**Operatori**

http://msdn.microsoft.com/it-it/library/s0b0c0z1.aspx

Gli operatori vengono utilizzati per eseguire calcoli, assegnare valori a variabili, testare l'uguaglianza o la disuguaglianza ed eseguire altre operazioni.

Nelle sezioni seguenti vengono elencati alcuni degli operatori più utilizzati nel linguaggio C#. Per un elenco completo di tutti gli operatori, vedere più avanti.

* **Operatori di assegnazione e di uguaglianza**

In C#, l'operatore del segno di uguale (=) ha la stessa funzionalità che in C e C++:

|  |  |
| --- | --- |
| **Operatore** | **Funzione** |
| = | Assegna un valore. |
| = = | Esegue il test per verificare l'uguaglianza. |

### Esempio

|  |
| --- |
| int x = 100;  if (x == 100)  {  System.Console.WriteLine("X is equal to 100");  } |

* **Operatori matematici e logici**

Di seguito viene riportato un elenco di operatori matematici di base, in ordine di precedenza. Utilizzare le parentesi per imporre un altro ordine.

|  |  |
| --- | --- |
| **Operatore** | **Funzione** |
| \*, /, % | Moltiplicazione, divisione, modulo |
| +, - | Addizione, sottrazione |
| & | AND logico |
| ^ | XOR logico |
| | | OR logico |

### Esempio

|  |
| --- |
| int x = 1;  int y = x + 10 \* 100; // multiplication first y = 1001  int z = (x + 10) \* 100; // addition first z = 1100 |

* **Operatori di incremento e decremento**

I collegamenti di tipo C/C++ sono supportati, compresi gli operatori suffissi e prefissi, come illustrato negli esempi seguenti:

|  |  |
| --- | --- |
| **Operatore** | **Funzione** |
| V++ | Incrementare la variabile v di un'unità. |
| V += N | Incrementare la variabile v di n unità. |
| V \*= N | Moltiplicare la variabile v per n unità. |
| V -= N | Sottrarre n unità dalla variabile v. |

### Esempio

|  |
| --- |
| int x = 0;  int y = x++; // x is 1, y is 0  System.Console.WriteLine("{0} {1}", x, y);  int z = ++x; // x is 2, z is 2  System.Console.WriteLine("{0} {1}", x, z); |

* **Operatori relazionali.**

Gli operatori riportati di seguito confrontano due valori e restituiscono un risultato Booleano:

|  |  |
| --- | --- |
| **Operatore** | **Funzione** |
| = = | Esegue il controllo per verificare l'uguaglianza. |
| != | Esegue il controllo per verificare la disuguaglianza. |
| > | Maggiore di |
| < | Minore di |
| >= | Maggiore o uguale a |
| <= | Minore o uguale a |

### Esempio

|  |
| --- |
| int x = int.Parse(System.Console.ReadLine());  if (x > 100)  {  System.Console.WriteLine("X is greater than 100");  } |

* **Operatori logici di condizione**

Gli operatori logici vengono utilizzati per creare istruzioni di condizione più flessibili tramite la combinazione di più clausole:

|  |  |
| --- | --- |
| **Operatore** | **Funzione** |
| && | AND condizionale. |
| || | OR condizionale. |
| ! | NOT condizionale. |

### Esempio

|  |
| --- |
| int x = int.Parse(System.Console.ReadLine());  if ((x >= 100) && (x <= 200))  {  System.Console.WriteLine("X is between 100 and 200");  } |

* **Operatori matematici più avanzati**

Per eseguire operazioni matematiche più avanzate, ad esempio di trigonometria, utilizzare la classe Math di Frameworks. Nell'esempio seguente, vengono utilizzati i metodi Sin (seno), Sqrt (radice quadrata) e la costante PI:

### Esempio

|  |
| --- |
| double d = System.Math.Sin(System.Math.PI/2);  double e = System.Math.Sqrt(144); |

* **Elenco completo degli operatori**

In C# è disponibile un elevato numero di operatori, ovvero simboli utilizzati per specificare quali operazioni eseguire in un'espressione.

