**Variables & Structures de contrôle – 26 mai 2022 / GL**

Un programme informatique manipule des **données** (stockées dans des **variables**)

```
byte B;      // chaque variable doit être déclarée afin de lui réserver de l'espace en
             // mémoire
B=5;         // B est égal à 5 (affectation)
             // le type byte occupe 8 bit si bien que les valeurs possibles sont comprises
             // entre 0 et 255 (0 <= B <= 255)
```

Il est donc vital de comprendre le calcul binaire !
Quelques affectations identiques :

```
B=5;         // Par défaut la base utilisée est décimale (comme vos 10 doigts)
B=0b101;     // Base binaire (valeur possible = 0 ou 1)
             // Le poids de chaque bit est égale à 2^N
             // Le bit de poids faible est situé à droite ; son poids = 2^0 = 1
             // On l'appelle LSB = Least Significant Bit
             // Le bit suivant a un poids = 2^1 = 1 mais il n'est pas présent
             // Le bit suivant a un poids = 2^2 = 4 → B est bien égal à 5
```

La calculatrice de votre PC, en mode programmeur, peut vous aider dans votre apprentissage des bases (décimale, binaire, octale, hexadécimale) et faciliter les conversions


**A propos des bases :**

- La base 10 (= base décimale) comprend 10 symboles différents = 0123456789
- La base 2 (= base binaire) 2 01
- La base 8 (=base octale) 8 01234567
- La base 16 (=base hexadécimal) 16 0123456789abcdef

Quel est le résultat de : `byte B; B=1000; ?`
Calculatrice → `B=1000;` est équivalent à `B=0b1111101000;`
byte de 8 bit `B=0b0011101000;` → `B=232;`
La donnée 1000 a été tronquée sur 8 bit = taille disponible du type byte

**Travail pratique :** mettre en œuvre ces éléments théoriques

1. Lancer Arduino IDE
2. File – New
3. Editer les lignes précedentes à l'intérieur de setup() {…}
4. Respecter la syntaxe
5. Lancer la compilation 
6. Corriger au besoin en vous aidant du(des) message(s) d'erreur
7. Charger l'exécutable sur la carte Arduino 

Quelques types de données (extrait de https://www.arduino.cc/reference/en/) :

```
char c;      // occupe 1 byte
word d;      // occupe 2 byte consécutifs → valeur = 0 à 65535
long e;      // occupe 4 byte consécutifs → Référence
int f;       // occupe 4 byte consécutifs
```

**Baptiser une variable est une opération IMPORTANTE & PERSONNELLE**

Certain préfère la simplicité (mon cas) `char c;`

d'autres priviégient des noms explicites tels que `char caractere_en_provenance_du_clavier;`

Il peut être utile au débutant de préciser le **type des variables** pour éviter des erreurs telles que `B=1000;`

```
byte B_byte;
byte B_8bit;
```

Tous les langages de programmation imposent des règles strictes appelées **syntaxe** du langage

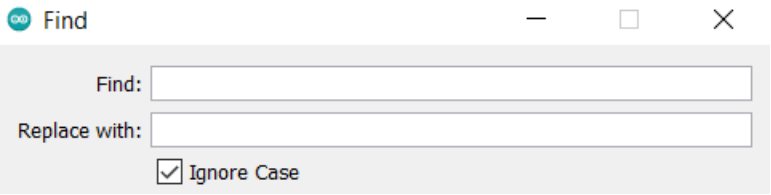
Le langage couramment utilisé dans la communauté Arduino est le langage C qui exige :

- le point-virgule après chaque instruction
- 2 barres obliques pour un commentaire
- …

Pour plus de détails → https://fr.wikibooks.org/wiki/Programmation_C/Bases_du_langage

La syntaxe du langage C utilisé distingue majuscule et minuscule → on parle de **case sensitive**

La recherche avec Find dans l'IDE est par défaut = Ignore Case Sensitive




Un programme informatique manipule des données selon un **algorithme** = une séquence ordonnée d'instructions pouvant inclure des **tests** (sauts conditionnels) et des **boucles** :

```
byte B;
if (B>5) {Serial.println("B est plus grand que 5");}   // Afficher ce texte si B > 5
// (Condition)
//       {Action}

if      (B<3) {Serial.println("B est inférieur à 3");}  // Vrai pour B=0,1,2
else if (B>5) {Serial.println("B est supérieur à 5");}  // Vrai pour B>5

if      (B<3) {Serial.println("B est inférieur à 3");}  // Vrai pour B=0,1,2
else if (B>5) {Serial.println("B est supérieur à 5");}  // Vrai pour B>5
else          {...}                                     // Vrai pour B>=3 et B<=5
```

Les 5 tests précédents sont exclusifs : si B est inférieur à 3
sinon, si B supérieur à 5
sinon …

```
switch (B) {
  case 1 :          Serial.println("B=1");                          break;
  case 2 : case 5 : Serial.println("B=2 ou B=5");                   break;
  default :         Serial.println("B est différent de 1, 2, 5");   break;
}
```

**L'instruction break est vitale pour sauter à la fin de switch !**

Essayer d'enlever le premier break pour comprendre le mécanisme !!!

