ▶ Un **identificador** es un *nombre inventado por el programador* para identificar (nombrar) a un “elemento” (variable, función, clase, el propio programa...) del programa
- ▶ Un **tipo** es un conjunto de valores (enteros, reales, cadenas de texto, etc).
- ▶ El tipo **double** es el tipo de los *reales con doble precisión en coma flotante*. . . es un tipo que permite almacenar valores reales (3.45, 7.34927e12, etc)

- Para qué se utilizarán estas variables?

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

7/59

Entendiendo el programa

```
string baseString, alturaString;
```

- Declara los **identificadores** `baseString` y `alturaString` como **variables** de **tipo** `string`: cadenas de texto.

- Estas variables se utilizarán para almacenar la base y la altura del triángulo. . . pero como **cadenas de texto** en vez de números.

. . . pero por qué guardamos esos datos como cadenas de texto (cuando en realidad queremos tratarlas como valores numéricos)?

↪ porque a priori el usuario puede introducir como entrada cualquier secuencia de caracteres cuando se le pida la base o la altura del triángulo. Leemos como cadena de texto y luego convertimos al valor oportuno **si tiene sentido** (si no, error!)

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

8/59

Entendiendo el programa

Las tres instrucciones siguientes:

```
Console.Write ("Base del triángulo: ");
```

Escribe en pantalla Base del triángulo:

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

Lee de teclado una cadena de texto y la guarda en la variable `baseString` (asignación de un valor a una variable)

```
baseT = double.Parse(baseString);
```

Parsea (analiza) esa cadena de texto, comprueba que efectivamente representa un número (double) y guarda el valor en la variable `baseT` (otra vez, asignación)

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

9/59

Entendiendo el programa

```
Console.Write ("Altura del triángulo: ");  
alturaString = Console.ReadLine();  
alturaT = double.Parse(alturaString);
```

Análogo al código anterior.

- En este momento tendremos en memoria dos variables `baseT` y `alturaT` con dos números correspondientes a la base y la altura del triángulo proporcionado por el usuario.

Ahora podemos hacer las cuentas...

Entendiendo el programa

```
areaT = baseT*alturaT/2;
```

Esta instrucción

- ▶ calcula el área del triángulo (utilizando la fórmula conocida)
 - ▶ y almacena el valor resultante en la variable `areaT`
- Como las variables `baseT` y `alturaT` se han declarado de tipo `double`, *tiene sentido* hacer esta operación con ellas. Y *tiene sentido* guardar el resultado en `areaT` (también de tipo `double`)

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

11/59

Entendiendo el programa

```
Console.WriteLine ("El área del triángulo es: " + areaT);
```

En primer lugar se calcula un valor:

- ▶ El resultado de *concatenar* la cadena de texto (string) "El área del triángulo es: "
- ▶ con... *el resultado de convertir* el valor `areaT` de tipo `double` a cadena de texto (string).
Por ejemplo, el número 45,27 se convierte en la cadena "45,27"
En este caso la conversión de tipo (`casting`) es *implícita*.

Es decir, se concatenan dos cadenas de texto y la cadena resultante se escribe en pantalla.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

12/59

Entendiendo el programa

Algunas cosas más, para ir aprendiendo:

- ▶ Las sentencias o instrucciones se separan con ;
- ▶ Los bloques de código se encierran entre { y }
- ▶ Toda variable tiene que haber sido declarada antes de ser utilizada (si no, error)
- ▶ Una variable puede almacenar valores del tipo con el que se ha declarado. Si se intenta guardar un valor de otro tipo, error

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

13/59

Entendiendo el programa

- Al **declarar** una variable de un tipo, C# reserva una **zona de memoria** del *tamaño adecuado* para guardar valores de ese tipo.
 - ▶ (simplificando) C# tiene tipos para almacenar valores **booleanos** (*bool*: true o false), numéricos **enteros** y **reales**, **caracteres** simples (*char*) y **cadena de caracteres** (*string*).
- Internamente asocia esa variable con la **dirección de memoria** de esa zona reservada.
 - ▶ El compilador traduce los nombres de variable a direcciones concretas de memoria.
 - ▶ Pero de cara al programador (nosotros) se puede asociar **nombre de variable** con **zona de memoria**
 - ▶ Accedemos a los datos en memoria a través de variables.
- Informalmente: la ejecución de un programa en definitiva está *jugando* (mediante un algoritmo) con los valores almacenados en memoria para obtener un resultado: uno o varios valores almacenados en memoria (en determinadas variables/direcciones).