Operatore di negazione logica

!

Questo operatore calcola la negazione logica dell'operando; se l'operando è true, il risultato sarà false. Se l'operando è false, il risultato sarà true.

Parentesi

(T)x

Oltre che per specificare l'ordine delle operazioni in un'espressioni, le parentesi vengono utilizzate anche per specificare i cast, ovvero le conversioni del tipo:

double x = 1234.7;

int a;

a = (int)x; // Cast double to int

Addizione

+

Gli operatori + binari sono predefiniti per i tipi numerici e tipi stringa. Per i tipi numerici, l'operatore + calcola la somma dei due operandi. Quando entrambi gli operandi sono di tipo stringa, l'operatore + concatena le rappresentazioni in formato stringa degli operandi.

Sottrazione

-

Incremento

++x e x++

L'operatore di incremento (++) incrementa il proprio operando di 1 e può essere visualizzato prima o dopo tale operando:

Il primo formato rappresenta un'operazione di incremento prefissa. Il risultato dell'operazione corrisponde al valore dell'operando dopo l'incremento.

Il secondo formato rappresenta un'operazione di incremento suffissa. Il risultato dell'operazione corrisponde al valore dell'operando prima dell'incremento.

class MainClass

{

static void Main()

{

double x;

x = 1.5;

Console.WriteLine(++x);

x = 1.5;

Console.WriteLine(x++);

Console.WriteLine(x);

}

}

/\*

Output

2.5

1.5

2.5

\*/

Decremento

--x e x--

L'operatore di decremento (**--**) decrementa il proprio operando di 1 e può essere visualizzato prima o dopo tale operando: --variable e variable-- Il primo formato rappresenta un'operazione di decremento prefissa. Il risultato dell'operazione corrisponde al valore dell'operando dopo il decremento. Il secondo formato rappresenta un'operazione di decremento suffissa. Il risultato dell'operazione corrisponde al valore dell'operando prima del decremento.

using System;

class MainClass

{

static void Main()

{

double x;

x = 1.5;

Console.WriteLine(--x);

x = 1.5;

Console.WriteLine(x--);

Console.WriteLine(x);

}

}

/\*

Output

0.5

1.5

0.5\*/

Moltiplicazione

\*

Divisione

/

L'operatore di divisione (/) Divide il primo operando dal secondo operando. Quando si dividono due numeri interi, il risultato è sempre un numero intero. Ad esempio, il risultato di 7 / 3 è 2. Per determinare la parte restante del 7 / 3, utilizzare l'operatore di resto (%). Per ottenere un quoziente come numero razionale o frazione, assegnare al dividendo o al divisore il tipo float o double. Se è necessario esprimere il dividendo o il divisore come numero decimale inserendo una cifra a destra del separatore decimale, come illustrato nell'esempio riportato di seguito, è possibile assegnare in modo implicito il tipo.

class Division

{

static void Main()

{

Console.WriteLine("\nDividing 7 by 3.");

// Integer quotient is 2, remainder is 1.

Console.WriteLine("Integer quotient: {0}", 7 / 3);

Console.WriteLine("Negative integer quotient: {0}", -7 / 3);

Console.WriteLine("Remainder: {0}", 7 % 3);

// Force a floating point quotient.

float dividend = 7;

Console.WriteLine("Floating point quotient: {0}", dividend / 3);

Console.WriteLine("\nDividing 8 by 5.");

// Integer quotient is 1, remainder is 3.

Console.WriteLine("Integer quotient: {0}", 8 / 5);

Console.WriteLine("Negative integer quotient: {0}", 8 / -5);

Console.WriteLine("Remainder: {0}", 8 % 5);

// Force a floating point quotient.

Console.WriteLine("Floating point quotient: {0}", 8 / 5.0);

}

}

// Output:

//Dividing 7 by 3.

