TP - Programmation

Corrigé de la Série de TP N°3 – Sous-Programmes : Procédures et Fonctions

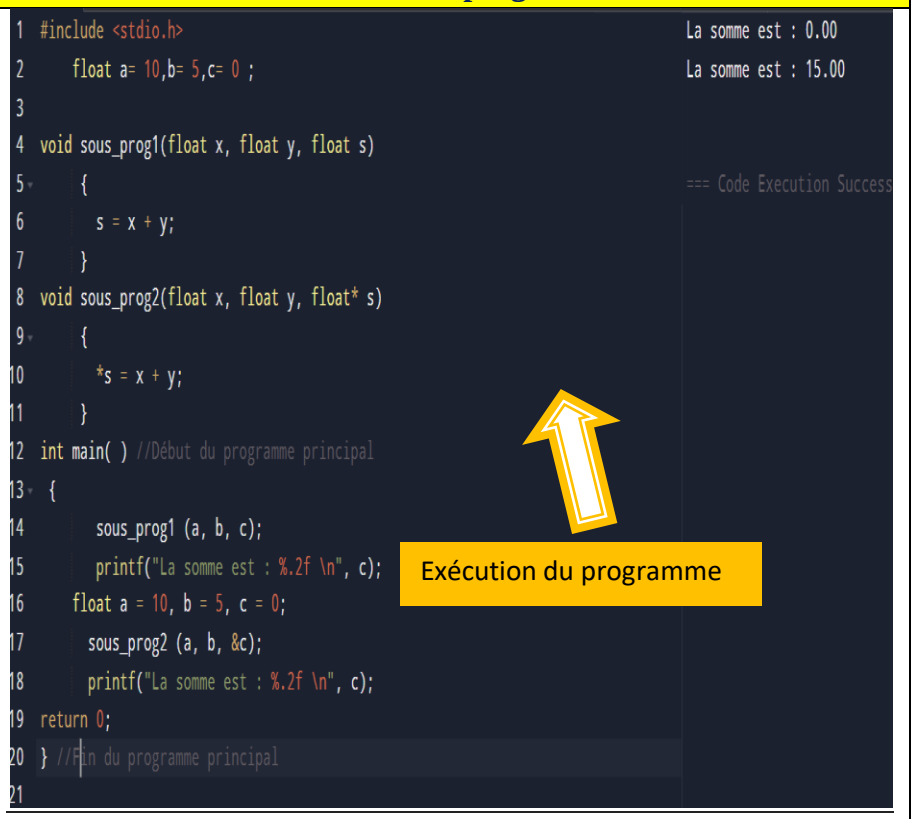
Exercice N°1 :

1) Exécuter le programme:

Programme

```
#include <stdio.h>
float a= 10 , b= 5 , c= 0 ;
void sous_prog1(float x, float y, float s)
    {
     s = x + y;
    }
void sous_prog2(float x, float y, float* s)
    {
     *s = x + y;
    }
int main( )
 {
     sous_prog1 (a, b, c);
     printf("La somme est : %.2f \n", c);
   float a = 10; b = 5; c = 0;
    sous_prog2 (a, b, &c);
    printf("La somme est : %.2f \n", c);
return 0;
}
```

Exécution du programme



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** d'abord **sous_prog1** , puis **sous_prog2** .

2) Quelle est la différence entre les deux procédures sous_prog1 et sous_prog2 ?

La différence entre **sous_prog1** et **sous_prog2** réside dans la manière dont le paramètre **s** est passé.

- **sous_prog1** utilise un paramètre de type (**float s**) passé **par valeur**, ce qui signifie que toute modification de s à l'intérieur de la fonction n'affecte pas la variable **c** dans le programme principal. En effet, une copie de la valeur de **s** est utilisée dans la fonction, et toute modification reste locale à cette fonction.
- **sous_prog2**, quant à elle, utilise un paramètre de type (**float* s**) passé **par référence** (ou **via un pointeur**), ce qui permet de modifier directement la variable **c** dans le programme principal. Ici, l'adresse mémoire de la variable **c** est transmise à la fonction, et toute modification de ***s** affecte directement la variable **c** dans main.

3) **Quels sont 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`, `s`	`x`, `y`
Paramètres à passage par variable	Aucun paramètres	`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`, `s`	`x`, `y`, `s`

5) **Quels sont les paramètres effectifs ?**

Les **paramètres effectifs** des deux procédures :

Paramètre/Procédure	Procédure 1	Procédure 2
Paramètres effectifs	`a`, `b`, `c`	`a`, `b`, `c`

Rappel :

Les **paramètres formels** sont les paramètres utilisés dans la **déclaration des procédures** et **fonctions**. Ce sont les variables qui apparaissent dans la signature de la procédure ou de la fonction, et qui servent à définir leur structure interne.

