FICHE DE RÉVISION :
- PROGRAMMATION - FONCTIONS

DÉFINITION ET UTILITÉ

OBJECTIF : Comprendre comment déclarer, définir et utiliser des fonctions en langage C pour structurer et réutiliser le code.

Une fonction est un sous-programme réalisant une tâche précise.
Elle permet :

- de découper un programme en parties plus simples ;
- de réutiliser du code ;
- de clarifier la structure du programme.

CARACTÉRISTIQUES D'UNE FONCTION :

- **nom :** même consigne de composition d'un nom de variable.
- **paramètres :** éventuels, informations transmises à la fonction pour qu'elle puisse effectuer sa tâche
- **résultat :** éventuel, information retournée par la fonction, qui pourra être récupérée par le programme appelant

DÉFINITION D'UNE FONCTION

→ En C, une fonction **se déclare puis se définit.**
Objectif de la déclaration : annoncer l'**existence de la fonction**, en précisant **son nom**, le **type de son résultat**, le nombre de **ses paramètres** et le **type de chacun**

DECLARATION

SYNTAXE :

type_resultat nom_fonction (type_param1, type_param2, ...)

EXEMPLES :

- fonction appelée carre qui élève au carré un réel:
  float carre (float)

- fonction qui affiche le carré d'un réel donné ne renvoie aucun résultat au programme qui l'a appelée :
  void afficher_carre (float)

REMARQUES :


- si une fonction ne renvoie pas de résultat, alors il faut utiliser le type **void**.
- si la fonction n'a pas de paramètres, il suffit juste de ne rien mettre entre les parenthèses

ECRITURE DES INSTRUCTIONS

Définition de la fonction = écrire toutes les instructions permettant de réaliser sa tâche.

Ces instructions sont placées dans **un bloc** reprenant les **informations fournies lors de la déclaration de la fonction**, donnant en plus un **nom à chaque paramètre** pour pouvoir les utiliser dans les **instructions**, comme des variables locales à la fonction.

```
/* définition d'une fonction */
type_resultat nom_fonction(type_param1 p1, type_param2 p2,...) {
    /* déclaration des variables */
    type_resultat variable_resultat ;

    /* instructions  utilisant p1, p2, … et variable_resultat */

    return(variable_resultat);     /* renvoi du résultat */
}
```

DANS UN PROGRAMME C

Positionnement des déclarations et définitions de fonctions dans un programme C :

→ les déclarations des fonctions peuvent utiliser les **constantes nommées définies**

La fonction principale et toutes les fonctions définies ensuite savent quelles fonctions sont à leur disposition, grâce aux **inclusions de bibliothèques et aux déclarations des fonctions**.

```
/* inclusion de bibliothèques (#include) */

/* déclaration de constantes nommées (#define) */

/* déclaration des fonctions */

int main() {     /* fonction principale */
    /* déclaration des variables */

    /* instructions */
    return(0);
}

/* définition des fonctions */
```

UTILISATION D'UNE FONCTION

Une fonction déclarée (et définie) peut être appelée **dans la fonction principale** ou **dans n'importe quelle autre fonction.**

Il suffit de :

- utiliser son **nom**,
- préciser entre parenthèses l**es paramètres nécessaires** (en respectant bien sûr l'ordre de leur déclaration),
- **exploiter le résultat** que renverra la fonction, soit en l'affectant dans une variable, soit en l'utilisant directement dans une expression.

```
#include <stdio.h>

float carre (float);
void afficher_carre (float);

int main() {
    float v;  // variable locale à la fonction main

    printf("Nombre : "); scanf("%f",&v);
    afficher_carre (v);

    return(0);
}

float carre (float x) {
    float res ;  // variable locale à la fonction carre
    res = x*x ;
    return (res);
}

void afficher_carre (float y) {
    printf("%f\n", carre(y));
}
```

MODE DE PASSAGE DES PARAMÈTRES

2 MODES DE PASSAGE

→ **Passage par valeur** : La fonction reçoit une copie de la variable → la valeur d'origine n'est pas modifiée. Mode de passage par défaut en langage C.

→ **Passage par référence** : Permet à la fonction de modifier la variable originale. Le paramètre contient l'adresse mémoire de la variable il faut ajouter une étoile après le type du paramètre.

Exemple : type_resultat nom_fonction (type_param1 *p1, type_param2 p2)

→ pour accéder à la valeur stockée à cette adresse, on utilise *p1. (si p1 est une adresse, *p1 est la valeur à cette adresse.)

Lors de l'appel d'une fonction, **si un paramètre doit être transmis par référence**, il faut donc préciser qu'on ne transmet pas sa valeur mais l'adresse de sa valeur. Cela se fait en ajoutant **&** devant le nom de la variable contenant la valeur (comme dans scanf)

EXEMPLE :

La fonction suivante échange les valeurs de ses deux paramètres.

Les variables x et y sont des adresses d'entiers ; donc *x et *y sont les entiers à ces adresses.

```
void echange (int *x, int *y) {
    int z ;

    z = *x ;
    *x = *y ;
    *y = z ;
}
```

```
int a=4, b=5 ;

echange (&a, &b) ;
printf("a=%d, b=%d\n", a, b) ;  // Affiche : a=5, b=4
```

CAS PARTICULIER

LES TABLEAUX

→ Une **variable de type tableau** ne contient que **l'adresse du premier élément**.

- **Pas besoin** de mettre de **&** quand on le **passe en paramètre.**
- Il ne peut donc **pas être retourné avec un return.**

GLOSSAIRE

- **Déclaration :** la déclaration d'une fonction sert à annoncer son existence, et surtout à montrer comment l'utiliser (nom, paramètres, résultat).
- **Définition:** la définition d'une fonction consiste à écrire toutes les instructions à exécuter pour qu'elle réalise la tâche pour laquelle est est créée.
- **Résultat :** le résultat d'une fonction est l'information renvoyée (par l'instruction return) au moment de quitter la fonction ; d'autres résultats peuvent être transmis indirectement, par modification de paramètres transmis par référence.
- **Variable locale :** une variable se déclare dans une fonction ; elle est donc locale à cette fonction, car pas visible des autres fonctions ; sa valeur ou son adresse peut cependant être transmise à une autre fonction via un paramètre.
- **Void :** mot-clef désignant un type vide, utilisé pour préciser qu'une fonction ne renvoie pas de résultat.

BON COURAGE ! 💙