# Programmation en C/C++

# Série 4
# Les fonctions
# Passages de paramètres

 Important !

 Difficile !



**Objectifs**
Comprendre et utiliser les fonctions en langage C, notions de paramètres, variables globales et locales.

## Introduction

Jusqu'à présent nous avons créé des programmes où le code source était présent dans le programme principal uniquement (main). Quand le programme devient un peu plus complexe ou un peu plus long on pourra faire appel à des fonctions qui vont réaliser une tâche bien précise. Lorsque nous avons abordé les tableaux nous avons géré la saisie des données puis l'affichage de celles-ci au niveau du *main*. On peut envisager un programme principal, qui, cette fois, fait appel à une fonction pour gérer la saisie des données et à une autre fonction pour l'affichage des données du tableau. En quelque sorte, le programme principal est déchargé de certaines tâches et joue un rôle de chef d'orchestre en appelant les différentes fonctions qui s'exécutent.

**Intérêt des fonctions**
- Lisibilité du code source accrue
- Programme principal simplifié
- Non réécriture de portions de code

**Exemple**
Écrivons un programme qui demande de saisir deux nombres entiers dans des variables nommées *a* et *b* et qui affiche leur somme.

**Programme sans fonction**
```
#include <stdio.h>
int main()
{
```

```
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",(a+b));
return 0;
}
```

Écrivons maintenant le même programme avec une fonction *Somme* qui réalise l'addition de *a* et *b* et renvoie le résultat au programme principal.

**Programme avec une fonction**

```
#include <stdio.h>

// Fonction Somme
int Somme (int a, int b)
{
   return (a+b);
}

// programme principal
int main()
{
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",Somme(a,b));
return 0;
}
```

**Explications**

1. Au niveau du programme principal (*main*) on demande de saisir les deux entiers et les valeurs sont stockées dans les variables *a* et *b*.
2. *Somme(a,b)* est l'appel à la fonction *Somme* ; le programme principal lui envoie les arguments ou paramètres correspondant aux valeurs contenues dans les variables *a* et *b.*
3. La fonction *Somme* s'exécute alors et renvoie au programme principal le résultat de *a+b* grâce à l'instruction *return*.
4. Au niveau du programme principal ce résultat est affiché car il est situé dans un *printf*.

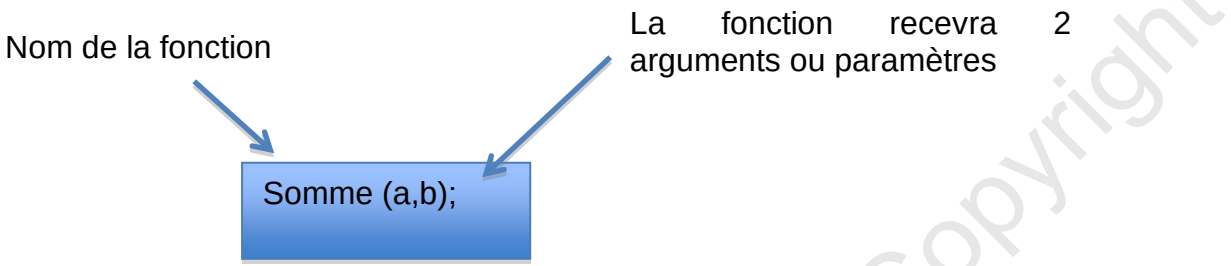
**Remarques**

*Somme(a,b));* correspond à l'appel de la fonction *Somme*
*int Somme (int a, int b)* est l'en-tête de la fonction *Somme*

