Chapitre 2 : Le langage Arduino

I. Introduction

La programmation sur Arduino se fait dans un langage qui s'inspire à la fois du C et du C++. Le C++ intervient surtout pour la création de librairies.

II. Les structures de condition et d'itération

II.1 Structure conditionnelle IF ... ELSE

La structure conditionnelle **IF ... ELSE** permet de vérifier une condition logique. Elle repose donc sur l'évaluation d'une expression simple ou complexe, à laquelle elle donne la valeur Vrai (**TRUE** ou **1**) ou la valeur Faux (**FALS**E, ou 0).

Exemple

```
int Led = 13;
void setup()
{
   pinMode(Led, OUTPUT);
}
int tempo = 1000;
void loop()
{
    tempo= tempo - 100;
    if(tempo <= 0){
      tempo = 1000;
    }
    digitalWrite(Led, HIGH);
    delay(tempo);
    digitalWrite(Led, LOW);
    delay(tempo);
}
```

II.2 Structure d'itération FOR

Il est parfois utile d'exécuter un traitement un grand nombre de fois. On veut par exemple parcourir un tableau - pour en faire la moyenne -, ou faire clignoter un nombre défini de fois une LED, ..etc.

Exemple

```
int Led = 9;
void setup()
{
  pinMode(Led, OUTPUT);
}
void loop()
{
  for(int i = 0; i < 4; i++){
    digitalWrite(Led, HIGH);
    delay(200);
    digitalWrite(Led, LOW);
    delay(200);
  }
  delay(1000); // 1 second
}
```

II.3 Structure d'itération WHILE

Dans ce cas, la condition d'arrêt dépend d'un événement qui inervient durant le processus de bouclage. On attend par exemple une entrée au clavier, ou un signal Bluetooth, ou par exemple une valeur située dans un intervalle. Quand un WHILE démarre, on ne sait pas à l'avance quand - et si - il va s'arrêter.

Exemple

On attend l'activation d'un bouton poussoir. Tant que celui-ci n'est pas activé, on fait clignoter une LED signalant un mauvais fonctionnement. Une fois la pression effectuée, la LED est en position éteinte, et on allume une LED signalant le rétablissement des fonctions du système.

```
int LED1 = 3, LED2 = 4,button=5;
void setup()
{
  pinMode( LED1 , OUTPUT);
  pinMode( LED2 , OUTPUT);
  pinMode( button , INPUT);
}
void loop()
{
int etat = LOW ; // on suppose le bouton inactivé
while (etat == LOW)
{
digitalWrite(LED1, HIGH) ; // on allume la LED d'alarme
delay(500) ; // on attend une demi-seconde
```

```
digitalWrite(LED1, LOW) ; // on éteint la LED d'alarme
delay(500) ; // on attend une demi-seconde
state = digitalRead(button) ;
}
digitalWrite(LED2, HIGH) ; // on allume la LED indiquant un bon fonctionnement
}
```

III. Les tableaux

Un tableau est un ensemble de variables qui sont indexées par un numéro d'index.

Exemple

```
int Leds[4] = {2,3,4,5}; // LED connectés aux broches 2-5
void setup()
 {
 for(int i = 0; i < 4; i++){
 pinMode(Leds[i], OUTPUT);
 }
 }
 void loop()
 {
 for(int i = 0; i < 4; i++){
 digitalWrite(Leds[i], HIGH);
 delay(100);
 }
 for(int i = 4 - 1; i >= 0; i--){
digitalWrite(Leds[i], LOW);
 delay(100);
 }
 }
```

IV. Les entrées/sorties numériques

Dans cette section, nous allons apprendre à lire l'état d'une entrée numérique. Il faut savoir qu'une entrée numérique ne peut prendre que deux états, HAUT (HIGH) ou BAS (LOW). L'état haut correspond à une tension de +5V sur la broche, tandis que l'état bas est une tension de 0V.

Exemple

Dans notre exemple, nous allons utiliser un simple bouton. Dans la réalité, on peut utiliser n'importe quel capteur qui possède une sortie numérique. Il faut utiliser la fonction *digitalRead*() pour lire l'état logique d'une entrée logique. Cette fonction prend un paramètre qui est la broche à tester et elle retourne une variable

de type int.

```
int bouton = 2;
int led = 13;
int etat; //variable qui capture l'état du bouton
void setup()
{
   pinMode(led, OUTPUT);
   pinMode(bouton, INPUT);
  }
void loop()
{
   etat = digitalRead(bouton);
   if(etat == HIGH) //test si le bouton a un niveau logique HAUT
          digitalWrite(led,HIGH);
   else
          digitalWrite(led,LOW);
  }
```

V. Les entrées/sorties analogiques

V.1 Les entrées analogiques (analogRead())

La fonction analogRead() permet de lire la valeur de la tension présente sur la broche spécifiée. La carte Arduino comporte 6 voies connectées à un convertisseur analogique-numérique 10 bits. Cela signifie qu'il est possible de transformer la tension d'entrée entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. Il en résulte une résolution (écart entre 2 mesures) de : 5 volts / 1024 intervalles, autrement dit une précision de 0.0049 volts (4.9 mV) par intervalle.