En revanche, les **paramètres effectifs** sont les valeurs ou les variables réelles passées lors de l'appel de ces procédures et fonctions. Ces valeurs sont transmises aux paramètres formels pour permettre l'exécution de la procédure ou de la fonction.

6) **Quels sont les variables locales et globales dans le programme**

➢ **Variables globales :**

Les **variables globales** sont déclarées en dehors de toute fonction, ce qui signifie qu'elles peuvent être accessibles dans toutes les fonctions du programme, y compris la fonction main, **sous_prog1** et **sous_prog2**. Dans le programme, les variables suivantes sont globales :

- **float a = 10, b = 5, c = 0;** (déclaration globale en dehors de main et des autres fonctions).

Ces variables sont définies avant la fonction main, elles sont donc visibles et accessibles dans tout le programme, y compris dans les sous-programmes (fonctions **sous_prog1** et **sous_prog2**).

➢ **Variables locales :**

Les **variables locales** sont déclarées à l'intérieur d'une fonction et ne sont accessibles qu'à partir de cette fonction. Elles ne sont pas visibles ni modifiables dans d'autres fonctions. Voici les variables locales dans le programme :

❖ **Dans la fonction main :**
- **float a = 10, b = 5, c = 0;** (Ces variables sont locales à la fonction **main**).

❖ **Dans la fonction sous_prog1 :**
- float x, y, s; (Variables locales à la fonction **sous_prog1**).

❖ **Dans la fonction sous_prog2 :**

- **float x, y;** (Variables locales à la fonction **sous_prog2**).
- **float* s;** (Pointeur local à la fonction **sous_prog2**).

7) 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 transmission 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

8) 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 de type **float** à une fonction qui attend un paramètre de **type int**. Cela peut provoquer un comportement **indéfini,** car le programme pourrait 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 des appels de fonctions, en particulier lorsqu'on utilise des paramètres passés par adresse (passage par référence).

Exercice N°2 :

Programme en C

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

Solution :

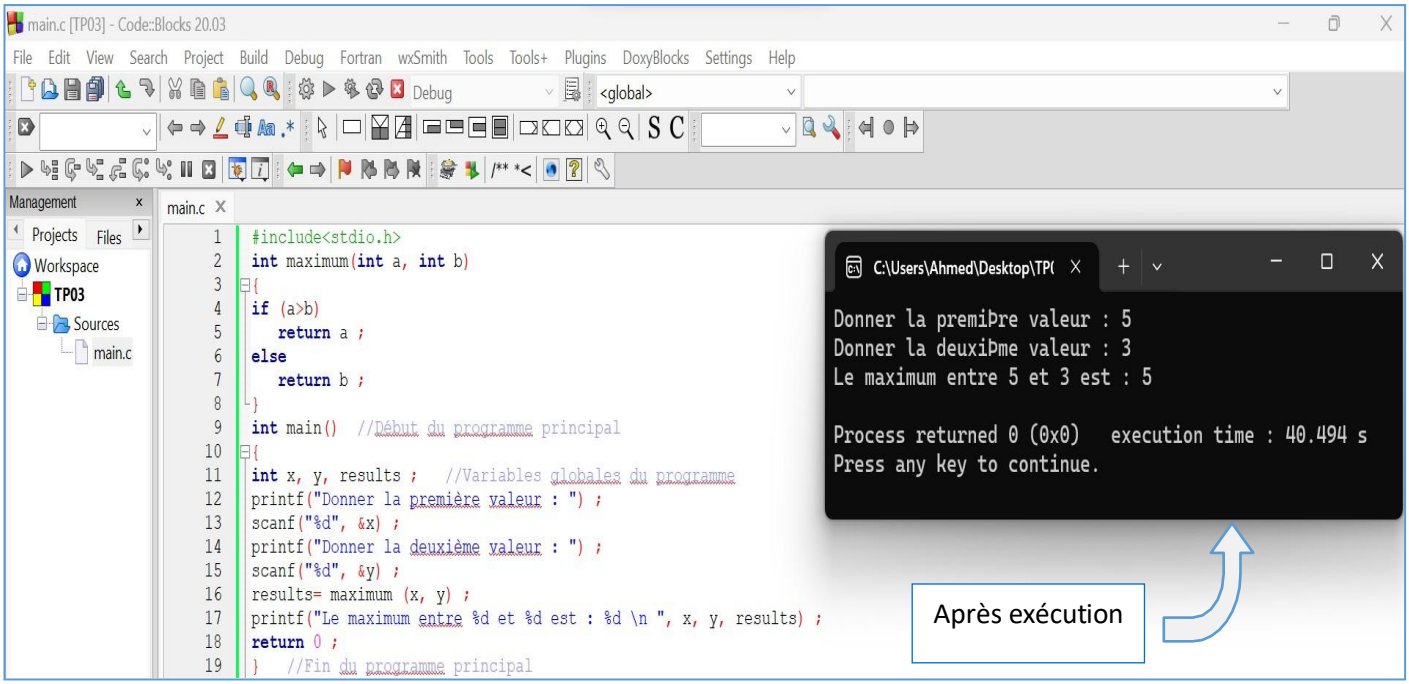
Explication ☺ :