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

14/59

Entendiendo el programa

- Ya hemos visto la primera instrucción elemental: la asignación

variable = expresión;

expresión debe ser una **expresión válida del mismo tipo que el declarado para *variable***

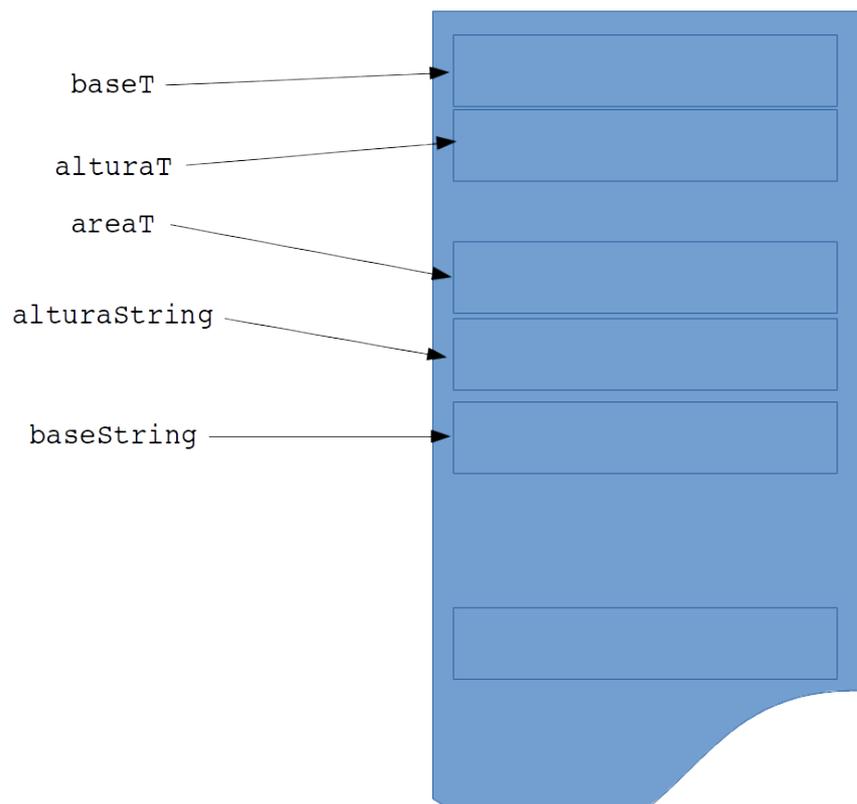
- ▶ El efecto de la asignación es:
 - ▶ **evaluar** el valor de *expresión*
 - ▶ **almacenar** ese valor en memoria, en la zona reservada para *variable*, **destruyendo** el valor previo que hubiese en esa zona de memoria.

Por eso a veces se habla de *asignación destructiva* el valor destruido es irrecuperable (no hay vuelta atrás).

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

15/59

Entendiendo el programa. La memoria



Qué efecto tiene cada asignación del programa?

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

16/59

Y el resto de código en el programa? El envoltorio

```
using System;
```

- Indica al compilador que utilizaremos nombres del **espacio de nombres System**.
 - ▶ **using** es una **palabra reservada**: no podemos utilizarla como nombre de identificador en nuestro programa.
 - ▶ Un espacio de nombres es un *contexto* donde tienen sentido algunos nombres concretos (y son únicos).
- El espacio de nombres **System** contiene la clase **Console**, que tiene métodos para escribir en pantalla (**Write**, **WriteLine**) y leer de teclado (**ReadLine**).
- Sirve para abreviar código: si no incluimos esta línea tendríamos que escribir `System.Console.Write(...)` y `System.Console.Read(...)`

17/59

Entendiendo el programa

```
class AreaTriangulo
```

- Declara la **clase AreaTriangulo**.
 - ▶ Una clase es una plantilla para agrupar datos (**atributos**) y código de programa (**métodos**) con un propósito específico.
 - ▶ En nuestro caso esta clase define un único método **Main** que realizará todo el trabajo para resolver nuestro problema.
- En C# (y en cualquier lenguaje orientado a objetos):
 - ▶ Los programas se estructuran en clases.
 - ▶ Cada clase tiene un identificador (un nombre), que por convención debe coincidir con el nombre del archivo que la contiene (la clase **AreaTriangulo** se almacenaría en el archivo **AreaTriangulo.cs**).

