

# *Éléments d'informatique – Cours 7. Fonctions*

## *(1)*

Pierre Boudes

8 novembre 2011



This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.



## *Cours en plusieurs parties sur les fonctions* ~~✎~~

1. Principe de base, intérêt et analogie mathématique
2. Fonctions sans entrée ou sans sortie, effets de bord
3. Autres types que les entiers pour les paramètres et la sortie
4. Pile d'appel, fonctions récursives

## *Plan de la séance* ✎

*Définition et intérêt des fonctions en informatique*

*Analogie mathématique*

La notion mathématique de fonction

*Les fonctions en langage C*

Appeler des fonctions

Créer des fonctions : déclarer, définir

Traces

Fonctions sans valeurs de retour (void)

## *Définition et intérêt des fonctions en informatique*

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*



## *Définition et intérêt des fonctions en informatique*

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*

Intérêt des fonctions :

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique.



## Définition et intérêt des fonctions en informatique

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*

Intérêt des fonctions :

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique.
- *réutilisation* : une fonction peut être utilisée par plusieurs programmes (bibliothèque) ;



## Définition et intérêt des fonctions en informatique

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*

Intérêt des fonctions :

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique.
- *réutilisation* : une fonction peut être utilisée par plusieurs programmes (bibliothèque) ;
- *lisibilité* : regrouper un ensemble d'instructions dans une fonction, nommée de façon explicite, facilite la relecture du code et cache les détails de codage ;



## Définition et intérêt des fonctions en informatique

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*

Intérêt des fonctions :

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique.
- *réutilisation* : une fonction peut être utilisée par plusieurs programmes (bibliothèque) ;
- *lisibilité* : regrouper un ensemble d'instructions dans une fonction, nommée de façon explicite, facilite la relecture du code et cache les détails de codage ;
- *structuration* : découper en sous-problèmes ; distribuer leur programmation à différentes personnes, à différentes étapes de la réalisation d'un projet.



## Définition et intérêt des fonctions en informatique

*En informatique, une fonction est une portion de code représentant un sous programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme. (wikipédia)*

Intérêt des fonctions :

- *factorisation* : éviter la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique.
- *réutilisation* : une fonction peut être utilisée par plusieurs programmes (bibliothèque) ;
- *lisibilité* : regrouper un ensemble d'instructions dans une fonction, nommée de façon explicite, facilite la relecture du code et cache les détails de codage ;
- *structuration* : découper en sous-problèmes ; distribuer leur programmation à différentes personnes, à différentes étapes de la réalisation d'un projet.

## *Analogie mathématique*

- De même que le terme *variable* peut avoir deux sens différents en informatique et en mathématiques, le terme *fonction* recouvre des réalités différentes.

## *Analogie mathématique* ✎

- De même que le terme *variable* peut avoir deux sens différents en informatique et en mathématiques, le terme *fonction* recouvre des réalités différentes.
- Pour les fonctions, les deux notions sont historiquement très proches. C'est bien pour écrire des fonctions mathématiques que les fonctions informatiques ont été introduites.

## *Analogie mathématique* ✎

- De même que le terme *variable* peut avoir deux sens différents en informatique et en mathématiques, le terme *fonction* recouvre des réalités différentes.
- Pour les fonctions, les deux notions sont historiquement très proches. C'est bien pour écrire des fonctions mathématiques que les fonctions informatiques ont été introduites.
- Mais qu'est-ce qu'une fonction mathématique ?



## *La notion mathématique de fonction*

Une fonction mathématique est :



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?
- Une expression avec inconnues ( $x^2 + 1$ ), une formule ?



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?
- Une expression avec inconnues ( $x^2 + 1$ ), une formule ?
- Un terme dans une algèbre de fonctions ( $f \circ g$ ,  $f'$ ) ?



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?
- Une expression avec inconnues ( $x^2 + 1$ ), une formule ?
- Un terme dans une algèbre de fonctions ( $f \circ g, f'$ ) ?
- Une relation entre deux ensembles, entrées et sorties, telle que chaque entrée est en relation avec au plus une sortie (toujours la même). Le graphe de la fonction.



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?
- Une expression avec inconnues ( $x^2 + 1$ ), une formule ?
- Un terme dans une algèbre de fonctions ( $f \circ g, f'$ ) ?
- Une relation entre deux ensembles, entrées et sorties, telle que chaque entrée est en relation avec au plus une sortie (toujours la même). Le graphe de la fonction.



## *La notion mathématique de fonction*

Une fonction mathématique est :

- Une courbe ?
- Une expression avec inconnues ( $x^2 + 1$ ), une formule ?
- Un terme dans une algèbre de fonctions ( $f \circ g, f'$ ) ?
- Une relation entre deux ensembles, entrées et sorties, telle que chaque entrée est en relation avec au plus une sortie (toujours la même). Le graphe de la fonction.

