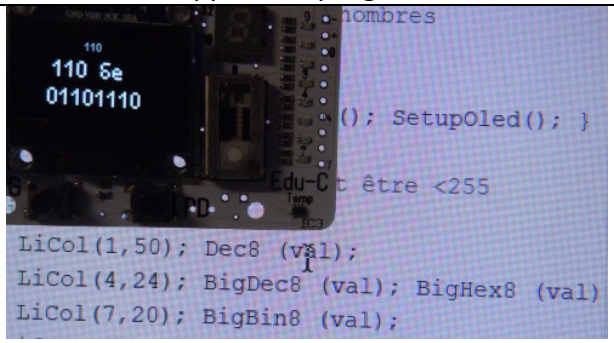
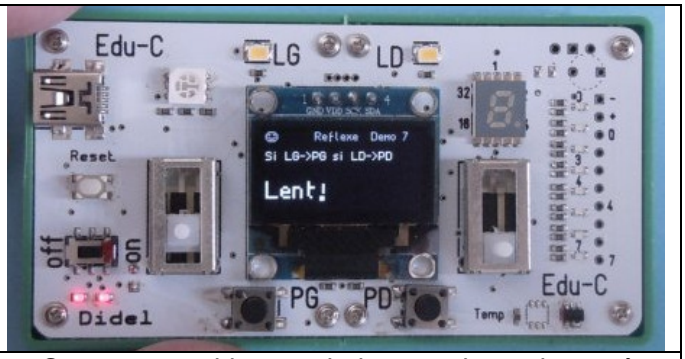
www.didel.com/educ/EduC-Mod0.pdf

EduC – Programmer en C/Arduino - Module 0
Le if et le while

On apprend à programmer	
	
	On peut aussi jouer – le jeu que l'on a inventé

Introduction

La carte EduC est en fait une carte Arduino déjà équipée avec des leds, des poussoirs, des affichages. Une "librairie" permet de nommer les actions avec des noms prédéfinis, par exemple LedGOn pour "allumer la led gauche". Le détail du travail du processeur pour arriver à cela est affaire de spécialiste. Si on apprend à faire du vélo, on ne commence pas par vouloir savoir comment est construite la poignée du guidon et la chaîne!
Avec le Dauphin simulé https://www.epsitec.ch/dauphin/ vous pouvez bien comprendre comment un ordinateur fonctionne au plus bas niveau (simulateur sur PC seulement, mais la brochure est sur le site en pdf).
Arduino est pour nous un "IDE", un environnement de programmation basé sur du C. Nous avons ajouté nos propres couches de librairie, qui simplifient considérablement l'écriture de programmes motivants.

Installation

Avant de pouvoir exécuter des programmes avec la carte Edu-C, il faut avoir installé l'IDE Arduino les les libraires EduC et Oled, selon la documentation sous www.didel.com/educ/EduC-InstallExpress.pdf ou www.didel.com/educ/EduC-Install.pdf .
L'expérience a montré que cette installation nécessite parfois l'intervention d'une personne expérimentés. Une configuration initiale non prévue ou une erreur de clic peut conduire à un comportement difficile à interpréter.

Sources des programmes exemples et installation des librairies

Les programmes s'appuient sur deux librairies: EduC.h et Oled.h. Chaque programme Arduino est dans un dossier, dans lequel on trouve le fichier programme, avec l'extension .ino, et les fichier de définitions et fonctions préparés pour la carte Edu-C, les fichiers EduC.h et Oled.h.
Le tout est dans le fichier que l'on prend sous www.didel.com/educ/Educ-Mod.zip.
Il y a deux de préparer le PC ou Mac pour progresser dans le cours en perdant un minimum de temps à charger les fichier.
Suivre la procédure décrite dans ArduinoInstall.pdf et charger EduC.zip et Oled.zip. Les programmes suivent et on les charge via les menus "Fichier-Exemples-EduC-Eduxx"

Une autre solution à utiliser si une librairies a été oubliée, ou que l'on veut la modifier, est d'ajouter sous "Croquis-Ajouter un fichier" la librairie déclarées manquante ou à modifier. Le compilateur regarde d'abord si le fichier inclus est dans le dossier du programme, si non, il cherche dans la zone des librairies Arduino.

Avertissement pour l'encadrant: ces opérations ne sont pas familières à un jeune. Il faut réserver suffisament de temps pour l'aider à installer, s'il doit utiliser un portable personnel, ce qui est préférable.

