Primi programmi in C++ - Parte III
Conversioni di tipo

Corso AMAT C++

Liceo Scientifico Statale "A. Volta"

A.S. 2018/19

Conversione implicita di tipo (type coercion)

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double somma;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7
8    somma = a+b;
9    return 0;
```

Conversione implicita di tipo (type coercion)

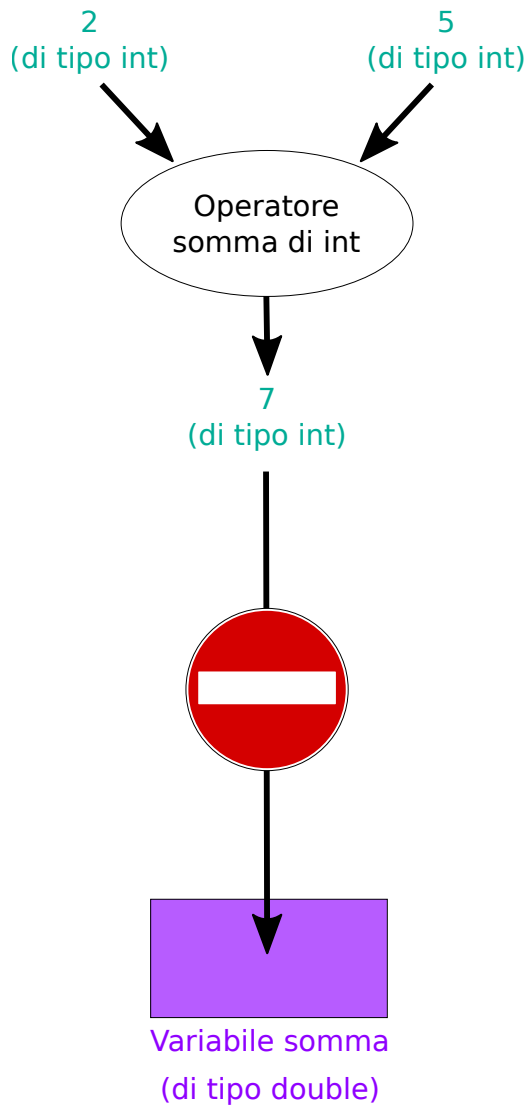
```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double somma;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7
8    somma = a+b;
9    return 0;
```

Quando calcola l'espressione a destra dell'uguale, il computer somma due valori `int` mediante l'operatore *somma di numeri* int, che fornisce un risultato `int`: tale risultato potrà essere immagazzinato in una variabile `double` solo mediante una *conversione di tipo* `int`$\rightarrow$`double` (che è, in questo caso, *implicita*, in quanto viene introdotta automaticamente dal compilatore senza che sia stata espressamente invocata da chi ha scritto il programma).

Conversione implicita di tipo (type coercion)

Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `int`) ...

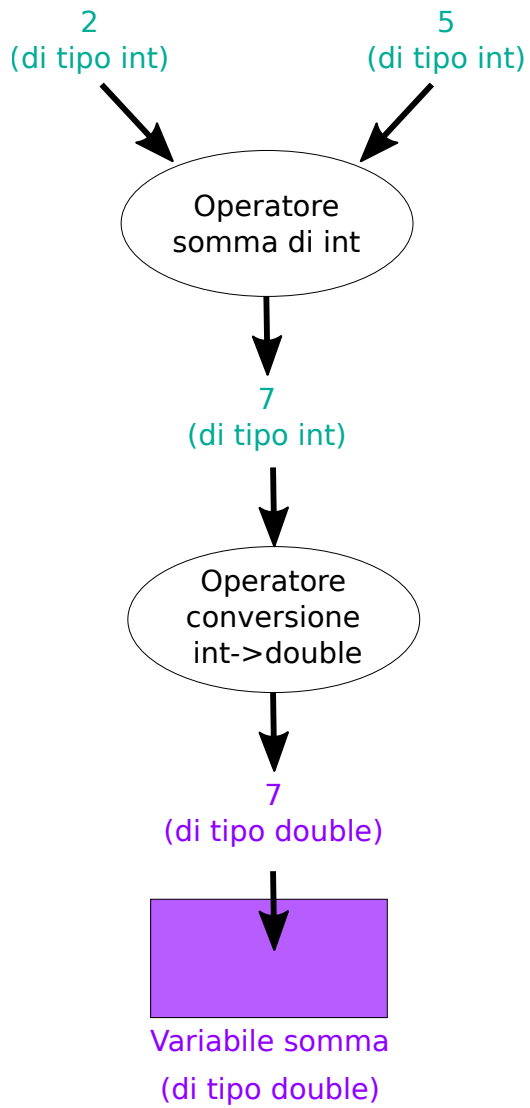
Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `int`) ...

- Il risultato della somma (nel nostro caso, 7) è di tipo `int` e non può essere assegnato ad una variabile `double`...

Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `int`) ...

- Il risultato della somma (nel nostro caso, 7) è di tipo `int` e non può essere assegnato ad una variabile `double`...
- Allora il compilatore introduce *motu proprio* un "passo aggiuntivo" di conversione `int`→`double`.

Conversione implicita di tipo (type coercion)