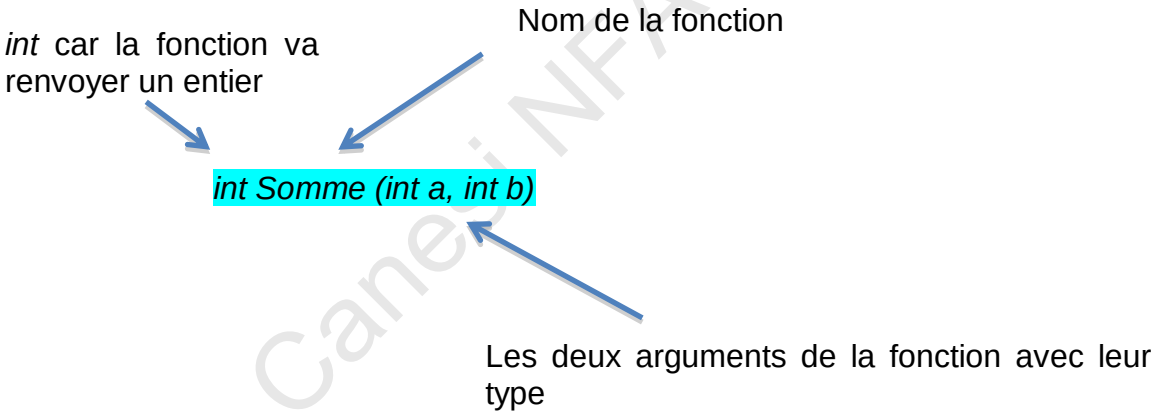
*return (a+b);* correspond à la valeur de renvoi de la function *Somme.*

**Détails**

**Appel de la fonction au niveau du programme principal**




**En-tête de la fonction**




**Corps de la fonction**

Le corps de la fonction correspond aux lignes de code situées juste au dessous de l'en-tête de la fonction, entre les accolades.

```
// Fonction Somme
int Somme (int a,
int b)
{
   return (a+b);
}
```

Corps de la fonction en vert

# Déclaration d'une fonction / notion de prototype

Si une fonction est définie dans le code source avant la portion de code qui l'utilise, il n'est pas besoin de la déclarer. Dans le cas contraire il faut la déclarer en plaçant son *prototype* au tout début du programme, après les directives de compilations.

Le prototype d'une fonction indique :

1- Le type de donnée renvoyée,
2- Le nom de la fonction,
3- Les différents types de ses paramètres, successivement.

**Exemple de prototype**

*int Somme (int , int);*

Ici, la fonction s'appelle Somme ; elle renvoie un entier ; elle recevra 2 paramètres de type entier.

**Rq**

Placer un point virgule à la fin du prototype !

**Rq**

Pas de virgule à la fin de l'en-tête d'une fonction !

**Exemple 1** : Sans prototype car la fonction Somme est écrite **avant** le programme principal qui l'utilise.

```
#include <stdio.h>

// Fonction Somme
int Somme (int a, int b)
{
   return (a+b);
}

// programme principal
int main()
{
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",Somme(a,b));
return 0;
}
```

**Exemple 2** : Présence d'un prototype car la fonction Somme est écrite **<u>après</u>** le programme principal qui l'utilise.

```
#include <stdio.h>

// prototype de la function Somme
int Somme (int, int);

// programme principal
int main()
{
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",Somme(a,b));
return 0;
}

// Fonction Somme
int Somme (int a, int b)
{
   return (a+b);
}
```

## Valeur renvoyée ou retournée

L'instruction *return*, termine toujours une fonction ; elle permet de renvoyer ou retourner une valeur au programme qui l'a appelée.

**Exemple**

```
int Somme (int a, int b)
{
   return (a+b);
}
```

**Remarque**

- La fonction Somme renvoie un entier qui correspond au résultat de la somme des contenus des variables a et b.

- Une fonction peut ne renvoyer aucune valeur ; dans ce cas, on place le mot clé *void* devant son nom.

- Le "*return ;*" en fin de fonction est facultatif.

**Exemple**

```
void année (int a) {
   printf("Année %d, a);
```

```
return;
}
```

**Remarque**

En langage C la distinction entre fonction et procédure n'existe pas comme dans de nombreux autres langages. Il n'existe que des fonctions qui peuvent retourner une (seule) valeur ou ne rien renvoyer (*void*).