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);			results=?				
maximum (x,y); *(l'appel à la fonction maximum avec x=5 et y=3)*							
=> *La transmission des paramètres* If(a>b) → 5>3 vrai return a; => *La valeur de maximum est retournée à la variable globale results*				5	3	5	
resultls = maximum (x,y); results = 5			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?

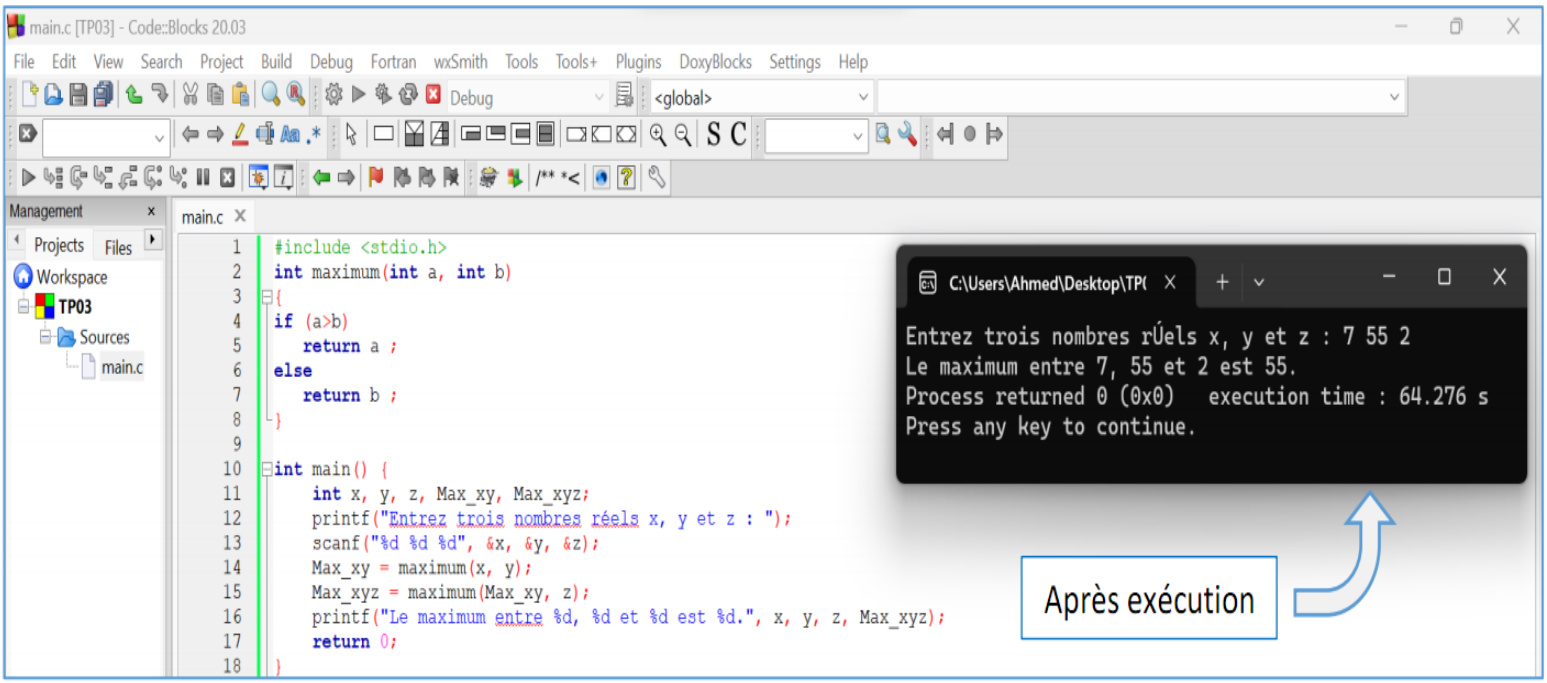
Les paramètres formels de la fonction sont: **a**,**b**

4) Quels sont les paramètres effectifs?

