Cours : C:\Users\coral\Desktop\School\Cours\SOURCE MATERIAL

Nous allons désormais apprendre à découper notre programme en **fonction**. On l'a vu, chaque fonction exécute des actions et renvoie un résultat ; c'est un morceau de code qui permet de donner des consignes exactes à l'ordinateur. Chaque fonction possède une **entrée** et une **sortie**.

H1: Créer Une Fonction

Voici la syntaxte pour créer une fonction. C'est quelque chose que nous avons déjà vu, et nous allons donc reconnaître rapidement ce qui se passe.

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

📝 Explications

`type` **renvoie à la sortie**. Il s'agit du *type* de fonction. Si notre fonction renvoie un nombre décimal, on mettra `double` ; si c'est un entier, on utilise `int`. On peut aussi créer des fonctions qui ne renvoient à rien, comme par exemple `void`.

`nomFonction` **renvoie simplement au nom de la fonction.** On la nomme comme le désire, sachant qu'elle se plie aux mêmes règles que les variables (pas d'accents, pas d'espaces...)

et finalement `parametres` **qui renvoie à l'entrée de la fonction.** Il s'agit en fait simplement des valeurs avec lesquelle la fonction va travailler. On le note entre *parenthèses*.

🖍️ Exemple

Admettons une fonction `triple`. Son utilité est de multiplier par trois une valeur initiale de type `int`.

Une telle fonction se présentera comme ceci :

```
int triple(int nombre)
{
    int resultat = 0;

    resultat = 3 * nombre;  // On multiplie le nombre fourni par 3
    return resultat;        // On retourne la variable resultat qui vaut le triple de nombre
}
```

On voit appaître une nouvelle commande dans cet exemple : `return`. Cette commande indique au programme de s'interrompre et de renvoyer la valeur d'une variable, ici `resultat`.

La variable `resultat` est déclarée au sein de `{ }` de `triple` ; cela signifie qu'elle est propre à cette fonction. Elle ne peut pas être utilisée à l'extérieur de la fonction.

⚠ La variable `resultat` **doit** être de type `int` car la fonction renvoie un `int`.

⚠ On note que la fonction `triple` est de type `int`. Il est **impératif** qu'elle renvoie une valeur de type `int`. Les variables sur lesquelles la fonction se base sont entre les parenthèses. Ici, `nombre` , qui est une variable de type `int` ; tout va donc bien.

Questions

❓ La variable dans `{}` doit-elle être de même type que sa fonction ?

❓ Dans quel cas vaut-il mieux créer les variables hors de leur fonction et à l'inverse, dans quels cas faut-il créer des variables propres ?

H2: Abréger une fonction

Il n'est pas nécessaire de rédiger la fonction exemple plus haut de manière aussi exhaustive. On peut aussi l'écrire comme ceci :

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

On voit qu'on a fait sauter `resultat` pour simplement indiquer au programme d'afficher le triple de `nombre`. Le résultat sera parfaitement identique.

💡 Les fonctions sont généralement bien plus longues que ça !

H1: Affecter Des Parametres À Une Fonction

Une fonction peut contenir autant de paramètres que désiré, y comprit aucun !

Pour affecter plusieurs paramètres à une fonction, il suffit d'ajouter nos variables entre parenthèses en les séparant par une virgule, comme par exemple notre fonction `addition` juste ici :

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

A l'inverse, on peut également n'affecter aucun paramètre à une fonction ; ça se fait notamment pour les fonctions qui font toujours la même chose, comme par exemple afficher du texte à l'écran. Cependant, le texte affiché sera toujours le même, puisqu'aucune variable ne vient modifier son comportement.

Admettons par exemple la fonction suivante :

```
void bonjour()
{
    printf("Bonjour");
}
```

Cette fonction affiche *Bonjour* à l'écran dès qu'elle est appellée. On remarque qu'il n'y a rien entre les parenthèses, puisque la variable n'a aucun paramètre associé. On notera également que la variable n'a pas de `return` : elle ne retourne rien.

📝 Une fonction qui ne retourne rien est de type `void`.

H1: Appeler Une Fonction Dans Le `main`

On reprend notre fonction `triple` qu'on va appeler dans le `main`.

On rédige cela de la façon suivante :

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

int triple(int nombre) // on déclare la fonction triple et la variable nombre

{
    return 3 * nombre; // consignes de la fonction triple : valeur de "nombre x 3"
}

int main(int argc, char *argv[]) // on ouvre la fonction main

{
    /* on déclare les deux variable dont on a besoin pour notre programme : "saisie" pour le
nombre
    que va saisir l'utilisateur, saisie_triple pour afficher le résultat à l'utilisateur.*/

    int saisie = 0, saisie_triple = 0;

    printf("Saisir un nombre... \n");
    scanf("%d", &saisie); // l'utilisateur saisi une valeur

    /*on déclare que la valeur de saisie_triple est égale à saisie "passé" dans la fonction
triple*/

    saisie_triple = triple(saisie);

    printf("Le triple de ce nombre est %d !\n", saisie_triple); // on affiche le résultat du
calcul.

    return 0;
}
```

📝**Explications**

Concentrons nous un instant sur la ligne qui appelle la fonction.