```
1  #include <iostream>
2  int main(){
3     double a; double b;
4     int somma;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     somma = a+b;
9     return 0;
10 }
```

Conversione implicita di tipo (type coercion)

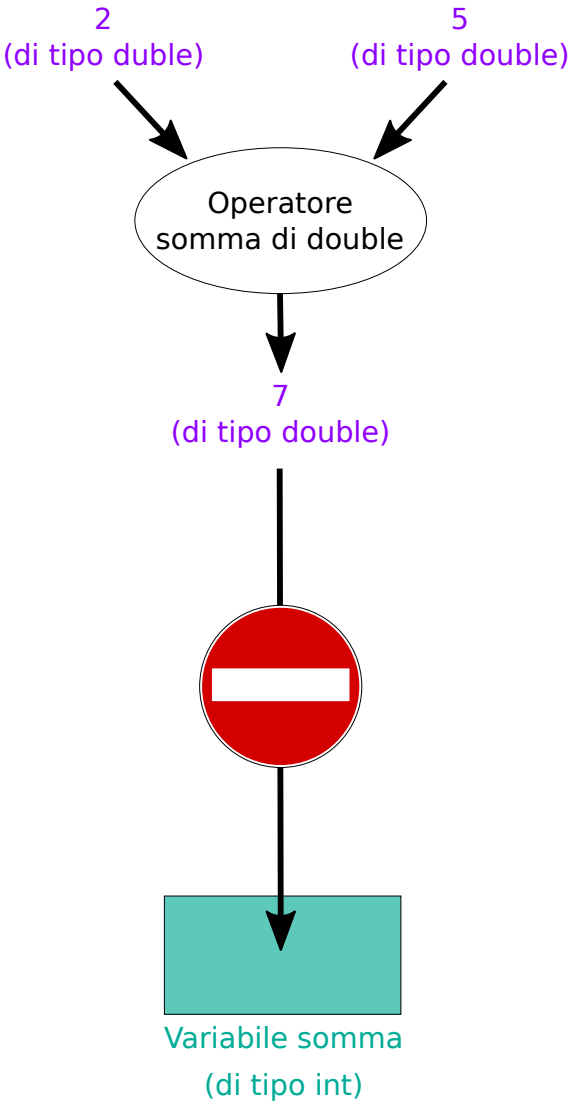
```
1  #include <iostream>
2  int main(){
3     double a; double b;
4     int somma;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     somma = a+b;
9     return 0;
10 }
```

Qui la situazione è speculare: per valutare l'espressione, il computer deve sommare due numeri `double`; lo farà mediante l'operatore *somma di numeri `double`*, che restituirà un risultato di di tipo `double`: ma un valore `double` può essere assegnato ad una variabile `int` solo mediante una *conversione di tipo* `double`→`int`, che comporta un troncamento del risultato.

Conversione implicita di tipo (type coercion)

Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `double`) e 5 come valore di `b` (tipo `double`)...

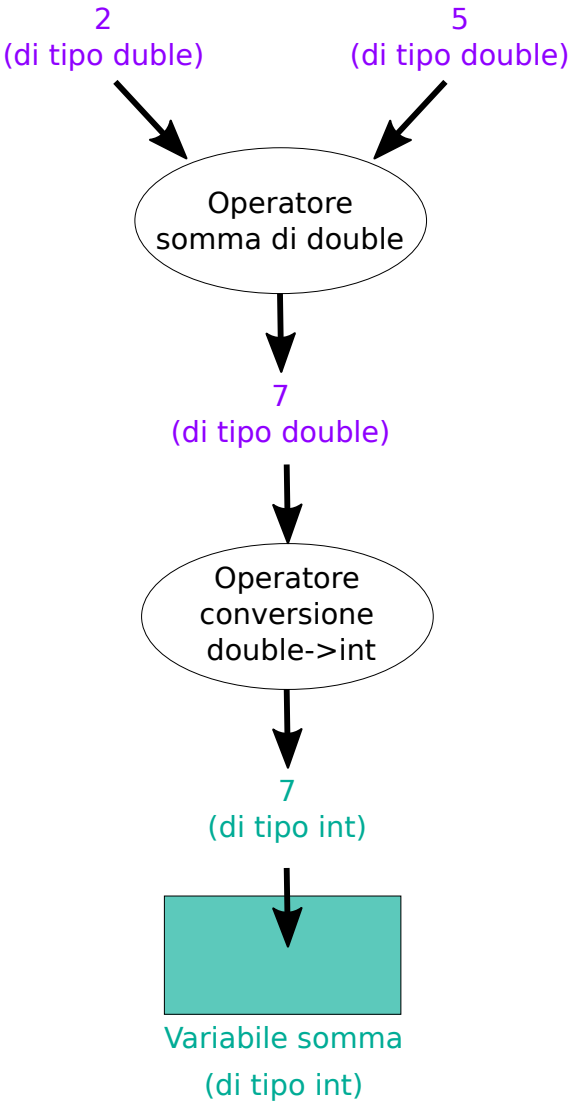
Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `double`) e 5 come valore di `b` (tipo `double`)...

- Il risultato della somma (nel nostro caso, 7) è di tipo `double` e non può essere assegnato ad una variabile `int`...

Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `double`) e 5 come valore di `b` (tipo `double`)...

- Il risultato della somma (nel nostro caso, 7) è di tipo `double` e non può essere assegnato ad una variabile `int`...
- Allora il compilatore introduce *motu proprio* un "passo aggiuntivo" di conversione `double`→`int`.

Conversione implicita di tipo (type coercion)

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     double somma;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     somma = a+b;
9     return 0;
10 }
```

