Tema 3:
Control de flujo

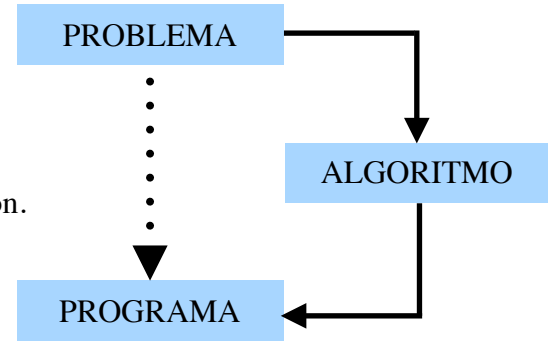
***Objetivos del tema**: En el tema anterior comenzamos a dar los primeros pasos en la programación. Sin embargo, ahora mismo sólo sabemos escribir programas que se ejecutan de modo secuencial: todas las instrucciones del programa son ejecutadas, y cada una de ellas se ejecuta una sola vez. Esto es muy poco flexible. En este tema veremos cómo solucionarlo introduciendo sentencias de control de flujo.*

CEU

1. Algoritmos

La resolución de un problema mediante un ordenador consiste en, partiendo de una especificación del problema, construir un programa que lo resuelva. Los procesos necesarios para la creación de un programa son:

1. Especificación y análisis del problema en cuestión.
2. Diseño de un algoritmo que resuelva el problema.
3. Codificación del algoritmo en un lenguaje de programación.
4. Validación del programa.



Un **algoritmo** es una secuencia ordenada de operaciones tal que su ejecución resuelve un determinado problema. Las características fundamentales que debe tener todo algoritmo son:

- Debe ser preciso, es decir, indicar el orden de realización de cada paso.
- Debe estar definido, esto es, si se ejecuta varias veces partiendo de las mismas condiciones iniciales debe obtenerse siempre el mismo resultado.
- Debe ser finito (debe tener un número finito de pasos).
- Debe ser independiente del lenguaje de programación que se emplee para implementarlo.

En cualquier algoritmo se pueden distinguir tres partes: la entrada de datos (la información sobre la cual se va a efectuar operaciones), procesamiento y salida del resultado (la información que debe proporcionar).

Un ejemplo clásico de algoritmo es el algoritmo de Euclides, que sirve para encontrar el máximo común divisor de 2 números enteros positivos A y B. Veámoslo como ejemplo:

- **Paso 1.** Tomar el número mayor como dividendo y el menor como divisor.
- **Paso 2**. Calcular el resto de la división entera.
- **Paso 3**. Si el resto es igual a cero entonces ir al Paso 4. En caso contrario, tomar el divisor (número menor) como nuevo dividendo y el resto como divisor y volver al Paso 2.
- **Paso 4**. El m.c.d. es el divisor de la última división.

El algoritmo de Euclides para calcular el m.c.d. es definido, prevé todas las situaciones posibles para el resto la división (ser cero o diferente de cero); no es ambiguo: las acciones de dividir y comparar el resto con cero son claras y precisas, al igual que la obtención del número mayor; es finito, contando con cuatro pasos diferentes; y acaba en un tiempo finito cuando se cumple la condición del Paso 3 que hace avanzar al Paso 4, siendo este el punto de fin. Como ejemplo podemos considerar en el caso en el que A=9 y B=24;

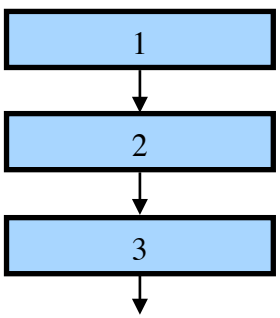
- **Paso 1**-> D:=24, d:=9
- **Paso2**-> R:=6(24%9)
- **Paso 3**-> D:=9, d:=6
- **Paso2**-> R:=3
- **Paso 3**-> D:=6, d:=3
- **Paso2**-> R:=0
- **Paso 4**-> m.c.d. := 3

1.1. Estructuras básicas en un algoritmo

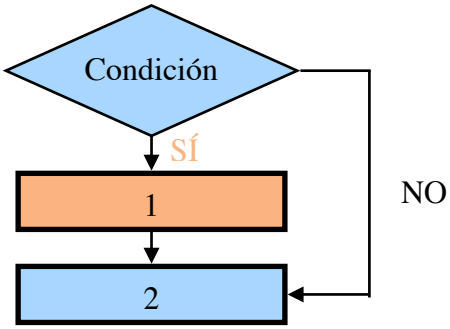
Un programa se puede definir como una secuencia ordenada de instrucciones cuyo propósito es realizar una determinada tarea. Por tanto, aparece el concepto de **flujo de ejecución** de un programa: este será el orden que siguen dichas instrucciones durante la ejecución del programa.

