Estructuras de control

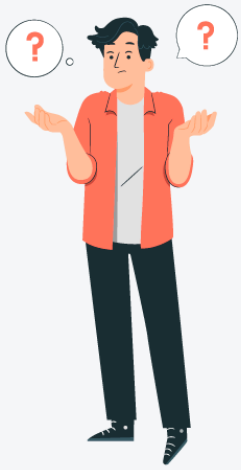
Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1º D
Libro: Estructuras de control

Imprimido por: Eduardo Moreno
Día: miércoles, 19 de marzo de 2025, 23:06

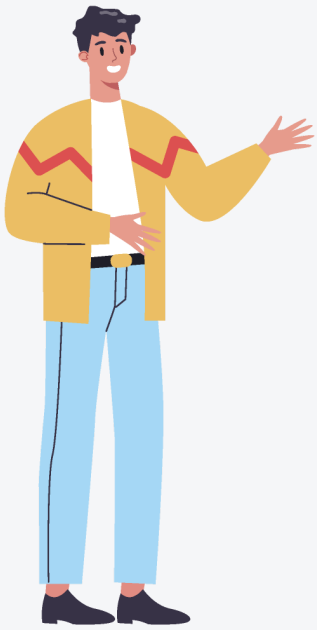
Tabla de contenidos

Preguntas orientadoras

- ¿Qué son las estructuras de control y cuál es su propósito en un programa?
- ¿Cuáles son las tres estructuras de control básicas en la programación y cómo se utilizan?
- ¿Cómo se implementa una estructura de control condicional (if-else) en C# y qué situaciones son adecuadas para su uso?
- ¿Qué es un bucle (loop) y cuál es su utilidad en la programación? ¿Cuáles son los diferentes tipos de bucles en C#?
- ¿Cuándo es apropiado utilizar un bucle for en lugar de un bucle while o viceversa?



Operadores

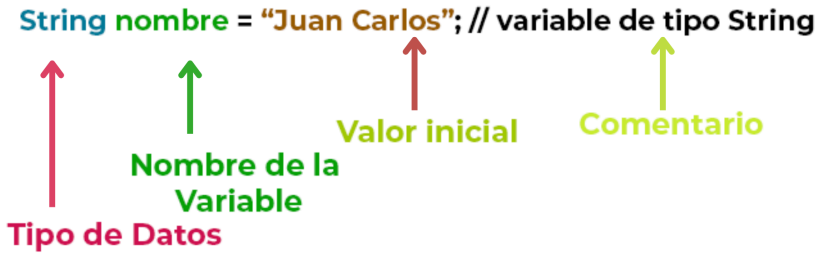


En programación, los operadores son símbolos especiales que se utilizan para realizar diferentes operaciones en los datos. Por ejemplo, los operadores matemáticos como +, -, *, / y % se utilizan para realizar operaciones aritméticas en los números. Los operadores lógicos como &&, || y ! se utilizan para realizar operaciones booleanas y lógicas en las expresiones.

En C#, hay muchos tipos diferentes de operadores que se pueden utilizar para realizar una amplia variedad de operaciones en los datos. Algunos de los tipos más comunes de operadores en C# incluyen:

- **Operadores aritméticos:** se utilizan para realizar operaciones matemáticas en los números, como la suma (+), la resta (-), la multiplicación (*), la división (/) y el módulo (%).
- **Operadores de asignación:** se utilizan para asignar valores a las variables, como el operador de asignación simple (=) y los operadores compuestos como +=, -= y *=.
- **Operadores de comparación:** se utilizan para comparar dos valores y producir un resultado booleano (true o false), como los operadores de igualdad (==), desigualdad (!=), mayor que (>), menor que (<), mayor o igual que (>=) y menor o igual que (<=).
- **Operadores lógicos:** se utilizan para combinar expresiones booleanas y producir un resultado booleano (true o false), como los operadores && (y), || (o) y ! (no).

