

Tema 2. Elementos del lenguaje

1. Introducción a los tipos de datos básicos.	1
2. Constantes y Variables	3
3. Operaciones Primitivas	10
4. Expresiones y Sentencias en C#.....	13
5. Manejo de Errores de Ejecución. Try - Catch.....	15

1. Introducción a los tipos de datos básicos.

El **tipo de datos** de un elemento de programación hace referencia a la clase de datos que puede contener y a cómo se almacenan los datos. El sistema de tipos comunes (*Common Type System – CTS*) define los tipos de datos que soporta el Common Language Runtime.

Visual C# .NET utiliza tipos de datos que se corresponden directamente con los tipos de datos del sistema de tipos comunes.

El sistema de tipos comunes tiene una gran importancia en la creación de aplicaciones para la plataforma Microsoft .NET. Hace posible que un desarrollador pueda crear un proyecto en Visual Basic .NET e integrarlo con un componente creado por otro desarrollador en Microsoft Visual C# y una función escrita por un tercer desarrollador en otro lenguaje compatible con .NET.

Tipos de datos en Visual C#

- **Enteros.**

El tipo de datos enteros define a los números enteros positivos y negativos. Se representan en Visual C# mediante el tipo de datos *int*.

- **Reales.**

Representa el conjunto de los números reales o decimales.
Las palabras reservadas para definir un número real pueden son:

float: número real de 32 bits.
double: número real de 64 bits.

- **Lógico o booleano.**

Es un tipo especial de datos, muy utilizado en programación y que solamente puede tomar el valor cierto o falso.

Se definen con la palabra reservada **bool**, y solo pueden tomar el valor **true** o **false**.

- **Carácter.**

Un dato de tipo carácter contiene un solo carácter Unicode.

Se definen con la palabra reservada **char** y representa a los caracteres contenidos en el código Unicode.

- **Cadena de caracteres o string.**

Se definen con la palabra reservada **string**, y se representan entre comillas dobles.

Un ejemplo de valor de tipo string podría ser el nombre de una persona:

“David González”

Ejemplo. ¿Qué tipos de datos se podrían utilizar para los siguientes conceptos?:

- **Edad:** Entero, int.
- **Código Postal:** Cadena de caracteres o string (Es un número no operable: 03005).
- **D.N.I.:** Cadena de caracteres o string
- **Altura:** Entero (int) o Real (double).
- **Sexo:** Carácter (char)
- **¿Casado?:** Lógico(bool)
- **I.V.A. a aplicar:** Numérico Real (double) o Entero (int).

2. Constantes y Variables

Lógicamente, cualquier programa informático necesita almacenar y tratar la información. Para ello se utilizan las **constantes** y las **variables**.

Las **constantes** serán datos que mantendrán el mismo valor a lo largo de la ejecución de todo el programa

Las **variables** nos servirán para almacenar datos cuyo valor podrá variar durante la ejecución del programa.

Tanto las **constantes** como las **variables** tienen una serie de atributos:

- Nombre o **identificador**. Servirá para identificarlo y referenciarlo en el programa. El nombre o identificador de la variable o constante debe describir la información que va a contener el dato.

Las reglas para construir un identificador en Visual C# son las siguientes:

- Están formados por letras (alfabeto inglés), dígitos y el carácter de subrayado.
- El primer carácter debe ser una letra.
- No debe contener caracteres blancos ni símbolos.
- No utilizar palabras clave como **int**, **for** o **date**.

Es conveniente que el identificador describa la función del elemento ya sea una variable, una constante, un subprograma... De esta forma al ver el nombre del elemento sabremos para qué se utilizar en nuestro programa.

Ejemplos de identificadores válidos en Visual C#:

num, nombre, Apellidos, PI, empresa_1, CosteInicial, MAX, unidad_1_apartado_2.

- **Valor**. Será el dato que guarda la constante o variable. Para una constante será siempre el mismo, pero en el caso de una variable tal y como su nombre indica podrá variar su valor durante la ejecución del programa.
- **Tipo de datos**. El tipo de una variable o constante representará el conjunto de valores que puede tomar el elemento y determinará además el conjunto de operaciones que se pueden realizar con él.

En el apartado anterior vimos cuáles son los tipos de datos básicos en Visual C# y en temas posteriores ampliaremos los tipos de datos.

