DESARROLLO DE SOFTWARE

IDS340-02

Resumen Nivel 1 / Lección 1

Daniel Germán ID: 1128127

20 de Noviembre, 2024

En los lenguajes de programación existen los operadores, símbolos que indican a un sistema que debe hacer una parte de un programa.

CATEGORÍA	SÍMBOLO	NOMBRE	APLICACIÓN	DESCRIPCIÓN
Unarios	checked	Verificación de desbordamiento	checked(x)	Verifica si una variable llega **al máximo de números que puede contener,** mandando una excepción del sistema sin que reinicie la variable.
	unchecked	No verificación de desbordamiento	unchecked(x)	**No envía la excepción** por desbordamiento, reiniciando la variable.
	+	Valor positivo de	+x	**Convierte a positivo** el valor de x.
	-	Valor negativo de	-x	**Convierte a negativo** el valor de x.
	!	*Not*	!x	**Funciona con valores booleanos;** si es falso, lo convierte a verdadero y viceversa.
	~	Complemento	~x	Si 00000010 binario, entonces 11111101 binario, **transforma el valor** a su complemento binario.

CATEGORÍA	SÍMBOLO	NOMBRE	APLICACIÓN	DESCRIPCIÓN
Unarios	++	Preincremento	++x (x = 1;) (y = ++x;) (y es 2)	**Incrementa en uno** la variable en la misma sentencia.
		Post-incremento	x++ (x = 1;) (y = x++;) (y es 1)	**Incrementa en uno** la variable en la siguiente sentencia.
	--	Predecremento	--x (x = 1;) (y = --x;) (y es 0)	**Decrementa en uno** la variable en la misma sentencia.
		Posdecremento	x-- (x = 1;) (y = x--;) (y es 1)	**Decrementa en uno** la variable en la siguiente sentencia.
	()	*cast*	(int)x	Realiza la **conversión explícita.**
	*	Valor en dirección	*x	Obtiene **el valor al que apunta** la dirección. (No seguro)
	&	Dirección del valor	&x	Obtiene **la dirección** del valor. (No seguro)

CATEGORÍA	SÍMBOLO	NOMBRE	APLICACIÓN	DESCRIPCIÓN
Aritméticos	*	Multiplicación	x * y	**Multiplica** los argumentos.
	/	División	x / y	**Divide** los argumentos.
	%	Residuo	x % y	Obtiene **el residuo de la división** de los dos argumentos.
	+	Suma	x + y	**Suma los dos** argumentos.
	-	Resta	x - y	**Resta los dos** argumentos.
Desplazamiento	<<	Desplaza a la izquierda	x << y	Desplaza **hacia la izquierda** la cantidad de bits definidos en el segundo argumento, 00000001 se desplaza a 00000010.
	>>	Desplaza a la derecha	x >> y	Desplaza **hacia la derecha** la cantidad de bits definidos en el segundo argumento, 00001010 se desplaza a 00000101.
Asignación	=	Asignación	x = y	Asigna **el valor del segundo** argumento al primero.
	*=	Auto multiplicación	x *= y	Multiplica los argumentos y **asigna el resultado al primero.**

CATEGORÍA	SÍMBOLO	NOMBRE	APLICACIÓN	DESCRIPCIÓN
Asignación	/=	Auto división	x /= y	**Divide los argumentos** y asigna el resultado al primero.
	+=	Auto suma	x += y	**Suma los argumentos** y asigna el resultado al primero.
	-=	Auto resta	x -= y	**Resta los argumentos** y asigna el resultado al primero.
	<<=	Auto desplazamiento a la izquierda	x <<= y	**Desplaza a la izquierda** el número de bits del **segundo argumento** y asigna el resultado al primero.
	>>=	Auto desplazamiento a la derecha	x >>= y	**Desplaza a la derecha** el número de bits del **segundo argumento** y asigna el resultado al primero.
	as	Conversión de tipo	x as y	**Realiza la conversión** de tipo de primer argumento con el tipo del segundo si éstas son compatibles **de no serlo se regresa un nulo.**

CATEGORÍA	SÍMBOLO	NOMBRE	APLICACIÓN	DESCRIPCIÓN
Unario	?.	Condicional nulo	x?.y	Evalúa si el primer o el segundo argumento **son nulos.** Si alguno de los dos lo es, regresa **un nulo.**
Relacional	<	Menor que	x < y	Si **x** es menor que **y**, la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).
	>	Mayor que	x > y	Si **x** es mayor que **y**, la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).
	<=	Menor igual a	x <= y	Si **x** es menor o igual a **y**, la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).
	>=	Mayor igual a	x >= y	Si **x** es mayor o igual a **y**, la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).
	is	Compatibilidad de tipo	x is y	Si los tipos de valor o referencia son **compatibles,** la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).
Igualdad	==	Igual	x == y	Si **x** es igual a **y**, la expresión es verdadera (*true*); de lo contrario, será falsa (*false*).

Igualdad	!=	Diferente	x != y	Si **x** es diferente a **y**, la expresión es verdadera (*true*); si es igual entonces será falsa (*false*).
Lógicas	&	And	x & y	Realiza el equivalente a una compuerta **and** por cada bit de las dos variables, por ejemplo: 10101010 & 11110000 = 10100000
	^	Xor	x ^ y	Realiza el equivalente a una compuerta **xor** por cada bit de las dos variables, por ejemplo: 10101010 ^ 11110000 = 01011010
	|	Or	x | y	Realiza el equivalente a una compuerta **or** por cada bit de las dos variables, por ejemplo: 10101010 | 11110000 = 11111010
Condicionales	??	Unión nula	x = y ?? w	Si **y** es nulo, entrega **w**; de lo contrario, entrega **y**.
	?:	Condicional	x ? y : w	Si la expresión **x** es verdadera (*true*), entrega **y**; de lo contrario, entrega **w**.
	&&	Condicional and	x && y	La expresión es verdadera (*true*) sólo si **x** y **y** son verdaderas.
	||	Condicional or	x || y	La expresión es verdadera (*true*) sólo cuando **x** o **y** son verdaderas.

“Null” se utiliza para definir valores que no existen. Un arreglo es una estructura de datos que permite almacenar varios elementos del mismo tipo en una única variable. Es una colección de elementos, que se acceden a través de un índice numérico. Los arreglos en C# son de tamaño fijo, lo que significa que una vez que se define el tamaño de un arreglo, no se puede cambiar durante la ejecución del programa.

Una estructura es un tipo de valor que permite agrupar datos relacionados en un solo objeto. A diferencia de las clases, las estructuras son tipos de valor, lo que significa que se copian completamente cuando se pasan a métodos o se asignan a otras variables. Las estructuras son útiles cuando se necesita almacenar datos de diferentes tipos bajo un mismo nombre sin la sobrecarga que implica el uso de clases.

Sentencias de selección: Permiten decidir qué bloque de código ejecutar dependiendo de una condición.
Sentencias iterativas: Permiten repetir un bloque de código en un loop.
Sentencias de salto: Permiten alterar el flujo de ejecución del programa.

```
using System;

class Prueba
{
    static void Main()
    {
        int[] fibonacci = new int[100];
        fibonacci[0] = 0;
        fibonacci[1] = 1;

        for (int i = 2; i < 100; i++)
        {
            fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
        }

        Console.WriteLine("Los primeros 100 números de la serie de
Fibonacci son:");
        for (int i = 0; i < 100; i++)
        {
            Console.WriteLine(fibonacci[i]);
        }
    }
```

```
}
```