www.didel.com/EduC-Mod1.pdf

EduC – Programmer en C/Arduino - Module 1
Les nombres et les variables

Bits, bytes et int

Les circuits logiques et les bascules mémoires ne connaissent que 2 états, 0 et 1, appelés bits.
Avec 2 bits, on peut coder 2*2=4 possibilités 00, 01, 10, 11.
Avec 8 bits, on forme un **byte**, qui donne 2*2*2*2*2*2*2*2 = 256 possibilités de codage.
On peut donc coder les lettres de l'alphabet, et les chiffres (code ASCII). On peut avec ces 256 possibilités de codage représenter les nombres de 0 à 255. On peut dire que 8 bits, c'est 2 fois 4 bits et qu'avec 4 bits, on a 16 possibilités, avec lesquelles on va coder les chiffres plus efficacement qu'avec le code ASCII. Par exemple 75 se code 0111 0101 (c'est du BCD, décimal codé binaire).
Avec 2 `bytes`, on a un mot de 16 bits appelé **`int`** avec le C.

Ne compliquons pas les choses pour le moment et travaillons en décimal, avec la contrainte que le plus grand nombre dans un `byte` est 255, et dans un `int` ~32000. La fonction `DelMS();` a une durée de type `int`, donc le délai maximum est 32 secondes et quelques.
Les nombres sont fixes dans le programme. En définissant une variable, on peut faire varier le délai.
Une variable est comme un tiroir dans lequel on a rangé un nombre. Le tiroir est fermé, il a un nom. On peut l'ouvrir pour lire le nombre qu'il contient ou changer sa valeur. Il y a des tiroirs pour des `byte` et des tiroirs pour des `int`. A l'intérieur, les mots binaires peuvent représenter ce que l'on veut, c'est le programme qui sait que le tiroir "`lum`" contient un motif lumineux.

En C, les "tiroirs" sont des positions mémoires. Leur nom commence par une minuscule.
On les réserve en écrivant **`byte toto;`** **`int duree;`** et on peut leur donner une valeur initiale
`byte compte=5;`
On peut calculer avec des variables: `toto= 3*4;` `toto= toto+(titi*2);`
Très souvent on doit ajouter 1 à un compteur, par exemple `compte = compte +1;` Le C a deux raccourcis: `compte +=1` (ou une autre valeur) et si c'est seulement pour ajouter 1, `compte++;` que l'on verra souvent, ainsi que `compte--;`

Exemple

On veut clignoter la LedG en diminuant la période. On amodifie le programme Edu01bCli.ino

```
//Edu11aCliDimi.ino clignotement à période qui diminue
#include "EduC.h"
void setup () { SetupEduC(); }

int tempsOn=1000;
void loop() {
   LedGToggle; DelMs(tempsOn);
   tempsOn -= 50 ;  // on diminue le temps
}
```

On voit que les flash diminuent et se rapprochent et que cela s'arrête. Que se passe-t-il? La fonction DelMs() se bloque si le paramètre vaut zéro, ou bien le délai est top long?

Variante

Il faut naturellement éviter de tomber dans un blocage. Avec un `if,` on détecte les valeurs limites. C'est le prochain programme.

```
//Edu12aCliAug.ino la durée augmente et on recommence
#include "EduC.h"
void setup () { SetupEduC(); }

int tempsOn=100;
void loop() {
   LedGToggle; DelMs(tempsOn);
      tempsOn += 50;  // on augmente le temps
   if (tempsOn>1000) {tempsOn=100;}
}
```

Les signes de comparaison sont `<` `>` `<=` `>=` `!=` (différent) et `==` (égal).
Attention, avec un simple = on a une assignation `tempsOn=1000;` - nouvelle valeur
Avec un double `==` on a une comparaison: est-ce que `tempsOn est égal à 100?` – vrai?

Questions : que se passe-t-il si la valeur initiale est 10, 5000?
Si on ne donne pas une valeur initiale, elle vaut 0. que se passe-t-il?

