2. Estructuras de control

2.1. Estructuras alternativas

2.1.1. if

Vamos a ver cómo podemos comprobar si se cumplen condiciones. La primera construcción que usaremos será **"si ... entonces ...**". El formato es

```
if (condición) sentencia;
```

Vamos a verlo con un ejemplo:

```
/*---------------------------*/
/*  Ejemplo en C# nº 5:      */
/*  ejemplo05.cs             */
/*                           */
/*  Condiciones con if       */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo05
{
    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 algo más adelante. En todos los casos, la condición que queremos comprobar deberá indicarse entre paréntesis.

Este programa comienza por un comentario 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:

- **(2.1.1.1)** Crear 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) ...).
- **(2.1.1.2)** Crear un programa que pida al usuario dos números enteros y diga cuál es el mayor de ellos.
- **(2.1.1.3)** Crear 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 en C# nº 6:       */
/*  ejemplo06.cs              */
/*                            */
/*  Condiciones con if (2)    */
/*  Sentencias compuestas     */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo06
{
  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 "Ejemplo06" */
```

En este caso, si el número es positivo, se hacen dos cosas: escribir un texto y luego... ¡escribir otro! (En este ejemplo, esos dos "WriteLine" podrían ser uno solo, en el que los dos textos estuvieran separados por un carácter especial, el símbolo de "salto de línea"; 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 "Ejemplo06" está un poco más a la derecha que la cabecera "public class Ejemplo06", 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 algo más a la derecha que la orden "if". Este "**sangrado**" del texto se suele llamar "**escritura indentada**". Un tamaño habitual para el sangrado es de 4 espacios, aunque en este texto muchas veces usaremos sólo dos espacios, para no llegar al margen derecho del papel con demasiada facilidad.

Ejercicios propuestos:

- **(2.1.2.1)** Crear un programa que pida al usuario un número entero. Si es múltiplo de 10, se lo avisará 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 en C# nº 7:       */
/*  ejemplo07.cs              */
/*                            */
/*  Condiciones con if (3)    */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo07
{
  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:

- **(2.1.3.1)** Crear un programa que multiplique dos números enteros de la siguiente forma: pedirá al usuario un primer número entero. Si el número que se que 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.
- **(2.1.3.2)** Crear un programa que pida al usuario dos números enteros. Si el segundo no es cero, mostrará el resultado de dividir entre el primero y 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 en C# nº 8:      */
/*  ejemplo08.cs             */
/*                           */
/*  Condiciones con if (4)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo08
{
  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 en C# nº 9:      */
/*  ejemplo09.cs             */
/*                           */
/*  Condiciones con if (5)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo09
{
```

