EXPRESIONES RELACIONALES

Las **expresiones relacionales** se usan para expresar la **relación** que existe entre dos valores. Los valores pueden estar contenidos adentro de **variables** o ser colocados **explícitamente**. Estas expresiones, al igual que las expresiones aritméticas, tienen sus propios **operadores**. La expresión será evaluada, pero el resultado de la evaluación tendrá únicamente dos valores posibles: **true** o **false**.
Ambos valores son de tipo **bool** y **true** es usado para indicar que la expresión evaluada es **verdadera**. El valor de **false** por su parte se utiliza para indicar que la expresión evaluada es **falsa**.

Operadores relacionales
En la **tabla 1** podemos apreciar los operadores relacionales en C#. La forma como están escritos sus signos es la forma como debemos colocarlos en nuestro programa.

SIGNO	OPERADOR
==	Igualdad
!=	No igual
>	Mayor que
<	Menor que
>=	Mayor que igual
<=	Menor que igual

Tabla 1. Esta tabla nos muestra los diferentes operadores relacionales de C#.

El operador de **igualdad** se representa con dos signos igual juntos y sin espacio. Colocar un solo signo igual, indica que el operador será de **asignación**. Esto puede derivar en errores de lógica, porque una asignación siempre evalúa a verdadero, pero una igualdad no.

EL USO DE "IF"

Este tipo de expresiones se usa en diferentes casos, pero generalmente en las estructuras selectivas o repetitivas. Las estructuras selectivas son aquellas que nos permiten hacer una selección entre dos o varias rutas de ejecución posibles. La selección se llevará a cabo según el valor de una expresión. Esta expresión puede ser una expresión relacional. Nuestra primera estructura selectiva se conoce como if, que es un **si condicional**. Si tal cosa sucede, entonces haz tal cosa. El uso del if es sencillo:

```
if(expresión)
    Sentencia a ejecutar
```

El uso de if requiere que coloquemos una expresión a evaluar entre paréntesis. Si el resultado de esta expresión es true, entonces la sentencia a ejecutar se lleva a cabo. Si el resultado de la evaluación es **false**, entonces la sentencia a ejecutar nunca se lleva a cabo, o dicho de otra forma, es ignorada.

BLOQUE DE CÓDIGO CON IF

Como vimos, **if** efectivamente puede sernos de mucha utilidad para que el programa lleve a cabo la decisión correcta, sin embargo, solamente ha podido ejecutar una sentencia. Con frecuencia nos encontraremos con el caso cuando necesitamos que se ejecute más de una sentencia si la condición se cumple. Una buena opción para resolver este inconveniente encontrado es colocarle un **bloque de código** a **if**. En este caso, el bloque de código se ejecutará siempre que la expresión dentro de **if** sea evaluada como **true**, y el bloque de código completo será ignorado siempre que la expresión sea evaluada como **false**.

El bloque de código es un conjunto de sentencias agrupadas y debe ser abierto con **{** y cerrado con **}**, quedará compuesto de la siguiente forma:

```
if(expresión)
{
    Sentencia 1; Sentencia 2;
    ...
    Sentencia n;
}
```

EL USO DE ELSE

Puede ocurrir que a veces necesitamos que una sentencia se ejecute cuando se cumple la condición y que otra sentencia se ejecute cuando esa condición no se cumpla. Si nos encontramos en este caso, una forma de resolverlo sería colocar un **if** con una condición y luego otro **if** con una condición que sea complemento de la primera. Esto ya lo hemos hecho anteriormente en el programa de los números positivos y negativos. Sin embargo, existe otra forma más sencilla que la utilizada en este programa para hacerlo. Esta forma puede sernos de mucha utilidad para simplificar la lógica y que no nos veamos obligados de tener que colocar tantos **if** dentro de una sentencia. Para lograr esto haremos uso de **else**. Siempre utilizaremos **else** a continuación de una sentencia **if**, ya que **else** no se puede usar sólo. Tenemos que colocar nuestro código de la siguiente forma:

```
if(condición)
    Sentencia1;
else
    Sentencia2;
```

Si la condición es evaluada como verdadera, entonces se ejecuta la sentencia 1, en cambio, cuando la condición se evalúa como falsa, se ejecuta la sentencia 2.

ELSE CON BLOQUE DE CÓDIGO

Al igual que con **if**, nosotros podemos colocar un bloque de código en **else**. Esto nos permitiría que más de una sentencia se ejecute cuando no se cumple la condición del **if**. No debemos olvidar colocar el bloque de código empezando con una llave, **{**, y finalizándolo con la llave de cierre **}**. Adentro del bloque de código podemos colocar cualquier sentencia válida de C#.

CÓMO USAR IF ANIDADOS

La sentencia o el bloque de código que ejecuta **if** puede ser cualquier sentencia válida o bloque de código válido en C#, esto incluye a otro **if**. Esto significa que nosotros podemos colocar **if** adentro del código a ejecutar de un **if** anterior. Cuando hacemos esto hay que ser cuidadosos para evitar errores de lógica. Esto se conoce como **if anidados**.

ESCALERA DE IF-ELSE

Otra estructura que se puede utilizar es la **escalera de if-else**. Una de sus funciones principales es optimizar la ejecución del código. Algunas veces son necesarias porque así lo requiere la lógica del programa. Al igual que con los **if** anidados, no hay que utilizarlos sin planeación, ya que, al no programarlos correctamente, nos puede llevar a errores de lógica que pueden resultar difíciles de solucionar.

```
if(expresión 1)
    Sentencia 1
else if(expresión 2)
        Sentencia 2
    else if(expresión 3)
        Sentencia 3
```

EXPRESIONES LÓGICAS

Al igual que las expresiones aritméticas y relacionales, las expresiones lógicas tienen sus propios operadores. Éstos son: y, o y no. En inglés los conocemos como: and, or y not.

