Cours 3. Fonctions en langage C

Dimitri Galayko

1 Renseignements sur les fonctions

Une fonction est une unité de programme qui comporte un code exécutant une tâche. Les fonctions permettent *une programmation structurée*: une décomposition d'un programme complexe en un nombre d'unités (fonctions, modules) plus simples, et donc, facilement compréhensibles et maîtrisable par un humain.

1.1 La définition de la fonction: le syntaxe

Une fonction est un objet créé par le programmeur. Une fonction est spécifiée par les paramètres suivants:

– L'identificateur (le nom) de la fonction, choisi par le programmeur, dans la limite des règles qui s'appliquent aux identificateur en langage C (cf. le cours 1).

– Le type de résultat retournée par la fonction, spécifiée *avant* l'identificateur de la fonction,

– Le nombre et le type des arguments (les paramètres d'entrée), spécifiées par une liste insluse entre paranthèses et où les éléments sont séparés par des virgules.

– Le corps de la fonction: le code enfermé entre les accolades (obligatoires), juste après la liste des arguments.

Le syntaxe permettant de définir une fonction est le suivant:

```
<type de résultat> nom_de_la_fonction(type1 arg1, type2 arg2, ...){
<ici, le code de la fonction>
// les accolades délimitant le code sont obligatoires
}
```

Notez que les accolades délimitant le code de la fonction sont obligatoires, même si la fonction est composée d'une seule instruction.

Le type du résultat et les types des arguments sont choisis par le programmeur parmis les types disponibles dans le langage C (cf. le cours 1).

1.2 La généricité

Non seulement une fonction est une unité de code, mais en plus, c'est une unité de code *à paramètres*. Les paramètres d'une fonction s'appellent *arguments*. Leur nombre et leur type sont définis par le programmeur.

Exemple: lorsque l'on écrit un programme calculant un sinus *sin(x)*, on aimera la reutiliser pour différentes valeurs de l'arguent x. On mettra le programme dans une fonction que l'on appellera *sin* (par exemple), et on définira un argument de la fonction pour donner la valeur de x. Ainsi, on écrit un code qui est valable non pas pour un seul cas (une seule valeur de x), mais pour *une plage des valeur*. On dit, que l'on a définit une fonction "générique". La généricité est la caractéristique principale des programmes informatiques.

Un autre exemple: on peut écrire une fonction qui dessine une ligne droite à l'écran, de position et de taille définie. Mais on peut définir la longueur et la position de la ligne comme paramètres: ainsi, on aura une *famille* de fonctions qui dessinent *n'importe quelle ligne droite*.

Un troisième exemple: la fonction qui calcule la factorielle d'un nombre.

```
int factorielle(int x){
        // ici on écrit le code pour calculer la factorielle de x
        // on suppose que x est défini
```

```
        int i,f;
        f=1;
        for(i=1;i<=x; i=i+1)
                f=f*i;
        // on retourne le résultat, cf. le paragraphe suivant
        return f;
}
```

1.3 Le résultat de la fonction

Les fonctions en C sont capables de retourner une valeur, et une seule. Le type de la valeur est choisi par le programmeur, parmis les types disponible en langage C. Une fonction ne peut pas retourner des tableaux.

Le retour de la valeur se fait avec l'instruction *return* (cf. la listing de la factorielle plus haut). Le mot clé *return* est suivi de l'expression donnant la valeur retournée. Cette valeur doit être du même type que celui donnée en première position devant l'identificateur (nom de la fonction).

Parfois on ne souhaite pas que la fonction retourne un résultat (par exemple, lorsque l'on veut que la fonction dessine ou affiche des informations à l'écran). Dans ce cas là, on utiliser le mot clé *void* pour le type de retour.

Exemple: dessin d'une ligne entre les points (x0,y0) et (x1,y1):

```
void dessiner_ligne(int x0, int y0, int x1, int y1){
        // ici on écrit le code pour dessiner une ligne
        // ....
        // Quand on a fini, on écrit le mot clé return suivi de point virgule:
        return;
}
```

Lorsque l'instruction *return* est rencontrée, la fonction s'arrête (cf. le paragraphe suivant).

1.4 Utilisation de la fonction

Pour utiliser la fonction, il faut *l'appeller* à partir d'un code d'une autre fonction – qui peut être la fonction *main*. Par exemple, pour afficher une ligne entre les points (1,1) et (100,100), on écrira:

```
...
dessiner_ligne(1, 1, 100, 100);
...
```

Ainsi, une fonction peut être vue comme une instruction que le programmeur a créé et qui fait maintenant partie de son environnement de programmation. Il peut "oublier" les détails de sa réalisation, et l'utiliser comme un instrument pour faire des programmes plus complexes. Une autre personne peut également l'utiliser.

