11.5 Le conversioni di tipo

I valori di una variabile possono essere convertiti da un tipo a un altro. Questa conversione può avvenire implicitamente o può essere comandata esplicitamente. In ogni caso il programmatore deve sempre tener conto degli eventuali effetti delle conversioni implicite di tipo; quelle ammesse sono le seguenti:

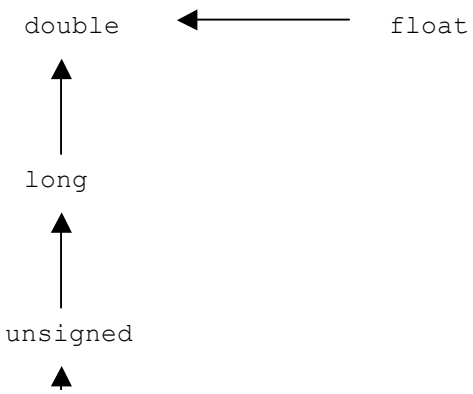
1) 1) il tipo `char` è implicitamente convertibile in `int`, `short int` e `long int`. La conversione di un carattere in un valore più lungo può riguardare o meno l'estensione del segno, a seconda del tipo di macchina;
2) 2) il tipo `int` è implicitamente convertibile in `char`, `short int`, `long int`. La conversione a un intero più lungo produce l'estensione del segno; la conversione a un intero più corto provoca il troncamento dei bit più significativi. Il tipo `int` è implicitamente convertibile anche in `float`, `double` e `unsigned int`. Per quest'ultimo caso la sequenza dei bit dell'`int` è interpretata come `unsigned`;
3) 3) i tipi `short int` e `long int` possono essere convertiti implicitamente come `int` (punto 2);
4) 4) il tipo `float` può essere convertito in `double` senza alcun problema. Il tipo `float` può essere anche convertito in `int`, `short int` e `long int`. Il risultato non è definito quando il valore `float` da convertire è troppo grande, perché cambia da macchina a macchina;
5) 5) il tipo `double` può essere convertito in `float`, potendo provocare troncamento nell'arrotondamento. Inoltre, per il `double` vale quanto detto per il `float` a proposito di `int`, `short int` e `long int`.

> ✓ **NOTA**
> Le conversioni implicite di tipo vengono effettuate per rendere conformi, se possibile, i tipi di due operandi in un'espressione e i tipi dei parametri attuali e formali nel passaggio dei parametri di una funzione. In C sono possibili molte conversioni implicite di tipo, ma ci limiteremo a considerare solo quelle che il programmatore può ragionevolmente ammettere. Per tutte le altre conversioni, non è saggio affidarsi al compilatore, soprattutto per motivi di leggibilità e portabilità delle applicazioni, e, pertanto, si raccomanda di usare la conversione esplicita di tipo.

In C l'esecuzione degli operatori aritmetici effettua la conversione implicita degli operandi se questi non sono omogenei. Lo schema di conversione usato è comunemente detto *conversione aritmetica normale*, ed è descritto dalle regole seguenti.

- • Convertire gli operandi `char` e `short int` in `int`; convertire gli operandi `float` in `double`.
- • Se uno degli operandi è di tipo `float`, allora l'altro operando, se non è già `double`, è convertito in `double`, e il risultato è `double`.
- • Se uno degli operandi è di tipo `long`, allora l'altro operando, se non è già `long`, viene convertito in `long`, e il risultato è `long`.
- • Se un operando è di tipo `unsigned`, allora l'altro operando, se non è già `unsigned`, è convertito in `unsigned`, e il risultato è `unsigned`.
- • Se non siamo nella situazione descritta dai casi 2 e 4, allora entrambi gli operandi debbono essere di tipo `int` e il risultato è `int`.

In forma più sintetica, queste regole di conversione possono essere illustrate dal diagramma seguente.



Si osservi dal diagramma come le conversioni orizzontali vengano sempre effettuate, mentre quelle verticali avvengano solo se necessario. Nel frammento di codice

```
short i;
...
i = i + 4;
```

si ha che la variabile `i` di tipo `short` è convertita in `int` e sommata alla costante intera 4. Il risultato della somma è convertito in `short`, prima dell'assegnazione. In quest'altro esempio:

```
double dd;
int    i;
...
i = i + dd;
```

la variabile intera `i` è convertita in `double`, sommata alla variabile `double dd` e il risultato della somma, prima dell'assegnazione, è convertito in `int`, con eventuale perdita di precisione.
Un'espressione di un certo tipo può essere esplicitamente convertita in un altro tipo per mezzo dell'operazione detta di *cast*. Abbiamo già incontrato il cast nel Capitolo 9, a proposito delle funzioni di allocazione e deallocazione dinamica della memoria `malloc`, `calloc` e `free`, e dell'indirizzamento assoluto della memoria. La sintassi generale del cast è:

```
(nome_tipo) espressione
```

Il valore di *espressione* è trasformato in un valore il cui tipo è *nome_tipo*. Esempi di *nome_tipo* sono:

```
char
char[8]
char *
char ()
int
void
```

In pratica *nome_tipo* è lo stesso che si usa nella dichiarazione delle variabili. Si consideri il semplice esempio:

```
char *pc;
int   a;
...
a = 0177777;
pc = (char *)a;
```

Alla variabile `a` di tipo `int` viene assegnata la costante ottale di tipo `int 0177777`. Successivamente questo valore contenuto in `a` è esplicitamente convertito nel tipo puntatore a `char` e assegnato alla variabile di tipo `char *`, `pc`. Si faccia bene attenzione: il tipo della variabile `a`, e così il suo contenuto, rimane `int` anche dopo il cast. Semplicemente il compilatore valuta l'espressione `a`, e il risultato di tale espressione è temporaneamente convertito in `char *`, e così convertito è assegnato alla variabile `pc`.
I cast sono principalmente usati per:

1. 1. convertire puntatori di tipo diverso ;
2. 2. eseguire calcoli con la precisione desiderata senza dover introdurre una variabile temporanea.

Finora abbiamo sempre usato il cast per operare conversioni tra puntatori che puntano a tipi diversi, ma può essere utile usarlo anche nelle espressioni aritmetiche:

```
int i;
...
```

```
i = ((long)i * 10) /9;
```

Trasformare la variabile `int i` in `long`, prima della moltiplicazione, garantisce che le operazioni di moltiplicazione e divisione vengano eseguite con una precisione maggiore. Nell'esempio:

```
int i, k;
...
k = (i+7)*3 + (double)(k+2*i);
```

la moltiplicazione tra `(i+7)*3` è una moltiplicazione tra `int` ed è eseguita per prima, mentre il risultato dell'espressione `(k+2*i)` è convertito in `double`, e la somma tra le due espressioni `(i+7)*3 e (k+2*i)` è di tipo `double`. Il risultato della somma è poi implicitamente convertito in `int`, con eventuale perdita di precisione.