Programmation ARDUINO

STRUCTURE D'UN PROGRAMME ARDUINO

La structure de base du langage de programmation Arduino est assez simple et comprend au moins deux parties :

- `setup()` est la préparation et l'initialisation du programme
- `loop()` est l'exécution du programme.

Ces deux parties (ou fonctions), contiennent des blocs d'instructions et sont impérativement requises pour que le programme fonctionne.

```
void setup()
{
        blocs d'instructions;
}
void loop()
{
        blocs d'instructions;
}
```

`setup()` :Initialisation du programme

La fonction `setup()` doit suivre la déclaration des variables au tout début du programme. Il s'agit de la première fonction à exécuter dans le programme. Elle n'est appelée qu'une fois au démarrage du programme. Elle sert à initialiser le mode des broches (ex : `pinMode`) ou fixer le débit de la communication série (ex : `Serial.begin`). Elle doit être dans tous les programmes même si il n'y a aucune instruction à exécuter.

```
void setup()
{
        pinMode(broche, OUTPUT); //met la "broche" comme sortie
}
```

`loop()` :Boucle du programme

La fonction `loop()` suit la fonction `void setup()`. La fonction `void loop()` fait une boucle (loop en anglais) infinie permettant de contrôler la carte arduino tant que le programme ne change pas. Elle comprend un code à exécuter en continue - lisant les capteurs en entrée et déclenchant les actionneurs en sortie, etc. Cette fonction est le noyau de tout programme Arduino et réalise l'essentiel du travail.

```
void loop()
{
        digitalWrite(broche, HIGH);  //met la broche en "ON"
        delay(1000);                        //pause pendant une seconde
        digitalWrite(broche, LOW);   //met la broche en "OFF"
        delay(1000);                        //pause pendant une seconde
}
```

Les fonctions : blocs de code particuliers

Une fonction est un bloc de code possédant un nom et exécutant ses instructions à chaque appel. Les fonctions `void setup()` et `void loop()` sont les 2 fonctions obligatoires en Arduino, mais d'autres peuvent être créées.

Il est possible de créer des fonctions afin d'améliorer des programmes répétitifs et pour rendre plus compréhensif un programme. On commence par donner un type aux fonctions. Il s'agit du type de valeur qui sera renvoyée par la fonction, par exemple un "int" donnera une fonction de type entier (`integer`). Si aucune valeur n'est renvoyée, le type est `void` (ex : `void setup()/void loop()`). Après le type, on attribue un nom à la fonction puis, dans les parenthèses, on donne les paramètres passés à la fonction.

```
type nomFonction(paramètres)
{
        instructions;
}
```

La fonction entière (`int`) suivante : `delayVal()`, va permettre de changer l'intervalle de la valeur lue sur un potentiomètre. On commence par déclarer une variable locale (`v`), elle contient la valeur du potentiomètre, c'est à dire entre 0 et 1023. On divise cette valeur par 4 afin de passer à une valeur comprise entre 0 et 255 puis on renvoie cette valeur au programme.

```
int delayVal()
{
        int v;                      //création de la variable locale "v"
        v = analogRead(pot);        //lecture de la valeur du potentiomètre
        v /= 4;                     //conversion de 0-1023 à 0-255
        return v;                   //renvoie de la valeur finale
}
```

Les accolades : {}

Les accolades (ouvrantes ou fermantes) définissent le début et la fin d'une fonction ou d'un bloc d'instructions, comme pour la fonction `void loop()` ou les instructions `if` et `for`.

```
type fonction()
{
        instructions;
}
```

Une accolade ouvrante « { » est toujours suivie par une accolade fermante «} ». Une accolade non fermée donne des erreurs. L'environnement arduino inclue un système de vérification des accolades. Il suffit de cliquer sur une accolade pour voir sa "paire" se surligner.

Le point-virgule : ;

Le point-virgule est utilisé pour finir une instruction et séparer les éléments d'un programme. On l'utilise, par exemple, pour séparer les éléments de la boucle for.

```
int x = 13;     //on déclare la variable x comme un entier valant 13
```

Note : Oublier le point-virgule à la fin d'une ligne donne une erreur de compilation. Le message d'erreur peut parfois faire référence au point-virgule oublié, mais pas toujours. Si le message d'erreur est illogique et incompréhensible, une des premières choses à faire est de vérifier s'il n'y a pas un point-virgule manquant.

Les paragraphes de commentaire/*...*/

Un commentaire (sur une ligne ou sur plusieurs) est une zone de texte qui est ignoré par le programme lors de la compilation et qui est utilisé afin de décrire le code ou pour le commenter pour faciliter sa compréhension auprès de personnes tiers. Un commentaire commence par /* et fini avec */ et peut faire plusieurs ligne.

```
/*  ceci est un paragraphe de commentaire
commencé ne pas oublier de le refermer */
```

De par le fait que les commentaires soient ignorés et qu'ils ne prennent pas d'espace mémoire, ils peuvent être utilisés généreusement, par exemple pour débugger une partie du code.

Note : Il est possible de faire un paragraphe d'une seule ligne avec /* */ mais on ne peut pas inclure un second bloc dans le premier.

Les lignes de commentaires : //

Une seule ligne de commentaire commence par // et fini à la fin de la ligne de code. Tout comme le paragraphe de commentaires, la ligne est ignorée lors de la compilation et ne prend pas de mémoire.

```
//ceci est une ligne de commentaires
//ceci est une autre ligne de commentaires
```

La ligne de commentaire est souvent utilisée pour donner des informations ou pour rappeler quelque chose à propos d'une ligne.

Remarque :

VARIABLES

Les variables

Une variable est une façon de nommer et de stocker une information/valeur afin de l'utiliser plus tard dans le programme. Comme son nom l'indique, une variable est une valeur qui peut être continuellement modifiée à l'opposé d'une constante qui ne change jamais de valeur. Une variable a besoin d'être déclarée et on peut lui assigner une valeur que l'on veut stocker.
Le code suivant déclare une variable nommé `inputVariable` et lui assigne une valeur lue via `analogRead` sur la broche 2.

```
int inputVariable = 0;   //déclare la variable et lui assigne la valeur 0

inputVariable = analogRead(2);   //assigne la valeur de la broche analogique 2 à la variable
```

"`inputVariable`" est une variable. La première ligne donne le type de la valeur contenue : `int` (diminutif de integer, entier). La seconde ligne assigne la valeur de la broche analogique 2 à cette variable. Cela permet de rendre accessible dans le code la valeur de la broche analogique 2.
Une fois que la variable a été assignée ou réassignée, il est possible de la tester pour voir si elle remplit certaines conditions ou de l'utiliser directement. Afin d'illustrer 3 opérations habituelles sur les variables, le code suivant vérifie si `inputVariable` est plus petit que 100, si oui, elle lui assigne la valeur 100 et le programme marque une pause de `inputVariable` millisecondes, au minimum 100 donc.

```
if (inputVariable < 100)      //on vérifie si la variable est plus petite que 100
{
   inputVariable = 100;       //si oui, on lui attribue la valeur 100
}
delay(inputVariable);         //on utilise la variable comme temps de pause (fonction delay())
```

Note : Pour rendre le code plus lisible, on peut donner aux variables un nom explicatif. Les noms de variables tels que `boutonAppuie (pushButton)` ou `mouvementCapteur (tiltSensor)` aident le programmateur ainsi que les personnes lisant le code afin de savoir ce que représentent les variables. Les noms de variables tels que `var` (pour variable) ou `value` (valeur) ne rendent pas vraiment le code plus lisible et ne sont utilisés que dans les exemples. Une variable peut avoir n'importe quel mot comme nom tant que celui-ci n'est pas un mot clé du langage arduino.

La déclaration des variables

Toutes les variables doivent être déclarées avant d'être utilisées. La déclaration d'une variable définie son type, tel que `int`, `long`, `float`, ... lui attribue un nom spécifique et peut lui assigner une valeur initiale. On peut modifier la valeur tout au long du programme par des assignations ou des opérations arithmétiques.
L'exemple suivant déclare `inputVariable` comme un `int`, ou type entier (integer) et lui assigne 0 comme valeur initiale.

```
int inputVariable = 0;
```

Visibilité des variables pour le programme

Une variable peut être déclarée au début du programme avant l'initialisation (`void setup()`), à l'intérieur de fonctions et parfois au sein d'un bloc d'instruction comme les boucles. La zone où la variable est déclarée détermine la "visibilité" de celle-ci, autrement dit la possibilité pour certaines parties d'un programme à utiliser cette variable.

Une variable globale est une variable qui peut être vue et utilisée par l'ensemble du programme. Cette variable doit être déclarée au début du programme, avant la fonction `void setup()`.
Une variable locale est une variable définie à l'intérieur d'une fonction ou dans une partie d'une boucle for. Elle est uniquement visible et utilisable à l'intérieur de la fonction où elle a été déclarée. Il est de ce fait possible d'avoir 2 variables ou plus ayant le même nom dans différentes parties du programme avec des valeurs différentes. Permettre à une seule fonction d'avoir accès à ses variables permet de réduire les erreurs potentielles de programmation.