## Les arguments ou paramètres

### Type void (vide)

Une fonction peut ne pas recevoir d'arguments on mettra le mot clé *void* entre les parenthèses de l'en-tête.

**Exemple avec aucun argument**

*int Blue(void);*

Ici, la fonction *Blue* retournera un entier mais ne recevra pas d'argument.

### Passage d'arguments par valeur

En C, lorsqu'on appelle une fonction avec des variables en paramètres, les variables sont des paramètres passés par valeur : la fonction ne reçoit pas les variables mais le contenu de celles-ci. Le contenu de chaque variable est recopié dans une variable (de même nom ou pas) au niveau de la fonction : une fonction ne pourra donc pas modifier directement une variable !

**Exercice de cours**

Qu'affiche exactement le programme suivant ? Réfléchissez bien !
Programmez ensuite cet exercice.

```
#include<stdio.h>

//prototype
void F1 (int );

//main
int main() {
int alpha = 0;
F1(alpha);
printf ("Valeur de alpha dans le main : %d\n", alpha);
/* il s'affichera toujours 0 */
return 0;
}
```

```
//function
void F1 (int a) {
a--;
printf ("valeur de a dans F1 : %d\n", a);
/* il s'affiche  -1 c'est la valeur de a et non de alpha*/
}
```

**Corrigé**

```
valeur de a dans F1 : -1
Valeur de alpha dans le main : 0
```

*Capture d'écran de l'exécution du prg*

**Rq**

Le contenu de la variable *alpha* est envoyé et recopié dans la variable *a*, au niveau de la fonction *F1*. La variable *a* est connue par la fonction, pas *alpha*. La variable *alpha* est connu du *main* mais pas de la fonction.

## Passage d'arguments par adresse

Pour modifier le contenu d'une variable à l'aide d'une fonction, il faudra passer en paramètre **l'adresse** de cette variable. La fonction pourra alors modifier le contenu à l'adresse qu'elle connaitra. Une variable contenant une adresse est un **pointeur** ; nous verrons donc cet aspect au chapitre sur les pointeurs.

**Cas particulier du passage d'un tableau en argument de fonction**

Le passage d'un tableau en argument se réalise par adresse et non pas par valeur (on y reviendra plus tard). On envoie donc l'adresse du tableau à la fonction et il faudra envoyer également le nombre d'éléments du tableau. Du coup, une fonction recevant un tableau en argument peut le modifier.

**Exemple d'en-tête** d'une fonction recevant un tableau
*int FonctionZZ (int tab[ ], taille)*
On envoie un tableau d'entiers nommé *tab* et *taille* est la taille de ce tableau : la fonction recevra deux arguments.

**Exemple de prototype avec un tableau**
*int FonctionZZ (int [ ], int)*
***ou***
*int FonctionZZ (int *, int)*
Tableau d'entier et entier en paramètres, au niveau du prototype.

**Rq**

Une fonction ne peut pas renvoyer un tableau.

## Variables locales et globales

Une variable est dite "**locale**" quand elle est déclarée à l'intérieur d'une fonction ou d'un bloc d'instruction. Après exécution de cette portion de code source, cette variable n'existe plus et son contenu est donc perdu.

Une variable est dite "**globale"** quand elle est déclarée en début de programme, en dehors de toute fonction ou du programme principal. Cette variable possède une adresse mémoire fixe et est connue de toutes les lignes de code qui la suivent.

Les variables locales ont donc une "**portée**" bien moindre que les variables globales.

Essayez de prévoir précisément les résultats affichés par les 2 programmes ci-après. Puis, compilez et exécutez les afin de vérifier.

**Exemple 1**

```
# include <stdio.h>

int Somme (int x, int y)
{
    return (x+y);
}

int main()
{
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",Somme(a,b));
return 0;
}
```

Lors de l'appel de la fonction *Somme*, le contenu de *a* est recopié dans la variable *x* et le contenu de *b* est recopié dans la variable *y*. Donc si on a saisi 10 et 14 au clavier il s'affichera 24.
Utiliser *a* et *b* dans *Somme* générerait une erreur de compilation car *Somme* ne connaît pas les variables locales *a* et *b*. Utiliser *x* et *y* dans le programme principal générerait également une erreur de compilation car ce dernier ne connaît pas les variables *x* et *y*.

