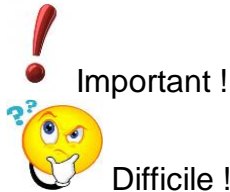


# Programmation en C/C++

## Série 5

### Fonctions récursives



#### Objectifs

Maîtriser l'écriture de fonctions récursives.

#### Introduction

Pour comprendre ce chapitre il est nécessaire de bien maîtriser les fonctions en langage C.

Il est important de comprendre cette notion car de nombreux algorithmes sont écrits de façon récursive. Nous utiliserons, plus tard, la récursivité au chapitre sur les listes chaînées et pour l'écriture de certains programmes.

Jusqu'à présent nous avons écrit des fonctions itératives de façon classique. Le langage C autorise l'écriture de fonctions récursives. Une fonction est dite récursive lorsque dans son corps il y a une ligne de code qui l'appelle. Concept particulier demandant un certain niveau d'abstraction pour le comprendre : on écrit une fonction et elle s'appelle elle-même ! ...

Le corps d'une fonction récursive devra disposer d'une ligne d'appel (appel à la fonction elle-même) et d'un critère d'arrêt afin d'éviter un appel infini qui provoquerait un plantage du programme faute de mémoire suffisante.

Le code source des fonctions récursives est plus compact, parfois difficile à trouver mais plus lisible.

#### Exemple

Reprenons l'exemple classique et courant du calcul de la factorielle d'un entier n. La fonction factorielle avait été écrite, en itératif, comme suit :

```
int Facto (int n)    {  
    int i;  
    int f=1;  
    for ( i=1; i <= n; i++)    {  
        f *= i;  
    }
```

```

    }
    return f;
}

```

## Rappels

On sait que  $n! = n * (n-1) * (n-2) * (n-3) * \dots * 1$  avec  $1! = 1$

Donc la factorielle de  $n$  peut s'écrire de façon récursive :  $n! = n * (n-1)!$

Dit en d'autres termes : calculer factorielle  $n$  revient à calculer le produit de  $n$  par factorielle  $(n-1)$ .

Voyons maintenant son écriture de façon récursive :

```

int Facto (int n) {
    if (n > 1)
        return (n * Facto (n - 1));
    else
        return 1;
}

```

## Explications

### En-tête de la fonction

*int Facto (int n)*

La fonction se nomme *Facto*, elle recevra un entier (*int n*) et retournera un entier (*int* placé devant *Facto*)

### Appel récursif

*return (n \* Facto (n - 1));*

Sur cette ligne on fait appel à nouveau à la fonction *Facto* en lui envoyant cette fois-ci la valeur  $(n-1)$ .

### Critère d'arrêt

*return 1;*

Si la valeur de  $n$  n'est pas strictement supérieure à 1 on n'appelle plus la fonction *Facto* mais on renvoie 1.



## Fonctionnement

Exemple en voulant calculer *Facto* (3).

Dans notre exemple :  $3! = 3 * 2!$  Avec  $2! = 2 * 1!$  et  $1! = 1$

On appelle donc *Facto* en lui envoyant la valeur 3. La valeur 3 est recopiée dans  $n$  et comme le test  $n > 1$  est vrai on peut écrire l'égalité suivante :

$Facto(3) = 3 * Facto(2)$  : le calcul de *Facto* (3) est donc mis en attente du résultat de *Facto* (2) : on a appelé à nouveau *Facto* avec la valeur 2 (appel récursif).

Or,  $Facto(2) = 2 * Facto(1)$  : on appelle à nouveau *Facto* avec la valeur 1.

On peut donc écrire, en remplaçant :

$\text{Facto}(3) = 3 * \text{Facto}(2) = 3 * 2 * \text{Facto}(1).$

On appelle à nouveau *Facto* avec la valeur 1. Il y a donc un empilement des appels à la fonction *Facto*.  $\text{Facto}(3)$  est en attente du résultat de  $\text{Facto}(2)$ , lui-même en attente du résultat de  $\text{Facto}(1)$ .

$\text{Facto}(1)$  : on appelle *Facto* avec le paramètre 1. Le test  $n > 1$  n'est plus vrai on passe donc dans le "else" et du coup, on arrête les appels récursifs puisque 1 est renvoyé (*return 1;*). *Facto(1) vaut donc 1.*

Comme 1 est renvoyé on dépile les appels. Comme  $\text{Facto}(1)$  vaut 1 on peut calculer  $\text{Facto}(2)$  qui vaudra  $2 * 1$  soit 2. Du coup, on pourra calculer  $\text{Facto}(3)$  qui vaudra  $3 * 2$  soit 6.

$\text{Facto}(3)$  retournera finalement la valeur 6 au programme qui l'appelle.