Les parameters 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 trios nombres reels 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 entiers 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°3 :

Écrire un programme en C qui lit les 10 valeurs entières d'un tableau à une dimension, comportant une procédure permet d'ordonner le tableau dans l'ordre croissant.

Programme en langage C

```
#include<stdio.h>
// Procédure pour trier le tableau en ordre croissant (tri par sélection)
void croissant (int T[ ], int N)
{
      int i, j, temp;

      for (i = 0;  i<N;  i++)
        {
              for (j = i + 1; j <N; j++)
             {
                 if (T[i] >T[j])
                   {
                   temp = T[i];
                   T[i] = T[j];
                   T[j] = temp;
                    }
             }
       }
}
int main( )
  {
    int T[10], i;
```

```
    printf("Entrez 10 valeurs pour le tableau:\n");
   // Lecture des 10 valeurs
   for (i = 0;  i< 10;  i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &T[i]);
     }
// Appel de la procédure de tri
      croissant(T, 10);

// Affichage du tableau trié
   printf("\nTableau trié en ordre croissant:\n");
     for (i = 0; i< 10; i++) {
         printf("%d  ", T[i]);
      }
   return 0;
}
```

La procédure **croissant** permet de comparer les éléments du vecteur T et les ordonner du plus petit au plus grand élément en utilisant la variable temporaire **temp.**

Compilation et exécution du programme

```
1  #include<stdio.h>
2  // Procédure pour trier le tableau en ordre croissant (tri par sélection)
3  void croissant (int T[ ], int N)
4  {
5         int i, j, temp;
6
7         for (i = 0;  i<N-1;  i++)
8            {
9                   for (j = i + 1; j <N; j++)
10                 {
11                      if (T[i] >T[j])
12                         {
13                         temp = T[i];
14                         T[i] = T[j];
15                         T[j] = temp;
16                          }}}}
17 int main( )
18    {
19      int T[10], i;
20      printf("Entrez 10 valeurs pour le tableau:\n");
21     // Lecture des 10 valeurs
22     for (i = 0;  i< 10;  i++) {
23            printf("Element %d: ", i + 1);
24            scanf("%d", &T[i]);
25         }
26 // Appel de la procédure de tri
27         croissant(T, 10);
28
29 // Affichage du tableau trié
30     printf("\nTableau trié en ordre croissant:\n");
31        for (i = 0; i< 10; i++) {
32              printf("%d  ", T[i]);
33         }
34     return 0;
35 }
36
```

```
C:\Users\Maison\Desktop\lhklklglgljgjgj\FONC\bin\D...
Entrez 10 valeurs pour le tableau:
Element 1: 3
Element 2: 4
Element 3: 2
Element 4: 5
Element 5: 6
Element 6: 7
Element 7: 8
Element 8: 9
Element 9: 0
Element 10: 1

Tableau triÚ en ordre croissant:
0  1  2  3  4  5  6  7  8  9
Process returned 0 (0x0)   execution time : 17.365 s
Press any key to continue.
```