//Integer quotient: 2

//Negative integer quotient: -2

//Remainder: 1

//Floating point quotient: 2.33333333333333

//Dividing 8 by 5.

//Integer quotient: 1

//Negative integer quotient: -1

//Remainder: 3

//Floating point quotient: 1.6

%

L'operatore modulo (%) calcola il resto dopo aver diviso il primo operando per il secondo.

Esempio

class MainClass6

{

static void Main()

{

Console.WriteLine(5 % 2); // int

Console.WriteLine(-5 % 2); // int

Console.WriteLine(5.0 % 2.2); // double

Console.WriteLine(5.0m % 2.2m); // decimal

Console.WriteLine(-5.2 % 2.0); // double

}

}

/\*

Output:

1

-1

0.6

0.6

-1.2

\*/

Relazione e verifica del tipo

<

>

<=

>=

Uguaglianza

==

!=

AND logico

&

L'operatore & valuta entrambi gli operatori indipendentemente dal valore del primo.

Esempio:

int i = 0;

if (false & ++i == 1)

{

// i is incremented, but the conditional expression evaluates to false, so

// this block does not execute.

}

Esempio

class BitwiseAnd

{

static void Main()

{

Console.WriteLine(true & false); // logical and

Console.WriteLine(true & true); // logical and

}

}

/\*

Output:

False

True

\*/

XOR logico

^

OR logico

|

AND condizionale

&&

L'operatore AND condizionale (&&) esegue un AND logico degli operandi bool, ma valuta il secondo operando solo se necessario.

L'operazione

x && y

corrisponde all'operazione

x & y

con la differenza che, se x è false, y non verrà valutato, in quanto il risultato dell'operazione di AND è false indipendentemente dal valore di y . Questa condizione viene denominata anche valutazione "short circuit".

L'operatore AND condizionale non può essere sottoposto a overload. Tuttavia, anche gli overload dei normali operatori logici e degli operatori true e false vengono considerati, con qualche limitazione, overload degli operatori logici condizionali.

Esempio

class LogicalAnd

{

static bool Method1()

{

Console.WriteLine("Method1 called");

return false;

}

static bool Method2()

{

Console.WriteLine("Method2 called");

return true;

}

static void Main()

{

Console.WriteLine("regular AND:");

Console.WriteLine("result is {0}", Method1() & Method2());

Console.WriteLine("short-circuit AND:");

Console.WriteLine("result is {0}", Method1() && Method2());

}

}

/\*

Output:

regular AND:

Method1 called

Method2 called

result is False

short-circuit AND:

Method1 called

result is False

\*/

OR condizionale

||

Assegnazione

=

+=

-=

\*=

/=

%=

&=

|=

^=

Condizionale

?:

L'operatore condizionale (?:) restituisce uno tra due valori in base al valore di un'espressione booleana. L'operatore condizionale presenta il formato:

condition ? first\_expression : second\_expression;

I calcoli che richiedono una costruzione if-else possono essere espressi più concisamente ed elegantemente con l'operatore condizionale. Per evitare, ad esempio, una divisione per zero nel calcolo della funzione sin, è possibile scrivere

if(x != 0.0)

s = s \* s;

else

s = 1.0;

oppure, utilizzando l'operatore condizionale,

s = x != 0.0 ? s \* s : 1.0;

??

L'operatore ?? viene chiamato operatore null-coalescing. Restituisce l'operando sinistro se non è Null. In caso contrario, restituisce l'operando destro.

int? x = null;

// Set y to the value of x if x is NOT null; otherwise,

// if x = null, set y to -1.

int y = x ?? -1;

**Nota:**

Mediante il modificatore dei tipi nullable, in C# è possibile creare variabili di tipo di valore che indicano un valore non definito. È possibile utilizzare qualsiasi tipo di valore come base per un tipo nullable. Di seguito è riportato un esempio:

double? d1 = 3.14;

bool? flag = null;

int?[] arr = new int?[10];