### *À retenir*

En mathématiques, la manière dont la fonction est calculée ne fait pas partie de l'objet défini. Une fonction est une boîte noire.

*Déclarer, appeler, définir.*

## *Déclarer, appeler, définir.*

- *Déclarer* : comme les variables, les fonctions doivent être déclarées avant usage pour fixer **le nombre** et le type des arguments et de la sortie.

## *Déclarer, appeler, définir.*

- *Déclarer* : comme les variables, les fonctions doivent être déclarées avant usage pour fixer **le nombre** et le type des arguments et de la sortie.
- *Appeler* : utiliser une fonction, faire appel à son résultat en fixant les valeurs des arguments (paramètres effectifs).

## *Déclarer, appeler, définir.*

- *Déclarer* : comme les variables, les fonctions doivent être déclarées avant usage pour fixer **le nombre** et le type des arguments et de la sortie.
- *Appeler* : utiliser une fonction, faire appel à son résultat en fixant les valeurs des arguments (paramètres effectifs).
- *Définir* : décrire le corps de la fonction, c'est à dire la suite d'instructions qui constitue son calcul (sur des paramètres formels).



## *Utiliser des fonctions : appeler*

Supposons que l'on dispose d'une fonction `maximum`, qui lorsqu'on lui donne deux entiers nous renvoie le plus grand des deux.



## *Utiliser des fonctions : appeler*

Supposons que l'on dispose d'une fonction `maximum`, qui lorsqu'on lui donne deux entiers nous renvoie le plus grand des deux. Nous pouvons l'utiliser de différentes façons :

```
int main ()
{
    int x = 3;
    int y = 4;
    int z;

    z = maximum(x, y);
```



## *Utiliser des fonctions : appeler*

Supposons que l'on dispose d'une fonction `maximum`, qui lorsqu'on lui donne deux entiers nous renvoie le plus grand des deux. Nous pouvons l'utiliser de différentes façons :

```
int main ()
{
    int x = 3;
    int y = 4;
    int z;

    z = maximum(x, y);
    z = 2 + maximum(3 * 2, z);
}
```



## *Utiliser des fonctions : appeler*

Supposons que l'on dispose d'une fonction `maximum`, qui lorsqu'on lui donne deux entiers nous renvoie le plus grand des deux. Nous pouvons l'utiliser de différentes façons :

```
int main ()
{
    int x = 3;
    int y = 4;
    int z;

    z = maximum(x, y);
    z = 2 + maximum(3 * 2, z);
    x = maximum(maximum(3, y) + 1, z - 1);
}
```



## *Utiliser des fonctions : appeler*

Supposons que l'on dispose d'une fonction `maximum`, qui lorsqu'on lui donne deux entiers nous renvoie le plus grand des deux. Nous pouvons l'utiliser de différentes façons :

```
int main ()
{
    int x = 3;
    int y = 4;
    int z;

    z = maximum(x, y);
    z = 2 + maximum(3 * 2, z);
    x = maximum(maximum(3, y) + 1, z - 1);
}
```

Chaque entrée prend une valeur en fonction de l'expression passée en argument. La fonction prend alors sa valeur de sortie.



## *Créer des fonctions (1) déclarer*

Déclarer une fonction c'est donner son prototype (ou signature), c'est à dire son nom, le type de ses arguments, le type de sa sortie.



## *Créer des fonctions (1) déclarer*

Déclarer une fonction c'est donner son prototype (ou signature), c'est à dire son nom, le type de ses arguments, le type de sa sortie.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn);
```



## *Créer des fonctions (1) déclarer*

Déclarer une fonction c'est donner son prototype (ou signature), c'est à dire son nom, le type de ses arguments, le type de sa sortie.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn);
```

### *Remarque*

Seule la déclaration des types est importante pour l'analyse sémantique. Cependant, le choix de noms de paramètres explicites fournit une documentation minimale.



## *Créer des fonctions (1) déclarer*

Déclarer une fonction c'est donner son prototype (ou signature), c'est à dire son nom, le type de ses arguments, le type de sa sortie.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn);
```

### *Remarque*

Seule la déclaration des types est importante pour l'analyse sémantique. Cependant, le choix de noms de paramètres explicites fournit une documentation minimale.

### *Exemple*

```
int puissance(int, int);  
int puissance(int base, int exposant);
```



## *Créer des fonctions (1) déclarer*

Déclarer une fonction c'est donner son prototype (ou signature), c'est à dire son nom, le type de ses arguments, le type de sa sortie.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn);
```

### *Remarque*

Seule la déclaration des types est importante pour l'analyse sémantique. Cependant, le choix de noms de paramètres explicites fournit une documentation minimale.

### *Exemple*

```
int puissance(int, int);  
int puissance(int base, int exposant);
```

**La déclaration doit se trouver avant tout appel.**



## *Créer des fonctions (2) définir*



## *Créer des fonctions (2) définir*

Définir une fonction c'est donner l'ensemble des instructions qui permettent, à partir des paramètres d'entrée, de calculer la valeur de la fonction, dite valeur de retour, valeur renvoyée.



