Operatori

- Il C++ fornisce degli operatori per la manipolazione di variabili e costanti, che nel nostro linguaggio sono un insieme di parole chiave e simboli speciali.
- Gli operatori che vedremo sono:
  - Assegnamento (=)
  - Operatori aritmetici (+, -, *, /, %)
  - Operatori di assegnazione composti (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)
  - Incremento e decremento
  - Operatori relazionali (==, !=, >, <, >=, <=)
  - Operatori logici (!, &&, ||)
  - Operatore condizionale (?)
  - Operatori bit a bit (&, |, ^, ~, <<, >>)
  - Operatori espliciti di conversione di tipo (casting)
  - Operatore sizeof()

Assegnamento

- L'operatore di assegnamento serve per assegnare un valore ad una variabile `( a = 5;)`
- La parte alla sinistra dell'operatore = è nota come lvalue (left value) e la parte destra rvalue (right value)
- lvalue deve essere sempre una variabile mentre la parte destra può essere una costante, una variabile, il risultato di una operazione o una qualsiasi combinazione di essi
- L'operatore di assegnazione opera sempre e solo da destra verso sinistra
- `( a = b; )` assegna alla variabile **a** (lvalue) il valore della variabile **b** (rvalue) indipendentemente dal valore che era precedentemente memorizzato nella variabile **a**
- Possiamo pensare l'lvalue di **a** come l'indirizzo della zona di memoria riservata per memorizzare il valore di **a** mentre l'rvalue di **b** lo dobbiamo pensare come il valore memorizzato nella zona di memoria riservata per memorizzare il valore di **b**
- Notiamo inoltre che noi stiamo soltanto assegnando ad **a** il valore di **b** e che una modifica successiva di **b** non cambia il nuovo valore di **a**

Assegnamento

- Esempio di codice dove i commenti mostrano l'evoluzione del contenuto delle variabili:

```
int a, b;     // a:?  b:?
a = 10;       // a:10 b:?
b = 4;        // a:10 b:4
a = b;        // a:4 b:4
b = 7;        // a:4 b:7
```

- Una caratteristica dell'operatore di assegnamento presente in C++ è che un assegnamento si può usare come rvalue (o parte di un rvalue) in un altro assegnamento. Ad esempio:

```
a = 2 + (b = 5);
```

è equivalente a

```
b = 5;
a = 2 + b;
```

- Quindi, anche la seguente espressione è valida in C++:

```
a = b = c = 5;
```


Operatori aritmetici (e di assegnamento composti)

- I cinque operatori aritmetici previsti dal linguaggio sono:
  + somma - differenza * moltiplicazione / divisione % modulo (o resto)
- Gli operatori di somma, differenza, moltiplicazione e divisione denotano le usuali quattro operazioni numeriche.
- L'operatore di modulo fornisce il resto della divisione di due numeri interi. Ad esempio, se scriviamo `( a = 11 % 3; )`, viene assegnato alla variabile **a** il valore 2 in quanto 2 è proprio il resto che si ottiene dividendo 11 per 3.

Operatori di assegnamento composti:

- permettono di modificare il valore di una variabile con una sola operazione:

`valore += incremento;`	è equivalente a	`valore = valore + incremento;`
`a -= 5;`	è equivalente a	`a = a - 5;`
`a /= b;`	è equivalente a	`a = a / b;`
`prezzo *= numero + 1;`	è equivalente a	`prezzo = prezzo * (numero + 1);`

Incremento e decremento

- Gli operatori di incremento (++) e di decremento (--) aumentano o diminuiscono di 1 il valore di una variabile e sono equivalenti a += 1 e -= 1 rispettivamente. Quindi:

  `a++;  a += 1;  a = a + 1;` sono istruzioni equivalenti
- Questi operatori si possono usare sia come prefissi che come postfissi, ossia possono essere scritti prima della variabile (`++a`) o dopo di essa (`a++`).
- ATTENZIONE !
  - se si usa `++a` , il valore della variabile viene incrementato prima della valutazione dell'espressione e quindi l'espressione viene valutata usando il valore incrementato;
  - se si usa `a++` , il valore della variabile viene incrementato dopo la valutazione dell'espressione e quindi l'espressione viene valutata usando il valore non incrementato.
- Ecco alcuni esempi della differenza tra i due modi di usare l'operatore:

```
b = 3;
a = ++b;
// a è 4, b è 4
```

```
b = 3;
a = b++;
// a è 3, b è 4
```