**Exemple 2**

```
# include <stdio.h>
int a, b;

int Somme ()
{
   return (a+b);
}

int main()
{
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("La somme : %d",Somme());
return 0;
}
```

Ici *a* et *b* sont des variables globales puisque déclarées en dehors de toute fonction. Le programme principal appelle *Somme()* sans lui envoyer de paramètres et *Somme()* retourne le contenu de (*a+b*) qui est alors affiché par le programme principal. *Somme()* connaît les variables *a* et *b* qui sont globales. Donc si on a saisi 10 et 14 au clavier il s'affichera 24.

**Question**

Que se passe-t-il si, dans l'exemple précédent, on ajoute la ligne *int a, b;* juste avant le *return (a+b);* de la fonction *Somme()* ? Testez en exécutant le programme.

**Corrigé**

Cela signifie que deux variables locales *a* et *b* (sans lien avec les variables globales) sont déclarées et (souvent) initialisées par défaut à 0 dans la fonction *Somme(). Somme()* retournera donc 0, quoiqu'on saisisse ! Donc si on a saisi 10 et 14 au clavier il s'affichera 0. Dans certains cas la valeur affichée sera une valeur quelconque et loufoque car les variables n'ont pas été initialisées à 0. Le programme récupère alors des valeurs présentes à l'adresse où sont rangées les variables *a* et *b* et retourne donc une valeur loufoque.

Nous reparlerons des variables locales et globales au chapitre sur les pointeurs. Il est nécessaire de bien comprendre ces exemples en les programmant et en les testant dans tous les sens à l'exécution.

# EXERCICES

## Exercice 1

### Énoncé

Écrire le programme qui demande de saisir trois entiers et les envoie à une fonction nommée "Produit" qui retourne leur produit (multiplication). Cette fonction sera écrite après le programme principal il faudra donc écrire son prototype.

### Corrigé

```
#include <stdio.h>

// prototype de la fonction
int Produit (int, int, int);

// programme principal
int main() {
int a,b,c;

printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("Valeur de c ?");
scanf("%d", &c);
printf("Le produit vaut  : %d\n",Produit(a,b,c));

return 0;
}

// Fonction Produit
int Produit (int x, int y, int z) {
return (x*y*z);
}
```

**Rq** :  les contenus des variables a, b et c sont recopiés dans x, y et z.

## Exercice 2

### Énoncé

Écrire le programme qui demande de saisir deux réels et les envoie à une fonction nommée "MinNB" qui retourne le plus petit des deux.

**Consigne** : Vous utiliserez l'opérateur ternaire pour l'écriture de MinNB.

**Corrigé**

```
#include <stdio.h>
// prototype de la fonction MinNB
float MinNB (float, float);

// programme principal
int main() {
float a,b; printf("Valeur de a ?");
scanf("%f", &a);
printf("Valeur de b ?");
scanf("%f", &b);
printf("Le plus petit vaut : %f\n",
MinNB(a,b));
return 0;
}

// Fonction MinNB
float MinNB (float x, float y) {
return (x<y ? x : y);
}
```

# Exercice 3

**Énoncé**

Écrire le programme qui demande de saisir deux entiers *a* et *b* et les envoie à une fonction nommée "Puiss" qui retourne la valeur $a^b$.

Rappel : $a^b$ = *a*a*a*……*a* ; *b* fois.