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.



Tablas de operadores

Operadores numéricos

Tipo de operador	Operadores asociados
Cambio de signo	-,+
Aritméticos	+,-,*, /, %
Incremento y decremento	++, - -

Operadores de comparación

Operador	Significado
>	Mayor que
<	Menor que
==	Igual a
>=	Mayor o igual que
<=	Menor o igual que
!=	Distinto que

Si condicional (if)

Una de las estructuras de control de flujo más utilizadas es la estructura condicional if que permite realizar una acción determinada en función de una condición booleana. En términos simples, el if se utiliza para tomar decisiones en el código y ejecutar diferentes bloques de código en función del resultado de una evaluación lógica, permitiendo ejecutar diferentes bloques de código según una determinada condición se cumpla o no.

En C#, el condicional "if" (si en español) se utiliza para tomar decisiones en función de una expresión booleana.

La sintaxis básica del condicional "if" es la siguiente:

```
if (expresión booleana)
{
    // Código a ejecutar si la expresión es verdadera
}
```

Ejemplo 1

Próximo a las elecciones, se desea pedir al usuario/a que ingrese su edad. En caso de ser menor y no poder votar, mostrar un mensaje por pantalla indicando su condición.

```
static void Main(string[] args)
{
    {
        const int EDAD_MINIMA = 16;
        int edad;
        Console.WriteLine("Ingresa tu edad:");
        edad = int.Parse(Console.ReadLine());
        if (edad < EDAD_MINIMA)
        {
            Console.WriteLine("Sos menor de edad y no podes votar");
        }
    }
}
```

La expresión booleana debe evaluarse como verdadera o falsa, y dependiendo del resultado, se ejecutará el bloque de código dentro de las llaves que sigue a la expresión.

Si la expresión es verdadera, el bloque de código se ejecutará, y si es falsa, se omitirá.

Si condicional (if - else)

También es posible agregar un bloque de código opcional que se ejecutará si la expresión booleana es falsa, utilizando la cláusula "else":

```
if (expresión booleana)
{
    // Código a ejecutar si la expresión es verdadera
}
else
{
    // Código a ejecutar si la expresión es falsa
}
```

Ejemplo 2

Próximo a las elecciones, se desea pedir al usuario/a que ingrese su edad. En caso de ser menor de 16 años, mostrar un mensaje por pantalla indicando su condición, y en caso de tener al menos 16 años indicar que puede votar.

```
static void Main(string[] args)
{
    const int EDAD_MINIMA = 16;
    int edad;
    Console.WriteLine("Ingresa tu edad:");
    edad = int.Parse(Console.ReadLine());
    if (edad < EDAD_MINIMA)
    {
        Console.WriteLine("Sos menor de edad y no podes votar");
    }
    else
    {
        Console.WriteLine("Estás en condición de votar");
    }
}
```

Si condicional (if - else if)

En algunos casos, puede ser necesario evaluar múltiples expresiones booleanas y tomar diferentes acciones en función de los resultados de cada una. En estos casos, se puede utilizar la cláusula "else if" para evaluar múltiples expresiones booleanas:

```
if (expresión booleana 1)
{
    // Código a ejecutar si la expresión 1 es verdadera
}
else if (expresión booleana 2)
{
    // Código a ejecutar si la expresión 2 es verdadera
}
else
{
    // Código a ejecutar si ninguna de las expresiones es verdadera
}
```

Ejemplo 3

Próximo a las elecciones, se desea pedir al usuario/a que ingrese su edad. En caso de ser menor de 16 años, mostrar un mensaje por pantalla indicando su condición; de ser mayor de 18 y menor a 70 indicar que debe votar.

