Chapitre 17 - Analyse des tâches à exécuter et écriture du programme

Initiation à la programmation

Tout le monde sait programmer mais tout le monde ne sait pas qu'il sait programmer !

Je vais essayer de vous démontrer ce que je viens d'affirmer. Imaginez que vous vouliez changer les quatre roues de votre voiture pour mettre des roues munies de pneus pour la conduite en conditions hivernales. Comment feriez-vous ?

Nous allons voir comment **décomposer ce travail en tâches élémentaires**, en essayant de ne rien oublier. Vous avez positionné votre véhicule sur un terrain plat et serré le frein à main.

Première roue :

- Retirer l'enjoliveur
- Débloquer les quatre boulons
- Positionner le cric à l'endroit repéré au plus près de la roue
- Monter le cric jusqu'à ce que la roue décolle du sol
- Retirer complètement les quatre boulons
- Dégager la roue
- Positionner la nouvelle roue à pneus spéciaux
- Remettre les quatre boulons
- Descendre le cric complètement jusqu'à ce que la voiture soit sur le sol
- Dégager le cric de la voiture
- Serrer à fond les quatre boulons
- Remettre l'enjoliveur

Roue N°2, N°3 et N°4 :

- Faire comme pour la première roue

Vous êtes parfaitement capable de suivre la procédure écrite ci-dessus, car vous savez lire ce langage tiré de la langue de Molière. Notons au passage qu'il y a une répétition des consignes parce qu'il y a quatre roues à traiter de la même façon. Nous pourrions écrire une boucle, sorte de procédure capable de **répéter un certain nombre de fois une série d'instructions**.

Je vais maintenant transcrire cette procédure écrite en langue de Molière dans un **langage inventé de toutes pièces** pour un hypothétique robot que vous venez d'acquérir.

```
SETUP {
Voiture_a_plat ;
Frein_serré ;
}
PROGRAM {
        BOUCLEx4 {
                Retirer_enjoliveur ;
                BOUCLEx4 {
                        Debloquer_boulon ;
                }
                Positionner_cric ;
                Monter_cric ;
                BOUCLEx4 {
                        Retirer_boulon ;
                }
                Degager_roue ;
                Positionner_nouvelle_roue ;
                BOUCLEx4 {
                        Mettre_boulon ;
                }
                Descendre_cric ;
                Degager_cric ;
                BOUCLEx4 {
                        Serrer_boulon ;
                }
                Mettre_enjoliveur ;
        }
}
```

Commentons un peu ce programme. Il est constitué de deux fonctions ; la fonction SETUP (repérée en vert) qui permet d'initialiser certaines conditions et la fonction PROGRAM (repérée en rouge) qui est le programme principal. Cela devrait déjà vous rappeler quelque chose…

On remarque que chaque instruction commence par une majuscule et se termine par un point-virgule, et ne comporte pas d'accent ; c'est la syntaxe de notre langage et toute entorse à ces règles génère une erreur de compilation de la part du robot.

La procédure BOUCLE permet de répéter un certain nombre de fois une série d'instructions comprises entre crochets (x4, donc quatre fois car la voiture a quatre roues qui ont chacune quatre boulons).

Il faut autant d'accolades ouvrantes que d'accolades fermantes dans le programme.

Un anglais aurait bien sûr du mal à comprendre ce programme car les instructions sont basées sur des mots français ; de même vous aurez peut-être un peu de mal à comprendre le langage d'Arduino car les instructions sont basées sur des mots anglais. Mais cela vaut la peine d'apprendre quelques mots clés pour s'y repérer, d'autant que le nombre d'instructions du langage est très restreint (voir le lexique donné en fin de chapitre).

Comme vous le voyez, programmer n'a rien de sorcier : il suffit de **savoir décomposer une tâche complexe en une suite de tâches élémentaires**, facile à exécuter. Ceci constitue le **travail d'analyse**.

Chaque tâche élémentaire est réalisée par une instruction, qui obéit à une syntaxe ; celle-ci doit être respectée. Ceci constitue le **travail de codage**. Bien entendu, il faut connaître les instructions du langage et quelles tâches elles réalisent.

La difficulté dans la mise au point d'un programme est le travail d'analyse ; s'il est bien fait, le travail de codage coule de source et le programme fonctionne du premier coup. Hélas, trop souvent les gens se mettent à coder sans avoir pris la peine d'analyser ; après, ils doivent mettre des rustines à droite et à gauche pour que cela fonctionne !

Si une série de tâches élémentaires doit être répétée, on peut utiliser des **boucles**. De même, le programme peut **tester des conditions** pour s'orienter vers telle ou telle série de tâches à réaliser ; on parle de tests conditionnels.