```
  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 8) 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 9), 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 en C# nº 10:     */
/*  ejemplo10.cs             */
/*                           */
/*  Condiciones con if (6)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo10
{
  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.");
  }
}
```

Ejercicio propuesto:

- **(2.1.4.1)** Mejorar la solución al ejercicio 2.1.3.1, usando "else".
- **(2.1.4.2)** Mejorar la solución al ejercicio 2.1.3.2, usando "else".

2.1.5. Operadores lógicos: &&, ||, !

Estas condiciones se puede **encadenar** con "y", "o", etc., que se indican de la siguiente forma

Operador	Significado
&&	Y
||	O
!	No

De modo que podremos escribir cosas como

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

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.

Ejercicios propuestos:

- **(2.1.5.1)** Crear un programa que pida al usuario un número enteros y diga si es múltiplo de 2 o de 3.
- **(2.1.5.2)** Crear 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.
- **(2.1.5.3)** Crear un programa que pida al usuario tres números reales y muestre cuál es el mayor de los tres.
- **(2.1.5.4)** Crear un programa que pida al usuario dos números enteros cortos 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 operador 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 nos avisarían de que quizá estemos asignando un valor sin pretenderlo, pero no es un "error" 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 "verdadero" o "falso" (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 en C# nº 11:    */
/*  ejemplo11.cs            */
/*                          */
/*  Condiciones con if (7)  */
/*    comparacion incorrecta */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo11
{
  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:

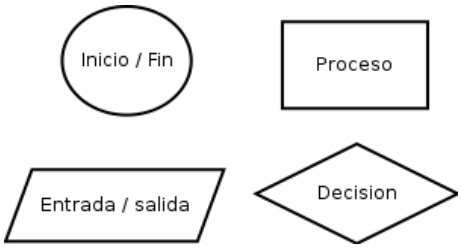
```
if (0 == numero)  ...
```

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 cuales 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.

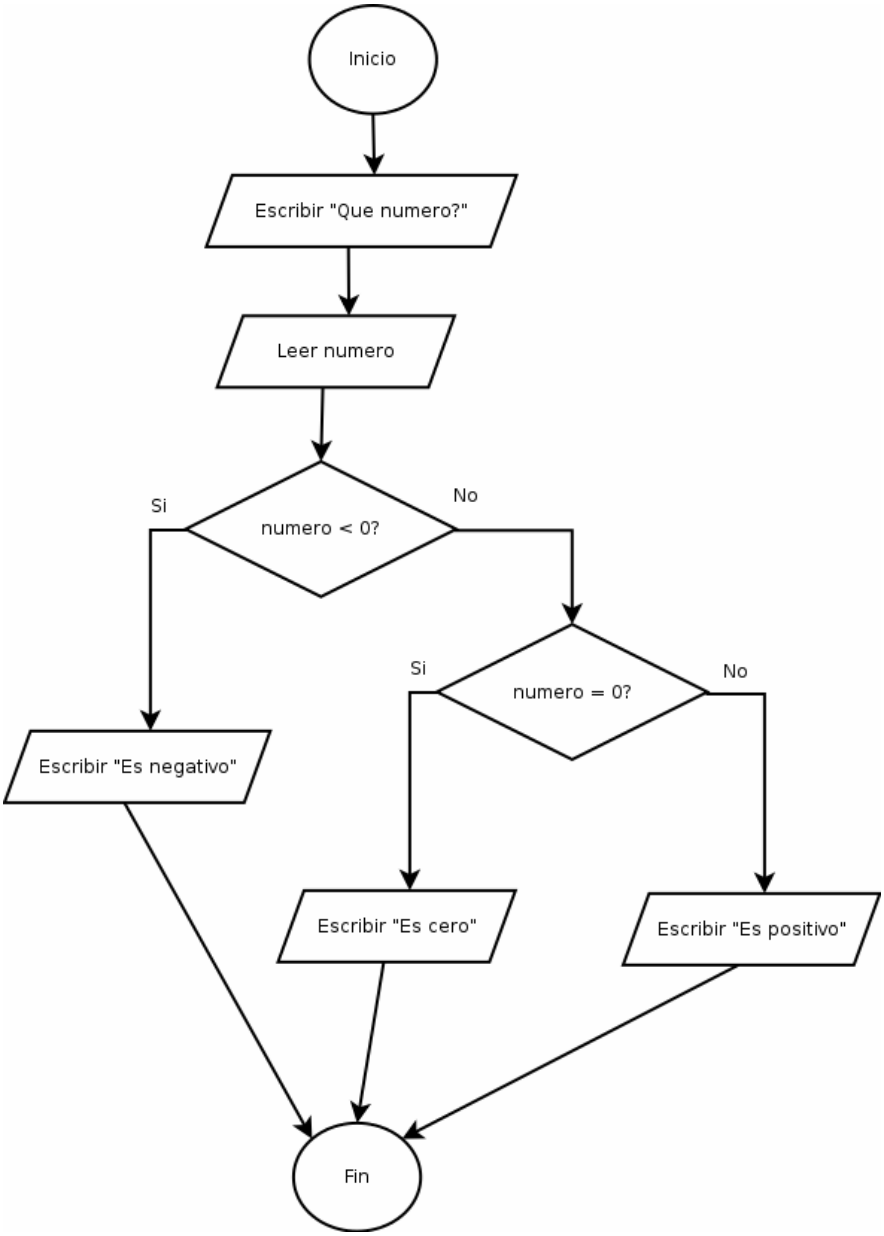
Para ayudarnos a centrarnos en el problema, existen notaciones gráficas, como los diagramas de flujo, que nos permiten ver mejor qué se debe hacer y cuando.

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



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 su lados superior e inferior horizontales, pero no tenga verticales los otros dos. Las decisiones se indican dentro de un rombo.

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



El paso de aquí al correspondiente programa en lenguaje C# (el que vimos en el ejemplo 11) debe ser casi inmediato: sabemos como leer de teclado, como 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".

Ejercicios propuestos:

- **(2.1.7.1)** Crear 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.
- **(2.1.7.2)** 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# hay otra forma de asignar un valor según se cumpla una condición o no. Es el "**operador condicional**" **?** : 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;
```