L'exemple suivant montre comment déclarer les différents types de variables et expliquent leur visibilité :

```
int value;           //"value" est visible par toutes les fonctions

void setup()
{
   //no setup needed
}

void loop()
{
  for (int i=0 ; i<20 ;)   //"i" est uniquement visible à l'intérieur de la boucle for
  {
     i++;
  }
  float f;   //"f" est uniquement visible dans la fonction void loop()
}
```

TYPE DE DONNEES

byte

`Byte` permet de stocker une valeur entière sur 8 bits (pas de chiffre à virgule donc). Cela permet d'aller de 0 à 255.

```
byte maVariable = 180;    //déclare "maVariable" en tant que byte
```

int

`Int` (pour integer, entier) est le type de base pour les chiffres (sans virgule) des variables. Le stockage se fait sur 16 bits, permettant d'aller de -32 768 à 32 767.

```
int maVariable = 1500;  //déclare "maVariable" en tant que int
```

Note : Si la valeur maximale ou minimale est passée pour le type `int`, sa valeur sera reportée. C'est à dire, si x = 32767 et que l'on y ajoute 1, x = x + 1 ou x ++, x vaudra alors - 32768.

long

Il s'agit d'un type de variable de grande taille : la donnée (chiffre sans virgule) est stockée sur 32 bits. Cela permet d'aller de 2 147 483 647 à - 2 147 483 648.

```
long maVariable = 90000;   //déclare "maVariable" en tant que long
```

unsigned

Il ne s'agit pas d'un type de variable à proprement dit mais d'un ajout permettant de supprimer les chiffres négatifs (on ne garde que les chiffres non-signés, les positifs). Cela permet d'avoir plus de place pour stocker les chiffres positifs. Par exemple :

- au lieu d'aller de 32 767 à - 32 768, un `int` couvrira l'intervalle de 0 à 65 535
- au lieu d'aller de 2 147 483 647 à - 2 147 483 648, un long couvrira l'intervalle de 0 à 4 294 967 295

float

Ce type `float` permet de stocker les chiffres "flottant" ou chiffre à virgule. Les float bénéfice d'une meilleure résolution que les entiers (int) : ils sont stockés sur 32 bits, soit de 3,4028235^38 à - 3,4028235^38.

```
float maVariable = 3,14;   //déclare "maVariable" en tant que chiffre flottant
```

Note : Les chiffres flottants ne sont pas exacts et il arrive que des résultats étranges ressortent de leur comparaison. De plus, réaliser des opérations mathématiques dessus est plus lents que sur des entiers, il est conseillé d'éviter si possible.

bool

Le type `bool` prend 2 valeurs : Vrai (`true`) ou Faux (`false`). Il est stocké sur 8 bits et se comporte ainsi :

- Si la valeur est 0, il sera vu comme Faux (False)
- Si la valeur est 1 ou plus, il sera vu comme Vrai (True)

```
bool maVariable = false;  //la valeur est 0 donc false (faux)
```

char

Le type `char` est utilisé pour stocker une lettre. Le stockage se fait sur 8 bits et permet de stocker une seule lettre.

```
char maVariable = 'A';    //déclare maVariable en tant que lettre
```

Note : Pour assigner une valeur à la variable char, on l'écrira entre '.

```
unsigned int maVariable = 65534;  //la variable est un entier non-signé valant 65534
```

arrays

Un tableau est une collection de valeur dont on accède via un index. Afin d'avoir accès aux valeurs contenues dans le tableau, il faut appeler le nom du tableau et le numéro d'index de la valeur. L'index 0 est la première valeur stockée. Un tableau, comme une variable, a besoin d'être déclaré mais l'assignation est facultative avant son utilisation.

```
int monTableau[] = {valeur0, valeur1, valeur2, ... };
```

Il est également possible de déclarer le tableau uniquement en lui donnant un type et une taille et d'assigner plus tard des valeurs aux index.

```
int monTableau[5];       //déclaration d'un tableau d'entier avec 6 cases
monTableau[3] = 10;     //on assigne à la 4e case la valeur 10
```

Pour sortir une valeur d'un tableau, on assigne à une case du tableau une variable :

```
x = monTableau[3];      //x vaut maintenant 10
```

Les tableaux sont souvent utilisés dans la boucle `for` où le compteur est utilisé pour choisir la case du tableau et atteindre la valeur. L'exemple suivant utilise un tableau pour modifier l'éclairage d'une LED. Le compteur, dans la boucle `for`, commence à 0, écrit la valeur contenu dans la case 0 du tableau `eclairage[]`, c'est à dire 180, à la broche 10. Puis marque une pause de 250ms et continue à la case suivante.

```
int brocheLED = 10;      //la LED est sur la broche 10
byte eclairage[] = {180, 30, 255, 60};
               //tableau avec 4 cases contenant des valeurs différentes

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

void loop ()
{
  for(int i=0;i<3;i++)                    //chaque valeur du tableau
  {                                       //est envoyée à la broche où
    analogWrite(brocheLED, eclairage[i]); //est reliée la LED
    delay(250);                           //avant de faire une pause de 250 millisecondes
  }
}
```

String

Il ne s'agit pas exactement d'un type de variable mais d'une classe. La classe `String` est comparable à une variable possédant ses propres fonctions. `String` permet de stocker une chaîne de caractères comme pourrait le faire un tableau de caractère.

```
String maVariable = "Aux sciences, citoyens";    //maVariable contient la chaîne de caractères
```

`String` permet de réaliser des opérations sur les chaînes de caractères plus complexes que ce qui se ferait sur un tableau de caractères. Dans l'exemple qui suit, on va passer un nombre en String, compter le nombre de caractères qui le compose et réaliser la comparaison avec un second String.

```
String chaineDeNombre = String(245);  //chaineDeNombre contient "245"

int nombreCaractere = chaineDeNombre.length();   //nombreCaractère vaut 3

int compare = maVariable.compareTo(chaineDeNombre);
//compare vaudra : un nombre négatif si maVariable vient avant chaineDeNombre, 0 si maVariable
//et chaineDeNombre sont égaux ou un nombre positif si maVariable vient après chaineDeNombre
```

Quand on parle de "venir avant", cela est lié à la place de chaque caractère en informatique : les chiffres viennent avant les lettres et les majuscules avant les minuscules.

Note : Les String fonctionnent en allocation dynamique de mémoire : la taille maximale n'est pas définie à la création, il faut donc faire attention à ne pas dépasser la taille maximum de mémoire autorisé par une carte/par une machine.

Déclaration de chaînes de caractères.

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Autres possibilités de déclaration. :

```
char myString[] = "This is the first line"
"this is the second line"
"etcetera";
```

Exemple :

```
char* myStrings[]={"This is string 1", "This is string 2", "This is string 3",
"This is string 4", "This is string 5","This is string 6"};

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

void loop()
{
  for (int i = 0; i < 6; i++)
  {
    Serial.println(myStrings[i]);
    delay(500);
  }
}
```

L'ARITHMETIQUE

Les opérations d'arithmétique incluent les additions, les soustractions, les multiplications et les divisions. Elles retournent respectivement la somme, la différence, le produit et le quotient des 2 opérants.

```
y = y + 3;
x = x - 7;
r = r / 5;
i = j * 6;
```

L'opération est réalisée en utilisant les types de données des opérants, de façon à ce que, par exemple, 9 / 4 donnera 2 plutôt que 2,25 car 9 et 4 sont des entiers (`int`), or le type "int" ne peut être un chiffre à virgule. Cela veut aussi dire que le résultat peut être trop grand pour le type de variable initiale.
Si les opérants sont de différents types, le type le plus grand est utilisé pour le calcul. Par exemple, si un des nombres (opérant) est de type "`float`" et l'autre de type "`int`", c'est le type "`float`" qui sera utilisé lors de l'opération.
Il faut donc faire bien attention lors du choix des types de variables afin que le résultat ne soit pas trop grand lors du calcul. Il faut donc savoir jusqu'à combien (aussi bien dans les positifs que dans les négatifs) la variable peut aller, par exemple 0 ou 1 ou de 0 à - 32768. Pour réaliser des opérations sur des fractions, on utilisera des "`float`", l'inconvénient étant que des grandes variables entrainent des calculs lents.

Note : Pour passer d'un type de variable à un autre, on utilisera l'opérateur de cast, c'est à dire `(int)monFloat` convertira `monFloat` en `int`. Par exemple :

```
i = (int)3,6 // Fixe i à la valeur 3
```

Opérateurs d'incrémentation

Les opérateurs d'incrémentation permettent de réaliser une opération simple, ils sont communément utilisés dans la boucle for, voir plus bas. Il en existe 2 : l'incrémentation et la décrémentation.

```
x ++    //équivaut à x = x + 1 ou d'incrémenter x par 1
x --     //équivaut  à x = x - 1 ou de décrémenter x par 1
```

Opérateurs d'assignation

Les opérateurs d'assignations sont la combinaison entre une assignation et une opération, de la même sorte que les opérateurs d'incrémentation.

```
x += y    //équivaut à x = x + y ou d'incrémenter x par y
x -= y    //équivaut à x = x - y ou de décrémenter x par y
x *= y    //équivaut à x = x * y ou de multiplier x par y
x /= y    //équivaut à x = x / y ou de diviser x par y
```

Note : Par exemple, x *= 3 va multiplier l'ancienne valeur de x par 3 et réassigner la nouvelle valeur à x.