Operatori Relazionali

- Per confrontare i valori di due espressioni si usano gli operatori relazionali.
- Il risultato di un operatore relazionale è di tipo bool e può assumere soltanto uno dei due valori booleani true o false, a seconda del risultato del confronto.
- Gli operatori relazionali in C++ sono i seguenti:

  == Uguale != Diverso > Maggiore < Minore >= Maggiore o uguale
  <= Minore o uguale
- Ecco alcuni esempi:

  `(7 == 5)` risultato `false`
  `(3 != 2)` risultato `true`
  `(5 < 5)` risultato `false`

  `(5 > 4)` risultato `true`
  `(6 >= 6)` risultato `true`

  Supponendo che `a=2`, `b=3` e `c=6`

  `(a == 5)` risultato `false`
  `(a*b >= c)` risultato `true` in quanto (2*3 >= 6)
  `(b+4 > a*c)` risultato `false` in quanto (3+4 > 2*6)
  `((b=2) == a)` risultato `true`

  ATTENZIONE ! L'operatore = (un solo uguale) è differente dal simbolo == (doppio uguale)

Operatori Relazionali

ATTENZIONE !

In molti compilatori precedenti la pubblicazione dello standard ANSI-C++, come pure in C, gli operatori relazionali

- non ritornano un valore bool (true o false)
- ma ritornano un valore int con
  - 0 che rappresenta "false" e
  - un valore diverso da 0 (in genere 1) che rappresenta "true"

Operatori Logici

- L'operatore ! è l'operatore logico di negazione NOT.
- Esso ha un unico operando (di tipo bool) posto alla sua destra e il suo risultato è l'opposto del valore dell'operando:
  - se l'operando è true esso ritorna false,
  - se l'operando è false esso ritorna true.

Ad esempio:

!(5 == 5) ritorna false in quanto l'espressione alla sua destra (5 == 5) è true.
!(6 <= 4) ritorna true in quanto (6 <= 4) è false.
!true ritorna false.
!false ritorna true.

- Gli operatori && e || sono gli operatori logici di congiunzione (AND) e disgiunzione (OR). Essi hanno due operandi (di tipo bool), uno alla sinistra ed uno alla destra

Ad esempio:

((5 == 5) && (3 > 6)) ritorna false (true && false).
((5 == 5) || (3 > 6)) ritorna true (true || false) .

Operatore Condizionale

- L'operatore condizionale (?) valuta una espressione booleana e ritorna un valore diverso a seconda che tale valore sia true oppure false.
- La sua forma è:

```
condizione ? risultato1 : risultato2
```

- se **condizione** è true
  - l'espressione ritorna **risultato1**,
  - altrimenti ritorna **risultato2**.

Ad esempio

7==5 ? 4 : 3	ritorna 3 perché 7 non è uguale a 5 .
7==5+2 ? 4 : 3	ritorna 4 perché 7 è uguale a 5+2 .
5>3 ? a : b	ritorna a, perché 5 è maggiore di 3 .
a>b ? a : b	ritorna il maggiore dei due, a o b .

Operatori bit a bit

- Gli operatori bit a bit operano in parallelo su tutti i bit degli operandi.
- Essi sono operatori di basso livello a cui corrispondono alcune operazioni assembler:

operatore	assembler	Descrizione
&	**AND**	AND bit a bit
|	**OR**	OR bit a bit.
^	**XOR**	OR esclusivo bit a bit.
~	**NOT**	NOT bit a bit (complemento a uno).
<<	**SHL**	Spostamento dei bit (shift) a sinistra
>>	**SHR**	Spostamento dei bit (shift) a destra

- Nella programmazione normale C++ essi non dovrebbero essere usati.

Operatori espliciti di conversione di tipo (casting)

- Gli operatori di conversione servono per convertire valori appartenenti ad un tipo in valori di un altro tipo.
- Vi sono diversi modi per fare questo in C++

Stile C

- Si fa precedere le espressioni che devono essere convertite dal nome del nuovo tipo racchiuso tra parentesi ():

```
int i;
float f = 3.14;
i = (int) f;
```

Converte il numero float 3.14 nel valore intero 3.
L'operatore di conversione è rappresentato da (int).

Stile C++

- far precedere l'espressione da convertire, racchiusa tra parentesi, dal nome del nuovo tipo

```
int i;
float f = 3.14;
i = int( f );
```

Usa la funzione "costruttore".