**Corrigé**

```
#include <stdio.h>

int Puiss (int, int);
int main() {
int a,b;
printf("Valeur de a ?");
scanf("%d", &a);
printf("Valeur de b ?");
scanf("%d", &b);
printf("%d puissance %d vaut : %d\n", a,b, Puiss(a,b));
return 0;
}

// Fonction Puiss
int Puiss (int x, int y) {

int i, resultat=1;
for (i = 1 ; i<=y ; i++)
        resultat *=x;
```

```
return (resultat);
 }
```

# Exercice 4

**Énoncé**

Écrire le programme qui demande de saisir un entier au clavier et qui utilise une fonction "Facto" qui renvoie la factorielle de n.

**Rappel** : Factorielle n (que l'on notera n!, en mathématiques) est définie par :
n!= n * (n-1)! avec 0 ! =1

Exemple : 4! = 4 *3 * 2* 1 = 24

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

# Exercice 5

**Énoncé**

Écrire le programme principal qui demande de saisir 5 réels dans un tableau et utilise ensuite une fonction pour les afficher.

**Corrigé**

```
#include<stdio.h>
#include<stdlib.h>
void Affiche ( float T []  , int MAXI) {
int a;
```

```
for(a=0;a<MAXI;a++)
        printf("%f\n",T[a]);
return;
}

int main() {
int a;   float T [5];

for ( a=0; a<5; a++)   {
        printf("Saisir un reel:");
        scanf("%f",&T[a]);
        }
Affiche(T, 5);
return 0;
}
```

## Exercice 6

**Énoncé**

Écrire le programme principal qui demande de saisir 5 entiers dans un tableau et utilise ensuite une fonction "Moyenne" qui renvoie la moyenne des éléments du tableau.

**Corrigé**

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

void Affiche ( int T []  , int MAXI) {
int a;

for(a=0;a<MAXI;a++)
        printf("%d\n",T[a]);

return;
 }

float Moyenne ( int T []  , int MAXI) {
int a, somme=0;

for(a=0;a<MAXI;a++)
        somme +=T [a]  ;

return (   (float)somme/MAXI  );
}

int main() {
int i;
int T [5];
```

```
for ( i=0; i<5; i++)   {
        printf("Saisir un entier:");
        scanf("%d",&T[i]);
        }
Affiche(T, 5);
printf("Moyenne : %.2f",Moyenne(T,5));
return 0;
}
```

# Exercice 7

**Énoncé**

Écrire le programme principal qui demande de saisir 5 entiers dans un tableau et utilise ensuite une fonction "PlusGrand" qui renvoie le plus grand des éléments du tableau.

**Corrigé**

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

void Affiche ( int T []  , int MAXI) {
int a;

for(a=0;a<MAXI;a++)
        printf("%d\n",T[a]);

return;
 }

int Plusgrand ( int T []  , int MAXI) {
int a, Max= T [0]  ;

for(a=0;a<MAXI;a++)
        if (T [a]  > Max)
                Max = T [a];

return (  Max  );
}

int main() {
int i;
int T [5];

for ( i=0; i<5; i++)   {
        printf("Saisir un entier:");
        scanf("%d",&T[i]);
        }
Affiche(T, 5);

printf("Plus grand element : %d",Plusgrand(T,5));
```

```
return 0;
}
```

# Exercice 8

## Énoncé

Écrire le programme principal qui demande de saisir 5 entiers dans un tableau et utilise ensuite une fonction "RangPlusGrand" qui renvoie le rang du plus grand des éléments du tableau.

## Corrigé

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

void Affiche (int T [ ]  , int MAXI) {
int a;

for(a=0;a<MAXI;a++)
        printf("%d\n",T[a]);

return;
 }

int RangPlusGrand ( int T []  , int MAXI) {
int a, Max= T [0]  ;
int Indice = 0;

for(a=0;a<MAXI;a++)
        if (T [a]  > Max) {
                Max = T [a];
                Indice = a;
        }

return (  ++Indice  );
}

int main() {
int i;
int T [5];

for ( i=0; i<5; i++)   {
        printf("Saisir un entier:");
        scanf("%d",&T[i]);
        }
Affiche(T, 5);

printf("Rang du plus grand element : %d",RangPlusGrand(T,5));
return 0;
}
```

**Rq** : Noter le *return (++Indice);* qui renvoie le contenu de *Indice* augmenté de 1, préalablement.

# Exercice 9

### Énoncé

Écrire le programme principal qui appelle une fonction qui affecte la valeur 1 aux 5 éléments d'un tableau d'entiers puis appelle une autre fonction qui affichera le contenu de ce tableau.

### Corrigé