Opérateur de comparaison

La comparaison d'une variable (ou d'une constante) avec une autre est souvent utilisée afin de tester si une condition spécifique est vraie ou non. Dans les exemples suivants, l'opérateur est utilisé pour indiquer une condition :

```
x == y  //x est égal à y
x != y  //x n'est pas égal à y
x < y   //x est plus petit que y
x > y   //x est plus grand que y
x <= y   //x est plus petit ou égal à y
x >= y   //x est plus grand ou égal à y
```

Opérateurs logiques

Les opérateurs logiques sont utilisés pour comparer deux expressions/conditions. Ils renvoient `TRUE` (vrai) ou `FALSE` (faux) selon l'opération. Il y a trois opérateurs logiques : AND (et), OR (ou) et NOT (contraire), ils sont souvent utilisés dans les structures `if`.

AND :

```
if (x > 0 && x < 5)       //vrai seulement si les deux conditions sont vraies
```

OR :

```
if (x > 0 || y > 0)       //vrai si une des condition est vraie
```

NOT :

```
if (!x > 0)               //vrai seulement si la condition est fausse
```

CONSTANTES

Le langage arduino possède des valeurs prédéfinies; elles sont appelée constantes. Elles sont utilisées pour rendre un programme plus simple à lire. Il en existe différent groupe.

Vrai/faux (TRUE/FALSE)

Il s'agit de variable du type booléen.

- FALSE signifie que la constante est à 0.
- TRUE désigne l'état différent de 0 : cela peut être 1 (par convention) mais cela peut aussi être -1, 2, -200, ...

```
if (b == FALSE)
{
   Faire;
}
```

Haut/bas (HIGH/LOW)

Il s'agit des constantes utilisées pour lire ou pour écrire sur les broches, soit le niveau HIGH (ON, 1, +5 volts) soit le niveau LOW (OFF, 0, 0 volts).

```
digitalWrite(13,HIGH);
```

Entrée/sortie (INPUT/OUTPUT)

Ces constantes sont utilisées avec la fonction `pinMode()`, elles définissent la nature d'une broche digitale :

- INPUT pour entrée
- OUTPUT pour sortie

```
pinMode(13, OUTPUT);
```

Remarque :

STRUCTURES DE CONTROLE

if

La structure de test "`if`" ou (si), vérifie si une certaine condition est atteinte. Par exemple, si une variable est égale à une certaine valeur, elle exécute des instructions contenues dans la structure {}. Si la condition est fausse, le programme quitte la structure et reprend à la ligne suivante.

```
if (maVariable == valeur)
{
   Faire;
}
```

Note : Attention à l'utilisation du "=" tel que

```
if( x = 10 )
```

, qui est grammaticalement valide, cela assigne la valeur 10 à la variable x et le résultat est toujours vraie. Cependant, il faut utiliser "==" tel que

```
if( x == 10 )
```

afin de tester si x est égal à 10. Moyen mémo-technique : "=" est "égale" alors que "==" est "est-il égale à".

if... else

`if... else` (si...sinon) est utilisé pour dire "sinon fait...". Par exemple, si on veut tester une entrée digitale et faire une action si elle vaut HIGH et d'autres actions si elle vaut LOW, on écrira :

```
if (inputPin == HIGH)
{
   Faire_A;
}
else
{
   Faire_B;
}
```

`else` peut précéder un autre test avec un "`if`", on peut ainsi en mettre une multitude qui seront exécutés les uns après les autres. Il est aussi possible d'avoir un nombre infini de branche "`else`". Cependant, il faut garder à l'esprit qu'une seule partie des instructions sera exécutée, selon les conditions vérifiées.

```
if (inputPin < 500)
{
  Faire_A;
}
else if (inputPin >= 1000)
{
  Faire_B;
}
else
{
  Faire_C;
}
```

Note : Un "`if`" test si la condition entre parenthèse est vraie ou fausse. Ainsi, le premier exemple ne vérifiera que si la variable vaut HIGH (ou +5V) mais pas il n'est pas possible d'avoir un autre état différent de vrai ou faux.

switch... case

La structure `switch... case` (commutateur de cas) permet de dresser une lister de test ou de cas (comme le "if") et d'exécuté le code correspondant si un des cas du test est vrai. La principale différence entre le `switch... case` et le `if... else` est que le `switch... case` permet de continuer les tests même si un des cas est vrai, contrairement au `if...else` qui quittera dans le même cas.

```
switch (maVariable)
{
  case A : Faire_A;
  case B : Faire_B;
}
```

`maVariable` est comparée à `A`, si le test est vrai, `Faire_A` sera exécuté. Une fois le code exécuté (ou si le test est faux), `maVariable` sera comparé à B et ainsi de suite. Une fois le code `Faire_x` exécuté, si l'on veut s'arrêter, on ajoute l'instruction `break`. De plus, si aucun des cas n'est vérifié, on peut ajouter le cas `default` qui sera exécuté si aucun test n'est vrai.

```
switch (maVariable)
{
  case A: Faire_A;    //si maVariable vaut A, Faire_A est exécuté
  break;                   //une fois le code exécuté, on sort de la structure
  case B: Faire_B;    //en l'absence de "break", si maVariable est égale à B puis à C
  case C: Faire_C;       // Faire_B et Faire_C seront exécutés

 default: Faire_Par_Default  //dans le cas où aucun cas ne serait vérifié,
                              //Faire_Par_Default sera exécuté puis on sortira de la structure
}
```

Note : Il n'est pas nécessaire que les cas soit dans l'ordre chronologique ou du plus petit au plus grand.

for

La structure `for` (pour) est utilisée afin de répéter une série d'instructions (comprise entre les accolades) le nombre de fois voulu. Un compteur à incrémenter est souvent utilisé afin de sortir de la boucle une fois le nombre de tour fait. Cette structure est composée de trois parties séparées par un point-virgule (;) dans l'entête de la boucle :

```
for (initialisation; condition; expression d'incrémentation)
{
  Faire;
}
```

L'initialisation d'une variable locale, ou compteur d'incrémentation, permet d'avoir une variable ne servant qu'une fois dans le programme. A chaque tour de boucle, la condition de l'entête est testée. Si cette condition est vraie, « Faire » est exécuté ainsi que l'expression d'incrémentation et la condition est de nouveau testée. Quand la condition devient fausse, la boucle prend fin.

L'exemple suivant commande avec un entier i à 0, on vérifie s'il est inférieur à 20 et si c'est vrai, on l'incrémente de 1 et on exécute le programme contenu dans les accolades :

```
for (int i = 0; i<20; i++)
{
  digitalWrite(13, HIGH);
  delay(250);
  digitalWrite(13,LOW);
  delay(250);
}
```

Note : Dans le langage C (d'où découle arduino), la boucle for est beaucoup plus flexible que dans les autres langages (par exemple le Basic). Certains (voire tous) éléments parmi les 3 de l'entête peuvent être oublié, seuls les points-virgules sont indispensables. Chacune des expressions (initialisation, condition et incrémentation) peut être une expression en C avec des variables différentes. Ce genre de cas rares peut être la solution pour certains problèmes de programmations également rares.

while

La boucle "`while`" (tant que) continuera infiniment tant que l'expression entre parenthèse est vraie. Quelque chose doit changer afin de modifier la condition sinon le programme ne sortira jamais de la boucle `while`. Cela peut être dans le code, comme l'incrémentation d'une variable, ou une condition extérieur, comme le test d'un capteur.

```
while (maVariable ?? valeur)
{
  Faire;
}
```

L'exemple suivant test si "maVariable" est inférieur à 200 et si c'est vrai, le code entre les accolades est exécuté et la boucle continue tant que "maVariable" est inférieur à 200.

```
while (maVariable < 200)          //test si "maVariable" est inférieur à 200
{
  Faire;                  //exécution du code
  maVariable++                    //incrémentation de "maVariable" par 1
}
```

do... while

La boucle do... while (faire... tant que) fonctionne de la même façon que la boucle while, à la seule exception que la condition est testée à la fin de la boucle, de ce fait le code dans la boucle s'exécute toujours au moins une fois.

```
do
{
  Faire;
} while (maVariable ?? valeur);
```

L'exemple suivant assigne à la variable "x" la valeur lireCapteur() (`readSensors()`), puis fait une pause de 50 millisecondes et continue infiniment tant que x est plus petit que 100.

```
do
{
  x = lireCapteur();              //assignation de la valeur de lireCapteur à x
  delay(50);                      //pause durant 50 millisecondes
} while (x < 100);                //on recommence la boucle si x est inférieur à 100
```

Remarque :

ENTREES ET SORTIES NUMERIQUES (DIGITAL I/O)

pinMode(pin, mode)

Utilisée dans le `void setup()`, cette fonction permet de configurer une broche soit en entrée (INPUT) soit en sortie (OUTPUT).

```
pinMode(broche, OUTPUT);  //fixe la broche en sortie
```

Les broches « digital » d'Arduino sont par défaut configurées en entrée, il n'est pas nécessaire de les déclarer explicitement comme telle avec `pinMode()`. Les broches configurées en entrée, sont dans un état de haute impédance.
Il y a également une résistance pull up de 20kΩ installée à l'intérieur de la puce Atmega (puce programmable de la carte Arduino). Pour y accéder, on utilisera :

```
pinMode(broche, INPUT);        //fixe la broche en entrée
digitalWrite(broche, HIGH);     //active la résistance pull up
```

Les résistances pull up sont normalement utilisées pour connecter ensembles des entrées, comme un commutateur. A noter qu'il ne faut pas convertir la broche en sortie, car il s'agit de la méthode pour activer la résistance pull up interne.
Les broches configurées en sortie sont en état de basse impédance et fournissent du 40 mA (milliampère) aux autres circuits/dispositifs. C'est suffisant pour faire briller une LED (ne pas oublier les résistances de protection en série) mais c'est insuffisant pour les relais, solénoïdes ou les moteurs.

