

CURSO DE PROGRAMACIÓN.NET

M.374.001.003



ESTRUCTURAS DE CONTROL

Index

Estructuras condicionales.....	3
Estructura if.....	3
Estructura switch.....	5
Operador ternario.....	6
Bloques y ámbito de las variables.....	7
Estructuras iterativas.....	8
Bucle while.....	8
Bucle for.....	9
Bucle foreach.....	11
Break y continue.....	11
Bucles infinitos.....	12
Bucles anidados.....	12

Estructuras condicionales

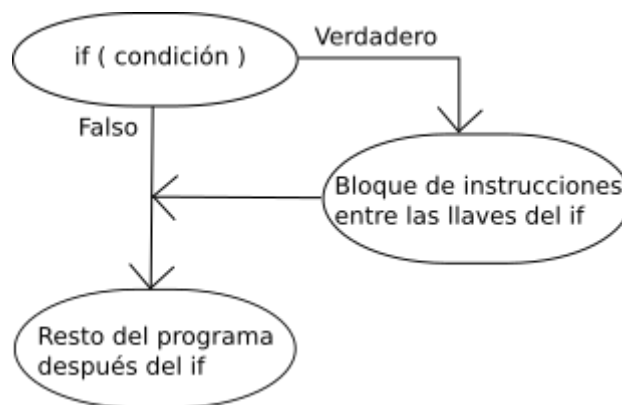
Por ahora hemos aprendido a asignar valores a variables, hacer operaciones matemáticas y lógicas, y escribir y leer de la consola. Sin embargo, todavía no podríamos hacer un programa que tomase decisiones en función de ciertos valores, o que comprobase si un valor almacenado en una variable es correcto, entre otras cosas. Para ello tenemos las estructuras condicionales.

Estructura if

La estructura if (“si” condicional en castellano), permite a partir de una condición o un valor booleano, verdadero o falso, ejecutar cierto código. La sintaxis es la siguiente:

```
if (condicion)
{
    // Instrucciones a ejecutar si la condición es "True"
}
```

De esta forma podemos incluir en el programa un conjunto de instrucciones que sólo se ejecutarán si la condición se cumple. Si no se cumple, no ejecutará ninguna instrucción dentro del bloque delimitado por las llaves.



```
Console.Write("Dime tu nombre: ");
string nombre = Console.ReadLine();
if (nombre == "") {
    Console.WriteLine(";El nombre no puede estar vacío!");
}
Console.WriteLine($"Hola {nombre}");
```

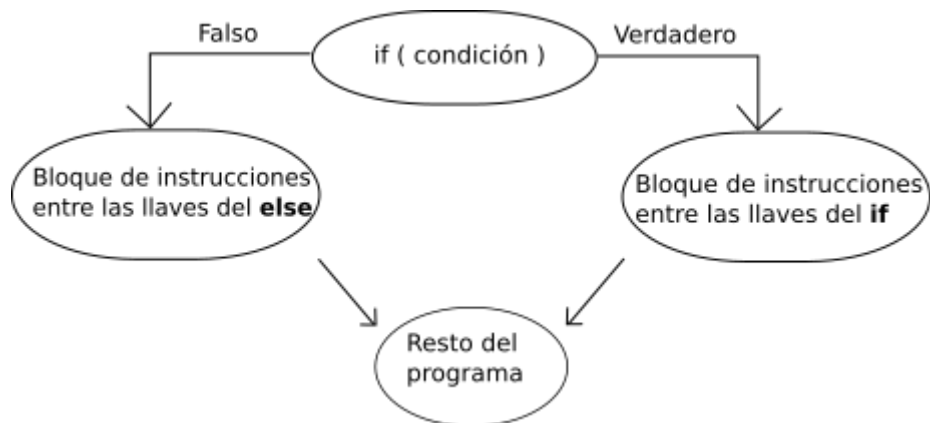
Sin embargo, muchas veces también necesitamos condicionar la ejecución de cierto código a que la condición sea falsa. Es decir, código diferente que se ejecutará en función de si la condición se evalúa a cierto o falso. Para ello tenemos el bloque **else**, que se pone a continuación del bloque if y que sólo se ejecutará cuando la condición sea falsa.

```
Console.Write("Dime tu nombre: ");
string nombre = Console.ReadLine();
if (nombre == "")
```

