Les procédures et les fonctions

Notion de réutilisabilité

- Pour l'instant, un programme est une séquence d'instructions mais sans **partage** des parties importantes ou utilisées plusieurs fois.
- Le bloc suivant peut être exécuté à plusieurs endroits

```
Répéter
        Ecrire("Entrez un nombre entre 1 et 100 : ")
        Lire(i)
TantQue ((i < 1) ou (i > 100))
```

- Bonne pratique : **Ne jamais dupliquer** un bloc de code **!**
- Ce qu'on veut recopier doit être mis dans une **fonction.**

Les procédures et les fonctions

- Résoudre le problème suivant :

Écrire un programme qui affiche, en ordre croissant, les notes d'une classe suivies de la note la plus faible, de la note la plus élevée et de la moyenne.

revient à résoudre les sous problèmes suivants :

-Remplir un tableau des notes saisies par l'utilisateur
-Afficher un tableau des notes
-Trier un tableau de notes en ordre croissant
-Trouver le plus petit réel d'un tableau
-Trouver le plus grand réel d'un tableau
-Calculer la moyenne d'un **tableau** de réels

Les procédures et les fonctions

Algorithme TraitTableau;

Variables tableau tab[100] : réel;

pPetit, pGrand, moyen : réel;

Début

Saisir(tab);
Afficher(tab);
pPetit←plusPetitElements(tab);
pGrand←plusGrandElements(tab);
moyen ←calculerMoyen(tab);
Trier(tab);

Fin

Les procédures et les fonctions

Chacun de ces sous-problèmes devient un nouveau problème à résoudre.

Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait " quasiment" résoudre le problème initial.

Donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial.

En algorithmique, il existe deux types de sous-programmes :

-Les fonctions

-Les procédures

- Un programme long est souvent difficile à écrire et à comprendre.
- C'est pourquoi, il est préférable de le décomposer en des parties appelées **sous-programmes** ou **modules.**
- Les **fonctions** et les **procédures** sont des modules (groupe d'instructions) indépendants désignés par un nom.

Les procédures et les fonctions

Elles ont plusieurs avantages :

- Permettent d'éviter de réécrire un même traitement plusieurs fois. En effet, on fait appel à la procédure ou à la fonction aux endroits spécifiés.
- Permettent d'**organiser le code** et améliorent la **lisibilité** des programmes.
- **Facilitent la maintenance** du code (il suffit de modifier une seule fois).
- Ces procédures et fonctions peuvent éventuellement être **réutilisées** dans d'autres programmes.

Les fonctions

- Le **rôle** d'une fonction en programmation est similaire à celui d'une fonction en mathématique : elle **retourne un résultat au programme appelant.**

- Le corps de la fonction est la portion de programme à réutiliser ou à mettre en évidence.

- Les paramètres de la fonction : (les «entrées», ou les «arguments») ensemble de variables extérieures à la fonction dont le corps dépend pour fonctionner.

Les fonctions

- Une fonction s'écrit en dehors du programme principal sous la forme:

```
Fonction nom_fonction (paramètres et leurs types) : type_fonction
  Variables // variables locales
Début
  Instructions; //le corps de la fonction
  retourne;       //la valeur à retourner
FinFonction
```

- Le nom_fonction : désignant le nom de la fonction.
- type_fonction est le type du résultat retourné
- L'instruction retourne sert à retourner la valeur du résultat.