Tout ce vocabulaire sera précisé au fur et à mesure des besoins. Ce qui compte pour débuter, c'est de connaître les différentes instructions possibles et ce qu'elles réalisent. Ceci se trouve à la page :

https://www.arduino.cc/en/Reference/HomePage

Nous verrons dans les prochains chapitres comment concilier l'électronique extérieure au microcontrôleur (par exemple des capteurs) avec l'informatique (le programme) qui l'exploite ; comme je vous le disais plus haut, les deux sont liés. Heureusement, d'autres programmeurs ont déjà résolu bien des situations et leur travail est généreusement mis au service de tous.

À retenir sur la programmation :

- Concevoir un programme revient à décomposer une tâche complexe en une série de tâches élémentaires.
- Chaque tâche élémentaire est traitée par une instruction obéissant à des règles de syntaxe qu'il faut respecter.
- On peut utiliser des boucles pour réaliser plusieurs fois de suite les mêmes séries d'instructions.
- On peut tester des conditions et en fonction des résultats, réaliser telle ou telle série d'instructions.
- Le langage d'Arduino contient un nombre restreint d'instructions qu'il est très facile de connaitre.
- La réussite d'un programme dépend avant tout du travail d'analyse de la tâche complexe à réaliser.

Ecriture du programme et « langage » d'Arduino

Maintenant que vous avez compris que programmer revient à décomposer une tâche complexe en une suite de tâches élémentaires, nous allons faire un rapide tour d'horizon des possibilités offertes par le « langage » d'Arduino (mot entre guillemets car ce n'est pas vraiment un langage mais un ensemble de fonctions réalisant des tâches précises dans le logiciel distribué pour programmer les modules Arduino).

Référez-vous au site Arduino, à cette page pour avoir le descriptif de toutes les fonctions (troisième colonne de la page) :

https://www.arduino.cc/en/Reference/HomePage

Je ne vais pas décrire toutes les fonctions qui existent ; je me contenterai de passer en revue ce qui est le plus souvent utilisé, le reste sera vu en fonction des besoins du cours. Ainsi, vous verrez que le nombre de fonctions à connaître est très restreint pour pouvoir tout de même commencer à utiliser vos modules.

Les lignes d'instructions **doivent être terminées par un point-virgule** ; dans ce chapitre, je ne le mettrai pas.

Voici avant tout **quelques points de repères** : Digital signifie numérique alors que Analog signifie analogique. Read signifie lire et Write écrire. Input est l'entrée, Output est la sortie. La broche (du module ou du microcontrôleur) se dit pin. Une attente (ou un retard) c'est delay.

Commentaires : les lignes commencent par // et sont ignorées par le compilateur ; on peut donc placer des commentaires où on veut. Si le commentaire tient sur plusieurs lignes, on peut encadrer celles-ci par /* et */ ce qui évite d'avoir à répéter //. Comme je vous le disais, **il est important de commenter ses programmes** pour pouvoir s'y retrouver quand on reprend le programme après une longue période d'interruption ou bien aussi pour que d'autres programmeurs comprennent comment le programme opère. Bien entendu, il n'est pas nécessaire de commenter ce qui est évident.

Digital I/O : ce sont les fonctions pinMode, digitalRead et digitalWrite qui permettent d'initialiser les broches soit en entrée, soit en sortie et de lire leur état ou de les positionner à un certain état. Nous y reviendrons en détail bientôt.

Analog I/O : ces fonctions servent à gérer une entrée analogique.

analogRead(pin) permet de lire la tension présente sur la broche pin. Le résultat est un nombre entier compris entre 0 et 1023 (0 pour 0 V et 1023 pour 5 V). Sur un Uno, pin va de 0 à 5 puisqu'il y a 6 entrées analogiques.

On peut lire une entrée analogique, mais il n'est pas possible d'aller mettre une tension sur une sortie analogique. Pourtant, il existe une fonction qui s'écrit analogWrite ; en fait, cette fonction sert à générer un signal de type PWM (voir le chapitre 12) sur la broche : ce signal est bien un signal numérique, mais on peut le considérer comme un signal de tension moyenne dépendant du rapport cyclique du signal PWM. Ainsi, pour créer un signal PWM de rapport cyclique 50%, il suffit d'écrire :

analogWrite (pin, value)

où pin est le numéro de la broche analogique (de 0 à 5) et value est compris entre 0 et 255 (0 pour 0% et 255 pour 100%, avec 127 on obtient un rapport cyclique de 50%).

Le signal généré reste sur la broche jusqu'à ce qu'on utilise cette broche pour autre chose. On aura l'occasion de revenir à cette fonction très utile pour moduler la vitesse de nos moteurs électriques.