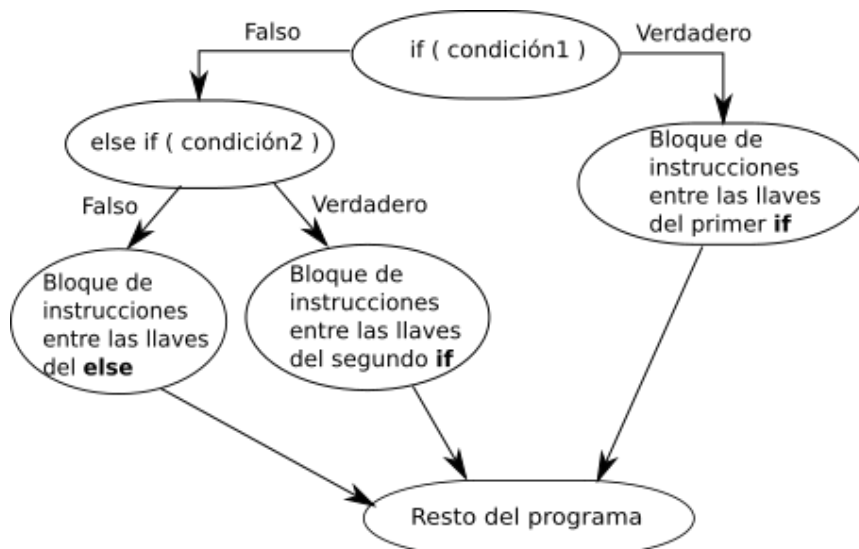
```

{
    Console.WriteLine(";El nombre no puede estar vacío!");
}
else
{
    Console.WriteLine($"Hola {nombre}");
}

```



La estructura condicional if también permite definir varios bloques de instrucciones en base a múltiples condiciones. Para ello utilizamos el bloque **else if**, que comprueba otra condición en caso de que la anterior no haya sido cierta. De esta manera ya no es una decisión binaria sino que podemos crear tantas comprobaciones como necesitemos.



```

Console.Write("Dime el prefijo de tu país: ");
// Así se convierte una cadena a número
int prefijo = Int32.Parse(Console.ReadLine());
if (prefijo == 34)
{
    Console.WriteLine("Tu país es España");
}
else if (prefijo == 49)
{
    Console.WriteLine("Tu país es Alemania");
}

```

```

}
else
{
    Console.WriteLine("No sé cual es tu país...");
}

```

Importante: Las llaves no son obligatorias en los bloques de una estructura if siempre y cuando solo vayamos a ejecutar una instrucción. Sin embargo, y para evitar errores (añadir una segunda instrucción y olvidarnos de las llaves), se recomienda ponerlas siempre.

```

int edad = 15;
if (edad >= 18)
    Console.WriteLine("Eres mayor de edad");
    Console.WriteLine("Ya puedes votar");
// Imprime "Ya puedes votar". El bloque if solo engloba la primera instrucción.

int edad = 15;
if (edad >= 18)
{
    Console.WriteLine("Eres mayor de edad");
    Console.WriteLine("Ya puedes votar");
}
// No imprime nada

```

Estructura switch

Una estructura similar a la anterior, es la estructura **switch**. Es útil sobre todo cuando necesitamos hacer varias comprobaciones, ya que su sintaxis es más limpia que utilizar varios **else if**. Sin embargo estamos limitados a comprobar el valor de una sola variable. Si nuestra condición implica comprobar el valor de 2 o más variables, o si queremos comparar con mayor, menor, distinto, debemos recurrir a la estructura if.

En este caso después de la palabra **switch**, en lugar de una condición, pondremos la variable a comprobar entre paréntesis. Dentro del bloque de la estructura pondremos tantas instrucciones **case** como valores queramos comprobar seguidos del valor y dos puntos. A continuación, después de la instrucción case, insertaremos las instrucciones que se ejecutarán cuando la variable valga ese valor.