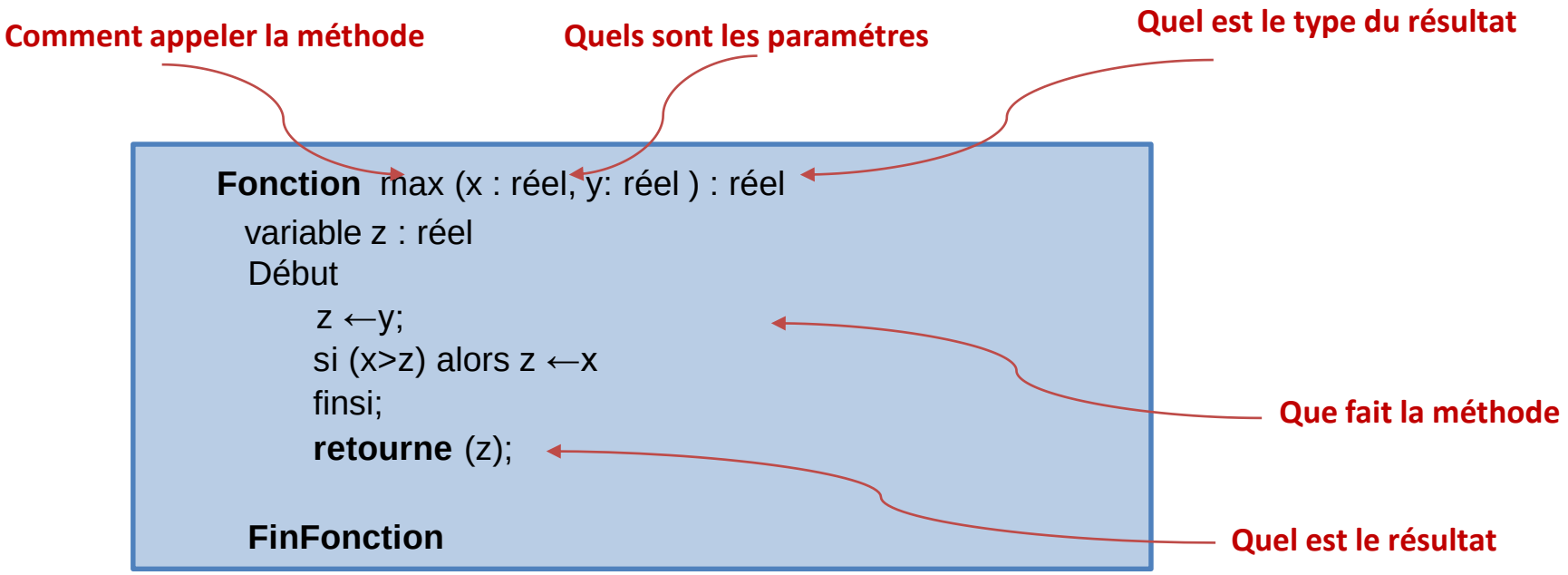
Les fonctions

Caractéristiques des fonctions

- La fonction est désignée par son nom.
- Une fonction ne modifie pas les valeurs de ses arguments en entrée.
- Elle se termine par une instruction de retour qui rend un résultat et un seul.
- Une fonction est toujours utilisée dans une expression (affectation, affichage,…) ayant un type compatible avec le type de retour de la fonction.
- On doit fournir une valeur pour chacun des arguments définis pour la fonction (certains langages acceptent des valeurs par défaut).

Les fonctions

- La fonction max suivante retourne le plus grand des deux réels x et y fournis en arguments :



- La fonction Pair suivante détermine si un nombre est pair :

```
Fonction Pair (n : entier ) : booléen
Debut
  retourne (n%2=0);
 FinFonction
```

Les fonctions

- L'utilisation d'une fonction se fera par simple écriture de son nom dans le programme principale. Le résultat étant une valeur, devra être affecté ou être utilisé dans une expression, une écriture, ...

- **Exemple: Algorithme exepmleAppelFonction**

```
variables c : réel, b : booléen
Début
 b ←Pair(3);
 c ←5*max(7,2)+1;
 écrire("max(3,5*c+1)= ", max(3,5*c+1));

Fin
```

- Lors de l'appel Pair(3); le paramètre **formel** n est remplacé par le paramètre **effectif** 3.