Exemple

Lire la valeur de la tension d'une résistance variable

```
int analogPin = 0;
// une résistance variable connectée sur la broche
//  analogique  0

int val = 0; // variable de type int pour stocker la valeur de la mesure
void setup()
{
  Serial.begin(9600);          //  initialisation de la connexion série
}
void loop()
{
```

```
   val = analogRead(analogPin);
   // affiche la valeur (comprise en 0 et 1023) dans la fenêtre terminal PC
   Serial.println(val);
}
```

V.2 Les sorties analogiques (analogWrite())

Génère une impulsion de largeur / période voulue sur une broche de la carte Arduino (PWM - Pulse Width Modulation). Ceci peut-être utilisé pour faire briller une LED avec une luminosité variable ou contrôler un moteur à des vitesses variables.

Après avoir appelé l'instruction analogWrite(), la broche générera une onde carrée stable avec un "duty cycle" (fraction de la période où la broche est au niveau haut) de longueur spécifiée (en %), jusqu'à l'appel suivant de l'instruction analogWrite() (ou bien encore l'appel d'une instruction digitalRead() ou digitalWrite() sur la même broche). La fréquence de l'onde PWM est approximativement de 490 Hz (soit 490 périodes par seconde).

Exemple

Fixer la luminosité d'une LED proportionnellement à la valeur de la tension lue depuis un potentiomètre.

```
int ledPin = 9;
int analogPin = 0;
int val = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()
{
val = analogRead(analogPin);
analogWrite(ledPin, val/4);
/* Résultat d'analogRead entre 0 to 1023,
  résultat d'analogWrite entre 0 to 255
   => division par 4 pour adaptation   */
}
```

Remarque: L'instruction analogWrite n'écrit pas à proprement parler une valeur analogique en tension mise sur une broche, mais permet de générer des phénomènes de variation "d'allure analogique" en jouant sur la largeur de l'impulsion générée. Pour générer une véritable tension analogique à partir d'une commande numérique, il faut utiliser un circuit intégré externe de conversion numérique analogique (DAC en anglais pour Digital to Analogic converter) qui sera commandé par plusieurs broches numériques.

IV. La liaison série

la liaison série est un moyen de communication utilisé pour faire communiquer entre eux plusieurs

dispositifs. On retrouve cette liaison sur les ordinateurs, par exemple, ou sur des appareils électroniques (onduleurs, ...). Cette liaison est aussi utilisée dans le milieu industriel.

La librairie Serial est utilisée pour les communications par le port série entre la carte Arduino et un ordinateur ou d'autres composants. Toutes les cartes Arduino ont au moins un port Série (également désigné sous le nom de UART ou USART) : Serial. Ce port série communique sur les broches 0 (RX) et 1 (TX) avec l'ordinateur via le port USB. C'est pourquoi, si on utilise cette fonctionnalité, on ne peut pas utiliser les broches 0 et 1 en tant qu'entrées ou sorties numériques.

On peut utiliser le terminal série intégré à l'environnement Arduino pour communiquer avec une carte Arduino. Il suffit pour cela de cliquer sur le bouton du moniteur série dans la barre d'outils puis de sélectionner le même débit de communication que celui utilisé dans l'appel de la fonction begin().

Remarque: Lors d'une communication série, une vitesse s'exprime en bits par seconde ou **bauds**. Ainsi, pour une vitesse de 9600 bauds on enverra jusqu'à 9600 '0' ou '1' en une seule seconde. Les vitesses les plus courantes sont 9600, 19200 et 115200 bits par seconde.

IV. 1 Le setup

Dans le but de créer une communication entre votre ordinateur et votre carte Arduino, il faut déclarer cette nouvelle communication et définir la vitesse à laquelle ces deux dispositifs vont communiquer. Si la vitesse est différente, l'Arduino ne comprendra pas ce que veut lui transmettre l'ordinateur et vice versa ! Ce réglage va donc se faire dans la fonction setup, en utilisant la fonction **`begin()`** de l'objet Serial :

```
void setup()
{
  Serial.begin(9600); //on démarre la liaison en la réglant à une vitesse de
 }                    // 9600 bits par seconde.
```

IV. 2 print() et println()

La fonction que l'on va utiliser pour débuter, s'agit de print() et de son acolyte println(). Ces deux fonctions sont quasiment identiques, mais à quoi servent-elles ?

- print() : cette fonction permet d'envoyer des données sur la liaison série. On peut par exemple envoyer un caractère, une chaine de caractère ou d'autres données dont je ne vous ai pas encore parlé.
- println() : c'est la même fonction que la précédente, elle permet simplement un retour à la ligne à la fin du message envoyé.

```
void setup()
{
   Serial.begin(9600); //établissement d'une nouvelle communication série
   Serial.print("Salut !"); //envoie de la chaine "Salut  !" sur la liaison }
```

V. Les fonctions

Le cas typique pour la création des fonctions est quand on a besoin d'effectuer la même action plusieurs fois dans un programme.

Exemple 1

```
void setup(){
    Serial.begin(9600);
```

```
            }
void loop() {
    int i = 2;
    int j = 3;
    int k;
   k = myMultiplyFunction(i, j); //
    Serial.println(k);
    delay(500);
}
int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

Exemple 2

Cette fonction va lire un capteur cinq fois avec analogRead(), calculer la moyenne des cinq lectures, mettre la donnée sur 8 bits (0-255), et enfin retourner le résultat final.

```
int ReadSens(){
  int i;
  int sval = 0;

  for (i = 0; i < 5; i++){
    sval = sval + analogRead(0);
  }

  sval = sval / 5;    // moyenne
  sval = sval / 4;    // 8 bits (0 - 255)
  return sval;
}
```

Pour appeler cette fonction nous il suffit de l'attribuer à une variable.

```
int sens;
sens = ReadSens();
```