En caso que la persona sea mayor a 70 años o que tenga entre 16 y 18, mostrar un mensaje indicando que no tiene obligación de realizar el voto.

```
static void Main(string[] args)
{
    const int EDAD_MINIMA = 18;
    const int EDAD_MAXIMA = 70;
    int edad;
    Console.WriteLine("Ingresa tu edad:");
    edad = int.Parse(Console.ReadLine());
    if (edad < EDAD_MINIMA)
    {
        Console.WriteLine("Sos menor de edad y no podes votar");
    }
    else if (edad >= EDAD_MINIMA && edad <= EDAD_MAXIMA)
    {
        Console.WriteLine("Tenes la obligación de votar");
    }
    else
    {
        Console.WriteLine("El voto es optativo");
    }
}
```

Es importante tener en cuenta que las expresiones booleanas pueden involucrar operadores lógicos como "&&" (and) y "||" (or) para combinar múltiples condiciones.

También se pueden utilizar operadores de comparación como "==" (igual a), "!=" (diferente de), "<" (menor que), ">" (mayor que), "<=" (menor o igual que) y ">=" (mayor o igual que) para comparar valores.

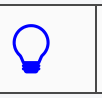
Switch

Es una estructura de control que se utiliza para tomar decisiones basadas en el valor de una variable. Es un condicional que divide la ejecución del programa según vayan cumpliendo ciertos valores.

Lo que busca es evitar muchos if anidados que dependan solo de una variable y solo usen la condición de igualdad “==” (es igual a).

> “La switch instrucción selecciona una lista de instrucciones para ejecutarla en función de la coincidencia de un patrón con una expresión.”
>
> Fuente:
>
> Instrucciones de selección `if`, `if-else` y `switch.` Disponible haciendo clic **aquí.**

El switch es un modo compacto de los if else anidados que sólo permite condiciones de igualdad.



Ejemplo

```
switch (variable) {
    case ‘a’: cuerpo1
    break;
    case ‘b’: cuerpo2
    break;
    default: cuerpo3
}
```

Si la variable es igual a 'a', se ejecuta el cuerpo1.

Si la variable es igual a 'b', se ejecuta el cuerpo2.

Si no cumple en ningún caso, se ejecuta 'default'.

Tener en cuenta que se ejecutan las sentencias desde el case que cumple la condición hasta el final del switch o bien hasta el primer break que encuentra.

Sentencia break

La sentencia break se usa para definir que ahí termina la ejecución de toda esa estructura. En este caso se suele usar siempre el break para indicar que solo queremos que haga ese caso particular, luego salga del switch y continúe con las siguientes sentencias que puedan estar debajo y fuera de la estructura del switch.

Pero en caso que no este el break va a seguir ejecutando los demás casos a partir de la coincidencia encontrada.

Ejemplo con uso del break

```
char letra;
Console.WriteLine("Ingrese una letra");
letra = char.Parse(Console.ReadLine().ToLower());
switch (letra)
{
    case'a':
        Console.WriteLine("Es vocal");
        break;
    case 'e':
        Console.WriteLine("Es vocal");
        break;
    case 'i':
        Console.WriteLine("Es vocal");
        break;
    case 'o':
        Console.WriteLine("Es vocal");
        break;
    case 'u':
        Console.WriteLine("Es vocal");
        break;

    default:
        Console.WriteLine("NO es vocal");
        break;

}
```

Ejemplo sin uso del break

Nos sirve para armar conjuntos ya que el switch no admite operadores lógicos ni condicionales.

```
char letra;
Console.WriteLine("Ingresa una letra: ");
letra = char.Parse(Console.ReadLine().ToLower());
switch (letra)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        Console.WriteLine("Ingresaste una vocal y es la: " + letra);
        break;
    default:
        Console.WriteLine("La letra " + letra + " no es una vocal.");
        break;
}
```

Ejemplo combinando conjuntos

```
switch (numero) {
  case 1:
  case 2:
  case 3:
    // Código a ejecutar si 'numero' es 1, 2 o 3
    break;
  case 4:
  case 5:
  case 6:
    // Código a ejecutar si 'numero' es 4, 5 o 6
    break;
  default:
    // Código a ejecutar si 'numero' no coincide con ninguno de los casos a
    break;
}
```

Ejercicio

Consigna día de semana

Se solicita al usuario ingresar un número del 1 al 7 para representar un día de la semana. El programa mostrará en la consola el día correspondiente al número ingresado.

