

COURS 3

FONCTIONS

EN

C

EXEMPLE ILLUSTRATIF

VOTRE (TRÈS
BEAU) PROF



ECRIVEZ-MOI UN PROGRAMME QUI RÉCUPÈRE
DEUX ENTIERS POSITIFS ET QUI AFFICHE UN MESSAGE
S'ILS ONT LE MÊME NOMBRE DE CHIFFRES -

PASSIONNANT

VOUS



(VOUS AUSEZ
VOUS ÊTES
BELLE/BEAU)

PAR EX., COMME 218 ET 937

MAIS PAS COMME 9999 ET 10001

```
Terminal
Fichier  Édition  Affichage  Terminal  Onglets  Aide
courtiel@N302L-G17P06:/tmp$ ./meme_nombre_de_chiffres
Saisissez un premier nombre : 287
Saisissez un second nombre : 953
Les deux nombres ont le même nombre de chiffres.
courtiel@N302L-G17P06:/tmp$ ./meme_nombre_de_chiffres
Saisissez un premier nombre : 99999
Saisissez un second nombre : 100000
Les deux nombres N'ont PAS le même nombre de chiffres.
courtiel@N302L-G17P06:/tmp$
```

EXEMPLE ILLUSTRATIF

ÉCRIREZ UN PROGRAMME QUI RÉCUPÈRE DEUX ENTIERS ET QUI AFFICHE UN MESSAGE S'ILS ONT LE MÊME NOMBRE DE CHIFFRES -

TENTATIVE

le code fonctionne
mais ...

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

int main() {

    int x, y;
    printf("Saisissez un premier nombre : ");
    scanf("%d",&x);
    printf("Saisissez un second nombre : ");
    scanf("%d",&y);

    int nombre_chiffres_x = 1;
    while ( x > 9 ) {
        x = x / 10;
        nombre_chiffres_x++;
    }

    int nombre_chiffres_y = 1;
    while ( y > 9 ) {
        y = y / 10;
        nombre_chiffres_y++;
    }

    if ( nombre_chiffres_x == nombre_chiffres_y ) {
        printf("Les deux nombres ont le même nombre de chiffres.\n");
    }
    else{
        printf("Les deux nombres N'ont PAS le même nombre de chiffres.\n");
    }

    return EXIT_SUCCESS;
}
```

CODE
REDONDANT!

MOTIVATION DERRIÈRE LES FONCTIONS



```
int nombre_chiffres_x = 1;
while ( x > 9 ) {
    x = x / 10;
    nombre_chiffres_x++;
}

int nombre_chiffres_y = 1;
while ( y > 9 ) {
    y = y / 10;
    nombre_chiffres_y++;
}
```

On veut factoriser le code (= regrouper les trauons de code similaires en un seul endroit)

Intérêts

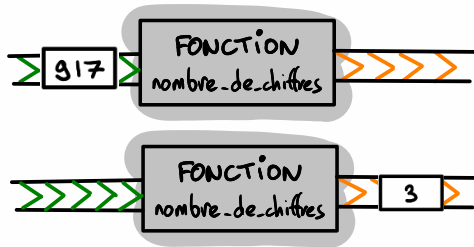
- 1 - Le code est plus compact, plus lisible
- 2 - Si on a fait une erreur dans un bout de code qu'on a répété, on est obligé de corriger ces erreurs pour chaque répétition.

Outre la factorisation, on verra que les fonctions permettent de structurer le code (= décomposer le programme en sous-programmes)
(et puis, on n'utilisera que ça sur celine!)

PRINCIPE

Une fonction est comme une "boîte noire" vers laquelle des tapis roulants acheminent des ingrédients (= paramètres ou arguments) et d'où sort l'objet d'un calcul (= sortie)

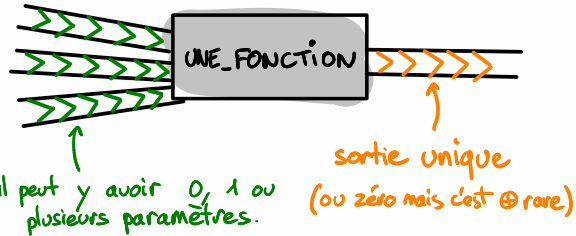
MÉTAPHORE



Notre exemple

SYNTAXE

