



CLUBSDAD

**Faculté des Sciences et Techniques Guéliz
Université Cadi Ayad,
Marrakech**

**Tutoriel de -C-
Semaine 02**

Les fonctions en langage C

Réalisé par : Z. ELMOURABIT ; K. NAIM ; I. ASAKOUR ; S. BASKAR

L'objectif :

Application des fonctions de base en langage C aux buts de comment faire la déclaration et l'appel des fonctions, et apprendre de travailler avec les fonctions récursives et itératives.

Prêts ?

On Commence !...

C'est quoi une fonction en C ?

Comme dans la plupart des langages, on peut en C découper un programme en plusieurs fonctions. Une seule de ces fonctions existe obligatoirement ; c'est la fonction principale appelée `main`. Cette fonction principale peut, éventuellement, appeler une ou plusieurs fonctions secondaires. De même, chaque fonction secondaire peut appeler d'autres fonctions secondaires ou s'appeler elle-même (dans ce dernier cas, on dit que la fonction est *réursive*).

Définition d'une fonction

La définition d'une fonction est la donnée du texte de son algorithme, qu'on appelle corps de la fonction. Elle est de la forme

```
type nom-fonction (type-1 arg-1, ..., type-n arg-n)
{[déclarations de variables locales ]
  liste d'instructions
}
```

La première ligne de cette définition est l'en-tête de la fonction. Dans cet en-tête, `type` désigne le type de la fonction, c'est-à-dire le type de la valeur qu'elle retourne. Contrairement à d'autres langages, il n'y a pas en C de notion de procédure ou de sous-programme. Une fonction qui ne renvoie pas de valeur est une

fonction dont le type est spécifié par le mot-clef `void`. Les arguments de la fonction sont appelés paramètres formels, par opposition aux paramètres effectifs qui sont les paramètres avec lesquels la fonction est effectivement appelée. Les paramètres formels peuvent être de n'importe quel type. Leurs identificateurs n'ont d'importance qu'à l'intérieur de la fonction. Enfin, si la fonction ne possède pas de paramètres, on remplace la liste de paramètres formels par le mot-clef `void`.

Le corps de la fonction débute éventuellement par des déclarations de variables, qui sont locales à cette fonction. Il se termine par l'instruction de retour à la fonction appelante, `return`, dont la syntaxe est

```
return(expression);
```

La valeur d'expression est la valeur que retourne la fonction. Son type doit être le même que celui qui a été spécifié dans l'entête de la fonction. Si la fonction ne retourne pas de valeur (fonction de type `void`), sa définition s'achève par

```
return;
```

Plusieurs instructions `return` peuvent apparaître dans une fonction. Le retour au programme appelant sera alors provoqué par le premier `return` rencontré lors de l'exécution. Voici quelques exemples de définitions de fonctions :

```
int produit (int a, int b)
{
    return(a*b);
}
int puissance (int a, int n)
{
    if (n == 0)
        return(1);
    return(a * puissance(a, n-1));
}
void imprime_tab (int *tab, int nb_elements)
{
    int i;
    for (i = 0; i < nb_elements; i++)
        printf("%d \t",tab[i]);
    printf("\n");
    return;
}
```

Appel d'une fonction

L'appel d'une fonction se fait par l'expression :

```
Appel d'une fonction

nom-fonction(para-1,para-2,...,para-n)
```


Declaration d'une fonction

```
type nom-fonction(type-1, ..., type-n);
```

Ex00 :

Ecrire une fonction qui lit de 2 variables a et b et qui affiche le plus petit le plus grand la somme et la moyenne de 2 nombres

Elle devra être prototypée de la façon suivante :



```
min_max  
  
void min_max (int n1, int n2);
```

Ex01 :

Ecrire une fonction qui prend en paramètre un nombre compris entre 1 et 12, et affiche le mois qui convient


Elle devra être prototypée de la façon suivante :

Avec exemple :

`ft_mois(1)` affiche janvier

`ft_mois(2)` affiche février

Elle devra être prototypée de la façon suivante :



```
Les mois  
  
void ft_mois(int n1);
```

EX02 :

Ecrire une fonction qui retourne la valeur absolue d'un nombre saisi par l'utilisateur dans le programme principal.