Conversione implicita di tipo (type coercion)

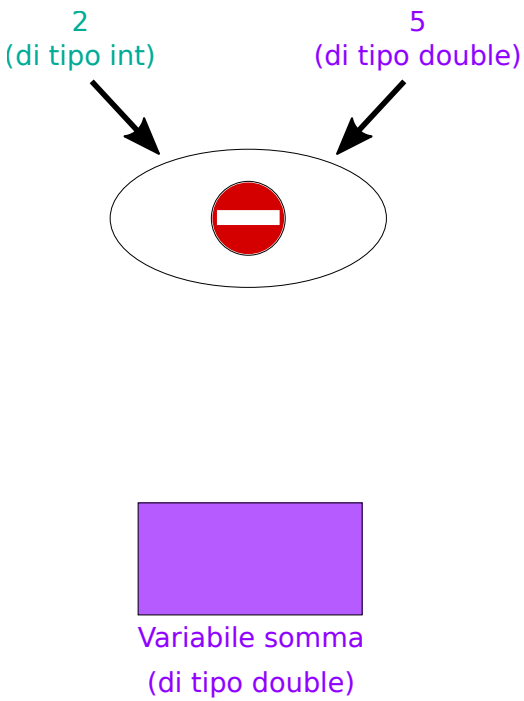
```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     double somma;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     somma = a+b;
9     return 0;
10 }
```

In questo caso, il "problema di tipo" nasce prima, al momento di calcolare la somma...