Aplicado a un programa sencillo, podría ser

```
/*----------------------------*/
/*  Ejemplo en C# nº 12:      */
/*  ejemplo12.cs              */
/*                            */
/*  El operador condicional   */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo12
{
  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);
  }
}
```

(La orden Console.Write, empleada en el ejemplo anterior, escribe un texto sin avanzar a la línea siguiente, de modo que el próximo texto que escribamos –o introduzcamos- quedará a continuación de éste).

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

```
/*----------------------------*/
```

```
/*  Ejemplo en C# nº 13:     */
/*  ejemplo13.cs             */
/*                           */
/*  Operador condicional - 2 */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*--------------------------*/

using System;

public class Ejemplo13
{
  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:

- **(2.1.8.1)** Crear un programa que use el operador condicional para mostrar un 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.
- **(2.1.8.2)** Usar el operador condicional para calcular el menor de dos números.

2.1.10. switch

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

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

Es decir, se escribe tras "**switch**" 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 se trata de ese 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 con 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 en C# nº 14:     */
/*  ejemplo14.cs             */
/*                           */
/*  La orden "switch"  (1)   */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo14
{
  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 cada caso, con la **única excepción** de que un caso no haga absolutamente nada que no sea dejar pasar el control al siguiente caso, y en ese caso se puede dejar totalmente vacío:

```
/*---------------------------*/
/*  Ejemplo en C# nº 14b:    */
/*  ejemplo14b.cs            */
/*                           */
```

```
/*  La orden "switch"  (1b)  */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*--------------------------*/

using System;

public class Ejemplo14b
{
  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 en C# nº 15:    */
/*  ejemplo15.cs            */
/*                          */
/*  La orden "switch"  (2)  */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Ejemplo15
{
  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:

- **(2.1.9.1)** Crear un programa que lea una letra tecleada por el usuario y diga si se trata de una vocal, una cifra numérica o una consonante (pista: habrá que usar un dato de tipo "char").
- **(2.1.9.2)** Crear un programa que lea una letra tecleada por el usuario y diga si se trata de un signo de puntuación, una cifra numérica o algún otro carácter.
- **(2.1.9.3)** Repetir el ejercicio 2.1.9.1, empleando "if" en lugar de "switch".
- **(2.1.9.4)** Repetir el ejercicio 2.1.9.2, 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

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.

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 { y }.

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

```
/*---------------------------*/
/*  Ejemplo en C# nº 16:     */
```

```
/*  ejemplo16.cs             */
/*                           */
/*  La orden "while"         */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo16
{
  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.

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

```
/*---------------------------*/
/*  Ejemplo en C# nº 16b:    */
/*  ejemplo16b.cs            */
/*                           */
/*  Contar con "while"       */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo16b
{
  public static void Main()
  {
    int n = 1;

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

```
}
```

Ejercicios propuestos:

- **(2.2.1.1)** Crear 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.
- **(2.2.1.2)** Crea un programa que escriba en pantalla los números del 1 al 10, usando "while".
- **(2.2.1.3)** Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "while".
- **(2.2.1.4)** Crear un programa calcule cuantas cifras tiene un número entero positivo (pista: se puede hacer dividiendo varias veces entre 10).
- **(2.2.1.5)** Crear 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.

2.2.2. do ... while

Este es el otro formato que puede tener la orden "while": la condición se comprueba **al final** (equivale a "repetir...mientras"). El punto en que comienza a repetirse se indica con la orden "do", así:

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

Al igual que en el caso anterior, si queremos que se repitan varias órdenes (es lo habitual), deberemos encerrarlas entre llaves.

Como ejemplo, vamos a ver cómo sería el típico programa que nos pide una clave de acceso y no nos deja entrar hasta que tecleemos la clave correcta:

```
/*----------------------------*/
/*  Ejemplo en C# nº 17:      */
/*  ejemplo17.cs              */
/*                            */
/*  La orden "do..while"      */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo17
{
  public static void Main()
  {

    int valida = 711;
    int clave;

    do
    {
```

```
      Console.Write("Introduzca su clave numérica: ");
      clave = Convert.ToInt32(Console.ReadLine());

      if (clave != valida)
        Console.WriteLine("No válida!");

    }
    while (clave != valida);

    Console.WriteLine("Aceptada.");

  }
}
```

En este caso, se comprueba la condición al final, de modo que se nos preguntará la clave al menos una vez. Mientras que la respuesta que demos no sea la correcta, se nos vuelve a preguntar. Finalmente, cuando tecleamos la clave correcta, el ordenador escribe "Aceptada" y termina el programa.