Por ahora, **definimos una única clase** y no profundizaremos en clases, objetos, herencia, polimorfismo...

18/59

Entendiendo el programa

```
static void Main ()
```

- `void Main ()`: prototipo del **método** (función) principal del programa:
 - ▶ `()` indica que `Main` no lleva parámetros de entrada (**input**), i.e., es una función sin argumentos.
 - ▶ Por ejemplo, la función matemática *sin* (seno) necesita un argumento, la suma dos argumentos... pero π entendida como función es una *función constante*, sin argumentos.
 - ▶ `void` indica que el método `Main` no devuelve nada (hace un trabajo y termina sin devolver nada).
 - ▶ En general, los métodos devolverán valores de salida (**output**)

El método `Main` es el *punto de entrada del programa* donde empieza la ejecución cuando arranca el programa (sin método `Main` el programa no haría nada).

- **static**: palabra reservada que indica que el método pertenece a la clase (y no al objeto)... no nos interesa por ahora.

19/59

Comentarios

Un buen programador debe **documentar** bien sus programas, en particular, **comentar su código**. Dos formas de incluir comentarios:

- ▶ En línea: los caracteres `//` convierten el resto de la línea en un comentario.
- ▶ En bloque: todo el texto delimitado entre `/*` y `*/` se considera comentario.

El compilador descarta automáticamente todos los comentarios. No tienen efecto para el programa, son solo anotaciones para el programador (documentación).

```
1 /*
2  * Ejemplo para FP
3  * Programa para el cálculo del area
4  * de un triángulo dadas la base y la altura
5  */
6
7 using System;
8
9 class MainClass
10 {
11     static void Main () // metodo principal
12     {
13         // declaracion de variables
14         double baseT, alturaT, areaT;
15         string baseString, alturaString;
16
17         // recogida de datos
```

20/59

Indentación (tabulación o sangrado de líneas)

Los *elementos* del lenguaje se separan con espacios (espacios en blanco, tabuladores, saltos del línea, ...)

En general:

- ▶ Donde hay un separador puede haber tantos como se quiera (y del tipo que se quiera)
- ▶ Podríamos escribir todo el programa en una sola línea

... pero, qué se lee mejor?

```
using System; class MainClass{static void Main
(){double baseT, alturaT, areaT; string
  baseString, alturaString; Console.Write
("Base del triángulo: "); baseString =
Console.ReadLine(); baseT =
  double.Parse(baseString);
...
}
```

```
using System;
class MainClass
{
  static void Main ()
  {
    double baseT, alturaT, areaT;
    string baseString, alturaString;

    Console.Write ("Base del triángulo: ");
    baseString = Console.ReadLine();
    baseT = double.Parse(baseString);
    ...
  }
}
```

Es buena práctica *indentar* los bloques de código y cuidar el estilo (el propio editor de texto ayudará).

Manipulando datos. Tipos y variables

Manipulado datos

Los programas **operan con datos**, **manipulan datos**. Se necesita:

- ▶ almacenarlos
- ▶ acceder a ellos (recuperarlos)
- ▶ modificarlos (jugar con ellos)

Un lenguaje de programación tiene que proporcionar mecanismos para almacenar datos (lo que *signifiquen* esos datos es cosa del programador) \rightsquigarrow variables.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

23/59

Variables y tipos

Una **variable** es una **posición de memoria identificada con un nombre** donden se pueden almacenar datos:

- ▶ Se puede asimilar con una *caja* de un tamaño concreto con un nombre en la tapa.
- ▶ El programador elige el **nombre** que le da. Es aconsejable que el nombre *refleje* lo que va a contener la variable (*base*, *altura*, *area*,...).
- ▶ También hay que decidir el **tipo de la variable**: el tamaño y la forma de la *caja*. C# proporciona una serie de tipos *predefinidos*.
- ▶ El tipo de la variable es parte es un *metadato* acerca de la variable: dato acerca del dato.