Ceci est exact car  $3! = 3 * 2 * 1 = 6$

# EXERCICES

## Exercice 1

### Énoncé

Testez l'exemple de cours avec la fonction factorielle récursive. Utilisez un prototype pour la fonction récursive *Facto*.

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```
int Facto ( int );
```

```
int main() {
int x;
```

```
printf(" Saisir un entier x \n");
scanf ("%d", &x);
printf(" %d! = %d", x, Facto(x));
return 0;
}
```

```
int Facto (int n) {
if (n > 1)
    return (n * Facto (n - 1));
else
    return 1;
}
```

## Exercice 2

### Énoncé

Écrire le programme qui demande de saisir un entier et affiche la factorielle de ce dernier en utilisant une fonction qui est récursive et utilise l'opérateur ternaire.

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```
int Facto ( int );
```

```
int main() {
int x;
```

```
printf(" Saisir un entier x \n");
scanf ("%d", &x);
```

```
printf(" %d! = %d", x, Facto(x));
return 0;
}

int Facto (int n) {
    return n > 1 ? n * Facto (n - 1) : 1;
}
```

### Exercice 3

#### Énoncé

Écrire le programme qui demande de saisir deux entiers  $x$  et  $y$  et affiche leur produit (multiplication de  $x$  par  $y$ ) en utilisant une fonction récursive nommée produit.  
Pour rappel :  $x * y = x + x + x + \dots x$  ;  $y$  fois

#### Corrigé

```
#include<stdio.h>
#include<stdlib.h>

int produit (int x,int y) {
    if (y==0)
        return 0;
    else
        return(x + produit( x,y-1));
}

int main () {
    int x,y;

    printf("Saisir x:\n ");
    scanf("%d",&x);
    printf ("Saisir y:\n");
    scanf("%d",&y);

    printf("Le produit de %d par %d vaut %d", x,y,produit(x,y));
    return 0;
}
```

### Exercice 4

#### Énoncé

Écrire le programme qui demande de saisir deux entiers  $x$  et  $n$  et affiche  $x^n$  ( $x$  puissance  $n$ ) en utilisant une fonction récursive nommée Puissance.  
Pour rappel :  $x^n = x * x * x * \dots x$  ;  $n$  fois et, pour tout  $x$  entier ou réel :  $x^0 = 1$ .  
Exemple :  $4^3 = 4 * 4 * 4 = 64$

#### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```

int Puissance(int x,int n) {
    if ( n==0)
        return 1;
    else
        return( x * Puissance( x,n-1));
}

```

```

int main () {
    int x,n;

```

```

    printf("Saisir x:\n ");
    scanf("%d",&x);
    printf ("Saisir n:\n");
    scanf("%d",&n);

```

```

    printf(" %d a la puissance %d vaut %d", x,n,Puissance(x,n));
    return 0;
}

```

## Exercice 5

### Énoncé

Soit la suite  $(U_n)$  définie par  $U_n = 2 * U_{n-1} + 3$  avec  $U_1 = 1$ .

Écrire le programme qui demande de saisir un entier long puis affiche la valeur de cette suite pour le rang saisi, en faisant appel à une fonction récursive.

**Exemple** : si on saisit 3 il sera calculé  $U_3$  qui vaut  $2 * U_2 + 3$  soit  $2 * (2 * U_1 + 3) + 3$  c'est-à-dire 13.

### Corrigé

```

#include<stdio.h>
#include<stdlib.h>

```

```

long Un (long n) {
    if (n==1)
        return 1;
    else
        return (2*Un(n-1)+3);
}

```

```

int main () {
    long n;

```

```

    printf("Saisir n:\n ");
    scanf("%ld",&n);
    printf("Le terme U%ld vaut %ld", n,Un(n));
}

```

## Exercice 6

### Énoncé

Écrire le programme qui demande de saisir un entier  $n$  et qui affiche la somme des  $n$  premiers nombres entiers grâce à une fonction récursive.

**Exemple** : si on saisit 4 il s'affichera 6 (qui correspond au calcul  $0+1+2+3$ ).

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```
int somme_des_n_entiers (int n) {
    if ( n==1)
        return 0;
    else
        return( (n-1)+ somme_des_n_entiers(n-1));
}
```

```
int main () {
    int n;

    printf("Saisir n:\n ");
    scanf("%d",&n);
    printf("La somme des %d premier(s) entier(s)
           vaut : %d", n,somme_des_n_entiers(n));

    return 0;
}
```

## Exercice 7

### Énoncé

Écrire le programme qui demande de saisir un entier  $n$  et qui affiche le  $n^{\text{ième}}$  nombre pair. Le programme fera appel à une fonction `n_iemeNbPair` récursive.