Conversione implicita di tipo (type coercion)

Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

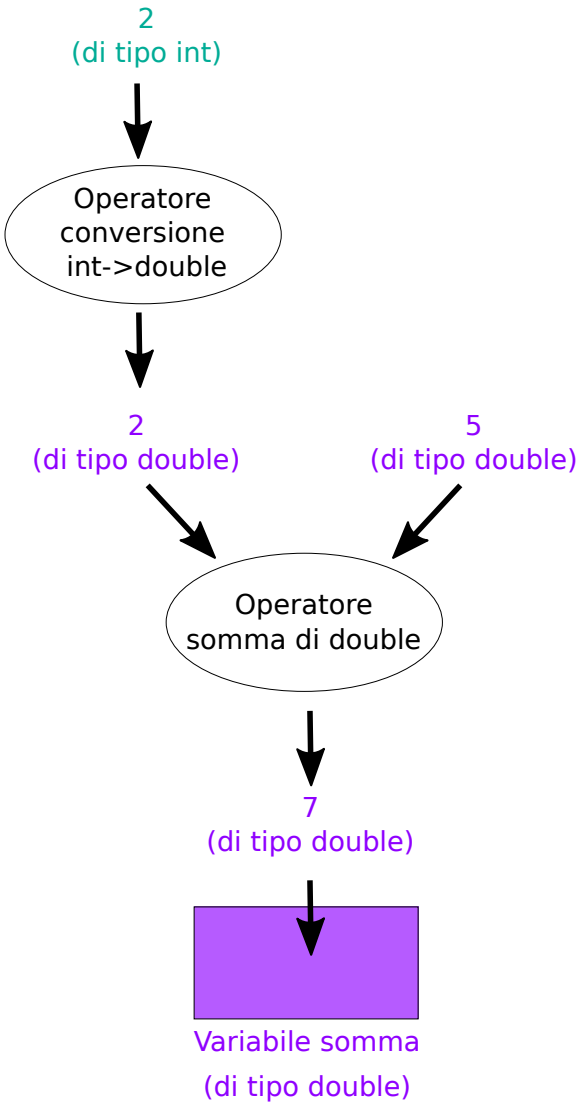
Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

- Non esiste un operatore somma definito tra un numero `int` e un numero `double`.

Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

- Non esiste un operatore somma definito tra un numero `int` e un numero `double`.
- Allora il compilatore converte automaticamente l'addendo `int` verso il tipo `double`, così da poter usare l'operatore `somma tra numeri double`.

Conversione implicita di tipo (type coercion)

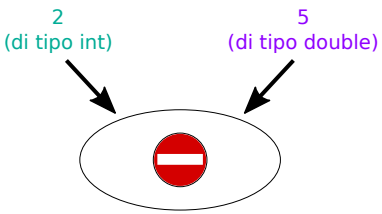
```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     int somma;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     somma = a+b;
9     return 0;
10 }
```

In questo caso, il compilatore introdurrà una doppia conversione di tipo, una prima, e una dopo il calcolo della somma:

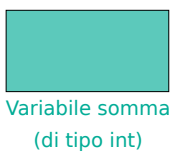
Conversione implicita di tipo (type coercion)

Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

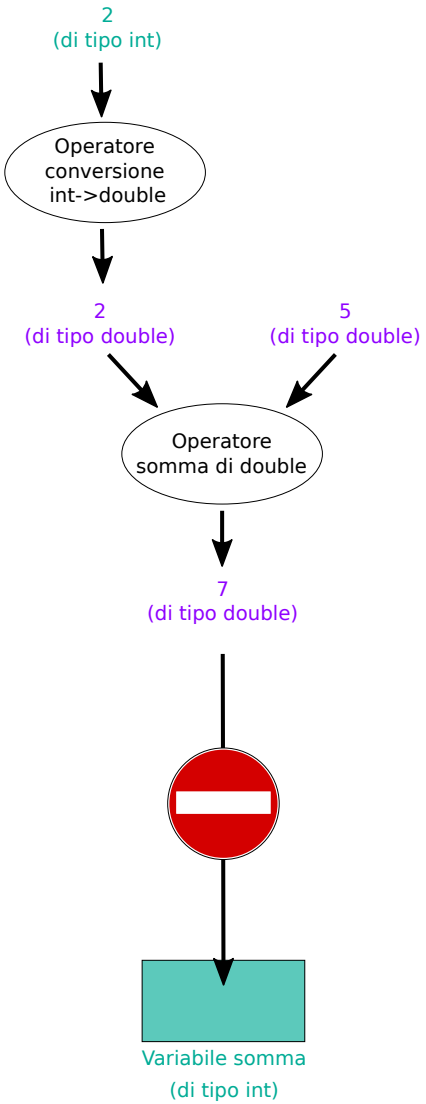
Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...



