# EduC – Programmer en C/Arduino - Module 5
# Tableaux et afficheur 7-segments

## Tableaux

Un tableau de variables est une suite de tiroirs qui se déclare comme une variable: il suffit de préciser entre `[]` combien de cases mémoire réserver. Les numéros des cases (index) commencent par 0. Par exemple, `int table [3];` réserve 3 mots de 16 bits en mémoire. que l'on peut voir comme 3 variables `table[0] table[1] table[2]`. On peut écrire par exemple `table[2]++;` pour incrémenter la 3e variable du tableau. 3e variable, numéro 2. Les habitudes données à l'école de numéroter à partir de 1 génèrent souvent des erreurs! Dans `int table [3];` 3 est une taille exprimée dans le langage courant. Les positions mémoire sont toujours à partir de zero, les compteurs sont toujours initialisés à zero.
On peut remplir la table en la définissant: `table[]={37,42,18,7}`. Le compilateur sait compter et va noter que la taille est 4. On peut demander cette taille avec la fonction `sizeof (nomDeLaTable);`

## Exemple (difficile?)

On a des produits numérotés 1,2,3,4 avec des prix différents: 12, 5, 7 et 13.
On peut faire une table avec les prix : `byte taPrix [] = {0,12,5,7,15};`
Il a fallu inventer une valeur pour le produit 0, qui n'existe pas.
Si on veut le prix du produit 2, on écrit `prix= taPrix [2];` et `Dec8(prix);` va afficher `5`.
On ne sait pas ce qui se trouve en dehors de la table.
Si le client demande le prix de l'objet nn. Que faut-il programmer comme fonction GetPrix (nn) ?

```
byte  GetPrix (byte nn) {
  byte pp;
   if (nn==0)  { pp=0; Text("Pas de produit 0"); }
   else if  (nn> sizeof (taPrix)) { pp=0; Text("Pas de produit si grand"); }
   else {pp= taPrix [nn];} Text("Le prix est "); Dec8 (pp);
}
```

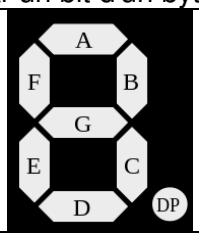
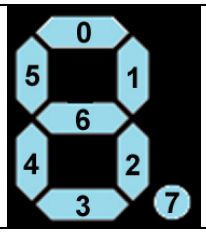
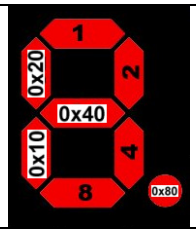
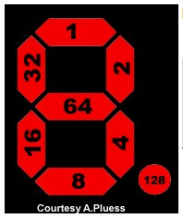
Exemple complet

```
//Edu51aPrix.ino Exemple dâccès à un tableau
#include "EduC.h"
#include "Oled.h"
void setup(){ SetupEduC(); SetupOled();}

byte taPrix [] = {0,12,5,7,15};
void  GetPrix (byte nn) {
  LiCol (2,0);
  byte pp;
   if (nn==0)  {
     Text("Pas de produit 0");
   }
   else if (nn> sizeof (taPrix)) {
     Text("Pas de produit si grand");
   }
   else {
     pp= taPrix [nn]; Text("Le prix est "); Dec8 (pp);
   }
}
byte noProduit,prixProduit;
void loop() {
  noProduit=2;
  GetPrix (noProduit);
  while(1);
}
```

## Affichons des chiffres

Un affichage 7-segments est une solution économique pour afficher des chiffres, et même des lettres (le t, v,w,x,y,z posent problème). Les segments sont toujours nommés de la même façon et chaque segment est piloté par un bit d'un byte.

| Le segment a est câblé sur le bit 0, son poids est 1<br>Le segment g est câblé sur le bit 6, son poids est 40 en hexa, 64 en décimal |  |  |  |  |
|---|---|---|---|---|
| | noms | no de bits | poids en hexa | poids en décimal |

Pour fomer un 4 on allume B C F G donc les bits 1 2 5 6 sont actifs (à 0V pour notre circuit). Le code est 01100110=0x66. Le microprocesseur devra envoyer l'inverse (10011001).On raisonne toujours avec ce qui est fonctionnel. Sîl faut inverser, c'est une instruction qui le fera. Nous, on risque de se tromper.
On peut raisonner avec les poids en décimal: 2+4+64+32 = 102. C'est le compilateur qui traduira en binaire.

## Table des segments

| s | BCD | G | F | E | D | C | B | A |
|---|---|---|---|---|---|---|---|---|
| 0 | 0000 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0001 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0010 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 3 | 0011 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0100 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 5 | 0101 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 6 | 0110 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 7 | 0111 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1001 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

```
#define Zero  0x3F
#define Un    0x06
#define Deux  0x5B
#define Trois 0x4F
#define Quatre 0x66
#define Cinq  0x6D
#define Six   0x7D
#define Sept  0x07
#define Huit  0x7F
#define Neuf  0x6F
```

```
byte taseg [10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F };
```

Avec les #define, on peut écrire

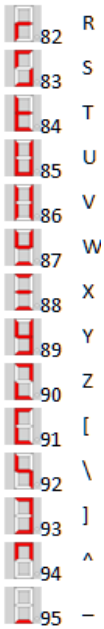
```
byte taseg [10] = {Zero, Un, Deux, Trois, Quatre, Cinq, Six, Sept, Huit, Neuf};
```

C'est plus clair et plus facile si on doit modifier des segments.

Pour allumer les segments, il faut savoir comment ils sont câblés. C'est le travail du programmeur système, qui connaît l'électronique, le schéma et a écrit la fonctions Digit (dd) que l'on va utiliser.
Il a écrit d'abord une fonction Seg() qui écrit un byte sur les segments et la fonction Digit() qui est très simple: surveille le paramètre donné et passe par la table. Si c'est supérieur à 9, on sature.

```
void Digit (byte hh) {  // 0-9
  if (hh>9) hh=9;
  Seg (taseg [hh]);
}
```

La table est dans `EduC.h` avec les fonctions `Seg()` et `Digit()`.
*Pas besoin de savoir comment est fait un couteau, mais il faut savoir le tenir par le manche.*




En fait Digit est un peu différent en librairie, pour accepter les chiffres hexa, jusqu'à F.

Affichons les chiffres dans l'ordre:

```
//Edu52aChif.ino Affiche les chiffres de 0 à 9 en boucle
#include "EduC.h"
void setup(){ SetupEduC();}

byte cnt;
void loop() {
  Digit (cnt++);
  if (cnt>9) cnt=0;  // (cnt>0xF) pour hexa
  DelMs(500);
  while (PousG) ; // Attend si on pèse pour
}
```

## Chiffres hexadécimaux

Pour afficher des chiffres hexadécimaux, il faut définir le codage pour les chiffres de A à F.
Le programme 52a permet de tester un code à la fois. On utilise la fonction `Seg(v8)` qui envoie le mot 8 bits en paramètre directement sur les segments.

```
//Edu53aSeg.ino Affiche un code
#include "EduC.h"
void setup(){ SetupEduC();}

byte cnt;
void loop() {
  Seg (cnt++);
  DelMs(100);
}
```

}On peut aussi jouer à faire toutes les lettres de l'alphabet et afficher son nom en séquence.

jdn 180102/180205