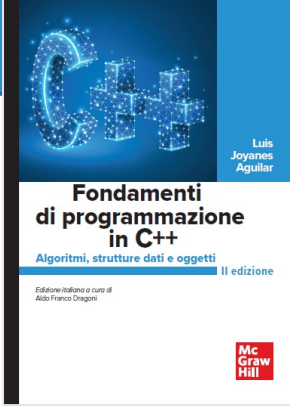
Capitolo 3

pag. 65-87

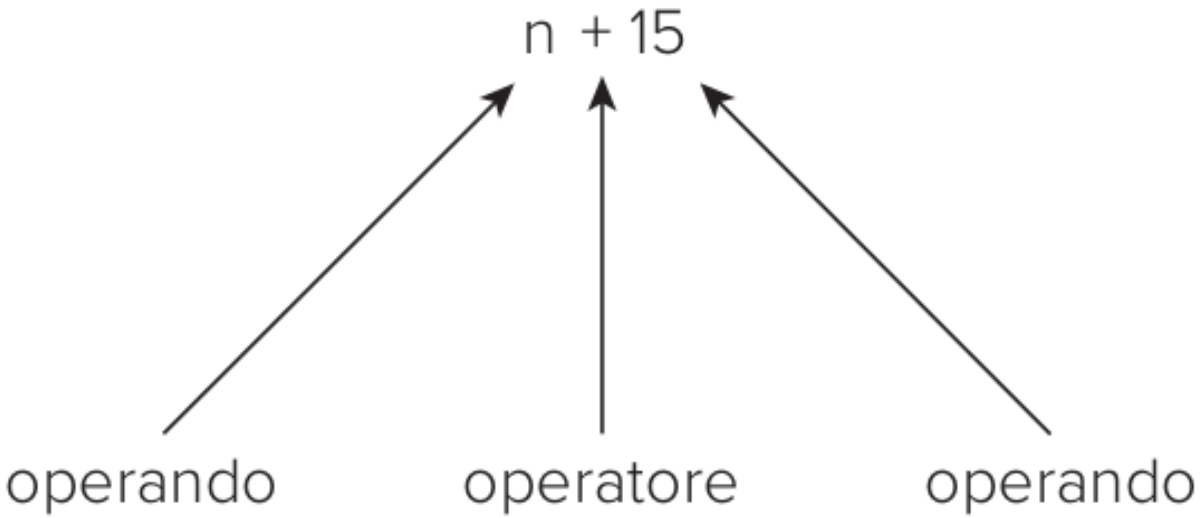
Operatori ed espressioni

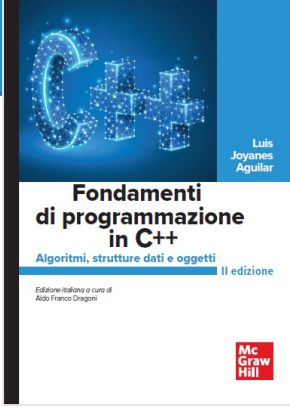
Presenta: Prof. Misael Mongiovì



espressioni

Un'espressione è costituita da uno o più operandi che si combinano tra loro tramite operatori per impostare un'operazione e restituire un risultato.



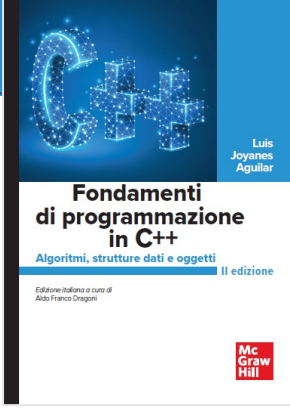


Supponiamo definite e inizializzate due variabili di tipo intero, `i` *e* `s`*;*

```
int i=1, s=15;
```

quella che segue è una lista di espressioni (alcune sono anche istruzioni)

