PARTIE 2 : LES VARIABLES

A noter que le ; est TRES important on le met à la fin de chaque ligne quasiment et l'oublier = impossible de compiler le programme.
Une ligne qui ouvre des accolades : { } n'a pas besoin de point-virgule à la fin c'est la seule exception. C'est pour ça que int main(void) n'a pas de point-virgule a la fin de la ligne

Ce n'est pas mal les variables hein ?

Il est important de comprendre qu'une variable sert à stocker une information c'est pour cela que nous avons créé nombre3. Car nombre3 va prendre le résultat de l'addition entre les deux variables (nombre1 + nombre2). Ce n'est pas obligatoire mais si tu ne stock pas le résultat dans une variable, ton ordinateur va faire le calcul mais n'affichera rien à l'écran.

On peut aussi faire : **nombre2 = nombre1 + nombre2;**

Ça fonctionne aussi bien car nombre2 stockera le résultat après l'addition. On n'a donc pas besoin de créer une variable nombre3 mais c'est plus propre avec une troisième variable.

Par contre nombre2 après l'addition ne vaudra plus 1 mais le nouveau résultat.

Je te laisse essayer avec la multiplication et la soustraction !

Pourquoi il y a plusieurs types de variable pour les nombres ? parce que avant la mémoire les ordinateurs étaient très limitée ! il fallait utiliser le moins de mémoire possible. Or plus un nombre est grand plus il prend de la place. En informatique tous les chiffres et nombres sont codé sur 1 octet.

1 octet = 8bit. Je m'explique :

On a l'habitude de compter, jusqu'à 9 après on passe à 10 etc ... on appelle ça **la base 10** car on il y a 10 chiffres (0-9).

On le chiffre maximum est 9 (après c'est des nombres). Pour les nombres on utilise les 10 premiers chiffres. => 1 et le 0 pour faire 10.

Il existe d'autre moyens comme l'hexadécimal, c'est une base 16 c'est à dire qu'on va compter 0 1 2 3 4 5 6 7 8 9 A B C D E F

A = 10
B = 11
...
F= 15

Après F il y a 10, sauf qu'en hexadécimal 10 = 16.

Et on retrouve la base 2 qui est le binaire. Il n'y a que deux chiffres 1 et 0, donc pour compter jusque 5 ça donne :

```
 0    = 0
 1    = 1
10    = 2
11    = 3
100   = 4
101   = 5
```

Voir la figure après.

Figure1 : base 10 (il manque des ‘s’ un peu partout mais OSEF)

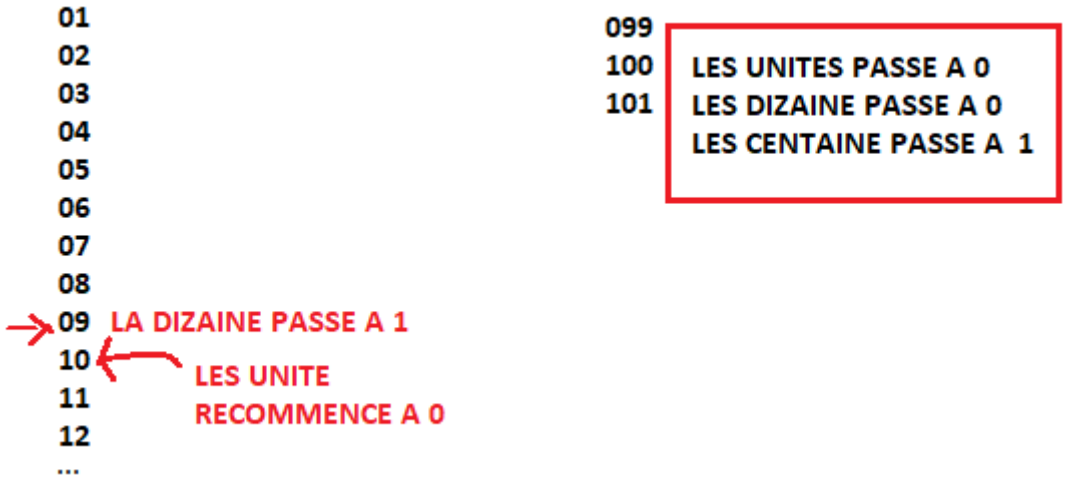
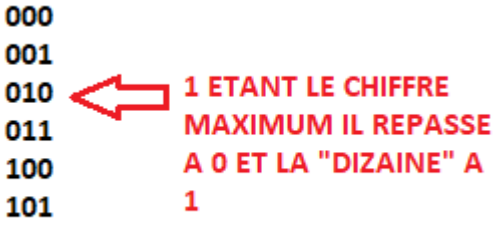


Figure2 : base 2



Tout ça pour dire que l'ordinateur ne comprend que le binaire, que des 0 et des 1.
On a dit tout à l'heure 1octet = 8bit, 1 bit au final c'est un chiffre.