Para cada *tipo* de datos en C# existen valores **literales** (o simplemente literales): valores constantes de ese tipo.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

24/59

Declaración de variables

En general las variables se declaran del siguiente modo:

tipo variable₁, variable₂, ..., variable_n;

Donde:

- ▶ *tipo* es un tipo válido (predefinido o definido por el programador)
- ▶ *variable₁, variable₂, ..., variable_n* es un secuencia de identificadores válidos con $n \geq 1$ (se puede declarar una sola variable o tantas como queramos del mismo tipo).

Nota: el **valor inicial de una variable** declarada (pero aun no inicialida) es **indeterminado**. Para asumir algún valor para dicha variable, hay que asignarlo!

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

25/59

Tipos numéricos en C#

Se distinguen dos categorías numéricos básicos:

- ▶ **Enteros:** número de alumnos matriculados, DNI (la parte numérica), ...
 - ▶ **Reales:** area de un triángulo, importe de un producto (en euros), ...
- Para los enteros **podemos almacenar el valor exacto**, sin pérdida de información.
 - Pero para los reales esto no es posible en general:
Por ejemplo: área de un círculo de radio de radio 3

$$\pi * 3^2 = 28.2743338823081 \dots$$

¿Cómo almacenamos este área? ¿Con qué **precisión**?

No olvidemos que los datos se almacenan en forma de bits (ceros y unos) \rightsquigarrow precisión limitada para los reales.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

26/59

Tipos numéricos en C#

Los enteros y los reales se representan y almacenan de forma distinta en memoria:

- ▶ Los enteros tienen una representación bastante sencilla.
 - ▶ Los reales se representan *de manera aproximada* en forma de **coma flotante**: representación eficiente, compacta y más compleja.
- Además del conjunto de valores representados, la forma de almacenamiento es completamente distinta \rightsquigarrow la declaración de tipo sirve para reservar el hueco necesario y la forma de almacenamiento de los datos.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

27/59

Enteros en C#

Es el tipo más sencillo de representar y el más utilizado... en realidad hay varios tipos para enteros.

<code>sbyte</code>	8 bits	-128 to 127
<code>byte</code>	8 bits	0 to 255
<code>short</code>	16 bits	-32,768 to 32,767
<code>ushort</code>	16 bits	0 to 65,535
<code>int</code>	32 bits	-2,147,483,648 to 2,147,483,647
<code>uint</code>	32 bits	0 to 4,294,967,295
<code>long</code>	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>ulong</code>	64 bits	0 to 18,446,744,073,709,551,615

(tomado de *C# Programming Yellow Book*, Rob Miles)

El más utilizado de estos tipos es `int`.

Qué relación hay entre el número de bits y el rango de valores de cada tipo?

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

28/59

Operaciones con enteros

C# proporciona para los enteros:

- ▶ las operaciones aritméticas habituales: +, -, - unario, / (división entera), % (resto de la división entera)...
- ▶ operaciones de comparación: == (igualdad), <, ≤, >, ≥, != (distinto)
- ▶ también se pueden leer de teclado y escribir en pantalla

La precedencia y asociatividad de las operaciones, así como el uso de paréntesis son los habituales.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

29/59

Enteros y rangos (I)

Un comportamiento curioso: qué ocurre al ejecutar el siguiente programa?

```
int i;  
  
i = 2147483647;  
Console.WriteLine ("Valor inicial de i: " + i);  
  
i = i+1;  
Console.WriteLine ("Valor posterior de i: " + i);
```

Obtenemos en pantalla:

```
Valor inicial de i: 2147483647  
Valor posterior de i: -2147483648
```

Esto ocurre porque los tipos enteros tienen un *comportamiento cíclico*: cuando se alcanza el último valor del rango se continúa con el primero.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

30/59

Enteros y rangos (II)

El compilador puede detectar *algunos* errores de rango al utilizar literales. Por ejemplo:

```
int i;  
i = 2147483649;
```

Produce un error en compilación, al detectar que ese valor literal excede el rango de los enteros.

Pero en general (como hemos visto) los rangos no se comprueban en ejecución. Esto no produce error:

```
int i;  
i = 2147483647;  
i = 2*i;
```

Por qué se detecta el problema de rango en el primer caso pero no el segundo?

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

31/59

Valores literales de tipo entero

Los **literales** enteros son sencillamente valores enteros, i.e., secuencias de dígitos (sin punto decimal), como:

23

(Naturalmente, los literales de tipos con signo admiten signo)

Los literales se pueden asignar a variables. Por ejemplo, si *i* es una variable de tipo `int` podemos hacer:

```
i = 23 ;
```

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

32/59

Tipos para valores *reales*

En rigor no hay tipos *reales*, sino *racionales*... por qué?
Por abuso de lenguaje se habla de tipos reales y su representación en **coma flotante**: dependiendo del valor, el punto decimal *flota* en las posiciones decimales del número (ajustando el exponente).