Si une fonction retourne une valeur arithmétique, elle peut être utilisée dans une expression. Exemples:

```
...

// ici la variable a prendra la valeur de l'expression 2*5!
a=2*factorielle(5);
// ici la fonction printf affichera la valeur de 10!
printf("La factorielle de 10 est %d\n", factorielle(10));
...
```

La fonction à partir de laquelle une autre fonction est appelée s'appelle "l'appelant, ou la fonction appelante". On dit "la fonction appelée retourne un résultat à la fonction appelante". Le retour vers la fonction appelante se fait à la commande *return*.

2 Exemples des fonctions

2.1 Fonction "Lecture d'une valeur entière au clavier".

On souhaite écrire une fonction qui demande l'utilisateur de saisir une valeur entière au clavier, dans un intervalle donné [a,b]. Si l'utilisateur se trompe, un message d'erreur est généré, et le programme redemande de saisir une valeur correcte.

```
// la fonction lecture_valeur retournera la valeur saisie au clavier, dans l'
    intervalle [a,b] limites incluses.

int lecture_valeur(int a, int b){
        int x; //la valeur saisie
        printf("Saisir une valeur entière entre %d et %d:", a, b);
        do{
                scanf("%d", x);
                if ((x>=a) && (x<=b))
                        return x;
                else
                        printf("La valeur saisie est incorrecte. Merci de saisir de
                            nouveau:");
        }while(1); // une boucle infinie
                        // le programme ne quittera la boucle que si
                        // une valeur correcte est saisie.
                        // Le retour se fera avec la ligne return.
}
```

Ainsi, dans le programme main, il suffira d'écrire :

```
int main(){
int g;
....
g=lecture_valeur(5,20);
....
```

La variable *g* prendra la valeur que l'utilisateur saisira au clavier. Pour la personne qui écrit le programme main(), la lecture de la valeur sera transparente.

2.2 Lecture d'une suite de nombres au clavier et calcul de la moyenne

On souhaite écrire un programme qui:

– lit des valeurs entières, dans l'intervalle [a,b]. On suppose que l'intervalle proposé ne comprend que des nombres non-négatives.

– L'utilisateur saisit une valeur après l'autre. Il saisit -1 pour signifier "fin de la liste".

– La fonction calcule la moyenne des valeurs saisies (en excluant -1, bien sûr).

On écrit deux fonction: l'une pour saisir les valeurs, l'autre pour faire le traintement principal. On ne peut pas reutiliser directement la fonction "lecture_valeur", car celle-ci ne permet pas la saisie de -1 qui est en déhors de l'intervalle. On la modifie de la manière suivante:

```
// la fonction lecture_valeur retournera la valeur saisie au clavier, dans l'
    intervalle [a,b] positif limites incluses, -1 est admis comme valeur correcte.

int lecture_valeur_positive(int a, int b){
        int x; //la valeur saisie
        printf("Saisir une valeur entière entre %d et %d:", a, b);
        do{
                scanf("%d", &x);
                if (((x>=a) && (x<=b)) || (x==-1)) // on ajoute ici une condition de
                    validite x==-1
                        return x;
```

```
                else
                        printf("La valeur saisie est incorrecte. Merci de saisir de
                            nouveau:");
        }while(1); // une boucle infinie
                        // le programme ne quittera la boucle que si
                        // une valeur correcte est saisie.
                        // Le retour se fera avec la ligne return.
}
```

On écrit maintenant la fonction qui lit une liste de valeurs non-négatives dans l'intervalle [a,b] et qui calcule la moyenne des valeurs lues:

```
double moyenne_liste(int a, int b){
        int nombre, somme, x;
        double moyenne;
        somme=0; // on initialise la somme
        nombre=0; // on initialise le compteur des valeurs saisie
        do{
                x=lecture_valeur_positive(a,b);
                if (x!=-1){
                        nombre=nombre+1;
                        somme=somme+x;
                }
        }while(x!=-1);
        if (nombre!=0)
                moyenne=(double)somme/nombre;
        else
                moyenne=-1; // si la moyenne est négative, aucun nomre n'a été saisi
                                // car autrement la moyenne est positive ou nulle
        return moyenne;
}
```