Operatore sizeof()

- L'operatore sizeof() ha un parametro che può essere sia il nome di un tipo che una espressione.
- Esso ritorna la memoria, in byte, necessaria a memorizzare un valore di tale tipo o il valore dell'espressione:

```
a = sizeof (char);
```

ritorna 1 in **a** perché un valore di tipo char occupa un byte.

```
a = sizeof (double);
```

ritorna 8 in **a** perché un valore di tipo double occupa 8 byte.

- Il valore ritornato da sizeof() è una costante.
- Tale valore è quindi determinato a tempo di compilazione (prima dell'esecuzione del programma).
- Viene utilizzato (soprattutto in C) per stabilire quanta memoria allocare.

Priorità degli operatori

- Quando scriviamo espressioni complesse con molti operatori e operandi possono sorgere dei dubbi sull'ordine in cui gli operatori vengono valutati. Ad esempio, con l'espressione:

```
a = 5 + 7 % 2;
```

può sorgere il dubbio tra le seguenti due interpretazioni:

```
a = 5 + (7 % 2); // con risultato 6, o
a = (5 + 7) % 2; // con risultato 0
```

- L'interpretazione corretta è la prima, quella con risultato 6.
- Vi è un ordine di priorità prestabilito tra tutti gli operatori (sia aritmetici che non aritmetici).
- La precedenza degli operatori in una espressione si può modificare o rendere evidente usando le parentesi (e) .

Priorità degli operatori

Priorità	Operatore	Descrizione	Associatività
1	`::`	scopo	Sinistra
2	`() [ ] -> . sizeof`		Sinistra
3	`++ --`	incremento/decremento	Destra
	`~`	Complemento a uno (bit a bit)	
	`!`	NOT	
	`& *`	Referenziazione e Dereferenziazione (puntatori)	
	`(tipo)`	Conversione di tipo	
	`+ -`	Operatori di segno unario	
4	`* / %`	Operatori aritmetici moltiplicativi	Sinistra
5	`+ -`	Operatori aritmetici additivi	Sinistra
6	`<< >>`	Spostamento dei bit	Sinistra
7	`< <= > >=`	Operatori relazionali	Sinistra
8	`== !=`	Operatori relazionali di uguaglianza	Sinistra
9	`& ^ |`	Operatori bit a bit	Sinistra
10	`&& ||`	Operatori logici	Sinistra
11	`?:`	Operatore condizionale	Destra
12	`= += -= *= /= %= >>= <<= &= ^= |=`	Assegnamento	Destra
13	`,`	Operatore virgola, separatore	Sinistra

Comunicazione da console

- La console è l'interfaccia base del calcolatore, essa è di solito composta da una tastiera (usata come unità di input standard) ed un video (usato come unità di output standard).
- Nalla libreria standard **iostream** del C++ le operazioni di input ed output di un programma vengono gestite da due flussi di dati (stream):
  - **cin** per l'input (normalmente assegnato alla tastiera)
  - **cout** per l'output (normalmente diretto al video)
- Sono definiti inoltre altri due flussi - **cerr** e **clog** - il cui scopo è quello di segnalare eventuali messaggi di errore. Tali flussi possono essere mandati anch'essi sul video oppure inviati in un file di log.
- Usando questi due flussi un programma può interagire con un utente mostrando messaggi sullo schermo e ricevendo l'input da parte dell'utente dalla tastiera.

Output (cout)

- Il flusso **cout** viene usato assieme all'operatore "sovraccaricato" << (una coppia di segni di "minore"):

```
cout << "Frase di output"; // stampa Frase di output sullo schermo
cout << 120;              // stampa il numero 120 sullo schermo
cout << x;                // stampa il valore di x sullo schermo
```

- L'operatore << è noto come operatore di inserimento in quanto esso inserisce il valore alla sua destra nel flusso di dati indicato alla sua sinistra.
- Nel primo esempio esso inserisce nel flusso di output standard **cout** la stringa costante Frase di output racchiusa tra doppi apici ("), questo perché essa è una stringa di caratteri.
- Quando vogliamo usare una stringa costante di caratteri dobbiamo racchiuderla tra doppi apici per distinguerla da un identificatore di variabile:

```
a cout << "Salve"; // stampa Salve sullo schermo
cout << Salve;     // stampa il valore della variabile Salve sullo schermo
```

Output (cout)

- L'operatore di inserzione (<<) può essere usato più di una volta nella stessa frase. Ad esempio:

```
cout << "Salve, " << "io sono " << "una frase C++";
```