VARIABLES

Vamos a introducir brevemente la forma de declarar variables y explicaremos algunos conceptos. En temas posteriores se irán ampliando estos conceptos sobre variables.

Declaramos una variable para especificar su **nombre** y sus características, indicando de qué **tipo** es la misma.

La sintaxis para declarar una variable será la siguiente:

tipo nombre;

Ejemplo de declaración de variables:

```
int num;  
int i, j, k;  
int multiplicando, multiplicador, resultado;  
float num_real;  
double num_doble_precision_1, num_doble_precision_2;  
string nombre;
```

En este pequeño ejemplo de declaración de variables vemos que primero se declara una variable llamada **num** de tipo entero, a continuación, tres variables **i, j, k** de tipo entero, después otras tres variables de tipo entero: **multiplicando, multiplicador y resultado**, seguimos con números de coma flotante de doble precisión y una cadena de caracteres.

Se puede dar el valor a la variable en el momento de declararla:

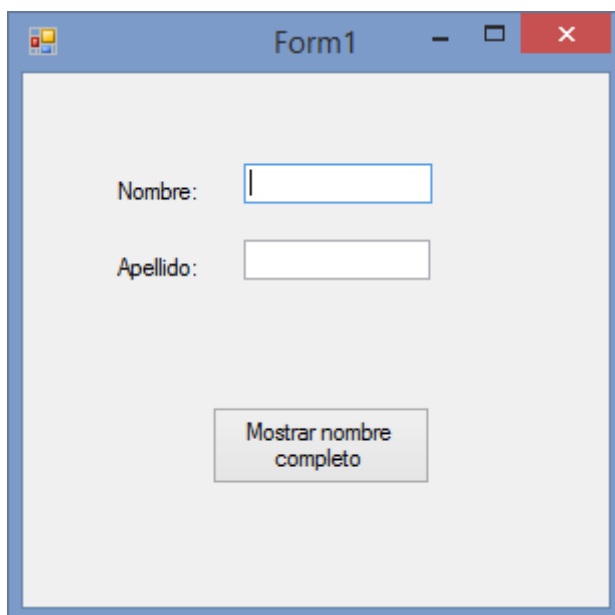
```
int num = 10;
```

Las variables en Visual C# se pueden declarar dentro de las funciones (variables locales), fuera de cualquier función, en la clase de nuestro formulario (variables de módulo). En temas posteriores recordaremos y entenderemos estos conceptos.

Nosotros, de momento, **declararemos las variables en los eventos** de los componentes de nuestro formulario. Por ejemplo en el evento click de un botón.

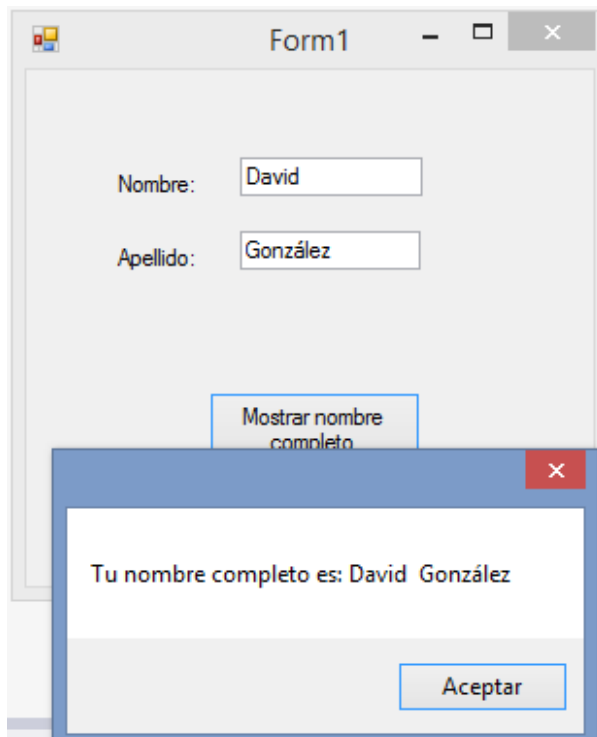
Ejemplo variables string:

Vamos a ver a continuación un ejemplo en el que hagamos un programa en el que vamos a recoger valores en **variables de tipo string**. En concreto haremos un formulario en el que se introducirá nuestro nombre y nuestro apellido:



The image shows a standard Windows application window titled 'Form1'. Inside the window, there are two text input fields. The first is labeled 'Nombre:' and the second is labeled 'Apellido:'. Below these two fields, centered, is a button with the text 'Mostrar nombre completo'.

