www.didel.com/EduC-ModResume.pdf

EduC – Programmer en C/Arduino - Résumé

Notions importantes des modules 0 à 6

Module 0 Le if et le while www.didel.com/educ/EduC-Mod0.pdf

```
#include "EduC"

void setup() {SetupEduc();}

void  loop () {
  //instruction du programme
  LedGToggle; DelMs(nb_ms);
}
```

Le programme appelle une librairie avec un `#include "EduC"`.
Le setup configure le microcontrôleur selon ce qui a été connecté.
Le processeur boucle ensuite sans fin dans des instructions élémentaires à 16 Mhz, 1 million d'instructions et fonctions simples en C par seconde.
Minuscules pour les instructions et variables.
Majuscule en première lettre d'une fonction et les constantes/définitions
{ } accolades entre les groupes d'instructions. ( ) autour des paramètres
; point virgule à la fin de chaque instruction et fonction.

```
if ( vraifaux )  { faire ceci; }
if ( truc==3 )  { faire ceci; }
 ==  >  <  >=  <=  != (différend)
```

`vraifaux` est une variable ou un état qui est TRUE=1 ou FALSE =0. Le résultat d'une comparaison est aussi vrai ou faux. **Attention == compare = assigne**

if (PousG) { faire si le poussoir est pressé; }
if (**!** PousG) { faire si le poussoir est non pressé; }
! inversion logique

if (PousG || PousD) L'un ou l'autre pressés
if (!PousG && !PousD) Les deux relâchés
&& ET logique || OU logique (| clavier ctrl7)

```
if (cond) {faire si condition vraie}
else {faire si condition fausse}
```
ne rien faire **{ }** ou **;** , c'est faire!

```
if (cond1) {faire si cond1}
else if (cond2) {faire si cond2}
else {faire dans les autres cas}
```

```
while (vraifaux ) { faire ceci; }
while (1) { faire toujours }
while (1) {} ou while (1);
```

On fait tant que la condition est vraie
La condition est toujours vraie. On fait en boucle.
Le processeur se tourne les pouces!

Le `if` est traversant. On teste, on fait si vrai et on continue
Le `while` est bloquant. On teste et re-teste tant que la condition est vraie.

Faites des tas de programmes! Un programme marche – vous le sauvez
Vous voulez le modifier – modifiez le titre et sauvez avec le nouveau nom
Après chaque modif – sauvez (ou mode sauvegarder dans Préférences)

Module 1 Les nombres et les variables www.didel.com/educ/EduC-Mod1.pdf

La mémoire "flash" contient le programme.
La mémoire vive du processeur est réservée pour les variables. Ce sont des mots de 8 bits que l'on peut associer pour représenter des nombres, des textes, des informations codées.

Le C définit en premier niveau 2 types de variables qu'il faut déclarer avant d'utiliser

```
byte v8;  //réserve 8 bits pour l'objet v8
int v16;  // réserve 16 bits
```

La variable v8 peut être un nombre
de 0 à 0b11111111 en binaire,
de 0 à 0xFF en hexadécimal,
de 0 à 255 en décimal

La variable v8 peut être une lettre de l'alphabet.
C connaît leur code, il suffit d'écrire 'a' pour avoir le code ASCII du **a**, qui est **0x61 = 97**
Un texte se code avec une chaîne **"abcd"**

On effectue des opérations arithmétiques sur des variables de même type:
x= a+b; x= a-b; x= a*b; x=a/b;

On peut aussi comparer des nombres
`if (x==0) {x=33;}` **Rappel: == compare, = assigne**
> < >= <= != ==

Raccourcis **a = a+3;** --> **a+=3;** aussi **a/=2;**
a = a+1; --> **a++;** aussi **a--;**

Attention: **if (a++>3)** est vrai si **a>3.**
a est augmenté après **.**

Une expression est calculable par le compilateur.
a+(3*c); n'est pas la même chose que (a+3)*c;

Si a est un byte, si `a=255;` après `a++;` on a `a=0`
(comme un compteur de voiture, après 99999 on a 00000).

