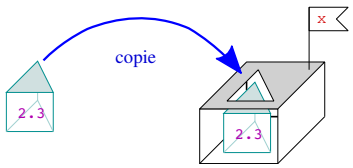
Évaluation d'un appel de fonction

```
double moyenne (double x, double y)
{
  return (x + y) / 2.0;
}
```

Que se passe-t-il lors de l'appel suivant :

```
z = moyenne( 1.5 + 0.8, 3.4 * 1.25 );
```

1. évaluation des expressions passées en arguments :
   `1.5 + 0.8` ⟶ `2.3`
   `3.4 * 1.25` ⟶ `4.25`
2. affectation des paramètres :
   `x = 2.3`
   `y = 4.25`
3. exécution du corps de la fonction :
   rien dans ce cas (corps réduit au simple `return`)
4. évaluation de la valeur de retour (expression derrière `return`)
   `(x + y) / 2.0` ⟶ `3.275`
5. replacement de l'expression de l'appel par la valeur retournée :
   `z = 3.275;`



Évaluation d'un appel de fonction (résumé)

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 au corps de *f*)

   Concrètement, ces deux premières étapes reviennent à faire :

   x1 = *arg1*, x2 = *arg2*, ..., xN = *argN*
3. le programme correspondant au corps de la fonction *f* est exécuté
4. l'expression suivant la première commande `return` rencontrée est évaluée...
5. ...et retournée comme résultat de de l'appel :
   cette valeur remplace l'expression de l'appel, i.e. l'expression

   f(arg1, arg2, ..., argN)

Évaluation d'un appel de fonction (résumé)

L'évaluation de l'**appel** d'une fonction s'effectue de la façon suivante :

1. les *expressions* passées en argument sont évaluées
2. les valeurs correspondantes sont **affectées** aux paramètres de la fonction
3. le corps de la fonction est exécuté
4. l'expression suivant la première commande `return` rencontrée est évaluée...
5. ...et retournée comme résultat de de l'appel :
   cette valeur remplace l'expression de l'appel

Les étapes 1 et 2 n'ont pas lieu pour une fonction sans arguments.

Les étapes 4 et 5 n'ont pas lieu pour une fonction sans valeur de retour (`void`).

L'étape 2 n'a pas lieu lors d'un passage par référence (voir plus loin).

Appel : autre exemple

Une fonction peut appeler une autre fonction.
Il faut simplement respecter la règle d'or : avoir prototypé la fonction avant l'appel

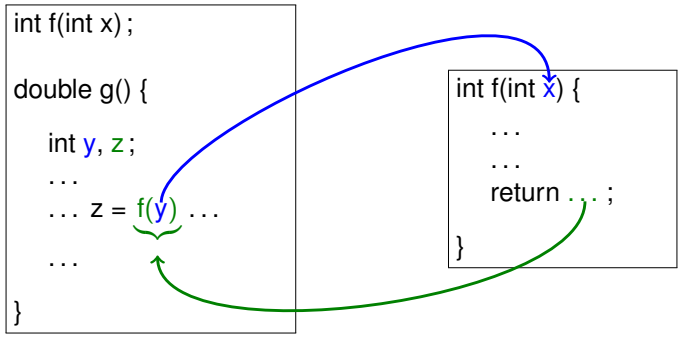
```
int score (double points, double temps_jeu);
void affiche_score (int joueur, double points, double temps_jeu);

void affiche_score(int joueur, double points, double temps)
{
  cout << "  Joueur " << joueur
       << score(points, temps) << " points" << endl;
}

int score (double points, double temps_jeu)
{ // ... comme avant ...
}
```

Appel : résumé

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



Résumé du jargon

« *Appeler la fonction* `f` » = utiliser la fonction `f` : `x = 2 * f(3);`

« *3 est passé en argument* » = (lors d'un appel) la valeur 3 est copiée dans un paramètre de la fonction :

`x = 2 * f(3);`

« *la fonction retourne la valeur de y* » = l'expression de l'appel de la fonction sera remplacée par la valeur retournée

```
 return y;
}
...
x = 2 * f(3);
```

Autres exemples : « `cos(0)` retourne le cosinus de 0 », « `cos(0)` retourne 1 ».