Al pulsar el botón mostraremos un mensaje con el nombre completo:



El código en el evento click del botón será el siguiente:

```
private void button1_Click(object sender, EventArgs e)
{
    string nombre, apellido;
    string nombreCompleto;

    // Aquí recogemos el nombre del textbox correspondiente
    nombre = txtNombre.Text;
    // Aquí el apellido
    apellido = txtApellido.Text;

    // Aquí montamos la frase final, utilizando la concatenación.
    nombreCompleto = "Tu nombre completo es: " + nombre + " " + apellido;

    MessageBox.Show(nombreCompleto);
}
```

Las líneas que aparecen en verde con // al principio son comentarios. Los comentarios son líneas que no tienen ninguna influencia sobre el código del programa. Sirven para “comentar” o aclarar puntos de nuestro código.

Si quisiéramos comentar varias líneas de código lo podemos hacer con los caracteres /* y */:

```
/* De esta manera
 * podemos
 * comentar varias líneas */
```

Como se puede ver en nuestro código hemos declarado tres variables de tipo string. En las variables **nombre** y **apellido** hemos introducido el texto de los textbox.

En la variable **nombreCompleto** hemos formado la frase a mostrar utilizando el operador `+` que nos permite concatenar dos o más strings. (Más adelante estudiaremos el concepto de operador).

Por último con `MessageBox` lo hemos mostrado por pantalla.

Ejemplo variables enteras:

En esta ocasión vamos a ver un ejemplo con variables enteras, es decir, vamos a empezar a trabajar con números, de manera que podamos recoger un valor numérico y mostrarlo por pantalla.

En concreto vamos a ver un ejemplo en el que damos valor a una variable entera (`int`) y al pulsar un botón mostramos por pantalla su valor con un `messagebox`.

El código quedaría algo así:

```
private void bMostrarEntero_Click(object sender, EventArgs e)
{
    // Declaramos la variable de tipo entero
    int numero;

    // Le damos valor
    numero = 125;

    // La mostramos por pantalla.
    MessageBox.Show(numero);
}
```

Si intentamos **compilar y ejecutar este código** nos surge un error de compilación en el `MessageBox`. Esto es así porque C# es un **lenguaje fuertemente tipado**, lo que significa que no realiza conversiones automáticas entre variables de distinto tipo.

Como la variable *numero* es de tipo entero, y lo que espera *MessageBox.Show* es un string, necesitamos convertir el entero en string:

```
MessageBox.Show(numero.ToString());
```

`ToString` nos permite convertir el entero en una cadena de caracteres o string (**atención a los paréntesis**).

Ejemplo con variables de tipo real.

Vamos a hacer ahora un ejemplo similar al anterior, pero en el que damos valor a una variable de tipo double y a otra de tipo float.

```
private void bMostrarReales_Click(object sender, EventArgs e)
{
    // Declaramos las variables.
    double numeroDouble;
    float numeroFloat;

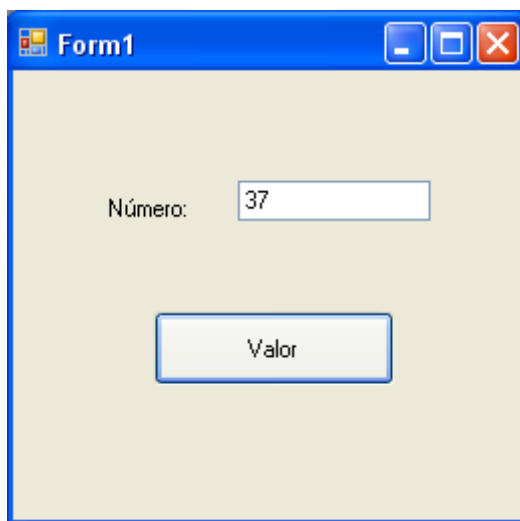
    // Le damos valor
    numeroDouble = 223.25;
    numeroFloat = 3.1416F;

    // Las mostramos por pantalla.
    MessageBox.Show(numeroDouble.ToString());
    MessageBox.Show(numeroFloat.ToString());
}
```

Fijar la atención en la variable de tipo float. Al darle valor debemos poner una F al final del mismo para indicar que es de tipo float, no double.

