TP Programmation

Corrigé de la série de TP N°4 – Procédures et fonctions

Exercice N°01 :

Soit le programme C suivant :

```
#include <stdio.h>

void sous_prog1(float x, float y, float s) {
   s = x + y;
}
void sous_prog2(float x, float y, float* s) {
   *s = x + y;
}
//Début du programme principal
int main() {
   float a, b, c ; //Variables globales du programme

   a = 10; b = 5; c = 0;
   sous_prog1(a, b, c);
   printf("La somme est : %.2f \n", c);

   a = 10; b = 5; c = 0;
   sous_prog2(a, b, &c);
   printf("La somme est : %.2f \n", c);

   return 0;
} //Fin du programme principal
```

Questions :

1) Exécuter le programme.
2) Quelle est la différence entre les deux procédures sous_prog1 et sous_prog2 ?
3) Quels sont les paramètres à passage par valeur et ceux à passage par variable ?
4) Quels sont les paramètres formels des deux procédures ?
5) Quels sont les paramètres effectifs ?
6) Dérouler le programme.
7) Exécuter le programme en donnant le type « int » à la variable c. Que se passe-t-il ? pourquoi ?

Solution :

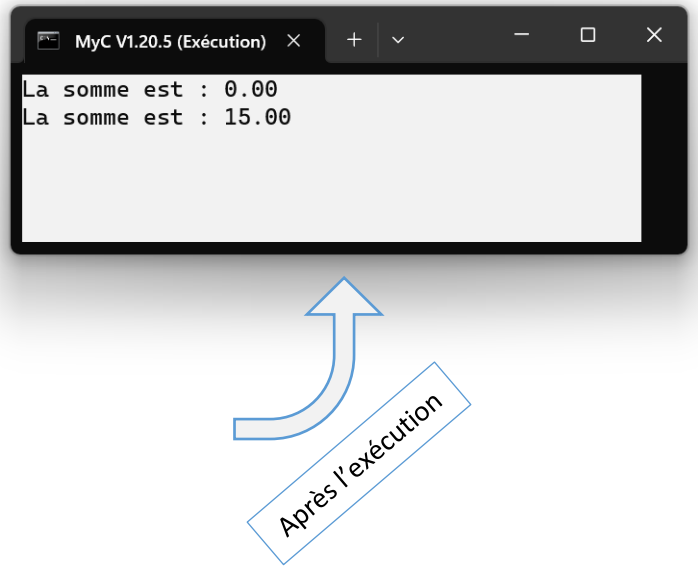
Explication :

Le programme comporte deux sous-programmes **sous_prog1** et **sous_prog2** qui ont pour objectif de calculer la somme de deux nombres de type **float** passés en paramètres et de stocker le résultat dans une variable **s** également de type **float**.

Le programme principal utilise ces deux sous-programmes pour calculer la somme de deux nombres **a** et **b** et stocker le résultat dans une variable **c**, en appelant d'abord **sous_prog1**, puis **sous_prog2**.

1. Exécuter le programme.

```c
#include <stdio.h>

void sous_prog1(float x, float y, float s) {
    s = x + y;
}
void sous_prog2(float x, float y, float* s) {
    *s = x + y;
}
//Début du programme principal
int main() {
    float a, b, c ; //Variables globales du programme

    a = 10; b = 5; c = 0;
    sous_prog1(a, b, c);
    printf("La somme est : %.2f \n", c);

    a = 10; b = 5; c = 0;
    sous_prog2(a, b, &c);
    printf("La somme est : %.2f \n", c);

    return 0;
} //Fin du programme principal
```



2. C'est quoi la différence entre les deux procédures *sous_prog1* et *sous_prog2* ?

La différence entre **sous_prog1** et **sous_prog2** est que **sous_prog1** utilise un paramètre de type variable (float * s) pour modifier directement la valeur de la variable **s**, tandis que **sous_prog2** utilise un paramètre de type valeur (float s), ce qui signifie que la modification de **s** à l'intérieur de la procédure ne se propage pas à la variable **c** dans le programme principal.