Avec une ligne d'instruction, nous avons obtenu un signal PWM de fréquence approximative 490 Hz (ou 980 Hz si broche 5 ou 6 sur le Uno) ; ceci est à comparer au montage donné par la figure 12.9 ou 12.10 du chapitre 10. Comme je vous le disais, l'électronique programmable est plus simple et moins chère.

Time : ces fonctions gèrent le temps. Pour commencer, deux fonctions sont à connaître :

delay(duree) et delayMicroseconds(duree) où duree est la durée en millisecondes pour la première et en microsecondes pour la secondes.

L'inconvénient de ces deux fonctions est qu'**elles bloquent le fonctionnement du microcontrôleur qui ne fait rien d'autre pendant qu'il attend**. On verra comment surpasser cet inconvénient grâce à millis() et micros() qui seront décrites ultérieurement. On a déjà utilisé delay dans le programme Blink.

Random numbers : ces deux fonctions randomSeed et random permettent de générer des nombres aléatoires. Cela ne vous servira pas à gagner au loto mais peut vous aider à reproduire des phénomènes aléatoires sur un réseau comme par exemple un lampadaire d'éclairage public qui a un mauvais contact ou bien les flashes émis par un soudeur à arc.

La suite de la colonne de fonctions de la page Arduino sera décrite en fonction des besoins. Par exemple, vous utiliserez assez peu en modélisme ferroviaire, les fonctions mathématiques ou trigonométriques. Par contre, le reste concerne des fonctions pour manipuler des octets ou bien des bits individuels ; cela n'est pas nécessaire pour débuter et sera vu au fil du cours. Il y a aussi des fonctions de **communication** qui sont très utiles lorsque nous utilisons le moniteur inclus dans l'IDE.

Les structures

Intéressons-nous maintenant à la première colonne qui décrit les structures.

setup et loop sont les deux fonctions principales d'un programme Arduino et doivent absolument figurer, même si elles sont vides de toute instruction.

```
void setup () {                    }

void loop () {                     }
```

On met les instructions **à l'intérieur des accolades**. Ces deux fonctions sont automatiquement générées par l'IDE version 1.6.12 et il n'y a plus qu'à les compléter. On a déjà donné des exemples d'utilisation de ces deux fonctions.

Control structures : on a déjà un peu évoqué ce sujet plus haut en parlant de boucles ou de tests conditionnels.

Boucles : pour créer une boucle, on peut utiliser un compteur qu'on initialise à une certaine valeur et qu'on incrémente (ou décrémente) jusqu'à ce qu'il soit égal à une valeur finale qui attestera que la boucle a bien été répétée un certain nombre de fois. La **boucle for** est la plus couramment utilisée et sa syntaxe est :

```
for (initialisation ; condition ; incrément) {

mettre ici les instructions ;

}
```

Par exemple, si j'appelle k mon compteur (k est un entier donc int pour integer), j'aurais :

```
for (int k = 1 ; k < 5 ; k++) {

Mettre ici les instructions

}
```

k++ signifie qu'on incrémente le compteur à chaque fois (on aurait pu mettre k-- pour le décrémenter)

La condition est : quand k devient égal à 5, on sort de la boucle.

D'autres boucles existent sans compteur, comme **while** (tant que condition valide, faire) ou bien **do… while** (faire et tant que condition valide, refaire). La différence entre ces deux boucles est le moment où on teste la condition ; while, la condition est testée **avant** chaque itération, do...while, la condition est testée **après** chaque itération. La do… while est, de ce fait, **exécutée au moins une fois**, ce qui n'est pas forcément le cas de la boucle while.

Il y a aussi les tests qui permettent de **tester des conditions** et en fonction du résultat d'exécuter telle ou telle série d'instruction.

If (si la condition est vraie, alors exécuter ce qui suit)

If… else (si la condition est vraie, alors exécuter ce qui suit, **autrement** allez exécuter ce qu'il y a là)

Switch (utilisé avec case) : en fonction de la valeur d'une variable (1, 2, 3, …) on exécute le cas 1 ou le cas 2 ou le cas 3. On sort des différents cas grâce à l'instruction break qui permet d'ignorer les lignes suivantes.

return se positionne à la fin d'une fonction pour lui dire de revenir au programme principal à l'endroit où il en était lorsqu'il a appelé la fonction.

goto : aller à. **On évitera de l'utiliser** car sinon, le programme ne sera pas structuré et ressemblera à du BASIC des années 80 !

Arithmetic operators : ce sont les signes d'opérations bien connues. % est l'opérateur modulo qui donne le reste de la division entière.