Structure de tous les programmes

Le processeur de la carte Edu-C sait faire beaucoup de choses, et on va lui demander très peu, mais il faut dire exactement quoi, Il faut le configurer, dire que telle pin est une sortie, telle autre une entrée. C'est le "setup" qui se fait une fois au début. Ensuite on entre dans une boucle principale, qui peut contenir des boucles d'attente ou de décision. On ne peut pas stopper ou faire attendre le processeur: on le met dans une boucle où il ne fait rien! Il exécute plus de un million d'instructions par seconde, et il ne se fatigue pas à poser un million de fois la même question.

1er programme: clignoter

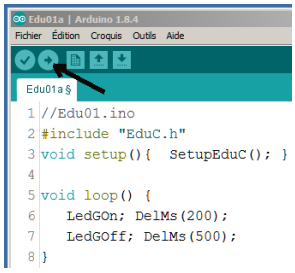
Sur Edu-C il y a 2 leds en haut marquées LG et LD. On peut les allumer avec les instructions `LedGOn;` `LedDOn;` et les éteindre avec les instructions `LedGOff; LedDOff;`
Jouons d'abord avec les deux leds marquées LG, LD, en haut de la carte.

Dans la boucle de programme, on va allumer, attendre, éteindre, attendre.
Avant, il faut dire que l'on a ajouté aux instructions du langage C une couche de définitions qui est un fichier appelé EduC.h et il faut appeler dans le setup la fonction SetupEduC(); qui met tout en place.
Ce fichier EduC.h, a été caché dans la zone bibliothèque d'Arduino lors de l'installation.
Pour trouver le programme, on passe par – Fichier – Exemple – EduC – Edu01aCli – (si EduC a été correctement installé avec la version de jjanvier 2018).

Voilà enfin le programme:

```
//Edu01aCli.ino clignote
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
   LedGOn; DelMs(200);
   LedGOff; DelMs(500);
}
```



Vous avez compris que `DelMs(200);` est une attente de 200 millisecondes.
Vous avez aussi vu que les instructions se terminent toujours par un `;`
On verra plus tard pourquoi il y a des parenthèses, parfois vides. Et pourquoi ce "void".
Et naturellement minuscules et majuscules comptent. Attention au l et L, au 0, O et o.
Le compilateur dit "not declared" s'il n'a pas reconnu.
Pratique: sous – Fichier – Preferences – demandez l'affichage du numéro de ligne. Quand il y a beaucoup d'erreurs, c'est pratique.

Le programme se compile et se transfère en cliquant la flèche dans le disque bleu. Pendant le transfert, la LED tricolore de Edu-C clignote en vert.

Modifions le programme

Avant de modifier la fonctionnalité d'un programme, il faut toujours changer son titre, le renommer et le sauver (donner le nom sans le .ino). Préparez un dossier avec un nom et une date, type ExoEduC180113, dans le bureau ou ailleurs, évidemment pas dans une zone réservée par Arduino.
On sauve ensuite fréquemment avec le 5e rectangle bleu. Il y a aussi sous – Preferences – une option pour que sauvetage soit automatique.

Les programmes Arduino sont des dossier qui contiennent le texte du programme avec l'extension .ino et éventuellement des librairies propres avec un .h. Quand on sauve, on donne seulement le nom du dossier. Attention de ne pas sauver dans le dossier d'un programme, à côté d'un .ino.
Vous pouvez garder autant de .ino dans votre écran que vous voulez, il y a la petite croix pour les détruire. Utilisez le copier-coller pour éviter les fautes de frappe.

Retour à notre programme.

Que peut-on essayer de modifier dans la boucle? Les durées évidemment. Est-ce que l'on voit encore la led si on l'allume seulement 1ms? Est-ce que si on met 0 ms, la led ne s'allume plus? Que faire d'un peu plus compliqué puisque j'ai 2 leds? Il y a la LedG, et les nons pour la tricolore c'est Rouge Vert Bleu.