Les petits circuits sur les broches d'Arduino et les courants extrêmes peuvent endommager, voire détruire, les broches de sorties ou la puce Atmega entière. C'est pour cela que l'on ajoute souvent aux sorties des résistances en série (principalement 470Ω ou 1kΩ) intercalées entre elles et les composants externes.

digitalRead(pin)

Cette fonction permet de lire la valeur à la broche digitale indiquée. La valeur lue étant soit HIGH soit LOW. La broche ciblée ne peut être qu'une variable ou une constante comprise entre 0 et 13 (pour les modules Arduino UNO).

```
valeur = digitalRead(broche);    //assigne à "valeur" la valeur lue à la broche
```

digitalWrite(pin, value)

Il s'agit du contraire de `digitalRead()` : cette fonction permet de fixer une sortie digitale soit au niveau HIGH soit au niveau LOW. Cette sortie est comprise entre 0 et 13 sur les modules Arduino UNO et doit être spécifiée soit par une variable soit par une constante.

```
digitalWrite(broche, HIGH)    //fixe la sortie "broche" à HIGH
```

L'exemple suivant va lire la valeur sur une broche digitale à laquelle est relié un bouton. Lors de l'appuie sur le bouton, la valeur renvoyée sera HIGH alors qu'au repos ça sera LOW. Ainsi, il suffira d'affecter cette valeur à une LED afin de l'allumer quand le bouton est appuyé :

```
int led = 13;   //la led est reliée à la broche 13
int broche = 7;  //le bouton est relié à la broche 7
int valeur = 0;   //la variable où ça sera stockée la valeur lue

void setup()
{
  pinMode(led, OUTPUT);           //la broche 13 est une sortie
  pinMode(pin, INPUT);            //la broche 7 est une entrée
}

void loop()
{
  valeur = digitalRead(broche);   //assignation de la valeur lue à la variable "valeur"
  digitalWrite(led, valeur);      //on fixe à la LED la même valeur qu'au bouton
}
```

Remarque :

ENTREES ET SORTIES ANALOGIQUES (ANALOG I/O)

analogRead(pin)

Cette fonction permet d'aller lire la valeur sur une entrée analogique avec 10 bits de résolution. Elle ne fonctionne que sur les broches analogiques (0-5) du module Arduino UNO et renvoie un résultat compris entre 0 et 1023.

```
valeur = analogRead(broche);   //assigne à "valeur" ce qui est lue sur la broche
```

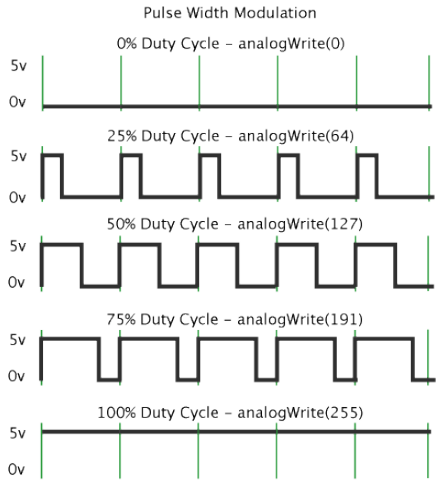
Note : Les broches analogiques ne sont pas des broches digitales, elle ne nécessite pas d'être déclarée comme des sorties ou des entrées.

analogWrite(pin, value)

`analogWrite` permet d'envoyer un pseudo-signal analogique en utilisant le matériel via les broches possédant un "pulse width modulation" (PWM ou modulation de largeur d'impulsions). Sur les modules Arduino les plus récents (équipé du contrôleur ATmega168), cette fonctionnalité se trouve sur les broches 3, 5, 6, 9, 10 et 11. Sur les plus anciennes (ATmega8), seules les broches 9, 10 et 11 étaient équipées. Ce signal est envoyé sous forme de valeur, comprise entre 0 et 255.

```
analogWrite(broche, valeur);  //envoie "valeur" sur la "broche" analogique
```

La valeur 0 va générer une tension stable de 0 volts en sortie; la valeur 255 générera une tension stable de 5 volts en sortie. Pour les valeurs comprises entre 0 et 255, la broche va rapidement alterner entre 0 et 5 V - c'est à dire la plus grande valeur possible, celle correspondant au HIGH. Par exemple, pour une valeur de 64, sera en fait composé de 0 volts durant les trois quarts du temps et de 5 volts pendant le quart restant. De même pour 128, la moitié du temps la valeur sera 0 volts et l'autre moitié à 5 volts. Et pour une valeur de 192, pendant un quart du temps, la valeur sera à 0 volts et pendant les trois quarts restant à 5 volts.



Etant donné que l'on utilise le matériel, le signal généré par une fonction `analogWrite` ne s'arrêtera que si on appelle une seconde fois `analogWrite` (ou `digitalRead` ou `digitalWrite` sur la même broche), autrement le signal restera en fond.

L'exemple suivant lit la valeur analogique sur une entrée analogique, divise la valeur par 4 et génère un signal PWM en sortie sur une broche PWM :

```
int led = 10;              //LED branchée à une résistance 220Ω sur la broche 10
int broche = 0;            //broche 0 où est reliée un potentiomètre
int valeur;                //variable où sera stockée la valeur lue sur le potentiomètre

void setup() {}            //pas de paramètre ici

void loop()
{
  valeur : analogRead(broche);    //on assigne à "valeur" le résultat lu sur "broche"
  valeur /= 4;                    //conversion de 0-1023 à 0-255
  analogWrite(led, valeur);       //envoi du signal en sortie PWM sur la broche "led"
}
```

GESTION DE LA DUREE

delay(ms)

delay permet de marquer une pause dans le programme, le temps est en millisecondes (1000 ms donnant 1 seconde).

```
delay(1000);   //pause d'une seconde
```

millis()

Cette fonction renvoie le nombre de millisecondes écoulé depuis que l'arduino a commencé à exécuté le programme actuel sous l'état d'un `unsigned long`.

```
valeur = millis();  //assigne à "valeur" le nombre de millisecondes écoulé
```

Note : Le nombre est remis à 0 (par dépassement de tailles de variables) au bout d'environ 9 heures.

Remarque :

MATH

min(x,y)

Calcule le plus nombre le plus petit entre x et y, quel que soit le type de variable utilisé et le renvoie.

```
valeur = min(valeur,100);

//assigne le nombre le plus petit entre "valeur" et 100 à la place de la valeur de départ contenu dans "valeur"
```

max(x,y)

Calcule le plus nombre le plus grand entre x et y, quel que soit le type de variable utilisé et le renvoie.

```
valeur = max(valeur,100);

//assigne le nombre le plus grand entre "valeur" et 100 à la place de la valeur de départ contenu dans "valeur"
```

HASARD

randomSeed(seed)

Cette fonction permet de fixer un point de départ comme base/graine pour la fonction `random()`.

```
randomSeed(valeur);  //affecte "valeur" comme base de l'aléatoire
```

La carte arduino est incapable de créer un vrai nombre aléatoire c'est pourquoi la fonction `randomSeed()` permet d'aider à la génération d'un nombre aléatoire via une variable, une constante ou une fonction servant de base. Il y a différent type de bases de fonctions pouvant servir de bases telles que `millis()` ou `analogRead()` permettant de lire le bruit statique sur la broche analogique.

random(min, max)

La fonction `random()` renvoie un nombre pseudo-aléatoire dans un intervalle défini par un minimum et un maximum.

```
valeur = random(100,200);   //affecte à "valeur" une valeur aléatoire entre 100 et 200
```

Note : La fonction `random()` est a utilisée après `randomSeed()`.

