Academy .NET

Operatori, espressioni e istruzioni

Sommario

- Operatori in C# e precedenza degli operatori
- Operatori aritmetici
- Operatori logici
- Operatori bit per bit
- Operatori di confronto
- Operatori di assegnazione
- Altri operatori
- Conversioni di tipo implicite ed esplicite

Cosa è un operatore?

- L'operatore svolge una operazione eseguita a **runtime**
  - Accetta uno o più argomenti (operandi)
  - Produce un nuovo valore

- Gli operatori hanno la precedenza:
  - La precedenza definisce quale sarà invocato per primo
  - Le espressioni sono sequenze di operatori e operandi valutati in base a un singolo valore

Operatori in C#

- Operatori in C#

  Unario: operazione con un solo operando

  Binario: operazione con due operandi

  Ternario (**? :**) - operazione con tre operandi

- Ad eccezione degli operatori di assegnazione, tutti gli operatori binari sono associativi a sinistra

- Gli operatori di assegnazione e l'operatore condizionale (? :) sono associativi a destra

Categorie di operatori in C#

Categoria	Operatori
Aritmetici	+ - * / % ++ --
Logici	&& || ^ !
Binari	& | ^ ~ << >>
Di Comparazione	== != < > <= >=
Di Assegnazione	= += -= *= /= %= &= |= ^= <<= >>=
Concatenazione di Stringhe	+
Conversione di Tipo	is as typeof
Altri	. [] () ?: new

Precedenza degli operatori

Precedenza	Operatori
Precedenza maggiore	()
	++ -- (postfix) new typeof
	++ -- (prefix) + - (unary) ! ~
	* / %
	+ -
	<< >>
	< > <= >= is as
	== !=
	&
Precedenza minore	^

Precedenza degli operatori

Precedenza	Operatori
Precedenza maggiore	|
	&&
	||
	?:
Precedenza minore	= *= /= %= += -= <<= >>= &= ^= |=

- L'operatore fra parentesi ha sempre la precedenza

Operatori Aritmetici

- Gli operatori aritmetici **+**, **-**, ***** sono gli stessi della matematica
- L'operatore di divisione **/** se utilizzato su numeri interi restituisce un numero intero (senza arrotondamenti) o **un'eccezione**
- L'operatore di divisione **/** se utilizzato su numeri in virgola mobile restituisce il **numero reale** o **Infinity** o **NaN**
- L'operatore **modulo %** restituisce il resto dalla divisione di numeri interi
- L'operatore **++** incrementa una variabile

Operatori Aritmetici - Esempio

```
int squarePerimeter = 17;
double squareSide = squarePerimeter / 4.0;
double squareArea = squareSide * squareSide;
Console.WriteLine(squareSide); // 4.25
Console.WriteLine(squareArea); // 18.0625
int a = 5;
int b = 4;
Console.WriteLine( a + b ); // 9
Console.WriteLine( a + b++ ); // 9
Console.WriteLine( a + b ); // 10
Console.WriteLine( a + (++b) ); // 11
Console.WriteLine( a + b ); // 11

Console.WriteLine(12 / 3); // 4
Console.WriteLine(11 / 3); // 3
```

Operatori Aritmetici – Esempio 2

```
Console.WriteLine(11.0 / 3); // 3.666666667
Console.WriteLine(11 / 3.0); // 3.666666667
Console.WriteLine(11 % 3);   // 2
Console.WriteLine(11 % -3);  // 2
Console.WriteLine(-11 % 3);  // -2

Console.WriteLine(1.5 / 0.0);  // Infinity
Console.WriteLine(-1.5 / 0.0); // -Infinity
Console.WriteLine(0.0 / 0.0);  // NaN

int x = 0;
Console.WriteLine(5 / x); // DivideByZeroException
```

Operatori Aritmetici – Overflow

```
int bigNum = 2000000000;
int bigSum = 2 * bigNum; // Integer overflow!
Console.WriteLine(bigSum); // -294967296

bigNum = Int32.MaxValue;
bigNum = bigNum + 1;
Console.WriteLine(bigNum); // -2147483648

checked
{
  // This will cause OverflowException
  bigSum = bigNum * 2;
}
```

Operatori Logici

- Gli operatori logici accettano operandi booleani e restituiscono il risultato booleano
- L' operatore **!** restituisce l'operatore booleano **inverso**

- Comportamento degli operatori **&&** e **||** (1 == true, 0 == false):

Operazione	||	||	||	||	&&	&&	&&	&&				
Operando1	0	0	1	1	0	0	1	1				
Operando2	0	1	0	1	0	1	0	1				
Risultato	0	1	1	1	0	0	0	1				