```
int nombre_de_chiffres (int x) {  
    int res = 1;  
    while ( x < 9 ) {  
        x = x / 10;  
        res ++;  
    }  
    return res;  
}
```



En général

```
type_sortie ma_fonction (type1 param1,  
                          type2 param2, ... ) {
```

indentation

```
    INSTRUCTIONS QUI SERONT EXÉCUTÉES  
    return SORTIE;
```

Ici type vaut int, float, char, ...
 Si la fonction n'a pas de sortie,
 alors type_sortie vaudra void

COMMENT UTILISER UNE FONCTION ?

Une fonction se place juste avant le "main" (sauf rares exceptions)

au passage, main est aussi une fonction!

Une fonction s'appelle.

Pour cela, on écrit le nom de la fonction puis entre parenthèses des paramètres, séparés par des virgules.

Les paramètres peuvent être :

- ① des valeurs
- ② des expressions
- ③ des contenus de variable

Pour récupérer le résultat de l'appel de la fonction (= la sortie), on doit la stocker dans une variable.

CE QUI EST AFFICHÉ

Il y a 3 chiffres dans 342.
exemple vaut 8.

À vous de rentrer un nombre : 16
Il y a 2 chiffres dans 16.

EXEMPLE

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

int nombre-de-chiffres (int x) {
    int res = 1;
    while (x < 9) {
        x = x / 10;
        res++;
    }
    return res;
}

int main() {
    printf("Il y a %d chiffres dans\n342.\n", nombre-de-chiffres(342)); ①
    int exemple;
    exemple = nombre-de-chiffres(10004-10); ②
    exemple = 2 * exemple;
    printf("exemple vaut %d.\n", exemple);
    printf("À vous de rentrer un nombre :\n");
    int a;
    scanf("%d", &a);
    printf("Il y a %d chiffres dans\n%d.\n", nombre-de-chiffres(a), a);
    return EXIT_SUCCESS; ③
}
```

RÉPONSE AU PROBLÈME ORIGINEL

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

int nombre_de_chiffres( int x ) {
    /* Cette fonction prend un entier en entrée
    et renvoie son nombre de chiffres en base décimale. */
    int res = 1;
    while ( x > 9 ) {
        x = x / 10;
        res++;
    }
    return res;
}

int main() {

    int x, y;
    printf("Saisissez un premier nombre : ");
    scanf("%d",&x);
    printf("Saisissez un second nombre : ");
    scanf("%d",&y);

    if ( nombre_de_chiffres(x) == nombre_de_chiffres(y) ) {
        printf("Les deux nombres ont le même nombre de chiffres.\n");
    }
    else{
        printf("Les deux nombres N'ont PAS le même nombre de chiffres.\n");
    }

    return EXIT_SUCCESS;
}
```

notre
fonction
"nombre-de-chiffres"

fonction
principale

2 appels à
notre fonction
"nombre-de-chiffres"

REMARQUE
IMPORTANTE

Les variables portant un même nom dans deux fonctions différentes sont des variables différentes. Les variables sont locales -

Par ex, le `xc` dans `nombre-de-chiffres` n'a rien à voir avec le `xc` du `main`.

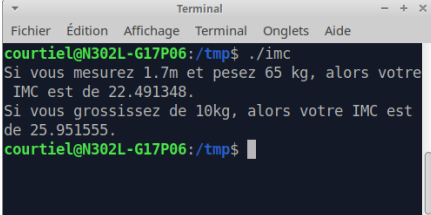
D'AUTRES EXEMPLES DE FONCTIONS

Exemple de fonction avec plusieurs paramètres : l'IMC

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

float imc(int poids, float taille){
    /* Entrée : le poids (en kg), la taille (en m)
       Sortie : l'indice de masse corporelle */
    return poids/(taille*taille);
}

int main() {
    int poids = 65;
    printf("Si vous mesurez 1.7m et pesez 65 kg, ");
    printf("alors votre IMC est de %.1f\n", imc(poids,1.7) );
    printf("Si vous grossissez de 10kg, ");
    printf("alors votre IMC est de %.1f\n", imc(poids+10,1.7) );
    return EXIT_SUCCESS;
}
```