Cuando queramos realizar alguna acción si el valor no entra dentro de los que estamos comprobando, usamos la instrucción **default**. Se pone siempre la última, y sería equivalente a la instrucción else en la estructura if.

```

Console.Write("Dime el prefijo de tu país: ");
// Así se convierte una cadena a número

int prefijo = Int32.Parse(Console.ReadLine());
switch (prefijo)
{
    case 34:
        Console.WriteLine("Tu país es España");
        break;
    case 49:
        Console.WriteLine("Tu país es Alemania");
        break;
}

```

```

default:
    Console.WriteLine("No sé cual es tu país...");
    break;
}

```

Importante: Se debe usar la instrucción **break** al final de cada bloque case, o continuará ejecutando los siguientes bloques case hasta que llegue a una instrucción break (debe haber una como mínimo en el último bloque). Eso no significa que a veces interese hacerlo así para agrupar una serie de instrucciones dentro de posibles valores.

```

string tipoUsuario = "Administrador";
switch (tipoUsuario)
{
    case "Administrador": // Ejecuta lo mismo que si es "jefe"

    case "Jefe":
        Console.WriteLine("Tienes permiso para acceder");
        break;
    case "Usuario":
        Console.WriteLine("No tienes permiso");
        break;
    default:
        Console.WriteLine("Usuario desconocido");
        break;
}

```

Operador ternario

Cuando queremos asignar un valor a una variable en función de una condición, se puede hacer de una forma más resumida utilizando el operador ternario. Es un concepto similar a una estructura if/else. En este caso, a la derecha de la asignación pondríamos la condición seguida de una interrogación, y 2 posibles valores separados por 2 puntos ':'. Cuando la condición sea cierta, se asignará el valor de la izquierda, mientras que si es falsa, se asigna el de la derecha.

```

int n1 = 4;
int n2 = 7;
int diferencia = n1 > n2 ? n1 - n2 : n2 - n1;
Console.WriteLine(diferencia); // 3

```

Se puede usar el operador ternario directamente para imprimir un valor u otro en una cadena sin necesidad de crear una variable para ello. El único requisito es poner la expresión entera entre paréntesis.

```

int num = 23;
Console.WriteLine($"{num} es {(num % 2 == 0 ? "par" : "impar")}");

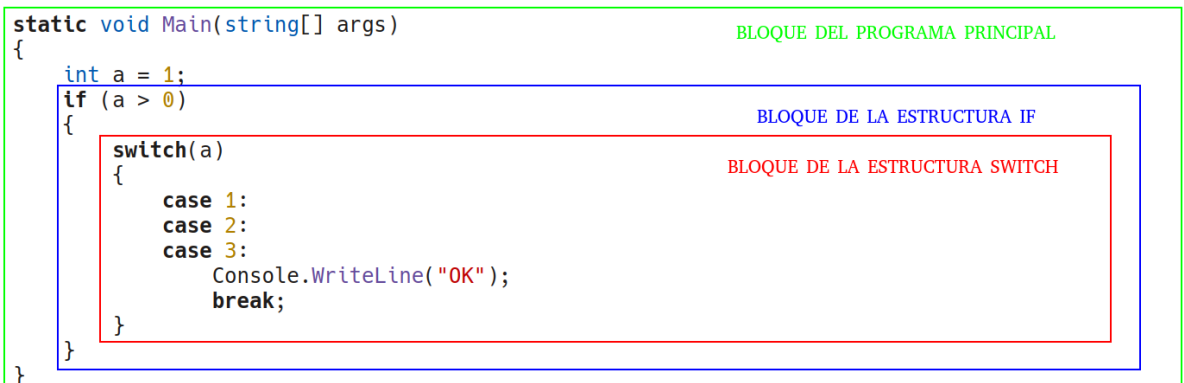
```

Bloques y ámbito de las variables

En los lenguajes de programación como C#, un bloque se define como un conjunto de instrucciones que se sitúan dentro de 2 llaves, una de apertura y otra de cierre. Además, se distinguen las instrucciones dentro de dicho bloque porque están más desplazadas a la derecha que las llaves. Estructuras de control como if, u otras que veremos más adelante como los bucles, métodos, etc., crean bloques nuevos con las instrucciones que contienen.

Los bloques pueden anidarse, es decir, dentro de un bloque de instrucciones podemos crear otro y así indefinidamente.

```
class Program
{
    static void Main(string[] args)
    {
        int a = 1;
        if (a > 0)
        {
            switch(a)
            {
                case 1:
                case 2:
                case 3:
                    Console.WriteLine("OK");
                    break;
            }
        }
    }
}
```



Cuando declaramos una variable, esta existe dentro del bloque donde se declara, y en todos los bloques que este contenga. Una vez acaba el bloque, esta deja de existir. Esto es lo que se conoce como una **variable local** a un bloque.