(1 = domingo; 2 = lunes; 3 = martes, etc)

En caso que el número no sea válido, se mostrará el mensaje 'Número inválido'.

Resolución con if anidados

```
const int LUNES = 1, MARTES = 2, MIERCOLES = 3,
    JUEVES = 4, VIERNES = 5, SABADO = 6, DOMINGO = 7;
int numDia;
string diaSemana;
Console.WriteLine("Ingrese el numero del dia");
numDia = int.Parse(Console.ReadLine());

if (numDia == LUNES)
{
    diaSemana = "LUNES";
}
else if (numDia == MARTES)
{
    diaSemana = "MARTES";
}
else if (numDia == MIERCOLES)
{
    diaSemana = "MIERCOLES";
}
else if (numDia == JUEVES)
{
    diaSemana = "JUEVES";
}
else if (numDia == VIERNES)
{
    diaSemana = "VIERNES";
}
else if (numDia == SABADO)
{
    diaSemana = "SABADO";
}
else if (numDia == DOMINGO)
{
    diaSemana = "DOMINGO";
}
else
{
    diaSemana = "NO ES UN DIA VALIDO";
}
Console.WriteLine(diaSemana);
```

Resolución con Switch

```
43              const int LUNES = 1, MARTES = 2, MIERCOLES = 3,
44                  JUEVES = 4, VIERNES = 5, SABADO = 6, DOMINGO = 7;
45              int numDia;
46              string diaSemana;
47              Console.WriteLine("Ingrese el numero del dia");
48              numDia = int.Parse(Console.ReadLine());
49              switch (numDia)
50              {
51                  case LUNES:
52                      diaSemana = "LUNES";
53                      break;
54                  case MARTES:
55                      diaSemana = "MARTES";
56                      break;
57                  case MIERCOLES:
58                      diaSemana = "MIERCOLES";
59                      break;
60                  case JUEVES:
61                      diaSemana = "JUEVES";
62                      break;
63                  case VIERNES:
64                      diaSemana = "VIERNES";
65                      break;
66                  case SABADO:
67                      diaSemana = "SABADO";
68                      break;
69                  case DOMINGO:
70                      diaSemana = "DOMINGO";
71                      break;
72                  default:
73                      diaSemana = "NO ES UN DIA VALIDO";
74                      break;
75
76              }
77              Console.WriteLine(diaSemana);
```

Ciclos

Otras estructuras de control muy utilizadas son los ciclos que permiten repetir bloques de código varias veces hasta que se cumpla una condición determinada. Los ciclos son fundamentales en la programación porque permiten automatizar tareas repetitivas y reducir la cantidad de código que se necesita escribir.

En C# existen tres tipos principales de ciclos: while, do-while y for. Cada uno de ellos tiene sus propias características y se utiliza en diferentes situaciones.

- while

El ciclo while se utiliza cuando no se sabe cuántas veces se debe repetir el bloque de código

- do-while

El ciclo do-while se utiliza cuando se quiere asegurar que el bloque de código se ejecute al menos una vez

- for

El ciclo for se utiliza cuando se sabe exactamente cuántas veces se debe repetir el bloque de código.

