Fonction

Mettre au point du code robuste requiert d'effectuer les calculs ***le plus localement possible****, chaque fonction agissant comme une* boîte noire *indépendante du reste du code. C'est pourquoi on privilégie le passage par valeur ou par référence constante.*

Passage par valeur*: à privilégier*

La valeur de l'argument fourni à l'appel est copiée et sert à initialiser un paramètre qui se comporte comme une variable locale à la fonction.

Passage par référence*: pour modifier une variable*

L'argument fourni à l'appel doit être un nom de variable du même type que le paramètre. Le paramètre agit comme un second nom pour accéder et éventuellement modifier la variable depuis la fonction.

Si **const** est associé au paramètre ; la variable n'est pas modifiable : *à retenir pour des entités qui seraient lourdes à transmettre par valeur*.

Passage de la valeur d'un pointeur

A utiliser seulement si on ne peut pas exploiter les deux autres méthodes (fiche distincte).

R. Boulic, adapté de J-C Chappelier et J. Sam

Prototype: une seule fois en début de fichier.
Valeur par défaut possible mais avec restriction.

```
type nomf(type nom_param,…);
```

Définition = prototype suivi du corps de la fonction détaillant toutes ses instructions.

```
type nomf(type nom_param,…)
{
    …
    return expr;//même type que la fonction
}
```

Appel = possible dans le code après le prototype.
La *valeur effective* de chaque **argument** est associée au paramètre correspondant, sinon c'est la valeur par défaut qui est transmise.

```
nomf(argument,…);
```

L'appel peut faire partie d'une expression si la fonction retourne une valeur.

Fin de fonction

return quitte une fonction de type **void**

return ***valeur*** quitte la fonction en renvoyant une valeur du même type que la fonction

exit(*entier*) ; cet appel quitte aussi le programme avec succes: exit(0); avec échec : exit(1);