L'exemple suivant créé un nombre entre 0 et 255 et l'envoi ensuite sur une sortie PWM :

```
int nbrAleatoire;                 //variable pour stocker une valeur aléatoire
int led = 10;                     //LED branchée à une résistance 220Ω sur la broche 10

void setup() {}                   //pas de paramètre nécessaire

void loop()
{
  randomSeed(millis());           //affecte millis() comme base
  nbrAleatoire = random(255);     //génération d'un nombre aléatoire entre 0 et 255
  analogWrite(led, nbrAleatoire); //envoi du signal sur une broche PWM
  delay(500);                     //on marque une pause d'une demi-seconde
}
```

Remarque :

COMMUNICATION AVEC LE PORT SERIE

serial.begin(débit)

Permet le port série et donne la vitesse de transmission (ou débit, en bits par seconde ou bauds). La vitesse usuelle est de 9600 bauds avec les ordinateurs mais d'autres vitesses sont supportées.

```
void setup()
{
  Serial.begin(9600);     //Ouvre le port série et fixe le débit à 9600 bauds
}
```

Note : Lorsque la liaison série est utilisée, les broches digitales 0 (RX) et 1 (TX) ne peuvent être utilisées en même temps.

serial.println(donnée)

Permet d'envoyer des données sur le port série et d'automatiquement retourner à la ligne ensuite. La commande `Serial.print()` fonctionne de la même façon mais sans le retour à la ligne, pouvant être moins facile à lire via le moniteur série.

```
Serial.println(analogValeur);              //envoi la valeur contenue dans "analogValeur"
```

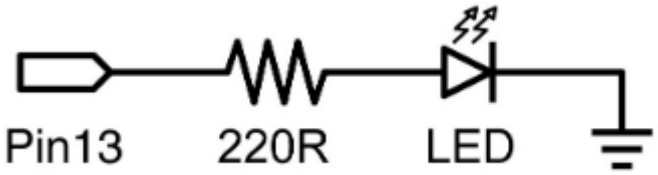
L'exemple suivant permet de lire les valeurs sur la broche analogique 0 et l'envoi vers l'ordinateur à chaque seconde.

```
void setup()
{
  Serial.begin(9600);                      //affecte le débit à 9600 bauds
}

void loop()
{
  Serial.println(analogRead(0));           //envoi la valeur analogique
  delay(1000);                             //marque une pause d'une seconde
}
```

Remarque :

APPENDICE

Sortie numérique

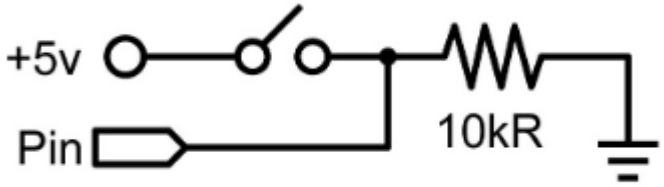


Il s'agit du programme de base : le "`hello world`" permettant d'activer et de désactiver quelque chose. Dans cet exemple, une LED est connectée sur la broche 13 et elle clignote à chaque seconde.

```
int brocheLED = 13;                     //LED sur la broche 13

void setup()                            //ne sera exécuté qu'une seule fois
{
  pinMode(brocheLED, OUTPUT);           //affecte la sortie 13 comme une sortie
}

void loop()                             //sera exécuté encore et encore
{
  digitalWrite(brocheLED, HIGH);        //allume la LED
  delay(1000);                          //pause d'une seconde
  digitalWrite(brocheLED, LOW);         //éteint la LED
  delay(1000);                          //pause d'une seconde
}
```

Entrée numérique

Il s'agit de la plus simple forme d'entrée avec uniquement deux états : ON ou OFF. Cet exemple permet de lire un simple changement d'état ou un appuie sur un bouton relié à la broche 4. Quand le circuit est fermé (c'est à dire le bouton appuyé), la broche lit HIGH et allume la LED.




```
int brocheLED = 13;       //broche où est reliée la LED
int brocheEntree = 4;     //broche d'entrée 4 (pour un bouton par exemple)

void setup()
{
  pinMode(brocheLED, OUTPUT);      //déclare la broche LED comme une sortie
  pinMode(brocheEntree,INPUT);     //déclare la broche 4 comme une entrée
}

void loop()
{
  if (digitalRead(brocheEntree) == HIGH)  //vérification si l'entrée est en HIGH (état haut)
  {
     digitalWrite(brocheLED,HIGH);        //allume la LED
     delay(1000);                         //pause d'une seconde
     digitalWrite(brocheLED, LOW);        //éteint la LED
     delay(1000);                         //pause d'une seconde
   }
}
```

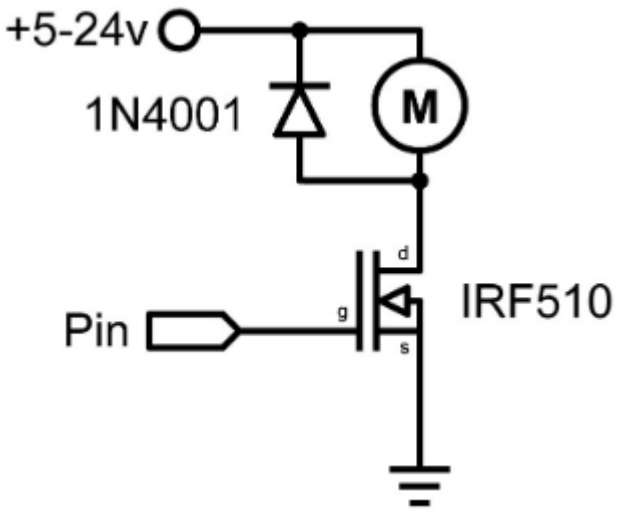
Remarque :

Courant de sortie haut

Il est parfois nécessaire de contrôler plus de 40mA via une Arduino. Dans ce cas, un MOSFET ou transistor peut être utilisé pour les contrôler. L'exemple suivant permet de changer un transistor entre marche et arrêt, cinq fois par seconde.

Note : Le schéma montre un moteur et une diode de protection mais un autre système non-inductif peut être utilisé à la place de la diode.

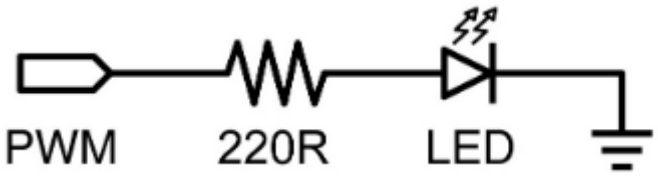
La diode 1N4001 est une diode de roue libre qui protège le transistor IRF510 des courants induits par le moteur lors du blocage du transistor.




```
int brocheSortie = 5;                       //sortie vers le transistor

void setup()
{
  pinMode(brocheSortie,OUTPUT);             //affecte la broche 5 en sortie
}

void loop()
{
  for (int i=0; i<=5; i++)                  //on fait 5 tours de boucles
  {
     digitalWrite(brocheSortie, HIGH);      //met le transistor en ON
     delay(250);                            //pause d'un quart de seconde
     digitalWrite(brocheSortie, LOW);       //met le transistor en OFF
     delay(250);                            //pause d'un quart de seconde
   }
   delay(1000);                              //pause d'une seconde
}
```

Sortie PWM (Pulse Width Modulation - modulation de largeur d'impulsions)

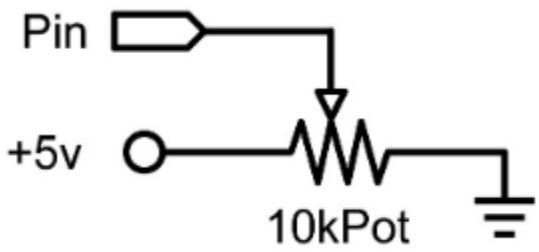
Arduino permet de synthétiser un signal analogique grâce à la modulation de largeur d'impulsions. On peut utiliser ce système pour faire varier la luminosité d'une LED ou pour contrôler un servomoteur. L'exemple suivant modifie lentement la luminosité d'une LED via deux boucles.




```
int brocheLED = 9;                          //broche PWM où est reliée la LED

void setup() {}                             //pas de paramètre nécessaire

void loop()
{
   for (int i=0; i<=255; i++)               //on augmente la valeur de i à chaque tour
   {
       analogWrite(brocheLED, i);           //fixe la luminosité à la valeur de i
       delay(10);                           //pause de 10ms
   }
   for (int i=255; i>=0; i--)               //on diminue la valeur de i à chaque tour
   {
       analogWrite(brocheLED, i);           //fixe la luminosité à la valeur de i
       delay(10);                           //pause de 10ms
   }
}
```

Entrée avec potentiomètre

Si on relie un potentiomètre à une broche analogique (analog-to-digital conversion ou ADC) de la carte Arduino, il est possible de lire la valeur analogique sur l'intervalle 0 à 1023. L'exemple suivant utilise un potentiomètre pour contrôler à quel rythme la LED clignote.




```
int brochePot = 0;    //broche d'entrée reliée au potentiomètre
int brocheLED = 13;   //broche de sortie pour la LED

void setup()
{
   pinMode(brocheLED, OUTPUT);    //déclare brocheLED en sortie
}

void loop()
{
   digitalWrite(brocheLED, HIGH);     //allume la LED
   delay(analogRead(brochePot));      //fait une pause
   digitalWrite(brocheLED, LOW);      //éteint la LED
   delay(analogRead(brochePot));      //fait une pause
}
```

