Les fonctions

Réfs : www.siteduzero.com

c.developpez.com/

fr.wikipedia.org/wiki/Fonction_(informatique) (plus complet en anglais)

1 Introduction

En Informatique théorique un programme est vu comme la réalisation d'une fonction mathématique : f(x->) = y->, avec x-> les données en entrée et y-> les données en sortie. La notion de fonction mathématique a été importée et adaptée à la programmation par Wilkes, Wheeler et Gill en 1951, parlant de sous-routines ou sous-programmes. En langage C, on parle de fonctions, qui sont un ensemble d'instructions réalisant une certaine tâche : main, printf, scanf, rand, time sont des fonctions.

L'intérêt des fonctions est multiple :

- factorisation : évite la duplication de code en remplaçant les parties dupliquées par un appel à une fonction unique;
- réutilisation : une fonction peut être mise dans une bibliothèque de fonctions et utilisée par plusieurs programmes;
- lisibilité : en regoupant un ensemble d'instructions dans une fonction, nommée de façon explicite, la relecture du code est facilitée; cache les détails de codage à l'utilisateur;
- structuration : découpage en sous-problèmes résolus par des fonctions; distribution de leur programmation à différents développeurs, à différents étapes de la réalisation d'un projet.

Exemple de factorisation avec les fonctions en mathématiques :

$f(x,y) = ((x^2 + 3y - 5) + (x - 1)) / ((2x)^2 + 3y - 5)$

devient :

Soit $g(x,y) = (x^2 + 3y - 5)$:

$f(x,y) = (g(x,y) + (x - 1)) / (g(2x,y)$

Comme pour une fonction mathématique, l'utilisation d'une fonction en C nécessite une partie déclaration et une partie définition :

f : N -> N <----- déclaration

x -> x^2 <----- définition

Cependant, l'analogie avec les fonctions mathématiques s'arrête là, et en programmation, nous pouvons définir des fonctions sans paramètres, sans valeur de retour, avec des effets de bord : des lecteurs/écritures (par exemple afficher un message à l'écran, lire des données au clavier, jouer un son, ou bien piloter une imprimante), des modifications de variables qui ne sont pas définies dans les fonctions.

Exemple de fonctions en C :

```
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */
#include <math.h> /* sqrt, pow */

/* déclaration de fonctions utilisateurs */
/* ma fonction calculant x puissance y */
int puissance(int base,int exposant);

int main()
{
	printf(« la racine carrée de 2 vaut %d.\n »,sqrt(2)); /* sqrt(2) s'évalue comme racine de2 */
	printf(« 2 puissance 8 vaut %d.\n »,pow(2,8)); /* pow(2,8) s'évalue comme 2^8 */
	printf(« avec ma fonction, 2 puissance 8 vaut %d.\n »,puissance(2,8)); /* ma fonction calcule et renvoie 2^8 */

	/* valeur fonction */
	return EXIT_SUCCESS; /* la fonction vaut la valeur de l'expression EXIT_SUCCESS */
}

/* definition de fonctions utilisateur */
int puissance(int base,int exposant)
{
	int res; /* résultat du calcul puissance */
	int i; /* var. Boucle */

	res = 1; /* elt neutre de la multiplication */
	for(i = 0;i < exposant;i = i + 1) /* n fois */
	{
		/* multiplication par base */
		res = res * base;
	}

	/* valeur fonction */
	return res; /* la fonction vaut la valeur de l'expression res */
}
```

2 Déclaration d'une fonction

On parle de déclaration, prototype (ou signature). Cela consiste à déclarer son nom, le type de ses arguments (son domaine) et le type de sa valeur de retour (son co-domaine). Il ne peut y avoir qu'un type de base en retour.

Un prototype de fonction a la forme :

```
type_retour nom_fonction(type1 paramètre1,type2 paramètre2,...,typen
paramètren);
```

La liste peut être vide (()), mais si plusieurs, ils sont séparés par des virgules.

Exemple :

puissance : NxN -> N

x x y -> x^y

int puissance(int x,int y);

La déclaration est obligatoire pour l'analyse sémantique de la compilation et elle doit précéder l'utilisation de la fonction. Une bonne pratique est de déclarer l'ensemble des fonctions avant la fonction main.

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

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

3 Définition d'une fonction

La définition consiste au codage de la fonction, c'est-à-dire à 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. Elle a la forme suivante :

```
type_retour nom_fonction(type1 paramètre1,type2 paramètre2,...,typen paramètren)
{
      /* declaration et initialisation variables */

      /* instructions */
}
```

Une définition reprend la déclaration (sans le point-virgule), suivi d'un bloc ({}) correspondant au corps de la fonction. Les noms des paramètres sont nécessaires car ils sont utilisés dans le corps de la fonction pour calculer la valeur de la fonction. On les appelle les paramètres formels car ils donnent une forme à la définiton de la fonction et peuvent prendre différentes valeurs à chaque appel. Dès que la valeur de la fonction est calculée, on utilise une instruction particulière du C :

```
return expression_résultat;
```

expression_résultat est l'expression qui s'évalue comme la valeur de la fonction. Cette instruction marque la fin du calcul de la fonction et il peut y avoir plusieurs fins conditionnelles.

Exemple :

```
/* definition de fonctions utilisateur */
/* calcul le minimum de a et b */
int min(int a,int b)
{
	if(a < b)
	{
		return a;
	}
	else /* b <= a*/
	{
		return b;
	}
}
```

La définition est obligatoire pour que l'édition de liens réussise et que l'exécutable soit créé. La définition vaut déclaration mais une bonne pratique est de les séparer et de définir l'ensemble des fonctions utilisateur après la fonction main. Cela permet de rendre le code plus lisible, en ayant connaissance de ce qui est déclaré en début de programme et en cachant les détails d'implantation à la fin du fichier.

Le corps de la fonction peut déclarer des variables additionnelles (comme on l'a toujours fait avec la fonction main), qui sont locales à la fonction et n'existent que pour les instructions de la fonction, comme nous allons le préciser au paragraphe suivant.

4 Appel d'une fonction

On appelle appel d'une fonction une expression de la forme :

```
nom_fonction(expression1,expression2,...,expressionN)
```

où nom_fonction est un nom de fonction précédemment déclaré, et expression1,...,expressionN sont les valeurs de ses paramètres formels paramètre1,...,paramètreN, respectivement. On les appelle les paramètres effectifs de la fonction. Cette expression s'evalue comme la valeur retournée par la fonction.

Précisément, un appel de fonction se passe de la façon suivante :

1. allocation d'une zone mémoire propre à la fonction pour contenir les variables
2. déclaration et initialisation des paramètres formels, de la façon suivante :
   - type1 paramètre1 = expression1;
   - type1 paramètre2 = expression2;
   - ...
   - type1 paramètren = expressionn;
3. déclaration et initialisation éventuelles des variables locales
4. saut à la première instruction

5. fin d'exécution de la fonction à la fin de l'exécution de l'instruction return
6. libération de la zone mémoire allouée à la fonction
7. saut à l'instruction suivant l'appel, pour poursuivre l'exécution dans la fonction appelante

Après l'appel, l'expression vaut la valeur retournée.

N.B. : les paramètres formels et les variables déclarées dans le corps de la fonction sont locales à la fonction et n'ont de sens que dans la fonction. Autrement dit, les variables définies dans d'autres fonctions ne sont pas connues localement et on peut avoir plusieurs variables avec le même nom dans différentes fonctions (ex. les variables de boucles).