C# proporciona tres tipos para números en coma flotante:

Tipo	tamaño (bit)s	valores	precisión
<code>float</code>	32	1.5E-45 to 3.4E48	7 dígitos
<code>double</code>	64	5.0E-324 to 1.7E308	15-16 dígitos
<code>decimal</code>	128	1.0E-28 to 7.9E28	28-29 dígitos

`decimal` no es habitual (se utiliza para calculos financieros).

Por qué no utilizamos siempre el tipo `decimal` que es el más preciso? \rightsquigarrow consume más **memoria** y **tiempo** para operar con sus valores.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

33/59

Operaciones con reales en coma flotante

- ▶ Las aritméticas habituales: $+$, $-$, $-$ unario, $/$
- ▶ Trigonométricas: seno (`Math.sin(...)`), coseno, ..., exponenciales, logaritmos, ...
- ▶ Operaciones de comparación: `==` (igualdad), $<$, \leq , $>$, \geq , `!=` (distinto)
- ▶ También se pueden leer de teclado y escribir en pantalla

La precedencia y asociatividad de las operaciones, así como el uso de paréntesis son los habituales.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

34/59

Valores literales en coma flotante

Los literales en coma flotante se expresan en *notación científica*: **coeficiente exponente**, donde el coeficiente es un real con un solo dígito en la parte entera, seguido de varios dígitos en la parte decimal, y el exponente es un entero. Por ejemplo:

$$\underbrace{-1.23456789}_{\text{coeficiente}} E \underbrace{3}_{\text{exponente}}$$

representa el número $-1.23456789 * 10^3 = -1234.56789$.

Además en C#:

- ▶ Los literales de tipo `float` llevan `f` al final, como:

`3.14f` o `-1.23456789E3f`

- ▶ Los de tipo `double` no llevan nada al final.
- ▶ Los de tipo `decimal` llevan `m`.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

35/59

Operaciones en coma flotante

La conversión de valores entre tipos en coma flotante (por ejemplo entre `float` y `double`) es *delicada* puesto que puede haber pérdida de precisión. El programador debe ser cuidadoso a la hora de hacer estas conversiones.

De hecho, en general la representación en coma flotante no es exacta (no puede serlo):

```
double unSexto, uno;
unSexto = 1.0 / 6.0;
uno = unSexto + unSexto + unSexto + unSexto + unSexto + unSexto;

Console.WriteLine( uno == 1.0);
```

La expresión `uno == 1.0` **compara** ambos valores (en este caso de tipo `double`) y devuelve `True` o `False`.

Qué escribirá este programa? **False** \rightsquigarrow errores de redondeo!!

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

36/59

Almacenando texto

Podemos almacenar:

- ▶ caracteres simples: tipo `char`
- ▶ cadenas de texto: tipo `string`

El tipo `char`:

- ▶ Una variable de tipo `char` puede almacenar un **carácter simple** (una letra, un dígito, un símbolo, ...).
- ▶ Hay distintas formas de representar los caracteres en memoria. C# utiliza el conjunto de caracteres **UNICODE**, que puede manejar hasta 65.000 caracteres distintos (además de los caracteres arábigos estándar, otros de otras lenguas).

Literales de tipo char: Se expresan encerrando el carácter entre comillas simples. Por ejemplo:

`'A'` `'x'` `'9'` `'!'` `','`
Jaime Sánchez, Sistemas Informáticos y Computación, UCM

37/59

Secuencias de escape

Cómo expresamos el carácter de comilla simple `'`? \rightsquigarrow con **secuencias de escape**:

- ▶ *escape* significa que representa un *carácter especial* (escapa de la convención de caracteres)
- ▶ el **carácter de escape** es `\`

Character	Escape Sequence name
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\\</code>	Backslash
<code>\0</code>	Null
<code>\a</code>	Alert
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical quote

Jaime Sánchez, Sistemas Informáticos y Computación, UCM
(tomado de *C# Programming Yellow Book*, Rob Miles)