Elle devra être prototypée de la façon suivante :



```
int ft_absolue(int n1);
```

EX03 :

Ecrire une fonction qui calcul la soustraction $n1 - n2$ sans utiliser l'opérateur '-'

Elle devra être prototypée de la façon suivante :




```
void ft_Soustraire(int n1, int n2);
```


EX04 :

Écrire une fonction itérative qui renvoie un nombre. Ce nombre est le résultat de l'opération factorielle à partir du nombre passé en paramètre.

En cas d'erreur, la fonction devra retourner 0.

Elle devra être prototypée de la façon suivante :

A code editor window with a dark background and a purple-to-blue gradient border. The title bar contains three dots and the text 'iterative_factorial'. The code is:


```
void ft_iterative_factorial(int nb);
```

```
iterative_factorial  
  
void ft_iterative_factorial(int nb);
```

EX05 :

Ecrire une fonction qui calcul le factoriel d'une manière récursive

Elle devra être prototypée de la façon suivante :

A code editor window with a dark background and a purple-to-blue gradient border. The title bar contains three dots and the text 'recursive_factorial'. The code is:

```
int ft_recursive_factorial(int nb);
```

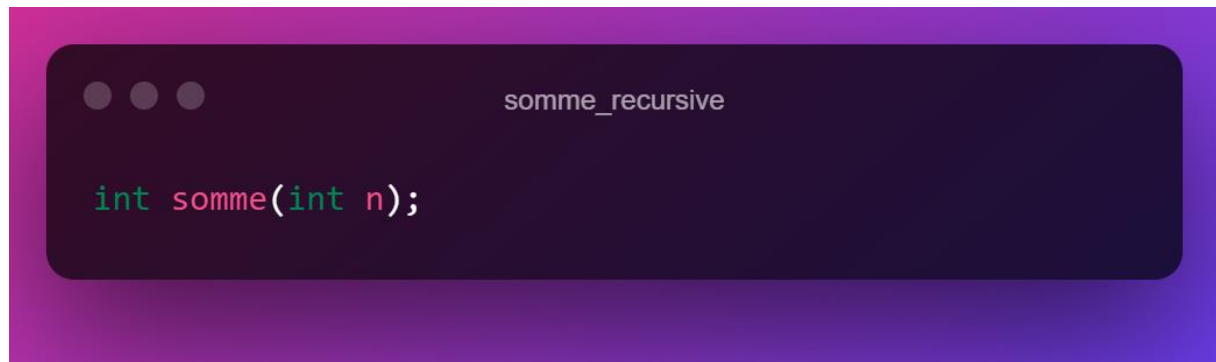
```
recursive_factorial  
  
int ft_recursive_factorial(int nb);
```

EX06 :

Ecrire une fonction qui renvoie la somme de tous les nombres inférieur ou égale à n d'une manière récursive

Exemple : $n = 6$, La somme est $6+5+4+3+2+1$

Elle devra être prototypée de la façon suivante :

A screenshot of a code editor with a dark background and purple accents. The title bar reads "somme_recursive". The code shown is a function prototype:

```
int somme(int n);
```

EX07 :

Écrire une fonction itérative qui renvoie une puissance d'un nombre. Une puissance < 0 renverra 0. Les overflows ne devront pas être gérés.