A terminal window titled "Terminal" with a menu bar (Fichier, Édition, Affichage, Terminal, Onglets, Aide). The prompt is "courtiel@N302L-617P06:/tmp\$./imc". The output shows the calculation of BMI for 65 kg and 1.7 m, resulting in 22.491348, and then for 75 kg, resulting in 25.951555.

```
courtiel@N302L-617P06:/tmp$ ./imc
Si vous mesurez 1.7m et pesez 65 kg, alors votre
IMC est de 22.491348.
Si vous grossissez de 10kg, alors votre IMC est
de 25.951555.
courtiel@N302L-617P06:/tmp$
```

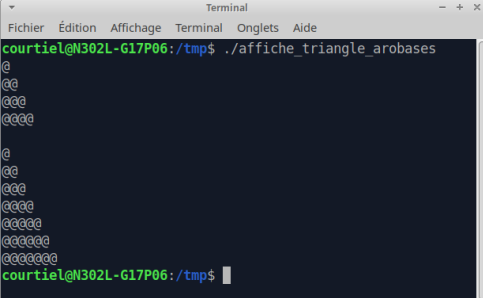
Exemple de fonctions sans sortie : affichage de triangles

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

void affiche_ligne_arobases(int nb_arobases){
    for ( int i = 1 ; i <= nb_arobases ; i++){
        printf("@");
    }
    printf("\n");
}

void affiche_triangle(int nb_lignes){
    for ( int i = 1 ; i <= nb_lignes ; i++){
        affiche_ligne_arobases(i);
    }
}

int main() {
    affiche_triangle(4);
    printf("\n");
    affiche_triangle(7);
    return EXIT_SUCCESS;
}
```



A terminal window titled "Terminal" with a menu bar (Fichier, Édition, Affichage, Terminal, Onglets, Aide). The prompt is "courtiel@N302L-617P06:/tmp\$./affiche_triangle_arobases". The output shows two triangles of '@' characters: one with 4 lines and one with 7 lines.

```
courtiel@N302L-617P06:/tmp$ ./affiche_triangle_arobases
@
@@
@@@
@@@@

@
@@
@@@
@@@@
@@@@@
@@@@@@
@@@@@@@
courtiel@N302L-617P06:/tmp$
```


DANGER DE MORT !!

printf et scanf servent uniquement à interagir avec l'utilisateur
→ printf affiche des messages pour lui (c'est cosmétique)
→ scanf récupère ce qu'il saisit (entité extérieure au programme)

scanf est à bannir de vos fonctions*

*: dans 95% des cas... Seul cas où c'est autorisé :

Quand la consigne vous dit de récupérer une valeur de l'utilisateur

Ex