38/59

Códigos de caracteres

Todo lo que el ordenador procesa internamente son números ~→ los caracteres también se manipulan como números:

- ▶ C# utiliza el estándar **Unicode** para representar los caracteres como números (es una biyección entre el conjunto de caracteres y un subrango de los naturales)
- ▶ Los caracteres se pueden expresar también mediante sus correspondientes valores numéricos, pero en **base hexadecimal y utilizando 4 posiciones numéricas** (en realidad es poco frecuente trabajar de este modo)
- ▶ Por ejemplo el valor Unicode para el carácter 'A' es 65, que en hexadecimal es 41 y se representa como '\x0041' (completando con ceros las 4 posiciones)

Que pasa si ejecutamos

```
char letraA = '\x0041'  
Console.WriteLine(letraA);      // letra "a"  
  
Console.WriteLine('\x03b1');    // letra "alpha"
```

39/59

El tipo string

- ▶ Los literales se expresan entre comillas dobles "
- ▶ Permite almacenar **cadena de texto** de cualquier longitud: desde la cadena vacía "", cadenas con un solo carácter como "a" o "7", hasta textos completos.
- ▶ Puede con saltos de línea como:

```
string s = "Este es un texto\nque contiene\nvarias líneas";
```

- ▶ También puede contener secuencias de escape:

```
string s = "\x0041BCDE\a";
```

Esto imprime en pantalla la cadena "ABCDE" y emite un beep (por la secuencia '\a').

El tipo string

Se pueden escribir cadenas en modo *verbatim* (se escriben en pantalla de modo literal, incluyendo saltos de línea:

```
string s =
@"El movimiento de las olas,
día y noche, viene del mar,
tú ves las olas, pero, ¡qué extraño!
no ves el mar.
-- Rumi --";
```

Este texto sale tal cual (literal) en pantalla al escribir la variable `s`.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

41/59

El tipo de los booleanos

El tipo `bool` (abreviatura de *boolean*) tiene dos posibles valores: `true` y `false` (cierto y falso). Es un tipo muy utilizado!

- ▶ Cuenta con las operaciones estándar:

! (negación), `&&` (and), `||` (or), `==` (igualdad), `!=` (distinto)
`&&` y `||` se dice que son de **circuito corto** (o de evaluación perezosa), i.e., evalúan el segundo argumento solo cuando es estrictamente necesario.

Por ejemplo:

`false && expresión`

no evalúa *expresión* y produce directamente `false`
(También existen los operadores `&` y `|`, no perezosos).

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

42/59

Declarar tipos adecuados

Pensar el tipo adecuado para las las variables!!

- ▶ Los enteros son preferibles a los reales en coma flotante cuando es posible: la representación de reales no es exacta y acumular error de redondeo.
- ▶ Incluso, para variables *aparentemente* reales, a veces se puede pensar alguna representación alternativa y utilizar enteros. Por ejemplo:
 - ▶ Expresar *el precio de un producto* en céntimos en vez de euros permite utilizar un entero en vez de un real y aplicar **aritmética modular** (división entera y resto) para calcular cambios.
- ▶ Otro aspecto a concretar es el **tamaño de las representaciones**:
 - ▶ En enteros, conociendo el rango de los valores a tratar, podemos ajustar el tipo y ahorrar memoria. P.e., n° de alumnos de un grupo \rightsquigarrow `ushort`, n° de DNI \rightsquigarrow `uint`,...
 - ▶ En coma flotante, debemos pensar en la precisión que necesitamos para nuestros datos, operaciones a realizar, etc.

43/59

Nombres de identificadores

Un **identificador** es un nombre elegido por el programador para representar un *elemento* de su programa, como una variable:

- ▶ tiene que comenzar con una letra
- ▶ y continuar con letras, números o es subrayado ”_”
- ▶ C# es sensible a mayúsculas y minúsculas (no es lo mismo `base` que `Base`).

La convención en C# para los identificadores es *concatenar palabras*, comenzando con mayúscula en las intermedias, como:

`precioMedio` `fistValChange` `secondValChange`

Hacer una buena elección para los nombres de identificador ayuda mucho a leer y comprender los programas.

