Le passage des arguments

En C++, on distingue pour les fonctions
2 types de **passage d'arguments**:

☞ le *passage par **valeur***:

dans lequel la fonction travaille sur
des **copies locales** des arguments transmis

☞ et le *passage par **référence***:
dans lequel la fonction travaille effectivement
l'argument **transmis** lui-même**.**

Passage par valeur (1)

La variable locale associée à un argument formel
passé par valeur correspond à une ***copie locale*** de l'argument
utilisé lors de l'appel de la fonction.

C'est donc une entité distincte,
mais de même valeur littérale, et en conséquence,
les modifications éventuelles apportées à l'argument formel
à l'intérieur de la fonction n'affectent pas la variable originale:

Les modifications effectuées à l'intérieur de la fonction
ne sont pas répercutées à l'extérieur.

Passage par valeur (2)

Exemple

```
void f(int x)
{
   x = x + 2;
   cout << "'x' vaut: " << x << endl;
}

void main()
{
   int val(0);
   f(val);
   cout << "'val' vaut: " << val << endl;
}
```

```
'x' vaut: 2
'val' vaut: 0
```

La modification de l'argument formel `x` effectuée dans la fonction `f()` laisse inchangée la valeur de la variable `val` utilisée comme argument (associé à `x`) lors de l'appel de la fonction..

De ce fait, il est tout à fait possible d'utiliser des valeurs littérales comme argument lors des appels aux fonctions [pour les arguments passés par valeur].

Passage par référence (1)

La variable locale associée à un argument formel ***passé par référence*** ne correspond qu'à un autre nom (synonyme local) pour l'argument utilisé lors de l'appel de la fonction[7].

Aucune nouvelle variable n'est crée, et en conséquence, les modifications éventuelles apportées à l'argument formel à l'intérieur de la fonction affectent également l'argument original:

> Les modifications effectuées à l'intérieur de la fonction **sont répercutées à l'extérieur**.

Le passage par référence doit être explicitement indiqué. Pour ce faire, on ajoute le symbole «`&`» au type des arguments formels concernés (par exemple: `int&`, `bool&`, ...).

7. Ce type de «variable» est appelé ***référence***, d'où le nom de *passage par référence*.

Passage par référence (2)

Exemple

```
void f(int& x) {
   x = x + 2;
   cout << "'x' vaut: " << x << endl;
}
void main() {
   int val(0);
   f(val);
   cout << "'val' vaut: " << val << endl;
}
```

```
'x' vaut: 2
'val' vaut: 2
```

Remarquons que dans le cas d'arguments passés par référence, les valeurs littérales ne peuvent être utilisés comme arguments lors de l'appel à la fonction que pour des arguments formels déclarés constants.