Como veremos un poco más adelante, si preferimos que la clave sea un texto en vez de un número, los cambios al programa son mínimos, basta con usar "string":

```
/*----------------------------*/
/*  Ejemplo en C# nº 18:      */
/*  ejemplo18.cs              */
/*                            */
/*  La orden "do..while" (2) */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo18
{
  public static void Main()
  {

    string valida = "secreto";
    string clave;

    do
    {
      Console.Write("Introduzca su clave: ");
      clave = Console.ReadLine();

      if (clave != valida)
          Console.WriteLine("No válida!");

    }
    while (clave != valida);

    Console.WriteLine("Aceptada.");

  }
}
```

Ejercicios propuestos:

- **(2.2.2.1)** Crear un programa que pida números positivos al usuario, y vaya calculando la suma de todos ellos (terminará cuando se teclea un número negativo o cero).
- **(2.2.2.2)** Crea un programa que escriba en pantalla los números del 1 al 10, usando "do..while".
- **(2.2.2.3)** Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "do..while".
- **(2.2.2.4)** Crea un programa que pida al usuario su identificador y su contraseña (ambos numéricos), y no le permita seguir hasta que introduzca como identificador "1234" y como contraseña "1111".
- **(2.2.2.5)** Crea un programa que pida al usuario su identificador y su contraseña, y no le permita seguir hasta que introduzca como nombre "Pedro" y como contraseña "Peter".

2.2.3. for

Ésta es la orden que usaremos habitualmente para crear partes del programa que **se repitan** un cierto número de veces. El formato de "for" es

```
for (valorInicial; CondiciónRepetición; Incremento)
  Sentencia;
```

Así, para **contar del 1 al 10**, tendríamos 1 como valor inicial, <=10 como condición de repetición, y el incremento sería de 1 en 1. Es muy habitual usar la letra "i" como contador, cuando se trata de tareas muy sencillas, así que el valor inicial sería "i=1", la condición de repetición sería "i<=10" y el incremento sería "i=i+1":

```
for (i=1; i<=10; i=i+1)
    ...
```

La orden para incrementar el valor de una variable ("i = i+1") se puede escribir de la forma abreviada "i++", como veremos con más detalle en el próximo tema.

En general, será preferible usar nombres de variable más descriptivos que "i". Así, un programa que escribiera los números del 1 al 10 podría ser:

```
/*----------------------------*/
/*  Ejemplo en C# nº 19:      */
/*  ejemplo19.cs              */
/*                            */
/*  Uso básico de "for"       */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo19
{
  public static void Main()
  {
```

```
    int contador;

    for (contador=1; contador<=10; contador++)
      Console.Write("{0} ", contador);

  }
}
```

Ejercicios propuestos:

- **(2.2.3.1)** Crear un programa que muestre los números del 15 al 5, descendiendo (pista: en cada pasada habrá que descontar 1, por ejemplo haciendo `i=i-1`, que se puede abreviar `i--`).
- **(2.2.3.2)** Crear un programa que muestre los primeros ocho números pares (pista: en cada pasada habrá que aumentar de 2 en 2, o bien mostrar el doble del valor que hace de contador).

En un "for", realmente, la parte que hemos llamado "Incremento" no tiene por qué incrementar la variable, aunque ése es su uso más habitual. Es simplemente una orden que se ejecuta cuando se termine la "Sentencia" y antes de volver a comprobar si todavía se cumple la condición de repetición.

Por eso, si escribimos la siguiente línea:

```
for (contador=1; contador<=10; )
```

la variable "contador" no se incrementa nunca, por lo que nunca se cumplirá la condición de salida: nos quedamos encerrados dando vueltas dentro de la orden que siga al "for". El programa no termina nunca. Se trata de un "bucle sin fin".