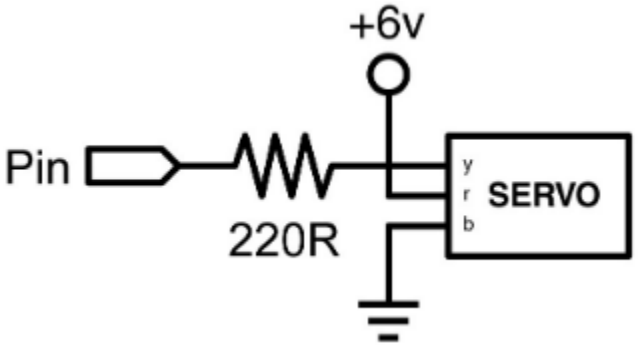
Entrée avec résistance variable

Les photorésistances, les thermorésistantes, les capteurs de flexions, et bien d'autres fonctionnent comme des résistances variables. L'exemple suivant utilise une fonction pour lire la valeur analogique et l'affecter au temps de pauses, permettant ainsi de contrôler la vitesse de clignotement de la LED.


```
int brocheLED = 9;                 //broche PWM reliée à la LED
int brocheAnalogique = 0;          //résistance variable reliée à la broche analogique 0

void setup() {}                    //pas de paramètre nécessaire

void loop()
{
   for (int i=0; i<=255; i++)    //on augmente la valeur de i à chaque tour
   {
       analogWrite(brocheLED, i);   //fixe la luminosité à la valeur de i
       delay(pause());    //Marque une pause selon la valeur renvoyée par la fonction pause()
   }
   for (int i=255; i>=0; i--)     //on diminue la valeur de i à chaque tour
   {
       analogWrite(brocheLED, i);   //fixe la luminosité à la valeur de i
       delay(pause());    //Marque une pause selon la valeur renvoyée par la fonction pause()
   }
}

int pause()
{
   int v;     //créé une variable temporaire
   v = analogRead(brocheAnalogique);    //lecture de la valeur analogique
   v /= 8;     //convertie la valeur comprise entre 0 et 1024 en une valeur entre 0 et 128
   return v;    //renvoi la valeur finale
}
```

Sortie vers Servomoteur

Les servomoteurs « Hobby » sont un type de moteur indépendant pouvant tourner à 180°. Ils nécessitent une pulsation envoyée toute les 20ms. L'exemple suivant utilise la fonction `pulsationServo` pour faire avancer le servomoteur de 10° à 170° puis revenir en arrière.



```
int brocheServo = 2;      //servomoteur connecté en broche digitale 2
int monAngle;             //angle réalisé par le servo, de 0 à 180°
int pulsation;            //variable servant à la fonction pulsationServo

void setup()
{
   pinMode(brocheServo, OUTPUT);     //affecte la sortie 2 comme une sortie
}

void pulsationServo(int brocheServo, int monAngle)
{
   pulsation = (monAngle*10) + 600;         //détermine la pause (delay)
   digitalWrite(brocheServo, HIGH);         //active le servomoteur
   delayMicroseconds(pulsation);            //pause en microseconde
   digitalWrite(brocheServo, LOW);          //désactive le servomoteur
}

void loop()
{
   //le servomoteur commence à 10° et tourne jusqu'à 170°
   for (monAngle=10; monAngle<=170; monAngle++)
   {
      pulsationServo(brocheServo, monAngle);       //envoi à la fonction la broche et l'angle
      delay(20);                                   //ralentie le cycle
   }
   //le servomoteur commence à 170° et tourne jusqu'à 10°
   for (monAngle=170; monAngle>=10; monAngle--)
   {
      pulsationServo(brocheServo, monAngle);       //envoi à la fonction la broche et l'angle
      delay(20);                                   //ralentie le cycle
   }
}
```

LIBRAIRIES

Une bibliothèque est un ensemble de fonctions utilitaires, regroupées et mises à disposition des utilisateurs de l'environnement Arduino afin de ne pas avoir à réécrire des programmes parfois complexes. Les fonctions sont regroupées en fonction de leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc). Arduino comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez les sélectionner dans *Sketch > Import Library*.

L'instruction suivante sera alors ajoutée au début de votre programme : `#include <la_bibliothèque.h>`

Cette commande inclut au code source tout le contenu de la bibliothèque. Les fonctions qu'elle contient peuvent alors être appelées au même titre que les fonctions de base.

Note : les bibliothèques logicielles se distinguent des exécutables par le fait qu'elles ne s'exécutent pas "seules" mais sont conçues pour être appelées par d'autres programmes.

Les bibliothèques Arduino sont composés d'au moins deux fichiers : un fichier d'en tête finissant par `.h` et un fichier source finissant par `.cpp`.

Le fichier d'en tête contient les définitions des fonctions disponibles et le fichier source contient l'implémentation du code. C'est à dire le code à l'intérieur des fonctions qui ont été définies dans le fichier d'en tête.

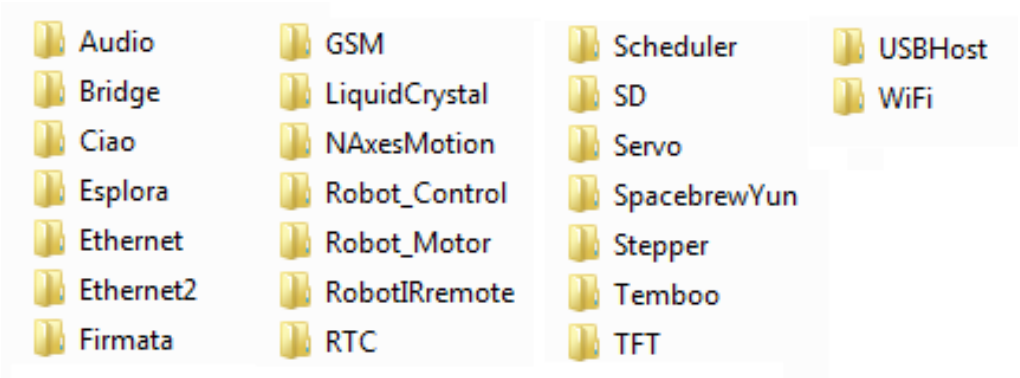
Bibliothèques fournies par défaut dans le logiciel Arduino

- **EEPROM** : lecture et écriture de données dans la mémoire permanente.
- **Ethernet** et **Ethernet2** : pour se connecter à Internet en utilisant le *Shield Ethernet*.
- **Firmata** : pour rendre l'Arduino directement accessible à des applications en utilisant un protocole sériel.
- **LiquidCrystal** : pour contrôler les afficheurs à cristaux liquides (LCD).
- **SD** : pour la lecture et l'écriture de données sur des cartes SD.
- **Servo** : pour contrôler les servomoteurs.
- **SPI** : pour communiquer avec les appareils qui utilisent le protocole de communication SPI (Serial Peripheral Interface).
- **SoftwareSerial** : pour établir une communication sérielle supplémentaire sur des entrées et sorties numériques (la carte Arduino dispose d'un seul port sériel *hardware*).
- **Stepper** : pour commander des moteurs « pas à pas ».
- **Wire** : pour interfacer plusieurs modules électroniques sur un bus de données utilisant le protocole de communication TWI/I2C.