Module 2 La boucle for, des clignotements et des sons www.didel.com/educ/EduC-Mod2.pdf

La boucle for est pratique pour répéter une action **`for (byte i=0; i<3; i++)`** { faire ceci ; } `i` est une variable locale à la boucle, inconnue en dehors Répéte tant que i<3 (3 fois si on part de 0 et ajoute 1)	On peut faire la même chose avec un while : **`byte i=0; while (i++<3)`** { faire ceci ; } Dans ce cas, i est une variable globale.

La boucle for est très générale et très flexible for (initialisation ; vrai? ; modif) { faire ; }	**`for (byte k=20; i>3; i--)`** { faire ; } **`for (byte i=170;i<300;i+=13)`** { faire ; }

L'instruction **`do { faire ceci ;} while();`** fait une fois et continue à faire tant que la condition est vraie.	Rappel: while (condition) {faire} ne fait pas si la condition n'est pas vraie. C'est alors traversant.

L'instruction **`break;`** s'utilise rarement, mais elle est parfois indispensable pour détecter une condition qui doit casser une boucle. ! Seulement dans for, while, do et switch !	L'instruction **`goto etiquette;`** ne doit s'utiliser que comme sortie en catastrophe. `if(test)goto erreur;` en fin de programme on a `erreur: message; while(1);`

Module 3 Hasard, analogique en entrée et en sortie www.didel.com/educ/EduC-Mod3.pdf

`byte v8;`
`v8 = random (Min,Max) ;` --> v8 entre Min et Max-1
`int v16;`
`v16 = random (Min,Max) ;` --> v16 entre Min et Max-1
	`byte v8; byte min,max;`
	`v8 = random (min,max) ;`
	
	`randomSeed(GetPotG());`
	(dans le setup)

Fonctions dans EduC.h **`PousD PousG`** **`LedGOn; LedGOff; LedGToggle;`** Idem **`LedD.. Hp..`** idem **`Rouge.. Vert.. Bleu..`** **`Blanc`** sans Toggle	*Géré par interruption* LedG(v5); LedD(v5); Rouge(v5); Vert(v5); Bleu(v5); v5 est déclarée byte , valeur 0..31 saturé à 31 v8= GetPotG(); v8=GetPotD;

Fonctions dans Oled.h **`LiCol (ligne,colonne);`** ligne 0-7 col 0-127 **`Format Big:`** `ligne 1-7` **`byte toto[]={liste de bytes};`** **`MySprite (toto);`**	**Car(cc); Car('a'); Big(); Text("txt"); Bin8(); Hex8(v8); Hex16(v16); Dec8(v8); Dec16(v16); BigBin8(v8); BigHex8(v8); BigHex16(v16); BigDec8(v8); BigDec8(v16); BigDec9999(v12); Sprite(smile); Sprite(sad);**

Module 4 Les fonctions www.didel.com/educ/EduC-Mod4.pdf

Paramètres éventuels en entrée seulement **`void`** `Nom` **`(byte p1, int p2) {`** les instructions utilisent p1 p2 **`}`**	! Définition de variables: `byte aa,bb;` ! Annonce de variables utilisées dans une fonction: `(byte p1, byte p2, int p3) { . . . }`

Un paramètre en sortie, para en entrée comme avant **`type`** `Nom` **`(para) {`**`... return (xxx); }` Il faut annoncer le type de la variable rendue, et dire avec l'instruction **`return (xxx);`** ce que la fonction "rend".	Exemple: **`byte triple (byte nn) {`** **`return (3*nn);`** **`}`** Appel **`var=triple(7);`**

Dans les situation plus complexes, on initialise une variable locale qui sera rendue (du type annoncé), et on peut définir autant de variable locale que nécessaire et utiliser les paramètres d'entrée.
Mais une seule variable peut être rendue.

Un fonction peut travailler sur une variable globale qui doit être définie avant le loop() .