**Exemple** : Si on saisit 6 il s'affichera 10 (le  $6^{\text{ième}}$  nombre pair est 10 (0, 2, 4, 6, 8, 10)).

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```
int n_iemeNbPair (int)
```

```
int main () {
    int n;

    printf("Saisir le rang n du nb pair:\n");
    scanf("%d",&n);
    printf("Le %d ieme entier pair est: %d",n,n_iemeNbPair(n));

    return 0;
}
```

```

}

int n_iemeNbPair(int n) {
    if (n==1)
        return 0;
    else
        return(2+ n_iemeNbPair(n-1));
}

```

## Exercice 8

### Énoncé

Écrire un programme qui utilise une fonction récursive nommée *capital\_a\_terme* pour donner la valeur du capital (capital initial + intérêts) après un certain nombre d'années de placement. Il est saisi au clavier : le capital initial, le taux d'intérêt fixe, le nombre d'années du placement. Les intérêts acquis une année sont à leur tour productifs d'intérêts les années suivantes.

### Corrigé

```

#include<stdio.h>
#include<stdlib.h>

```

```

float capital_a_terme(float c, float taux, int annees) {
    if( annees == 0)
        return c;
    else
        return capital_a_terme ( ( c+c*taux/100), taux, annees-1);
}

```

```

int main () {
    float c,taux;
    int annees;

    printf("Saisir le Capital initial:\n");
    scanf("%f",&c);
    printf("Saisir le taux d'interet:\n");
    scanf("%f",&taux);
    printf("Saisir le nb d'annees:\n");
    scanf("%d",&annees);
    printf("Pour un capital initial de:%.2f; un taux de:%.2f; un nb d'annees de : %d;\n le capital a terme sera de:%.2f", c, taux,annees, capital_a_terme(c,taux,annees)); }

```



## Exercice 9

### Énoncé

Écrire un programme qui demande de saisir le nombre d'éléments contenus dans un tableau d'entiers puis de saisir ces derniers au clavier. Le programme appelle ensuite une fonction récursive pour afficher le tableau.



## Corrigé

```
#include<stdio.h>
#include<stdlib.h>

//fonction affichant le tableau
void Affiche(int TabEntiers[],int indice,int n) {
if(indice==(n-1))//dernière case
    printf("%d\n", TabEntiers[n-1]);//affiche dernier
else {
    printf("%d ; ", TabEntiers[indice]);
    Affiche(TabEntiers,indice+1,n);
}
}

int main() {
int n,i;

printf("Saisir n, le nb d'entiers dans le tableau :\n");
scanf("%d", &n);
int TabEntiers[n];
int indice =0;

// saisie
for(i=0;i<n;i++) {
    printf("Saisir un entier du tableau :\n");
    scanf("%d",&TabEntiers[i]);
}
printf("Affichage du tableau :\n");
Affiche(TabEntiers,indice,n);

return 0;
}
```

## Exercice 10

### Énoncé

Écrire un programme qui demande de saisir le nombre d'éléments dans un tableau d'entiers puis de saisir ceux-ci au clavier. Le programme appelle ensuite une fonction récursive pour afficher la somme des éléments du tableau.

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>

int SommeEntiers(int TabEntiers[],int indice,int n) {
if(indice==(n-1))//dernière case
    return TabEntiers[indice];
else
    return(TabEntiers[indice]+SommeEntiers(TabEntiers,(indice+1),n));
}
```

```

}

int main() {
    int n, i;
    int indice = 0;

    printf("Saisir n, le nb d'entiers du tableau :\n");
    scanf("%d", &n);
    int TabEntiers[n];

    for(i=0; i<n; i++) {
        printf("Saisir un entier du tableau :\n");
        scanf("%d", &TabEntiers[i]);
    }

    printf("Affichage du tableau :\n");
    for(i=0; i<n; i++)
        printf("%d\n", TabEntiers[i]);

    printf("La somme des elts de ce tableau d'entiers est %d\n", SommeEntiers(TabEntiers, indice, n));
    return 0;
}

```

## Exercice 11

On lâche une balle depuis une hauteur  $h$ , à chaque rebond la balle remonte de  $h/2$ . Au bout de  $n$  rebonds, à quelle hauteur se trouve la balle ? Écrire le programme qui demande la hauteur initiale  $h$  exprimée en mètres et le nombre de rebonds  $nb$  puis affiche la hauteur du dernier rebond.

Exemple : une hauteur  $h=2$  mètres ; si on a saisi 2 rebonds il s'affichera 0,5 mètres.