En principio, el flujo de ejecución de un programa será el orden en el cual se escriban las instrucciones (ejecución secuencial). Sin embargo, este esquema de ejecución impone serias limitaciones: a menudo hay ciertas porciones de código que sólo se deben ejecutar si se cumplen ciertas condiciones y tareas repetitivas que hay que ejecutar un gran número de veces. Por ello, se han definido una serie de estructuras de programación, independientes del lenguaje concreto que se está utilizando, que permiten crear programas cuyo flujo de ejecución sea más versátil que una ejecución secuencial. Los tipos de estructuras básicas que se pueden emplear en un algoritmo son:

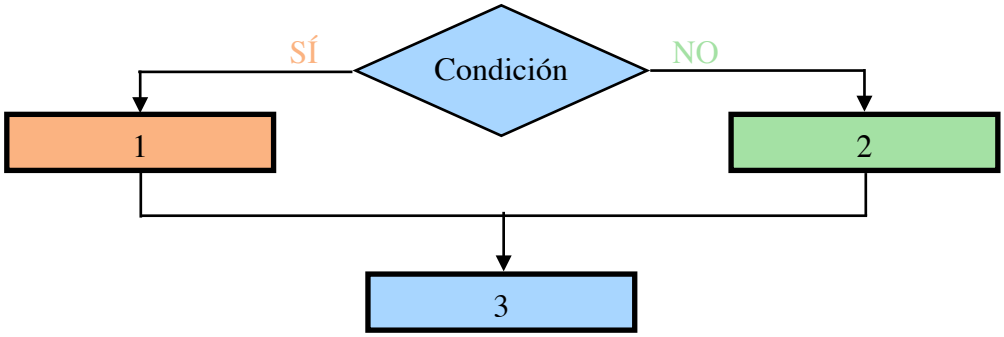
- **Secuencia**: constituido por 0, 1 o N instrucciones que se ejecutan según el orden en el que han sido escritas. Es la estructura más simple y la pieza más básica a la hora de componer estructuras.



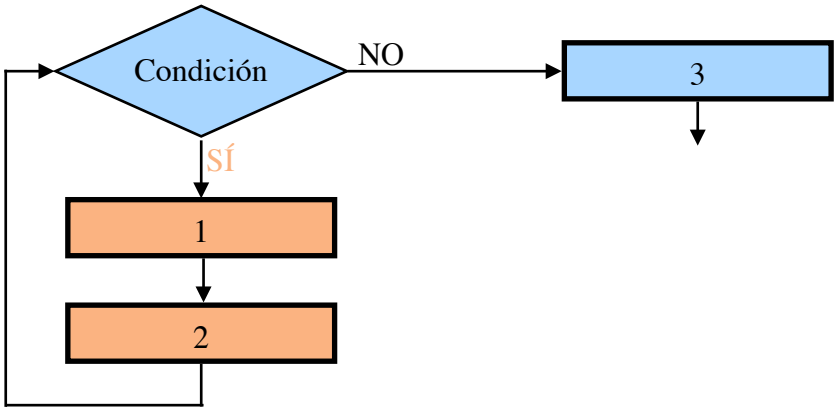
- **Selección, bifurcación o alternativa**: consta de una instrucción especial de decisión. La sentencia de decisión genera un resultado delimitado dentro de un rango seleccionado (generalmente verdadero o falso) y, dependiendo del resultado obtenido, se ejecuta o no una secuencia de instrucciones delimitada.



También puede contar con una respuesta alternativa si no cumpliese la condición:



- **Iteración, bucle o repetición**: consta de una instrucción especial de decisión y de una secuencia. La instrucción de decisión sólo genera dos tipos de resultado (verdadero o falso) y la secuencia de instrucciones se ejecutará de modo reiterativo mientras que la instrucción de decisión genere el resultado verdadero; en caso contrario finalizará la ejecución de la secuencia. Los bucles pueden tener la instrucción de decisión al principio o al final. Si la condición está al final, el bucle siempre se ejecuta al menos una vez.



2. Sentencias

Las sentencias de un programa especifican una acción a realizar. Las sentencias en C# suelen finalizar con un punto y coma. Una expresión es una secuencia de operadores y operandos que especifica un valor. Por ejemplo, la sentencia 4+5; está formada por la expresión 4+5, que utiliza el operador suma sobre dos operandos constantes enteras y cuyo valor es 9.

En un programa C# hay una parte de declaraciones y otra de instrucciones. En la parte de declaraciones se pueden utilizar sentencias para:

- Declaración de tipos de datos.
- Declaración de variables.
- Declaración de funciones.

Y en la parte de instrucciones, se utilizan tres tipos principales de sentencias:

- Sentencias de asignación.
- Sentencias de control.
- Llamadas a funciones.

3. Sentencias de control de flujo

Las sentencias que se tratan en este apartado permiten realizar las estructuras de bifurcación y bucle que forman parte de los algoritmos. Permiten, por tanto, construir programas cuyo flujo de ejecución sea más versátil que la ejecución secuencial.

3.1. Bloque de sentencias

Se denomina bloque de sentencias a un grupo de sentencias cualquiera encerradas entre llaves { }. Los bloques de sentencias sirven para agrupar varias sentencias bajo el control de otra (bifurcación, bucle). **En cualquier parte del programa C# donde pueda ir una sentencia, puede ir también un bloque de sentencias.**

3.2. Sentencia condicional simple: if

La sentencia *if* se utiliza para hacer una estructura de selección o bifurcación en el flujo del programa. Puede aparecer de modo aislado o junto con la sentencia *else*:

```
if(condición)
   sentencia;
```

```
if(condición)
    sentencia1;
else
    sentencia2;
```

En el ejemplo de la izquierda, si la *condición* es cierta, se ejecuta *sentencia*; si es falsa, se salta. En cualquier caso, después se ejecutan las sentencias que sigan al *if*. En el de la derecha, si la *condición* es cierta, se ejecuta *sentencia1,* si es falsa, se ejecuta *sentencia2*. Después se ejecutan las sentencias que sigan al *if*. Obsérvese que o bien se ejecutan las sentencias de la parte *if*, o las de la parte *else*, nunca ambos grupos. Como se aprecia en el ejemplo de la izquierda, la parte del *else* es opcional. En caso de ser necesarias varias sentencias en la parte *if* o en la *else* se utilizará un bloque de sentencias, como se muestra en el siguiente ejemplo:

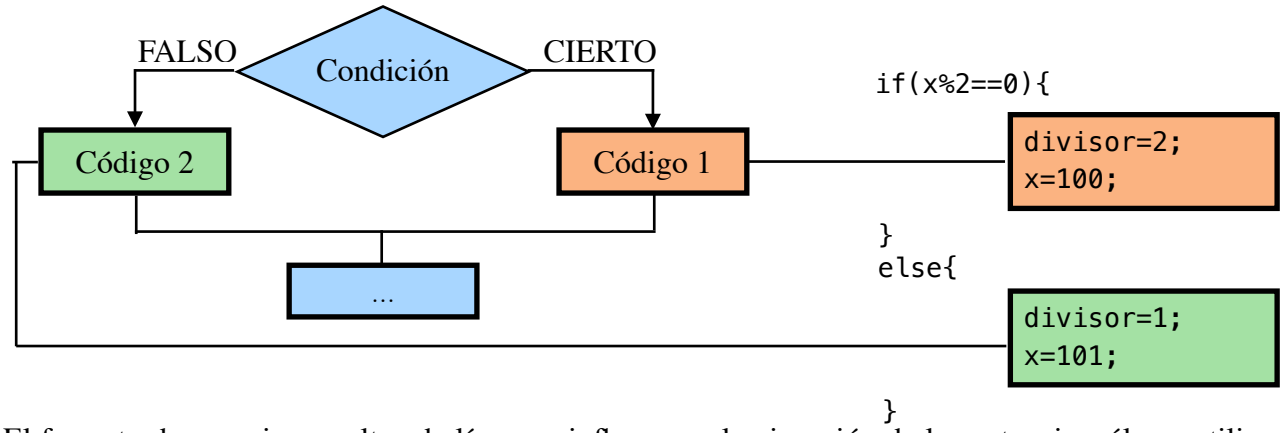
```
if (a>=0) {
    Console.WriteLine(“a es positivo");
    b=a;
 } else
    Console.WriteLine("a es negativo”);
```

En el ejemplo anterior, la rama *if* utiliza un bloque de sentencias, que va desde la llave { hasta la }. La rama *else* comienza justo después de dicha palabra y termina en el primer punto y coma después de ella. Para saber el alcance de una sentencia *if* o *else* se sigue la siguiente regla: si no se utiliza un bloque de sentencias, tanto la rama *if* como la *else* terminan en el primer punto y coma después de dichas palabras, es decir, si no se utilizan llaves ({}) el *if* y el *else* solo contarán con una sentencia (la primera). Por tanto, en el ejemplo siguiente:

```
mayor_edad =0;
if (edad>18)
    mayor_edad = edad – 18;
 Console.WriteLine(“Llevas “+ mayor_edad +” años siendo +18 ");
```

la sentencia *Console.WriteLine* no está comprendida dentro del *if*, ya que éste termina en el punto y coma que sigue al cálculo de *mayor_edad*. Si se coloca una sentencia *else* después de *Console.WriteLine*, el compilador no la reconocerá como parte del *if* y se producirá un error. Aunque no es obligatorio, es recomendable emplear siempre las llaves detrás del *if* y del *else*, de este modo se incrementa la legibilidad del código y disminuye la posibilidad de cometer errores en el futuro cuando, por ejemplo, añadamos más sentencias que estén bajo la acción del *if*.

A continuación mostramos el diagrama de flujo correspondiente con un condicional tipo *if-else*:



El formato de espacios o saltos de línea no influyen en la ejecución de la sentencia, sólo se utilizan para que la estructura del programa resulte más clara al leerlo. Normalmente, las sentencias que se ejecutan bajo el control de otra se separan dos espacios hacia la derecha o una tabulación. Por esta razón , la sentencia *Console.WriteLine()* del último ejemplo, que no pertenece al *if*, comienza dos espacios más atrás.

3.2.1 Expresiones condición

Se denomina expresión condicional a la que se evalúa como verdadero o falso. Generalmente relaciona 2 o más expresiones mediante operadores relacionales y/o lógicos. Los operadores relacionales se utilizan para comparar expresiones numéricas, como b*b - 4*a*c > 0 o K<=sin(x). Los operadores lógicos se utilizan para hacer una expresión condicional compleja a partir de condiciones más simples, por ejemplo, la condición de la sentencia *if* siguiente:

```
|  if (numero>0 && numero%2==0) Console.WriteLine("Numero par y positivo");
```

En C#, el condicional deben de ser siempre valores booleanos (true o false) o expresiones que tengan este como resultado (Ej: a>5). Esto significa que no se permite utilizar cualquier expresión como condición en una sentencia *if*. Ejemplo:

```
    bool acierto = true;
    if(acierto) Console.WriteLine("Es cierto");
    if (!acierto) Console.WriteLine("Es falso");
```

En este ejemplo, vemos el símbolo not (!), muy utilizado en la programación. Este operador invierte el valor de una expresión booleana. El segundo condicional es equivalente a usar un *else* en el primer *if*, ya que ambos representan escenarios opuestos. En otras palabras, si la primera condición no se cumple, la segunda, que es su negación, si lo hará.