Ejemplo obtención de valores numéricos de textbox.

En el ejemplo que vamos a ver a continuación vamos a obtener valores numéricos a partir de un textbox:



El código para el botón que nos permite mostrar el valor introducido en el textbox podría ser algo así:

```
private void bValor_Click(object sender, EventArgs e)
{
    int numero;

    numero = textBox1.Text;

    MessageBox.Show(numero.ToString());
}
```

Sin embargo, nos da un error de compilación en la línea:

```
numero = textBox1.Text;
```

Esto ocurre porque un `textBox.Text` devuelve, lógicamente, un texto (string) y, al ser C# un lenguaje fuertemente tipado, necesitamos **convertir ése texto en el tipo de la variable** a la cual queremos dar valor.

Esto se hace mediante el método `Parse`. Quedaría de la siguiente forma:

```
numero = int.Parse(textBox1.Text);
```

Si quisieramos obtener valores de tipo `float` o tipo `double` también tendríamos que convertir el texto:

```
float numeroFloat;  
  
numeroFloat = float.Parse(textBox1.Text);
```

o bien con `double`:

```
double numeroDoble;  
  
numeroDoble = double.Parse(textBox1.Text);
```


CONSTANTES

Como hemos dicho antes, las constantes son datos cuyo valor no puede ser cambiado a lo largo del programa.

Las constantes se pueden utilizar en un programa de forma literal, o bien se puede definir una constante con un identificador que nos indique qué valor representa la constante.

Esto sirve para dos cosas. La primera, nos da **mayor claridad** a la hora de leer el programa, y la segunda, si utilizáramos la constante **en varios sitios del programa** y en el futuro cambiara su valor, únicamente necesitaríamos cambiar ese valor en un sólo sitio.

Por ejemplo, imaginemos que en un programa de gestión comercial tuviéramos que aplicar el IVA cuyo valor sea un 21%. Si nosotros definimos una **constante** llamada **kIVA** y la utilizamos en el programa, además de dar claridad, si en el futuro el IVA cambiara por ejemplo de un 21 a un 23% no tendríamos que buscar todos los sitios donde se utilice 21 sino cambiar el valor de la constante.

Las constantes se declaran de manera similar a como se declaran las variables pero utilizando las palabra **const** delante, y además al declararla debemos darle un valor. Es una buena práctica poner el nombre de las constantes en mayúsculas:

`const Tipo nombreConstante = Valor;`

Ejemplo de definición de constantes:

```
const double IVA = 0.21;  
const double PI = 3.1416;  
const int NUMERO = 18;  
const string FRASE = "Bienvenido al módulo de Programación.";
```

Las dos primeras líneas definen dos constantes de tipo real, la siguiente (NUMERO) de tipo entero y por último FRASE será de tipo cadena de caracteres.

3. Operaciones Primitivas

El desarrollo de un programa requerirá muchas veces de la capacidad de efectuar operaciones con los datos, cada tipo de datos tendrá un conjunto de operadores asociados. Los operadores se clasifican en:

- **Aritméticos:**

Servirán para realizar operaciones entre datos de tipo numérico dando como resultado otro valor también numérico. Los operadores aritméticos en Visual C# serán:

+ suma

- resta

* producto

/ división

% módulo o resto de la división entera

Los operadores +, -, *, / no precisan mayor explicación. Realizan las operaciones matemáticas equivalentes.

El operador % es el operador módulo o resto de la división entera. Por ejemplo:

5 % 2 devuelve el valor 1

7 % 4 devuelve el valor 3.

[Referencia a los operadores aritméticos en C# \(Microsoft\)](#)

- **Relacionales**

Se utilizan para comparar expresiones numéricas y devuelven siempre un valor **lógico** o **booleano**.

Las operaciones relacionales darán como resultado **false** cuando la operación sea falsa (por ejemplo $3 > 5$) y darán **true** cuando la operación sea cierta (por ejemplo $3 \leq 5$).

Los operadores relacionales son <, >, == (igual), <=, >=, != (distinto).

Podrían ser posibles expresiones relacionales:

$5 > 3$ (* Devuelve TRUE *)

$5 + 1 \geq 12 - 4$ (* Devuelve FALSE *)

- **Lógicos**

El tipo lógico tiene una serie de operadores que aplicados a éstos dan como resultado otro valor lógico.