Defi

Avec des `while`, faire en sorte que les impulsions diminuent , puis augmentent (Edu12b).
Indication: tant que c'est trop lent (donc supérieur à la période limite, on accélère, …

Minuterie d'escalier

Le principe est simple: on pèse sur un bouton et l'éclairage vient pour 5 minutes. C'est une simplification du programme Edu11a.ino.
Si on veut 5 minutes, c'est 5*60*1000 ms (en C le signe de la multiplication est *) . Le délai max documenté est 32000 ms, donc 32 secondes, c'est trop court. On va donc répéter un délai de 100ms 5*60*10 fois.
En plus, cela va ajouter une fonctionnalité intéressante. C'est assez désagréable quand la lumière s'éteint de retrouver le poussoir dans l'obscurité. Il faudrait pouvoir ré-armer le délai pour 5 minutes au bout de 3, 4 minutes si on voit que cela va prendre plus longtemps.

Le délai est bloquant, il ne fait rien d'autre qu'attendre, et ignore le poussoir.

Le compilateur sait calculer sans faute, on va lui laisser le soin de calculer 5*60*10 mais il faut que l'on réserve une variable compteur qui a assez de place pour les délais prévus. Le calcul donne 3000, donc il faut une variable de type `int` pour compter, max 32000, donc la durée max pour la minuterie sera de 32000/(60*10) ~= 50 minutes. C'est bien assez. On va l'initialiser à (max50)*60*10.

```
//Edu13aMinuS.ino minuterie d'escalier simple
#include "EduC.h"
void setup () { SetupEduC(); }

#define TempsOn 5*60*1000  // 5minutes de 60 seconde de 1000ms
int cnt;
void loop() {
   while (!PousG) {} // inutile d'attendre si relâché
   BlancOn; cnt = 0;
   DelMs(TempsOn);   // pour tester mettre DelMs(1000);
   BlancOff;
}
```

Si on coupe le délai de 5 minutes en tranches de 100ms, on peut chaque fois regarder si le poussoir est activé et relancer le délai !
C'est un truc pour que le processeur fasse 2 choses, "en même temps" pour l'utilisateur, alors que lui, il ne pourra toujours que faire une chose à la fois.

```
//Edu13bMinuR.ino minuterie d'escalier avec redémarrage
#include "EduC.h"
void setup () { SetupEduC(); }

#define TempsOn 5*60*10  // cycles de 100ms
int cnt;
void loop() {
   while (!PousG) {}   // inutile d'attendre si relâché
   BlancOn; cnt = 0;
   while (cnt++ < TempsOn) {
     DelMs(100);
     if (PousG) {cnt=0;}  // on réinitialise
   }
   BlancOff;
}
```

La condition `(cnt++<TempsOn)` fait 2 choses dans un ordre précis. C'est compact, mais il faut avoir compris comment le compilateur va traduire en instructions. Le décision vrai/faux se fait d'abord: est-que `cnt` est inférieur à `TempsOn ?` Ensuite on augmente `cnt` de 1 (ou dit souvent incrémenter).

Pourquoi ne tient-on pas compte des rebonds de contact? C'est inutile, dès que le premier rebond est activé, on ne le surveille plus pendant 100ms, et on ne s'intéresse pas ici de savoir combien de temps il reste pressé.

Defi

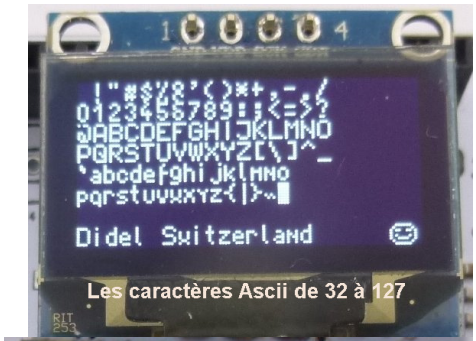
La gérance demande une amélioration. Il y a des utilisateurs qui coincent une allumette dans le poussoir et oublient de l'enlever. Il faudrait que si on presse trop longtemps (1-2) secondes, que cela coupe la lumière immédiatement. Celui qui coincerait le poussoir se trouverait ainsi bien embêté dans l'obscurité!
Une autre solution est de couper de toute façon au bout de 30 minutes (c'est facile à faire avec un 2e compteur), voir à ce moment que le bouton est pressé et attendre qu'il soit décoincé (dans l'obscurité) pour reprendre le programme au début.
On verra que c'est plus facile à programmer avec une machine d'état (module 6).