```
int recupere_entier () {  
    printf("Veuillez saisir un entier : \n");  
    int m;  
    scanf("%d", &m);  
    return m;  
}
```

← au passage
cette fonction n'a
pas de paramètre

En outre, il ne faut pas confondre printf et return.
On ne peut pas récupérer une variable avec un printf
On utilise printf quand la consigne dit d'afficher quelque chose.

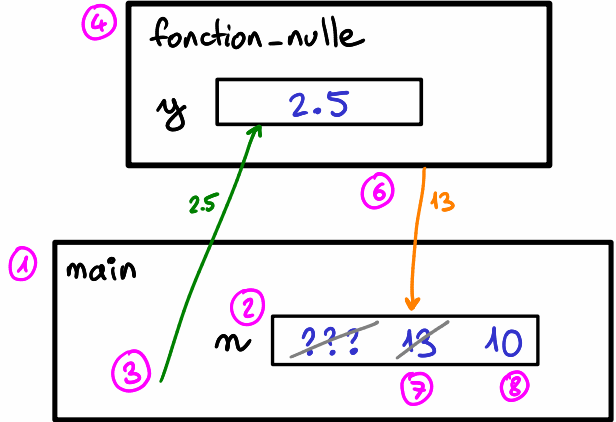
MECANISMES DERRIERE UN APPEL DE FONCTION

EXEMPLE
1

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

④ int fonction_nulle(float y){
  ⑤ if (y > 7) {
    return 4;
  }
  else {
    return 13; ⑥
  }
}

① int main() {
  ② int n;
  n = fonction_nulle(2.5); ③
  n = n - 3;
  ⑤ printf("n vaut %d.\n",n);
  return EXIT_SUCCESS;
}
```



Différentes étapes (dans l'ordre chronologique):

- ① La fonction main est appelée
- ② La variable n est déclarée (valeur inconnue)
- ③ La fonction fonction_nulle est appelée avec pour paramètre 2.5
On abandonne pour l'instant main
- ④ fonction_nulle va être maintenant exécutée avec une variable y initialisée à 2.5
- ⑤ On teste si la valeur de y > 7
- ⑥ C'est faux, donc on renvoie 13.
L'exécution de fonction_nulle s'arrête.
- ⑦ main reprend la main et on stocke la sortie, 13, dans la variable n.
- ⑧ n passe de 13 à 10.
- ⑨ "n vaut 10." est affiché.

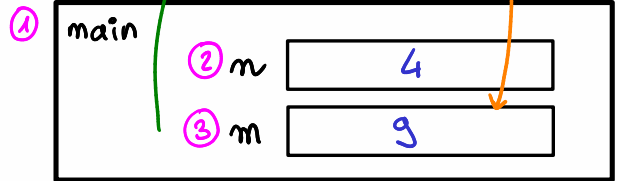
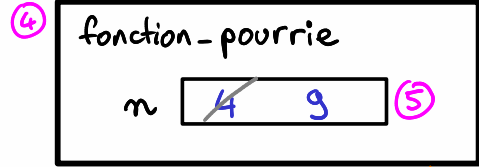
MECANISMES DERRIÈRE UN APPEL DE FONCTION

EXEMPLE 2

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

```
④ int fonction_pourrie(int n) {
    ⑤ n = 2*n + 1;
    ⑥ return n;
}

① int main() {
    ② int n = 4;
    ③ int m = fonction_pourrie(n);
    ⑦ printf("n vaut %d et m vaut %d.\n", n, m);
    return EXIT_SUCCESS;
}
```



Différentes étapes (dans l'ordre chronologique):

- ① La fonction main est appelée
- ② La variable n est initialisée à 4.
- ③ La variable m est déclarée et fonction_pourrie est appelée avec pour paramètre la valeur de n : 4
- ④ fonction_nulle va être maintenant exécutée avec une variable locale n initialisée à 4
- ⑤ Sa valeur passe de 4 à 9.
- ⑥ On renvoie la valeur de n : 9 - On revient dans la fonction main et on stocke 9 dans m.
- ⑦ "n vaut 4 et m vaut 9" est affiché

Bien que la variable n ait changé à l'intérieur de fonction_pourrie, celle dans main n'a pas été modifiée!

