www.didel.com/EduMod4.pdf

EduC – Programmer en C/Arduino - Module 4
Les fonctions

Pourquoi des fonctions?

Pour être efficace, il faut classer, regrouper et nommer. En écrivant `LedGOn;` on a caché des instructions liées au câblage de la LED. C'est intéressant à étudier, mais notre but est de programmer des séquences lumineuses. Pour envoyer un message morse, il a fallu écrire 15 lignes. Formons une fonction avec ces 15 lignes et donnons le nom `SOS();` Dans un programme qui détecte que le bateau coule on écrira simplement `if(Coule()) {SOS();}. Coule()` sera une fonction compliquée qui tient compte de l'information d'une quantité de capteurs.
On voit qu'une fonction peut être l'équivalent d'une variable vrai/faux (Coule ou coule pas?) ou d'une action (envoyer le SOS). Il y a toujours une parenthèse pour 0,1,2,.. paramètres.

Pour faire juste, il faut respecter des règles du C, que l'on comprendra mieux quand on aura pratiqué. Pour la fonction SOS(), il faut écrire:

```
void SOS() {
  les instruction pour envoyer le SOS, voir Edu22a.ino
}
```

et la boucle du programme Edu22a devient

```
void loop() {
  SOS();
  while (!PousG);  //wait
```

Ce qui évidemment est plus lisible.

Le programmes `Edu41aSosFonc.ino` dans le zip montre le programme complet. Les fonctions doivent être déclarées avant le programme, avant qu'on les appelle; on ne peut pas travailler avec une variable ou une fonction qui n'a pas été déclarée, c'est logique.
C'est le problème du compilateur C. Il ne peut travailler qu'avec ce qu'il a déjà vu, et il met parfois des messages qui n'aident vraiment pas (voir le document EduC-MiseAuPoint).

Fonctions avec paramètres d'entrée

Les fonctions se placent avant d'être appelées et elles ont en général des paramètres.
La fonction qui clignote 3 fois est facile à écrire. Elle est plus générale si elle permet de clignoter n fois. n est un paramètre en entrée. On lui donne un nom dans la description de la fonction qui est un nom local, interne au libellé de la fonction.

```
void CliLedGNfois (byte nn) {
  for (byte i=0; i < 2*nn ; i++) {
     LedGToggle; DelMs(200);
  }
}
```

Et on pourra écrire dans le programme `CliLedGNfois(3);`

Dans la fonction, on a besoin de variables. Ces variables sont locales, pour décrire le fonctionnement. Le compilateur choisit les positions mémoire de ces variables à la compilation. On leur donne donc des noms abrégés. Ici, `nn` est le nombre de clignotements. Comme il faut deux "toggle" pour un clignotement, on a `2*nn` dans la boucle `for`.

La durée demi-période peut aussi être un paramètre en entrée.

```
void CliLedGNfois (byte nn, int dd) {
  for (byte i=0; i<2*nn; i++) {
     LedGToggle; DelMs(dd);
  }
}
```

On voit que la variable période est un `int` 16 bits. Avec un `byte`, il y a problème si le paramètre donné est supérieur à 255.
On voit aussi que le type de chaque variable d'entrée doit être déclaré. Si on a 2 variables d'entrée de type byte, il faut écrire `(byte aa, byte bb)` car `bb` pourrait être un `int`. Quand on définit des variables générales, on écrit `byte aa,bb;` avec un `;` final. Dans une fonction, on annonce que l'on va travailler avec des variables, on ne leur réserve pas de place; cela se fera quand la fonction sera appelées.

```
//Edu42aCliNx.ino
#include "EduC.h"
void setup () { SetupEduC(); }

void CliLedGNfois (byte nn, int dd) {
  for (byte i=0; i<2*nn; i++) {
     VertToggle; DelMs(dd);
  }
}
void loop() {
  CliLedGNfois (4,400);
  while (!PousG) ;
}
```

Essayez `CliLedGNfois(4,400); CliLedGNfois(0,400); CliLedGNfois( ,400); CliLedGNfois(4,0);`
Si maintenant les constantes sont des variables, qui peuvent valoir zéro, vous voyez pourquoi un programme "juste" peut être mort quand une variable a passé par une mauvaise valeur.
Vous avez remarqué que l'éditeur Arduino essaie de vous faciliter le travail. Quand vous tapez "accolade – retour" il ajoute à la ligne suivante l'accolade terminale et décale de 2 caractères là ou vous devez continuer à taper.

Plus besoin d'apprendre le livret

Le programme suivant affiche le livret de nn en utilisant le fonction Livret (nn) qui donne les 8 premières valeurs.

```
//Edu43aLivret.ino Livret de n
#include "EduC.h"
#include "Oled.h"
void setup () { SetupEduC(); SetupOled(); }

void Livret(byte nn) {  // affiche les 8 multiple de nn
  byte res;  // variable locale
  for (byte i=0; i<8; i++) {
    res = nn*i;
    LiCol (i,0); Dec8(res);
    DelMs(100);
  }
}

byte nombre =2 ; // premier livret
void loop() {
  Clear();  // efface l'écran
  Livret(nombre++);
  DelMs (1000);
  while(nombre > 12) ;  // on bloque à 12
}
```

Exercice: Fonction Max

On a 2 nombres, écrire la fonction maximum qui permet d'écrire c = Max (a,b);

Defi

Ecrire la fonction CliGD (periodeG,periodeD). Les 2 leds clignotent ensemble à des périodes différentes. Indication: toutes les 10ms, avec 2 compteurs, on décide pour chaque led si on l'allume ou l'éteint.

