Operatori tra interi: esempi

37 / 3	→	12
37 % 3	→	1
7 < 3	→	0
7 >=3	→	1
5 >=5	→	1
0 || 1	→	1
0 || -123	→	1
15||0	→	1
12 && 2	→	1
0 && 17	→	0
! 2	→	0

Operazioni su reali: esempi

5.0 / 2 → 2.5
2.1 / 2 → 1.05
7.1 < 4.55 → 0
17== 121 → 0

➔ A causa della rappresentazione finita, ci possono essere errori di conversione. Ad esempio, i test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.

(x / y) * y == x

Meglio utilizzare "un margine accettabile di errore":
(X == Y) → **`(X<=Y+epsilon) && (X>=Y-epsilon)`**

dove, ad esempio: **`float epsilon=0.000001;`**

Operazioni sui char: esempi

Esempi:

'A' < 'C'	→	1	(infatti 65 < 67 è vero)
' '' ' + ' ! '	→	'C'	(codice('')+codice(!)=67)
! 'A'	→	0	(codice(A) è diverso da 0)
'A' && 'a'	→	1	

OVERLOADING

- In C (come in Pascal, Fortran e molti altri linguaggi) operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo (ad esempio, le operazioni aritmetiche su reali o interi).
- In realtà l'operazione è diversa e può produrre risultati diversi. Per esempio:

```
int X,Y;
se X = 10 e Y = 4;
X/Y vale 2
```

```
int X; float Y;
se X = 10 e Y = 4.0;
X/Y vale 2.5
```

```
float X,Y;
se X = 10.0 e Y = 4.0;
X/Y vale 2.5
```

Esempio:

```
main()
{
/*definizioni variabili: */
char y='a'; /*codice(a)=97*/
int    x,X,Y;
unsigned int Z;
float SUM;
double r;

/* parte istruzioni: */
X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10; /* qui X=30, Y=100, Z=70 */
X = X + 70;
Y  = Y % 10;
Z = Z + X -70;
SUM = Z * 10;  /* X=100, Y=0, Z=100 , SUM=1000.0*/
x=y;     /* char -> int: x=97*/
x=y+x; /*x=194*/
r=y+1.33; /* char -> int -> double*/
x=r; /* coercizione -> troncamento: x=98*/
}
```

Assegnamento come operatore

Formalmente, **l'istruzione di assegnamento è un'espressione**:

- Il simbolo **=** è un operatore:
  - → l'istruzione di assegnamento è una espressione
  - → ritorna un valore:
    - il valore restituito è quello assegnato alla variabile a sinistra del simbolo =
    - il tipo del valore restituito è lo stesso tipo della variabile oggetto dell'assegnamento

Ad esempio:

```
int valore=122;
int K, M;

K=valore+100; /* K=122;l'espressione
                 produce il
                 risultato 222 */
M=(K=K/2)+1;  /* K=111, M=112*/
```

Incremento & decremento: esempi

```
int A=10, B;
char C='a';

B=++A;        /*A e B valgono 11 */
B=A++;        /* A=12, B=11 */
C++;                /* C vale 'b' */

int i, j, k;
k = 5;
i = ++k; /*  i = 6, k = 6 */
j = i + k++; /* j=12, i=6,k=7 */

j = i + k++; /*equivale a:j=i+k; k=k+1;*/
```

- In C l'ordine di valutazione degli operandi non e' indicato dallo standard: si possono scrivere espressioni il cui valore e` difficile da predire:

```
k = 5;
j = ++k * k++; /* quale effetto ?*/
```

Operatore sequenziale

Un'espressione sequenziale (o di ***concatenazione***) si ottiene concatenando tra loro più espressioni con l'operatore virgola (**`,`**).

(<espr1>, <espr2>, <espr3>, .. <esprN>)

- Il risultato prodotto da un'espressione sequenziale e` il risultato ottenuto dall'ultima espressione della sequenza.
- La valutazione dell'espressione avviene valutando nell'ordine testuale le espressioni componenti, **da sinistra verso destra**.

Esempio:

```
int A=1;
char B;
A=(B='k', ++A, A*2);        /* A=4 */
```

Esempi

Sia V=5, A=17, B=34. Determinare il valore delle seguenti espressioni :

A<=20 || A>=40	→	***vero*** {A<20}
! (B=A*2)	→	***falso*** {B=34=17*2}
A<=B && A<=V	→	***falso*** {A>V}
A<=(B&&A)<=V	→	***vero*** {A<=1<=V}
A>=B && A>=V	→	***falso*** {A<B}
! (A<=B && A<=V)	→	***vero***
! (A>=B) || !(A<=V)	→	***vero***
(A++, B=A, V++, A+B+V++)	→	***42*** (A=18, B=18, V=7)