3. Quels sont les paramètres à passage par valeur et ceux à passage par variable ?

Les paramètres à passage par valeur et ceux à passage par variable :

Paramètre/Procédure	**Procédure 1**	**Procédure 2**
paramètres à passage par valeur	x, y et s	x et y
paramètres à passage par variable	Aucun paramètre	s

4. Quels sont les paramètres formels des deux procédures ?

Les paramètres formels des deux procédures :

Paramètre/Procédure	**Procédure 1**	**Procédure 2**
Paramètres formels	x, y et s	x, y et s

5. Quels sont les paramètres effectifs ?

Les paramètres effectifs des deux procédures :

Paramètre/Procédure	**Procédure 1**	**Procédure 2**
Paramètres effectifs	a, b et c	a, b et &c

Rappel 1 :

Les *paramètres formels* sont les paramètres utilisés dans la déclaration des procédures et fonctions. Par contre, les *paramètres effectifs* sont les paramètres utilisés lors de l'appel aux procédures et fonctions. (Les paramètres formels sont séparés par des *virgules*).

6. Dérouler le programme

Instructions	Programme Principal			Procédure Proc1			Procédure Proc2			Affichage
	a	***b***	***c***	***x***	***y***	***s***	***x***	***y***	***s (var)***	
a=10;b=5; c=0;	10	5	0							
sous_prog1(a, b, c); *(L'appel à Proc1)* *=> La trasmission des paramètres* s=x+y ; *(La valeur de s dans Proc1 n'a pas été retournée à la variable globale c)*	"	"	"	10	5	0 **15**				
printf("La somme est : %.2f \n", c);	10	5	**0**							La somme est : 0.00
a:=10; b:=5; c:=0;	10	5	0							
sous_prog2(a, b, &c); *(L'appel à Proc2)* *=> La trasmission des paramètres* s=x+y ; *(La valeur de s dans Proc2 a été retournée à la variable globale c)*			**15**				10	5	0 **15**	
printf("La somme est : %.2f \n", c);	10	5	**15**							La somme est : 15.00

7. Exécuter le programme en donnant le type « int » à la variable c. Que se passe-t-il ? pourquoi ?

Si on exécute le programme en donnant le type "int" à la variable **c**, le compilateur va émettre un avertissement (warning) car on passe un paramètre par variable (adresse) vers un type **float** à une fonction qui attend un paramètre par variable (adresse) vers un type **int**. Cela peut provoquer un comportement indéfini car le programme peut tenter de lire ou d'écrire dans une zone mémoire qui ne lui appartient pas.

Il est important de respecter le type des variables lors de l'appel de fonctions, en particulier lorsqu'on utilise des paramètres à passage par variables.

Exercice N°02 :

Ecrire une procédure qui prend deux entiers en entrée et qui échange leurs valeurs.

Solution :

Voici une procédure qui échange les valeurs de deux entiers passés en entrée :

```
void echanger(int* a, int* b) {
    int temp ;
    temp= *a;
    *a = *b;
    *b = temp;
}
```

La procédure **echanger** prend en entrée deux paramètres à passage par adresse d'entiers **a** et **b**. Elle utilise une variable temporaire **temp** pour stocker la valeur de **a**, puis elle affecte à **a** la valeur de **b**, et à **b** la valeur de **temp**.

Pour appeler cette procédure, il suffit de passer les adresses des variables que l'on souhaite échanger. Par exemple :

```
int x = 5, y = 10;
printf("Avant l'echange : x=%d, y=%d\n", x, y);
echanger(&x, &y);
printf("Apres l'echange : x=%d, y=%d\n", x, y);
```

Cet exemple crée deux variables **x** et **y**, et les affiche avant et après l'appel à la procédure **echanger**. On remarque que les valeurs ont été échangées.