100 = 3bit

Et un chiffre même 0 et toujours égal à 1 octet. L'ordinateur écrit 0000 0000 au lieu d’écrire 0 comme nous.

2 = 0000 0010

C’est comme si en base 10 l'ordinateur écrivait 0000 0150 au lieu de 150. Nous, on supprime les 0 inutiles devant les chiffres => on n’écrit pas 05 mais, 5.

fiou ! voilà ce qu'est un octet !

En fait le type char est spéciale : il peut stocker des caractères et des nombres. Pourquoi ? car quand on fait : char c = ‘a’ ;
L’ordinateur va transformer ‘a’ en 97 au moment de lancer le programme. Mais il est le seul type à faire ça.
Donc le type char lui est coder sur 1 octet donc son chiffre maximum en binaire = 1111 1111 soit 255 en base 10.
Si on essaye de faire :

char a = 345;

Très probablement un bug va apparaitre !
Parce que l'ordinateur ne comprend pas, on lui a dit que le maximum était 255,
Cependant, le nombre minimum qui peut être stocké dans un char est le nombre : -127 et le maximum = +128.

128+127 = 255, le compte est bon. Si on fait :

char a = 130 ;

Très probablement un bug va apparaitre !

Pour stocker des plus grands nombres comme 3000 ou 190 ;

Il faut utiliser un int qui lui est coder sur 4 octet son nombre maximum est 1111 1111 1111 1111 soit 2 147 483 647 ! c'est tout de même mieux mais ça prend + de place en mémoire. De nos jours nos mémoires sont incroyablement grandes donc ce n'est plus trop un problème par convention on utilise toujours int pour déclarer un nombre entier et float pour un nombre à virgule ! (Même si le nombre est 3 on utilise int mais un char est tout à fait possible)

Il existe aussi le type :

unsigned

Le unsigned se place devant char par exemple.

unsigned char a = 3

La particularité du unsigned c'est qu'il permet de doubler le nombre maximum qu'on peut stocker dans un type. Dans un unsigned char par exemple le nombre maximum à pouvoir être stocker n'est pas 128 mais, 255.
En contrepartie on ne peut plus y mettre un nombre négatif. Donc on peut y stocker un nombre entre 0 et 255.

Voici un résumé des types possibles :

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$3.4*10^{-38}$ à $3.4*10^{38}$
double	Flottant double	8	$1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant double long	10	$3.4*10^{-4932}$ à $3.4*10^{4932}$

Bon ce n'est pas évident à comprendre donc voici un résumé :

- **Pour les nombres entiers on utilisera le type int**
- **Pour les nombres à virgule on utilisera le type float**
- **Pour les caractères on utilisera le type char**

Mais il faut faire attention à ne pas dépasser les valeurs maximums positives et négatives autorisées par les types !

Pour les variables, c'est fini pour le moment.

On va voir les opérations standard disponibles :

Addition	+
Soustraction	-
Multiplication	*
Divisons	/
Modulo	%

Mais qu'est-ce que le modulo ? peut-être que tu le sais mais je vais quand même l'expliquer au cas où.
Le modulo c'est une division mais qui va nous donner que le reste (ça fait une division euclidienne). Par exemple : 10 / 3 = 3.33 mais en programmation si on n'utilise pas le type **float** le résultat de la division sera 3. Donc 10 / 3 = 3 en programmation sauf si on utilise le type float.

En division euclidienne **10 / 3 = 3** et il **reste 1**. C'est le modulo en programmation. **10 % 3 = 1.**
Le modulo n'affiche que le reste de la division.

Très pratique ! Tu veux un exemple concret ? super ! mini EXERCICE :