



SAGIM 2024/2025

Langage C

Les fonctions & procédures

Résumé avec des exercices d'explication +



LES FONCTIONS

Fonctions en C

Une fonction est **un bloc** de code nommé qui effectue une **tâche** spécifique et **retourne** une valeur.

Déclaration d'une fonction

Une fonction en C se définit ainsi :

```
type_retour nom_fonction(paramètres) {  
    // Corps de la fonction  
    return valeur;  
}
```

Exemple de fonction

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

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

```
int main() {  
    int x = 5, y = 10;  
    int resultat = somme(x, y);  
    printf("La somme de %d et %d est : %d\n", x, y, resultat);  
    return 0;  
}
```

Exercice 1

Énoncé :

Écrire une fonction carre qui prend un entier en paramètre et retourne son carré.

SOLUTION

```
#include <stdio.h>
int carre(int n) {
    return n * n;
}
```

```
int main() {
    int nombre;
    printf("Entrez un nombre : ");
    scanf("%d", &nombre);
    printf("Le carré de %d est : %d\n", nombre, carre(nombre));
    return 0;
}
```



LES PROCEDURES

Procédures en C

Une procédure en C est une fonction qui **ne retourne pas** de valeur.

Elle est déclarée avec **void** comme type de retour.

Exemple de procédure

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

```
void afficherMessage() {  
    printf("Bienvenue dans le cours de C !\n");  
}
```

```
int main() {  
    afficherMessage(); // Appel de la procédure  
    return 0;  
}
```

Exercice 2

Énoncé :

Écrire une procédure `afficherTable()` qui prend un entier et affiche sa table de multiplication jusqu'à 10.

Solution

```
#include <stdio.h>
void afficherTable(int n) {
    printf("Table de multiplication de %d :\n", n);
    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", n, i, n * i); } }
int main() {
    int nombre;
    printf("Entrez un nombre : ");
    scanf("%d", &nombre);
    afficherTable(nombre);
    return 0; }
```

Résumé

- **Une fonction** retourne **une valeur** et peut être utilisée dans des expressions.
- **Une procédure** (fonction void) exécute une action mais **ne retourne pas** de valeur.
- Les fonctions permettent une meilleure **organisation** du code et facilitent la réutilisation.



Passage par Valeur & Passage par Adresse

Passage par Valeur

Dans le passage **par valeur**, **une copie** de la valeur de l'argument **est transmise** à la fonction.

Cela signifie que toute **modification** de cette valeur à **l'intérieur** de la fonction n'affectera pas la valeur **originale** dans le programme appelant.

EXEMPLE

```
#include <stdio.h>
void incrementer(int x) {
    x = x + 1;
    printf("Valeur de x dans la fonction: %d\n", x);
}
int main() {
    int nombre = 5;
    printf("Avant l'appel : %d\n", nombre);
    incrementer(nombre); // Passage par valeur
    printf("Après l'appel : %d\n", nombre); // La valeur reste inchangée
    return 0;
}
```


Explication

1. La variable **nombre** contient 5 dans main().
2. Lorsqu'on appelle `incrémenter(nombre)`, une copie de **nombre** est créée dans x.
3. La modification de x ne modifie pas **nombre**.
4. Après l'exécution de la fonction, **nombre** reste inchangé dans main().

👉 Le passage par valeur **ne permet pas** à la fonction **de modifier la variable originale**.

Passage par Adresse

Dans le passage par adresse, l'adresse de la variable est transmise à la fonction à l'aide d'un pointeur.

Cela permet à la fonction de modifier directement la variable originale.



Mots-clés de Contrôle

Mots-clés de Contrôle

break : Sort immédiatement d'une boucle ou d'un switch.

continue : Passe à l'itération suivante d'une boucle.

return : Termine l'exécution d'une fonction et retourne une valeur.

Exemple:

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) { break; // Sort de la boucle quand i == 5 }  
    if (i % 2 == 0) { continue; // Passe à l'itération suivante si i est pair }  
    printf("i = %d\n", i);  
}
```