En el siguiente ejemplo, declaramos la variable sueldo tanto dentro del bloque if como dentro del bloque else. Cuando intentamos acceder a ella fuera de ese bloque nos da un error ya que no encuentra dicha variable.

```
int experiencia = 5;
if(experiencia > 3)
{
    int sueldo = 20000;
}
else
{
    int sueldo = 15000;
}
Console.WriteLine(sueldo); // ¡ERROR!, ¡La variable sueldo no existe!
```

¿Qué pasa si declaramos una variable con el mismo nombre que ya existe fuera del bloque actual?. Aunque hay lenguajes que sí nos lo permiten y se comportan como variables diferentes, en el caso de C# no podemos

```
int n = 1;
if (n > 0)
{
```

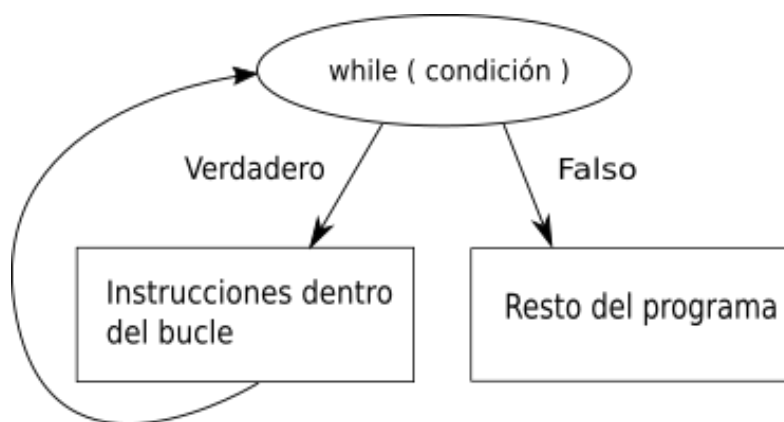
```
int n = 4; // Error. La variable n ha sido declarada ya  
}
```


Estructuras iterativas

Hasta ahora hemos visto como ejecutar código en función de si se cumple una determinada condición. Sin embargo, existen estructuras que nos permiten repetir la ejecución de una serie de instrucciones en bucle mientras se cumpla cierta condición.

Bucle while

La estructura **while** tiene una sintaxis muy similar a la estructura **if**. De hecho, la principal diferencia es que al terminar el bloque se vuelve a comprobar la condición, y **mientras** (de ahí el nombre de la estructura) siga siendo cierta, se vuelve a repetir el bloque de instrucciones.



```
int total = 0;
while (total < 100)
{
    Console.WriteLine($"Escribe un número (acumulado: {total}): ");
    int num = Convert.ToInt32(Console.ReadLine());
    total += num;
}
Console.WriteLine($"El total acumulado ha sido {total}");
```

```
Escribe un número (acumulado: 0): 34
Escribe un número (acumulado: 34): 23
Escribe un número (acumulado: 57): 13
Escribe un número (acumulado: 70): 24
Escribe un número (acumulado: 94): 45
El total acumulado ha sido 139
```

La estructura **while** no tiene una instrucción equivalente a **else**, por lo que si queremos comprobar otras posibilidades por las que no se ha cumplido la condición del bucle, debemos hacerlo al terminar de definirlo.