Pour les noms des programmes, on a Edu01a, 02a, ... pour nos exemples. 01b, 01c, .. est le nom conseillé pour vos variantes.

```
//Edu01bCli.ino clignote plus vite
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
  . . . vos modifications sur Edu01aCli.ino
}
```

Toggle veut dire basculer. Comment clignoter avec le moins d'instructions possibles?

```
//Edu01cCli.ino clignote plus simple
#include "EduC.h"
void setup () { SetupEduC(); }
void loop() {
  LedGToggle; DelMs(500);
}
```

LedGToggle; LedDToggle; existent. Il y a aussi RougeToggle, VertToggle; BleuToggle; et naturellement en On/Off.
On essaie un petit show lumineux? On peut mettre plusieurs instructions sur une ligne (ce n'est pas conseillé pour les programmes complexes), mais il faut toujours regrouper intelligemment.

2e programme: interagir avec le `if`

Il y a 2 poussoirs pour lesquels on a défini en bibliothèque 2 noms: PousG et PousD.
Ces noms peuvent prendre 2 valeurs: 0 ou faux si le poussoir est relâché. 1 ou vrai s'il est pressé et que le contact électrique envoie la bonne tension vers le processeur.
Il faut utiliser maintenant l'instruction **`if`** du langage C.

```
if ( vrai? )  { faire ceci; }
```

Dans la parenthèse, on a une condition, vrai/faux, 0/1
Dans l'accolade, on a une action, une ou plusieurs instructions terminées par un **;**
Ne pas confondre parenthèses et accolades. Les espaces sont libres.

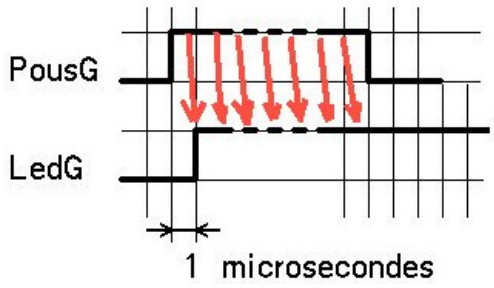
Pour allumer la LED gauche si on presse sur le bouton gauche, il faut écrire

```
if ( PousG )  { LedGOn; }
si ( vrai ou faux?) { faire si vrai }
```

Mettons cette instruction dans la boucle d'un programme

```
//Edu02aPous.ino allume si on presse
#include "EduC.h"
void setup(){ SetupEduC(); }

void loop() {
   if ( PousG )  { LedGOn; }
}
```



Il faut bien comprendre que le programme tourne sans arrêt dans la boucle. Si `PousG` est vrai il allume ou redonne l'ordre qui allume. Si `PousG` est faux, il ne fait rien mais repose la question sans cesse, environ un million de fois par secondes!

On charge le programme et il s'exécute (il tourne dans la boucle). La LedG est éteinte. On pèse, elle s'allume, Mais elle reste allumée!
Normal, on ne lui a jamais dit de s'éteindre.

Quels sont les choix pour éteindre cette LED?

1. Quand on relâche. Pour inverser un état, le C utilise le signe **!** Donc `!PousG` est vrai si on ne presse pas. Ajoutons dans la boucle l'instruction (programme Edu02b.ino à faire et sauver)
   ```
   if ( !PousG )  { LedGOff; }
   ```

2. On presse sur le 2e bouton pour éteindre en ajoutant l'instruction
   ```
   if ( PousD )  { LedGOff; }
   ```
   (programme Edu02c)

3. On éteint au bout d'une seconde. Il faut modifier la première instruction et écrire (Edu02d)
   ```
   if ( PousG )  { LedGOn; DelMs(1000); LedGOff; }
   ```

Tiens, cela ne serait-il pas plus logique d'écrire (Edu02e)
```
if ( PousG )  { LedGOn; }
DelMs(1000); LedGOff;
```
Essayez, cela marche parfois seulement. Pourquoi? Vous comprendrez avec le programme suivant.

Vous avez bien créé et testé les programmes `Edu02b.ino Edu02c.ino . . .?` Vous verrez que c'est très utile de pouvoir retrouver les anciens programmes que l'on a fait, mêmes très simple.

Nouveau programme

Comment programmer "La LED s'éteint et s'allume à chaque pression"?
J'imagine que la solution est d'attendre après chaque pression:
```
//Edu03aeEtAl.ino Devrait allumer-éteindre
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
if ( PousG )  { LedGOn; }
  DelMs(1000);
if ( PousG )  { LedGOff; }
  DelMs(1000);
}
```
Voyons, je pousse et j'attends une seconde. Je repousse, cela doit s'éteindre. Tiens, je presse et il ne se passe rien.
La raison est que le `if` n'attend pas! Il teste, décide et passe plus loin. L'instruction suivante c'est 1 seconde d'attente sans s'occuper du poussoir. Si vous pressez rapidement, le `if` a une très petite chance de tomber au bon moment. Si vous pressez plus d'une seconde, le 2e `if` va aussi voir votre action. Il faut trouver autre chose!