OPERADOR	SIGNIFICADO
&&	y
||	o
!	no

Tabla 2. *Aquí tenemos los operadores lógicos utilizados en C#.*

EL USO DE LA CONJUNCIÓN

Empecemos a conocerlos. El primer operador es y, conocido como conjunción. Para usar este operador es necesario tener dos expresiones. Una expresión a la izquierda y la otra a la derecha. Las expresiones se evalúan devolviendo valores true o false. Con la conjunción, la expresión se evalúa como verdadera únicamente cuando ambas expresiones son verdaderas. La siguiente es la tabla de verdad para este operador:

P	Q	P&&Q
true	**true**	**true**
false	**true**	**false**
true	**false**	**false**
false	**false**	**true**

Tabla 3. *La tabla de verdad de la conjunción.*

Supongamos que tenemos que decidir si debemos llenar el tanque de gasolina del vehículo. Esto lo hacemos si el tanque tiene menos del 50% y si la distancia a re- correr es de más de 200 km. Como vemos, tenemos que usar la conjunción ya que tenemos dos condiciones y ambas se deben cumplir para que suceda la carga de la gasolina. Si colocamos esto como expresión quedaría:

```
if(tanque < 50 && distancia > 200)
    Cargar gasolina
```

Supongamos que tanque es 25 y distancia es 350, al evaluar 25 < 50 y 350 > 200 te- nemos lo siguiente.

```
if(true && true)
    Cargar gasolina
```

En nuestra tabla sabemos que cuando ambas expresiones son verdaderas, entonces se cumple la conjunción y la expresión lógica nos evalúa a true.

```
if(true)
    Cargar gasolina
```

Por lo que cargaremos gasolina.

Ahora supongamos que **tanque** tiene el valor de **90** y **distancia** es nuevamente **350**. Las primeras evaluaciones nos darían:

```
if(false &&  tue)
    Cargar gasolina
```

EL USO DE LA DISYUNCIÓN

Para la disyunción hacemos uso del operador o. Éste también necesita dos expresiones, una en el lado derecho y otra en el lado izquierdo. Esta disyunción tiene la siguiente tabla de verdad:

P	Q	P||Q
true	true	true
false	true	true
true	false	true
false	false	false

Tabla 4. *La tabla de verdad de la disyunción.*

Veamos un ejemplo con la disyunción. Tenemos que tomar la sombrilla si llueve o si hay mucho sol. La expresión podría quedar de la siguiente manera:

```
if(lluvia == true  || muchosol  == true)
    Tomar sombrilla
```

Supongamos que hay lluvia, pero no hay sol, tendríamos la expresión:

```
if(true || false)
    Tomar sombrilla
```

En este caso, si vemos la tabla de verdad, podemos observar que el resultado de eva- luar la expresión nos da true, por lo que sí debemos tomar la sombrilla.

Ahora veamos qué sucede si no hay lluvia y no hay sol. La expresión quedaría como:

```
if(false || false)
    Tomar sombrilla
```

En este caso, la expresión se evalúa como false y no se toma la sombrilla.

EL USO DE LA NEGACIÓN

Nos falta conocer un operador más, el de la **negación**. Ese operador es muy sencillo y solamente necesita hacer uso de un operando del lado derecho. Con estas condiciones dadas, esta expresión será negada por el operador.

p	Negación true
false	
false	true

Tabla 5. *La tabla de verdad de la negación.*

Esto quiere decir que si el operando del lado derecho es **true**, el operador regresa **false**. Y si el operando del lado derecho es **false**, el operador regresa **true**.

Esto quiere decir que si el operando del lado derecho es **true**, el operador regresa **false**. Y si el operando del lado derecho es **false**, el operador regresa **true**.

Veamos un ejemplo para poder comprender mejor esto.
Tenemos una habitación con una bombilla eléctrica y supongamos que tenemos que prender la luz del cuarto cuando **no es** de día. Entonces nuestra expresión queda de la siguiente forma:

```
if(!dia == true) Prender la luz
```

Veamos qué sucede cuando la expresión **dia** tiene como valor verdadero. La expresión **dia == true** se evalúa como **true** y luego éste se niega, lo que al final nos da como resultado un valor **false**. Como **if** recibe el valor **false**, se ejecutará la orden o el método **prender la luz**.
Pero si **dia** es **false**, entonces la expresión **dia == true** se evalúa como **false**, que es negado e **if** recibe **true**. Como **if** recibe **true**, se ejecuta **prender la luz**.

EL USO DE SWITCH

Cuando usamos **switch** es necesario colocar entre paréntesis la variable que utilizaremos para llevar a cabo las comparaciones. Luego tenemos que crear un bloque de código y colocar adentro de él los casos y el código a ejecutar para cada caso. Para indi- car un caso, usamos **case** seguido del valor de comparación y dos puntos.
Existe un caso llamado **default,** que podemos utilizar si lo deseamos. Este caso siempre debe ser el último caso definido. Cuando la variable de comparación no ha en- contrado su valor en ninguno de los casos, entonces se ejecuta el código del caso **default.** En los casos podemos colocar cualquier código C# que sea válido.
Para indicar dónde termina el código de un caso debemos hacer uso de **break**, que nos permitirá salir de la ejecución de una estructura selectiva o repetitiva. Esto lo veremos con más detalle en otro capítulo. Si no hacemos uso de **break** y el caso es- tá vacío, entonces el programa continuará ejecutando el código del próximo caso y así sucesivamente hasta el final del bloque del código que pertenece al **switch**.