Initiation aux fonctions

Il arrive souvent que l'on ait besoin au cours d'un programme de répéter plusieurs fois un ensemble d'actions (affichages répétitifs, formule à calculer fréquemment, …). Si ces actions sont programmées à l'intérieur du **main** , cela présente les inconvénients suivants :

- lourdeur et volume du fichier-source
- risque d'erreurs lors des recopies multiples
- nécessité en cas de modifications d'une action, de les reporter partout, avec là-aussi un risque d'erreurs

Une fonction permet de ne programmer qu'à un seul endroit une action utilisée à plusieurs endroits. Le code C de la fonction est alors en-dehors du **main** (et de préférence avant d'être appelée pour la 1ère fois) :

```
#include …


type fonction1( … )
{
        …
}

…

void main()
{
        …
}
```

Remarque Le **main** n'est lui-même rien d'autre qu'une fonction : la fonction « principale ».

Structure

void ***nom_fonction*(*liste_des_paramètres*)**
{
zone de déclarations des variables locales

bloc(s) d'instructions contenant 1 ou plusieurs fois : **return ;**
}

ou

***type_fonction nom_fonction*(*liste_des_paramètres*)**
{
zone de déclarations des variables locales

bloc(s) d'instructions contenant au moins une fois : **return(*objet_renvoyé*) ;**
}

avec

return(*objet_renvoyé*) ; est l'instruction permettant à la fonction ***nom_fonction*** d'interrompre son déroulement et de renvoyer le résultat de son action au **main** (ou à une autre fonction) l'ayant appelé. Le retour se fait à l'endroit où la fonction a été appelée.
objet_renvoyé est du type ***type_fonction*** .
Attention : si ***type_fonction*** n'est pas précisé, le compilateur suppose que la fonction renvoie une valeur de type **int** !! (==> **A éviter absolument** : ***type_fonction*** doit toujours être présent !)

Remarque une fonction de type **void** ne renvoie aucun résultat : **return** seul interrompt la fonction et retourne à l'endroit où elle a été appelée.

liste_des_paramètres décrit les paramètres que l'on doit « passer » à la fonction lors de son « appel » séparés par des virgules s'il y en a plusieurs. Cette liste est de la forme :

***type_param1 nom_param1* , *type_param2 nom_param2* , …**

type_param étant un type C et ***nom_param*** étant le nom du paramètre utilisé à l'intérieur de la fonction (Attention : il peut être différent de celui utilisé à l'endroit de l'appel !)

liste_des_paramètres peut être vide si la fonction n'a pas de paramètre (les parenthèses sont néanmoins obligatoires)

L'appel de cette fonction (dans le **main** ou dans une autre fonction) peut être de la forme :

…
***variable_réceptacle = nom_fonction*(*paramètres*) ;**
…
si l'on veut récupérer le résultat dans une variable (de type ***type_fonction***)

ou

***…nom_fonction*(*paramètres*)…**
l'appel de la fonction apparaissant au sein d'une expression quelconque utilisant son résultat

ou

***nom_fonction*(*paramètres*) ;**
si la fonction ne renvoie aucun objet

Exemple

```
#include <stdio.h>

void affiche_etoiles(int nb_etoiles)
{
/* Cette fonction affiche une ligne de nb_etoiles etoiles */

        int i ;

        for (i=1 ; i<=nb_etoiles ; i++)
        {
                printf("*");
        }
        return;
}

float f1(float x,char aff)
{
/* Cette fonction renvoie x2 – 3x + 9 , mais l'affiche auparavant si aff
vaut 'o' */

        int resul;

        resul=x*x-3*x+9;
        if(aff == 'o')
        {
                printf("Resultat = %g\n",resul);
        }

        return(resul);
}

void main()     /* En fait, main est lui-meme une fonction, souvent, comme ici,
                sans parametre */
{

        float t4;
        char oui='o' , non='n' ;

        /* Affichage d'une ligne de 15 etoiles */
        affiche_etoiles(15) ;

        /* Calcul de f1(4) avec affichage dans la fonction */
        t4=f1(4 , oui) ;  /* f1 est appelée et son résultat est stocké dans t4 */

        /* Affichage de 8*f1(0.5) sans affichage dans la fonction */
        printf("8f1(0.5) = %g\n", 8.0 * f1(0.5 , non)) ;  /* f1 est appelée au
                                                            sein d'une expression */

}
```

Variables locales et globales

Les variables déclarées au sein d'une fonction ne sont connues que de cette fonction et non à l'extérieur (même si, à l'extérieur, une variable de même nom existe : ce sont alors 2 variables différentes ➔ *à éviter donc car source de confusions*). On dit que ces variables sont locales à la fonction.

De même, les variables déclarées dans le **main** sont locales au **main** .

Pour qu'une variable soit connue du **main** et de toutes les fonctions d'un même fichier-source, il faut les déclarer avant le **main** et toutes les fonctions : ce sont alors des variables globales .

Modification des paramètres

Une fonction peut-elle modifier les paramètres avec lesquels elle est appelée ?

Oui , mais pour ce faire, elle doit disposer des adresses des paramètres qu'elle désire modifier : ces adresses doivent donc lui être fournies lors de l'appel.

Si la fonction est appelée avec l'adresse d'un paramètre, il s'agit d'un passage de paramètre(s) « par adresse ».
Sinon, c'est un passage de paramètre(s) « par valeur » : toute modification du paramètre est alors impossible.

Exemple :

```
#include <stdio.h>

int g=2;        /* Cette variable est globale et donc connue de tout le
                   monde */

int f2(int k , float tab[])
{
/* Cette fonction pourra modifier tab mais non k */

        int z ;  /* Cette variable est locale à la fonction f2 et inconnue
                    ailleurs */

        …
        k=7 ;
        g=4 ; /* Inutile de faire figurer g parmi les parametres : elle
        est globale, donc connue de f2 */

        tab[1]=2.3 ;
        …

}

void main()
{
        float tt[10] ;   /* Ces variables sont locales au main */
        int i , j=5 ;

        tt[0]=98.123; tt[1]=65.102;

i = f2(j , tt);  /* On transmet a f2, la valeur de j et l'adresse tt du
                    tableau (i.e. celle du 1er element tt[0]) */
/* On est oblige de transmettre j et tt en parametres, car elles sont
locales au main et donc inconnues de f2 */

/* Apres l'appel de f2, g contient 4, tt[1] contient 2.3, mais j
contient toujours 5 (bien qu'on ait essayé de lui affecter 7 dans f2) */


}
```

Finalement : pour qu'une fonction connaisse une variable, celle-ci doit être globale, ou, si elle est locale à la fonction appelante, elle doit être passée en paramètre ; si on passe l'adresse de la variable, celle-ci pourra être modifiée par la fonction, sinon non.