### Corrigé

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Prototype
float final_height(float, int);

int main(){
    float h; //la hauteur initiale de la balle
    int nb; //le nombre de rebonds
    printf("Saisir la hauteur initiale de la balle\n");
    scanf("%f", &h);
    printf("Saisir le nombre de rebonds\n");
    scanf("%d", &nb);
    float res = final_height(h, nb);
    printf("La hauteur finale est %.3f cm\n", res);
}

```

```
float final_height(float h, int nb){
if (nb == 0)
    return h;
else
    return final_height((h/2.0), nb - 1);
}
```



## Exercice 12

### Énoncé

Écrire un programme qui demande de saisir le nombre d'éléments contenus dans un tableau d'entiers puis de saisir ces derniers au clavier. Le programme appelle ensuite une fonction récursive *Affiche()* pour afficher le tableau (exercice vu précédemment). Le programme fait ensuite appel à une fonction récursive *ESTPALINDROME()* qui retourne 1 si le tableau est un palindrome, autrement 0.

Un tableau est un tableau palindrome si l'on obtient la même séquence lorsqu'il est lu dans un sens ou dans l'autre.

### Exemples

Le tableau ci-dessous n'est pas un tableau palindrome :

14	45	15
----	----	----

Le tableau ci-dessous est un tableau palindrome :

1	10	100	100	10	1
---	----	-----	-----	----	---

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
```

```
void Affiche(int T[],int indice,int n) {
if(indice==(n-1))//dernière case
printf("%d\n", T[n-1]);//affiche dernier
else {
    printf("%d\t ; ", T[indice]);
    Affiche(T,indice+1,n);
}
}
```

```
int ESTPALINDROME (int T [], int n, int indice)
{ if (indice >=1)
    return 1;
else
    if (T[indice]!=T[n-1-indice])
        return 0;
else
    return (ESTPALINDROME (T,--n,++indice));
```

```

}

int main() {
    int n, i;
    int indice = 0;
    printf("Saisir n, le nb d'entiers du tableau :\n");
    scanf("%d", &n);
    int TabEntiers[n];

    for(i=0; i<n; i++) {
        printf("Saisir un entier du tableau :\n");
        scanf("%d", &TabEntiers[i]);
    }

    printf("Affichage du tableau : ");

    Affiche (TabEntiers, indice, n);

    if (ESTPALINDROME (TabEntiers, n, indice) )
        printf ("\nTableau palindrome\n");
    else
        printf ("Tableau non palindrome\n");

    return 0;
}

```



## Exercice 13 Les lapins et la suite de Fibonacci

### Énoncé

Le mathématicien Leonardo Fibonacci a laissé son nom à une suite mathématique définie par :

- $U_0 = 0$
- $U_1 = 1$
- $U_n = U_{n-1} + U_{n-2}$

Les premiers termes de cette suite sont donc :

0 ; 1 ; 1 ; 2 ; 3 ; 5 ; 8 ; 13 ; 21 ; 34 ; 55 ; 89 ....

On suppose qu'un couple de lapins est en âge de se reproduire au bout d'un mois. Tous les lapins en âge de se reproduire donne un nouveau couple de lapins chaque mois.

On part d'un couple de lapin ; écrivez le programme qui demande de saisir une durée en mois et appelle une fonction réursive (basée sur la suite de Fibonacci) qui calcule le nombre de couples présents au bout de cette durée.

On suppose qu'il n'y a pas de mortalité de lapins.

### Corrigé

```
#include<stdio.h>
#include<stdlib.h>
long Fibonacci (long n) {
    if (n==0 || n==1)
        return 1;
    else
        return (Fibonacci(n-1)+Fibonacci (n-2));
}
```

```
int main ()
{
    int n;
    printf("Saisir le nombre de mois :\n ");
    scanf("%d",&n);
    printf("Nombre de lapins au bout de %d mois: %ld", n, Fibonaccilter(n));
    return 0;
}
```

## Exercice 14 Les lapins (suite)

### Énoncé

Ecrivez maintenant la fonction non récursive qui retourne le nombre de lapins...  
Testez ensuite les 2 versions (itérative et récursive) pour des valeurs 44, 45, 46, 47... Qu'observez-vous ?

### Corrigé

```
long Fibonaccilter (long n) {
    long Primo = 0 ;
    long Secondo = 1 ;
    long Tampon ;
    int i ;
    //if (n<2)
        // return 1;
    //Fibonacci(n-1)+Fibonacci (n-2)
    for (i=1; i<=n;i++)
    {
        Tampon = Primo + Secondo;
        Primo = Secondo;
        Secondo = Tampon;
    }

    return (Primo);
}
```

### Rq

On observe qu'à partir de 43 (43 mois), la fonction récursive devient de plus en plus lente ; à partir de 48 il faut attendre des dizaines de secondes alors que la fonction

itérative donne le résultat instantanément ; au-delà de 52 c'est en minutes qu'il faut attendre...

Cnam Canesi NFA037 Copyright