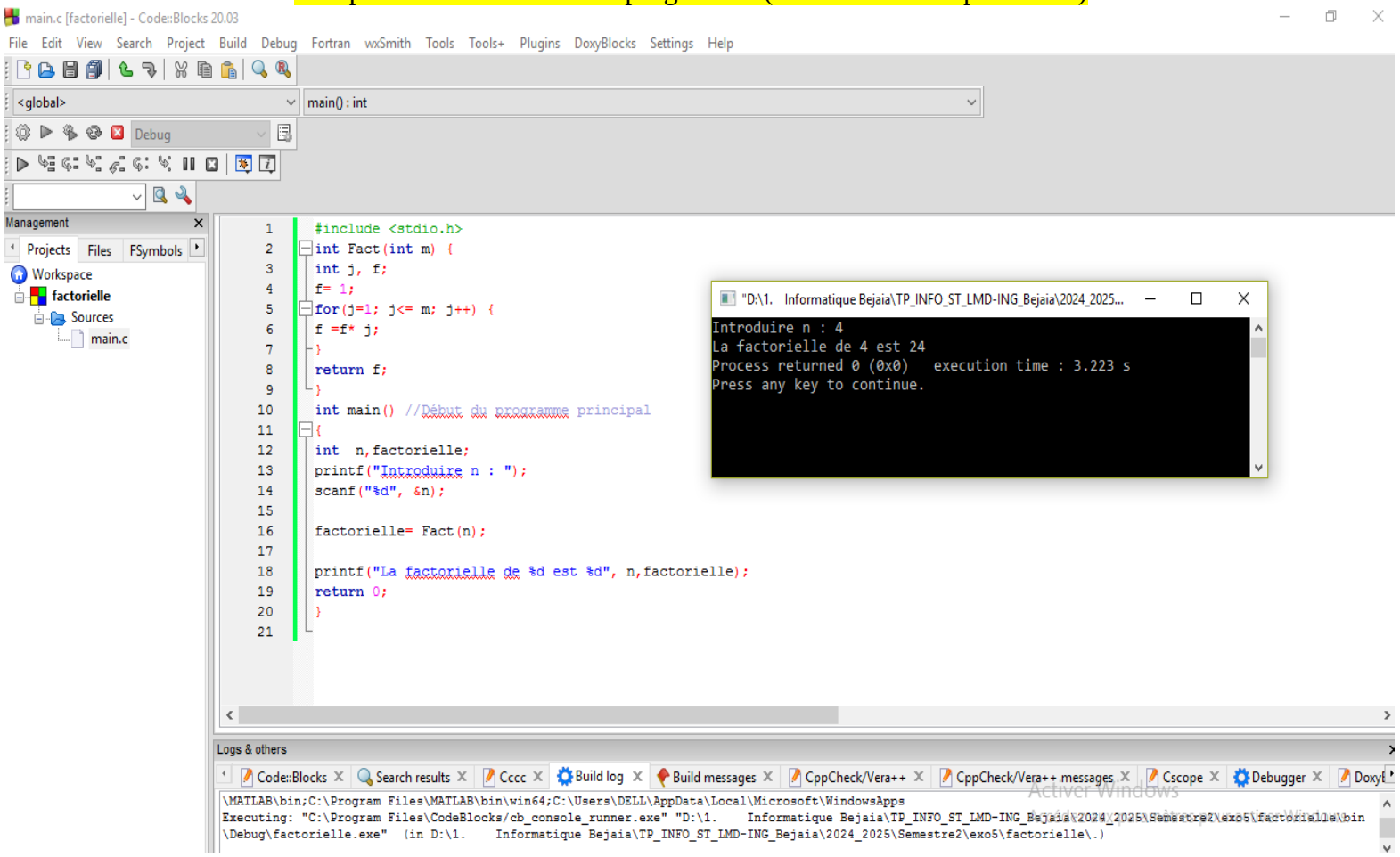
Exercice N°4 :

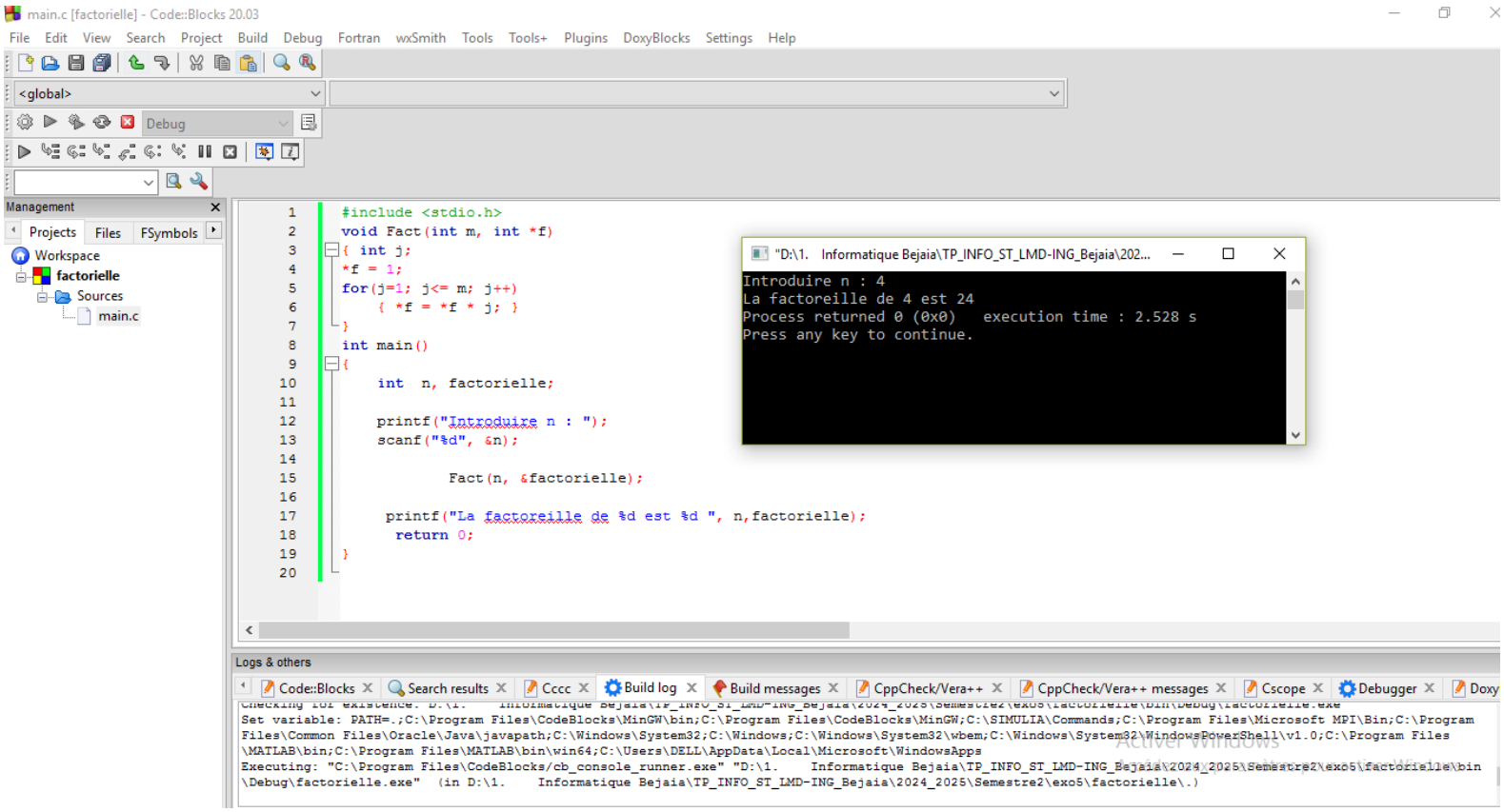
Ecrire un programme en C qui permet de calculer N! , en utilisant la fonction fact. Ensuite réécrire le programme en remplaçant la fonction par une procédure.

Programme avec une fonction	Programme avec une procédure

`#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 n,factorielle;`
`printf("Introduire n : ");`
`scanf("%d", &n);`
`factorielle= Fact(n);`
`printf("La factorielle de %d est %d", n,factorielle);`
`return 0;`
`}`
	`#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 n, factorielle;`
	`printf("Introduire n : ");`
	`scanf("%d", &n);`
	`Fact(n, &factorielle);`
	`printf("La factorielle de %d est %d ", n,factorielle);`
	`return 0;`
	`}`

Compilation et exécution du programme (avec fonction & procédure)




main.c [factorielle] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Files FSymbols
Workspace
factorielle
Sources
main.c
#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 n, factorielle;
printf("Introduire n : ");
scanf("%d", &n);
Fact(n, &factorielle);
printf("La factoreille de %d est %d ", n,factorielle);
return 0;
}
"D:\1. Informatique Bejaia\TP_INFO_ST_LMD-ING_Bejaia\202...
Introduire n : 4
La factoreille de 4 est 24
Process returned 0 (0x0) execution time : 2.528 s
Press any key to continue.
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugger
Set variable: PATH=.;C:\Program Files\CodeBlocks\MinGW\bin;C:\Program Files\CodeBlocks\MinGW;C:\SIMULIA\Commands;C:\Program Files\Microsoft MPI\Bin;C:\Program Files\Common Files\Oracle\Java\javapath;C:\Windows\System32;C:\Windows;C:\Windows\System32\wbem;C:\Windows\System32\WindowsPowerShell\v1.0;C:\Program Files\MATLAB\bin;C:\Program Files\MATLAB\bin\win64;C:\Users\DELL\AppData\Local\Microsoft\WindowsApps
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\1. Informatique Bejaia\TP_INFO_ST_LMD-ING_Bejaia\2024_2025\Semestre2\exo5\factorielle\bin\Debug\factorielle.exe" (in D:\1. Informatique Bejaia\TP_INFO_ST_LMD-ING_Bejaia\2024_2025\Semestre2\exo5\factorielle\.)