Conversions implicites de type

Octobre 2015

Conversions implicites de type

Les types intégral et virgule flottante peuvent être librement mélangés dans les affectations et les expressions. Les valeurs sont converties chaque fois qu'elles peuvent l'être, de telle sorte qu'aucune information ne soit perdue. Des conversions entraînant une perte d'informations sont malheureusement aussi réalisées implicitement. Cette section décrit les règles et les problèmes de conversion avec leurs résolutions.

Promotions

On fait couramment référence aux conversions implicites qui préservent les valeurs sous le nom de promotions. Avant toute exécution d'une opération arithmétique, on utilise une promotion intégrale pour créer des int à partir de types entiers plus courts. Notez que ces promotions ne vont pas promouvoir vers un type long. Cela reflète l'objectif original de ces promotions en langage C : donner aux opérandes une taille « naturelle » pour les opérations arithmétiques. Les promotions intégrales sont les suivantes :

- Un char, signed char, unsigned char, short int ou unsigned short int est converti en un int si ce type peut représenter toutes les valeurs du type source. Dans le cas contraire, il est converti en un signed int.
- Un wchar_t ou un type énumération est converti vers le premier des types suivants qui peut représenter toutes les valeurs de son type sous-jacent : int, unsigned int, long ou unsigned long.
- Un type bool est converti vers un int, false devient 0 et true devient 1.

Les promotions font partie des conversions arithmétiques habituelles. ===À savoir== Un type wchar_txe "wchar_t" permet de recevoir des caractères appartenant à un jeu plus étendu, tel que Unicode. La taille de ce type est définie par l'implémentation, et elle est suffisamment grande pour recevoir le jeu de caractères le plus large supporté par les paramètres locaux (langue et pays) et l'implémentation. Ce nom bizarre est un héritage du langage C. En C, wchar_t est un typedef. Le suffixe _t a été rajouté pour le distinguer des typedef standard. Notez que les types de caractères sont des types intégral. Les opérations arithmétiques et logiques peuvent donc être appliquées.

Conversions

Les possibilités de conversion d'un type fondamental vers un autre sont extrêmement nombreuses. Par exemple :

```
void f(double d)
{
  char c = d;
// attention : conversion virgule flottante double-précision vers char
}
```

En écrivant votre code, vous devrez toujours éviter soigneusement un comportement indéfini et des conversions qui pourraient faire perdre des informations. Un compilateur peut émettre un avertissement

lorsqu'il détecte des conversions douteuses. Un grand nombre de ces derniers proposent ce service.

Conversions vers les types booléen, caractère et entier

Un entier peut être converti vers un autre type d'entiers. On peut convertir une valeur énumération vers un type d'entiers. Si le type destinataire est non signé, la valeur obtenue contiendra autant de bits de la source que ne pourra en recevoir la destination (les bits de poids fort sont abandonnés si nécessaire). Plus précisément, le résultat est le plus petit entier non signé conforme à l'entier source modulo 2 de rang n, où n est le nombre de bits utilisés pour représenter le type non signé. Par exemple :

```
unsigned char uc = 1023;
```

L'équivalent binaire est 1111111111. uc devient donc 11111111; c'est-à-dire, 255. Lorsque le type destinataire est signé, la valeur n'est pas modifiée si elle peut y être représentée. Dans le cas contraire, la valeur est définie par votre environnement de développement.

```
signed char sc = 1023;
```

Des résultats plausibles pourraient être 255 et –1. Une valeur booléenne ou énumération peut être convertie implicitement vers son équivalent entier.