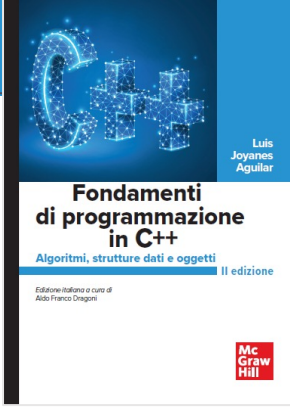
```
false          // restituisce il valore false
s              // restituisce il valore 15 (valore della variabile)
25             // restituisce il valore 25 (costante letterale)
s + 25         // restituisce il valore 40 (espressione aritmetica)
s == 40        // operatore di confronto, verifica l'uguaglianza dei
               // valori, in questo caso restituisce il valore false
               // perché 15 è diverso da 40
i++;           // istruzione che utilizza l'operatore di post-incremento;
               // aumenta di 1 il valore della variabile i, ma restituisce
               // il valore che essa aveva prima dell'incremento, cioè 1
i = s + 5;     // istruzione di assegnamento; anch'essa è un'espressione
               // e restituisce un "left value", cioè l'indirizzo della
               // variable i (non restituisce 20)
(10+s)/(i*3) // espressione che restituisce 0
```



operatori predefiniti del C++ precedenze ed associatività

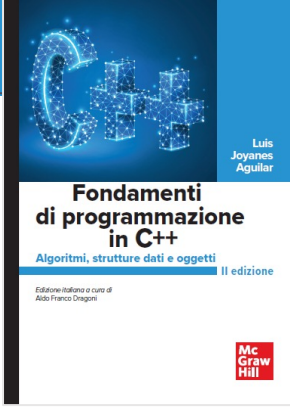
Priorità	Operatori		Associatività
17	::	risolutore di visibilità	
16	a++ a--	post-incremento, post-decremento	sinistra
	type() type{}	"convertitore di tipo" funzionale	
	a()	chiamata funzionale	
	a[]	indice vettoriale	
	. ->	accesso al membro	
15	++a --a	pre-incremento, pre-decremento	destra
	+a -a	più e meno unari	
	! ~	"not" logico, "not" bit-a-bit	
	(type)	"convertitore di tipo" stile "C"	
	*a	operatore di "indirezione"	
	&a	"indirizzo-di"	
	sizeof	"dimensione-di"	
	co_await	await-espressione (dal C++20)	
	new new[]	allocazione memoria dinamica	
	delete delete[]	deallocazione memoria dinamica	

Capitolo 3. Operatori ed espressioni



Priorità	Operatori		Associatività
14	.* ->*	puntatore-a-membro	
13	a*b a/b a%b	moltiplicazione, divisione, resto	
12	a+b a-b	addizione, sottrazione	
11	<< >>	shift a sinistra, shift a destra	
10	<=>	confronto trimodale (dal C++20)	
9	< <=	minore, minore o uguale	sinistra
	> >=	maggiore, maggiore o uguale	
8	== !=	uguale, diverso	
7	&	"and" bit-a-bit	
6	^	"or" esclusivo bit-a-bit	
5	|	"or" bit-a-bit	
4	&&	"and" logico	
3	||	"or" logico	
2	a?b:c	operatore condizionale	destra
	throw	operatore "throw"	
	co_yield	yield-espressione (dal C++20)	
	=	assegnamento	
	+= -=	assegnamento composto	
	*= /= %=	assegnamento composto	
	<<= >>=	assegnamento composto	
	&= ^= |=	assegnamento composto	
1	,	virgola	sinistra

operatori predefiniti del C++ precedenze ed associatività



operatori aritmetici

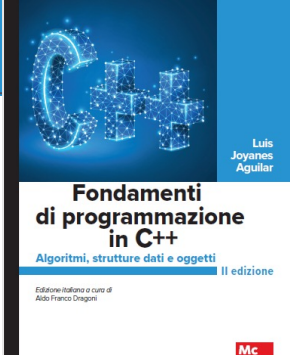
Operatore	Tipi interi	Tipi reali	Esempio
+	somma	somma	4 + 5
-	sottrazione	sottrazione	7 - 3
*	prodotto	prodotto	4 * 5
/	quoziente della divisione intera	divisione	8 / 5
%	resto della divisione intera		12 % 5