Elle devra être prototypée de la façon suivante :

A screenshot of a code editor with a dark background and purple accents. The title bar reads "iterative_power". The code shown is a function prototype:

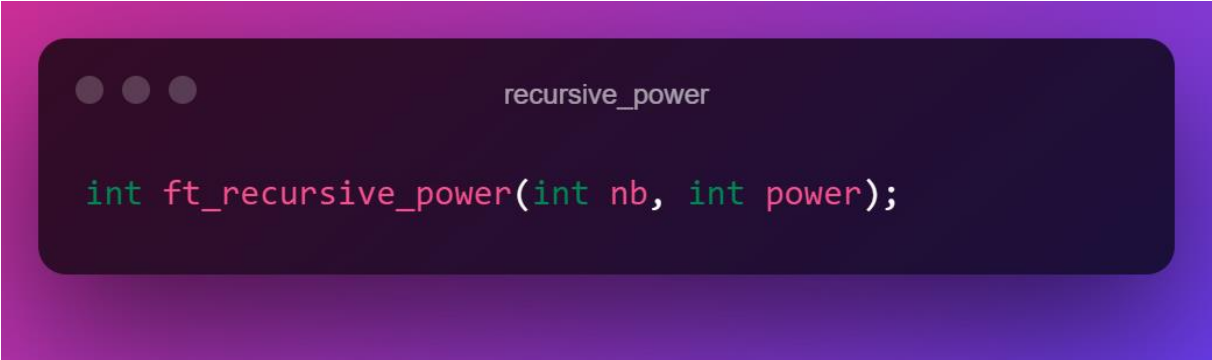
```
int ft_iterative_power(int nb, int power);
```

EX08 :

Ecrire une fonction qui calcul la puissance récursive

Elle doit gérer les mêmes cas que la fonction précédente.

Elle devra être prototypée de la façon suivante :



```
recursive_power

int ft_recursive_power(int nb, int power);
```

EX09 :

Écrire une fonction `ft_fibonacci` qui renvoie le n-ième élément de la suite de Fibonacci, le premier élément étant à l'index 0.

Nous considérerons que la suite de Fibonacci commence par 0, 1, 1, 2.

Elle devra être prototypée de la façon suivante :



```
Fibonacci

int ft_fibonacci(int index);
```

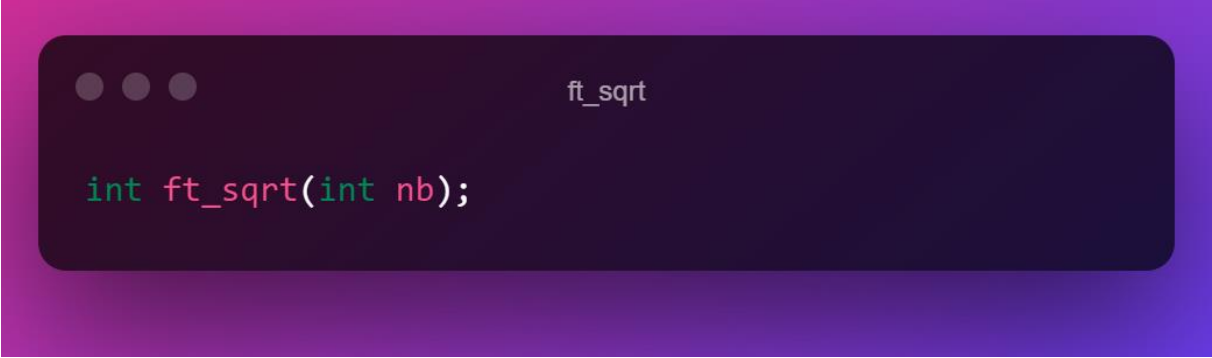
- Evidement, `ft_fibonacci` devra être récursive.
- Si `index` est négatif, la fonction renverra -1.

EX10 :

Écrire une fonction qui renvoie la racine carrée entière d'un nombre si elle existe, 0

Si la racine carrée n'est pas entière.

Elle devra être prototypée de la façon suivante :

A code editor window with a dark background and a purple-to-blue gradient border. The window title is "ft_sqrt". It contains the following C code:

```
int ft_sqrt(int nb);
```

```
ft_sqrt

int ft_sqrt(int nb);
```

Votre fonction doit donner son résultat en moins de deux secondes.