Module 5 Tableaux et afficheur 7-segments www.didel.com/educ/EduC-Mod5.pdf

`table [n];` Réserve n positions table[0] table[1] etc **`table []= {45,2,48};`** Assigne et remplit des positions **`sizeoff (table);`** donne la taille de la table (3 ci-dessus)	

Nouvelles fonctions dans EduC.h **`Seg(v8);`** Allume les segments **`Digit(v4);`** Affiche le chiffre hexa	**`Leds(v8);`** Allume les leds vertes **Leds(0xff);HpOff;** Eteint les leds vertes et l'affichage 7-seg

Module 6 Le switch case www.didel.com/educ/EduMod6.pdf

La fonction **switch** est associées à une variable qui peut avoir différentes valeurs. Les valeurs qui nous intéressent sont sous **case**. Si une valeur n'est pas dans la liste des **case**, un bloc **default** spécifie ce qui doit se passer.
Dans les cas usuels, chaque bloc est terminé par un **break;** qui saute par-dessus tous les `case` suivants.
L'exemple ci-contre montre que le `switch` envoie dans des programmes de démo séparés qui se bloquent sur eux-même.
Le `switch case` est efficace pour programmer une machine d'état. On traverse le `switch` et exécute le `case` correspondant à l'état. Dans cet état on teste des conditions extérieures et on modifie l'état qui sera exécuté à la prochaine boucle.
`enum {prep, avance, obstaDevant, ..};` assigne des nombres à chaque nom pour rendre le programme lisible.

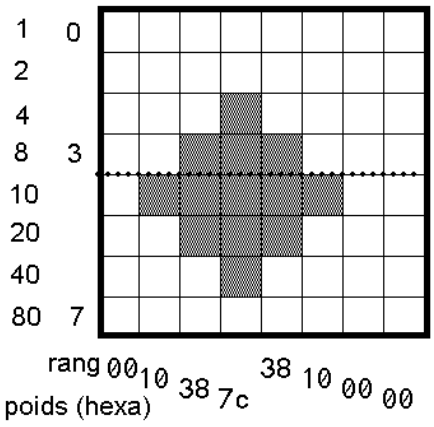
```
//Demo0.ino
#include "EduC.h"
void setup () { SetupEduC(); }

byte nPous;
void loop() {
  nPous = GetPous () ;
  switch (nPous) {
    case 0:
      while(1){
        LedGOn; // demo 0
        while(1);
      } // end while loop case 0
    case 1:
      while(1){
        VertOn; // demo 1
        while(1);
      } // end while loop case 1
    default:
      while(1){
        Digit(0xF); // pas de demo
        while(1);
      } // end default
  }  // end switch
}  // end main loop
```

Ecran graphique de l'affichage Oled	L'origine est en haut à gauche 128 pixels en x 64 pixels en y

Nouvelles fonctions dans Oled.h **Dot(x,y); DDot(x,y);** **Vline(x); Hline(y);** `byte x,y;` est déjà déclaré	**Ball(x,y);** si x=y=0 la balle touche le coin sup gauche **Raq (x,y,h);** x,y centre h hauteur Autres fonctions en développement pour le déplacement et les obstacles.

Les fonctions graphiques seront complétées pour faciliter l'écriture de jeux sur l'écran. Ceci devra se faire en interaction avec les programmeurs intéressés.

Un lutin (sprite) doit être défini dans une table. Cette table est en librairie pour les sprites Smile,et Sad que l'on a rencontré, appelés par Sprite(Smile);
Si le sprite est dans la mémoire programme de l'utilisateur, comme la croix ci-contre, il faut déclarer
`byte croix[]= {0x00,0x10,0x38,0x7c,0x38,0x10, 0x00,0x00};`
L'instruction pour copier à partir du pointeur est
`MySprite(croix);`
La longueur est évidemment inférieure à 127.



Demo - 11 petits programmes dans un, voir www.didel.com/Demo.pdf

Idées de jeux

Deviner le nombre: Le programme tire un nombre au hasard, entre 1 et 99 par exemple. On dit ce que l'on a deviné avec le pot. Si c'est en dessous, led1 en dessus led2, juste bingo.
Simon: Le programme joue une séquence sur 2 leds, Il faut la répéter sur les poussoirs.
Suivi: Le programme dessine une courbe ou un zig-zag de points sur l'écran. Il faut suivre avec un potentiomètre, à la vitesse imposée par le programme (toutes les 0,5s, on fige le point et on avance.. On mesure l'écart pour donner des points.
Jeu de la vie: Web et programme exemple dans la brochure Dauphin (il difficile).
Ping-pong, casse-brique et autre: Revoir les primitives écran.

jdn 180205