Exercice N°03 :

Soit le programme C suivant :

```
#include<stdio.h>
int maximum(int a, int b)
{
if (a>b)
  return a ;
else
  return b ;
}
int main()  //Début du programme principal
  {
  int x, y, results ;   //Variables globales du programme
  printf("Donner la première valeur : ") ;
  scanf("%d", &x) ;
  printf("Donner la deuxième valeur : ") ;
  scanf("%d", &y) ;
  results= maximum (x, y) ;
  printf("Le maximum entre %d et %d est : %d \n ", x, y,
  results) ;
  return 0 ;
}   //Fin du programme principal
```

Questions :

1) Exécuter le programme pour x = 5 et y = 3.
2) Dérouler le programme pour x = 5 et y = 3.
3) Quels sont les paramètres formels da la fonction ?
4) Quels sont les paramètres effectifs ?
5) Réécrire le programme pour déterminer le maximum entre trois nombres entiers x, y et z.

Solution :

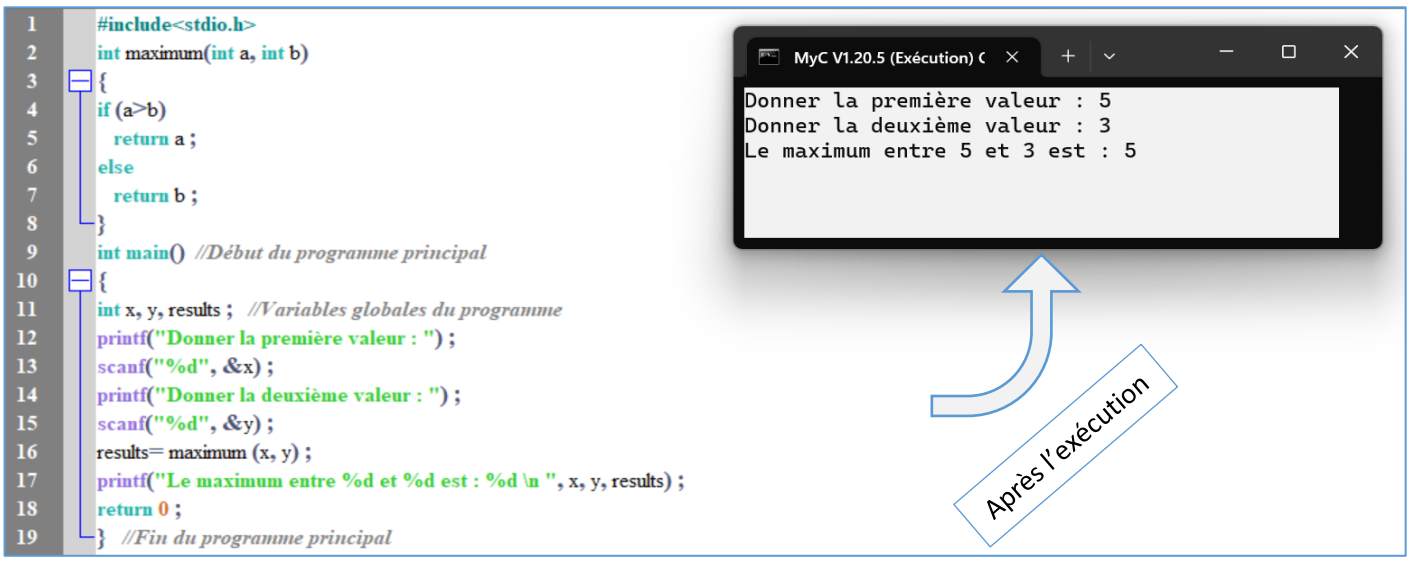
Explication :

Le programme comporte une fonction **maximum** qui prend deux nombres entiers **a** et **b** en entrée, compare les deux valeurs et retourne le maximum des deux.

Le programme principal utilise cette fonction pour déterminer le maximum entre deux nombres entiers saisis par l'utilisateur à l'aide de la fonction **scanf**. Le programme vérifie également que les entrées sont des entiers valides avant de les utiliser dans le programme. Si une entrée n'est pas valide, le programme affiche un message d'erreur et demande à l'utilisateur de saisir une nouvelle entrée.

NB : ce programme ne gère pas le cas où **a** et **b** sont égaux.

