Chapitre 10 : Programmation – les fonctions spé ISN – TS

En programmation, il arrive fréquemment qu'on ait besoin d'utiliser plusieurs fois le même ensemble d'instructions. Plutôt que de réécrire cet ensemble d'instructions à chaque utilisation, on le reporte dans une fonction, écrite en dehors de la fonction principale *main*. Seul l'appel de la fonction sera alors fait plusieurs fois dans le *main*.

I. Déclaration d'une fonction : le prototype

L'écriture d'un code incluant des fonctions se fait de la manière suivante :

```
int ma_fonction(int parametre1, float parametre2) ;  // prototype de la fonction

int main() {                                          // code de la fonction principale
  ..............
}

int ma_fonction(int parametre1, float parametre2) { // code de la fonction
  ..............
}
```

L'ensemble des instructions de la fonction étant placé après le main, il est nécessaire d'indiquer, avant le main, la structure de la fonction. Ce **prototype** permet au compilateur de vérifier, à la lecture du main, que l'appel de la fonction respecte bien le nombre de paramètres et le type de chacun.

Exemple : prototype d'une fonction multiplication :

```
int  multiplication (int valeur1, int valeur2) ;
```

Ce prototype indique que la fonction multiplication prend deux entiers en entrée, et retourne un autre entier en fin d'appel.

Question 1 : on considère le prototype suivant :

```
float  fonction (int num, float dem, int truc, int machin) ;
```

1. Combien de paramètres la fonction suivante utilise-t-elle en entrée ?
2. Quel est le type de variable retournée par la fonction ?

Il est possible qu'une fonction ne renvoie rien. Dans ce cas, le type de valeur de retour est void.

```
void fonction (int valeur1, int valeur2) ;
```

Question 2 : quel type de variable la fonction main renvoie-t-elle ?

II. Contenu d'une fonction

Les instructions contenues dans une fonction sont identiques à celles qu'on aurait employées directement dans le main. Par exemple :

Sans fonction :

```
#include <stdio.h>
int  main() {
    int i = 2, j = 3 ;
    int produit = i * j ;
    printf("%d", produit) ;
    return 0 ;
}
```

Avec fonction :

```
#include <stdio.h>
int produit (int nombre1, int nombre2) ;
int  main() {
    int i = 2, j = 3, resultat ;
    resultat = produit(i, j) ;
    printf("%d", resultat) ;
    return 0 ;
}
int produit (int nombre1, int nombre2)
{
    return nombre1 * nombre2 ;
}
```

Dans cet exemple, le gain n'est pas évident, car on n'appelle la fonction qu'une seule fois, et celle-ci est très simple. Mais pour de longs blocs d'instructions qu'il faut appeler très souvent, l'usage d'une fonction rend rapidement le code beaucoup plus lisible et facile à débogger.

Remarque : l'appel d'une fonction dont la variable de sortie est de type void, dans le main s'écrit simplement : `fonction (valeur1, valeur2) ;`

En réalité, le compilateur remplace, dans le main, la ligne d'appel par le bloc d'instructions correspondant. Ce sont donc bien des instructions identiques.

Question 3 : que fait le code suivant ?

```
#include <stdio.h>
int fonction(int seuil) ;
int  main() {
    int seuil, resultat ;
    printf("Définir un seuil") ;
    scanf("%d", &seuil) ;
    resultat = fonction(seuil) ;
    printf("%d", resultat) ;
    return 0 ;
}
```

```
int fonction (int seuil) {
    int n = 0, puissance = 1 ;
    while (puissance < seuil) {
        puissance = puissance * 2 ;
        n = n + 1 ;
    }
    return n ;
}
```

III. Fonctions qui modifient des paramètres

Avec l'écriture précédente, les fonctions utilisent une copie des paramètres passés en entrée. Ainsi, elles ne modifient pas leur valeur de manière pérenne : une fois l'appel de la fonction terminé, les paramètres reprennent la valeur qu'ils avaient avant l'appel.

Exemple :

```
#include <stdio.h>
int fonction(int parametre) ;
int  main() {
    int nombre = 3 ;                          // Ligne 1
    int resultat = fonction(nombre) ;         // Ligne 2
    printf("%d et %d", nombre, resultat) ;    // Ligne 3
    return 0 ;
}
int fonction(int parametre) {
    parametre = 2 * parametre ;               // Ligne a
    return parametre ;                        // Ligne b
}
```

Ici :

- En ligne 1 : nombre vaut 3.
- En ligne 2 : la fonction prend une copie de nombre pour paramètre, soit 3.
- On démarre l'appel de la fonction :
  - en ligne a, parametre prend comme valeur le double de la valeur d'entrée, soit 6.
  - la fonction retourne la valeur de parametre, soit 6.
- L'appel de la fonction est terminé.
  - nombre garde sa valeur d'avant l'appel de la fonction, soit 3.
  - la fonction retourne la valeur 6, qui est stockée dans la variable resultat.
- En ligne 3, on affiche les valeurs de nombre et resultat, soit 3 et 6.

On dit ici que les paramètres sont passés **par valeur**.
Pour les modifier de manière pérenne, on ne doit pas accéder à la valeur du paramètre, mais modifier directement sa valeur à l'endroit où il est stocké en mémoire. On fera donc un passage de paramètre **par adresse**.

Exemple :

```
int  fonction (int * parametre_par_adresse, int parametre_par_valeur) ;
```

La notation int *, qui se lit « pointeur sur entier », permet de manipuler l'adresse de la variable. *On ne développera pas ici la notion de pointeur.*

Dans l'exemple suivant, en ligne 2, on utilise l'adresse de la variable nombre : &nombre.
Ici, en ligne a, la valeur du paramètre d'entrée nombre est durablement modifiée.
Ainsi, en ligne 3, on obtiendra 6 et 6.

```
#include <stdio.h>
int fonction(int * parametre) ;
int  main() {
    int nombre = 3 ;                          // Ligne 1
    int resultat = fonction(&nombre) ;        // Ligne 2
    printf("%d et %d", nombre, resultat) ;    // Ligne 3
    return 0 ;
}
int fonction(int * parametre) {
    parametre = 2 * parametre ;               // Ligne a
    return parametre ;                        // Ligne b
}
```

Question 4 : on considère la déclaration de fonction suivante :

```
int  fonction (int parametre1, int * parametre2, int * parametre3) ;
```

1. Quels sont les paramètres passés par valeur ?
2. Quels sont les paramètres qui seront durablement modifiés par l'appel de la fonction ?

IV. Manipulation de tableaux dans une fonction

La manipulation de tableaux en paramètres de fonctions nécessite une rédaction particulière :

```
int  fonction (int * tableau, int dimension) ;
```

Ici, la fonction prend deux paramètres :

- un tableau (pour être précis, c'est un pointeur sur le début du tableau) ;
- la dimension du tableau, passée par valeur (*on ne doit pas changer la taille d'un tableau après sa déclaration*).

L'appel de la fonction se rédige ainsi : `variable = fonction(tableau, dimension) ;`

Exemple : fonction permettant d'afficher l'ensemble des valeurs du tableau :

```
void fonction(int * tableau, int dimension)) {
    int i ;
    for (i = 0 ; i < dimension ; i = i + 1) {
        printf("%d ", tableau[i]) ;
    }
}
```

Question 5 : que fait le code suivant ?

```
void fonction(int * tableau, int dimension)) {
    int i ;
    for (i = 0 ; i < dimension ; i = i + 1) {
        tableau[i] = i * i ;
    }
}
```

Évaluation : questionnaire « Les fonctions» sur Socrative (ISNJVERNE).