Conversions vers le type virgule flottante

Une valeur en virgule flottante peut être convertie vers un autre type virgule flottante. Si la valeur source peut être représentée exactement dans le type destinataire, le résultat est la valeur numérique originale. Si la valeur source se situe entre deux valeurs destinataires adjacentes, le résultat est l'une de ces valeurs. Sinon, le comportement est indéfini. Par exemple :

```
float f = FLT_MAX; // valeur virgule flottante maxie

double d = f;  // ok : d == f
float f2 = d;  // ok : f2 == f
double d3 = DBL_MAX; // valeur double maxie
float f3 = d3; // indéfinie Si FLT_MAX<DBL_MAX
```

Conversions des pointeurs et références

Tout pointeur d'un type d'objet peut être implicitement converti vers un void* (voir chapitre 3). Le pointeur (référence) d'une classe dérivée peut être implicitement converti en un pointeur (référence) de base accessible et non ambigu. Notez qu'un pointeur de fonction ou un pointeur de membre ne peut être implicitement converti en un void*. Une expression constante qui est évaluée en 0 peut être implicitement convertie en tout pointeur ou pointeur de type membre. Un T* peut être implicitement converti en un const T* (voir chapitre 2). De la même façon, un T& peut être implicitement converti en un const T& (voir chapitre 3).

Conversions booléennes

Les valeurs en virgule flottante, les types booléen, caractère et entier et les pointeurs peuvent être convertis implicitement en bool. Une valeur non nulle est convertie en true, une valeur nulle est convertie en false. Par exemple :

```
void f(int* p, int i)
{
  bool non_nul = p; // true si p!=0
  bool b2 = i; // true si i!=0
}
```

Conversions virgule flottante vers booléen, caractère ou entier

Lorsqu'une valeur virgule flottante est convertie vers une valeur entière, la partie fractionnaire est supprimée. En d'autres termes, une conversion d'un type virgule flottante vers un type entier entraîne une troncation. La valeur de int (1.6), par exemple, est 1. Le comportement est indéfini si la valeur tronquée ne peut être représentée dans le type destinataire. Par exemple :

```
int i = 2.7;     // i devient 2
char b = 2000.7;  // indéfini: 2000 ne peut être représenté
                  // par un char sur 8 bits
```

La qualité des conversions types entiers vers types virgule flottante d'un point de vue mathématique dépend du matériel. Une perte de précision se produit si une valeur de type booléen, caractère ou entier ne peut être représentée exactement comme une valeur du type virgule flottante. Par exemple :

```
int i = float(1234567890);
```

La valeur de i est 1234567936 sur une machine dans laquelle les deux types int et float sont représentés sur 32 bits. De toute évidence, il est préférable d'éviter des conversions implicites susceptibles de détruire des valeurs. En fait, les compilateurs peuvent détecter et émettre un avertissement contre certaines conversions dangereuses, telles que les conversions virgule flottante vers booléen, caractère ou entier, et long int et vers char.

Conversions arithmétiques usuelles

Ces conversions s'appliquent sur les opérandes d'un opérateur binaire pour leur affecter un type commun, qui sera ensuite utilisé pour le résultat :

- Si l'un des opérandes est de type long double, l'autre est converti en un long double.
  - Sinon, si l'un des opérandes est de type double, l'autre est converti en un type double.
  - Sinon, si l'un des opérandes est de type float, l'autre est converti en un type float.
  - Sinon, les promotions intégrales (voir plus haut section « Promotions ») sont appliquées sur chaque opérande.
- Ensuite, si l'un des opérandes est de type signed long, l'autre est converti en un type unsigned long.
  - Sinon, si un opérande est de type long int et l'autre de type unsigned int, et donc si un type long int peut représenter toutes les valeurs d'un type unsigned int, le type unsigned int est converti en un long int. Dans le cas contraire, les deux opérandes sont convertis en unsigned long int.
  - Sinon, si l'un des opérandes est de type long, l'autre est converti en un type long.
  - Sinon, si l'un des opérandes est de type unsigned, l'autre est converti en un type unsigned.
  - Sinon, les deux opérandes sont de type int.

Le texte original de cette fiche pratique est extrait de «Tout sur le C++» (Christine EBERHARDT, Collection

CommentCaMarche.net, Dunod, 2009)

‹ Précédent

- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61

Suivant ›

Réalisé sous la direction de Jean-François PILLOU,
fondateur de CommentCaMarche.net.