L'exemple précédent avec l'instruction if

```
if (B==1) {Serial.println("B=1");}  // Piège à éviter pour le débutant
```

**Ex1** : utiliser l'instruction **switch** pour gérer les données du clavier

Le **tableau** est un type de données très utile :

```
char Tab[5];       // déclaration d'un tableau de 5 caractères
Tab[0]='a';        // le premier élément (=0) du tableau est égal au caractère ASCII a
Tab[1]='b';        // le 2ème élément (=1)                                          b
...
```

Manière plus simple d'initialiser ce tableau :

```
char Tab[] = {'a','b','c','d','e'};
```

Le très ancien code ASCII est encore utilisé aujourd'hui.
Il utilise 7 bit si bien que la colonne Dec (=décimale) démarre à 0 et se termine avec 127
Les 3 colonnes de droite correspondent à des caractères imprimables :

```
 97 61 141 &#97;  a
 98 62 142 &#98;  b
 99 63 143 &#99;  c
100 64 144 &#100; d
101 65 145 &#101; e
```

Le code ASCII du caractère a ('a') est égal à 97

L'initialisation du tableau peut donc aussi s'écrire : `char Tab[] = {97,98,99,100,101};`

L'instruction **for** va faciliter ce travail répétitif :

```
byte N;      // compteur
for (N=0; N<5 ;N++) {Serial.print(Tab[N]);}
```

Pour N=0,1,2,3,4

La **fonction** convient également au travail répétitif :

```
void Print(char C) {Serial.print(C)}
```

**Ex2** : afficher le contenu du tableau ligne par ligne, en décalant chaque ligne de 1 caractère :

```
abcde
 bcdea
  ...
```

Créer la fonction PrintLine(int Begin, int End)
Solution → http://gelit.ch/Arduino/Ex2.ino

L'instruction **while** mérite d'être connue !

```
B=1; Serial.println("B=1");
while(1);
B=2; Serial.println("B=2");  // cette instruction n'est jamais exécutée car le processeur
                             // boucle sur l'instruction while(1);
```

L'utilité peut être d'arrêter l'exécution en cas d'erreur fatale

```
if (...) {Serial.println("Panne électrique"); while(1);
```

L'instruction **do ... while** est très utile !

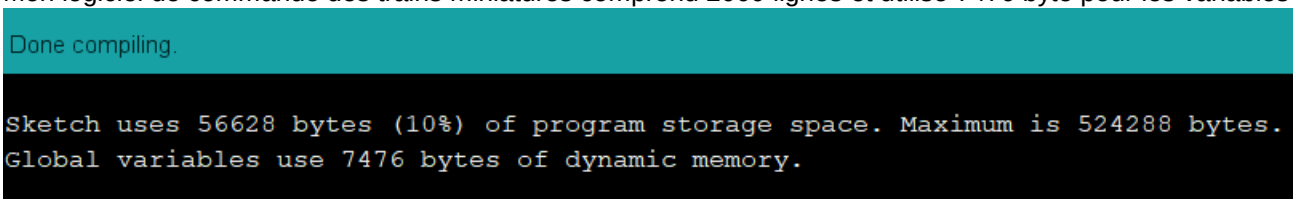
```
char c;
do {} ... while (Serial.available > 0)  // Attendre un caractère en provenance du clavier
c = Serial.read();                      // Lire ce caractère
```

Les **données** (du clavier, temporelles, …) sont par définition **variables** et résident dans la mémoire RAM

Le compilateur (langage C dans nos exemples) va traduire le programme (code) source en instructions du processeur codées sur 32 bit

Arduino Due comporte un puissant processeur 32 bit travaillant à la fréquence de 84 MHz
Son espace de stockage pour les données avec accès en lecture & écriture = 96 k byte
Remarque le "kilo" k est une puissance de 2 qui vaut 1024

Mon logiciel de commande des trains miniatures comprend 2000 lignes et utilise 7476 byte pour les variables




Les instructions du programme sont mémorisées sur une mémoire flash semblable à une mémoire USB
La capacité occupée dans l'exemple = 56628 byte ; soit environ 10% de l'espace disponible