Ciclo while

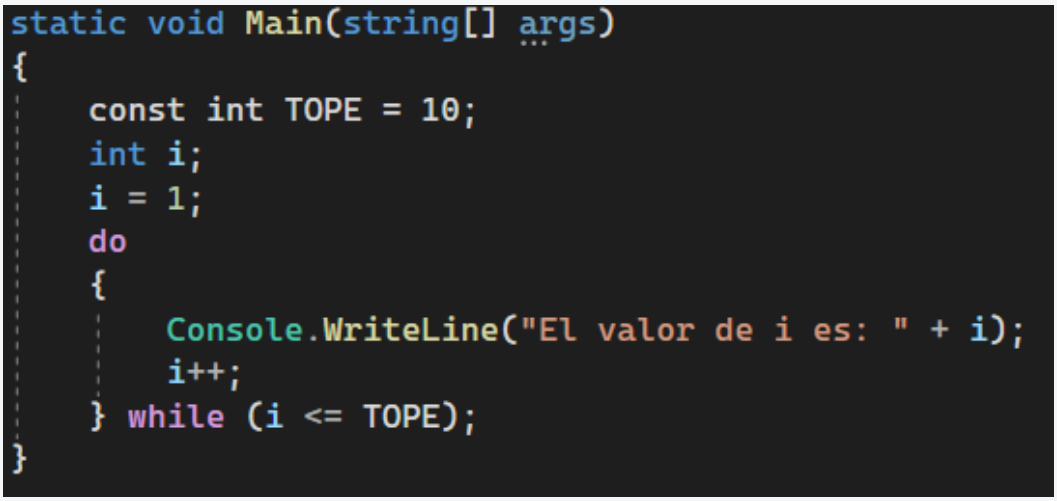
Ciclo "while": se utiliza para repetir un bloque de código mientras una expresión booleana sea verdadera.

```
static void Main(string[] args)
{
    const int TOPE = 10;
    int i;
    i = 1;
    while (i <= TOPE)
    {
        Console.WriteLine("El valor de i es: " + i);
        i++;
    }
}
```

En este ejemplo, se utiliza el ciclo "while" para repetir el bloque de código dentro de las llaves mientras la variable "i" sea menor o igual que 10. La variable "i" se incrementa en cada iteración hasta que alcanza el valor 11 saliendo del ciclo, ya que la condición de ser menor o igual al TOPE es falsa.

Ciclo do while

Ciclo "do-while": similar al ciclo "while", pero se ejecuta el bloque de código al menos una vez antes de verificar la expresión booleana.



```
static void Main(string[] args)
{
    const int TOPE = 10;
    int i;
    i = 1;
    do
    {
        Console.WriteLine("El valor de i es: " + i);
        i++;
    } while (i <= TOPE);
}
```

En este ejemplo, se utiliza el ciclo "do-while" para repetir el bloque de código dentro de las llaves mientras la variable "i" sea menor o igual que 10. La diferencia con el ciclo "while" es que en este caso, el bloque de código se ejecutará al menos una vez, ya que la verificación de la expresión booleana se realiza al final de cada iteración.

Ciclo for

Ciclo "for": se utiliza para repetir un bloque de código un número fijo de veces.

En este ejemplo, se utiliza el ciclo "for" para repetir el bloque de código dentro de las llaves 10 veces, ya que la variable "i" se inicializa en uno y se incrementa en cada iteración hasta que alcanza el valor 11 que es cuando sale del ciclo.

Notar que siempre sale del ciclo con un valor más al indicado, es decir, cuando i valga 10 ejecuta las sentencias que estén dentro de las llaves, incrementa en 1 el valor de i (valiendo 11) y luego evalúa si es <= TOPE. De cumplirse esa evaluación ejecuta las sentencias dentro del ciclo, caso contrario sale del mismo.

```
static void Main(string[] args)
{
    const int TOPE = 10;
    for (int i = 1; i <= TOPE; i++)
    {
        Console.WriteLine("El valor de i es: " + i);
    }
}
```

Uno más... Ciclo Foreach

Estos son algunos de los ciclos más comunes en C#. Es importante recordar que cada tipo de ciclo tiene su propia sintaxis y puede ser útil en diferentes situaciones dependiendo del problema a resolver.

