Algorithmique et programmation – Fonctions (1)

Pierre Boudes

24 septembre 2012

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

Démo

Définition et intérêt des fonctions en informatique

Analogie mathématique

Les fonctions en langage C

- Faire appel à des fonctions
- Créer des fonctions : déclarer, définir

Traces

Procédures et fonctions sans arguments

- Fonctions sans arguments
- Fonctions sans valeurs de retour (void)

Démos et fin

Démo

- Calcul de $y = 3 - |x|$?

Démo

- Calcul de $y = 3 - |x|$?
- Calcul de `somme` = $|x| + |y| + |z|$?

Démo

- Calcul de $y = 3 - |x|$?
- Calcul de `somme` = $|x| + |y| + |z|$?
- Autre façon de définir `abs(x)`.

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 différente 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 différente personnes, à différentes étapes de la réalisation d'un projet.

Analogie mathématique 

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.

Analogie mathématique 

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.
En informatique, on utilise les fonctions comme des boîtes noires mais le mécanisme à l'œuvre dans la boîte, l'algorithme employé, doit être clairement décrit pour définir la fonction.

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). Un appel de fonction est une expression qui s'évalue en la valeur de retour de la fonction.

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). Un appel de fonction est une expression qui s'évalue en la valeur de retour de la fonction.
- *Définir* : décrire le corps de la fonction, c'est à dire la suite d'instructions qui constitue son calcul (sur des paramètres formels) qui se termine par le retour d'une valeur.

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);
  x = maximum( -x, x);
```

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);
  x = maximum( -x, x);
```

Chaque entrée prend une valeur effective fixée par l'expression passée en argument. L'appel de fonction prend alors sa valeur de sortie.

Créer des fonctions (1) déclarer

Déclarer une fonction c'est donner son nom, le type de ses arguments et le type de sa sortie. De cette façon :

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

Créer des fonctions (1) déclarer

Déclarer une fonction c'est donner son nom, le type de ses arguments et le type de sa sortie. De cette façon :

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

On décrit ainsi l'interface de la fonction, sans expliciter l'intérieur de la boîte. On parle encore dde *signature* ou, en C, de *prototype*.

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 nom, le type de ses arguments et le type de sa sortie. De cette façon :

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

On décrit ainsi l'interface de la fonction, sans expliciter l'intérieur de la boîte. On parle encore dde *signature* ou, en C, de *prototype*.

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 nom, le type de ses arguments et le type de sa sortie. De cette façon :

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

On décrit ainsi l'interface de la fonction, sans expliciter l'intérieur de la boîte. On parle encore dde *signature* ou, en C, de *prototype*.

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 des autres 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 des autres 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 des autres 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 des autres 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 réussise 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 locale) 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`, ...)

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 arguments

Déclarer

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

Appeler

```
int n;
int secret;

secret = nombre_aleatoire();

n = saisie_utilisateur();
```

Définir

Comme d'habitude.

Fonctions sans valeurs de retour (void)

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

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).

Démos et fin