Es frecuente confundir el operador de asignación (=) con el de comparación (==). En las expresiones lógicas hay que utilizar el de comparación, si no, la sentencia no funcionará correctamente. Por ejemplo, en la sentencia:

```
|  if (a=5) Console.WriteLine("A es 5");
```

Dará un error porque no se está comprobando nada en el *if*, la manera correcta de comprobar si *a* es igual a 5 sería:

```
 |  if (a==5) Console.WriteLine("A es 5");
```

Ejercicio de clase: Suma tres números reales. Imprime por pantalla un mensaje en el que muestre si es positivo o no lo es.

3.2.2 Sentencias if anidas

Se dice que una sentencia *if* está anidada si se ejecuta dentro de una de las ramas de otra sentencia *if*. El programa siguiente muestra un grupo de sentencias *if* anidadas para traducir una nota numérica a un mensaje.

```
using System;
public class Ejemplo1 {
    public static void Main(string[] args)
    {
        float nota;
        Console.WriteLine("Introduce la nota de tu alumno:");
        string leido = Console.ReadLine();
        nota = float.Parse(leido);
        if (nota < 5)
        {
            Console.WriteLine("Suspenso");
        }
        else
        {
            if (nota < 7)
            {
```

```
            Console.WriteLine("Aprobado");
        }
        else if (nota < 9)
        {
            Console.WriteLine("Notable");
        }else if (nota < 9.5)
        {
            Console.WriteLine("Sobresaliente");
        }
        else
        {
            Console.WriteLine("Matrícula de hornor");
        }
    }
}}
```

3.3. Bucles en C#

Un bucle está formado por un conjunto de sentencias que se repiten bajo el control de una condición. Por lo tanto, hay que buscar una condición adecuada para que las sentencias del bucle se repitan constantemente hasta que la condición se cumpla. Los bucles pueden llevar la condición al principio o al final. Los bucles con la condición al final se realizan con la sentencia *do-while* y los que llevan la condición al principio con las sentencias while, *for* y *foreach*.

3.3.1 Bucle do-while

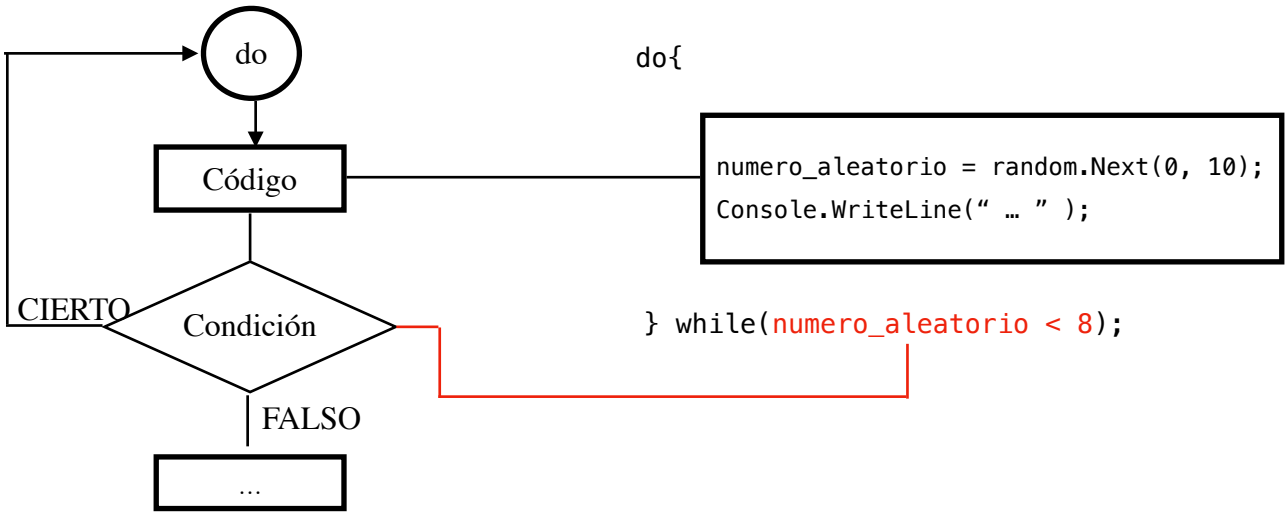
La forma general de un bucle *do-while* es:

```
do{
    sentencias;
 } while (condición);
```

En caso de querer incluir más de una sentencia, se utiliza un bloque de sentencias. El principio del bucle es la sentencia *do* y el final la sentencia *while*. Las sentencias entre ambas son el cuerpo del bucle. La ejecución de este bucle es como sigue: se ejecutan las sentencias que siguen al *do*, hasta el *while*. Se comprueba la condición. Si es cierta, se vuelve a la primera sentencia del bucle. Si es falsa, se sale del bucle y se ejecuta la sentencia que sigue al *while*. Como la condición se comprueba al final, el bucle siempre se ejecuta al menos 1 vez. Veamos un ejemplo:

```
using System;
public class Ejemplo2{
    public static void Main(string[] args) {
        int numero_aleatorio=0;
        //inicializamos una rutina de creación de numeros aleatorios
        Random random = new Random();
        do{
            //generamos un numero aleatorio entre 0 y 10
            numero_aleatorio = random.Next(0, 10);
            Console.WriteLine("Numero aleatorio es: " + numero_aleatorio);
            //comprobamos si el numero aleatorio es menor que 8 se vuelve al do
        } while (numero_aleatorio < 8);
    }
}
```

En este ejemplo vemos un bucle, que va a repetir sus sentencias internas hasta que el numero aleatorio sea mayor de 8. En las sentencias del bucle aprendemos como generar un número aleatorio entre 0 y 10 en C#. A continuación presentamos el diagrama de flujo correspondiente a este programa:



3.3.2 Bucle while

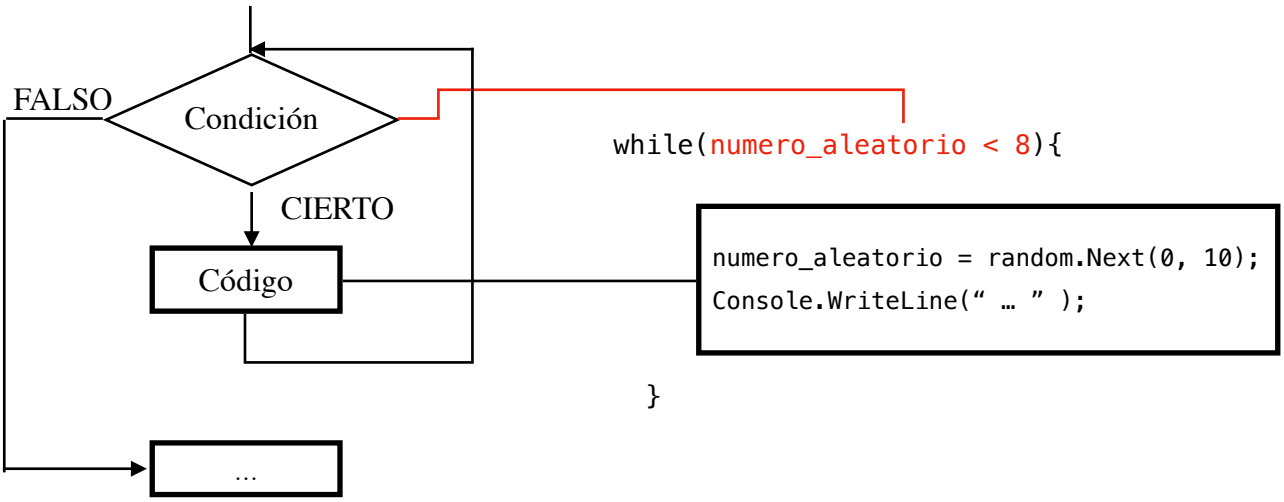
Es similar al anterior, pero con la condición al principio. En este caso, se prueba la condición antes de ejecutar la primera vez el bucle. Si la condición es falsa inicialmente, el bucle no se ejecuta ninguna vez:

```
while (condición){
    sentencias;
 }
```

Nuevamente, en caso de requerir más de una sentencia, se utiliza un bloque de sentencias. Mostramos el código del ejemplo anterior rehecho con este nuevo tipo de bucle:

```
using System;
public class Ejemplo3{
    public static void Main(string[] args) {
        int numero_aleatorio=0;
        //inicializamos una rutina de creación de numeros aleatorios
        Random random = new Random();
        //comprobamos si el numero aleatorio es menor que 8
        // al principio esta inicializado a 0 por lo que si cumple la condición
        while (numero_aleatorio < 8){
            //generamos un numero aleatorio entre 0 y 10
            numero_aleatorio = random.Next(0, 10);
            Console.WriteLine("Numero aleatorio es: " + numero_aleatorio);
        }
    } }
```

Su diagrama de flujo es:



Ejercicio de clase: Incrementa una variable el doble de su número hasta que alcance o supere 500.

3.3.3 Bucle for

Es un bucle con la condición al principio, como el while, pero que además permite incluir una sentencia de inicialización del bucle y otra de actualización del mismo. Estos tres elementos se separan por un punto y coma.

```
for (expresion_inicio; condición; expresion_actualización) {
    sentencias;
 }
```

Todo bucle *for* es equivalente a un bucle *while* con la expresión inicial antes del bucle y la expresión de actualización al final del mismo.

```
expresion_inicio
while (condición) {
   sentencias;
   expresion_actualización;
}
```

La expresión de inicialización se ejecuta una sola vez al comenzar el bucle. La expresión de actualización se ejecuta cada vez que se llega al final del bucle, antes de comprobar la condición. Debido a esta característica, el bucle *for* se utiliza con frecuencia para bucles con contadores fijos

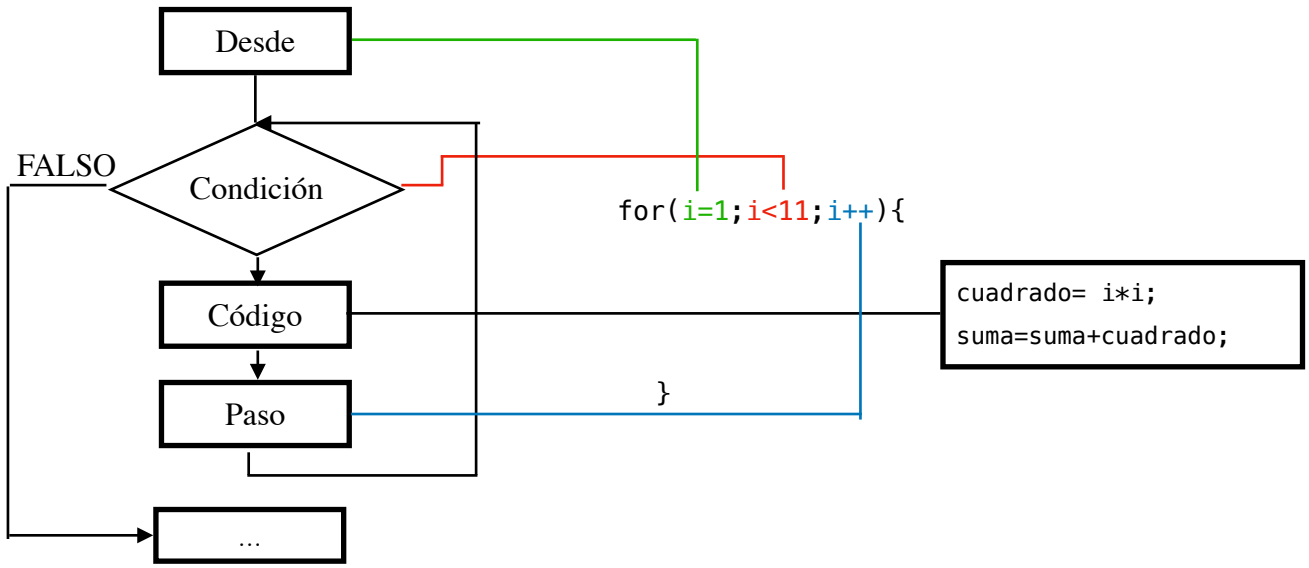
como, por ejemplo, este bucle *for* para sumar los cuadrados de los números de 1 a 10 y bucle *while* equivalente:

```
for (int i=1; i<11 ; i++) {
    cuadrado= i*i;
    suma=suma+cuadrado;
 }
```