Los operadores lógicos son ! (Not o Negación), && (and o conjunción), || (or o disyunción).

Su funcionamiento está determinado por las llamadas **tablas de verdad**:

Operador **!**:

A	! a
True	false
False	true

Operador **&&** (AND)

A	B	a && b
False	false	false
False	true	false
True	false	false
True	true	true

Operador **||** (OR)

A	B	a b
false	false	false
False	true	true
True	false	true
True	true	true

Como resumen de las tablas de verdad podemos decir que el operador **!** (**not**) dará como resultado lo contrario, el operador **&&** (**and**) dará falso cuando uno o más de los operadores sea falso, y el operador **||** (**or**) dará cierto cuando uno o más de los operadores sea cierto.

- **Alfanuméricos**

+ Concatenación de cadenas de caracteres.

Por ejemplo: "David" + "González" dará como resultado "DavidGonzález"

- **Precedencia.**

Los operadores en Visual C#, al igual que ocurre con los operadores matemáticos, tienen un orden de precedencia.

1. (,), [,]
2. !
3. *, /, %
4. +, -
5. <, >, <=, >=
6. ==, <>
7. &&
8. ||

Cuando encontremos operadores con igual precedencia se realizarán las operaciones de izquierda a derecha.

Los paréntesis permiten alterar el orden de preferencia de los operadores.

Ej:

$2 + 3 * 5$ dará como resultado 17, sin embargo

$(2 + 3) * 5$ dará como resultado 25

[Referencia a la prioridad de operadores \(Microsoft\)](#)

4. Expresiones y Sentencias en C#

Una **expresión** será un dato (una constante o variable) o una lista de datos (operandos) unidos mediante símbolos (operadores), que se evalúa (se opera) y de la cual se obtiene un resultado. El tipo del resultado dependerá de los operandos y de los operadores. Se pueden ver de forma similar a las operaciones matemáticas.

Una **sentencia** es una línea o varias de código que nos permiten ejecutar algo en nuestro programa...

Por ejemplo, cuando escribimos la línea:

```
MessageBox.Show("Hola Mundo. Este es mi primer programa en C#");
```

es una sentencia que se ejecuta y permite mostrar por pantalla un mensaje.

Dentro de las sentencias en C# vamos a estudiar la **asignación**.

La asignación nos permitirá **dar un valor a una variable**. En Visual C# se utiliza el símbolo = para efectuar una asignación. En la parte izquierda del símbolo de asignación se pone la variable a la cual vamos a dar valor, y en la parte derecha se coloca la expresión que da valor a esta variable. La parte derecha podrá ser un valor o cualquier expresión válida.

```
variable= valor;
```

```
variable= expresión;
```

Es importante no confundir el operador = (asignación) con el operador lógico == (comparación).

Ejemplo de sentencias en Visual C#.

En la siguiente página tenemos un ejemplo en el que vamos a declarar unas variables y vamos a ir asignándoles valores y expresiones.

Al lado se irá poniendo entre comentarios qué valores van tomando las variables.

Los comentarios son texto detrás del símbolo //. Recordar que un comentario no tiene ningún efecto sobre el código del programa, simplemente sirve para aclarar detalles del código.

