EduC – Programmer en C/Arduino - Module 2
La boucle for, des clignotements et des sons

La boucle `for`

On veut allumer 3 fois une Led, et attendre avant de recommencer. On peut le faire avec un compteur et un while. La construction `for` est plus élégante et on aura beaucoup d'occasions de l'utiliser.
Pour clignoter 3 fois, on écrit en respectant les parenthèses et les ;

```
for (byte i=0; i<3; i++) { faire ceci ; }
```

On voit une variable de comptage `i` déclarée et initialisée à zéro, et on dit de répéter l'action tant que `i<3`, en augmentant `i` a chaque boucle. L'action peut aussi utiliser la variable `i`, dont le nom est quelconque. C'est aussi un nom "local"; vous pouvez réutiliser ce nom ailleurs.
Les 3 parties de la parenthèse donnent beaucoup de possibilités. On se bornera ici à l'utilisation simple comme répéteur d'actions.

```
//Edu21aCli3x.ino clignoter 3 fois
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  for (byte i=0; i<3*2; i++) {   // 3*2 car …Toggle fait la moitié du travail!
     LedGToggle; DelMs(100);
  }
  DelMs(1000);
}
```

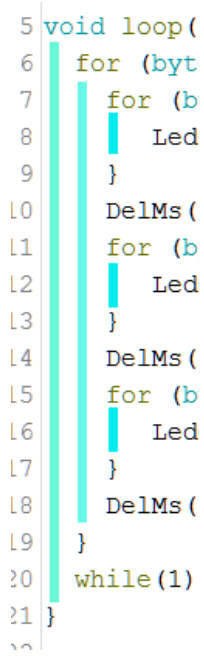
Il est important maintenant de bien respecter la position des accolades qui paranthèsent l'action. En dessous du `for` on retrouve l'accolade finale `}` , bien alignée. Les instructions exécutées sont décalées.

Boucle `for` dans une boucle `for`

On peut imbriquer des boucles for, par exemple pour lancer 5 fois un SOS.

```
//Edu22aSos.ino SOS
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  for (byte j=0; j<6; j++) {
    for (byte i=0; i<3; i++) {  // S
       LedGOn; DelMs(200); LedGOff; DelMs(200);
    }
    DelMs(500);
    for (byte i=0; i<3; i++) {  // O
       LedGOn; DelMs(600); LedGOff; DelMs(600);
    }
    DelMs(500);
    for (byte i=0; i<3; i++) {  // S
       LedGOn; DelMs(200); LedGOff; DelMs(200);
    }
    DelMs(3000);
  }
  while(1);  //stop - il faut faire un reset
}
```



Les décalages mettent en évidence la structure du programme, avec ses trois modules qui se répètent. L'instruction d'arrêt précède l'accolade finale.

Pour cette instruction d'arrêt, c'est plus pratique d'écrire

```
while(!PousG)) ;  // attend une action sur PousG
```

En pressant sur le poussoir, on continue, donc on se retrouve au début de la boucle et on recommence. Plus rapide que le reset, et très utile pour dépanner ; on peut mettre cette ligne provisoirement dans le programme pour vérifier qu'il a bien fait ce qu'il doit faire jusqu'à ce point.

Essayez d'oublier une accolade, une parenthèse, un ; pour voir quel message d'erreur est généré.

Afficher les caractères ASCII

```
//Edu23aGenCar.ino
#include "EduC.h"
#include "Oled.h"
void setup(){ SetupEduC(); SetupOled();}

void loop(){
  LiCol (0,0); for (byte i= 32; i<48; i++) Car(i);  // Espace, signes
  LiCol (1,0); for (char i= 48; i<64; i++) Car(i);  // 0 1 2 3 ...
  LiCol (2,0); for (char i= 64; i<80; i++) Car(i);  // @ A B C ...
  LiCol (3,0); for (char i= 80; i<96; i++) Car(i);  // P Q R S ...
  LiCol (4,0); for (char i= 96; i<112; i++) Car(i); // ` a b c ...
  LiCol (5,0); for (byte i= 112; i<128; i++) Car(i);// p q r s ...
  LiCol (6,0); Car (65); Car (' '); Car ('A');
  Car (10); // codes < 32 ignorés, code 32 --> espace
  while (1);
}
```

Flexibilité de la boucle for

La boucle for a 3 parties séparées par un point-virgule:

`for (`initialisation; vrai? ; modification`) { faire ; }`

L'initialisation définit en général une variable locale et sa valeur initiale.
La condition au centre décide si on refait la boucle. Si l'expression est vraie, on refait.
Il faut modifier quelque chose à chaque boucle pour que l'on finisse.
La variable locale (i, k dans nos exemples) est le plus souvent utilisée dans le `faire;` , c'est son intérêt.

Exemples encore simples:

```
for (byte k=20; k>3; k--) { faire ; }

for (byte i=170; i<300; i+=13) { faire ; }
```

Faire des sons

La membrane du haut parleur s'excite comme quand on allume-éteint une Led.
`HpOn;` colle la membrane. `HpOff;` relâche. `HpToggle;` change d'état; Avec une période de 1ms et `HpToggle;`, on génère du 500 Hz.
Pour programmer une mélodie écoutable, il faut avoir des fréquences de notes très précises. Jouons d'abord à faire des bips et des sirènes.

```
//Edu24aBip.ino bip-bip
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  for (byte i=0; i<250 ; i++) {
     HpToggle; DelMs(1);
  }
  DelMs(1000);
}
```

Essayez `i<300.` Pourquoi est-ce que cela fait un son continu?
`i` a été défini comme un byte, maximum 255. Le compilateur aurait pu voir l'erreur, mais non! Il faut déclarer `int i;` et on peut continuer jusqu'à 32 secondes.