Te dejamos uno más.....

Ciclo "foreach": se utiliza para recorrer una colección de elementos, como un arreglo o una lista.

```
static void Main(string[] args)
{
    int[] numeros = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    foreach (int numero in numeros)
    {
        Console.WriteLine("El valor del número es: " + numero);
    }
}
```

En este ejemplo, se utiliza el ciclo "foreach" para recorrer el arreglo "numeros" y escribir en la consola el valor de cada elemento que lo lee en la variable numero.

Nota: cualquier ciclo, tanto for como do-while puede ser reemplazado por el ciclo while pero no a la inversa.

En resumen

Mediante el condicional if, hemos aprendido cómo tomar decisiones basadas en condiciones. Hemos explorado diversas formas de utilizar declaraciones if, anidando condicionales y combinando operadores lógicos y de comparación. Esta estructura nos permite ejecutar diferentes bloques de código según las condiciones que establezcamos, lo que nos brinda una gran flexibilidad en el diseño de nuestros programas.

La sintaxis básica del condicional "if" consta de una expresión booleana entre paréntesis, seguida de un bloque de código entre llaves que se ejecutará si la expresión booleana es verdadera. En caso contrario, se puede utilizar la cláusula "else" para especificar otro bloque de código que se ejecutará si la expresión booleana es falsa.

También se puede utilizar la cláusula "else if" para evaluar múltiples expresiones booleanas en orden y tomar diferentes acciones en función de cada una de ellas.

El uso adecuado del condicional "if" es fundamental para la creación de programas que respondan adecuadamente a las situaciones que se presenten durante su ejecución. Es importante tener en cuenta que la lógica de la expresión booleana debe ser cuidadosamente diseñada para garantizar que las condiciones se evalúen de forma correcta y que se tomen las decisiones adecuadas.

Estructura de control

Avanzando con el contenido, descubrimos la estructura de control switch, una alternativa al condicional if que nos permite manejar múltiples casos de una manera más compacta y legible. A través de casos y el bloque default, podemos tomar decisiones basadas en diferentes valores de una variable y ejecutar el bloque de código correspondiente. Esto resulta especialmente útil cuando tenemos una serie de opciones concretas que queremos evaluar.

Además vimos ciclos en C# que son la herramienta para repetir bloques de código varias veces, lo que permite realizar tareas de manera más eficiente y simplificar el código. Cada tipo de ciclo tiene su propia sintaxis y se utiliza para diferentes situaciones, dependiendo del problema a resolver.

Es importante tener en cuenta que el uso de ciclos puede afectar el rendimiento de la aplicación si no se utilizan adecuadamente. Por lo tanto, es necesario tener cuidado al diseñar los ciclos y asegurarse de que se estén utilizando de la manera más eficiente posible.

En conclusión, los ciclos son una parte esencial de cualquier lenguaje de programación, y en C# no son la excepción. Al aprender a utilizarlos adecuadamente, los desarrolladores pueden crear programas más eficientes y con una mejor estructura que permiten automatizar tareas repetitivas y hacer que el código sea más fácil de entender y mantener.

Ahora somos capaces de diseñar algoritmos lógicos y eficientes, tomando decisiones y repitiendo tareas según sea necesario.

Recordá que las estructuras de control son herramientas fundamentales en la programación y que su dominio te permitirá abordar problemas más complejos y desarrollar programas más robustos y flexibles.

Desarrollar un sistema es todo un desafío, ya que interpretar los requerimientos de un cliente muchas veces no es fácil pero no imposible. Lo importante es separar o sea desagregar el objetivo en pequeños procesos que sean independientes del resto, de esta forma la actualización en alguno de ellos no afectará al resto.

Importante

Recordá que esos procesos los transformaremos en líneas de código, que contendrán clases las cuales instanciaremos a objetos y estos se activan y transforman los datos a través de sus métodos.