2.1. Estructuras alternativas

2.1.1. if

La primera construcción que emplearemos para comprobar si se cumple una condición será "**si ... entonces ...**". Su formato es

```
if (condición) sentencia;
```

Es decir, debe empezar con la palabra "if", la condición se debe indicar entre paréntesis y a continuación se detallará la orden que hay que realizar en caso de cumplirse esa condición, terminando con un punto y coma.

Vamos a verlo con un ejemplo:

```
// Ejemplo_02_01_01a.cs
// Condiciones con if
// Introducción a C#

using System;

public class Ejemplo_02_01_01a
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero>0) Console.WriteLine("El número es positivo.");
    }
}
```

Este programa pide un número al usuario. Si es positivo (mayor que 0), escribe en pantalla "El número es positivo."; si es negativo o cero, no hace nada.

Como se ve en el ejemplo, para comprobar si un valor numérico es mayor que otro, usamos el símbolo ">". Para ver si dos valores son iguales, usaremos dos símbolos de "igual": if (numero==0). Las demás posibilidades las veremos dentro de muy poco. En todos los casos, la condición que queremos comprobar deberá indicarse entre paréntesis.

Este programa comienza por un comentario algo más detallado que los de los ejemplos anteriores, que nos recuerda de qué se trata. Como nuestros "fuentes" irán siendo cada vez más complejos, a partir de ahora incluiremos comentarios que nos permitan recordar de un vistazo qué pretendíamos hacer.

Si la orden "if" es larga, se puede partir en dos líneas para que resulte más legible:

```
        if (numero>0)
            Console.WriteLine("El número es positivo.");
```

Ejercicios propuestos:

Ejercicio propuesto 2.1.1.1: *Crea un programa que pida al usuario un número entero y diga si es par (pista: habrá que comprobar si el resto que se obtiene al dividir entre dos es cero: if (x % 2 == 0) ....*

Ejercicio propuesto 2.1.1.2: *Crea un programa que pida al usuario dos números enteros y diga cuál es el mayor de ellos.*

Ejercicio propuesto 2.1.1.3: *Crea un programa que pida al usuario dos números enteros y diga si el primero es múltiplo del segundo (pista: igual que antes, habrá que ver si el resto de la división es cero: a % b == 0).*

2.1.2. if y sentencias compuestas

Habíamos dicho que el formato básico de "if" es if (condición) sentencia; Esa "sentencia" que se ejecuta si se cumple la condición puede ser una sentencia simple o una compuesta. Las sentencias compuestas se forman agrupando varias sentencias simples entre llaves ({ y }), como en este ejemplo:

```
// Ejemplo_02_01_02a.cs
// Condiciones con if (2): Sentencias compuestas
// Introducción a C#

using System;

public class Ejemplo_02_01_02a
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero > 0)
        {
            Console.WriteLine("El número es positivo.");
            Console.WriteLine("Recuerde que también puede usar negativos.");
        }  // Aquí acaba el "if"
    }      // Aquí acaba "Main"
}          // Aquí acaba "Ejemplo
```

En este caso, si el número es positivo, se hacen dos cosas: escribir un texto y luego... ¡escribir otro! (no es gran cosa; más adelante iremos encontrando casos en lo que necesitemos hacer cosas "más serias" dentro de una sentencia compuesta).

Como se ve en este ejemplo, cada nuevo "bloque" se suele escribir un poco más a la derecha que los anteriores, para que sea fácil ver dónde comienza y termina cada sección de un programa. Por ejemplo, el contenido de " Ejemplo_02_01_02a " está un poco más a la derecha que la cabecera "public class Ejemplo_02_01_02a ", y el contenido de "Main" algo más a la derecha, y la sentencia compuesta que se debe realizar si se cumple la condición del "if" está aún más a la derecha. Este "**sangrado**" del texto se suele llamar "**escritura indentada**". Un tamaño habitual para el sangrado es de 4 espacios, aunque en este texto en algunas ocasiones usaremos sólo dos espacios, para que fuentes más complejos quepan entre los márgenes del papel.

Ejercicios propuestos:

Ejercicio propuesto 2.1.2.1: *Crea un programa que pida al usuario un número entero. Si es múltiplo de 10, informará al usuario y pedirá un segundo número, para decir a continuación si este segundo número también es múltiplo de 10.*

2.1.3. Operadores relacionales: <, <=, >, >=, ==, !=

Hemos visto que el símbolo ">" es el que se usa para comprobar si un número es mayor que otro. El símbolo de "menor que" también es sencillo, pero los demás son un poco menos evidentes, así que vamos a verlos:

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (distinto de)

Así, un ejemplo, que diga si un número no es cero sería:

```
// Ejemplo_02_01_03a.cs
// Condiciones con if (3): "distinto de"
// Introducción a C#

using System;

public class Ejemplo_02_01_03a
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero != 0)
            Console.WriteLine("El número no es cero.");
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.1.3.1: *Crea un programa que multiplique dos números enteros de la siguiente forma: pedirá al usuario un primer número entero. Si el número que se teclee es 0, escribirá en pantalla "El producto de 0 por cualquier número es 0". Si se ha tecleado un número distinto de cero, se pedirá al usuario un segundo número y se mostrará el producto de ambos.*

Ejercicio propuesto 2.1.3.2: *Crea un programa que pida al usuario dos números enteros. Si el segundo no es cero, mostrará el resultado de dividir el primero entre el segundo. Por el contrario, si el segundo número es cero, escribirá "Error: No se puede dividir entre cero".*

2.1.4. if-else

Podemos indicar lo que queremos que ocurra en caso de que no se cumpla la condición, usando la orden "else" (en caso contrario), así:

```
// Ejemplo_02_01_04a.cs
// Condiciones con if (4): caso contrario ("else")
// Introducción a C#

using System;

public class Ejemplo_02_01_04a
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero > 0)
            Console.WriteLine("El número es positivo.");
        else
            Console.WriteLine("El número es cero o negativo.");
    }
}
```

Podríamos intentar evitar el uso de "else" si utilizamos un "if" a continuación de otro, así:

```
// Ejemplo_02_01_04b.cs
// Condiciones con if (5): caso contrario, sin "else"
// Introducción a C#

using System;

public class Ejemplo_02_01_04b
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
```