Un caso todavía más exagerado de algo a lo que se entra y de lo que no se sale sería la siguiente orden:

```
for ( ; ; )
```

Los bucles "for" se pueden **anidar** (incluir uno dentro de otro), de modo que podríamos escribir las tablas de multiplicar del 1 al 5 con:

```
/*---------------------------*/
/*  Ejemplo en C# nº 20:     */
/*  ejemplo20.cs             */
/*                           */
/*  "for" anidados           */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo20
{
  public static void Main()
```

```
  {
    int tabla, numero;
    for (tabla=1; tabla<=5; tabla++)
      for (numero=1; numero<=10; numero++)
        Console.WriteLine("{0} por {1} es {2}", tabla, numero,
          tabla*numero);
  }
}
```

En estos ejemplos que hemos visto, después de "for" había una única sentencia. Si queremos que se hagan varias cosas, basta definirlas como un **bloque** (una sentencia compuesta) encerrándolas entre llaves. Por ejemplo, si queremos mejorar el ejemplo anterior haciendo que deje una línea en blanco entre tabla y tabla, sería:

```
/*---------------------------*/
/*  Ejemplo en C# nº 21:     */
/*  ejemplo21.cs             */
/*                           */
/*  "for" anidados (2)       */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo21
{
  public static void Main()
  {
    int tabla, numero;
    for (tabla=1; tabla<=5; tabla++)
    {
      for (numero=1; numero<=10; numero++)
        Console.WriteLine("{0} por {1} es {2}", tabla, numero,
          tabla*numero);
      Console.WriteLine();
    }
  }
}
```

Para "contar" no necesariamente hay que usar **números**. Por ejemplo, podemos contar con letras así:

```
/*---------------------------*/
/*  Ejemplo en C# nº 22:     */
/*  ejemplo22.cs             */
/*                           */
/*  "for" que usa "char"     */
```

```
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo22
{
  public static void Main()
  {

    char letra;

    for (letra='a'; letra<='z'; letra++)
      Console.Write("{0} ", letra);

  }
}
```

En este caso, empezamos en la "a" y terminamos en la "z", aumentando de uno en uno.

Si queremos contar de forma **decreciente**, o de dos en dos, o como nos interese, basta indicarlo en la condición de finalización del "for" y en la parte que lo incrementa:

```
/*---------------------------*/
/*  Ejemplo en C# nº 23:     */
/*  ejemplo23.cs             */
/*                           */
/*  "for" que descuenta      */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo23
{
  public static void Main()
  {

    char letra;

    for (letra='z'; letra>='a'; letra--)
      Console.Write("{0} ", letra);

  }
}
```

Ejercicios propuestos:

- **(2.2.3.3)** Crear un programa que muestre las letras de la Z (mayúscula) a la A (mayúscula, descendiendo).
- **(2.2.3.4)** Crear un programa que escriba en pantalla la tabla de multiplicar del 5.
- **(2.2.3.5)** Crear un programa que escriba en pantalla los números del 1 al 50 que sean múltiplos de 3 (pista: habrá que recorrer todos esos números y ver si el resto de la división entre 3 resulta 0).

Nota: Se puede incluso declarar una nueva variable en el interior de "for", y esa variable desaparecerá cuando el "for" acabe:

```
for (int i=1; i<=10; i++) …
```

2.3. Sentencia break: termina el bucle

Podemos salir de un bucle "for" antes de tiempo con la orden "**break**":

```
/*---------------------------*/
/*  Ejemplo en C# nº 24:     */
/*  ejemplo24.cs             */
/*                           */
/*  "for" interrumpido con   */
/*  "break"                  */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo24
{
  public static void Main()
  {

    int contador;

    for (contador=1; contador<=10; contador++)
    {
      if (contador==5)
        break;

      Console.Write("{0} ", contador);
    }

  }
}
```

El resultado de este programa es:

```
1 2 3 4
```

(en cuanto se llega al valor 5, se interrumpe el "for", por lo que no se alcanza el valor 10).

2.4. Sentencia continue: fuerza la siguiente iteración

Podemos saltar alguna repetición de un bucle con la orden "**continue**":

```
/*---------------------------*/
/*  Ejemplo en C# nº 25:     */
/*  ejemplo25.cs             */
/*                           */
/*  "for" interrumpido con   */
/*  "continue"               */
/*                           */
/*  Introduccion a C#,       */
```

```
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo25
{
  public static void Main()
  {

    int contador;

    for (contador=1; contador<=10; contador++)
    {
      if (contador==5)
        continue;

      Console.Write("{0} ", contador);
    }

  }
}
```

El resultado de este programa es:

```
1 2 3 4 6 7 8 9 10
```

En él podemos observar que no aparece el valor 5.