assegnamento composto

Simbolo	Uso	Equivale a	Descrizione
=	a = b		assegna il valore di *b* alla variabile *a*
+=	a += b	a = a + b;	somma *a* e *b* e assegna il risultato alla variabile *a*
-=	a -= b	a = a - b;	sottrae *b* ad *a* e assegna il risultato alla variabile *a*
*=	a *= b	a = a * b;	moltiplica *a* per *b* e assegna il risultato alla variabile *a*
/=	a /= b	a = a / b;	divide *a* per *b* e assegna il risultato alla variabile *a*
%=	a %= b	a = a % b;	mette in *a* il resto della divisione intera di *a* per *b*

operatori di incremento (++) e decremento (– –)

	Sintassi	Equivale a	Cioè a	Restituisce
Preincremento	`++x;`	`x += 1;`	`x = x + 1;`	la variabile `x`
Postincremento	`x++;`	`x += 1;`	`x = x + 1;`	il valore della variabile `x`
Predecremento	`--x;`	`x -= 1;`	`x = x - 1;`	la variabile `x`
Postdecremento	`x--;`	`x -= 1;`	`x = x - 1;`	il valore della variabile `x`
	`y=x++;`	`y = x;` `x += 1;`		
	`y=++x;`	`x += 1;` `y = x;`		



operatori relazionali

Operatore	Significato	Esempio
==	*Uguale a*	a == b
!=	*Non uguale a*	a != b
>	*Maggiore di*	a > b
<	*Minore di*	a < b
>=	*Maggiore o uguale di*	a >= b
<=	*Minore o uguale di*	a <= b

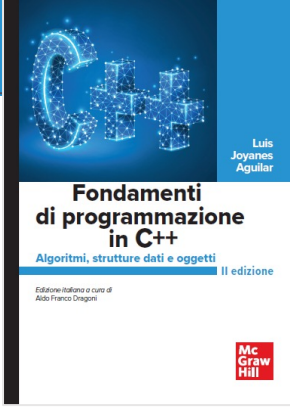
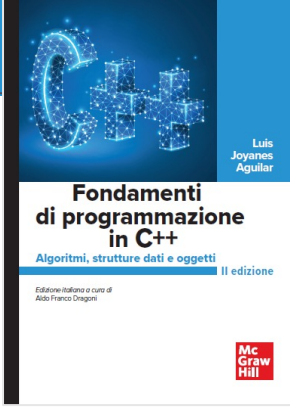


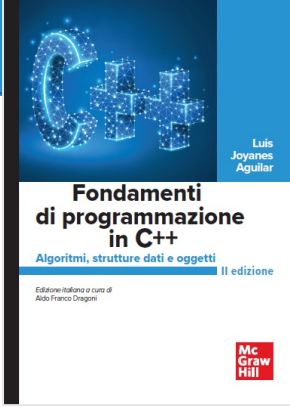
tabelle di verità degli operatori logici

a	b	a && b	a || b	!a
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true



operatori logici bit a bit (*bitwise*)