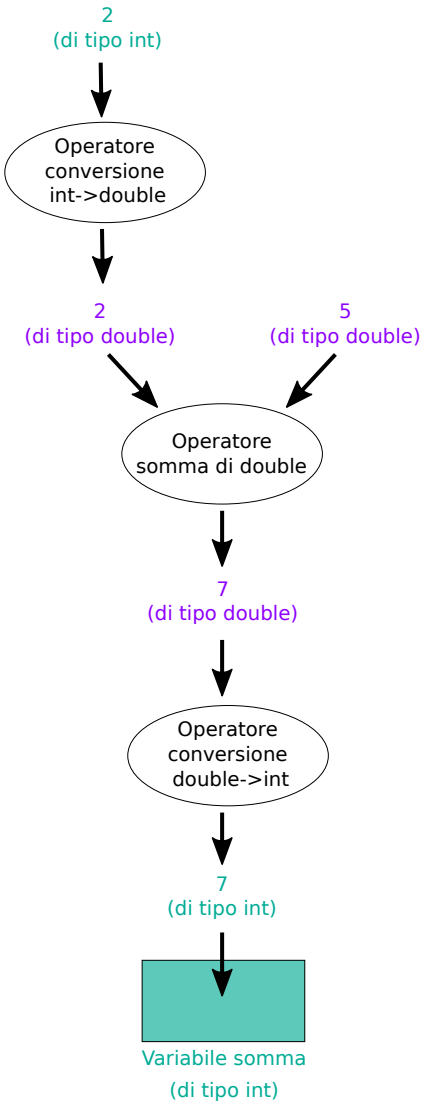
Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

- non esistendo una somma `int+double`, l'addendo di tipo `int` viene convertito in `double`, e diventa così possibile usare la `somma di double`

Conversione implicita di tipo (type coercion)



Immaginiamo, per esempio, che l'utente immetta 2 come valore di `a` (tipo `int`) e 5 come valore di `b` (tipo `double`)...

- non esistendo una somma `int`+`double`, l'addendo di tipo `int` viene convertito in `double`, e diventa così possibile usare la `somma di double`
- il valore `double` ottenuto sommando non può essere assegnato direttamente alla variabile `somma`, che è di tipo `int`: si rende indispensabile una conversione `double`→`int`

La divisione

Con il simbolo / si indicano, in C++, diversi operatori, il cui comportamento è **molto differente**. In particolare,

La divisione

Con il simbolo / si indicano, in C++, diversi operatori, il cui comportamento è **molto differente**. In particolare,

- si indica con / l'operatore *divisione tra numeri* `int`, che, presi due numeri interi di tipo `int`, restituisce il *quoziente della divisione euclidea* (intero, di tipo `int`);

La divisione

Con il simbolo / si indicano, in C++, diversi operatori, il cui comportamento è **molto differente**. In particolare,

- si indica con / l'operatore *divisione tra numeri* `int`, che, presi due numeri interi di tipo `int`, restituisce il *quoziente della divisione euclidea* (intero, di tipo `int`);
- si indica con / l'operatore *divisione tra numeri* `double`, che, presi due numeri reali di tipo `double`, restituisce il loro *rapporto* (reale, di tipo `double`).

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    int quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
9 }
```

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    int quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
9 }
```

Visto il tipo degli operandi, il compilatore interpreta / come l'operatore `divisione tra numeri int`.

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    int quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
9 }
```

Visto il tipo degli operandi, il compilatore interpreta / come l'operatore `divisione tra numeri int`. La variabile `quoziente`, al termine dell'esecuzione, vale 1 (di tipo `int`).

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    double a; double b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
```

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    double a; double b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
```

Visto il tipo degli operandi, il compilatore interpreta / come l'operatore `divisione tra numeri double`.

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1 #include <iostream>
2 int main(){
3    double a; double b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = a/b;
8    return 0;
```

Visto il tipo degli operandi, il compilatore interpreta / come l'operatore `divisione tra numeri double`. La variabile `quoziente`, al termine dell'esecuzione, vale 1.8 (di tipo `double`).

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; int b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; int b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Qui l'operatore / agisce sue due operandi di tipo `int`, e quindi viene interpretato dal compilatore come `divisione di numeri int`.