Palabras reservadas de C#

Son palabras que ya tienen significado en el lenguaje y no se pueden utilizar como identificadores:

abstract as base bool break byte case catch char
checked class const continue decimal default delegate
do double else enum event explicit extern false
finally fixed float for foreach goto if implicit in
in int interface internal is lock long namespace new
null object operator out out override params private
protected public readonly ref return sbyte sealed
short sizeof stackalloc static string struct switch
this throw true try typeof uint ulong unchecked
unsafe ushort using virtual void volatile while

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

45/59

Asignación (revisitada)

Ya hemos visto la instrucción de [asignación](#) (pág. 17)

variable = expresión;

Una *expresión* es cualquier construcción que puede ser evaluada para producir un resultado. Puede ser un valor simple, como un literal o una variable (la evaluación es trivial, ya está evaluada), o implicar un cálculo complejo.

Es posible (y frecuente) declarar variables y asignar valores *a la vez*:

```
int i = 0, valor, j = 45 ;  
float interes = 6.23f, resultado = 2.7f;
```

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

46/59

Cambiando el tipo de datos

Es posible cambiar el tipo de los datos, para poder hacer determinadas operaciones con ellos. Dos opciones:

- ▶ ampliar la representación del valor:

```
int i = 1 ;  
float x = i ;
```

x de tipo float toma un valor int... en realidad, el entero 1 se convierte en el real 1.0 (se cambia de representación) antes de la asignación. Conversión sencilla: el tipo int "cabe" en float.

- ▶ estrechar la representación del valor:

```
float x = 1.0f ;  
int i = x ;
```

↪ error de compilación porque float no "cabe" en int (aunque el 1.0 en concreto pueda convertirse fácilmente).

```
double d = 1.5 ;  
float f = d ;
```

↪ error porque double no "cabe" en float

47/59

Casting explícito (conversión forzada)

Se puede hacer:

```
float x = 1.0 ;  
int i = (int) x ; // i toma el valor 1 (de tipo int)
```

También

```
double d = 1.5 ;  
float f = (float) d ; // i toma el valor 1.5 (de tipo float)
```

... pero en general puede perder información!!

También se puede hacer

```
int i ;  
i = (int) 1.99 ;
```

i toma el valor 1 (de tipo int) ↪ truncamiento, pérdida de información!

Más conversiones

```
float f = 1.5 ;
```

Algún problema?... No compila!!

1.5 es un *literal de tipo double*. Dos opciones para arreglarlo:

```
float f = (float) 1.5 ;
```

O

```
float f = 1.5f ;
```

Pista: ante la duda hacer casting (*no pasa nada por convertir un float en float*)

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

49/59

Cuidado con las conversiones "forzadas"

```
double d = 0.57;  
int b = (int)(d * 100);  
Console.WriteLine ("valor: " + b);
```

Escribe... **0,56!!** ... pero además este comportamiento no ocurre con otros valores como 0,59

Y también pasa en otros lenguajes como C++ o Java (es un *problema* de la representación en coma flotante).

En general, para *convertir* valores reales en enteros hay que poner cuidado. Dos formas de hacer la conversión anterior:

- ▶ Haciendo primero un redondeo (cuidado: `Round` redondea, pero devuelve `double`) y luego un casting:

```
int b = (int) Math.Round(d) ;
```

- ▶ Utilizando una función específica del lenguaje para hacer esta conversión:

```
int b = Convert.ToInt32(d) ;
```

50/59

Conversión en expresiones

Algunas evaluaciones en C#:

- ▶ $1/2$ se evalúa a 0 **Recordar: si dividimos dos enteros, obtenemos otro entero**
- ▶ $1.0/2.0$ se evalúa a 0,5
- ▶ $1/2.0$ se evalúa a 0,5
- ▶ $1.0/2$ se evalúa a 0,5
- ▶ $1/2f$ se evalúa a 0,5
- ▶ $1.0/2$ se evalúa a 0,5
- ▶ `(float) 1/2` se evalúa a 0
- ▶ `(float) (1/2)` se evalúa a 0,5
- ▶ `((float) 1)/((double) 2)` se evalúa a 0,5

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

51/59

Constantes

En los programas suele haber datos que no cambian durante la ejecución, como constantes físicas/matemáticas, valores máximos y mínimos, etc. Estos valores pueden almacenarse en constantes con la palabra reservada `const`.

Convención: las constantes se escriben con mayúsculas.