MECANISMES DERRIERE UN APPEL DE FONCTION

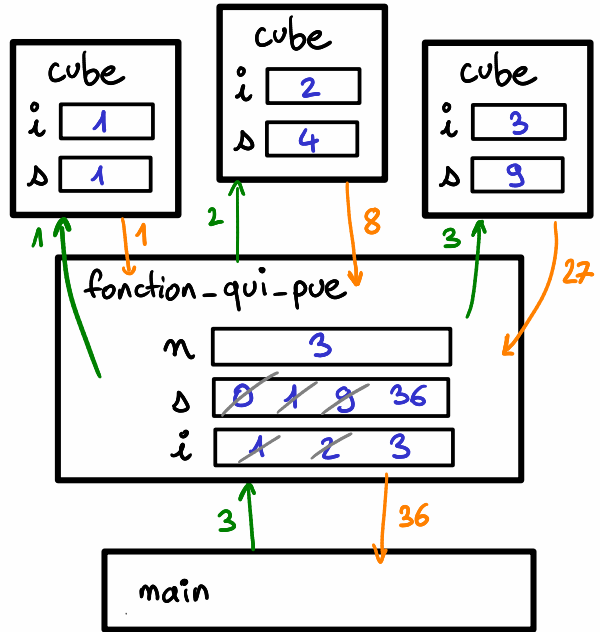
EXEMPLE 3

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

int cube(int i) {
    int s = i*i;
    return s*i;
}

int fonction_qui_pue(int n) {
    int s = 0;
    for (int i = 1; i <= n ; i++){
        s = s + cube(i);
    }
    return s;
}

int main() {
    printf("%d",fonction_qui_pue(3));
    return EXIT_SUCCESS;
}
```



Ici "36" est affiché

Une fonction peut appeler une fonction qui peut appeler une fonction...
On peut empiler les appels : on parle même de pile d'exécution

RÉCURSIVITÉ

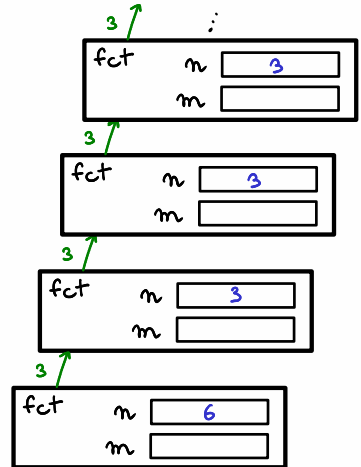
Une fonction peut-elle s'appeler elle-même? **Oui!**
→ c'est ce qu'on appelle une fonction récursive

⚠ Bien sûr, si on fait n'importe quoi, ça boucle à l'infini

(Ex)

```
int fct ( int m ) {  
    int m;  
    m = fct(3);  
    return m + m;  
}
```

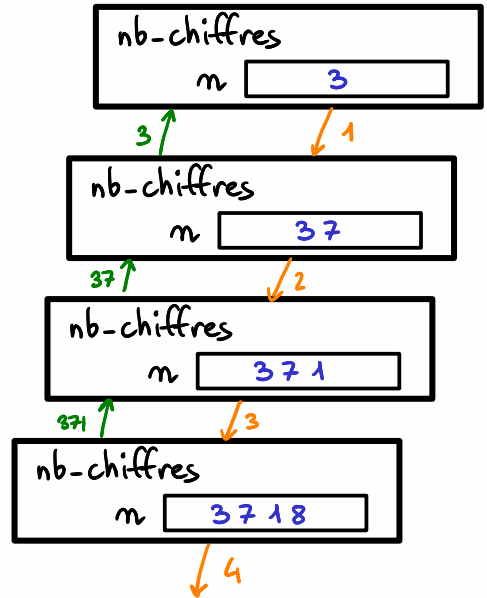
En vrai, il va y avoir trop d'appels de fonction:
Le programme va planter à cause d'un
débordement de la pile d'exécution
(stack overflow)



RÉCURSIVITÉ

Un exemple qui marche :

```
int nb_chiffres (int n) {  
    if (n <= 9) {  
        return 1;  
    }  
    else {  
        return 1 + nb_chiffres(n/10);  
    }  
}
```



Il faut un "cas de base" (ici $n \leq 9$) pour qu'une fonction récursive s'arrête.

PLUS DE DÉTAILS EN L2 !