## *Créer des fonctions (2) définir*

Définir une fonction c'est donner l'ensemble des instructions qui permettent, à partir des paramètres d'entrée, de calculer la valeur de la fonction, dite valeur de retour, valeur renvoyée.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn)
{
    /* declaration et initialisation variables */

    /* instructions */
}
```



## *Créer des fonctions (2) définir*

Définir une fonction c'est donner l'ensemble des instructions qui permettent, à partir des paramètres d'entrée, de calculer la valeur de la fonction, dite valeur de retour, valeur renvoyée.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn)
{
    /* declaration et initialisation variables */

    /* instructions */
}
```

Les noms des paramètres formels sont param1, ..., paramn, ils sont utilisés dans le corps du calcul.



## *Créer des fonctions (2) définir*

Définir une fonction c'est donner l'ensemble des instructions qui permettent, à partir des paramètres d'entrée, de calculer la valeur de la fonction, dite valeur de retour, valeur renvoyée.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn)
{
    /* declaration et initialisation variables */

    /* instructions */
}
```

Les noms des paramètres formels sont param1, ..., paramn, ils sont utilisés dans le corps du calcul.

Dès que la valeur de la fonction est calculée, on utilise l'instruction `return` qui marque la fin du calcul et donne sa valeur à la fonction.



## *Créer des fonctions (2) définir*

Définir une fonction c'est donner l'ensemble des instructions qui permettent, à partir des paramètres d'entrée, de calculer la valeur de la fonction, dite valeur de retour, valeur renvoyée.

```
type_sortie nom_fonction(type1 param1, ..., typen paramn)
{
    /* declaration et initialisation variables */

    /* instructions */
}
```

Les noms des paramètres formels sont param1, ..., paramn, ils sont utilisés dans le corps du calcul.

Dès que la valeur de la fonction est calculée, on utilise l'instruction `return` qui marque la fin du calcul et donne sa valeur à la fonction.

```
return expression_resultat;
```

La définition est obligatoire pour que l'édition de liens réussisse et que l'exécutable soit créé.



## *Variables locales, portée, durée*

Le corps de la fonction peut déclarer des variables additionnelles, qui sont locales à la fonction (portée) et se voient allouer un espace mémoire pour chaque appel de la fonction (durée).

### *Remarque*

`main` est une fonction (appelée au lancement du programme).

## Résumé

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, espace mémoire)



## Résumé

Utilisation des fonctions :

- *déclaration* (types des paramètres et de la valeur de retour)
- *définition* (code, paramètres formels)
- *appel* (paramètres effectifs, espace mémoire)

### *Convention de nommage*

Il est pratique de faire commencer le nom de chaque fonction par un verbe à l'infinitif `convertir_...`, `tester_...`, etc. ou éventuellement par un verbe conjugué : par exemple `est_` pour des fonctions à valeur de test (`est_majeur`, `est_premier`, .....)

Exceptions : la fonction est connue sous un autre nom (`factorielle`, `racine(x)`, `moyenne(...)`)



## Traces

Pour tester nos programmes, nous faisons la trace de chaque appel de chaque fonction que l'on a défini (les fonctions utilisateur, pas les fonctions externes, comme printf).

```
main()
```

ligne	n	Affichage
initialisation	9	
16		



## Traces

Pour tester nos programmes, nous faisons la trace de chaque appel de chaque fonction que l'on a défini (les fonctions utilisateur, pas les fonctions externes, comme printf).

```
main()
```

ligne	n	Affichage
initialisation	9	
16		

```
est_premier(9)
```

ligne	n	i	Affichage
initialisation	9	?	
34		2	
40		3	
38	renvoie FALSE		



## Traces

Pour tester nos programmes, nous faisons la trace de chaque appel de chaque fonction que l'on a défini (les fonctions utilisateur, pas les fonctions externes, comme printf).

```
main()
```

ligne	n	Affichage
initialisation	9	
16		
22		9 n'est pas premier
26		renvoie EXIT_SUCCESS

```
est_premier(9)
```

ligne	n	i	Affichage
initialisation	9	?	
34		2	
40		3	
38	renvoie FALSE		



## *Fonctions sans valeurs de retour (void)*

On parle plutôt de procédure ou de routine car l'analogie avec les fonctions mathématiques est perdu.

### *Déclarer*

```
void afficher_valeurs(int x, int y);
```

### *Appeler*

```
afficher_valeurs(5, 3);
```

### *Définir*

Comme d'habitude mais pas de return (ou return sans argument).

## *Fonctions sans arguments*

### *Déclarer*

```
int nombre_aleatoire();  
int saisie_utilisateur();
```

### *Appeler*

```
int n;  
int secret;  
  
secret = nombre_aleatoire();  
  
n = saisie_utilisateur();
```

### *Définir*

Comme d'habitude.