Conversioni implicite di tipo

Le conversioni implicite del C++ si possono suddividere in 4 categorie:

1. Le corrispondenze "esatte"
2. Promozioni
3. Conversioni standard
4. Conversioni implicite definite dall'utente

   Le categorie sarebbero 5, ma per semplicità *NON* prendiamo in considerazione le conversioni che si applicano nel caso della sintassi "...", ereditata dal C, che consente di avere un numero arbitrario di parametri per una funzione).

Corrispondenze "esatte"

Le corrispondenze esatte corrispondono ad alcune conversioni implicite di tipo che sono garantite preservare il valore dell'argomento.

> Vi sono comunque conversioni che, pur preservando il valore, non fanno parte delle corrispondenze esatte.

Le corrispondenze esatte si possono suddividere nelle seguenti sottoclassi:

1a. Identità (match perfetti)

In realtà, questa *NON* è una conversione, perché tipo di partenza e tipo di destinazione coincidono; è comunque comodo includerla come caso speciale nella classificazione, per poter ragionare in modo più semplice durante la risoluzione dell'overloading.

Esempio: (si assumono le dichiarazioni `int i; const int& r = i;`)

tipo parametro	argomento
int	5
int&	i
int*	&i
const int&	r
double	5.2

1b. Trasformazioni di lvalue

Esempio: (si assumono le dichiarazioni `int a[10];` e `void foo();`)

tipo parametro	argomento
int	i (da lvalue a rvalue)
int*	a (decay array-puntatore)
void()()	foo (decay funzione-puntatore)

1c. Conversioni di qualificazione

Viene aggiunto il qualificatore `const`. Esempio:

tipo parametro	argomento
const int&	i
const int*	&i

Torna all'indice

Le promozioni

Le promozioni corrispondono ad alcune conversioni implicite di tipo che, come le conversioni esatte, sono garantite preservare il valore dell'argomento.

Ogni implementazione del linguaggio C++ è specifica per una particolare architettura del processore (per esempio, con "parole" di 16, 32 o 64 bit). Tradizionalmente, i tipi `int` e `unsigned int` vengono forniti della dimensione adeguata per ottenere la massima efficienza su quella particolare architettura. Al contrario, i tipi interi più piccoli di `int` non sono direttamente rappresentabili all'interno del processore (per esempio, il processore sa effettuare la somma di due registri o farne il confronto, ma tipicamente non è dotato di istruzioni che effettuano la somma o effettuano il confronto tra porzioni di registri).

Quindi, ogni volta che si vuole operare su un valore di un tipo di dato più piccolo di `int`, questo deve essere "promosso" al tipo `int` (o `unsigned int`) per potere effettuare l'operazione.

2a. Promozioni intere

- dai tipi interi piccoli (char/short, signed o unsigned) al tipo int (signed o unsigned);
- da bool a int è promozione (caso speciale).

2b. Promozioni floating point

- da float a double

2c. Promozione delle costanti di enumerazioni del C++03

Al più piccolo tipo intero (almeno int) sufficientemente grande per contenerle.

Torna all'indice

Conversioni standard

Le conversioni standard sono tutte le altre conversioni implicite che non coinvolgono conversioni definite dall'utente.

> Attenzione: da int a long non è promozione, da char a double è conversione standard.

Tra le conversioni standard da tenere in considerazione vi sono le conversioni tra riferimenti e tra puntatori. In particolare:

- la costante intera `0` e il valore `nullptr` (di `std::nullptr_t`) sono le costanti puntatore nullo; esse possono essere convertite implicitamente nel tipo `T*` (per qualunque `T`); la costante intera `0` può essere convertita in `nullptr`;
- ogni puntatore `T*` può essere convertito nel tipo `void*`, che corrisponde ad un puntatore ad un tipo ignoto (si noti che non esiste una conversione implicita che vada in senso inverso, da `void*` a `T*`);
- se un classe `D` è derivata (direttamente o indirettamente) dalla classe base `B`, allora ogni puntatore `D*` può essere convertito implicitamente in `B*`; si parla di "up-cast" (*conversione verso l'alto*), perché tradizionalmente nei diagrammi che rappresentano le relazioni tra i tipi di dato, le classi base, che sono più astratte e quindi "leggere", vengono rappresentate sopra le classi derivate, che sono più concrete e quindi "pesanti". Una analoga conversione sui riferimenti consente di trasformare un `D&` in un `B&`; (anche in questo caso, si noti che non esiste una conversione implicita per il "down-cast", che trasformerebbe un `B*` in un `D*` o un `B&` in un `D&`).

Torna all'indice

Conversioni implicite definite dall'utente

1. Uso (implicito) di costruttori che possono essere invocati con un solo argomento (di tipo diverso) e non sono marcati `explicit` Esempio:

```
struct Razionale {
    Razionale(int num, int den = 1); // conv. da int a Razionale
    // ...
};
```

2. Uso di operatori di conversione da tipo utente verso altro tipo Esempio:

```
struct Razionale {
    operator double() const; // conversione da Razionale a double
    // ...
};
```

Torna all'indice