```
#include <stdio.h>

void Initialisation (int Tab[], int n) {
int i;

for(i=0; i<n; i++)
        Tab[i] = 1;
}

void Affichage (int Tab[], int taille) {
int i;

for(i=0; i<taille; i++)
        printf("%d\t", Tab[i]);

printf("\n");
}

int main() {
int T[5];
Initialisation(T, 5);
Affichage(T, 5);
return 0;
}
```



# Exercice 10

### Énoncé

Dans le programme principal, créer la structure de tableau nommée *T* contenant 5 réels. Le programme appelle ensuite une fonction *Modif()* qui va demander de saisir les données du tableau au clavier puis une autre fonction *Affiche()* qui va afficher les 5 éléments du tableau. Le programme principal répète une seconde fois l'opération de modification et d'affichage des données.

### Corrigé

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

void Affiche ( float T []  , int MAXI) {
int a;

for(a=0;a<MAXI;a++)
        printf("%f\n",T[a]);
return;
}

void Modif ( float T []  , int MAXI) {
int a;
for(a=0;a<MAXI;a++)
        {
        printf("Saisir un reel:");
        scanf("%f",&T[a]);
        }
return;
}

int main() {
int a;   float T [5];

Modif(T, 5);
printf("Affichage du tableau initial : \n");
Affiche(T, 5);
Modif(T, 5);
printf("Affichage du tableau modifié : \n");
Affiche(T, 5);
return 0;
}
```

**Rq** : on remarquera que la fonction *Modif()* permet bien de modifier le contenu du tableau *T*.

# Exercice 11

**Énoncé**

Écrire le programme qui demande de saisir un entier et qui affiche s'il est pair.
On utilisera une fonction nommée *EstPair()*.
**Rappel** : un nombre pair se termine par 0, 2, 4, 6, ou 8.

**Corrigé**

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

//Prototype
void EstPair (int);

int main(){
```

```
   int a;
   printf("Saisissez un entier :");
   scanf("%d", &a);

   // appel de le Fonction
   EstPair(a);
   return 0;
}

void EstPair(int n){
   if (n%2 == 0)
      printf("%d est pair\n", n);
   else
      printf("%d est impair\n", n);
}
```

## Exercice 12

**Énoncé**

Écrire le programme qui demande de saisir un entier et qui affiche s'il est premier.

On utilisera une fonction nommée *EstPremier()* qui retournera 1 si le nombre est premier et 0 s'il ne l'est pas.

**Rappel** :
- Un nombre premier n'est divisible que par lui-même et 1.
- 1 n'est pas un nombre premier. On gèrera ce fait là.

**Corrigé**

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

//Prototype int EstPremier (int);

int main(){
int a;
printf("Saisissez un entier :");
scanf("%d",&a);

(EstPremier(a)==1 && a!=1)? printf("%d est premier\n", a) : printf("%d n'est pas
premier\n", a);

return 0;
}

int EstPremier(int n){
int i;

for (i=2 ; i<n ; i++)      {
        if (n%i == 0)
                return 0;
```

```
}

return 1;

}
```

Un nombre est premier s'il est divisible uniquement par 1 et lui-même.
On saisit un nombre *n* ; on boucle entre 2 et le nombre *n*, non compris. Dès qu'on trouve un diviseur (fonction modulo (%)) on retourne 0 (qui entraîne une sortie de la fonction !). Si on arrive en fin de boucle cela signifie que l'on n'a pas trouvé de diviseur autre que 1 et le nombre lui-même : on peut donc renvoyer 1 (le nombre est pair). Le *main()* affiche que le nombre est premier si la valeur retournée est 1 **et** si la valeur saisie et testée n'est pas 1.