		AND	OR *inclusivo*	OR *esclusivo (XOR)*	Negazione
a	b	a & b	a | b	a ^ b	~a
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1



operatori bitwise composti

Simbolo	Uso	Descrizione
<<=	a <<= b	Sposta a sinistra b bit e assegna il risultato ad a
>>=	a >>= b	Sposta a destra b bit e assegna il risultato ad a
&=	a &= b	Assegna ad a il valore a & b
^=	a ^= b	Assegna ad a il valore a ^ b
|=	a |= b	Assegna ad a il valore a | b

operatore condizionale

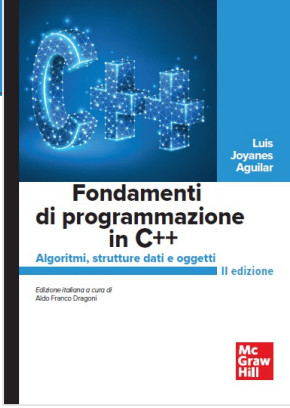
Il formato dell'operatore condizionale è:

espressione_booleana **?** *se_vera* **:** *se_falsa*

Si valuta *espressione_booleana*, se è `true` viene restituito il valore dell'espressione *se_vera* altrimenti viene restituito il valore dell'espressione *se_falsa*.

Esempio

```
n >= 0 ? 1 : -1    // restituisce 1 se n è positivo, -1 se è negativo
m >= n ? m : n     //restituisce il maggiore tra m ed n
```



operatore virgola

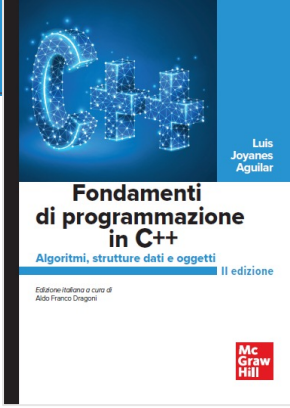
L'operatore *virgola* (,) permette di combinare due o più espressioni che verranno valutate da sinistra verso destra. L'espressione più a destra restituisce il risultato globale.

espressione_1, espressione_2, espressione_3, ..., espressione_n

Per esempio, in

```
int i = 10, j = 25;
```

viene prima dichiarata e inizializzata la `i` e poi la `j`.



operatore `sizeof`

L'operatore sizeof serve per conoscere la dimensione in byte di un tipo di dato o di una variabile. Il formato dell'operatore unario è

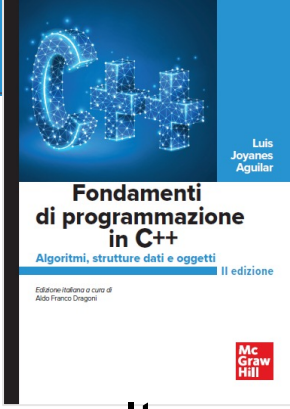
sizeof(*nome_variabile tipo_dato*)
sizeof *espressione*

Esempio

```
int main() {
   int i=10, j=11;
   cout << sizeof (char) << ', ';
   cout << sizeof (unsigned int) << ', ';
   cout << sizeof (float) << ', ';
   cout << sizeof (double) << ', ';
   cout << sizeof (i) << ', ';
   cout << sizeof (i + j) << endl;
}
```

produce a schermo:

```
1, 4, 4, 8, 4, 4
```

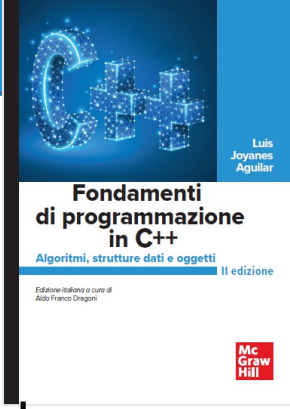


conversioni di tipo

In C/C++ si può convertire un valore di un tipo in un valore di un altro tipo. Tale azione si dice *conversione di tipo (casting)*

C/C++ realizza automaticamente parecchie conversioni (conversioni implicite), ma ha anche operatori di conversione che permettono al programmatore di impostarle esplicitamente (conversioni esplicite). Tipicamente, C/C++ converte implicitamente i tipi quando