```
        if (numero > 0)
            Console.WriteLine("El número es positivo.");

        if (numero <= 0)
            Console.WriteLine("El número es cero o negativo.");
    }
}
```

Pero el comportamiento **no es el mismo**: en el primer caso (ejemplo 02_01_04a) se mira si el valor es positivo; si no lo es, se pasa a la segunda orden, pero si lo es, el programa ya ha terminado. En el segundo caso (ejemplo 02_01_04b), aunque el número sea positivo, se vuelve a realizar la segunda comprobación para ver si es negativo o cero, por lo que el programa es algo más lento.

Podemos enlazar varios "if" usando "else", para decir "si no se cumple esta condición, mira a ver si se cumple esta otra":

```
// Ejemplo_02_01_04c.cs
// Condiciones con if (6): condiciones encadenadas o anidadas
// Introducción a C#

using System;

public class Ejemplo_02_01_04c
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());

        if (numero > 0)
            Console.WriteLine("El número es positivo.");
        else
            if (numero < 0)
                Console.WriteLine("El número es negativo.");
            else
                Console.WriteLine("El número es cero.");
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.1.4.1: *Mejora la solución al ejercicio 2.1.3.1, usando "else".*

Ejercicio propuesto 2.1.4.2: *Mejora la solución al ejercicio 2.1.3.2, usando "else".*

2.1.5. Operadores lógicos: &&, ||, !

Las condiciones se puede encadenar con "y", "o", "no". Por ejemplo, una partida de un juego puede acabar si nos quedamos sin vidas o si superamos el último nivel. Y podemos avanzar al nivel siguiente si hemos llegado hasta la puerta y hemos encontrado la llave. O deberemos volver a pedir una contraseña si no es correcta y no hemos agotado los intentos.

Esos operadores se indican de la siguiente forma

Operador	Significado
&&	Y
||	O
!	No

De modo que, ya con la sintaxis de C#, podremos escribir cosas como

```
if ((opcion==1) && (usuario==2))   ...
if ((opcion==1) || (opcion==3))    ...
if ((!(opcion==opcCorrecta)) || (tecla==ESC))    ...
```

Así, un programa que dijera si dos números introducidos por el usuario son cero, podría ser:

```
// Ejemplo_02_01_05a.cs
// Condiciones con if enlazadas con &&
// Introducción a C#

using System;

public class Ejemplo_02_01_05a
{
    public static void Main()
    {
        int n1, n2;

        Console.Write("Introduce un número: ");
        n1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Introduce otro número: ");
        n2 = Convert.ToInt32(Console.ReadLine());

        if ((n1 > 0) && (n2 > 0))
            Console.WriteLine("Ambos números son positivos.");
        else
            Console.WriteLine("Al menos uno no es positivo.");
    }
}
```

Una curiosidad: en C# (y en algún otro lenguaje de programación), la evaluación de dos condiciones que estén enlazadas con "Y" se hace "en **cortocircuito**": si la primera de las condiciones no se cumple, ni siquiera se llega a comprobar la segunda, porque se sabe de antemano que la condición formada

por ambas no podrá ser cierta. Eso supone que en el primer ejemplo anterior, if ((opcion==1) && (usuario==2)), si "opcion" no vale 1, el compilador no se molesta en ver cuál es el valor de "usuario", porque, sea el que sea, no podrá hacer que sea "verdadera" toda la expresión. Lo mismo ocurriría si hay dos condiciones enlazadas con "o", y la primera de ellas es "verdadera": no será necesario comprobar la segunda, porque ya se sabe que la expresión global será "verdadera".

Como la mejor forma de entender este tipo de expresiones es practicándolas, vamos a ver unos cuantos ejercicios propuestos...

Ejercicios propuestos:

Ejercicio propuesto 2.1.5.1: *Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 o de 3.*

Ejercicio propuesto 2.1.5.2: *Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 y de 3 simultáneamente.*

Ejercicio propuesto 2.1.5.3: *Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 pero no de 3.*

Ejercicio propuesto 2.1.5.4: *Crea un programa que pida al usuario un número entero y responda si no es múltiplo de 2 ni de 3.*

Ejercicio propuesto 2.1.5.5: *Crea un programa que pida al usuario dos números enteros y diga si ambos son pares.*

Ejercicio propuesto 2.1.5.6: *Crea un programa que pida al usuario dos números enteros y diga si (al menos) uno es par.*

Ejercicio propuesto 2.1.5.7: *Crea un programa que pida al usuario dos números enteros y diga si uno y sólo uno es par.*

Ejercicio propuesto 2.1.5.8: *Crea un programa que pida al usuario dos números enteros y diga "Uno de los números es positivo", "Los dos números son positivos" o bien "Ninguno de los números es positivo", según corresponda.*

Ejercicio propuesto 2.1.5.9: *Crea un programa que pida al usuario tres números y muestre cuál es el mayor de los tres.*

Ejercicio propuesto 2.1.5.10: *Crea un programa que pida al usuario dos números enteros y diga si son iguales o, en caso contrario, cuál es el mayor de ellos.*

2.1.6. El peligro de la asignación en un "if"

Cuidado con el comparador de igualdad: hay que recordar que el formato es if (a==b) ... Si no nos acordamos y escribimos if (a=b), estamos intentando asignar a "a" el valor de "b".

En algunos compiladores de lenguaje C, esto podría ser un problema serio, porque se considera válido hacer una asignación dentro de un "if" (aunque la mayoría de compiladores modernos al menos nos avisarían de que quizá estemos asignando un valor sin pretenderlo, pero no es un "error" que invalide

la compilación, sino un "aviso", lo que permite que se genere un ejecutable, y podríamos pasar por alto el aviso, dando lugar a un funcionamiento incorrecto de nuestro programa).

En el caso del lenguaje C#, este riesgo no existe, porque la "condición" debe ser algo cuyo resultado sea "verdadero" o "falso" (lo que pronto llamaremos un dato de tipo "bool"), de modo que obtendríamos un error de compilación "Cannot implicitly convert type 'int' to 'bool'" (no puedo convertir un "int" a "bool"). Es el caso del siguiente programa:

```
// Ejemplo_02_01_06a.cs
// Condiciones con if: comparación incorrecta
// Introducción a C#

using System;

public class Ejemplo_02_01_06a
{
    public static void Main()
    {
        int numero;

        Console.WriteLine("Introduce un número");
        numero = Convert.ToInt32(Console.ReadLine());
        if (numero = 0)
            Console.WriteLine("El número es cero.");
        else
            if (numero < 0)
                Console.WriteLine("El número es negativo.");
            else
                Console.WriteLine("El número es positivo.");
    }
}
```

Nota: en lenguajes como C y C++, en los que sí existe este riesgo de asignar un valor en vez de comparar, se suele recomendar plantear la comparación al revés, colocando el número en el lado izquierdo, de modo que si olvidamos el doble signo de "=", obtendríamos una asignación no válida y el programa no compilaría:

```
if (0 == numero)  ...
```

