

Langage C

[Le Langage C](#)

[Introduction](#)

[1 - Premier programme en C](#)

[2 - Un exemple de programme plus évolué](#)

[3 - Variables-constantes-affectation](#)

[4 - Opérateurs - Conversions](#)

[4.1. Quelles sont les priorités quand on mélange des opérateurs ?](#)

[4.2. Les opérateurs arithmétiques : + - * / % \(modulo\)](#)

[4.3. L'opérateur d'affectation = \(« reçoit »\)](#)

[4.4. Les conversions de type : implicites et explicites \(cast\)](#)

[4.4.1 Les conversions implicites](#)

[4.4.2 Les conversions explicites : l'opérateur de cast](#)

[4.5. Les opérateurs relationnels : inférieur, supérieur, égal, différent...](#)

[4.6. Les opérateurs logiques : ET OU NON](#)

[4.7. Opérateurs de manipulation de bits – masques \(ET bit à bit, décalage...\)](#)

[4.8. L'opérateur d'adresse &](#)

[4.9. Les opérateurs d'incrément et de décrémentation ++ --](#)

[4.10. L'opérateur sizeof \(taille en octets\)](#)

[5 - Les structures de contrôle](#)

[6 - Les entrées/sorties](#)

[7 - Utilisation de fonctions](#)

[8 - La bibliothèque de fonctions \(sinus, exp, valeur absolue...\)](#)

[9 - Définition de fonction](#)

[10 - La compilation séparée \(multi-fichiers\)](#)

[11 - Les tableaux](#)

[12 - Les chaînes de caractères](#)

[13 - Les pointeurs](#)

[14 - Pointeurs et tableaux à une dimension](#)

[15 - Pointeurs et chaînes de caractères](#)

[16 - Les structures](#)

[17 - Les fichiers](#)

[18 - Les simplifications d'écriture](#)

[19 - Les classes d'allocation mémoire](#)

[20 - Etes-vous un « bon » programmeur ?](#)

[Annexes](#)

[Moteur de recherche](#)

4.4.1 Les conversions implicites



Les conversions implicites

Les conversions implicites sont effectuées par le compilateur pour l'évaluation d'une expression.

Prenons le cas de l'affectation suivante :

```
var_destination = expression ;
```

où expression peut comporter un mélange de variables de types différents et d'opérateurs. Par exemple :

```
var_double = var_int * var_float ;
```

Les règles sont les suivantes :

- Le type de la variable de destination (*Leftvalue*) n'intervient pas pendant le calcul de l'expression située à droite de l'opérateur =. Ce n'est qu'**après** le calcul de celle-ci que la valeur est éventuellement convertie pour s'exprimer selon le type de la variable de destination.

- L'expression à droite de l'opérateur = est évaluée par défaut **de la gauche vers la droite** en respectant les **priorités** des opérateurs rencontrés.

- Afin de fournir deux opérandes de même type à l'opérateur qui va être appliqué, le compilateur convertit si nécessaire l'opérande le plus « faible » dans le type de la variable occupant le plus de place en mémoire. Il existe donc une hiérarchie pour les conversions :

char < short int < int < long int < float < double

Exemple 9, Conversions implicites :

```
1 var_double = var_int * var_float ;
```

Le produit est effectué dans le type dominant float (`var_int` est pour cela convertie en float par le compilateur). Puis son résultat est converti en double au moment de l'affectation à la variable `var_double` (sans perte d'information dans ce sens).

```
1 var_float = 45 + var_int * var_double ;
```

On commence par calculer le produit (prioritaire) qui est effectué dans le type dominant double. Puis son résultat (de type *double*) est ajouté (addition réelle) au *double* résultant de la conversion de l'entier 45. Enfin, le résultat *double* de l'addition est converti (c'est-à-dire tronqué) en float au moment de l'affectation à la variable `var_float`, avec perte d'information.

[« Précédent »](#) | [Suivant »](#)