Operatori Logici - Esempi

```
bool a = true;
bool b = false;
Console.WriteLine(a && b); // False
Console.WriteLine(a || b); // True
Console.WriteLine(a ^ b); // True
Console.WriteLine(!b); // True
Console.WriteLine(b || true); // True
Console.WriteLine(b && true); // False
Console.WriteLine(a || true); // True
Console.WriteLine(a && true); // True
Console.WriteLine(!a); // False
Console.WriteLine((5>7) ^ (a==b)); // False
```

Operatori di Confronto

- Gli operatori di confronto vengono utilizzati per confrontare le variabili

  ==, <,>,> =, <=,! =

Esempio di operatori di confronto:

```
int a = 5;
int b = 4;
Console.WriteLine(a >= b); // True
Console.WriteLine(a != b); // True
Console.WriteLine(a == b); // False
Console.WriteLine(a == a); // True
Console.WriteLine(a != ++b); // False
Console.WriteLine(a > b); // False
```

Operatori di Assegnazione

- Gli operatori di assegnazione vengono utilizzati per assegnare un valore a una variabile:

  =, +=, -=,...

Esempio di operatori di assegnazione:

```
int x = 6;
int y = 4;
Console.WriteLine(y *= 2);
Console.WriteLine(x += 3);
Console.WriteLine(x /= 2);
```

Altri Operatori

- L'operatore di concatenazione di stringhe **+** viene utilizzato per concatenare le stringhe
- Se il secondo operando non è una stringa, viene convertito automaticamente in stringa

```
string first = "First";
string second = "Second";
Console.WriteLine(first + second);
// FirstSecond
string output = "The number is : ";
int number = 5;
Console.WriteLine(output + number);
// The number is : 5
```

Altri Operatori

- L’operatore di accesso ai membri (**.**) viene utilizzato per accedere ai membri dell'oggetto

- L'operatore di **cast (type)** viene utilizzato per eseguire il cast di un tipo compatibile con un altro

Altri Operatori

- Operatore ternario **?:** ha la forma:

```
b ? x : y
```

- (se **b** è vero allora il risultato è **x** altrimenti il risultato è **y**)
- Il nuovo operatore viene utilizzato per creare nuovi oggetti
- L'operatore **typeof** restituisce l'oggetto **System.Type**
- L'operatore **is** controlla se un oggetto è compatibile con un dato tipo

Altri Operatori

- Operatore **null-coalescing ??** viene utilizzato per definire un valore predefinito sia per i tipi di valore **Nullable** che per i **tipi di riferimento**

- Restituisce l'operando sinistro se non è nullo, altrimenti restituisce l'operando destro

```
int? x = null;
int y = x ?? -1;
```

```
int? x = 1;
int y = x ?? -1;
```

Utilizzo di alcuni operatori

```
int a = 6;
int b = 4;
Console.WriteLine(a > b ? "a>b" : "b>=a"); // a>b
Console.WriteLine((long) a); // 6
int c = b = 3; // b=3; followed by c=3;
Console.WriteLine(c); // 3
Console.WriteLine(a is int); // True
Console.WriteLine((a+b)/2); // 4
Console.WriteLine(typeof(int)); // System.Int32
int d = new int();
Console.WriteLine(d); // 0
```

Conversioni Esplicita e Implicita

- **Conversione di tipo implicita**

  Conversione automatica del valore di un tipo di dato in valore di un altro tipo di dato

  Consentito quando non è possibile alcuna perdita di dati

  I tipi "più grandi" possono assumere implicitamente valori di tipi "più piccoli"

  Esempio:

```
int i = 5;
long l = i;
```

Conversioni Esplicita e Implicita

- **Conversione di tipo esplicita**

  Conversione manuale di un valore di un tipo di dato in un valore di un altro tipo di dato

  Consentito solo esplicitamente dall'operatore di **cast (type)**

  Necessario in caso di possibilità di perdita di dati o precisione

  Esempio:

```
long l = 5;
int i = (int) l;
```

Conversioni Esplicita e Implicita: Esempi

- Esempio di conversioni implicite ed esplicite:

```
float heightInMeters = 1.74f; // Explicit conversion
double maxHeight = heightInMeters; // Implicit

double minHeight = (double) heightInMeters; // Explicit

float actualHeight = (float) maxHeight; // Explicit

float maxHeightFloat = maxHeight; // Compilation error!
```

- **Nota**: la conversione esplicita può essere utilizzata anche se non richiesta dal compilatore

Domande & approfondimenti

Academy .NET