Notar que todas las sentencias en C# terminan con **un punto y coma (;)**.

```
const int CONSTANTE = 100;

int numa, numb, numc;
double reala, realb, realc;
bool test;
/* Hemos declarado tres variables de tipo entero, tres de tipo real
 * y una de tipo lógico */

numa = CONSTANTE;      // numa toma el valor 100
numb = 200;             // numb toma el valor 200
numc = numa + numb;     // numc toma el valor 300
numc = 300 / numa + numb; // numc toma el valor 203
numc = 300 / (numa + numb); // numc toma el valor 1

numc = numc + 1; // Incremento numc pasa a valer 2
numc++; // Esta expresión es equivalente a numc = numc + 1 Pasa a valer 3

test = numa == CONSTANTE; // test toma el valor TRUE
test = (numa == CONSTANTE) || (numb > 300); // test vale TRUE
test = (numa == 100) && (numb > 300); // test toma el valor FALSE
test = (numa > 50) || (numb > 50); // test toma el valor TRUE
test = (numa < 50) || (numb > 50); // test toma el valor TRUE
test = (numa < 50) || (numb < 50); // test toma el valor FALSE

test = (numa > 50) || (numb < 50) && (5 < 7); // test toma el valor TRUE
test = ((numa > 50) || (numb < 50)) && (5 < 7); // test toma el valor TRUE
test = ((numa > 50) || (numb < 50)) && (5 > 7); // test toma el valor FALSE

reala = 100;           // reala toma el valor 100.0
realb = 200.0;         // realb toma el valor 200.0
realc = reala / realb; // realc toma el valor 0.5

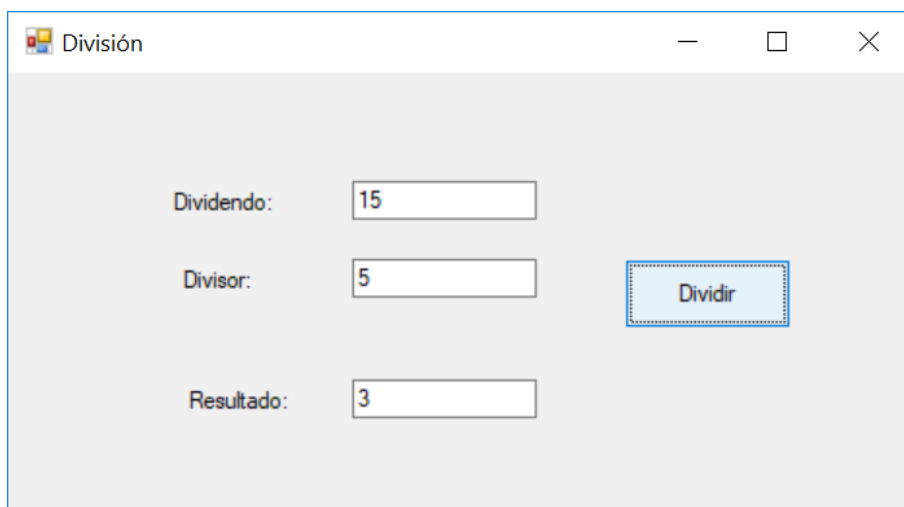
numa = 5;
numb = 2;
reala = numa / numb; // reala toma el valor 2. Es división entera
reala = (double)numa / (double)numb; // reala toma el valor 2.5.
//Hacemos(double) para que la división sea real

realc = realc + numa; // realc toma el valor 5.5
realc = realc + 100; // numc toma el valor 105.5
reala = 1.5;
realc = reala * 2; // realc toma el valor 3.0
```

5. Manejo de Errores de Ejecución. Try - Catch

Al empezar a trabajar con los ejemplos o ejercicios del tema 2 seguramente os habréis encontrado con errores de ejecución, bien al introducir datos, o incluso al operar con ellos.

Imaginemos que vamos a realizar un programa en el que queremos realizar una división entre dos números, siendo el usuario el que introduzca el dividendo y el divisor. Nuestra aplicación podría tener un aspecto similar al siguiente:

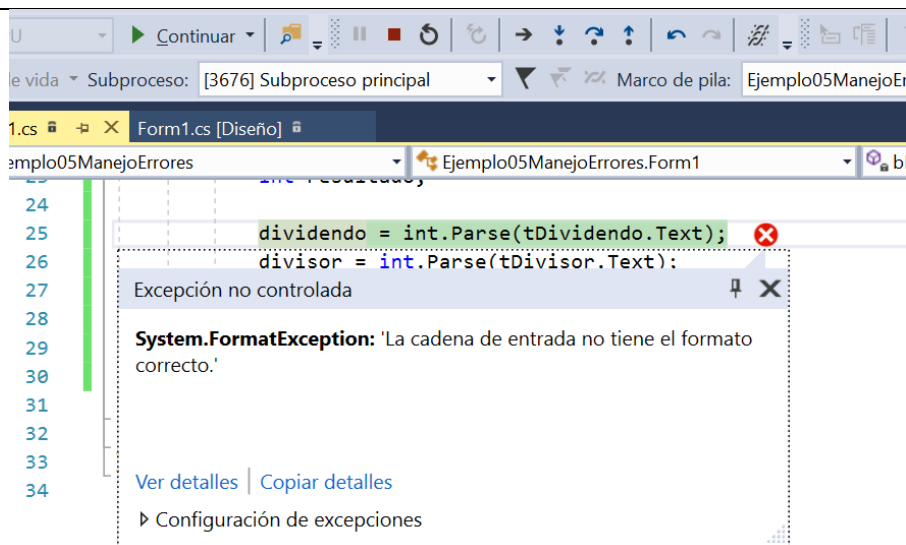


El código para el botón, de acuerdo a lo que hemos estudiado en el tema, podría ser el siguiente:

```
int dividendo, divisor;  
int resultado;  
  
dividendo = int.Parse(txtDividendo.Text);  
divisor = int.Parse(txtDivisor.Text);  
  
resultado = dividendo / divisor;  
  
tResultado.Text = resultado.ToString();
```