```
`saisie_triple = triple(saisie);`
```

Dans les paramètres, entre les parenthèses, on envoie à `triple` une variable en **entrée**. C'est la valeur avec laquelle elle va travailler. Elle est déterminée par l'utilisateur plus haut dans `main`.

`triple` renvoie ensuite la valeur qu'elle a calculée dans `saisie_triple`.

En gros, cette ligne dit à la machine "Demande à `triple` de calculer le triple de `saisie`, et renvoie la nouvelle valeur dans `saisie_triple`."

H1: Comprendre Comment La Machine Lit Une Fonction.

Reprenons notre exemple, commenté différemment.

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

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

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

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

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

    return 0; // 9
}
```

On peut le voir, l'ordinateur ne commence pas par le début ; il commence les instructions par le `main`.

Voilà ce qui se passe en bref dans la machine quand on lance le programme :

1. Le programme entre dans le `main`
2. Il lit les instructions ;
3. Il lit le `printf` et l'exécute ;
4. Idem pour le `scanf`
5. Dans l'instruction suivante, il faut qu'on appelle la fonction `triple` ; il va donc la chercher en ligne 6.
6. Il cherche le paramètre `nombre` dans et exécute les calculs demandés, puis...
7. ...il termine la fonction grâce à `return`.
8. Il retourne dans le `main` pour lire l'instruction suivante.
9. Il trouve un `return` - il sait donc que le `main` se termine, et le programme est terminé.

💡 Dans notre exemple, on a stocké le résultat du calcul dans `saisie_triple`. Cette étape est facultative ; on peut envoyer le résultat de `triple` directement au `printf`, comme si `triple(saisie)` était une variable.

On procède de la façon suivante :

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

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

int main(int argc, char *argv[])
{
    int saisie = 0; // on a supprimé saisie_triple

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

    // on envoie directement le calcul au printf en faisant comme si la fonction triple était une variable

    printf("Le triple de ce nombre de %d\n", triple(saisie));

    return 0;
}
```

Exercices

- [x] ~~Ecrire une fonction qui répète une ligne de texte.~~

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

void bisou(int how_much_bisou) // le void bisou prend un paramètre : combien de fois on veut dire que c'est le bisou.

{
    int amour = 1; // on initialise amour à 1 pour éviter les boucles infinies

    while (amour <= how_much_bisou) // tant que amour est inférieur ou égal à how_much_bisou, on printf

    {
        printf("It is %d l'amour \n", amour); // message à imprimer avec la valeur de amour pour suivre l'incrémentation
        amour++; // on incrémente amour
    }
}

int main(int argc, char *argv[]) // call main
{
    printf ("Do you know how much l'amour it is ? \n\n"); // print
    bisou(200); // on initialise how_much_bisou à 200

    return 0;
}
```

- [x] ~~Ecrire une fonction qui calcule l'aire d'un rectangle, basée sur des données saisies par l'utilisateur~~

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

int aire(int largeur, int longueur)
{
    return largeur * longueur;
}

int main(int argc, char *argv[])
{
    int largeur = 0, longueur = 0, resultat = 0;

    printf("Saisissez la largeur de votre rectangle... : ");
    scanf("%d", &largeur);

    printf("Saisissez la longueur de votre rectangle... : ");
    scanf("%d", &longueur);

    resultat = aire(largeur, longueur); // on appelle la fonction AVEC LES PARAMETRES !!!

    printf("L'aire de votre rectangle est %d ! \n", resultat);

    return 0;
}
```

- [x] ~~Ecrire une fonction qui présente un menu et renvoie le choix de l'utilisateur~~

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

int chemin()
{
    int choix = 0;
    while (choix < 1 || choix > 4)
    {
        printf("Please pick a path...\n\n");

        printf("1. The Waterway: a waist-deep stream of icy water that cuts through the
dungeon.\n\n");

        printf("2. The Lava Tubes: a network of tunnels that wind through the heart of an active
volcano.\n\n");

        printf("3. The Crypt: a dark and musty crypt that is home to the remains of ancient
kings and queens.\n\n");

        printf("4. The Goblin Warrens: a maze of twisting tunnels and caverns that are home to a
tribe of goblins.\n\n");

        printf("Which way will you go ?");
        scanf("%d", &choix);
    }

    return choix;
}

int main(int argc, int *argv[])

{
    switch (chemin())

    {
    case 1:
        printf("You have chosen to wade through The Waterway.\n\n");
        break;

    case 2:
        printf("You have chosen to brave The Lava Tubes.\n\n");
        break;

    case 3:
        printf("You have chosen to crawl in The Crypt.\n\n");
        break;

    case 4:
        printf("You have chosen to enter The Goblin Warrens.\n\n");
        break;
    }

    return 0;
}
```

H2: En bref

- Les fonctions s'appellent entre elles. `Main` peut appeller des fonctions toutes prêtes, telles que `printf` (ou d'autres qu'on peut inclure avec des bibliothèques), mais également des fonction qu'on écrit nous-même.
- Les fonctions récupèrent des paramètres en entrée. Il faut spécifier chaque paramètre lorsqu'on appelle une fonction, c'est à dire les répéter entre les parenthèses.
- La fonction réalise des opérations prédéfinies avec les paramètres, puis retourne la valeur résultat à l'aide de l'instruction `return`.