## Exercice 13

### Énoncé

Qu'affiche le programme suivant si l'on saisit 10 et 100 ?

```
# include <stdio.h>
int a, b;
void Somme ()
{
a--;
b++;
printf("Dans la fonction Somme : \t ");
printf("%d\t", a);
printf("%d\n", b);
}

int main() {

printf("Valeur de a ?\n");
scanf("%d", &a);
printf("Valeur de b ?\n");
scanf("%d", &b);

Somme();
printf("Dans le main : \t");
printf("%d\t", a);
printf("%d\n", b);

return 0;
}
```

### Corrigé

```
Valeur de a ?
10
Valeur de b ?
100
Dans la fonction Somme :          9      101
Dans le main :  9       101
```

Les valeurs 10 et 100 sont affectées au variables *a* et *b* qui sont globales ! La fonction *Somme()* modifie le contenu des deux variables globales *a et b*. *Somme*() affiche donc 9 et 101. On revient dans le *main()* et il s'affiche 9 et 101 car les deux variables *a* et *b*, globales, ont été modifiées par la fonction *Somme()!*

## Exercice 14

### Énoncé

Qu'affiche le programme suivant si l'on saisit 10 et 100 ?

```
# include <stdio.h>
void Somme (int a, int b) {
a--;
b++;
printf("Dans la fonction Somme : \t ");
printf("%d\t", a);
printf("%d\n", b); }

int main() {
int a,  b;
printf("Valeur de a ?\n");
scanf("%d", &a);
printf("Valeur de b ?\n");
scanf("%d", &b);
Somme(a,b);
printf("Dans le main : \t");
printf("%d\t", a);
printf("%d\n", b);
return 0;
}
```

### Corrigé

```
Valeur de a ?
10
Valeur de b ?
100
Dans la fonction Somme :          9      101
Dans le main :  10      100
```

Les valeurs 10 et 100 sont affectées au variables *a* et *b* du *main().* 10 et 100 sont recopiées dans des variables *a* et *b* spécifiques de la fonction *Somme()*. *Somme*() affiche donc 9 et 101. On revient dans le *main()* et il s'affiche 10 et 100 car les deux variables *a* et *b* du main() n'ont pas été modifiées !

# Exercice 15

**Énoncé**

Dans le programme principal, créer la structure de tableau nommée *Tab* qui contiendra un nombre variable de réels. Le programme demande ensuite combien de réels contiendra le tableau puis appelle ensuite une fonction *Modif()* qui va demander de saisir les données du tableau au clavier puis une autre fonction *Affiche()* qui va afficher tous les éléments du tableau.

**Corrigé**

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

void Affiche ( float T []  , int MAXI) {
int a;
for(a=0;a<MAXI;a++)
        printf("%f\n", T[a]);

return;
}

void Modif ( float T []  , int MAXI) {
int a;

for(a=0;a<MAXI;a++)  {
        printf("Saisir un reel:");
        scanf("%f",&T[a]);
        }
return;
}

int main() {
int a, NB;
float* Tab ;  // déclaration dynamique

printf("Nombre d'elements dans le tableau  ? : \f");
scanf("%d",&NB);

Tab =  malloc(NB * sizeof(float));

Modif(Tab, NB);
printf("Affichage du tableau  : \n");
Affiche(Tab, NB);

return 0;
}
```

# Exercice 16

**Énoncé**

Répondez au deux questions suivantes :

- Qu'affiche le programme suivant si on saisit la valeur 3 ?
- Que vaut la variable *a* du programme principal après exécution, pourquoi ?