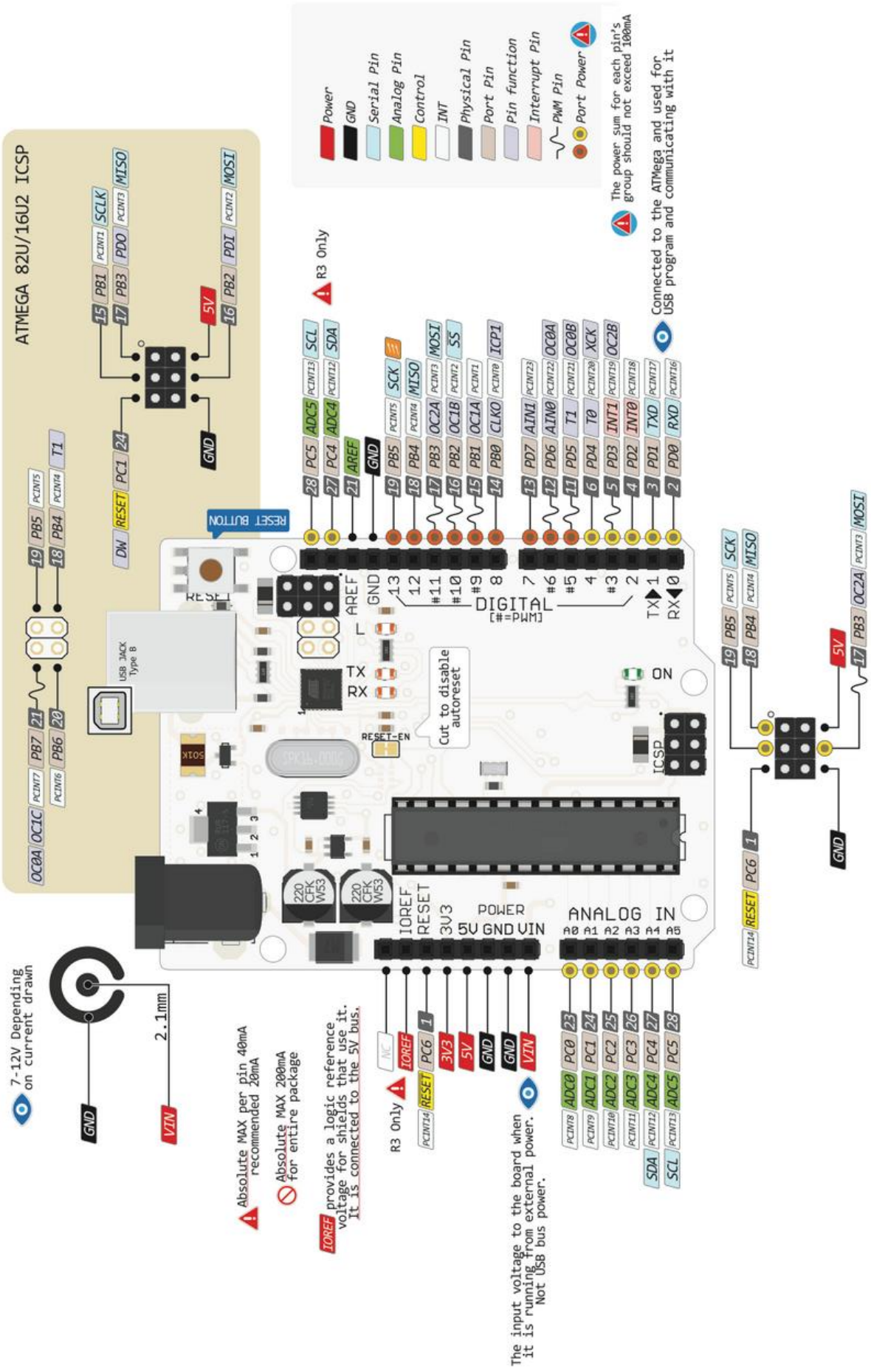
D'autres librairies sont disponibles en téléchargement à l'adresse suivante.

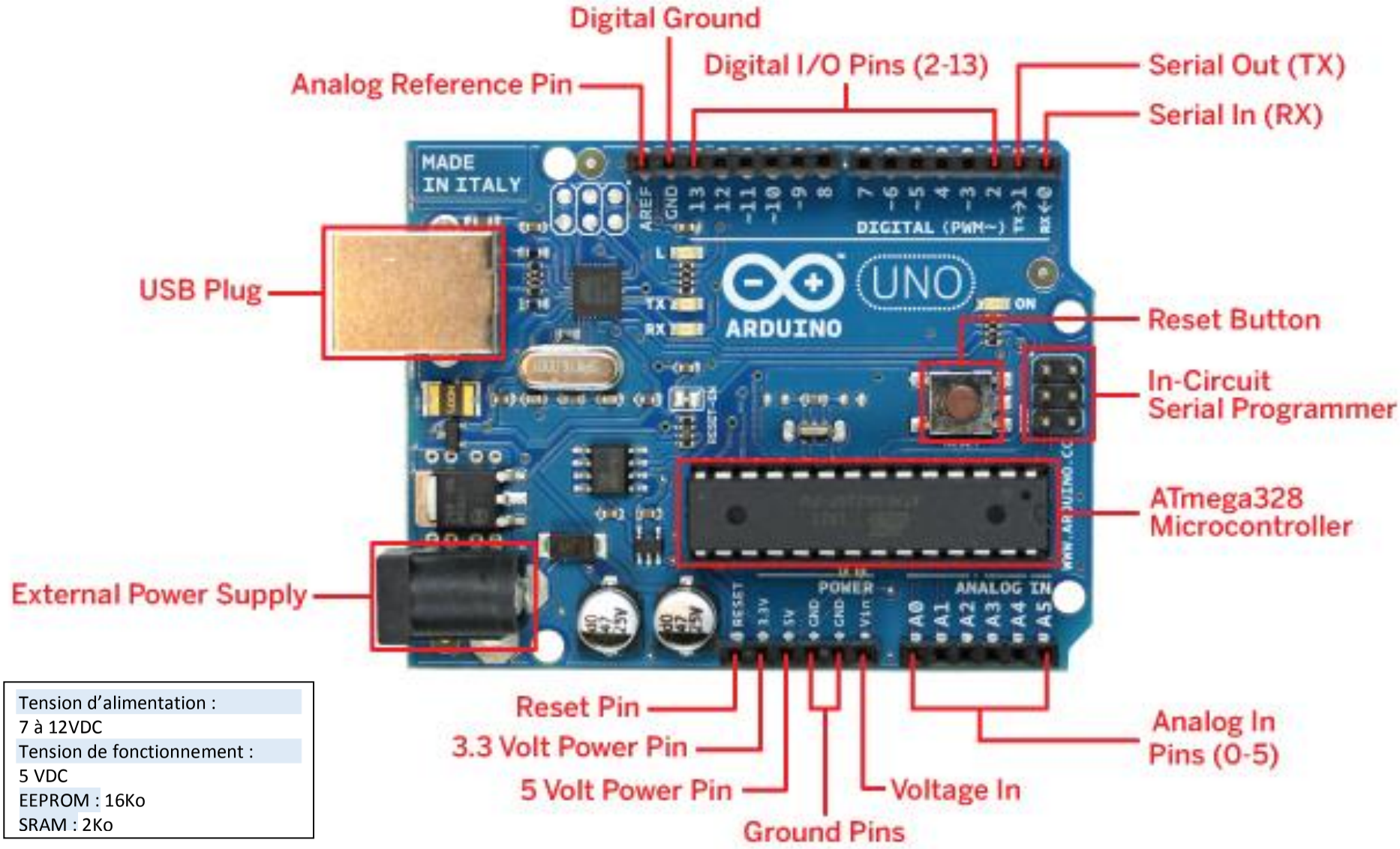
- http://www.arduino.cc/en/Reference/Libraries

Pour installer ces librairies provenant de tiers, il faut décompresser le fichier téléchargé et le stocker dans un répertoire appelé *libraries* situé dans le répertoire *sketchbook*. Sur Linux et Windows, le dossier *sketchbook* est créé au premier lancement de l'application Arduino dans votre dossier personnel. Sur Mac OS X, un dossier Arduino est créé dans le répertoire « Documents ».
Par exemple, pour installer la *librairie* DateTime, le fichier devra être déposé dans le dossier : /libraries/DateTime de votre *sketch*.



Cartographie des entrée/sorties ARDUINO UNO.