Les fonctions

- L'évaluation de l'**appel :** f(arg1, arg2, ..., argN); d'une fonction définie par :

  typeR f(type1 x1, type2 x2, ..., typeN xN) { ... }

  s'effectue de la façon suivante :

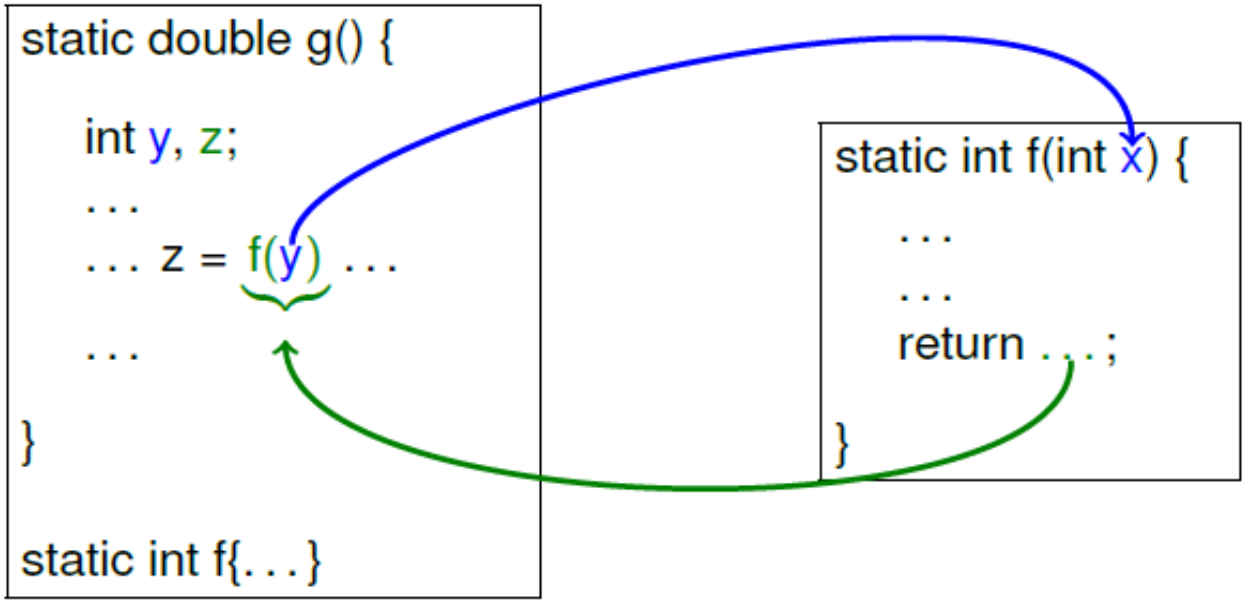
- 1. Les expressions arg1, arg2, ..., argN passées en argument sont évaluées.
- 2. les valeurs correspondantes sont **affectées** aux paramètres x1, x2, ..., xN de la fonction f (variables locales à f).

- Concrètement, ces deux premières étapes reviennent à faire : x1<-arg1; x2 <- arg2; ...;xN <- argN;

Les fonctions

- 3. les instructions correspondantes au corps de la fonction f sont exécutées.
- 4. l'expression suivant la première commande return rencontrée est évaluée...
- 5. ...et retournée comme résultat de l'appel : cette valeur remplace l'expression de l'appel, c-à-d l'expression f(arg1, arg2, ..., argN);

- L'évaluation de l'appel d'une méthode peut être schématisé de la façon suivante :



Les fonctions

- Ecrire une fonction qui calcule la somme de deux entiers donnés en paramètres.
- Ecrire une fonction qui calcule le produit de trois réels donnés en paramètres.
- Ecrire une fonction Min_2 qui calcule le minimum de deux entiers donnés en paramètres.
- Ecrire une fonction qui teste si n entier est divisible par 3 ou non.
- Ecrire une fonction qui prend en paramètres un nombre entier N et qui calcule la somme des nombres impaires compris entre 1 et N.