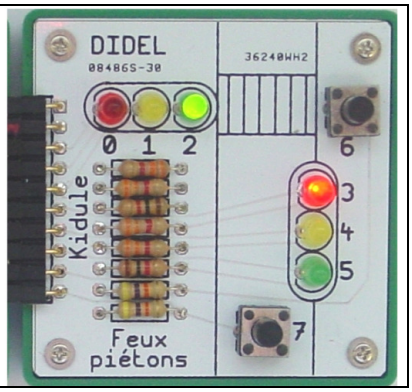
Kidule Feux

Programmer en C - bien dire pour bien faire

Ou l'on voit que le kidule Feux aide à comprendre la structure des programmes, à apprendre à respecter des règles d'écriture et à se familiariser avec quelques instructions C.
On comprendra aussi comment allumer une LED et lire un poussoir.

Le kidule Feux simule un feu pour piétons avec son bouton de demande. Les voitures arrivent par le bas de l'image. Le poussoir peut simuler la densité de trafic voiture ou jouer un rôle quelconque.
On voit que les LEDs et poussoirs ont un numéro, qui permet de savoir comment faire passer l'information vers le processeur.
Par exemple, la LED RougePieton est câblée sur la pinArduino 14. Plus tard, on verra que cette pin est en fait le bit 0 du portC.

	Bits	Arduino Pin	Name	Gnd +5V
PortC	0	14	Rpieton	0
	1	15	Jpieton	1
	2	16	Vpieton	
	3	17	Rvoit	
	4	18	Jvoit	4
	5	19	Vvoit	
PortD	2	2	PousPieton	
	3	3	PousVoit	7



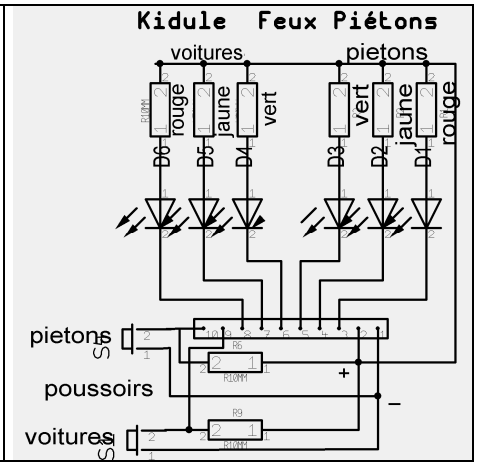
Structure d'un programme C et Arduino

Quel que soit le langage, on dot définir les objets avec lesquels on va travailler, puis initialiser le processus, puis exécuter la tâche répétitive.

Ce qui intéresse le débutant, c'est d'exécuter le programme. Si on veut voir une LED clignoter, il faut l'allumer, attendre, l'éteindre, attendre. Arduino vous facilite la compréhension en numérotant les pins et en ayant défini des fonctions pour les initialiser, les activer et pour attendre, c'est-à-dire adapter la vitesse des programmes à la vitesse de nos sens.

Faisons clignoter la **LED verte piéton**. On voit sur le schéma que pour l'allumer, il faut que la pin 2 (broche 5 du connecteur) soit à zéro (proche de 0 Volt). C'est la pin 16 Arduino.
La boucle de programme est simple; elle doit être précédée du setup qui dit que la pin est en sortie.

```
// Feux0.ino (dans CKiFeux.zip)
void setup () {
   pinMode (16,OUTPUT);
}
void loop () {
   digitalWrite (16,LOW);
   delay (1000) ;  // 1000 ms = 1 seconde
   digitalWrite (16,HIGH);
   delay (1000) ;
}
```



Ce programme est complet. On remarque les accolades qui parenthèsent les groupes d'instructions et les points-virgules à la fin des instructions. Le compilateur est non seulement intolérant, mais peu coopérant pour signaler clairement les erreurs faites.
Les programmes se trouvent sous www.didel.com/kidules/CKiFeux.zip

Déclarer c'est clarifier

Revenons sur l'instruction pour allumer la LED verte :

```
digitalWrite (16,LOW);
```

Elle n'apprend pas grand chose à quelqu'un qui vous lit ! Ou à vous-même dans 2 mois.
Il faut absolument donner des noms aux objets que l'on manipule, et mieux encore, donner des noms aux opérations que l'on exécute. L'objet est la LED verte du feu piéton, et on peut l'appeler selon les goûts *LedVerteDuFeuxPieton* ou *Led_verte_du_feux_pieton* (ne pas mettre de lettre accentuée) ou plus simplement ***VertPieton*** qui est assez explicite.