Ejercicios propuestos:

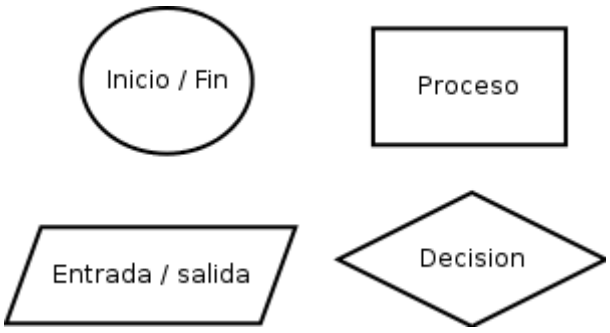
Ejercicio propuesto 2.1.6.1: *Crea una variante del ejemplo 02_01_06a, en la que la comparación de igualdad sea correcta y en la que las variables aparezcan en el lado derecho de la comparación y los números en el lado izquierdo.*

2.1.7. Introducción a los diagramas de flujo

A veces puede resultar difícil ver claro donde usar un "else" o qué instrucciones de las que siguen a un "if" deben ir entre llaves y cuáles no. Generalmente la dificultad está en el hecho de intentar teclear directamente un programa en C#, en vez de pensar en el problema que se pretende resolver.

Para ayudarnos a centrarnos en el problema, existen notaciones gráficas, como los diagramas de flujo, que nos permiten ver mejor qué se debe hacer y cuándo.

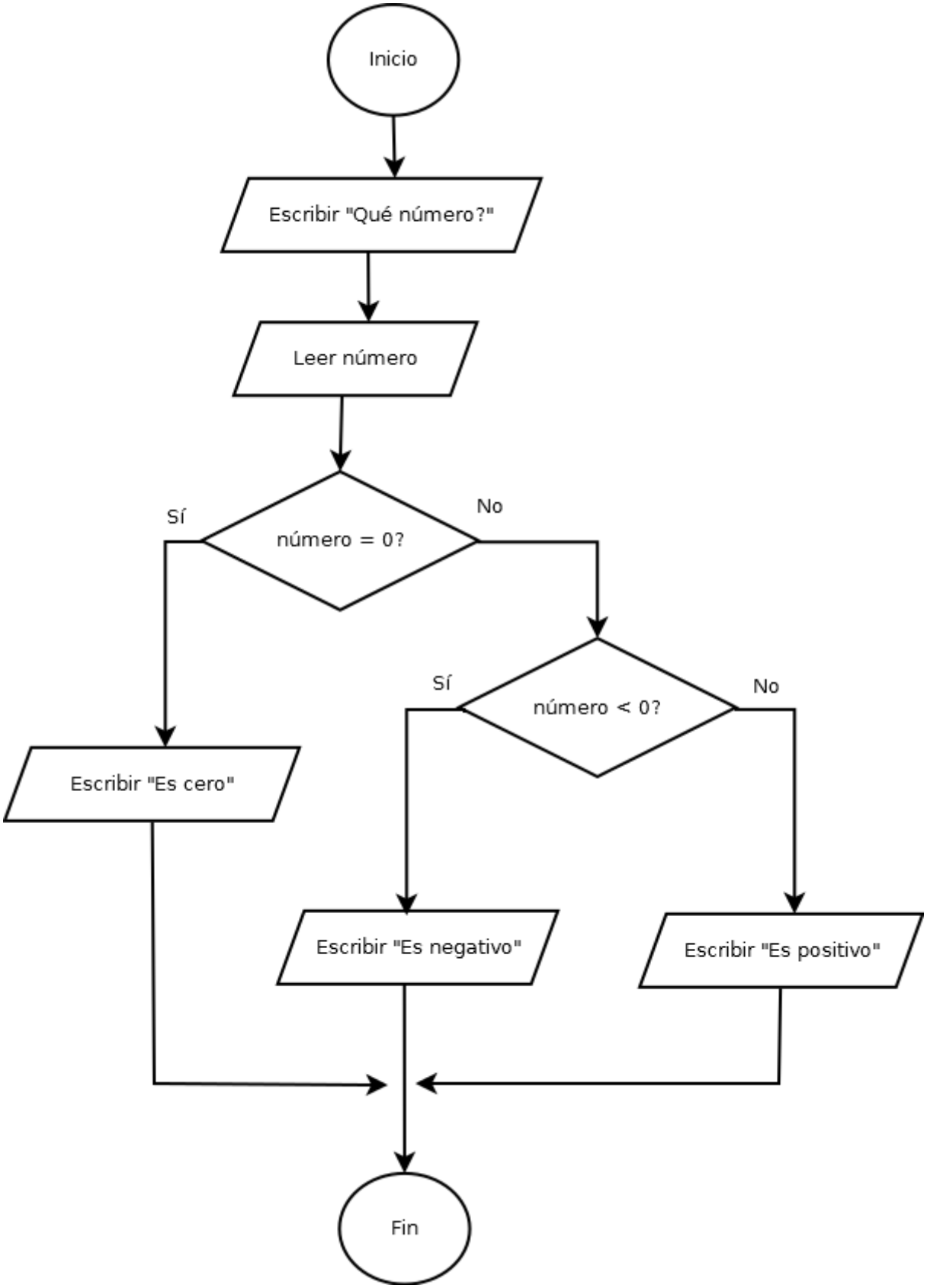
En primer lugar, vamos a ver los 4 elementos básicos de un diagrama de flujo, y luego los aplicaremos a un caso concreto.



Es decir:

- El inicio o el final del programa se indica dentro de un círculo.
- Los procesos internos, como realizar operaciones, se encuadran en un rectángulo.
- Las entradas y salidas (escrituras en pantalla y lecturas de teclado) se indican con un paralelogramo que tenga sus lados superior e inferior horizontales, pero no tenga verticales los otros dos.
- Las decisiones se indican dentro de un rombo, desde el que saldrán dos flechas. Cada una de ellas corresponderá a la secuencia de pasos a dar si se cumple una de las dos opciones posibles.

Vamos a aplicarlo al ejemplo de un programa que pida un número al usuario y diga si es positivo, negativo o cero:


Inicio
Escribir "Qué número?"
Leer número
número = 0?
Sí
No
Escribir "Es cero"
número < 0?
Sí
No
Escribir "Es negativo"
Escribir "Es positivo"
Fin

El paso de aquí al correspondiente programa en lenguaje C# (el que vimos en el Ejemplo_02_01_06a) debe ser casi inmediato: sabemos cómo leer de teclado, cómo escribir en pantalla, y las decisiones serán un "if", que si se cumple ejecutará la sentencia que aparece en su salida "si" y si no se cumple ("else") ejecutará lo que aparezca en su salida "no".

