

C 5 : fonctions

Juliusz Chroboczek

17 octobre 2023

Une *fonction* est un fragment de code qui peut être appelé (exécuté) une ou plusieurs fois dans un programme.

1 Syntaxe

Définition de fonction

```
int f(int x, int y)
{
    int z;
    z = 3 * x;
    return z + y;
}
```

Une *définition de fonction* commence par un entête de fonction. Cet entête consiste de trois parties :

- le *type du résultat* de la fonction, qui indique le type de la valeur qui sera retournée par la fonction;
- le *nom* de la fonction, qui est un identificateur arbitraire qui sert à identifier la fonction;
- la liste de *paramètres formels*, entre parenthèses et séparés par des virgules, dont chacun consiste d'un type et d'un identificateur (la même syntaxe que celle d'une déclaration de variable).

L'entête est suivi du *corps* de la fonction, qui est un bloc (une suite de déclarations et d'instructions), et se termine par l'instruction `return`.

Appel de fonction

```
int a = 3, b;
b = f(a + 1, 4);
```

Une *invocation de fonction* est une expression, elle peut par exemple être utilisée du côté droit d'une affectation. Elle consiste du nom de la fonction (ici « `f` ») suivi d'une liste de *paramètres effectifs* entre parenthèses. Chaque paramètre effectif est une expression ayant le type du paramètre formel correspondant; l'invocation elle-même a le type de la valeur de retour de la fonction.

2 Sémantique

```
b = f(a + 1, 4);
```

Pour appeler la fonction, on commence par évaluer chacun des paramètres effectifs. On sauvegarde ensuite la *continuation* de l'appel, c'est à dire ce qui reste à faire après l'appel à la fonction, sur la pile d'appels (voir ci-dessous); ici, la continuation consiste à affecter le résultat à `b` puis à exécuter la suite du programme. Enfin, on affecte chacun des paramètres formels la valeur du paramètre formel correspondant, et on exécute le corps de la fonction.

Lorsque l'instruction `return` est atteinte, la valeur du paramètre de `return` devient la valeur de l'appel de fonction. On restaure la continuation depuis la pile, et on revient à l'exécution de cette dernière.

Pile d'appels Il est possible qu'une fonction appelle elle-même une fonction. Dans ce cas, les continuations sont sauvegardées l'une après l'autre, et restaurées dans l'ordre inverse de leur sauvegarde; la suite des continuations constitue donc une *pile*. On l'appelle la *pile d'appels*.

Appel par valeur À la différence de certains autres langages de programmation, C a une sémantique d'*appel par valeur* : c'est la *valeur* de chaque paramètre formel qui est passée au corps de la fonction, et la valeur du paramètre de `return` qui est retournée. Il est donc impossible de modifier la valeur d'un paramètre effectif depuis une fonction : si l'on modifie la valeur d'un paramètre formel, la nouvelle valeur est perdue lors du retour de la fonction.

3 Le type void

Parfois, une fonction n'a pas de valeur à retourner. Le type `void`, utilisé comme valeur de retour, indique que la fonction ne retourne aucune valeur. Le corps d'une fonction retournant `void` peut soit se terminer sans l'instruction `return`, soit par un `return` sans paramètres.

4 Déclaration de fonction

Si l'appel à une fonction suit sa définition, alors il n'est pas nécessaire de la déclarer — la définition sert alors de déclaration. Si la définition suit l'appel, il faut déclarer la fonction avant de s'en servir.

Une *déclaration de fonction*, ou *prototype*, consiste d'un entête de fonction suivi d'un point-virgule. Par exemple, la fonction `f` ci-dessus aurait pu être déclarée par :

```
int f(int x, int y);
```

La directive `#include` sert à inclure textuellement un fichier dans un autre. Elle est typiquement utilisée pour inclure des *fichiers d'entêtes* tels que `stdio.h` qui contiennent des définitions de types et des déclarations de fonctions.