Fonctions avec un paramètre en sortie

Les fonctions avec un void devant exécutent les instruction avec les paramètres donnés.
Un fonction peut aussi "rendre " un paramètre, un seul. Par exemple une fonction qui vous donne le triple permet d'écrire `Dec8 (Triple(3));` ce qui affichera `9` .
Il faut réserver de la place, et dire avec l'instruction `return (xxx);`
ce que la fonction "rend".

```
byte Triple (byte nn) {
  return (3*nn)
}
```

Autre exemple

```
int Produit (byte aa, byte bb) {
int res;
  res = (aa * bb) ;
  return (res);
}
```

Exemple: Potentiomètre de 0 à 99

Un peu plus compliqué, on lit le potentiomètre, et on a vu que la fonction `GetPotG()` rend une valeur de 0 à 255. On veut une fonction `GetPot100G()` qui rende une valeur entre 0 et 99.
Le premier problème est de voir comment calculer. Il faut diviser les valeurs par 2.56 et le C ne travaille qu'en nombres entier ! Alors, soyons astucieux. Diviser pas 2.56, c'est diviser pas 256/100, c'est aussi multiplier par 100/256. Multiplier par 100, le processeur sait faire, mais il faudra réserver assez de place. Diviser par 256 est très facile, il a un truc (il faut avoir expliqué les décalages avant).

```
//Edu45aPot99.ino lecture pot 0..99
#include "EduC.h"
#include "Oled.h"
void setup(){  SetupEduC(); SetupOled(); }

// Fonction GetPot100G()  rend valeur 0..99
byte GetPot100G() {  // résultat 0..99
  return ((GetPotG()*100)/256);
}
void loop() {
   LiCol (3,0); BigDec8(GetPot100G());
}
```

Le pot à la place du clavier

Le potentiomètre permet de fabriquer un nombre entre des valeurs quelconques. Cela peut être utile pour des jeux. On sélectionne un nombre avec le pot, et on quittance avec un poussoir. On peut aussi "voir" que le pot n'a pas bougé pendant 0.5 secondes, c'est-à-dire écrire une fonction qui mesure la vitesse de déplacement du pot.

Les 2 poussoirs On, ou Off

Pas évident de devoir re-réfléchir chaque fois! Définissons 2 fonctions dont le nom est explicite;.

```
byte BothOn () {
  return (PousG&&PousD);
```

```
}
byte BothOff () {
  return (!PousG&&!PousD);
}
byte AnyOn () {
  return ( PousG||PousD);
}
```

Pour chaque fonction que l'on crée, il faut faire un programme de teste qui teste cette fonctions, et bien vérifier que tous les cas possibles sont corrects.

Application

On veut que l'allumage on/off soit commandé par l'un ou l'autre des deux poussoirs. C'est le problème d'une grande chambre avec un poussoir à deux endroits. A noter que si ce sont des interrupteurs à bascule, le problème est différent.

```
//Edu46aDeuxPous.ino 2 poussoirs pour une lampe
#include "EduC.h"
#include "Oled.h"
void setup(){  SetupEduC(); SetupOled(); }

byte BothOff () {
  return (!PousG&&!PousD);
}
byte AnyOn () {
  return ( PousG||PousD);
}

void loop() {
     while (BothOff()) {DelMs (20);}  // attend que qqn presse
     LedGToggle;
     while (AnyOn()) {DelMs (20);}  // attend relâchement
}
```

Attention

Une fonction qui n'a pas de paramètre en entrée doit avoir la () avant le ; Si cette () est oubliée, il n'y a pas de message d'erreur et le programme déraille!
Essayez dans le programme précédent d'écrire

```
NumeroPoussoir = NumPous;
```

Il y a bien qqchose d'affiché sur le 7-segments et la Led RGB est rouge. Mais cela dépend de ce qui traine dans la mémoire, des exemples précédents qui ont été testés! Un programme qui "plante" exécute des instructions, le processeur tourne toujours.

Rappel

LedGOn; , .. PousD, .. sont des définitions liées au câblage
SOS(); LedG(val); NumPous(); sont des fonctions, avec ou sans paramètre

Analogie

Il faut se souvenir que les variables sont des tiroirs. Une fonction qui a un paramètre en sortie, c'est comme un tiroir qui contient une machine. On donne à la machine des paramètres, et on ouvre le tiroir pour avoir le résultat.
Si le programme n'appelle pas la machine, il n'y a pas de code généré.

Fonctions bloquantes et non

On a vu qu'il y a des instructions bloquantes (while, for) et traversantes (if).
Il y a aussi des fonctions bloquantes (NumPous(); attend sur un poussoir) et traversantes (max(a,b);).