Existe otra forma de escribir la estructura **while**, y es usando una estructura llamada **do...while**. El concepto es el mismo, solo que la condición se comprueba al finalizar de ejecutar las instrucciones del bloque y no antes, por lo que siempre nos garantiza que se ejecutará al menos una vez.

```
// Generamos número aleatorio entre 1 y 10
int adivina = new Random().Next() % 10 + 1;
int intentos = 0;
int num;
do
{
    Console.Write($"Adivina el número del 1 al 10: ");
    num = Convert.ToInt32(Console.ReadLine());
    intentos++;
} while (adivina != num && intentos < 3);

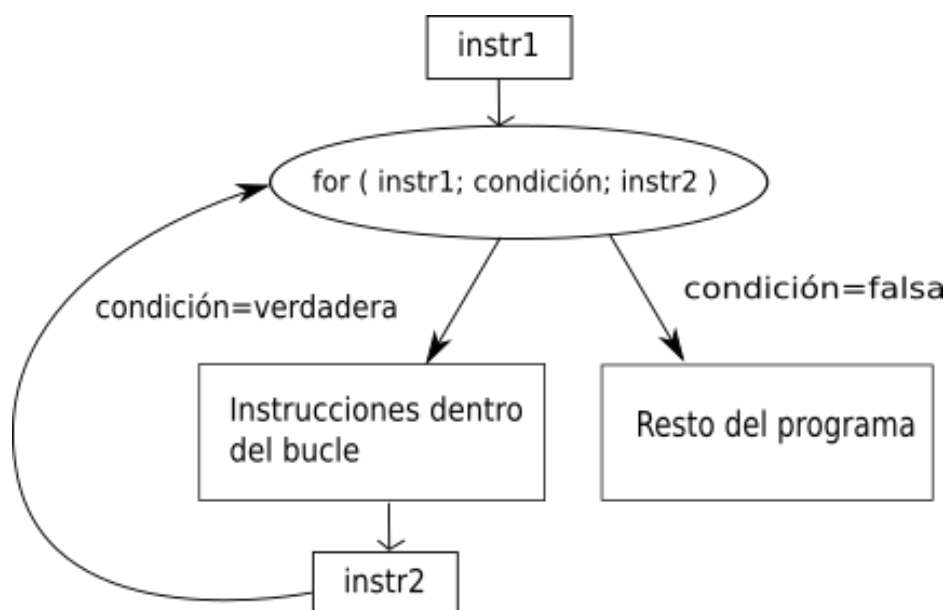
if (intentos <= 3)
{
    Console.WriteLine("Has acertado!");
}
```

```
Adivina el número del 1 al 10: 3
Adivina el número del 1 al 10: 9
Adivina el número del 1 al 10: 6
Has acertado!
```

Bucle for

La estructura **for** es una estructura repetitiva basada en una condición al igual que **while**. La principal diferencia es que dentro de los paréntesis, además de la condición, podremos inicializar una o más variables y modificarlas en cada iteración. Primero inicializamos una variable, después la condición de repetición (como en el bucle **while**) y finalmente una instrucción de actualización que se ejecutará automáticamente al final de cada repetición del bucle.

```
for (inicialización; condición; actualización)
{
    // Instrucciones a repetir
}
```



Normalmente es una estructura que se utiliza para gestionar contadores: variables con un valor inicial y otro final que se comprobará en la condición.

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine($"Repetición número {i+1}");
}
```

Repetición número 1

Repetición número 2
 Repetición número 3
 Repetición número 4
 Repetición número 5
 Repetición número 6
 Repetición número 7
 Repetición número 8
 Repetición número 9
 Repetición número 10

En este caso estamos creando un contador que empieza en cero y acaba en 9, ya que en cuanto su valor sea 10 la condición dejaría de cumplirse y el bucle terminaría. Aunque empezar en cero un contador es lo más común cuando programamos (veremos el porqué cuando lleguemos a la parte de arrays), se puede empezar en el número que se quiera.

```
// Cuenta del 1 al 10
for (int i = 1; i <= 10; i++);
...

// Cuenta del 10 al 1 (decrementa)
for (int i = 10; i > 0; i--);
...
```

Si queremos hacer lo mismo con un bucle while, simplemente tenemos que situar la instrucción de inicialización antes del bucle (para que se ejecute solo una vez antes de empezar) y la instrucción de actualización como la última del bucle.

```
int i = 0;
while (i < 10)
{
    Console.WriteLine($"Repetición número {i+1}");
    i++;
}
```

Se pueden inicializar y modificar tantas variables como queramos, siempre y cuando las separemos por comas.

```
for (int i = 1, j = 10; i <= 10 && j > 0; i++, j--)
{
    Console.WriteLine($"i = {i}, j = {j}");
}
```

i = 1, j = 10
 i = 2, j = 9
 i = 3, j = 8
 i = 4, j = 7
 i = 5, j = 6
 i = 6, j = 5
 i = 7, j = 4

```
i = 8, j = 3
i = 9, j = 2
i = 10, j = 1
```

Tampoco es obligatorio incluir las instrucciones de inicialización ni las de actualización. En ese caso, tendríamos un bucle **for** que se comportaría exactamente igual a un bucle **while**.

```
int total = 0;
for(;total < 100;)
{
    Console.Write($"Escribe un número (acumulado: {total}): ");
    int num = Convert.ToInt32(Console.ReadLine());
    total += num;
}
Console.WriteLine($"El total acumulado ha sido {total}");
```

Bucle foreach

Para recorrer colecciones de datos (arrays, listas, etc.) también podemos usar el bucle **foreach**. Como este tipo de colecciones las vamos a ver más adelante, por ahora solo vamos a ver un ejemplo sencillo, que entenderemos del todo al llegar a los arrays.