Esto sería lo mismo que hacer esto con *while*:

```
i=1;
while (i<11) {
    cuadrado= i*i;
    suma=suma+cuadrado;
    i++;
}
```

La utilización de *for* en lugar de *while* se hace sólo por claridad, al estar agrupadas las sentencias de control del bucle en una misma línea. A continuación mostramos el diagrama de flujo correspondiente con este bucle:



En un bucle for puede omitirse cualquiera de las dos sentencias, o la condición, o todas ellas; por ejemplo, es válido el siguiente bucle:

```
for ( ; ; ) {
    sentencia1;
    sentencia2;
 }
```

En este caso, el bucle no termina nunca, salvo que una sentencia interna break o exit (que veremos más tarde en este tema) lo haga terminar. Si se desea que en un bucle se inicialice o actualice más de una variable se utiliza la secuencia de expresiones. Consiste en separar las diferentes sentencias por una coma (,), como en el ejemplo siguiente:

```
  for ( s=0, i=0, a=1 ; i<10 ; i++, a= i-1 ,s= j+3) {
      s=s+1;
      a=a+1;
   }
```

A continuación mostramos un programa que imprime los caracteres ASCII desde el 40 al 127, junto con su valor entero equivalente empleando un bucle for:

```
using System;
public class Ejemplo3{
    public static void Main(string[] args) {
        char caracter;
        for (int i = 40; i < 127; i++)
        {
            caracter = (char)i;
            Console.WriteLine("El valor: " + i + " se corresponde con el
carácter: " + caracter);
        }
    }
}
```

Ejercicio de clase: Imprime por pantalla todos los números del 1 al 100 que son pares.

3.3.4 Bucles anidados

Se dice que un bucle está anidado cuando está dentro de otro bucle más amplio. En esta estructura, el bucle interno se ejecuta completamente (todas las repeticiones) para cada paso del bucle externo. Este tipo de construcciones son muy empleadas para procesar matrices, y para otros programas, como el siguiente, que escribe en pantalla una tabla:

```
using System;
public class Ejemplo4 {
    public static void Main(string[] args) {
```

```
        double resultado;
        for (int numero = 1; numero < 4; numero++)
        {
            for (int exponente = 0; exponente < 10; exponente++)
            {
                resultado = Math.Pow(numero, exponente);
                Console.WriteLine(numero + "^"+ exponente+ "= "+ resultado);
            }
        }
    } }
```

El bucle externo hace variar la base. Para cada base (cada paso del bucle externo), el bucle interno hace variar el exponente de forma que tome todos sus valores. Esto se repite para todos los valores de la base. La función Math.Pow(x,y) calcula x elevado a y.

3.3.5 Sentencias break y continue

A veces resulta necesario abandonar un bucle por alguna condición que se verifica cuando se está dentro del bloque de código, sin esperar a evaluar la condición del bucle. Para salir de un bucle puede utilizarse la sentencia **break**, en combinación con un *if*, como en el siguiente ejemplo:

```
using System;
public class Ejemplo5{
    public static void Main(string[] args) {
        int numero;
        Random random = new Random();
        int secreto = random.Next(0,10);
        Console.WriteLine("Adivina el numero secreto del 1 al 10”);
        Console.WriteLine(“Para salir pulse el 0");
        do {
            Console.WriteLine("\nDame un numero del 1 al 10 :  ");
            string leido= Console.ReadLine();
            numero = int.Parse(leido);
```

```
            if (numero == 0) break;   /* == corta el bucle == */
            if (numero != secreto){
                Console.WriteLine("Ese no es el numero secreto.");
                if (secreto > numero)
                    Console.WriteLine("El numero secreto es mayor.\n");
                else
                    Console.WriteLine("El numero secreto es menor.\n");
            }
            else
                Console.WriteLine(" Has acertado ! \n");
        } while (numero != secreto);
    }
}
```

La sentencia **continue** se utiliza dentro de un bucle para iniciar una nueva iteración del mismo, saltándose las sentencias que la siguen. Es equivalente a un salto al final del bucle, por tanto, si es un bucle for, se ejecuta la sentencia de actualización (aunque está colocada al principio, se ejecuta al final del bucle) y si es un bucle *do-while* se comprueba la condición.