Eso sí, hay que tener en cuenta que ésta es una **notación anticuada**, y que no permite representar de formar fiable las estructuras repetitivas que veremos dentro de poco, por lo que su uso actual es muy limitado.

Ejercicios propuestos:

Ejercicio propuesto 2.1.7.1: *Crea el diagrama de flujo para el programa que pide dos números al usuario y dice cuál es el mayor de los dos.*

Ejercicio propuesto 2.1.7.2: *Crea el diagrama de flujo para el programa que pide al usuario dos números y dice si uno de ellos es positivo, si lo son los dos o si no lo es ninguno.*

Ejercicio propuesto 2.1.7.3: *Crear el diagrama de flujo para el programa que pide tres números al usuario y dice cuál es el mayor de los tres.*

2.1.8. Operador condicional: ?

En C#, al igual que en la mayoría de lenguajes que derivan de C, hay otra forma de asignar un valor según se cumpla una condición o no, más compacta pero también más difícil de leer. Es el **"operador condicional" ?** : (también conocido como "operador ternario"), que se usa:

```
nombreVariable = condicion ?  valor1 : valor2;
```

y equivale a decir "si se cumple la condición, toma el valor valor1; si no, toma el valor valor2". Un ejemplo de cómo podríamos usarlo sería para calcular el mayor de dos números:

```
numeroMayor = a>b ?  a : b;
```

esto equivale a la siguiente orden "if":

```
    if ( a > b )
        numeroMayor = a;
    else
        numeroMayor = b;
```

Al igual que en este ejemplo, podremos usar el operador condicional cuando queramos optar entre dos valores posibles para una variable, dependiendo de si se cumple o no una condición.

Aplicado a un programa sencillo, podría ser

```
// Ejemplo_02_01_08a.cs
// El operador condicional
// Introducción a C#
```

```
using System;

public class Ejemplo_02_01_08a
{
    public static void Main()
    {
        int a, b, mayor;

        Console.Write("Escriba un número: ");
        a = Convert.ToInt32(Console.ReadLine());
        Console.Write("Escriba otro: ");
        b = Convert.ToInt32(Console.ReadLine());

        mayor = a>b ?  a : b;

        Console.WriteLine("El mayor de los números es {0}.", mayor);
    }
}
```

Un segundo ejemplo, que sume o reste dos números según la opción que se escoja, sería:

```
// Ejemplo_02_01_08b.cs
// El operador condicional (2)
// Introducción a C#

using System;

public class Ejemplo_02_01_08b
{
    public static void Main()
    {
        int a, b, operacion, resultado;

        Console.Write("Escriba un número: ");
        a = Convert.ToInt32(Console.ReadLine());

        Console.Write("Escriba otro: ");
        b = Convert.ToInt32(Console.ReadLine());

        Console.Write("Escriba una operación (1 = resta; otro = suma): ");
        operacion = Convert.ToInt32(Console.ReadLine());

        resultado = operacion == 1 ?  a-b : a+b;
        Console.WriteLine("El resultado es {0}.", resultado);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.1.8.1: *Crea un programa que use el operador condicional para mostrar el valor absoluto de un número de la siguiente forma: si el número es positivo, se mostrará tal cual; si es negativo, se mostrará cambiado de signo.*

Ejercicio propuesto 2.1.8.2: *Usa el operador condicional para calcular el menor de dos números.*

2.1.9. switch

Si queremos ver **varios posibles valores**, sería muy pesado tener que hacerlo con muchos "if" seguidos o encadenados. La alternativa es emplear la orden "switch", cuya sintaxis es

```
switch (expresion)
{
    case valor1: sentencia1;
            break;
    case valor2: sentencia2;
            sentencia2b;
            break;
    case valor3:
            goto case valor1;
     ...
    case valorN: sentenciaN;
            break;
    default:
            otraSentencia;
            break;
}
```

Es decir:

- Tras la palabra "**switch**" se escribe la expresión a analizar, entre paréntesis.
- Después, tras varias órdenes "**case**" se indica cada uno de los valores posibles.
- Los pasos (porque pueden ser varios) que se deben dar si la expresión tiene un cierto valor se indican a continuación, terminando con "**break**".
- Si hay que hacer algo en caso de que no se cumpla ninguna de las condiciones, se detalla después de la palabra "**default**".
- Si dos casos tienen que hacer lo mismo, se añade "**goto case**" a uno de ellos para indicarlo.

Vamos a ver un ejemplo, que diga si el símbolo que introduce el usuario es una cifra numérica, un espacio u otro símbolo. Para ello usaremos un dato de tipo "char" (carácter), que veremos con más detalle en el próximo tema. De momento nos basta que deberemos usar Convert.ToChar si lo leemos desde teclado con ReadLine, y que le podemos dar un valor (o compararlo) usando comillas simples:

```
// Ejemplo_02_01_09a.cs
// La orden "switch"  (1)
// Introducción a C#

using System;

public class Ejemplo_02_01_09a
{
    public static void Main()
    {
        char letra;

        Console.WriteLine("Introduce una letra");
        letra = Convert.ToChar( Console.ReadLine() );

        switch (letra)
        {
            case ' ': Console.WriteLine("Espacio.");
                      break;
```

```
            case '1': goto case '0';
            case '2': goto case '0';
            case '3': goto case '0';
            case '4': goto case '0';
            case '5': goto case '0';
            case '6': goto case '0';
            case '7': goto case '0';
            case '8': goto case '0';
            case '9': goto case '0';
            case '0': Console.WriteLine("Dígito.");
                      break;
            default:  Console.WriteLine("Ni espacio ni dígito.");
                      break;
        }
    }
}
```

Cuidado quien venga del lenguaje C: en C se puede dejar que un caso sea manejado por el siguiente, lo que se consigue si no se usa "break", mientras que C# siempre obliga a usar "break" o "goto" al final de cada caso (para evitar errores provocados por un "break" olvidado) con la única excepción de que un caso no haga absolutamente nada excepto dejar pasar el control al siguiente caso, y en ese caso se puede dejar totalmente vacío:

```
// Ejemplo_02_01_09b.cs
// La orden "switch"  (variante sin break)
// Introducción a C#

using System;

public class Ejemplo_02_01_09b
{
    public static void Main()
    {
        char letra;