```
int[] array = {9, 12, 26, 34};
foreach (int num in array)
    // Recorremos los números de arriba
{
    // En cada repetición, num toma el valor del siguiente número
    Console.WriteLine(num);
}

9
12
26
34
```

Break y continue

Además de la comprobación de la condición, existe otra forma de finalizar el bucle, y es usando la instrucción **break**. Si se ejecuta esta instrucción dentro de un bucle, este finaliza inmediatamente sin ejecutar las instrucciones restantes y sin volver a comprobar la condición.

```
int total = 0;
for(int i = 0; i < 5; i++) {
    Console.Write($"Dime un número del 1 al 10 (total {total}): ");
    int num = Convert.ToInt32(Console.ReadLine());
    if(num < 1 || num > 10)
    {
        Console.WriteLine("Número no válido");
        break;
    }
    total += num;
}
Console.WriteLine($"Total acumulado: {total}");
```

```
Dime un número del 1 al 10 (total 0): 3
Dime un número del 1 al 10 (total 3): 4
Dime un número del 1 al 10 (total 7): 7
Dime un número del 1 al 10 (total 14): 10
Dime un número del 1 al 10 (total 24): 6
Total acumulado: 30
```

```
Dime un número del 1 al 10 (total 0): 2
Dime un número del 1 al 10 (total 2): 7
Dime un número del 1 al 10 (total 9): 24
Número no válido
Total acumulado: 9
```

La instrucción **continue** rompe la iteración actual del bucle, ignorando el resto de instrucciones, pero en este caso, vuelve a comprobar la condición para seguir repitiendo la ejecución del bucle. Si estamos en un bucle for, la instrucciones de finalización, como el incremento de la variable se siguen realizando.

```
for (int i = 1; i <= 10; i++)
{
    if ( i % 2 > 0)
    {
        continue;
    }
    Console.WriteLine($"Repetición número {i}");
}
```

```
Repetición número 2
Repetición número 4
Repetición número 6
Repetición número 8
Repetición número 10
```

Hay que evitar abusar de las instrucciones break y continue por lo que se recomienda no usarlas si no está justificado. Por ejemplo, que la alternativa a usarlas introduzca bastante complejidad en el código. Esto es así porque resulta más complicado depurar un programa usando estas instrucciones para gestionar un bucle.

Bucles infinitos

Cuando, ya sea por error o deliberadamente, la condición de un bucle nunca va a ser falsa, se dice que estamos ante un bucle infinito. Si no tenemos una instrucción como break que pueda detener la ejecución del bucle, este se repetirá indefinidamente y la aplicación nunca acabará de ejecutarse, por lo que tendremos que cerrar el programa a la fuerza (En la consola se cierra un proceso con Ctrl+C).

```
// Condición errónea. Nunca termina
for (int i = 0; i >= 0; i++)

// No actualizamos la variable. La condición nunca será falsa
for (int i = 0; i < 10;)

// Bucle infinito a propósito (siempre true)
while(true);

// Bucle infinito a propósito (for sin condición)
```

```

for(;;);

while(true)
{
    Console.Write("Escribe un texto ('q' para salir): ");
    string texto = Console.ReadLine();
    if (texto == "q")
    {
        break; // Única forma de salir de aquí
    }
    Console.WriteLine($"Has escrito: {texto}");
}

```

Bucles anidados

Dentro de un bucle (de cualquier bloque en general) podemos tener la necesidad de anidar otro bucle (entre otras estructuras). La lógica para saber como tenemos que crear una estructura con bucles anidados puede resultar algo compleja al principio, y como todo, mejora cuanto más practiquemos.

A continuación vamos a ver algunos ejemplos.

```

for (int tabla = 1; tabla <= 5; tabla++)
{
    for (int num = 1; num <= 10; num++)
    {
        Console.WriteLine($"{tabla} x {num} = {tabla * num}");
    }
}

```

```

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
...
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

```

```

int alto = 5;
int ancho = 8;
for (int fila = 0; fila < alto; fila++)
{
    for (int col = 0; col < ancho; col++)
    {
        Console.Write("*");
    }
    Console.WriteLine(); // Generamos salto de línea
}

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

```