1) Exécuter le programme pour x = 5 et y = 3.



2) Dérouler le programme pour a = 5 et b = 3.

Instructions	Programme principal			La fonction maximum			Affichage
	x	y	results	a	b	maximum	
printf("Donner la première valeur : ");							Donner la première valeur :
scanf("%d", &x)	5						
printf("Donner la deuxième valeur : ");							Donner la deuxième valeur :
scanf("%d", &y)		3					
resultls = maximum (x,y);							
maximum (x,y); *(l'appel à la fonction maximum avec x=5 et y=3)*							
=> La transmission des paramètres If(a>b) then → 5>3 vrai return a; *=> Le retour du résultat dans le prog principal*			5	5	3	5	
printf("Le maximum entre %d et %d est : %d \n ",x,y,results);	5	3	5				Le maximum entre 5 et 3 est : 5

3) Quels sont les paramètres formels da la fonction ?

Les paramètres formels de la fonction sont : **a**, **b** et **maximum**

4) Quels sont les paramètres effectifs ?

Les paramètres effectifs sont : **x**, **y** et **results**

5) Réécrire le programme pour déterminer le maximum entre trois nombres entiers x, y et z.

```
#include <stdio.h>
int maximum(int a, int b)
{
if (a>b)
  return a ;
else
  return b ;
}

int main() {
    int x, y, z, Max_xy, Max_xyz;
    printf("Entrez trois nombres réels x, y et z : ");
    scanf("%d %d %d", &x, &y, &z);
    Max_xy = maximum(x, y);
    Max_xyz = maximum(Max_xy, z);
    printf("Le maximum entre %d, %d et %d est %d.", x, y, z, Max_xyz);
    return 0;
}
```

```c
#include <stdio.h>
int maximum(int a, int b)
{
if (a>b)
  return a ;
else
  return b ;
}

int main() {
   int x, y, z, Max_xy, Max_xyz;
   printf("Entrez trois nombres réels x, y et z : ");
   scanf("%d %d %d", &x, &y, &z);
   Max_xy = maximum(x, y);
   Max_xyz = maximum(Max_xy, z);
   printf("Le maximum entre %d, %d et %d est %d.", x, y, z, Max_xyz);
   return 0;
}
```

Ce programme demande à l'utilisateur d'entrer trois nombres réels x, y et z, puis utilise la fonction « maximum » pour calculer le maximum entre x et y, et le maximum entre le résultat précédent et z. Finalement, le programme affiche le maximum des trois nombres entiers.

Exercice N°04 :

Ecrire une fonction qui calcule la somme de deux nombres entiers passés en paramètres et retourne le résultat. Ecrire un programme qui utilise cette fonction pour calculer et afficher la somme de deux nombres saisis par l'utilisateur.

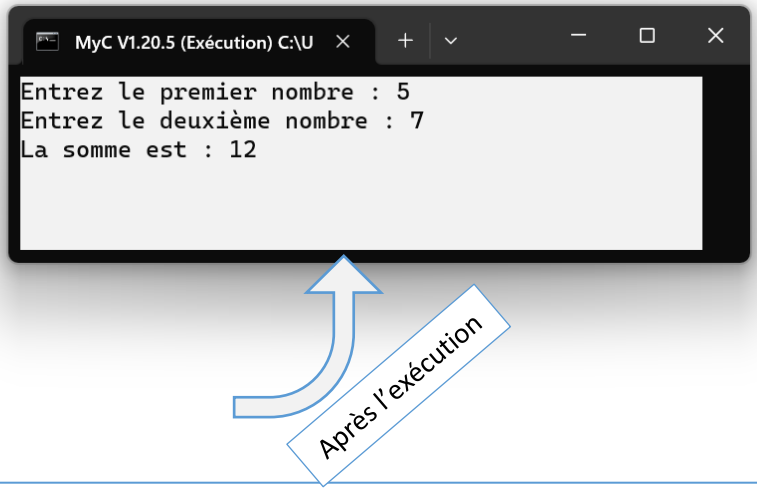
Solution :

```c
#include <stdio.h>

int somme(int a, int b){
 return a+b;
}
int main(){
    int num1, num2, resultat;
    printf("Entrez le premier nombre : ");
    scanf("%d", &num1);
    printf("Entrez le deuxième nombre : ");
    scanf("%d", &num2);
    resultat = somme(num1, num2);
    printf("La somme est : %d\n", resultat);
    return 0;
}
```

