

Cours: Algorithmes et Structures des Données

Chapitre 4: Les sous-programmes

Réalisé par:

Dr. Sakka Rouis Taoufik

1

Chapitre 4 : Les sous-programmes

I. Introduction

En C, les sous-programmes s'appellent des fonctions.

Les fonctions sont des parties de code source qui permettent de réaliser le même type de traitement plusieurs fois et/ou sur des variables différentes.

Une fonction a toujours un type de retour, qui correspond au type du résultat qu'elle peut renvoyer et qui peut être n'importe lequel des types que nous avons précédemment étudié.

Le type de retour peut être **void** si on souhaite que la fonction ne renvoie rien.

Une fonction a aussi un nom et une liste de zéro, un, ou plusieurs, paramètres.

2

Chapitre 4 : Les sous-programmes

II. Déclaration

Syntaxe :

```
TypeDeRetour NomFN (Type1 Parametre1, Type2 Parametre2)
{
    /*variables locales*/

    instructions de la fonction

    return ValeurDeRetour ; /*optionnelle si la fonction ne
                             retourne rien (de type void)*/
}
```

3

Chapitre 4 : Les sous-programmes

III. Les prototypes

Le prototype d'une fonction (voir chapitre 1) est une description d'une fonction qui est définie plus loin dans le programme. On place donc le prototype en début du programme (avant la fonction principale main()).

Le prototype d'une fonction reprend exactement l'en-tête de la fonction, mais pas son corps, qui est remplacé par un point-virgule.

Cette description permet au compilateur de « vérifier » la validité de la fonction à chaque fois qu'il la rencontre dans le programme.

4

Chapitre 4 : Les sous-programmes

III. Les prototypes

Syntaxe :

TypeDeRetour NomFN (Type1 Parametre1, Type2 Parametre2) ;

Exemples :

```
#include <stdio.h>
int somme (int a, int b); /*déclaration du prototype de la fonction somme */

void main () {
    int x, y, z;
    ...
    z= somme (x, y) ;
    ...
}
/* implémentation du corps de la fonction somme */
int somme (int a, int b) {
    ...
}
```

5

Chapitre 4 : Les sous-programmes

IV. Passage des paramètres

Tout paramètre d'une fonction peut être de type ordinaire (int, float,...) ou de type pointeur.

Dans le cas d'un paramètre ordinaire : le passage de ce dernier est appelé **passage par valeur**. Dans ce cas la fonction fait une copie de la valeur du paramètre. Elle n'utilise que la copie, qui peut être modifiée. Mais à la fin du traitement de la fonction, la valeur de la variable passée en paramètre n'a pas été modifiée, puisque c'est seulement la copie qui a été modifiée.

Exemple :

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

6

Chapitre 4 : Les sous-programmes

IV. Passage des paramètres

Dans le cas d'un paramètre de type pointeur : le passage de ce dernier est appelé **passage par adresse**. Dans ce cas la fonction ne fait pas de copie de la valeur du paramètre, comme c'est fait dans le cas du passage par valeur. Par conséquent, si le paramètre change de valeur, alors à la sortie de la fonction, la variable passée en paramètre contient la dernière modification, donc la dernière valeur.

Exemple :

```
void add (int a, int b, int *c) {
    *c = a + b;
}
```

7

Chapitre 4 : Les sous-programmes

V. Appels des fonctions

Une fonction est appelée comme étant une instruction du programme si elle ne retourne rien, sinon elle est appelée dans une expression d'une affectation, comparaison ou autre.

Exemple:

```
#include <stdio.h>
float carre (float X) {
    X= X*X;
    return (X) ;
}
void main(){
    float A = 2, B;
    B= carre (A) ;
    /*la valeur de A ne change pas après l'appel de la fonction carre */
    printf("le carre de %f est=%f", A , B);
}
```

8

Chapitre 4 : Les sous-programmes

V. Appels des fonctions

Exemple:

```
#include <stdio.h>
void permute (float * x, float * y) {
    float aux;
    aux= * x ;
    * x= * y ;
    * y =aux ;
}
void main(){
    float A = 2.3, B=3.2;
    permute (&A, &B) ;
    /*la valeur de A sera 3.2 et la valeur de B sera 2.3 */
    printf("A=%f et B=%f", A , B);
}
```

9

Chapitre 4 : Les sous-programmes

VI. Exercices d'application

Exercice 1 :

Écrire en C un algorithme d'une fonction Triangle qui permet de vérifier si les 3 nombres a, b et c peuvent être les mesures des côtés d'un triangle rectangle.

Remarque: D'après le théorème de Pythagore, si a, b et c sont les mesures des côtés d'un rectangle, alors $a^2 = b^2 + c^2$ ou $b^2 = a^2 + c^2$ ou $c^2 = a^2 + b^2$

Exercice 2 :

Écrire une fonction qui étant donné un entier n, renvoie la somme suivante:

$$\sum_{i=1}^n \sum_{j=1}^i (i + j)$$

10

Chapitre 4 : Les sous-programmes**VI. Exercices d'application****Exercice 3 :**

Ecrire un programme en C qui lit deux nombre naturel non nul m et n et qui détermine s'ils sont amis. Deux nombres entiers n et m sont qualifiés d'amis, si la somme des diviseurs de n est égale à m et la somme des diviseurs de m est égale à n (on ne compte pas comme diviseur le nombre lui-même et 1). Proposer une solution modulaire.

Exercice 4 :

Réaliser en C un algorithme d'une fonction qui recherche le premier nombre entier naturel dont le carré se termine par n fois le même chiffre.

Exemple : pour $n = 2$, le résultat est 10 car 100 se termine par 2 fois le même chiffre.

11