L'utilità di poter usare più volte l'operatore di inserzione nella stessa frase si vede quando dobbiamo stampare una combinazione di costanti e variabili o anche semplicemente più di una variabile:

```
cout << "Salve, io ho " << eta << " anni e il mio CAP e' " << cap;
```

- Se assumiamo che la variabile **eta** abbia il valore 24 e che la variabile **cap** abbia valore 65064 l'output che si ottiene è:

Salve, io ho 24 anni e il mio CAP e' 65064

- Attenzione: **cout** non va a capo dopo aver stampato a meno che non glielo si dica esplicitamente. Pertanto, con le due seguenti frasi:

```
cout << "Questa e' una frase.";
cout << "Questa e' un'altra frase.";
```

- otteniamo:

Questa e' una frase.Questa e' un'altra frase.

anche se sono state scritte con due distinte chiamate a **cout**

Output (cout)

- Per andare a capo bisogna inserire esplicitamente nel flusso un carattere speciale di nuove-linea che, in C++, si indica con **\n** :

```
cout << "Prima frase.\n ";
cout << "Seconda frase.\nTerza frase.";
```

stampa:

Prima frase.
Seconda frase.
Terza frase.

- Per andare a capo possiamo anche usare il *manipolatore* **endl** . Ad esempio:

```
cout << "Prima frase." << endl;
cout << "Seconda frase." << endl;
```

stampa:

Prima frase.
Seconda frase.

Input (cin)

- In C++ l'input standard si effettua applicando l'operatore di estrazione (>>) al flusso **cin** . Tale operatore deve essere seguito dalla variabile in cui deve essere memorizzato il valore da leggere. Ad esempio:

```
int eta;
cin >> eta;
```

  dichiara la variabile **eta** di tipo **int** e quindi aspetta un input da **cin** (tastiera) per poter memorizzare un valore intero in tale variabile.

- **Attenzione:** **cin** elabora l'input ricevuto da tastiera soltanto dopo che è stato premuto il tasto di invio ENTER.
- Quindi, anche se viene richiesto di leggere un singolo carattere, **cin** non elabora l'input fino a quando non viene premuto anche il tasto ENTER.

- Quando si usa l'estrattore (>>) su **cin** bisogna tener presente il tipo della variabile che si usa per memorizzare il valore letto.
- Se viene richiesto un intero deve essere introdotto un intero, se viene richiesto un carattere deve essere introdotto un carattere e se viene richiesta una stringa deve essere introdotta una stringa.

Input (cin)

```
// esempio di i/o

#include <iostream.h>

int main ()
 {
  int i;
  cout << "Dammi un intero: ";
  cin >> i;
  cout << "Il valore che mi hai dato e' " << i;
  cout << " e il suo doppio e' " << i*2 << ".\n";
  return 0;
 }
```

```
Dammi un intero: 702
Il valore che mi hai dato e' 702 e il suo doppio e' 1404.
```

Input (cin)

- In C++ l'input standard si effettua applicando l'operatore di estrazione (>>) al flusso **cin** . Tale operatore deve essere seguito dalla variabile in cui deve essere memorizzato il valore da leggere. Ad esempio:

```
int eta;
cin >> eta;
```

dichiara la variabile **eta** di tipo **int** e quindi aspetta un input da **cin** (tastiera) per poter memorizzare un valore intero in tale variabile.

- **Attenzione:** **cin** elabora l'input ricevuto da tastiera soltanto dopo che è stato premuto il tasto di invio ENTER.
- Quindi, anche se viene richiesto di leggere un singolo carattere, **cin** non elabora l'input fino a quando non viene premuto anche il tasto ENTER.

- Quando si usa l'estrattore (>>) su **cin** bisogna tener presente il tipo della variabile che si usa per memorizzare il valore letto.
- Se viene richiesto un intero deve essere introdotto un intero, se viene richiesto un carattere deve essere introdotto un carattere e se viene richiesta una stringa deve essere introdotta una stringa.

Input (cin)

- Si può anche usare **cin** per richiedere più di un dato alla volta:

```
cin >> a >> b;
```

è equivalente a:

```
cin >> a;
cin >> b;
```

- In entrambi i casi l'utente deve fornire due valori dei tipi appropriati, uno per la variabile a ed uno per la variabile b.
- Tali valori possono essere separati da uno o più caratteri spazio o tabulazione o nuova-linea.