`#define VertPieton 16` permet d'écrire dans le programme :

```
digitalWrite (VertPieton,LOW);
```

Cette instruction exécute "Allumer le vert" et dans tout le programme il faudra se souvenir que pour allumer il faut mettre LOW car la LED est câblée vers le + ! Vous avez un autre projet où c'est câblé vers le moins et vous allez confondre.
Donnons un nom à cette opération; ***VertPietonOn***. Si ce nom n'est pas assez parlant pour vous et vos lecteurs, modifiez-le.

```
#define  VertPietonOn  digitalWrite (16,LOW)  // Allume Vert piéton sur la pin 16
```

Notez que les deux informations, 16 et LOW, sont nécessaires pour caractériser la LED
Cette ligne n'est pas une instruction, le #define ajoute une ligne dans un dictionnaire. On écrit dans le programme `VertPietonOn ;` et le précompilateur va chercher l'équivalent dans le dictionnaire et remplace.

Encore un point avant d'écrire le programme qui clignote le vert. Les pins du microcontrôleur peuvent être des entrées ou des sorties. Arduino a défini la fonction
`pinMode (pin,OUTPUT);` pour initialiser le mode "sortie".

Cette instruction se met dans la fonction "setup" (configuration, initialisation), exécuté une seule fois avant que le programme parte dans sa boucle éternelle.

```
void setup ()
{
 pinMode (VertPieton,OUTPUT);
}
```

Voilà, on est enfin prêts pour écrire le programme qui clignote le vert piéton. On voit que la difficulté de compréhension n'est pas dans le programme, mais dans les entrées-sorties qui doivent être nommées et définies le plus clairement possible.

```
// Feux1.ino  Clignote VertPieton
#define  VertPieton  16
#define  VertPietonOn  digitalWrite (VertPieton,LOW)  // Allume
#define  VertPietonOff digitalWrite (VertPieton,HIGH) // Eteint

void setup ()
{
  pinMode (VertPieton, OUTPUT) ;
}

void loop ()
{
  VertPietonOn;
  delay (1000);
  VertPietonOff;
  delay (1000);
}
```

On pourrait préférer déclarer `VertPietonOn` comme une fonction, et dans ce cas il faudrait l'écrire `VertPieton();` Pour déclarer du matériel nous préférons le `#define`.

On peut jouer avec les autres LEDs, mais étudions d'abord les poussoirs sur la carte.

Lire un poussoir

On voit dans le schéma de la page précédente que le poussoir court-circuite la pin vers le 0V (LOW, 0) et que lorsque le poussoir n'est pas activé, la pin est au +5V (HIGH, 1).

La fonction Arduino `digitalRead (pin) ;` rend une valeur 0 ou 1 selon l'état de la pin, donc du poussoir. 0 si le poussoir est pressé, 1 si relâché.

En général, on teste cette valeur binaire (on dira dans certains cas valeur booléenne en pensant à vrai/faux) avec l'instruction C `if (condition) { faire .. }`

Comme premier exercice, on veut allumer la LED verte sur la pin 16 quand on presse le bouton piéton. On voit sur les figures précédentes que le poussoir piéton est sur la pin Arduino 2 et qu'il est actif à zéro .
Donc on peut écrire

```
void loop ()
{
   if (digitalRead(2) == 0)  { VertPietonOn}
   else  { VertPietonOff}
}
```

Le `if ( )` est l'instruction C la plus facile à comprendre: Si le contenu de la parenthèse est vrai, c'est-à-dire égal à 1 (différent de 0 serait plus correct), un groupe d'instructions entre accolades est exécuté. La condition est que la lecture du poussoir sur la pin 2 est à zéro (donc poussoir pressé). Si cette condition et vraie, on allume VertPieton, autrement (else) on éteint. Le `==` est nécessaire pour distinguer l'**assignation , par exemple** `a = b+4;` et la **comparaison**, "est-ce que `a` est égal à `b+4` ?"

Si vous avez compris ce qui a été dit avant, ce "digitalRead (2) " dans une boucle est incongru, mal élevé, pour ne pas dire immoral. Dépêchons-nous de lui donner un nom clair:

```
#define  PousPieton 2
```

Ce qui permettra d'écrire

```
digitalRead(PousPieton) == 0) {  }
```

Mais ce n'est pas encore assez clair. donnons un nom à la condition vraie/fausse "le piéton demande le passage?"

```
#define DemandePieton  digitalRead(PousPieton) == 0
```

Le programme a maintenant une lisibilité parfaite:

```
if (DemandePieton) {VertPietonOn ;}
else  { VertPietonOff ; }
```

Ce programme se trouve sous Feux2.ino avec un petit supplément pour activer le rouge

Parenthèse

Que pensez-vous de cette instruction ?

```
digitalWrite (VertPieton, digitalRead (PousPieton) ;
```

Elle fait la même chose que le if-else précédent !
C'est important de bien comprendre, et de se forcer à analyser en détail ce qui se passe.