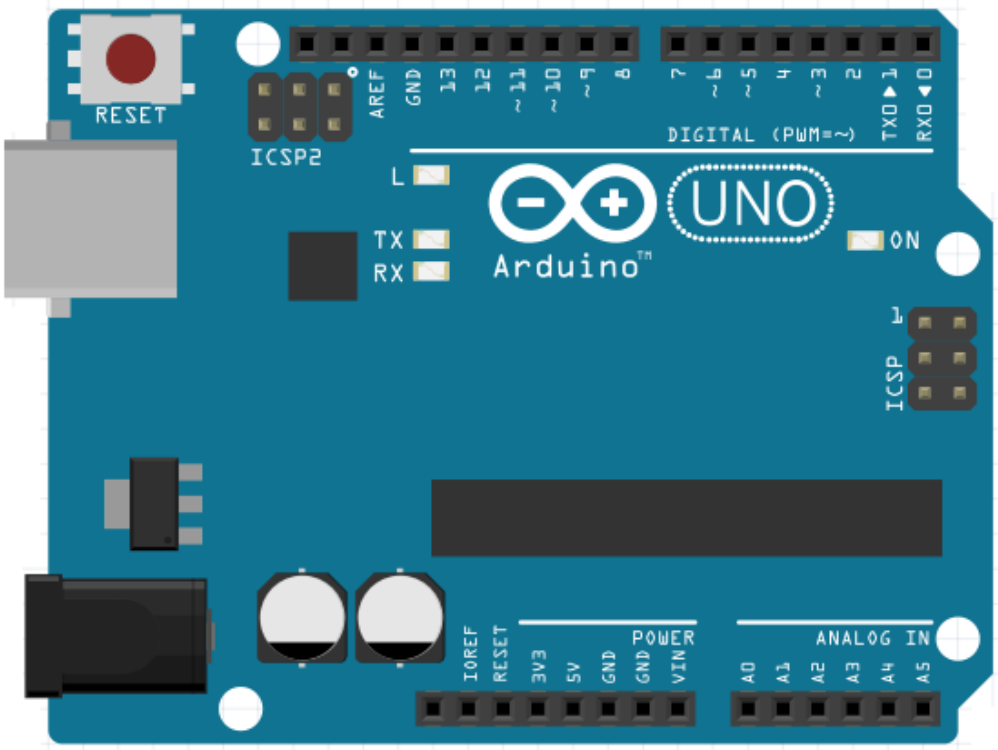


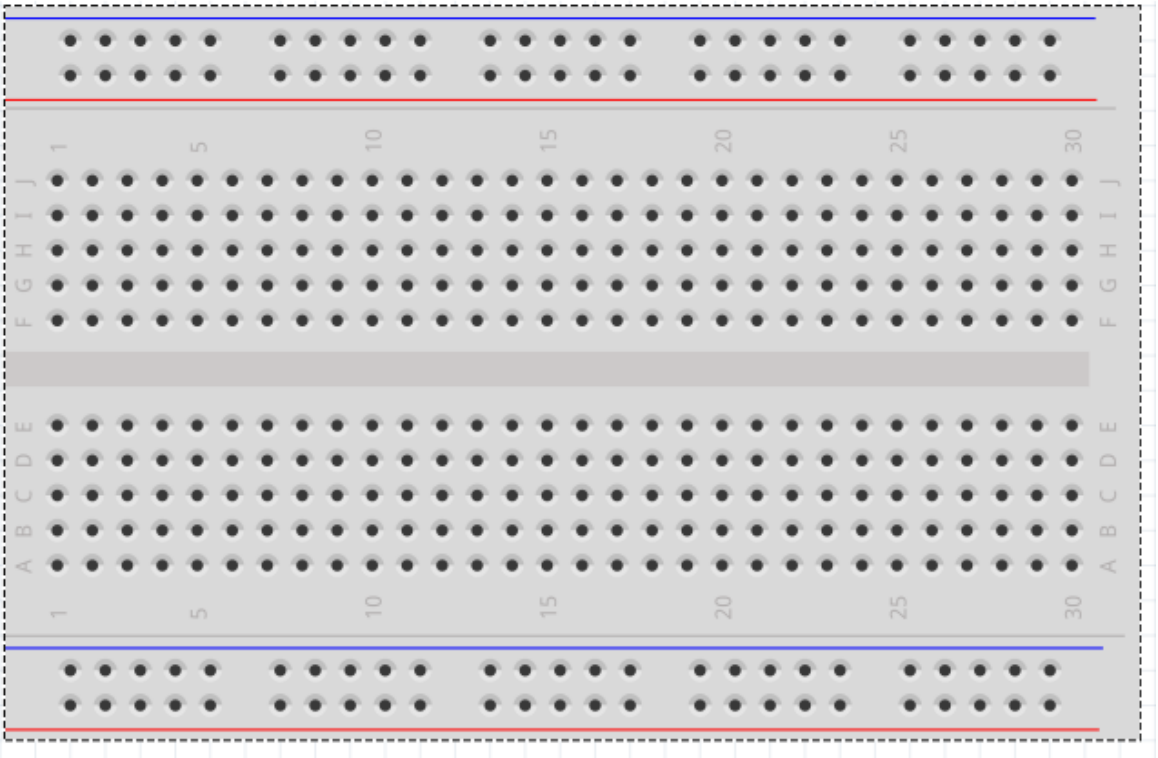


Tension d'alimentation :
7 à 12VDC
Tension de fonctionnement :
5 VDC
EEPROM : 16Ko
SRAM : 2Ko

Prototypage ARDUINO :

Exercice : ……





Programme : TestCapteursTMP36.

```
/*
 Mesure de la température de 4 capteurs TMP36 et affichage sur LCD 16x2

 Afficheur LCD: SHD 1602 LCD B
 Capteurs de température: TMP36-1 Analog Output - Pin A1 (analogique)
                          TMP36-2 Analog Output - Pin A2 (analogique)
                          TMP36-3 Analog Output - Pin A3 (analogique)
                          TMP36-4 Analog Output - Pin A4 (analogique
                          Résolution: 10 mV / degré Celsius avec un offset de 500 mv
*/

// Intégration de la librairie pour l'afficheur LCD:

#include <LiquidCrystal.h>

int tempCapteurPin1 = 1; // Variable associée à la Pin analogique 1 pour la lecture de la tension de sortie du TMP36-1 (Vout).
int tempCapteurPin2 = 2; // Variable associée à la Pin analogique 1 pour la lecture de la tension de sortie du TMP36-2 (Vout).
int tempCapteurPin3 = 3; // Variable associée à la Pin analogique 1 pour la lecture de la tension de sortie du TMP36-3 (Vout).
int tempCapteurPin4 = 4; // Variable associée à la Pin analogique 1 pour la lecture de la tension de sortie du TMP36-4 (Vout).

// Initialisation des pins utilisées pour l'afficheur LCD

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup(){
  // Initialisation du LCD:
  lcd.begin(16, 2);
  lcd.clear();
  Serial.begin(9600);
}

// Initialisation des températures affichées
int lastTemp1 = -100;
int lastTemp2 = -100;
int lastTemp3 = -100;
int lastTemp4 = -100;

void loop(){

  float temp1 = lectureTemp1();
  float temp2 = lectureTemp2();
  float temp3 = lectureTemp3();
  float temp4 = lectureTemp4();

  // Rafraichir le LCD que si la température a varié sensiblement
  if( abs(temp1-lastTemp1)<0.20 )
    return;
  lastTemp1 = temp1;

  // Rafraichir le LCD que si la température a varié sensiblement
  if( abs(temp2-lastTemp2)<0.20 )
    return;
  lastTemp2 = temp2;

  // Rafraichir le LCD que si la température a varié sensiblement
  if( abs(temp3-lastTemp3)<0.20 )
    return;
  lastTemp3 = temp3;
```

```
  // Rafraichir le LCD que si la température a varié sensiblement
  if( abs(temp4-lastTemp4)<0.20 )
    return;
  lastTemp4 = temp4;

  // Afficher la valeur Temp1
  lcd.setCursor(0,0);
  lcd.print( temp1 );
  lcd.print((char)223);  // affiche le signe degré "°"
  lcd.print( "C" );
  lcd.print((char)255);  // affiche une case pleine
  lcd.print((char)255);  // affiche une case pleine

  // Afficher la valeur Temp2
  lcd.setCursor(9,0);
  lcd.print( temp2 );
  lcd.print((char)223);  // affiche le signe degré "°"
  lcd.print( "C" );

// Afficher la valeur Temp3
  lcd.setCursor(0,1);
  lcd.print( temp3 );
  lcd.print((char)223);  // affiche le signe degré "°"
  lcd.print( "C" );
  lcd.print((char)255);  // affiche une case pleine
  lcd.print((char)255);  // affiche une case pleine

  // Afficher la valeur Temp4
  lcd.setCursor(9,1);
  lcd.print( temp4 );
  lcd.print((char)223);  // affiche le signe degré "°"
  lcd.print( "C" );

  // Efface les derniers caractères si la température chute subitement
  lcd.print( "      " );

  // Temporisation pour ne pas rafraichir trop souvent l'affichage
  delay(3000);

}

//Lecture de la température sur la pin A1
//Retour: La température 1 en degré Celcius.

float lectureTemp1(){
   // Lecture de la valeur sur l'entrée analogique A1
   // Retourner une valeur entre 0->1023* pour 0->5v
   int valeur1 = analogRead(tempCapteurPin1);

    Serial.print("Valeur1 : ");
    Serial.println(valeur1);
    Serial.println("---------------------");

   // Convertir la valeur CAN en température
   float temperature1 = ((valeur1 * 0.64375) - 77.09);

   return temperature1;
}
```

```
//Lecture de la température sur la pin A2
//Retour: La température 2 en degré Celcius.

float lectureTemp2(){
  // Lecture de la valeur sur l'entrée analogique A2
  // Retourner une valeur entre 0->1023 pour 0->5v
  int valeur2 = analogRead(tempCapteurPin2);

   Serial.print("Valeur2 : ");
   Serial.println(valeur2);
   Serial.println("----------------------");

  // Convertir la valeur CAN en température
  float temperature2 = ((valeur2 * 0.64375) - 77.09);

  return temperature2;
}

//Lecture de la température sur la pin A3
//Retour: La température 3 en degré Celcius.

float lectureTemp3(){
  // Lecture de la valeur sur l'entrée analogique A3
  // Retourner une valeur entre 0->1023 pour 0->5v
  int valeur3 = analogRead(tempCapteurPin3);

   Serial.print("Valeur3 : ");
   Serial.println(valeur3);
   Serial.println("----------------------");

  // Convertir la valeur CAN en température
  float temperature3 = ((valeur3 * 0.64375) - 77.09);

 return temperature3;
}

//Lecture de la température sur la pin A4
//Retour: La température 4 en degré Celcius.

float lectureTemp4(){
  // Lecture de la valeur sur l'entrée analogique A4
  // Retourner une valeur entre 0->1023 pour 0->5v
  int valeur4 = analogRead(tempCapteurPin4);

   Serial.print("Valeur4 : ");
   Serial.println(valeur4);
   Serial.println("----------------------");

  // Convertir la valeur CAN en température
  float temperature4 = ((valeur4 * 0.64375) - 77.09);

  return temperature4;
}
```