Affichons des nombres

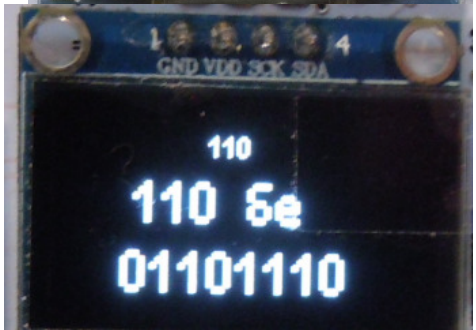
Affichons des nombres et des variables sur l'affichage Oled, qui a 8 lignes et 128 colonnes.
Il faut inclure Oled.h et initialiser avec SetupOled(); On parlera plus de cet écran plus tard.

On a à disposition les ordres suivants:

`Clear();` **efface l'écran**
`LiCol (no de ligne,no de colonne);`
`Texte ("mon texte");`
`Dec8 (val);` val est une variable ou un nombre 8 bits, converti et affiché en décimal.
`Bin8 (val);` val est une variable ou un nombre 8 bits, affichage en binaire.
`Hex8 (val);` val est une variable ou un nombre 8 bits, affichage en hexadécimal



L'hexadécimal est la sténo du binaire. Chaque groupe de 4 bits (nibble) est affiché par les chiffres de 0 à 9, complété par les lettres A . . . F.




```
//Edu14aAfNb.ino affiche des nombres
#include "EduC.h"
#include "Oled.h"
void setup () { SetupEduC(); SetupOled(); }

byte val=100;  // val doit être <255
```

```
void loop() {
   LiCol(1,50); Dec8 (val);
   LiCol(4,24); BigDec8 (val); BigHex8 (val);
   LiCol(7,20); BigBin8 (val);
   if (PousG){ val++;} DelMs(100);
   if (PousD){ val--;} DelMs(100);
}
```

La boucle réaffiche les valeurs et teste les poussoirs. Si un poussoir est pressé, on compte.
Pour démarrer et stopper le comptage avec un seul poussoir, vous vous souvenez comment faire (on avait allumé-éteint une led).

Supprimez les `DelMs();` On voit que le temps pour afficher des gros caractères est important. Vous pouvez évaluer ce temps?

Le processeur sait calculer?

Affichez les calculs suivants
50*3 50/3 50%3 (c'est le reste de la division)
10*100 10/100

Dans le programme Edu14a, renommé en 14b, écrivez

```
LiCol(4,24); BigDec8 (50*3); BigHex8 (50*3);
```

etc..
En fait, ce n'est pas le processeur qui a calculé, mais le compilateur.

Pour faire calculer le processeur, il faut donner des valeurs dans des positions mémoire (comme variables) et utiliser les instructions arithmétiques.

```
//Edu15aCalculs.ino Calculs
#include "EduC.h"
#include "Oled.h"
void setup(){SetupEduC(); SetupOled();}

byte nb1,nb2,res;
int val;
void loop() {
  nb1= 50; nb2= 3; val= 3333;
  res= nb1*nb2;
  LiCol(1,0); BigDec8 (res); BigHex8 (res);
  LiCol(3,0); BigDec8 (nb1/nb2); BigDec8 (nb1%nb2);
  LiCol(5,0); BigDec9999 (val*nb2);
}
```

On voit qu'il faut faire attention au type. Le compilateur ne sait pas avec quelles valeurs le processeur va calculer, et il ne peut pas anticiper les erreurs d'exécution.

Afficher des caractères, des textes et des sprites

Les caractères Ascii sont codés à patir de la valeur 32=0x20.
Les textes sont mis entre "" ". Les minuscules accentuées ne sont pas affichées correctement.
On verra comment fabriquer des sprites dans le module 5.

```
//Edu16a.ino affiche des caractères et textes
#include "EduC.h"
#include "Oled.h"
void setup () {
  SetupEduC(); SetupOled();
  LiCol(0,0); Sprite(smile);
  LiCol(0,50); Text ("Moi");
  LiCol(0,116); Sprite(sad);
}
```

```
byte val=100;  // val doit être <255
void loop() {
  LiCol (2,0);
  for  (byte i=0; i<96; i++) {
    Car(32+i);
    DelMs(100);
  }
  while(1);  // stop
}
```

On voit que la "mise en page" est en setup. Cela évite de reconstruire ce qui est fixe à chaque passage dans la boucle de programme.

jdn 171208/180205