Arguments par défaut

Lors de son prototypage, une fonction peut donner des **valeurs par défaut** à ses paramètres.
Il n'est alors pas nécessaire de fournir d'argument à ces paramètres lors de l'appel de la fonction.

La syntaxe d'un paramètre avec valeur par défaut est :

type identificateur = valeur

Attention : Les paramètres avec valeur par défaut doivent apparaître **en dernier** dans la liste des paramètres d'une fonction.

Arguments par défaut : exemple

Exemple :

```
void affiche_ligne(char elt, int nb = 5);

int main() {
  affiche_ligne('*');
  affiche_ligne('+', 8);
  return 0;
}

void affiche_ligne(char elt, int nb) {
  for(int i(0); i < nb; ++i) {
    cout << elt;
  }
  cout << endl;
}
```

Résultat :

```
*****
++++++++
```

Lors de l'appel `affiche_ligne('*')`, la valeur par défaut 5 est utilisée ; c'est strictement équivalent à `affiche_ligne('+', 5)`

Lors de l'appel `affiche_ligne('+', 8)`, la valeur explicite 8 est utilisée.

Arguments par défaut : Remarques

- Les arguments par défaut se spécifient dans le **prototype** et non pas dans la définition de la fonction
- Lors de l'appel à une fonction avec plusieurs paramètres ayant des valeurs par défaut, les arguments omis doivent être les derniers et omis **dans l'ordre** de la liste des paramètres.

  Exemple :
  `void f(int i, char c = 'a', double x = 0.0);`

  `f(1)` → correct (vaut `f(1, 'a', 0.0)`)
  `f(1, 'b')` → correct (vaut `f(1, 'b', 0.0)`)
  `f(1, 3.0)` → **incorrect !**
  `f(1, , 3.0)` → **incorrect !**
  `f(1, 'b', 3.0)` → correct

La surcharge de fonctions

En C++, il est de ce fait possible de définir **plusieurs fonctions de même nom** si ces fonctions n'ont pas les mêmes listes de paramètres : nombre ou types de paramètres différents.

Ce mécanisme, appelé **surcharge des fonctions**, est très utile pour écrire des fonctions « *sensibles* » au type de leurs arguments
c'est-à-dire des fonctions correspondant à des traitements de même nature mais s'appliquant à des entités de types différents.

La surcharge de fonctions : exemple

```
void affiche(int x) {
  cout << "entier : " << x << endl;
}
void affiche(double x) {
  cout << "reel : " << x << endl;
}
void affiche(int x1, int x2) {
  cout << "couple : " << x1 << x2 << endl;
}
```

`affiche(1)`, `affiche(1.0)` et `affiche(1,1)` produisent alors des affichages différents.

Remarque :

```
void affiche(int x);
void affiche(int x1, int x2 = 1);
```

est interdit ! ☞ ambiguïté