Comparison operators : opérateurs de comparaison. Par exemple if (x == 0) : si x est égal à 0. On remarque que **le signe égal est doublé** (==) ; en effet, on demande est-ce que x est égal à 0, on ne force pas x pour qu'il soit égal à 0.

Boolean operators : opérateurs booléen (voir le chapitre 9). Ce sont nos traditionnelles fonctions logiques (AND, OR, NOT) qui peuvent être recréées par logiciel plutôt que par des portes logiques.

Le reste de la colonne demande une certaine habitude de la programmation et sera donc étudié ultérieurement.

Enfin, la colonne du milieu traite des **variables** (dans le sens de toute donnée numérique ou alphanumérique) ; ceci sera traité dans un chapitre à part et en fonction des besoins. Pour l'instant, retenons que les variables doivent être déclarées pour que le compilateur réserve la place mémoire nécessaire ; nous verrons en effet que selon leur type, elles exigent une place mémoire différente pour y stocker un nombre d'octets qui dépend du type (nombre entier, nombre réel, caractère alphanumérique, etc.)

Rappelez-vous qu'on ne devient pas bilingue du jour au lendemain ; mes explications peuvent vous paraitre confuses mais elles seront **réexpliquées par la suite en fonction des exemples traités**. Pour l'instant, il s'agit de prendre ses marques et ceci n'est possible qu'en écrivant des programmes ou en analysant ceux qui ont été donnés en exemple et dont on peut s'inspirer. C'est donc bien par la pratique que tout cela va se mettre en place.

À retenir sur le « langage » d'Arduino :

- Le « langage » d'Arduino est constitué d'un nombre restreint de fonctions à connaitre mais toutes ne sont pas indispensables pour débuter une petite application.
- Quelques mots-clés anglais sont à connaitre pour mieux comprendre la fonction.
- Les fonctions pour initialiser les broches du module.
- Les fonctions pour lire (entrée) ou écrire (sortie) sur les broches du module, qu'elles soient analogiques ou numériques.
- Comment créer une attente dans un programme.
- Les trois différents types de boucles et surtout la boucle for.
- Les tests conditionnels avec if ou avec switch.
- Les différents opérateurs (arithmétiques, de comparaison, booléen).

Lexique anglais-français pour mieux comprendre Arduino

Pour ceux qui ne parlent pas l'anglais, et afin qu'ils ne se découragent pas du langage Arduino, voici un petit lexique qui permet de mieux comprendre certains termes du langage de programmation d'Arduino.

setup : initialisation
loop : boucle (ici, boucle sans fin puisque la fonction loop est rejouée sans arrêt)
if : si
if … else : si … autrement (sinon)
for : pour (un indice qui va de tant à tant)
switch case : se brancher en fonction du cas qui se présente
while : pendant que (ou tant que)
do … while : faire (une fois) puis continuer à faire pendant que (ou tant que)
break : rompre (ce que vous êtes en train de faire, donc sortir, arrêter, …)
continue : continuer
return : retourner (à ce que vous faisiez avant)
goto : aller à …
#define : définir
#include : inclure
HIGH, LOW : état HAUT, BAS (c'est-à-dire soit tension = 5 V, soit = 0 V)
INPUT : Entrée
OUTPUT : Sortie
INPUT_PULLUP : Entrée haute impédance
LED_BUILTIN : LED présente par construction sur le module (pour le Uno sur la sortie 13)
TRUE : Vrai
FALSE : Faux
integer : nombre entier
integer constants : constantes entières
floating point constants : constantes réelles (nombre à virgule, qui, en informatique ou en anglais, est un point)
void : vide
boolean : booléen (propre à l'algèbre de Boole qui définit les règles de logique)
char : caractère
unsigned : sans signe (donc positif)
byte : octet
int : entier (pour integer)

word : mot
long : long
short : court
float : nombre réel
double : double
string : ficelle, corde, mais aussi train (dans le sens de choses qui se suivent) et donc, en informatique, chaîne (de caractères)
array : tableau
variable : variable
scope : portée
static : statique
volatile : volatile
const : constante
sizeof : taille de …
pinMode : mode de fonctionnement d'une broche (pin) (en entrée ou en sortie)
digital : digital (ou encore numérique, composé de 0 et de 1)
Write : écrire
Read : lire
analog : analogique
random : aléatoire
Set : positionner à 1
Clear : remettre à zéro (ce qui a été positionné à 1), effacer
bit : un bit (0 ou 1)
Serial : série

Comme vous le voyez, certains mots sont identiques, d'autres se ressemblent fortement. Ce petit lexique vous aidera à mieux comprendre la signification des ordres du langage de programmation. À force de manipuler le langage Arduino, vous finirez par vous sentir très à l'aise, même si vous n'avez jamais pratiqué l'anglais.