PARTIE 3 : LES FONCTIONS

Bon on va revoir un peu les fonctions car jusqu'à maintenant nous avons vu qu'un usage très limité de celles-ci.

Reprenons la fonction main, son prototype est :

int main(void);

Qu'est-ce que c'est ce **int** devant ? tu te souviens qu'on met toujours un return à la fin d'une fonction ? pourquoi on fait ça ?

Il faut savoir que ce qu'il se passe dans une fonction reste dans la fonction ! (Comme à las Vegas youpiiii !).

Si je fais :

```
#include <stdio.h>

void                    test(void)
{
        int a;

        a = 5;
        return ;
}

int                     main(void)
{
        printf(a);
        return (0);
}
```

Tu peux essayer, tu verras que tu as une erreur car la variable **a** n'existe pas dans le main. Il ne faut pas oublier que le programme quand tu le lance commence par la fonction main en premier et exécute ligne par linge le code.
Le code dans la fonction **test()** fonctionne parfaitement seulement, dans la fonction **main()** aucune variable **a** n'est déclarée.

On pourrait penser que si on fait :

```
#include <stdio.h>

void                test(void)
{
        int a;

        a = 5;
        return ;
}

int                 main(void)
{
        int a;

        printf(a);
        return (0);
}
```

Ça fonctionne .... Et bah non ! ce qui reste dans une fonction reste dans une fonction ! sauf le **return();**
Ce petit filou est le moyen de communiquer entre une fonction et le main ! **Mais on ne peut avoir que 1 seul return par fonction car après le return() on sort de la fonction même si il reste du code après**!
Tu te demandes alors pourquoi on a pu afficher des nombres etc. avant ? grâce à l'appelle système write();
printf() aussi utilise write();
Dans une fonction, tout ce qui est créé, est "effacé" à la fin de cette fonction. Tout ce qu'on affiche à l'écran bah reste puisque qu'on la déjà affiché mais les variables elles ne sont plus là. On ne peut plus les utiliser. Après le return() la variable **a** est effacée ;

Mais des fois on souhaite récupérer cette variable dans le **main** comment peut-on faire alors ? en utilisant **return();** qui veut dire en français **retourner**, retourner dans le sens. Renvoyer quelque chose.

Il faut donc à la fin de la fonction test faire : **return(a);**

Mais on est d'accord que **a** est une variable de type **int**, donc c'est un nombre ? (Attention c'est un peu compliqué la suite)
Du coups, vu que la variable que l'on veut retourner est un **int** (une variable de type int) on doit modifier le début de la fonction. Ce n'est plus void mais :

```
int         test(void)
{
        int a;

        a = 5;
        return (a);
}
```

Bon jusque-là ça va ?
Maintenant occupons du **main()** car c'est pas fini ! (si j'écris un mot avec () juste après c'est que je parle d'une fonction).

Notre fonction test renvoie désormais **a** et **a = 5**, donc la fonction renvoie 5, il faut maintenant récupérer tout ça !
Tu te souviens que j'ai dit que les fonctions étaient des "raccourcis" ? bah c'est parce que l'ordinateur va remplacer

```
int     main(void)
{
        test();
        return(0);
}
```

par

```
int a;

a = 5;
return (a);
```

C'est un résumer un peu rapide. Mais c'est à peu près ça. Le programme fait faire toute la fonction test(); dans une petite bulle.

A ce stade si tu test ce code rien ne se passera car il faut stocker le return(); quelque part.

```
#include <stdio.h>

void            test(void)
{
        int a;

        a = 5;
        return ;
}

int             main(void)
{
        test();
        return (0);
}
```

On peut par exemple la stocker dans une variable et l'afficher :

```
#include <stdio.h>

void            test(void)
{
        int a;

        a = 5;
        return ;
}

int             main(void)
{
        int b;

        b = test();
```

```
        printf(b);
        return (0);
}
```

b vaut maintenant 5.
Ou sinon l'afficher directement dans printf:

```
#include <stdio.h>

void            test(void)
{
        int a;

        a = 5;
        return ;
}

int             main(void)
{
        int b;

        printf("la variable a = %d", test());
        return (0);
}
```

On peut effectivement mettre la fonction test() à la place de la variable dans printf. Seulement le résultat ne sera stocké nul part. C'est pratique si on veut juste afficher un résultat sans rien stocker.

On peut aussi faire une fonction test2 qui elle renvoie un caractère :

```
#include <stdio.h>

char    test2(void)
{
        char c;

        c = 'a';
        return (c);
}

int     main (void)
{
        char c2;

        c2 = test2();
        printf("c2 = %c", c2);
        return (0);
}
```

A NOTER : vu que ce qu'il se passe dans une fonction reste dans la fonction, on peut très bien appeler ces variables avec le même nom dans le main() et dans la fonction créée. Par contre on ne peut pas donner le même nom a 2 variables dans une même fonction. Exemple :

Interdit de faire ça :

```
int main(void)
{
        char c;
        char c;

        c = 'a';
        return (0);
}
```

Par contre ça c'est possible:

```
char            test(void)
{
        char c;

        c = 'a';
        return (c);
}

int             main(void)
{
        char c;
        char alibaba;

        c = 'b';
        alibaba = test();
        return (0);
}
```

Car main() et test() sont deux fonctions différentes. Mais attention : elles n'ont rien à voir entre elle. Ce sont deux variables différentes même si elles portent le même nom.

Bon passons maintenant aux arguments !
Une fonction peut prendre des arguments, qu'est-ce qu'un argument ?
C'est une variable qu'on met dans une fonction. Jusque-là, on a laissé les arguments en void:

```
int     test(void)   <= argument
{
}
```

A quoi servent-ils ? par exemple si on souhaite additionner 5 + 3, puis 8 + 15, puis 98 + 1034.
Ça serait embêtant de faire :

```
int		addition1(void)
{
	int nb;

	nb = 5 + 3;
	return (nb);
}

int		addition2(void)
{
	int nb;
	nb = 8 + 15;
	return (nb);
}

int		addition3(void)
{
	int nb;
	nb = 98 + 1034;
	return (nb);
}

int	main(void)
{
	int nb;

	nb = addition1();
	printf("add1 = %d", nb);

	nb = addition2();
	printf("add1 = %d", nb);


	nb = addition3();
	printf("add3 = %d", nb);

	return (0)

}
```

Un peu chiant et long non ? Pas de panique ! Les arguments sont là pour ça ! on va ainsi crée une seule fonction :

```
int     addition(int x, int y)
{
        int nb;

        nb = x + y;
        return (nb);
}

int     main(void)
{
        int nb;

        nb = addition(5, 3);
        printf("add1 = %d", nb);

        nb = addition(8, 15);
        printf("add1 = %d", nb);

        nb = addition(98, 1034);
        printf("add1 = %d", nb);

}
```

C’est mieux non ? En fait ce qu'on a fait, c'est qu'on a dit que la fonction addition aller prendre deux variable de types int donc, deux variables nombre qui vont s'appeler x et y.

Et dans la fonction main on additionne x et y.
x devient 5 puisque dans la fonction main c'est le premier chiffre entre parenthèse, et,
y devient 3

Puis quand on rappelle la fonction pour la deuxième fois cette fois on dit que
x devient 8
x devient 15

La fonction main prend aussi des arguments, maintenant qu’on sait ce que c’est un argument voyons ce de la fonction main !

```
Int     main(int argc, char **argv)
{
        Return (0) ;
}
```

Désormais, on écrira toujours la fonction main avec ses arguments ! comme tu l’as peut-être deviné.

Int argc

Est une variable de types, c’est donc un nombre.

char **argv

Est un type de variable qu’on a pas encore vu on y reviendra plus tard. On n’utilisera pas les arguments de la fonction main pour le moment.

Le but réel des fonctions c'est ça ! ne pas avoir à répéter le code, parce qu'un bon programmeur est un programmeur flemmard :D

Bon aller démonstration un peu plus concrète :
Même si c'est moche et peut être pas comme tu as l'habitude, On va créer un jeux d'aventure textuel au fil des cours et exercices. Donc pas de graphisme que des textes. Mais cela va t'apprendre beaucoup sur le début des jeux vidéo et tu pourras ensuite le modifier à ta guise.

Regarde cette vidéo pour mieux comprendre de quoi il s'agit.
https://www.youtube.com/watch?v=RSBTcqvETzU

Bon ça va se faire petit à petit parce que y'a plein de truc à apprendre encore !

Donc voici le début du jeu :

```
#include <stdio.h>

Int                    main(int argc, char **argv)
{
        int point_de_vie ;
        int atk_alex ;

        point_de_vie = 100 ;          //Le joueur part dans l'aventure avec 100 point_de_vie
        atk_alex  = 53 ;              //ton attaque de base = 53
        printf(« bienvenue dans mon nouveau jeu \n\n» ) ;
        printf(« Vous vous appelez Alex et vous avez %d  de point de vie\n», vie) ;  \\ on previent le joueur qu'il a des point_de_vie
        printf(« Vous marchez dans la foret quand tout a coups …. \n») ;
        point_de_vie = combat_dragon(point_de_vie, 2, atk_alex) ;   /* on va créer une fonction combat qui va faire perdre des points de vie a notre joueur, il perdra 2 point de vie */
}
```

La fonction se met en haut de la fonction main mais pour que tu puisses suivre étape par étape dans le cours je la mets en dessous.

Donc maintenant on créer notre fonction combat_dragon qu'on place au-dessus du main

```
int                     combat_dragon(int point_de_vie, int atk_dragon, int atk_alex)   //fonction qui
renvoie un int
{
        Int         pv_dragon ;

        pv_dragon = 100 ;
        printf(« Un dragon sauvage apparait ! \n») ;
        printf(« Le combat s'annonce violent … \n ») ;
        printf(« le dragon attaque et vous fais perdre %d de point de vie », coup) ; // coup = 2
        point_de_vie = point_de_vie  - atk_dragon ;
        printf(« il reste %d de point de vie a alex », point_de_vie) ;
        printf(« Alex attaque est fait perdre %d  de point de vie au dragon », atk_alex) ;
        pv_dragon = pv_dragon – atk_alex ;
        printf(« il reste %d de point de vie au dragon», pv_dragon) ;
        printf(« le dragon attaque et vous fais perdre %d de point de vie », coup) ; // coup = 2
        point_de_vie = point_de_vie – atk_dragon ;
        printf(« il reste %d de point de vie a alex », point_de_vie) ;
        printf(« Alex attaque est fait perdre %d  de point de vie au dragon\n», atk_alex) ;
        pv_dragon = pv_dragon – atk_alex ;
        printf(« le dragon attaque et vous fais perdre %d de point de vie\n», coup) ; // coup = 2
        point_de_vie = point_de_vie – atk_dragon ;
        printf(« il reste %d de point de vie a alex\n», point_de_vie) ;
        printf(« Alex attaque est fait perdre %d  de point de vie au dragon\n», atk_alex) ;
        pv_dragon = pv_dragon – atk_alex ;
        printf(« il reste %d de point de vie au dragon\n», pv_dragon) ;
        printf(« Le dragon est mort ! \n») ;
        return (point_de_vie) ;
}
```

Retournons dans le main et ajoutons d'autres combat :

```
Int                     main(int argc, char **argv)
{
        int point_de_vie ;
        int atk_alex ;

        point_de_vie = 100 ;
        atk_alex  = 53 ;
        printf(« bienvenue dans mon nouveau jeu \n\n» ) ;
        printf(« Vous vous appelez Alex et vous avez %d  de point de vie\n», vie) ;  \\ on previent le
joueur qu'il a des point_de_vie
        printf(« Vous marchez dans la foret quand tout a coups …. \n») ;
        point_de_vie = combat_dragon(point_de_vie, 2, atk_alex) ;
        printf(« super combat ! \n») ;
        printf(« alex marche encore pendant 5minutes  quand tout a coupps …») ;
        point_de_vie = combat_dragon(point_de_vie, 4, atk_alex) ;
}
```

Tu peux voir que à chaque combat les points de vie d'Alex descendent, et que pour combattre un dragon on est pas obligé de tout retaper à chaque fois, on peut même modifier l'atk du dragon dans les arguments de la fonction !
Là tu peux voir correctement tout ce que tu as appris mis en application.

Petit exercice pour le plaisir : tu peux créer une nouvelle fonction qui s'appelle par exemple : combat_ogre() et faire combattre Alex contre des dragons ou des ogres !

Ce n'est pas mal non ? bon pour le moment on peut pas trop interagir avec le programme, mais le plus gros problème c'est qu'on se retrouve avec des points de vie négatif a la mort du dragon.
Ce n'est pas très classe, dans un jeu on ne voit jamais :

« Il reste -6 pv au dragon »

Ce qui serait bien, ça serait de dire au programme « si les pv du dragon descendent à 0 ou en dessous de 0, on dit qu'il reste 0pv au dragon et il meurt ».

C'est ce qu'on va voir avec la prochaine partie justement ! **LES CONDITIONS !**