Une autre méthode :

```c
int somme(int a, int b){
 int S;
 S=a+b;
}
```

```c
#include <stdio.h>

int somme(int a, int b){
 return a+b;
}
int main(){
   int num1, num2, resultat;
   printf("Entrez le premier nombre : ");
   scanf("%d", &num1);
   printf("Entrez le deuxième nombre : ");
   scanf("%d", &num2);
   resultat = somme(num1, num2);
   printf("La somme est : %d\n", resultat);
   return 0;
}
```



Exercice N°05 :

Soit le programme C suivant :

```c
#include <stdio.h>
int Fact(int m) {
   int j, f;
   f= 1;
   for(j=1; j<= m; j++) {
      f =f* j;
   }
   return f;
}

int main()   //Début du programme principal
  {
   int i, n; float S;  //Variables globales du programme
   printf("Introduire n : ");
   scanf("%d", &n);
   S=0;
   for (i =1; i<= n; i++) {
      S =S+ Fact(i);
   }
   printf("La somme = %.2f", S);
   return 0;
}  //Fin du programme principal
```

1) Exécuter le programme pour n = 3.
2) Dérouler le programme pour n = 3.
3) Réécrire le programme en transformant la fonction Fact en une procédure Fact.
4) Soit la procédure « Puiss » suivante :

```c
void Puiss(float t, int k, float* P) {
  int i;
      *P=1;
      for(i = 1; i <= k; i++) {
         *P =*P*t;
      }
   }
```

4-a) Dérouler la procédure Puiss pour t=2 et k=4 et déduire ce qu'elle fait.

4-b) Transformer la procédure Puiss en une fonction.

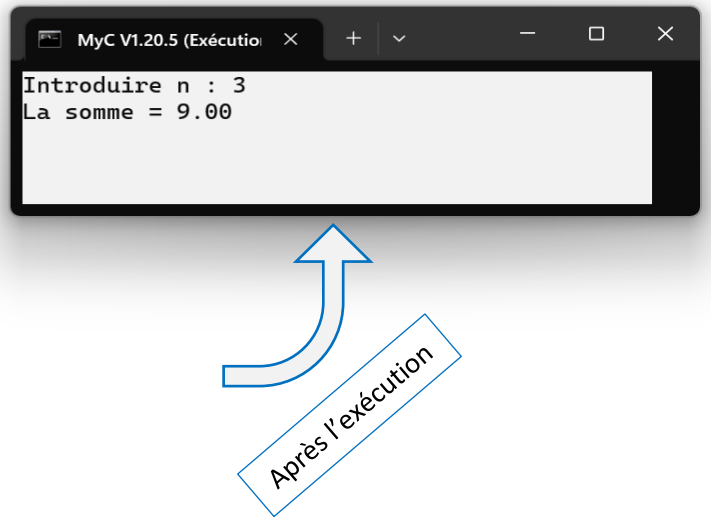
Solution :

Explication :

- Le programme commence par inclure la bibliothèque stdio.h pour utiliser les fonctions d'entrée/sortie standard.
- La fonction Fact est définie pour calculer la factorielle d'un nombre entier m. Elle utilise une boucle for pour calculer la factorielle en multipliant tous les nombres entre 1 et m.
- La fonction main est définie comme le programme principal. Elle commence par déclarer les variables entières i et n, ainsi que la variable flottante S initialisée à 0.
- Le programme demande à l'utilisateur d'entrer la valeur de n à l'aide de la fonction scanf.
- Le programme utilise une boucle for pour calculer la somme des factorielles de 1 à n et stocke le résultat dans la variable S.
- Enfin, le programme affiche le résultat avec la fonction printf en formatant le nombre à 2 décimales. Le programme se termine avec la valeur de retour 0.