`digitalRead (PousPieton)` fournit au processeur une valeur 0 ou 1, 0 si poussoir pressé, 1 si relâché. Cette valeur reste "en l'air" en quelque sorte. Il faut une instruction pour dire ce qu'il faut en faire.

`digitalWrite (VertPieton, xx);` écrit la valeur `HIGH=1, LOW=0,` ou une valeur `xx` calculée par une instruction précédente, ici notre digitalRead. Pas plus difficile que cela!

Quand est-ce qu'on programme?

Patience! Si vous voulez écrire des pages de digitalWrite, allez-y, vous n'avez pas besoin de mon aide.

On a vu qu'un programme a trois parties

- les définitions, qui décrivent le matériel connecté à la carte (signaux et fonctionnalité)
- le setup, qui initialise ce matériel
- la boucle de programme

Par la suite, on extraira de cette boucle de programme des fonctions pour faire comme en architecture des éléments préfabriqués que l'on peut utiliser pour diffférentes constructions.

Définitions du kiduleFeux

Ecrivons toutes les définitions qui expriment le câblage électrique et nous permettront de travailler avec des noms logiques.

```
#define RougePieton 14
#define  VertPietonOn  digitalWrite (14,LOW)
#define  VertPietonOff  digitalWrite (14,HIGH)
#define JaunePieton 15
#define  VertPietonOn  digitalWrite (15,LOW)
#define  VertPietonOff  digitalWrite (15,HIGH)
#define VertPieton 16
#define  VertPietonOn  digitalWrite (16,LOW)
#define  VertPietonOff  digitalWrite (16,HIGH)
```

```
#define RougeVoiture 17
#define  VertPietonOn  digitalWrite (17,LOW)
#define  VertPietonOff  digitalWrite (17,HIGH)
#define JauneVoiture 18
#define  VertPietonOn  digitalWrite (18,LOW)
#define  VertPietonOff  digitalWrite (18,HIGH)
#define VertVoiture 19
#define  VertPietonOn  digitalWrite (19,LOW)
#define  VertPietonOff  digitalWrite (19,HIGH)
```

```
#define  PousPieton 2
#define DemandePieton  digitalRead(PousPieton) == 0
#define  PousVoiture 3
#define VoiturePasse  digitalRead(PousVoiture) == 0
```

Pour le set-up, nous pouvons aussi préparer ce qui sera utilisé chaque fois au démarrage du programme: la configuration (les pinMode) et l'état initial.

```
void setup ()
{
  pinMode (RougePieton, OUTPUT) ;
  pinMode (JaunePieton, OUTPUT) ;
  pinMode (VertPieton, OUTPUT) ;
  RougePietonOn ;
  JaunePietonOff;
  VertPietonOff ;
```

```
  pinMode (RougeVoiture, OUTPUT) ;
  pinMode (JauneVoiture, OUTPUT) ;
  pinMode (VertVoiture, OUTPUT) ;
  pinMode (PousPieton, INPUT) ;
  pinMode (PousVoiture, INPUT) ;
  RougeVoitureOff ;
  JauneVoitureOff ;
  VertVoitureOn ;
}
```

Programme piéton simplifié

Ce premier programme ”réel” (Feux3.ino) coupe le trafic et donne le passage pour 2 secondes. Il n'affiche pas le jaune.

```
void Loop () {
  if (DemandePieton) {
    VertVoitureOff ;
    RougeVoitureOn ;
    delay (500);  //les Voitures s'arrêtent
    RougePietonOff ;
    VertPietonOn ;
    delay (2000); //le piéton passe
    VertPietonOff;
    RougePietonOn ;
    delay ((500) ; // laisser finir de traverser
    RougeVoitureOff ;
    VertVoitureOn ;
  }   // traffic rétabli, on recommence à tester
}
```

Exercice: Ajouter la transition par le jaune, affiché 200ms

Exercice: Attente pour ne pas redonner le vert piéton tout de suite
Il faut, tout en surveillant le poussoir piéton pour mémoriser qu’il y eu demande, décompter toutes les 100ms par exemple pour n’accorder cette demande qu’après le délais.

Exercices: Tenir compte du trafic (poussoir voiture) pour agir sur le temps d’attente piéton.
Ajouter un délai pour ne pas couper immédiatement le traffic si une 2^{e} demande est immédiate.
Comment faire si un piéton bloque le poussoir?

jdn 131104/140127/180601