```
#include<stdio.h>
#include<stdlib.h>
int Modif ( int a)    {
return --a;
}

int main() {
int a;

printf("Valeur de a ? : \n");
scanf("%d",&a);
printf("Valeur de a saisie : %d\n",a);
printf("Valeur renvoyee apres execution de la fontion Modif : %d\n",Modif(a));
return 0;
}
```

**Corrigé**

La variable *a* du programme principal contiendra donc 3. Cette valeur 3 est recopiée dans une autre variable *a* de la fonction *Modif*. *Modif* décrémente le contenu de sa variable *a* et renvoie sa valeur. Modif renvoie donc 2. La variable *a* du programme principal vaut toujours 3 !
Si vous en doutez, ajoutez la ligne suivante à la fin du programme principal :
*printf("Valeur de a apres execution de la fonction Modif : %d\n",a);*

Compilez, exécutez et vérifiez !

# Exercice 17

**Énoncé**

En reprenant l'exemple précédent on effectue la modification suivante : la fonction *Modif* ne reçoit pas de paramètre (il faudra donc également déclarer une variable *a* dans la fonction *Modif*). Écrire le programme et répondre aux questions suivantes :

- Qu'affiche le programme suivant si on saisit la valeur 3 ?

- Que vaut la variable *a* du programme principal après exécution, pourquoi ?

**Corrigé**

```
Valeur de a ? :
3
Valeur de a saisie : 3
Valeur renvoyee apres execution de la fontion Modif : -1
```

Dans certains cas, la valeur affichée sera une valeur quelconque et loufoque car les variables n'ont pas été initialisées à 0. Le programme récupère la valeur présente à l'adresse où est rangée la variable *a* et retourne donc une valeur loufoque.

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

int Modif ()    {
int a;
return --a;
}

int main() {
int a;

printf("Valeur de a ? : \n");
scanf("%d",&a);
printf("Valeur de a saisie : %d\n",a);
printf("Valeur renvoyee apres execution de la fontion Modif : %d\n",Modif());
return 0;
}
```

## Exercice 18

**Énoncé**

Écrire un programme principal qui demande à une fonction *Modif()* de modifier une variable globale nommée *n* de type entier et de renvoyer sa valeur. Le programme principal demandera de saisir le contenu de *n* au préalable.

**Corrigé**

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

```
int n;

int Modif ()    {
return --n ;
}

int main() {
int a;

printf("Valeur de n ? : \n");
scanf("%d",&n);
printf("Valeur de n saisie : %d\n",n);
printf("Valeur renvoyee apres execution de la fontion Modif : %d\n",Modif());
printf("Valeur de n apres execution de la fonction Modif : %d\n",n);

return 0;
}
```

**Rq**

On n'envoie aucun paramètre à *Modif().* Comme *n* est une variable globale, *Modif()* connaît *n*. La ligne *return -- n;* décrémente le contenu de la variable globale *n* et renvoie son contenu. Le programme principal récupère le contrôle et affiche le contenu de *n* qui a été modifié par *Modif().*

```
Valeur de n ? :
3
Valeur de n saisie : 3
Valeur renvoyee apres execution de la fontion Modif : 2
Valeur de n apres execution de la fonction Modif : 2
```

*Exemple d'exécution*

## Exercice 19

**Énoncé**

Qu'affiche le programme suivant si l'on saisit les valeurs 10 et 101 ?

```
# include <stdio.h>
int a, b;

int Somme () {
return (--a + b--);
}

int main() {
printf("Valeur de a ? \n ");
scanf("%d", &a);
```

```
printf("Valeur de b ? \n ");
scanf("%d", &b);
printf("La somme : %d\n ",Somme());

printf("a : %d\n ",a);
printf("b : %d\n ",b);

return 0;
}
```

**Corrigé**

```
Valeur de a ?
 10
Valeur de b ?
 101
La somme : 110
 a : 9
 b : 100
```

Variables globales ! La fonction *Modif()* a modifié la valeur de *a* (qui passe à 9) ; retourne la somme de a et *b (9 +101) puis décrémente b qui passe à 100.* Le *main()* voit les valeurs de *a* et *b* modifiées (9 et 100).

## Exercice 20

**Énoncé**

Créez des programmes avec des fonctions ; des variables globales et locales pour bien voir leur portée.