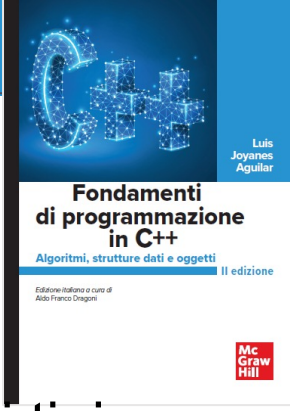
- si assegna un valore di un tipo a una variabile di un altro tipo aritmetico
- si combinano in espressioni tipi diversi
- si passano argomenti a funzioni.



conversioni di tipo

1. In un'espressione aritmetica, gli operatori binari richiedono operandi dello stesso tipo. Se questi non coincidono, il compilatore può compiere automaticamente conversioni di tipo implicite.
2. Nel caso di chiamata a una funzione, il tipo dell'argomento attuale deve corrispondere con il tipo dell'argomento formale, altrimenti il compilatore prova a convertire il tipo del parametro attuale a quello del parametro formale.

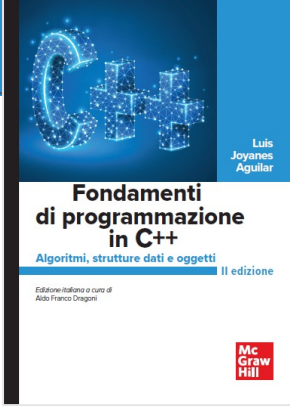
3 C/C++ ha operatori di *casting* di tipi (che permettono cioè di convertire i tipi esplicitamente), ma il compilatore può convertire automaticamente il tipo in certi casi ovvi.



conversioni implicite

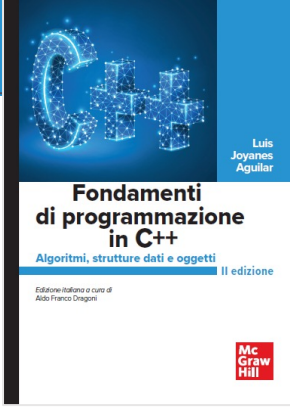
- Gli operandi di tipo a precisione più bassa si convertono nei tipi a precisione più alta (promozione di tipo).
  Ad esempio, in:
  pi = 3.141592 + 3;
  si assegna a pi la somma di un intero e di un reale, ovvero valori di tipi differenti, quindi pi potrebbe assumere valore 6 o 6.141592 (che sarebbe più logico) a seconda che venga compiuta la "somma fra interi" (troncando 3.141592 a 3) o la "somma fra reali" (convertendo 3 nel reale 3.0).

- Il compilatore sceglie preservando, se possibile, la precisione, quindi convertendo il numero intero in numero reale (ottenendo 6.141592).



conversioni aritmetiche

- assicurano che gli operandi di un operatore binario aritmetico o logico si convertano ad un tipo comune prima che si valuti l'espressione, e questo tipo comune sarà il tipo del risultato dell'espressione.
- il compilatore cerca di convertire senza perdere informazione, convertendo il tipo dell'operando a precisione più bassa in quello del tipo a precisione più alta.
- graduatoria di precisione:
- `long double,double,float,long,int,short,char,bool`
- I tipi più piccoli di `int` (`char, signed char, unsigned char, short e unsigned short`) sono promossi ad `int`. Il tipo bool si promuove ad int nel senso che false si promuove a 0 e true ad 1.



conversioni esplicite

Con il termine `casting` s'intende la conversione esplicita di tipi.
C++ offre nuove forme di `casting` rispetto al C.

Notazioni compatibili con il C

- *tipo_in_cui_convertire*`(`*espressione*`)//stile funzione`
- `(`*tipo_in_cui_convertire*`)` *espressione*`; // stile C`

Notazioni compatibili con lo standard ANSI/ISO C++

- **const_cast** *<tipo_in_cui_convertire> (espressione)*
- **dinamic_cast** *<tipo_in_cui_convertire> (espressione)*
- **reinterpret_cast** *<tipo_in_cui_convertire> (espressione)*
- **static_cast** *<tipo_in_cui_convertire> (espressione)*