

La grande souplesse du langage C++ permet de mélanger des données de différents types dans une expression. Avant de pouvoir calculer, les données doivent être converties dans un même type. La plupart de ces conversions se passent automatiquement, sans l'intervention du programmeur, qui doit quand même prévoir leur effet.

Parfois il est nécessaire de convertir une donnée dans un type différent de celui que choisirait la conversion automatique. Dans ce cas, nous devons forcer le changement de type explicitement à l'aide d'un opérateur spécial de conversion. Cette technique s'appelle le casting.

## Les conversions de type automatiques

### Calculs :

Lors d'un calcul entre valeurs de divers types on assiste de manière implicite à une conversion de type automatique par le compilateur. Ces manipulations implicites convertissent en général des types plus **petits** en des types plus **larges**, pour éviter une perte de précision. Cette technique s'appelle : **faire une promotion vers le type supérieur**.

### Affectations :

Lors d'une affectation, la donnée à droite située du signe d'égalité est convertie dans le même type proposé par la variable située à gauche du signe d'égalité. Dans ce cas, il peut y avoir une perte de précision si le type de la destination est plus faible que celui de la source.

### Exemples :

```
int i = 8;
float x = 12.5F;
double y = i * x; // Pour pouvoir être multiplié avec x, la valeur de i est convertie en float (le type le plus large des deux). Le résultat de la multiplication est du type float ( 100.0F ), mais avant d'être affecté à y, il est converti en double. Nous obtenons comme résultat :
y = 100.0
```

## Règles de conversion automatiques

Précision du plus petit au plus grand :

**bool ↔ char ↔ short ↔ int ↔ long ↔ float ↔ double ↔ long double**

Types	Description
bool	En interne, ce type est en fait considéré comme un entier. La constante littérale <b>false</b> est la valeur <b>0</b> , et la constante littérale <b>true</b> est la valeur <b>1</b> . Ce type bool est récent, et avant que ce type n'existe, <b>0</b> et <b>1</b> étaient utilisés pour palier ce manque. <b>0</b> désigné la non existence d'une valeur, et <b>1</b> l'existence (en fait, tout autre valeur que <b>0</b> indique l'existence).
char	Ce type représente un caractère. Il est alors possible de travailler directement avec des constantes littérales caractères qu'il faut alors placer entre les apostrophes. Toutefois, en interne, le compilateur considère le caractère comme une valeur entière représentée par son code ASCII.  char c = 'A'; <=> char c = 65; <=> char c = 0x41 ; unsigned char octet = 0x2A; // On peut réaliser du traitement binaire sur une valeur de 8 bits.
<b>Conversions automatiques lors d'une opération avec :</b>	
Deux entiers	Nous venons de voir que les types <b>bool</b> et <b>char</b> sont considérés comme des entiers. Alors, si nécessaire, les types <b>bool</b> , <b>char</b> et <b>short</b> sont convertis en <b>int</b> . Ensuite, l'ordinateur choisit le plus large des deux types dans l'échelle suivante :  int, unsigned int, long, unsigned long
Un entier et un réel	Le type entier est converti dans le type du réel.
Deux réels	Le compilateur choisit le plus large des deux types selon l'échelle suivante :  float, double, long double
Affectations et opérateur d'affectation	Lors d'une affectation, le résultat est toujours converti dans le type de la destination. Si ce type est plus faible, il peut y avoir une perte de précision.  double dval = 8.6 ; int ival = 5 ; ival += dval + 0.5 ; // assure l'arrondi

### Exemple :

Observons les conversions nécessaires lors d'une simple division.

```
int X;
float A=12.48F;
char B=4;
X=A/B;
```

**B** est converti en **float**. Le résultat de la division est du type **float** (valeur **3.12F**) et sera converti en **int** avant d'être affecté à **X**, ce qui conduit au résultat **X=3**.

### Exemple :

Dans cet exemple, nous divisons 3 par 4 à trois reprises et nous observons que le résultat ne dépend pas seulement du type de la destination, mais aussi du type des opérandes.

```
...
char A=3;
int B=4;
float C=4;
float D, E;
char F;
D = A/C;
E
=
A/B;
```

F = A/C;

...



Pour le calcul de D, A est converti en float et divisé par C. Le résultat (0.75F) est affecté à D qui est aussi du type float. On obtient donc:  
D = 0.75F.

Pour le calcul de E, A est converti en int et divisé par B. Le résultat de la division (type int, valeur 0) est converti en float. On obtient donc:  
E = 0.0F.

Pour le calcul de F, A est converti en float et divisé par C. Le résultat (0.75F) est retraduit en char. On obtient donc:  
F='0' ou F = 0.

## Perte de précision

Lorsque nous convertissons une valeur en un type qui n'est pas assez précis ou pas assez grand, la valeur est coupée sans arrondir et sans nous avertir...

### Exemple :

```
unsigned short A = 70000; // la valeur de A sera : 70000 mod 65536 = 4464
```

## Les conversions de type forcées (casting)

Effectuons l'expérience suivante :

```
char A=3;
int B=4;
double C;
C = A/B;
```



Quel est le type de division réalisée ?  
Quelle est la valeur de C ?  
Est-ce la valeur attendue ?

Nous désirions obtenir un résultat de type réel et donc a priori, d'avoir un traitement correspondant pour avoir le maximum de précision possible. Le problème, c'est que, d'après cette écriture, nous n'avons pas réalisé une division sur des nombres réels. En effet, la promotion de type indique qu'il s'agit uniquement d'une division entière.

Ce type de situation peut se présenter. Toutefois, il est possible de convertir explicitement une valeur en un type quelconque en forçant la transformation. Cette opération est appelée le **casting** (conversion de type forcée).

### Syntaxe :

```
(nouveau-type) Expression ;
```

### Exemple :

Reprenons l'exemple précédent. Nous divisons deux variables du type entier. Pour avoir plus de précision, nous voulons avoir un résultat de type réel. Pour ce faire, nous convertissons l'une des deux opérandes en double. Automatiquement, le compilateur convertira l'autre opérande en double et effectuera une division réelle:

```
char A=3;
int B=4;
double C;
C = (double)A / B;
```



La valeur de A est explicitement convertie en double. La valeur de B est automatiquement convertie en double. Le résultat de la division (type réel, valeur 0.75) est affecté à C.  
Résultat: C = 0.75



**Attention :**  
Les contenus de A et de B restent inchangés, seules les valeurs utilisées dans les calculs sont converties.

### Exercices :

Traitement	Résultats
char lettre = 'B' + 'a'-'A' ;	lettre = ?
char lettre = 'c' + ? ;	Déterminer '?' pour que : lettre = 'C'
0<a<2	Donner le résultat de l'expression avec : int a = 1 ; ----- 0<a<2 <=> ? int a = -1 ; ----- 0<a<2 <=> ? int a = 2 ; ----- 0<a<2 <=> ? Conclusion ? Trouvez une expression plus adaptée



**Corrections :** [Visualisation du résultat](#)

Travaux Pratiques