El siguiente programa añade al ejemplo anterior un número máximo de intentos, mientras estos no se consumen se sigue haciendo las iteraciones del bucle, si se llega al máximo, se ejecutarán las dos últimas lineas después del continue, y se terminará el programa.

```
using System;
public class Ejemplo7{
    public static void Main(string[] args){
        int numero;
        Random random = new Random();
        int secreto = random.Next(0, 10);
        int intentos = 0;
        do{
            Console.WriteLine("\nDame un numero entero:  ");
            string leido = Console.ReadLine();
            numero = int.Parse(leido);
            if (numero == 0) break;   /* == corta el bucle == */
            if (numero != secreto){
```

```
                intentos++;
                Console.WriteLine("Ese no es el numero secreto.");
                if (secreto > numero)
                    Console.WriteLine("El numero secreto es mayor.\n");
                else
                    Console.WriteLine("El numero secreto es menor.\n");
            }
            else
                Console.WriteLine(" Has acertado ! \n");
            if (intentos < 2) continue;
            Console.WriteLine(" Has consumido todas tus oportunidades ! \n");
            break;
        } while (numero != secreto);
    }}
```

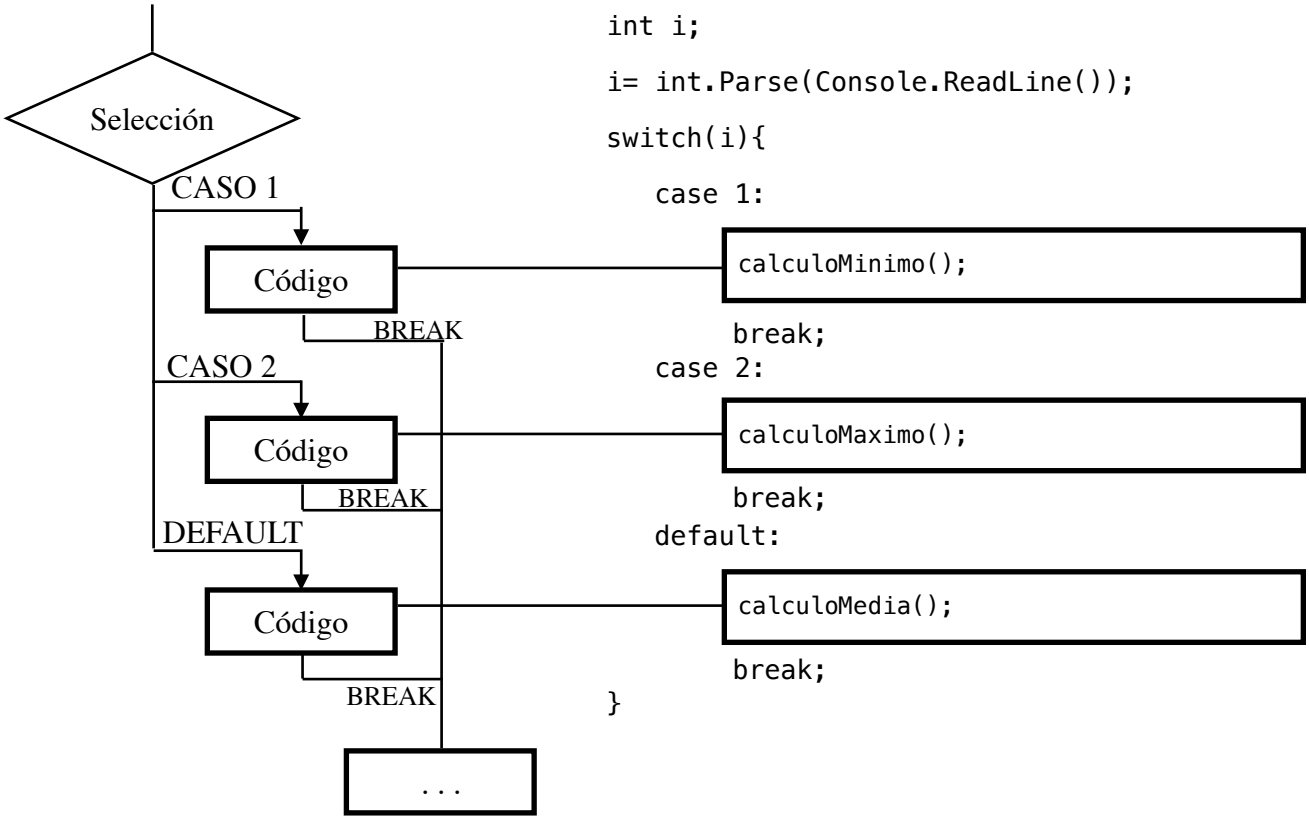
3.4. Bifurcación múltiple: sentencia switch

La bifurcación múltiple permite que un programa tome distintas acciones según el valor de una expresión de tipo int o char. Se realiza mediante la sentencia switch, que tiene la siguiente sintaxis:

```
switch ( selector ) {
    case valor1: Grupo de sentencias1; break;
    case valor2: Grupo de sentencias2; break;
    case valor3: Grupo de sentencias3; break;
    . . .
    default: statement;
 }
```

Dentro de los paréntesis del switch se coloca la expresión. Para cada valor para el cual se desea ejecutar un conjunto diferente de sentencias se pone una sentencia **case** seguida de dicho valor (en forma de constante, no sirve una variable). Cuando se ejecuta la sentencia switch, se compara el valor de la expresión con cada constante por orden, y si coincide con una, se ejecutan las sentencias que siguen al case. Una vez que se verifica una sentencia case, **se ejecutan las sentencias asociadas y las de todos los case que haya a continuación**.