Por ejemplo:

```
const double MIN_LONG = 0.0 ;
const double MAX_LONG = 10000.0 ;
const double PI = 3.141592654 ;
const long VELOCIDAD_LUZ = 300000 ; // en km/s
const int MAX_ALUMNOS_GRUPO = 50 ;
```

Utilidad:

- ▶ es más fácil escribir `PI` que su valor decimal (no necesitamos recordar su valor cada vez que lo usemos)
- ▶ si el máximo número de alumnos por grupo se incrementa, basta con cambiar el valor de la constante y recompilar, en vez de *rastrear* ese valor en todo el programa.

52/59

Instrucciones básicas

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

53/59

El repertorio básico de instrucciones

Hasta ahora hemos visto

- ▶ declaración de variables,
- ▶ lectura de datos (de teclado),
- ▶ escritura de datos (en pantalla)
- ▶ y una instrucción: **la asignación =**

Ahora necesitamos **control de flujo**:

- ▶ *condicional*: elección en función de una **condición** (*if-else*)
- ▶ *bucles*: repetir de acuerdo a una **condición** (*while,...*)

Con esto, tendríamos un lenguaje de programación **completo**.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

54/59

Condicional: *if-else*

Recordemos el programa que calcula el área de un triángulo (pág. 5): ¿qué ocurre si el usuario introduce un número negativo como altura o base del triángulo? \rightsquigarrow El programa "funciona" (no da ningún tipo de error) pero devuelve un resultado sin sentido.

Sería conveniente controlar el rango de valores permitidos para base y altura, por ejemplo, que ambos estén entre 0 y 10.000... cómo?

Condicional:

```
if (<<condicion>>)  
    <<código para el caso "true">>  
else  
    <<código para el caso "false">>
```

condicion debe ser una expresión booleana, i.e., una expresión que se evalúa a **true** o **false**.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

55/59

Condicionales

Por ejemplo, podemos escribir:

```
if (alturaT >= 0 && alturaT <= 10000  
    && baseT >= 0 && baseT <= 10000)  
    Console.WriteLine ("El área del triángulo es: " + (baseT*alturaT/2));  
else  
    Console.WriteLine ("Los datos de entrada no son válidos");
```

La condición más simple que podemos escribir es:

```
if (true)  
    <<código>>
```

(Evidentemente podemos omitir `if (true)`)

Qué tipo de condiciones podemos escribir?...

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

56/59

Condiciones complejas: expresiones booleanas

Recordemos: los literales booleanos son `true` y `false`.

Las expresiones booleanas se construyen con:

- ▶ **operadores relacionales:**

- ▶ `==` (igualdad), `!=` (desigualdad). Aplicables prácticamente a cualquier tipo. ¿Como se evalúa `1+2 == 3`?

Cuidado con la igualdad entre reales en coma flotante!!
(problemas de redondeo)

- ▶ operadores de orden `<`, `<=`, `>`, `>=`. Aplicables a tipos numéricos, `char`, ...

`!(2 == 3)` es lo mismo que `2 != 3`

- ▶ **conectivas lógicas:** `!` (negación), `&&` (and), `||` (or).

Aplicables a expresiones booleanas. Evaluación de circuito corto.

Notas: Las condiciones complejas se pueden descomponer en otras más simples y luego combinarlas. Conocer las precedencias de los operadores y las conectivas permite reducir el número de paréntesis, pero podemos poner *paréntesis extra*.

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

57/59

Bloques de código

El condicional que hemos visto tiene la forma

```
if (<<condicion>>)
  <<código para el caso "true">>
else
  <<código para el caso "false">>
```

... y si en las secciones de código necesitamos más de una instrucción? \rightsquigarrow **bloque de código** delimitado con `{` y `}`

```
if (<<condicion>>)
{
  areaT = baseT*alturaT/2;
  Console.WriteLine ("El área del triángulo es: " + areaT);
}
else
{
  Console.WriteLine ("Los datos de entrada no son válidos");
  Console.WriteLine ("Intoduzca valores en el rango adecuado");
}
```

Jaime Sánchez. Sistemas Informáticos y Computación, UCM

58/59

Bloques de código

- ▶ Dentro de un bloque se pueden escribir tantas instrucciones como se quiera.
- ▶ Se pueden anidar los bloques (unos dentro de otros) tanto como queramos.
- ▶ Convención: para facilitar la lectura cada vez que se abre un bloque, el código se indenta un *hueco* (tabulador) más a la derecha, i.e., el margen izquierdo se desplaza a la derecha.