Plus aigu

On peut utiliser la fonction Arduino `delayMicroseconds (tt);` pour avoir des sons de fréquence plus élevées et tester votre limite d'audition. Vous remarquez que le programme est nettement plus gros si on utilise des fonctions Arduino.

Sirène

Pour faire une sirène, il faut programmer notre propre boucle d'attente que l'on va raccourcir ou allonger. Il y a une période pour les notes, et une durée de répétition de la même note, pour que la sirène évolue lentement.
La ligne `for (int h=0; h < dur; h++) {}` dure environ `dur` microsecondes. Si on l'appelle 1000 fois on a 1ms comme `DelMs(1);` mais on peut maintenant l'appeler 999 fois pour avoir un délai, donc une note un peu différente. On a fabriqué un équivalent de `delayMicroseconds (tt);`
Un son est entre 2kHz et 200 Hz donc une demi-période de 100 us et 1000 us.

```
//Edu25aSirene.ino La fréquence augmente
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  for (int per=1000; per>100 ; per--) {
    for (byte i=0; i<20; i++) { //20 demi-périodes avant de changer
      HpToggle;
      for (int h=0; h < per; h++) {nop;}   // attente selon période
    }
  }  while(!PousG);   // on  recommence quand on veut
}
```

Nul, notre haut parleur !

Le haut-parleur –buzzer n'est pas fait pour jouer de la musique. On le voit avec la sirène, il y a des fréquences qui passent bien, et d'autres qui sont étouffées par des vibrations mécaniques parasites. A vous de choisir les bonne fréquences.

Notes avec la librairie Arduino

Arduino connaît trois fonctions pour jouer des notes, que l'on peut utiliser sans autre en sachant que le haut-parleur de la carte EduC est sur la pin 14

```
tone(pin_number, frequency, duration);   joue la fréquence
                                          pendant la durée en ms
tone(pin_number, frequency);     joue non stop
noTone(pin);                      stoppe
```

La fonction `tone(14,freq);` lance un son indépendamment de ce que fait le processeur, car l'instruction agit sur des circuits internes au processeur. On peut clignoter pendant qu'il y a un son. La minuterie d'escalier pourrait émettre un son toujours plus aigu quand le temps avance!

Exemple:

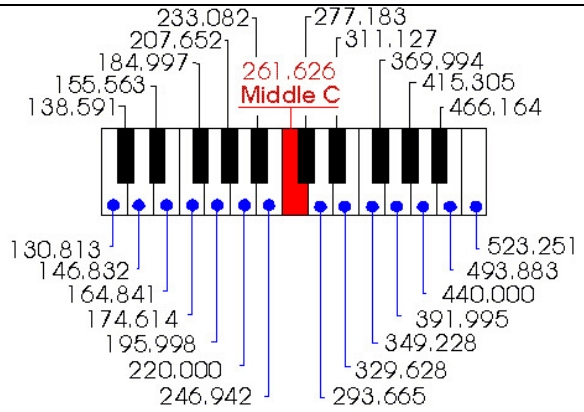
```
//Edu26aSons.ino Test fonction tone
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  tone (14,500,1000); // 500 Hz ?pendant 1 secondes
  DelMs(200);
  tone (14,1000); // on lance 1kHz
  DelMs(500);     // on attend 0.5 secondes
  noTone(14);     // on coupe - enlever cette instruction
  while(!PousG);  // on recommence si on veut
}
```

Pour jouer des mélodies

Vous trouvez des exemples sur internet, mais ce sont souvent des programmes Arduino naifs. Ils peuvent être adaptés en sachant que sur Edu-C, la pin haut parleur est la pin 14,

La musique, c'est assez compliqué, et on ne peut rien faire d'écoutable avec un microcontrôleur (il faut des circuits spécialisés) et avec un buzzer. La fonction tone (,) est peu précise, et n'accepte que des entiers. Mais c'est amusant quand même!



Fréquences des notes d'un piano

Ces frèquences sont mauvaises pour le buzzer, il faut les multilpier par 2 ou 3 ou autre. A essayer.

Ensuite, il faut définir chaque note (notes doublées par rapport à la figure, octave supérieure)

```
    #define Do  tone (14,524)
    #define Re  tone (14,554)
...
    #define Do2  tone (14,1048)
...
    #define Noire DelMs (200)
    #define Blanche DelMs (400)
...
```

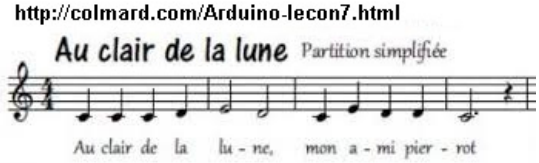
On voit qu'il n'y a pas de ; a la fin d'une définition. Il vient à la fin du Do; écrit dans le programme. Rappelons que le #define permet de remplacer un mot par une suite de lettres jusqu'à la fin de la ligne, en ignorant le commentaire éventuel.

Attention ! `do` (en minuscules) est un nom réservé par Arduino, avec `if, while,` etc. On ne les a pas tous vus.

Exemple

`//Edu27aLune.ino Au clair de la lune`
N'est pas un exemple bien écrit

http://colmard.com/Arduino-lecon7.html



Remarque

On entend que certaines notes ne passent pas à cause de la mauvaise courbe de réponse du buzzer. Puisque notre oreille est relative, on peut décaler tous les sons et trouver une zone un peu plus régulière pour notre buzzer et toujours reconnaître l'air. Une meilleure librairie sera disponible une fois sous www.didel.com/educ/EduC-Melodies.