Para cortar la ejecución de las sentencias y salir del switch, puede ponerse donde resulte oportuno una sentencia **break**, esto es, un salto incondicional fuera del switch. Si para varios valores de la expresión deben ejecutarse las mismas sentencias, pueden ponerse los case seguidos, con las sentencias en el último de ellos. La etiqueta default es opcional y, si se llega a ella (si no se sale con un break), siempre se ejecuta sea cual sea el valor de la expresión. A continuación mostramos el diagrama de control de flujo correspondiente con un switch:



Veamos un ejemplo de uso de esta estructura de control de flujo:

```
using System;
public class Ejemplo8
{
    public static void Main(string[] args)
      {
        int mes,dias;
        Console.WriteLine("Escriba el numero del un mes para saber sus dias");
        mes = int.Parse(Console.ReadLine());
```

```
        switch (mes)
        {
            case 2: dias = 28; //febrero
                break;
            case 4:           //abril
            case 6:           //junio
            case 9:           //septiembre
            case 11: dias = 30; //noviembre
                    break;
            default: dias = 31; //los demas
                break;
        }
        Console.WriteLine("El mes "+ mes+ " tiene "+ dias +" dias");
    }
}
```

Cuando el mes es 2, días toma el valor 28 y con el break se sale del switch. Si es 4, 6, 9 o 11, toma el valor 30 y se sale. Cualquier otro valor hace que tome el valor 31.

Ejercicio de clase: Genera caracteres aleatorios, el programa debe de determina si los caracteres generados son consonantes o vocales. Repítelo 10 veces.

Ejercicio de clase: Pide al usuario que te introduzca dos números y que operación quiere realizar entre ellos. Utilizando switch y el símbolo del operador, realiza las operaciones. Repítelo 10 veces.

3.5. El operador condicional

Aunque estrictamente hablando no es una sentencia de control de flujo sino un operador, el operador condicional permite elegir de entre dos expresiones cuál de ellas se ejecuta y devuelve el resultado de la ejecución de dicha expresión. Este operador es muy similar a un *if-else*. Veamos su sintaxis:

```
| condición ? expresión1 : expresión2
```

esta sentencia es análoga a:

```
if ( expresion1 ) {
   expresion2;
}else{
   expresión3;
}
```

Veamos un ejemplo donde se emplea el operador condicional para imprimir el mayor y el menor de dos números pedidos al usuario:

```
using System;
public class Ejemplo9
{
    public static void Main(string[] args)
    {
        String leido,leido2;
        Console.WriteLine("Primer número: ");
        leido=Console.ReadLine();
        Console.WriteLine("Segundo número: ");
        leido2 = Console.ReadLine();
        int n1 = int.Parse(leido);
        int n2 = int.Parse(leido2);

        int mayor = (n1 > n2) ? n1 : n2;
        int menor = (n1 < n2) ? n1 : n2;
        Console.WriteLine("El mayor es: "+ mayor);
        Console.WriteLine("El menor es: "+ menor);
}}
```

El empleo de un operador condicional es más compacto que el empleo de la estructura *if-else*; sin embargo la segunda resulta mucho más legible que la primera, por ello alguna gente recomienda no emplear el operador condicional.

4. Ejercicios

1. Calcular el factorial de un número introducido por teclado. Antes de realizar el cálculo deberá comprobarse que el número es positivo y en caso contrario se imprimirá un mensaje de error.

2. Calcular la suma de los 100 primeros números naturales.

3. Sumar los números pares menores que N por un lado y los impares menores que N por otro lado. N es un valor introducido por el usuario.

4. Calcular la suma de todos los múltiplos de 5 comprendidos entre 1 y 100. Calcular además cuantos hay y visualizar cada uno de ellos.

5. Escribe un programa que calcule el mínimo y el máximo de una lista de números enteros positivos introducidos por el usuario. La lista finalizará cuando se introduzca un número negativo.

6. Escribe un programa que lea un mes en número (1 para enero, 2 para febrero, etc.) y un año e indique el número de días de ese mes. Recuerda, el año es bisiesto si es divisible por cuatro, excepto cuando es divisible por 100, a no ser que sea divisible entre 400.

7. Escribe un programa que visualice por pantalla la tabla de multiplicar de los 10 primeros números naturales.

8. Escribe un programa que solicite continuamente números enteros al usuario; el programa se detendrá cuando la suma de los números enteros introducidos por el usuario supere 1000.

9. Implementa un algoritmo que imprima el numero de veces que es posible dividir un numero entre 2. Dicho número se pedirá al usurario.

10. Escribe un programa que lea un número entero del teclado y diga si ese número es o no primo.

11. Escribe un programa que muestre por pantalla los primeros 10 números generados al azar del 0 al 20 para que no haya repeticiones.

12. Escribe un programa que dada una cantidad de dinero indique cuál es la cantidad mínima de monedas (indicando el número de cada tipo de moneda) que se puede emplear para representar esa cantidad de dinero.

13. Escribe un programa que genere una cadena de vocales. EL programa deberá pedir al usuario el tamaño de la cadena, y a continuación generará aleatoriamente una cadena de dicho tamaño y la mostrará en pantalla.

14. Escribe un programa que realice las operaciones de suma, resta, multiplicación y división. El programa deberá mostrar siguiente menú:

    Programa calculadora:

    1) Realizar una suma.

    2) Realizar una resta.

    3) Realizar una multiplicación.

    4) Realizar una división.

    5) Salir del programa.

    Introduzca su opción: