Chapitre 6 : Les fonctions

I. Introduction

Jusque là nous n'avons vu qu'une fonction dans nos programmes. Il s'agissait de la fonction *main* présente dans tous les programmes C. Depuis le début, tous nos programmes ne font que quelques dizaines de lignes et donc on pouvait écrire l'intégralité de notre programme dans la fonction *main*.
Le but principal d'une fonction est de retourner un résultat après avoir exécutée plusieurs instructions qui lui sont associées. Par exemple une fonction peut s'occuper de renvoyer la somme ou la différence de deux nombres, ...
Pour qu'une fonction puisse s'exécuter convenablement :

- il faut qu'elle reçoive, ce que l'on appel des arguments (ou paramètres),
- une fois ces paramètres reçues la fonction s'exécute et renvoie un résultat final.

Par exemple *printf* est une fonction à laquelle on envoie des arguments pour la faire fonctionner, on appel cela **un appel de fonction**. Donc pour utiliser une fonction il faut l'appeler dans le programme.

II. Définition d'une fonction

II.1 Structure générale d'une fonction (Code c)

```
type nom(type nom_parametres)
{
   Instructions
}
```

Pour définir une fonction, il faut préciser plusieurs éléments :

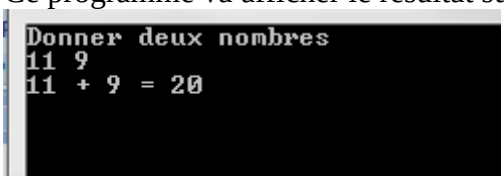
- **Le type de la fonction :** C'est le type du résultat que la fonction va renvoyer à la fin de son exécution (int, long, float, double, ...)
- **Son nom :** pour pouvoir l'appeler dans un programme il faut au préalable lui donner un nom par exemple : addition, soustraction…(le nom donné a la fonction doit respecter les mêmes règles que lors de la déclaration d'une variable).
- **Les paramètres** : ce sont par exemple les nombres que vous allez envoyer à la fonction. Il faut préciser le type de ces paramètres.
- Le but principal d'une fonction est de retourner une valeur, ceci se fait dans une fonction à l'aide du mot-clé ***return***.
- Comme dans notre fonction *main* les instructions à l'intérieur d'une fonction doivent **impérativement** se trouver entre accolades.

Exemple : fonction qui calcule l'addition de deux nombres.

```
#include<stdio.h>
#include<conio.h>
int addition (int nb1, int nb2)
 {
    return nb1 + nb2;
 }
int main()
```

```
{
   int nb1,nb2, resultat;
   printf("Donner deux nombres \n");
   scanf("%d%d",&nb1,&nb2);
   resultat = addition (nb1, nb2);  /*appel de la function addition*/
   /* la valeur de la variable resultat prend la valeur renvoyée par la fonction */
   printf("%d + %d = %d",nb1, nb2, resultat);
getch() ;
   return 0;
}
```

Ce programme va afficher le résultat suivant:



Remarque

On peut aussi placer une fonction après la fonction main. Dans ce cas il suffit de reprendre, exactement, la déclaration de votre fonction et d'y ajouter un point-virgule avant la fonction main.

Exemple :

```
#include<stdio.h>
#include<conio.h>
int addition (int nb1, int nb2);  /*déclaration de la fonction */
int main()
{
   int nb1,nb2,resultat;
   printf("Donner deux nombres \n");
   scanf("%d%d",&nb1,&nb2);
   resultat = addition (nb1, nb2);
   /* la valeur de la variable resultat prend la valeur renvoyée par la fonction */

   printf("%d + %d = %d",nb1, nb2, resultat);
getch() ;
   return 0;
}
int addition (int nb1, int nb2)
{
     return nb1 + nb2;
}
```

II.2 Variables locales et globales.

On distingue deux types de variables utilisées par les fonctions :

- Les variables locales
- Les variables globales

Les variables locales

Une variable déclarée dans une fonction n'est accessible qu'à l'intérieur de cette fonction. On dit que c'est une variable locale. Lorsque vous déclarez une variable dans une fonction, celle-ci est supprimée de la mémoire à la fin de la fonction.

Exemple :

```
int triple(int nombre)
{
    int resultat = 0; // La variable resultat est créée en mémoire
     resultat = 3 * nombre;
    return resultat;
} // La fonction est terminée, la variable resultat est supprimée de la mémoire
```

Une variable déclarée dans une fonction n'existe donc que pendant que la fonction est exécutée. Ceci veut dire que vous ne pouvez pas utiliser cette variable dans une autre fonction

Exemple :

```
#include <stdio.h>
#include <conio.h>
 int triple(int nombre)
  {
    int resultat = 0;
    resultat = 3 * nombre;
    return resultat;
  }
int main()
{
    printf("Le triple de 15 est %d\n", triple(15));
    printf("Le triple de 15 est %d", resultat); // Erreur
 getch();
    return 0;
}
```

Dans ce cas, on ‘a essayer d'accéder à la variable ‘resultat’ dans la fonction *main*, mais comme cette variable ‘resultat’ a été déclarée dans la fonction triple on ne peut pas l’utiliser dans la fonction main.

Les variables globales

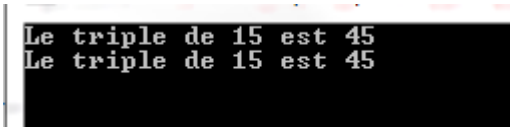
Une variable globale est une variable qui est accessible dans tous les fichiers. Il est possible de déclarer des variables qui seront accessibles dans toutes les fonctions de tous les fichiers du projet (ce qu’il faut éviter de faire). Pour déclarer une variable « globale » accessible partout, il faut la déclaration en dehors des fonctions. Généralement la déclaration se fait tout en haut après les #include.

Exemple :

```
#include <stdio.h>
#include <conio.h>
   int resultat = 0;
```

```
  int triple(int nombre)
   {
     resultat = 3 * nombre;
    return resultat;
   }
int main()
{
   printf("Le triple de 15 est %d\n", triple(15));
   printf("Le triple de 15 est %d", resultat);
 getch();
   return 0;
}
```

Ce programme va afficher le résultat suivant:



Dans cet exemple, la variable résultat est déclaré comme une variable globale et donc la fonction triple ainsi que la fonction main peuvent accéder à la variable resultat.

II.3 Passage des paramètres par valeur

Les paramètres d'une fonction sont simplement des variables locales qui sont initialisées par les valeurs obtenues lors de l'appel de la fonction. En C, le passage des paramètres se fait par valeur, c'est-à-dire que les fonctions n'obtiennent que les valeurs de leurs paramètres et n'ont pas la possibilité d'accéder aux variables elles-mêmes.

A l'intérieur de la fonction, nous pouvons donc changer les valeurs des paramètres sans influencer les valeurs originales dans les fonctions appelantes.

Exemple

On considère le programme suivant :

```
#include <stdio.h>
#include <conio.h>
void ETOILES(int N)
{
 while (N>0)
   {
     printf("*");
     N--;
   }
 printf("\n");
}
int main()
{
 int L;
 for (L=1; L<10; L++)
     ETOILES(L);
getch();
   return 0;
}
```

Dans ce programme on’ a :

- La fonction ETOILES dessine une ligne de N étoiles. Le paramètre N est modifié à l'intérieur de la fonction ETOILES.
- La fonction main, appelle la fonction ETOILES en utilisant la variable L comme paramètre. Au moment de l'appel, la valeur de L est copiée dans N.
- La variable N peut donc être décrémentée à l'intérieur de la fonction ETOILES, sans influencer la valeur originale de L.
- Schématiquement, le passage des paramètres (L et N) entre la fonction main et la fonction ETOILES peut être représenté par la trace suivante :

Main		ETOILES	
L			N
1	appel →		1
	← retour		0
2	appel →		2
	← retour		1
			0
3	appel →		3
	← retour		2
			1
			0
….			….

Le passage par valeur a l'avantage que nous pouvons utiliser les paramètres comme des variables locales bien initialisées. De cette façon, nous avons besoin de moins de variables d'aide.

II.4 fonctions sans paramètres

Premier cas

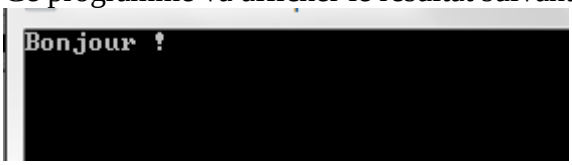
Il est possible en C de créer des fonctions sans paramètres ou ne renvoyant aucune valeur. Pour cela on utilise un type prévu à cet effet : ***void***

On utilise ce type pour des fonctions censées affichées uniquement des messages par exemple.

Exemple :

```
#include<stdio.h>
#include<conio.h>
void  afficher (void) /* pas de paramètres donc void */
{
   printf("Bonjour !\n");
}
int main ()
{
   afficher ();
    getch() ;
   return 0;
}
```

Ce programme va afficher le résultat suivant:



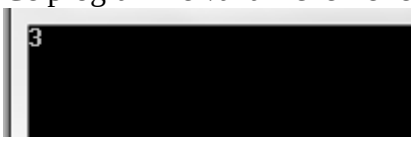
Remarque : on peut ne pas mettre *void* dans les parenthèses d'envoie des paramètres et donc on peut écrire afficher() au lieu de afficher(void).

Deuxième cas.
On peut aussi avoir une fonction qui renvoie un résultat de type *int* par exemple mais ne prenant aucun paramètres.

Exemple :

```
#include<stdio.h>
#include<conio.h>
int afficher ( ) /* pas de paramètres  */
  {
    return 3;
  }
int main ()
{
  printf("%d", afficher ()) ;
    getch() ;
    return 0;
}
```

Ce programme va afficher le résultat suivant:



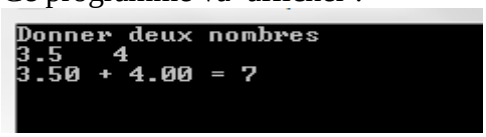
II.5 conversion de types
Il faut faire attention avec les types de fonctions, si on envoie comme paramètres à une fonction de type int, des variables de type float, le résultat que vous recevez serait de type int. Le résultat de la fonction est converti du type float au type int.

Exemple :

```
#include<stdio.h>
#include<conio.h>
int addition (float nb1, float nb2)
  {
     return nb1 + nb2;
  }
int main()
{
   float nb1,nb2;
   printf("Donner deux nombres \n");
```

```
   scanf("%f%f",&nb1,&nb2);
   printf("%.2f + %.2f = %d",nb1, nb2, addition(nb1, nb2));
getch() ;
   return 0;
}
```

Ce programme va afficher :



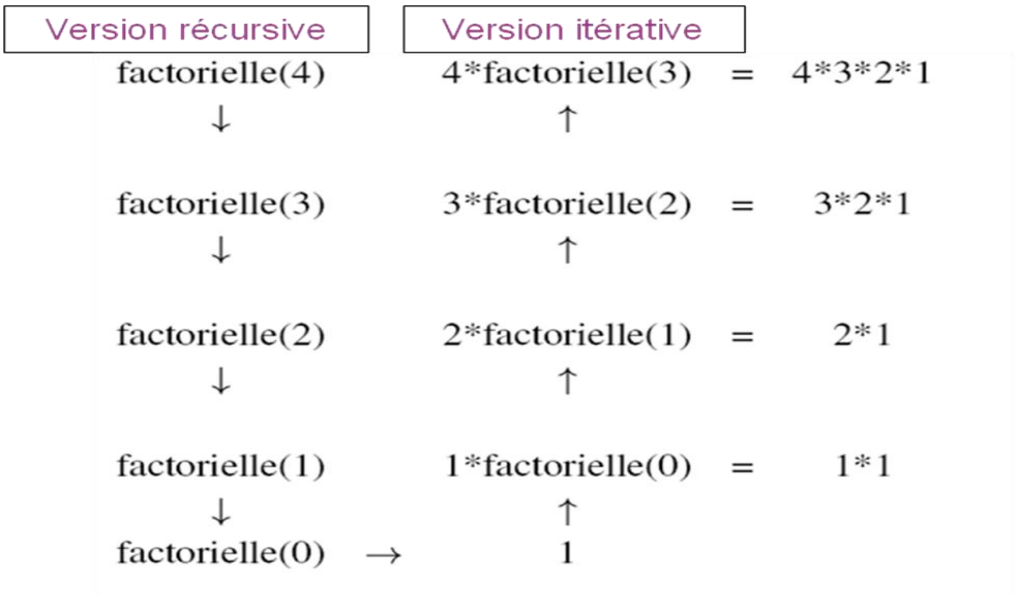
II.5 Fonction récursive

Le langage C permet d'écrire des fonctions récursives. Une fonction récursive est une fonction qui fait appel à elle-même.

Exemple

```
#include <stdio.h>
#include <conio.h>
 //Calcul du factoriel avec une fonction iterative (avec une boucle for).
 int factoriel_iterative (int n)
  {
     int i, fact = 1;
       if (n == 1 || n == 0)
          return 1;
          for (i = 1; i <= n; i++)
                  fact *= i;
                  return fact;
  }
 //  Calcul du factoriel avec une fonction recursive (la fonction appel elle même).
 int factoriel_recursive (int n)
   {
    if (n == 1 || n == 0)
      return 1;
      return n*factoriel_recursive (n - 1);
   }
int main (void)
{
       int n,fact1, fact2;
 //Fonction iterative.
   printf("donner un entier\t");
   scanf("%d",&n);
   fact1 = factoriel_iterative (n);
   printf ("la factoriel (iterative) de %d = %ld\n",n,fact1);
 // Recursivite simple.
    fact2 = factoriel_recursive(n);
    printf("la factoriel (recurssive) de %d =%d\n",n,fact2);
 getch();
}
```

Liste des appels de la fonction factoriel (cas du n=4)



Récursivité : observations

- Une fonction est définie récursivement lorsque la valeur de la fonction en un point x est définie par rapport à sa valeur en un point strictement « plus petit »
- De ce fait, le calcul se fait de proche en proche, jusqu'à atteindre le plus petit élément pour lequel il faut une valeur immédiate (c'est-à-dire non récursive)
- Le corps d'une fonction récursive doit toujours exprimer un choix, soit par une expression conditionnelle, soit par une définition par cas
- Au moins un cas terminal rend une valeur qui n'utilise pas la fonction définie.