Instruction `while() {}`

Avec le `if(),` on teste et on passe plus loin. Avec le while on attend (*tant que*)

```
while ( c'est encore vrai? )  { faire ceci; }
```

On fait *"tant que la condition est vraie"* et on passe à l'instruction suivante dès que la condition est fausse..
```
//Edu04aPousTog.ino Le poussoir fait basculer
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
   while (!PousG) { } // c'est relâché, on attend
   LedGToggle;            // on a pressé, on allume LedG
   while (PousG) { }  // c'est pressé, on attend
}
```

Les accolades vides sont nécessaires quand il n'y a rien à faire. On peut aussi mettre un ; (instruction vide).
Le programme est juste, mais cela n'est pas fiable, on expliquera dans la section suivante.

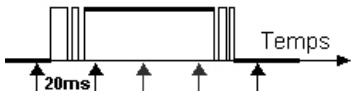
Modifiez ce programme pour utiliser LedGOn; et LedGOff; au lieu de LedToggle; Créez `Edu04bPousTog.ino`

Rebonds de contact

Un interrupteur mécanique ne passe pas d'un coup de l'état vrai (pressé) à l'état faux. Il y a des mauvais contacts, qui font comme si l'on pressait très rapidement (milliseconde et moins). La solution est de lire le poussoir toutes les 20 ms. On ne peut de toute façon pas presser plus vite que 10 fois par secondes. La règle à retenir est toutes les fois que l'on surveille le changement d'état d'un poussoir, il ne faut pas le faire plus souvent que toutes les 20ms.
Un programme fiable est donc:

```
//Edu05aSansReb.ino Poussoir sans rebonds
#include "EduC.h"
void setup () { SetupEduC(); }

void loop() {
   while (PousG) { DelMs(20); }
   LedGToggle;
   while (!PousG) { DelMs(20); }
}
```



Ce qu'il ne faut pas oublier

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.

On n'a pas vu mais c'est facile de comprendre:

```
if (vrai) {faire si vrai}
else {faire si faux}
```

On en verra des exemples plus loin.

Si j'écris **`while(1) { }`** que se passe-t'il? La condition est toujours vraie, et il n'y a rien à exécuter. Le processeur tourne à ne rien faire. C'est la "fin" d'un programme, plus rien ne change.
Pour quitter cette boucle, il faut charger un nouveau programme (ou le même) ou peser sur le bouton reset, ou couper l'alimentation et la remettre, et le programme recommence à la première instruction.

Dans la pratique, on écrit `while (1) ;`
Une accolade vide est une instruction neutre, comme un `;` seul.

Si j'écris **`while(1) { des tas d'instructions; }`** que ce passe-t-il? Le "tas d'instructions" va se ré-exécuter sans cesse. On a fait un "loop" dans le loop { } principal;

Si j'écris **`while(!Pous) { mon tas d'instructions; }`** tant que Pous n'est pas pressé, le programme tourne, il se bloque à la fin du tas d'instructions si vous pressez.

Vous verrez à la fin du programme Edu28a_lune l'instrucion **`while(!Pous);`** Elle évite que la mélodie tourne en boucle. On pèse quand on veut écouter, on enlève l'instruction ou on met une pincette sur le poussoir si on veut que la mélodie persiste.

Que font `if(0){}` et `if(1){}` ? -- rien du tout, le compilateur va même les ignorer puisqu'elles sont inutile.

Fonctions EduC.h utilisées dans ce module

DelMs (xx); Délai bloquant de xx= 1 à ~32000 millisecondes.

LedGOn; LedGOff; LedGToggle; LedDOn; LedDOff; LedDToggle;

RougeOn; RougeOff; VertOn; VertOff; BleuOn; BleuOff; BlancOn; BlancOff;

PousD PousG valeurs booléennes - PousD vaut HIGH (vrai, 1) si pressé .
!PousD !PousG valeurs inverses !PousD vaut HIGH (vrai, 1) si relâché.