1. Exécuter le programme pour n = 3.

```c
#include <stdio.h>
int Fact(int m) {
    int j, f;
    f= 1;
    for(j=1; j<= m; j++) {
        f =f* j;
    }
    return f;
}

int main() {
    int i, n;
    float S;
    printf("Introduire n : ");
    scanf("%d", &n);
    S=0;
    for (i =1; i<= n; i++) {
        S =S+ Fact(i);
    }
    printf("La somme = %.2f", S);
    return 0;
}
```



2. Dérouler le programme pour n = 3.

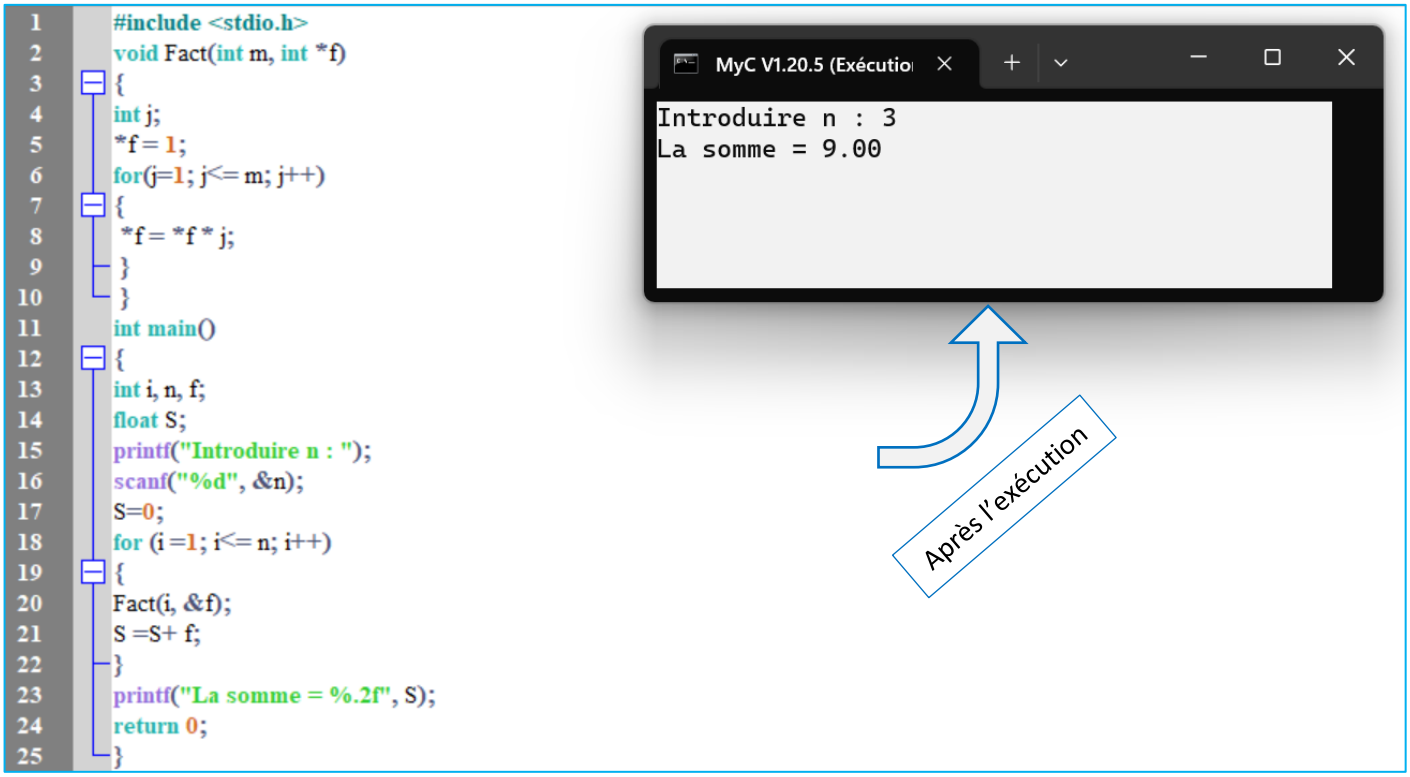
Instructions	Programme principal			La fonction Fact				Affichage
	n	i	S	m	j	f	Fact	
printf("Introduire n : ");								Introduire n :
scanf("%d", &n)	3							
S=0			0					
For i=1 S :=S+Fact(i)		1	0+?					
Fact(i) *(l'appel à Fact avec le paramètre i=1)*		1						
=> La transmission des paramètres f=1 for j=1 f=f*i → f=1*1 = 1 return f; *=> Le retour du résultat dans le prog principal*			0+**1**=**1**	**1**	1	1 1	**1**	
For i=2 S=S+Fact(i)		2	1+?					
Fact(i) *(l'appel à Fact avec le paramètre i=2)*		2						
=> La transmission des paramètres f:=1 for j=1 f=f*i → f=1*1 = 1 for j=2 f=f*i → f=1*2 = 2 return f ; *=> Le retour du résultat dans le prog principal*			1+**2**=**3**	**2**	1 2	1 1 2	**2**	
For i=3 S=S+Fact(i)		3	3+?					
Fact(i) *(l'appel à Fact avec le paramètre i=3)*		3						
=> La transmission des paramètres f=1 for j=1 f=f*i → f=1*1 = 1 for j=2 f=f*i → f=1*2 = 2 for j=3 f=f*i → f=1*2*3 = 6 return f; *=> Le retour du résultat dans le prog principal*			3+**6**=**9**	**3**	1 2 3	1 1 2 6	**6**	
printf("La somme = %.2f ",S);								La somme =9.00