        Console.WriteLine("Introduce una letra");
        letra = Convert.ToChar( Console.ReadLine() );

        switch (letra)
        {
            case ' ': Console.WriteLine("Espacio.");
                      break;
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case '0': Console.WriteLine("Dígito.");
                      break;
            default:  Console.WriteLine("Ni espacio ni dígito.");
                      break;
        }
    }
}
```

En el lenguaje C, que es más antiguo, sólo se podía usar "switch" para comprobar valores de variables "simples" (numéricas y caracteres); en C#, que es un lenguaje más evolucionado, se puede usar también para comprobar valores de cadenas de texto ("strings").

Una cadena de texto, como veremos con más detalle en el próximo tema, se declara con la palabra "**string**", se puede leer de teclado con ReadLine (sin necesidad de convertir) y se le puede dar un valor desde programa si se indica entre comillas dobles. Por ejemplo, un programa que nos salude de forma personalizada si somos "Juan" o "Pedro" podría ser:

```
// Ejemplo_02_01_09c.cs
// La orden "switch" con cadenas de texto
// Introducción a C#

using System;

public class Ejemplo_02_01_09c
{
    public static void Main()
    {
        string nombre;

        Console.WriteLine("Introduce tu nombre");
        nombre = Console.ReadLine();

        switch (nombre)
        {
            case "Juan":  Console.WriteLine("Bienvenido, Juan.");
                          break;
            case "Pedro": Console.WriteLine("Que tal estas, Pedro.");
                          break;
            default:      Console.WriteLine("Procede con cautela,
desconocido.");
                          break;
        }
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.1.9.1: *Crea un programa que pida un número del 1 al 5 al usuario, y escriba el nombre de ese número, usando "switch" (por ejemplo, si introduce "1", el programa escribirá "uno").*

Ejercicio propuesto 2.1.9.2: *Crea un programa que lea una letra tecleada por el usuario y diga si se trata de un signo de puntuación (. , ; :), una cifra numérica (del 0 al 9) o algún otro carácter, usando "switch" (pista: habrá que usar un dato de tipo "char").*

Ejercicio propuesto 2.1.9.3: *Crea un programa que lea una letra tecleada por el usuario y diga si se trata de una vocal, una cifra numérica o una consonante, usando "switch"*

Ejercicio propuesto 2.1.9.4: *Repite el ejercicio 2.1.9.1, empleando "if" en lugar de "switch".*

Ejercicio propuesto 2.1.9.5: *Repite el ejercicio 2.1.9.2, empleando "if" en lugar de "switch" (pista: como las cifras numéricas del 0 al 9 están ordenadas, no hace falta comprobar los 10 valores, sino que se puede hacer con "if ((simbolo >= '0') && (simbolo <='9'))").*

Ejercicio propuesto 2.1.9.6: *Repite el ejercicio 2.1.9.3, empleando "if" en lugar de "switch".*

2.2. Estructuras repetitivas

Hemos visto cómo comprobar condiciones, pero no cómo hacer que una cierta parte de un programa se repita un cierto número de veces o mientras se cumpla una condición (lo que llamaremos un "bucle"). En C# tenemos varias formas de conseguirlo.

2.2.1. while

2.2.1.1. Estructura básica de un bucle "while"

Si queremos hacer que una sección de nuestro programa se repita mientras se cumpla una cierta condición, usaremos la orden "while". Esta orden tiene dos formatos distintos, según comprobemos la condición al principio o al final del bloque repetitivo.

En el primer caso, su sintaxis es

```
while (condición)
  sentencia;
```

Es decir, la sentencia se repetirá **mientras** la condición sea cierta. Si la condición es falsa ya desde un principio, la sentencia no se ejecuta nunca. Si queremos que se repita más de una sentencia, basta agruparlas entre llaves: { y }.

Un ejemplo que nos diga si cada número que tecleemos es positivo o negativo, y que termine cuando tecleemos el número 0, podría ser:

```
// Ejemplo_02_02_01_01a.cs
// La orden "while": mientras...
// Introducción a C#

using System;

public class Ejemplo_02_02_01_01a
{
    public static void Main()
    {
        int numero;

        Console.Write("Teclea un número (0 para salir): ");
        numero = Convert.ToInt32(Console.ReadLine());

        while (numero != 0)
        {
            if (numero > 0) Console.WriteLine("Es positivo");
            else Console.WriteLine("Es negativo");

            Console.WriteLine("Teclea otro número (0 para salir): ");
            numero = Convert.ToInt32(Console.ReadLine());
        }
    }
}
```

En este ejemplo, si se introduce 0 la primera vez, la condición es falsa y ni siquiera se entra al bloque del "while", terminando el programa inmediatamente.

Ejercicios propuestos:

Ejercicio propuesto 2.2.1.1.1: *Crea un programa que pida al usuario su contraseña (numérica). Deberá terminar cuando introduzca como contraseña el número 1111, pero volvérsela a pedir tantas veces como sea necesario.*

Ejercicio propuesto 2.2.1.1.2: *Crea un "calculador de cuadrados": pedirá al usuario un número y mostrará su cuadrado. Se repetirá mientras el número introducido no sea cero (usa "while" para conseguirlo).*

Ejercicio propuesto 2.2.1.1.3: *Crea un programa que pida de forma repetitiva pares de números al usuario. Tras introducir cada par de números, responderá si el primero es múltiplo del segundo.*

Ejercicio propuesto 2.2.1.1.4: *Crea una versión mejorada del programa anterior, que, tras introducir cada par de números, responderá si el primero es múltiplo del segundo, o el segundo es múltiplo del primero, o ninguno de ellos es múltiplo del otro.*

2.2.1.2. Contadores usando un bucle "while"

Ahora que sabemos "repetir" cosas, podemos utilizarlo también para **contar**. Por ejemplo, si queremos contar del 1 al 5, usaríamos una variable que empezase en 1, que aumentaría una unidad en cada repetición y se repetiría hasta llegar al valor 5, así:

```
// Ejemplo_02_02_01_02a.cs
// Contar con "while"
// Introducción a C#

using System;

public class Ejemplo_02_02_01_02a
{
    public static void Main()
    {
        int n = 1;

        while (n < 6)
        {
            Console.WriteLine(n);
            n = n + 1;
        }
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.2.1.2.1: *Crea un programa que escriba en pantalla los números del 1 al 10, usando "while".*

Ejercicio propuesto 2.2.1.2.2: *Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "while".*

Ejercicio propuesto 2.2.1.2.3: *Crea un programa calcule cuántas cifras tiene un número entero positivo (pista: se puede hacer dividiendo varias veces entre 10).*

Ejercicio propuesto 2.2.1.2.4: *Crea el diagrama de flujo y la versión en C# de un programa que dé al usuario tres oportunidades para adivinar un número del 1 al 10 y le informe cuando lo haga.*

2.3. Saltar a otro punto del programa: goto

El lenguaje C# también permite la orden "goto", para hacer saltos incondicionales. Su uso indisciplinado está muy mal visto, porque puede ayudar a hacer programas llenos de saltos, muy difíciles de seguir. Pero en casos concretos puede ser muy útil, por ejemplo, para salir de un bucle muy anidado (un "for" dentro de otro "for" que a su vez está dentro de otro "for": en este caso, "break" sólo saldría del "for" más interno). Al igual que ocurría con la orden "break", será preferible replantear las condiciones de forma más natural, y no utilizar "goto".

El formato de "goto" es:

```
goto donde;
```

y la posición de salto se indica con su nombre seguido de dos puntos (:)

```
donde:
```

como en el siguiente ejemplo:

```
// Ejemplo_02_03a.cs
// "for" y "goto"
// Introducción a C#

using System;

public class Ejemplo_02_03a
{
    public static void Main()
    {
        int i, j;

        for (i=0; i<=5; i++)
            for (j=0; j<=20; j=j+2)
            {
                if ((i==1) && (j>=7))
                    goto salida;
                Console.WriteLine("i vale {0} y j vale {1}.", i, j);
            }

        salida:
```

```
            Console.Write("Fin del programa");

    }
}
```

El resultado de este programa es:

```
i vale 0 y j vale 0.
i vale 0 y j vale 2.
i vale 0 y j vale 4.
i vale 0 y j vale 6.
i vale 0 y j vale 8.
i vale 0 y j vale 10.
i vale 0 y j vale 12.
i vale 0 y j vale 14.
i vale 0 y j vale 16.
i vale 0 y j vale 18.
i vale 0 y j vale 20.
i vale 1 y j vale 0.
i vale 1 y j vale 2.
i vale 1 y j vale 4.
i vale 1 y j vale 6.
Fin del programa
```

Vemos que cuando i=1 y j>=7, se sale de los dos "for".

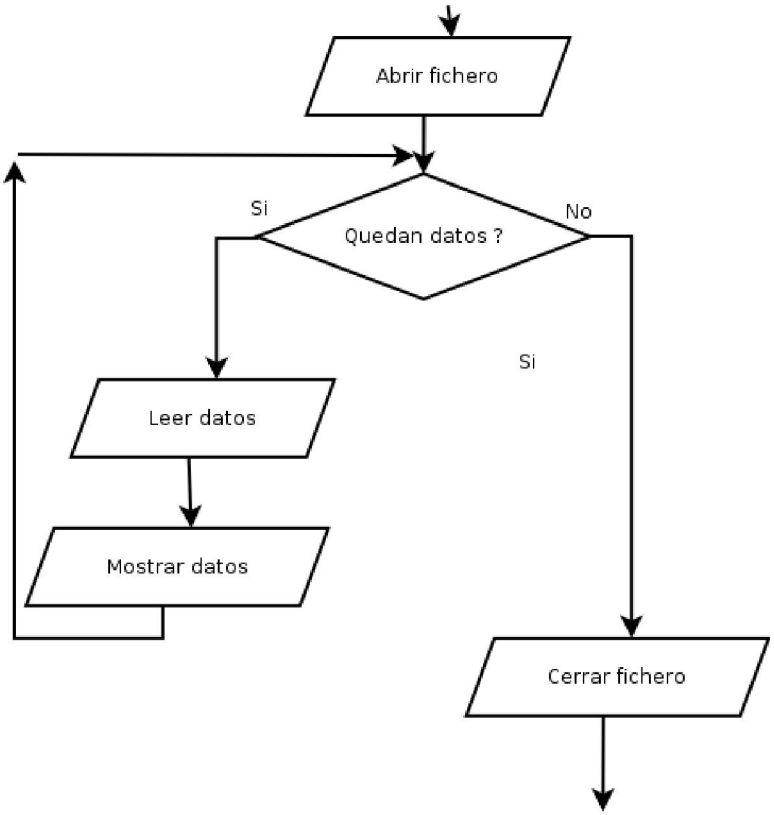
Ejercicios propuestos:

Ejercicio propuesto 2.3.1: *Crea un programa que escriba los números del 1 al 10, separados por un espacio, sin avanzar de línea. No puedes usar "for", ni "while", ni "do..while", sólo "if" y "goto".*

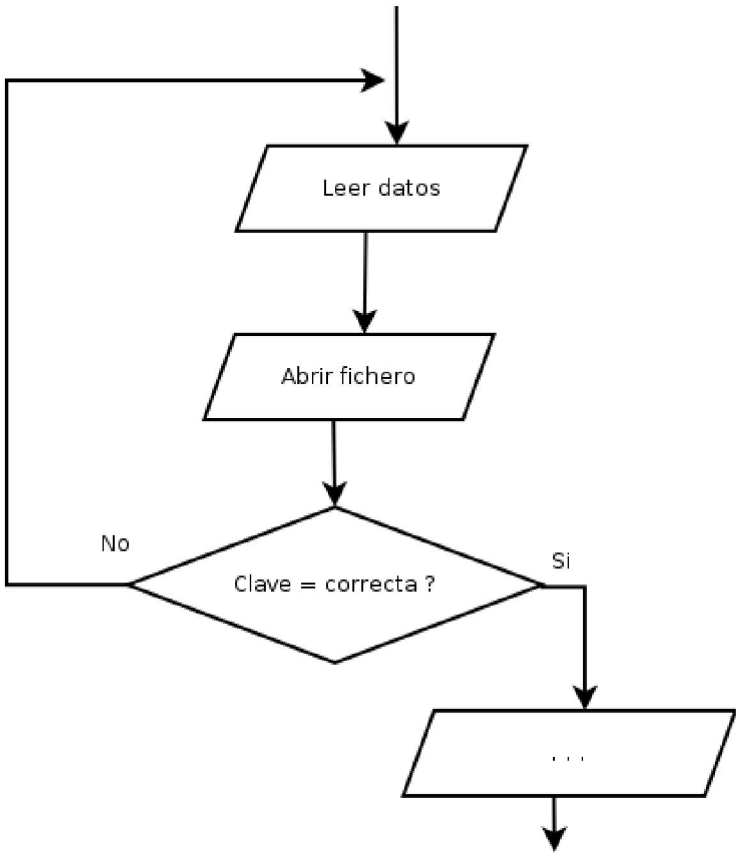
2.4. Más sobre diagramas de flujo. Diagramas de Chapin

Cuando comenzamos el tema, vimos cómo ayudarnos de los diagramas de flujo para plantear lo que un programa debe hacer. Si entendemos esta herramienta, el paso a C# (o a casi cualquier otro lenguaje de programación es sencillo). Pero este tipo de diagramas es antiguo, no tiene en cuenta todas las posibilidades del lenguaje C# (y de muchos otros lenguajes actuales). Por ejemplo, no existe una forma clara de representar una orden "switch", que equivaldría a varias condiciones encadenadas.

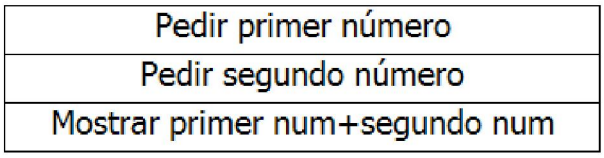
Por su parte, un bucle "while" se vería como una condición que hace que algo se repita (una flecha que vuelve hacia atrás, al punto en el que se comprobaba la condición):



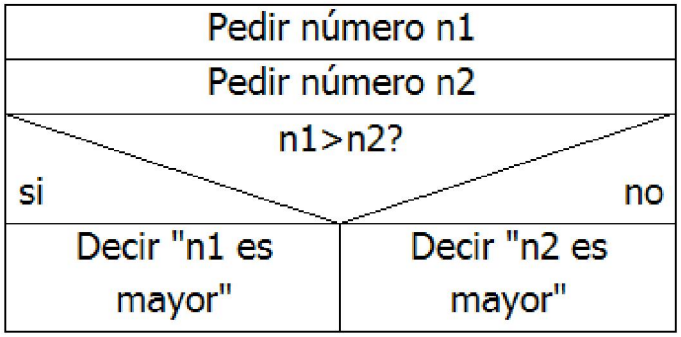
Y un "do...while" se representaría como una condición al final de un bloque que se repite:



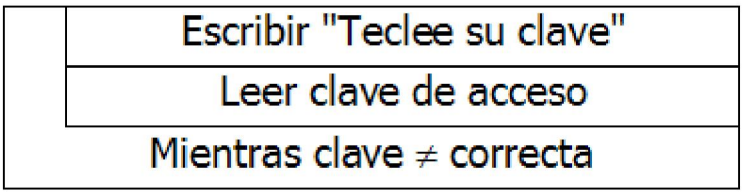
Aun así, existen otras notaciones más modernas y que pueden resultar más cómodas. Sólo comentaremos una: los diagramas de Chapin. En estos diagramas, se representa cada orden dentro de una caja:



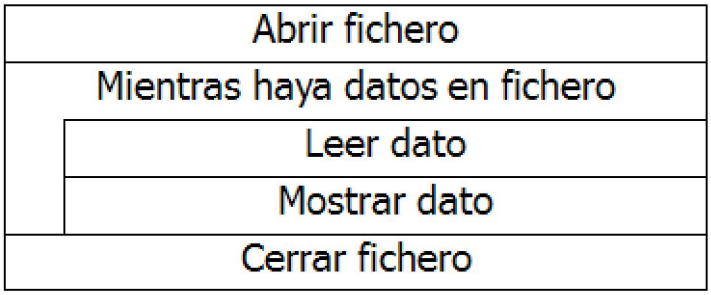
Las condiciones se denotan dividiendo las cajas en dos:



Y las condiciones repetitivas se indican dejando una barra a la izquierda, que marca qué zona es la que se repite, tanto si la condición se comprueba al final (do..while):



como si se comprueba al principio (while):



En ambos casos, no existe una gráfica "clara" para los "for".

2.5. foreach

Nos queda por ver otra orden que permite hacer cosas repetitivas: "foreach" (se traduciría "para cada"). La veremos más adelante, cuando manejemos estructuras de datos más complejas, que es en las que la nos resultará útil para extraer los datos de uno en uno. De momento, el único dato compuesto que hemos visto (y todavía con muy poco detalle) es la cadena de texto, "string", de la que podríamos obtener las letras una a una con "foreach" así:

```
// Ejemplo_02_05a.cs
// Primer ejemplo de "foreach"
// Introducción a C#

using System;

public class Ejemplo_02_05a
{
    public static void Main()
    {
        Console.Write("Dime tu nombre: ");
        string nombre = Console.ReadLine();
        foreach(char letra in nombre)
        {
            Console.WriteLine(letra);
        }
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 2.5.1: *Crea un programa que cuente cuántas veces aparece la letra 'a' en una frase que teclee el usuario, utilizando "foreach".*

2.6. Recomendación de uso para los distintos tipos de bucle

En general, nos interesará usar "**while**" cuando puede que la parte repetitiva no se llegue a repetir nunca (por ejemplo: cuando leemos un fichero, si el fichero está vacío, no habrá datos que leer).

De igual modo, "**do...while**" será lo adecuado cuando debamos repetir al menos una vez (por ejemplo, para pedir una clave de acceso, se le debe preguntar al menos una vez al usuario, o quizá más veces, si no la teclea correctamente).

En cuanto a "**for**", es equivalente a un "while", pero la sintaxis habitual de la orden "for" hace que sea especialmente útil cuando sabemos exactamente cuántas veces queremos que se repita (por ejemplo: 10 veces sería "for (i=1; i<=10; i++)"). Conceptualmente, si un "for" necesita un "break" para ser interrumpido en un caso especial, es porque realmente no se trata de un contador, y en ese caso debería ser reemplazado por un "while", para que el programa resulte más legible

Ejercicios propuestos:

Ejercicio propuesto 2.6.1: *Crear un programa que dé al usuario la oportunidad de adivinar un número del 1 al 100 (prefijado en el programa) en un máximo de 6 intentos. En cada pasada deberá avisar de si se ha pasado o se ha quedado corto.*

Ejercicio propuesto 2.6.2: *Crear un programa que descomponga un número (que teclee el usuario) como producto de sus factores primos. Por ejemplo, 60 = 2 · 2 · 3 · 5*

Ejercicio propuesto 2.6.3: *Crea un programa que calcule un número elevado a otro, usando multiplicaciones sucesivas.*

Ejercicio propuesto 2.6.4: *Crea un programa que "dibuje" un rectángulo formado por asteriscos, con el ancho y el alto que indique el usuario, usando dos "for" anidados. Por ejemplo, si desea anchura 4 y altura 3, el rectángulo sería así:*

```
****
****
****
```

Ejercicio propuesto 2.6.5: *Crea un programa que "dibuje" un triángulo decreciente, con la altura que indique el usuario. Por ejemplo, si el usuario dice que desea 4 caracteres de alto, el triángulo sería así:*

```
****
***
**
*
```