Pero, ¿qué ocurre si los datos que nos introduce el usuario no son “correctos”, por ejemplo, no mete un valor en alguno de los textBox de entrada, o mete un valor no numérico.

En ese caso, nuestro programa produce un error o excepción, porque no se puede ejecutar normalmente. Termina la ejecución y en el entorno de trabajo nos aparece un mensaje indicando cuál es la excepción que ha ocurrido:



Para evitar esta excepción podemos utilizar un bloque try-catch.

Cuando **escribimos código dentro de un bloque try**, se intenta ejecutar todas las sentencias dentro del bloque try, y si ninguna de las sentencias genera una excepción, se ejecutará completo.

Si ocurre una excepción el flujo de ejecución saltará fuera del bloque **try** a otro bloque de código que atraparará y manejará la excepción, el bloque **catch**.

Así quedaría nuestro código evitando que se produzca el error de ejecución:

```
int dividendo, divisor;
int resultado;

try
{
    dividendo = int.Parse(txtDividendo.Text);
    divisor = int.Parse(txtDivisor.Text);

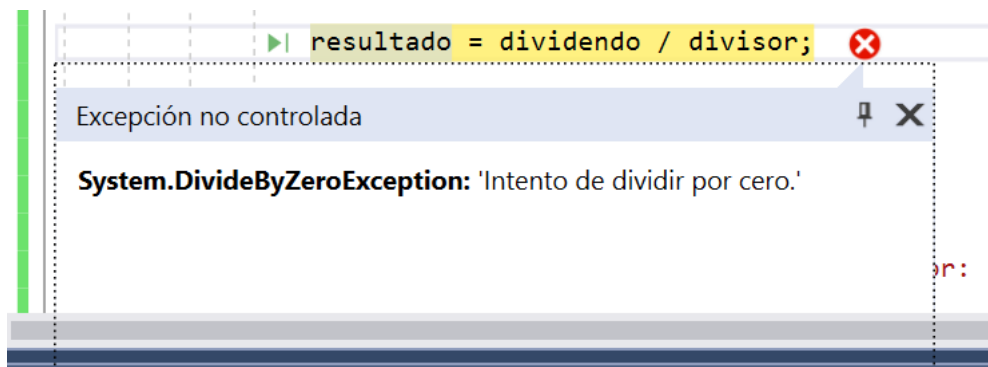
    resultado = dividendo / divisor;

    txtResultado.Text = resultado.ToString();
}
catch (FormatException fEx)
{
    MessageBox.Show("Se ha producido el error: " + fEx.Message);
}
```

Debemos tener en cuenta que podemos tener distintos tipos de excepciones según los programas y que solo evitamos aquellas que hayamos escrito el bloque catch.

Por ejemplo, ¿qué ocurre si el usuario nos mete valores numéricos correctos, pero en el divisor nos introduce un 0?

Pues en ese caso el código anterior no evitaría la excepción y tendríamos el siguiente error:



Esto lo podemos evitar añadiendo otro bloque catch que maneje esa excepción (fijáos que para saber qué excepción ocurre, no tenemos más que fijarnos en el error que aparece DivideByZeroException y utilizarla luego en nuestro código).

El código que evitaría ese error quedaría de la siguiente forma:

```
int dividendo, divisor;
int resultado;

try
{
    dividendo = int.Parse(tDividendo.Text);
    divisor = int.Parse(tDivisor.Text);

    resultado = dividendo / divisor;

    tResultado.Text = resultado.ToString();
}
catch (FormatException fEx)
{
    MessageBox.Show("Se ha producido el error: " + fEx.Message);
}
catch (DivideByZeroException oEx)
{
    MessageBox.Show(oEx.Message);
}
```

Resultado en la ejecución:

