

Chapitre8: **LES fonctions**

Syrine KHIARI - Wafa NEJI – Salma HAJJEM

ESPRIT



2012-2013

Ecole Supérieure Privée
d'Ingénierie et de Technologies



Introduction

Le langage C permet de découper un programme en plusieurs parties appelées **modules**.

Un module est un ensemble de données et d'instructions qui fournissent une solution à une petite partie d'un problème plus complexe.

Quelques avantages de la modularité :

- ✓ Meilleure lisibilité
- ✓ Diminution du risque d'erreurs
- ✓ Possibilité de tests sélectifs
- ✓ Réutilisation de modules déjà existants
- ✓ Simplicité d'entretien

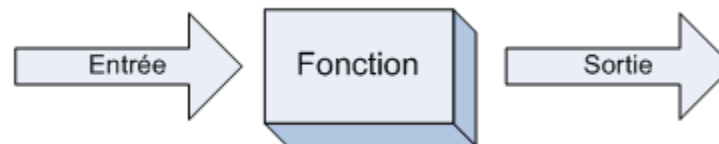
Les fonctions?!!

- *Définition :*

- En langage C, la structuration des programmes en modules se fait à l'aide des **fonctions** et **main()** est une de ces fonctions.

- Une fonction exécute des actions et renvoie un résultat. C'est un **morceau de code** qui sert à faire quelque chose de précis.

On dit qu'une fonction possède une entrée et une sortie.



Les fonctions?!!

- *Définition :*

- Une fonction possède un et un seul point de sortie, mais éventuellement plusieurs points d'entrée.

Type résultat Nom fonction Type Paramètre1 Nom Paramètre1

```
int max (int A, int B)
{
    /* déclarations locales */
    int maximum ;
    /* suite d'instructions */
    if (A > B) maximum = A;
    else maximum = B;
    /* renvoi du résultat */
    return (maximum);
}
```

Les fonctions?!!

- *Remarques:*

- Si la fonction n'a pas de paramètres, déclarez la liste des paramètres comme void ou ne rien mettre entre les ()

- **Exemple:**

`max () {...}` ou `int max (void) {... }`

- Quand une fonction ne fournit pas de résultat, indiquez comme void le type du résultat

- **Exemple:**

```
void Bonjour ( )  
    { printf ("Bonjour\n"); }
```

Appeler une fonction

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


int triple(int nombre) // 6
{
    return 3 * nombre; // 7
}

int main(int argc, char *argv[]) // 1
{
    int nombreEntre = 0, nombreTriple = 0; // 2

    printf("Entrez un nombre... "); // 3
    scanf("%d", &nombreEntre); // 4

    nombreTriple = triple(nombreEntre); // 5
    printf("Le triple de ce nombre est %d\n", nombreTriple); // 8

    return 0; // 9
}
```



- Il faut écrire la fonction triple **AVANT** la fonction main. Si vous la placez après, ça ne marchera pas.

Appeler une fonction

2) La fonction triple retourne (return) une valeur.

Cette valeur, c'est 3x le nombre qu'on lui a envoyé.

Cette valeur de retour est stockée dans la variable nombreTriple de la fonction main.

Le signe « = » permet donc de dire « Envoie le résultat de la fonction dans cette variable ».

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

int triple(int nombre)
{
    return 3 * nombre;
}

int main(int argc, char *argv[])
{
    int nombreEntre = 0, nombreTriple = 0;

    printf("Entrez un nombre... ");
    scanf("%d", &nombreEntre);

    nombreTriple = triple(nombreEntre);
    printf("Le triple de ce nombre est %d\n", nombreTriple);

    return 0;
}
```

1) La variable nombreEntre est envoyée en paramètre à la fonction triple. Celle-ci récupère cette variable dans une autre variable qui s'appelle « nombre ».

Note : on aurait aussi pu mettre le même nom de variable dans les 2 fonctions. Il n'y aurait pas eu de conflit, car une variable appartient à sa fonction.

Déclaration d'une fonction (**prototype**)

- Si vous mettez votre fonction après le main, ça ne marchera pas car l'ordinateur ne connaîtra pas encore la fonction.
- Pour résoudre ce problème nous allons « annoncer » nos fonctions à l'ordinateur en écrivant ce qu'on appelle des **prototypes**.
- Toute fonction doit être déclarée avant d'être utilisée (sauf la fonction main, on peut ne pas la déclarer)

Déclaration d'une fonction (prototype)

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

// La ligne suivante est le prototype de la fonction aireRectangle :
double aireRectangle(double largeur, double hauteur);

int main(int argc, char *argv[])
{
    printf("Rectangle de largeur 5 et hauteur 10. Aire = %f\n", aireRectangle(5, 10));
    printf("Rectangle de largeur 2.5 et hauteur 3.5. Aire = %f\n", aireRectangle(2.5, 3.5));
    printf("Rectangle de largeur 4.2 et hauteur 9.7. Aire = %f\n", aireRectangle(4.2, 9.7));

    return 0;
}

// Notre fonction aireRectangle peut maintenant être mise n'importe où dans le code source :
double aireRectangle(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

Nous pouvons écrire tout simplement

```
double aireRectangle(double, double);
```

Les headers

- Depuis que vous devez déclarer toutes les fonctions de C avant de pouvoir les utiliser, il serait utile de déclarations du groupe de fonctions connexes et de les gérer en un seul endroit. C'est ce que les **fichiers Header** peut faire.
- Les fichiers Header sont indispensables dans les grands projets car ils vous donnent un aperçu du code source sans avoir à parcourir chaque ligne de code.

Création d'un fichier header

```
math_functions.h
```

```
int  sum      (int x, int y);  
float average (float x, float y, float z);
```

```
math_functions.c
```

```
int sum (int x, int y)  
{  
    return (x + y);  
}  
  
float average (float x, float y, float z)  
{  
    return (x + y + z) / 3;  
}
```

Utilisation d'un fichier header

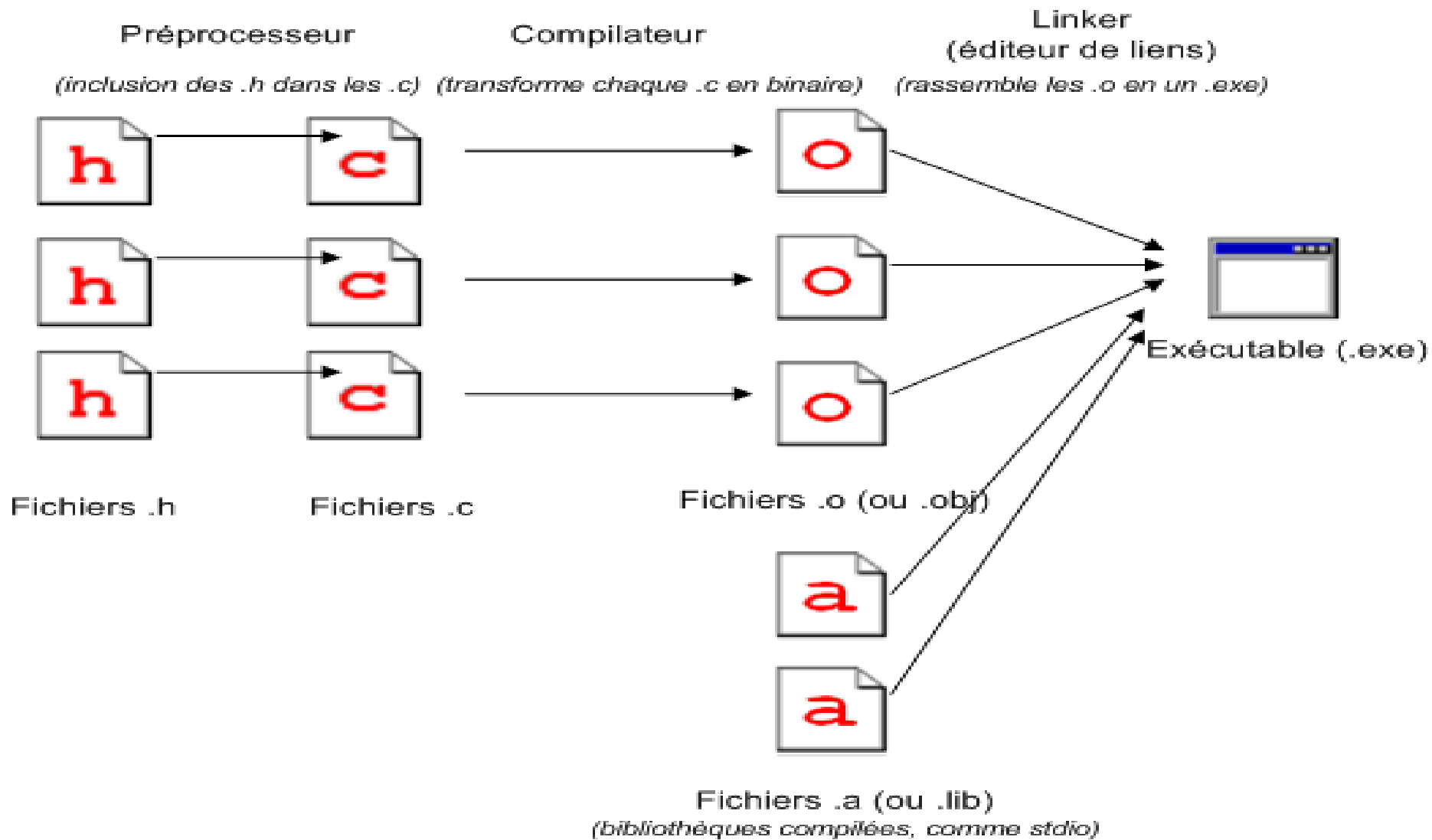
test3.c

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

main ()
{
    int    theSum      = sum (8, 12);
    float theAverage = average (16.9, 7.86, 3.4);

    printf ("the sum is: %i ", theSum);
    printf ("and the average is: %f \n", theAverage);
    printf ("average casted to an int is: %i \n", (int)theAverage);
}
```

La compilation séparée



Les variables locales

- Les variables déclarées dans une fonction sont uniquement **visibles à l'intérieur** de cette fonction. On dit que ce sont des **variables locales** pour cette fonction

```
int triple(int nombre)
{
    int resultat = 0; // La variable resultat est créée en mémoire

    resultat = 3 * nombre;
    return resultat;
} // La fonction est terminée, la variable resultat est supprimée de la mémoire
```

Les variables globales : à éviter

- Les variables déclarées au début du fichier, à l'extérieur de toutes les fonctions **sont accessibles dans tous les fichiers**. Ce sont des **variables globales**

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

int resultat = 0; // Déclaration de variable globale

void triple(int nombre); // Prototype de fonction

int main(int argc, char *argv[])
{
    triple(15); // On appelle la fonction triple, qui modifie la variable globale resultat
    printf("Le triple de 15 est %d\n", resultat); // On a accès à resultat

    return 0;
}

void triple(int nombre)
{
    resultat = 3 * nombre;
}
```

Variable globale accessible uniquement dans un fichier

- Pour créer une variable globale accessible uniquement dans un fichier, rajoutez simplement le mot-clé **static** devant :

```
static int resultat = 0;
```


Variable locale statique

- Si vous rajoutez le mot-clé **static** devant la déclaration d'une variable à l'intérieur d'une fonction, ça n'a pas le même sens que pour les variables globales.
En fait, la variable **static** n'est plus supprimée à la fin de la fonction. La prochaine fois qu'on appellera la fonction, la variable aura conservé sa valeur.

```
int triple(int nombre)
{
    static int resultat = 0; // La variable resultat est créée la première fois que la fonction est appelée

    resultat = 3 * nombre;
    return resultat;
} // La variable resultat n'est PAS supprimée lorsque la fonction est terminée.
```

Les fonctions locales à un fichier

- Normalement, quand vous créez une fonction, celle-ci est globale à tout le programme. Elle est **accessible depuis n'importe quel autre fichier** .c.

Il se peut que vous ayez besoin de créer des fonctions qui **ne seront accessibles que dans le fichier** dans lequel se trouve la fonction.

Pour faire cela, rajoutez le mot-clé **static** devant la fonction

Les fonctions locales à un fichier

```
static int triple(int nombre)
{
    // Instructions
}
```

```
static int triple(int nombre);
```

Paramètres d'une fonction

- Les paramètres se trouvant dans l'en-tête d'une fonction sont appelés des **paramètres formels**. Ils permettent au sein du corps de la fonction de décrire ce qu'elle doit faire.

```
int max (int x, int y);  
{  
    if (x > y)  
        return (x);  
    else  
        return (y);  
}
```

Paramètres d'une fonction

- Les paramètres fournis à l'appel de la fonction se nomment **des paramètres effectifs** . Ces paramètres vont remplacer les **paramètres formels** lors de **l'exécution du programme**.

```
int main ( )  
{   int C, A, B ;  
    A = 3;  
    B = 7;  
    C = max (A, B);  
    ...  
}
```