3. Réécrire le programme en transformant la fonction Fact en une procédure Fact.

```
void Fact(int m, int *f)
{
int j;  //Variables locales de la procédure Fact
*f = 1;
for(j=1; j<= m; j++)
{
 *f = *f * j;
 }
 }
```

Insertion de la procédure dans le programme :

```
#include <stdio.h>
void Fact(int m, int *f)
{
int j;
*f = 1;
for(j=1; j<= m; j++)
{
 *f = *f * j;
 }
 }
int main()
{
int i, n, f;
float S;
printf("Introduire n : ");
scanf("%d", &n);
S=0;
for (i =1; i<= n; i++)
{
Fact(i, &f);
S =S+ f;
}
printf("La somme = %.2f", S);
return 0;
}
```



4-a) Dérouler la procédure Puiss pour t=2 et k=4 et déduire ce qu'elle fait.

```
void Puiss(float t, int k, float* P) {
   int i;
        *P=1;
        for(i = 1; i <= k; i++) {
           *P =*P*t;
        }
}
```

Instruction	Variables de la procédure Puiss			
	t	**k**	**i**	**P**
{passage de paramètres par valeur}	2	4	-	-
P:=1;				1
For i:=1			1	
P:=P*t;				1*2=2
For i:=2			2	
P:=P*t;				$2*2=2^2$
For i:=3				
P:=P*t;			3	
For i:=4				$2^2*2=2^3$
P:=P*t;			4	
				$2^{3*}2=2^4$

D'après le déroulement, notamment dans la relation entre l'expression finale de P et les paramètres d'entrées *t* et *k*, on déduit que la procédure **Puiss** permet de calculer la puissance d'un nombre *t* par *k*, c'est-à-dire : $\mathbf{t^k}$.

4-b) Transformer la procédure Puiss en une fonction.

```
void Puiss(float t, int k, float* P) {
   int i;
        *P=1;
        for(i = 1; i <= k; i++) {
           *P =*P*t;
        }
                }
```

```
float puiss(float t, int k) {   //Suppression du paramètre résultat P et l'ajout de type du résultat renvoyé
   float P; int i;   //Ajout de la variable locale P
   P=1;
   for (int i=1; i <= k; i++) {
      P= P*t;
   }
   return P;  //Ajout l'instruction de retour du résultat P dans la fonction Puiss
       }
```