Ejercicio propuesto 2.6.6: *Crea un programa que "dibuje" un rectángulo hueco, cuyo borde sea una fila (o columna) de asteriscos y cuyo interior esté formado por espacios en blanco, con el ancho y el alto que indique el usuario. Por ejemplo, si desea anchura 4 y altura 3, el rectángulo sería así:*

```
****
*  *
****
```

Ejercicio propuesto 2.6.7: *Crea un programa que "dibuje" un triángulo creciente, alineado a la derecha, con la altura que indique el usuario. Por ejemplo, si el usuario dice que desea 4 caracteres de alto, el triángulo sería así:*

```
   *
  **
 ***
****
```

Ejercicio propuesto 2.6.8: *Crea un programa que devuelva el cambio de una compra, utilizando monedas (o billetes) del mayor valor posible. Supondremos que tenemos una cantidad ilimitada de monedas (o billetes) de 100, 50, 20, 10, 5, 2 y 1, y que no hay decimales. La ejecución podría ser algo como:*

```
Precio? 44
Pagado? 100
Su cambio es de 56: 50 5 1
Precio? 1
Pagado? 100
Su cambio es de 99: 50 20 20 5 2 2
```

Ejercicio propuesto 2.6.9: *Crea un programa que "dibuje" un cuadrado formado por cifras sucesivas, con el tamaño que indique el usuario, hasta un máximo de 9. Por ejemplo, si desea tamaño 5, el cuadrado sería así:*

11111
22222
33333
44444
55555

2.7. Una alternativa para el control errores: las excepciones

La forma "clásica" del control de errores es usar instrucciones "if", que vayan comprobando cada una de las posibles situaciones que pueden dar lugar a un error, a medida que estas situaciones llegan. Esto tiende a hacer el programa más difícil de leer, porque la lógica de la resolución del problema se ve interrumpida por órdenes que no tienen que ver con el problema en sí, sino con las posibles situaciones de error. Por eso, los lenguajes modernos, como C#, permiten una alternativa: el manejo de "excepciones".

La idea es la siguiente: "intentaremos" dar una serie de pasos, y al final de todos ellos indicaremos qué hay que hacer en caso de que alguno no se consiga completar. Esto permite que el programa sea más legible que la alternativa "convencional".

Lo haremos dividiendo el fragmento de programa en dos **bloques**:

- En un primer bloque, indicaremos los pasos que queremos "**intentar**" (try).
- A continuación, detallaremos las posibles situaciones de error (excepciones) que queremos "**interceptar**" (catch), y lo que se debe hacer en ese caso.

Lo veremos más adelante con más detalle, cuando nuestros programas sean más complejos, especialmente en el **manejo de ficheros**, pero podemos acercarnos con un primer ejemplo, que intente dividir dos números, e intercepte los posibles errores:

```
// Ejemplo_02_07a.cs
// Excepciones (1)
// Introducción a C#

using System;

public class Ejemplo_02_07a
{

    public static void Main()
    {
        int numero1, numero2, resultado;

        try
        {
            Console.WriteLine("Introduzca el primer numero");
            numero1 = Convert.ToInt32( Console.ReadLine() );

            Console.WriteLine("Introduzca el segundo numero");
            numero2 = Convert.ToInt32( Console.ReadLine() );
```

```
            resultado = numero1 / numero2;
            Console.WriteLine("Su división es: {0}", resultado);
        }
        catch (Exception errorEncontrado)
        {
            Console.WriteLine("Ha habido un error: {0}",
                errorEncontrado.Message);
        }
    }
}
```

(La variable "errorEncontrado" es de tipo "Exception", y nos sirve para poder acceder a detalles como el mensaje correspondiente a ese tipo de excepción: errorEncontrado.Message)

En este ejemplo, si escribimos un texto en vez de un número, obtendríamos como respuesta

```
Introduzca el primer numero
hola
Ha habido un error: La cadena de entrada no tiene el formato correcto.
```

Y si el segundo número es 0, se nos diría

```
Introduzca el primer numero
3
Introduzca el segundo numero
0
Ha habido un error: Intento de dividir por cero.
```

Una alternativa más elegante es no "atrapar" todos los posibles errores a la vez, sino uno por uno (con varias sentencias "catch"), para poder tomar distintas acciones, o al menos dar mensajes de error más detallados, así:

```
// Ejemplo_02_07b.cs
// Excepciones (2)
// Introducción a C#

using System;

public class Ejemplo_02_07b
{
    public static void Main()
    {
        int numero1, numero2, resultado;

        try
        {
            Console.WriteLine("Introduzca el primer numero");
            numero1 = Convert.ToInt32( Console.ReadLine() );

            Console.WriteLine("Introduzca el segundo numero");
            numero2 = Convert.ToInt32( Console.ReadLine() );

            resultado = numero1 / numero2;
            Console.WriteLine("Su división es: {0}", resultado);
        }
```

```
        catch (FormatException)
        {
            Console.WriteLine("No es un número válido");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("No se puede dividir entre cero");
        }
    }
}
```

Como se ve en este ejemplo, si no vamos a usar detalles adicionales del error que ha afectado al programa, no necesitamos declarar ninguna variable de tipo Exception: nos basta con construcciones como "catch (FormatException)" en vez de "catch (FormatException e)".

¿Y cómo sabemos qué excepciones debemos interceptar? La mejor forma es mirar en la "referencia oficial" para programadores de C#, la MSDN (Microsoft Developer Network): si tecleamos en un buscador de Internet algo como "msdn convert toint32" nos llevará a una página en la que podemos ver que hay dos excepciones que podemos obtener en ese intento de conversión de texto a entero: FormatException (no se ha podido convertir) y OverflowException (número demasiado grande). Otra alternativa más peligrosa es "probar el programa" y ver qué errores obtenemos en pantalla al introducir un valor no válido. Esta alternativa es la menos deseable, porque quizá pasemos por alto algún tipo de error que pueda surgir y que nosotros no hayamos contemplado. En cualquier caso, volveremos a las excepciones más adelante.

Ejercicios propuestos:

Ejercicio propuesto 2.7.1: *Crea un programa que pregunte al usuario su edad y su año de nacimiento. Si la edad que introduce no es un número válido, mostrará un mensaje de aviso. Lo mismo ocurrirá si el año de nacimiento no es un número válido.*

Ejercicio propuesto 2.7.2: *Crea un programa que pregunte al usuario su edad y su año de nacimiento. Si la edad que introduce no es un número válido, mostrará un mensaje de aviso, pero aun así le preguntará su año de nacimiento.*