Ejercicios resueltos:

- ¿Qué escribiría en pantalla este fragmento de código?

  ```
  for (i=1; i<4; i++) Console.Write("{0} ",i);
  ```

  Respuesta: los números del 1 al 3 (se empieza en 1 y se repite mientras sea menor que 4).

- ¿Qué escribiría en pantalla este fragmento de código?

  ```
  for (i=1; i>4; i++) Console.Write("{0} ",i);
  ```

  Respuesta: no escribiría nada, porque la condición es falsa desde el principio.

- ¿Qué escribiría en pantalla este fragmento de código?

  ```
  for (i=1; i<=4; i++); Console.Write("{0} ",i);
  ```

  Respuesta: escribe un 5, porque hay un punto y coma después del "for", de modo que repite cuatro veces una orden vacía, y cuando termina, "i" ya tiene el valor 5.

- ¿Qué escribiría en pantalla este fragmento de código?

```
for (i=1; i<4; ) Console.Write("{0} ",i);
```

Respuesta: escribe "1" continuamente, porque no aumentamos el valor de "i", luego nunca se llegará a cumplir la condición de salida.

- ¿Qué escribiría en pantalla este fragmento de código?

```
for (i=1; ; i++) Console.Write("{0} ",i);
```

Respuesta: escribe números crecientes continuamente, comenzando en uno y aumentando una unidad en cada pasada, pero sin terminar.

- ¿Qué escribiría en pantalla este fragmento de código?

```
for ( i= 0 ; i<= 4 ; i++) {
    if ( i == 2 ) continue ;
    Console.Write("{0} ",i);
}
```

Respuesta: escribe los números del 0 al 4, excepto el 2.

- ¿Qué escribiría en pantalla este fragmento de código?

```
for ( i= 0 ; i<= 4 ; i++) {
    if ( i == 2 ) break ;
    Console.Write("{0} ",i);
}
```

Respuesta: escribe los números 0 y 1 (interrumpe en el 2).

- ¿Qué escribiría en pantalla este fragmento de código?

```
for ( i= 0 ; i<= 4 ; i++) {
    if ( i == 10 ) continue ;
    Console.Write("{0} ",i);
}
```

Respuesta: escribe los números del 0 al 4, porque la condición del "continue" nunca se llega a dar.

- ¿Qué escribiría en pantalla este fragmento de código?

```
for ( i= 0 ; i<= 4 ; i++)
    if ( i == 2 ) continue ;
    Console.Write("{0} ",i);
```

Respuesta: escribe 5, porque no hay llaves tras el "for", luego sólo se repite la orden "if".

2.5. Sentencia 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, 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).

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 en C# nº 26:     */
/*  ejemplo26.cs             */
/*                           */
/*  "for" y "goto"           */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo26
{
  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:

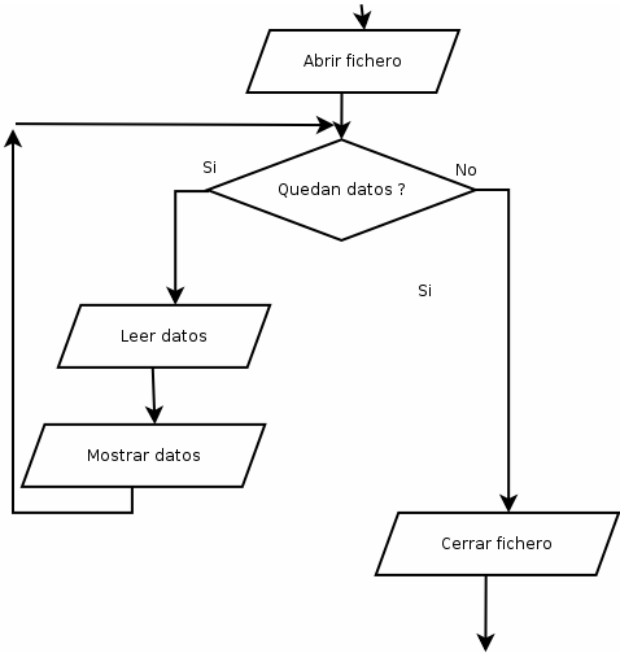
```
i vale 0 y j vale 0.
i vale 0 y j vale 2.
i vale 0 y j vale 4.
i vale 0 y j vale 6.
i vale 0 y j vale 8.
i vale 0 y j vale 10.
i vale 0 y j vale 12.
i vale 0 y j vale 14.
i vale 0 y j vale 16.
i vale 0 y j vale 18.
i vale 0 y j vale 20.
i vale 1 y j vale 0.
i vale 1 y j vale 2.
i vale 1 y j vale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

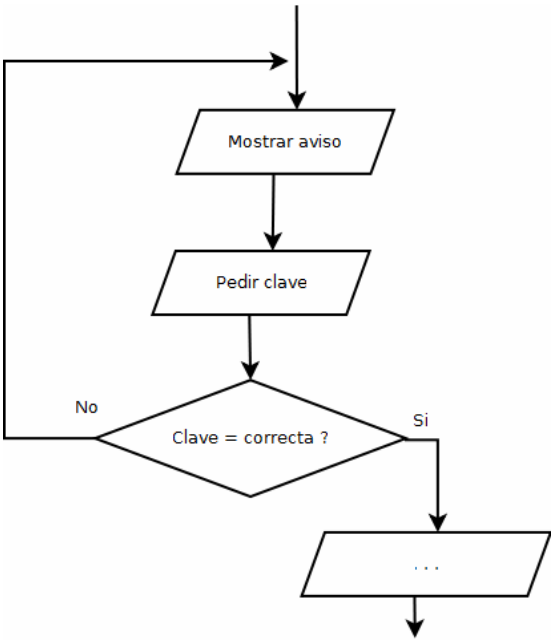
2.6. Más sobre diagramas de flujo. Diagramas de Chapin.

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" 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:

Pedir primer número
Pedir segundo número
Mostrar primer num+segundo num

Las condiciones se indican dividiendo las cajas en dos:

Pedir número n1	
Pedir número n2	
n1>n2?	
si	no
Decir "n1 es mayor"	Decir "n2 es mayor"

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

	Escribir "Teclee su clave"
	Leer clave de acceso
Mientras clave ≠ correcta	

como si se comprueba al principio (while):

Abrir fichero	
Mientras haya datos en fichero	
	Leer dato
	Mostrar dato
Cerrar fichero	

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

2.7. El caso de "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.

2.8. 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 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 cuantas veces queremos que se repita (por ejemplo: 10 veces sería "`for (i=1; i<=10; i++)`").

Ejercicios propuestos:

- **(2.8.1)** 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.
- **(2.8.2)** Crear un programa que descomponga un número (que teclee el usuario) como producto de su factores primos. Por ejemplo, 60 = 2 · 2 · 3 · 5
- **(2.8.3)** Crea un programa que calcule un número elevado a otro, usando multiplicaciones sucesivas.
- **(2.8.4)** 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:

  ****
  ****
  ****

- **(2.8.5)** 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:

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

- **(2.8.6)** 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:

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

- **(2.8.7)** 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:

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

- **(2.8.8)** Crear 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
```

2.9. 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 los posibles situaciones que pueden dar lugar a un error, a medida que estas situaciones llegan. 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 pasos hay que dar en caso de que alguno no se consiga completar. Esto permite que el programa sea más legible que la alternativa "convencional", que daba lugar a una serie de "if" encadenados.

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 excepciones que queremos "interceptar" (catch), y lo que se debe hacer en ese caso.

Lo veremos con más detalle cuando nos 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 en C# nº 26b:    */
/*  ejemplo26b.cs            */
/*                           */
/*  Excepciones (1)          */
/*                           */
/*  Introduccion a C#,       */
/*    Nacho Cabanes          */
/*---------------------------*/

using System;

public class Ejemplo26b
{

  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);
    }
  }

}
```

Se 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.
```

(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)

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 en C# nº 26c:     */
/*  ejemplo26c.cs             */
/*                            */
/*  Excepciones (1)           */
/*                            */
/*  Introduccion a C#,        */
/*    Nacho Cabanes           */
/*----------------------------*/

using System;

public class Ejemplo26b
{

  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).

Ejercicios propuestos:

- **(2.9.1)** Crear 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.