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; int b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Qui l'operatore / agisce sue due operandi di tipo `int`, e quindi viene interpretato dal compilatore come `divisione di numeri int`. Il risultato della divisione è il numero `int` 1, che viene convertito a `double` ed assegnato alla variabile `quoziente`, che quindi, al termine dell'esecuzione, vale 1 (di tipo `double`).

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Non esistendo un operatore che divida un `int` per un `double`, il compilatore premette alla divisione una conversione di tipo `int`→`double` per la variabile `a`, e interpreta poi l'operatore `/` come `divisione tra double`.

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     double quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Non esistendo un operatore che divida un `int` per un `double`, il compilatore premette alla divisione una conversione di tipo `int`→`double` per la variabile `a`, e interpreta poi l'operatore `/` come `divisione tra double`. Il risultato (`double`) della divisione viene infine assegnato alla variabile `quoziente`, che al termine dell'esecuzione vale quindi 1.8.

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     int quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Conversione implicita di tipo (type coercion): divisione

Si immagini che l'utente immetta prima 9 e poi 5.

```
1  #include <iostream>
2  int main(){
3     int a; double b;
4     int quoziente;
5     std::cout << "Inserisci a: "; std::cin >> a;
6     std::cout << "Inserisci b: "; std::cin >> b;
7
8     quoziente = a/b;
9     return 0;
10 }
```

Questa volta hanno luogo due conversioni implicite di tipo: `a` viene convertito `int`→`double`; viene quindi eseguita la `divisione tra numeri double`, e il risultato 1.8 viene sottoposto ad una conversione `double`→`int` per poter essere assegnato alla variabile `quoziente` (così, 1.8 viene troncato e il valore di `quoziente` al termine dell'esecuzione è 1).

Conversione esplicita di tipo (type casting)

- Un *operatore di conversione* `tipoA`→`tipoB` è un operatore unario che accetta un valore di tipo `tipoA` e restituisce un valore corrispondente di tipo `tipoB`.

Conversione esplicita di tipo (type casting)

- Un *operatore di conversione* `tipoA`→`tipoB` è un operatore unario che accetta un valore di tipo `tipoA` e restituisce un valore corrispondente di tipo `tipoB`.
- Nelle scorse diapositive, abbiamo visto in azione operatori di conversione `int`→`double` e `double`→`int` introdotti automaticamente dal compilatore ove necessario.

Conversione esplicita di tipo (type casting)

- Un *operatore di conversione* `tipoA`→`tipoB` è un operatore unario che accetta un valore di tipo `tipoA` e restituisce un valore corrispondente di tipo `tipoB`.
- Nelle scorse diapositive, abbiamo visto in azione operatori di conversione `int`→`double` e `double`→`int` introdotti automaticamente dal compilatore ove necessario.
- Gli operatori di conversione possono essere però anche utilizzati nelle espressioni direttamente dal programmatore! Si parla, in questo caso, di *conversione esplicita di tipo*, o *type casting*.

Conversione esplicita di tipo (type casting)

Immaginiamo di avere due variabili intere (tipo `int`), e di volerne calcolare il quoziente *come numeri reali*.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente =
8    return 0;
9 }
```

Conversione esplicita di tipo (type casting)

Immaginiamo di avere due variabili intere (tipo `int`), e di volerne calcolare il quoziente *come numeri reali*. Possiamo ricorrere ad una *conversione esplicita di tipo*.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente =
8    return 0;
9 }
```

Conversione esplicita di tipo (type casting)

Immaginiamo di avere due variabili intere (tipo `int`), e di volerne calcolare il quoziente *come numeri reali*. Possiamo ricorrere ad una *conversione esplicita di tipo*.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = double(a) / double(b);
8    return 0;
9 }
```

Conversione esplicita di tipo (type casting)

Immaginiamo di avere due variabili intere (tipo `int`), e di volerne calcolare il quoziente *come numeri reali*. Possiamo ricorrere ad una *conversione esplicita di tipo*.

```
1 #include <iostream>
2 int main(){
3    int a; int b;
4    double quoziente;
5    std::cout << "Inserisci a: "; std::cin >> a;
6    std::cout << "Inserisci b: "; std::cin >> b;
7    quoziente = double(a) / double(b);
8    return 0;
9 }
```

L'operatore `double(·)` è un operatore unario, che accetta in ingresso un numero di qualunque tipo (`int`,`float`,...) e restituisce il numero `double` che gli corrisponde.