http://www.didel.com/ **info@didel.com**

www.didel.com/diduino/Cours01.pdf

Cours Arduino/C 1ère partie

Les documents Diduino faciles à cliquer sont listés sous www.didel.com/Diduino.html

Ce document fait suite au document www.didel.com/diduino/Start.pdf , qui se borne à vérifier que l'environnement Arduino est opérationnel, et tester le programme qui clignote une Led.
Il a servi de base pour les documents http://www.didel.com/coursera/LC.pdf qui vont plus en profondeur et offrent quantité d'exercices avec le shield LearnCbot.
Ce document permet de se familiariser avec la programmation en C, en testant quelques exemples. Pour mieux comprendre les composants électroniques et leur programmation, voir www.didel.com/diduino/Composants01.pdf et les chapitres suivants.
Pour mieux comprendre la programmation en C et voir quelques fonctions intéressantes de l'AVR328 qui anime la carte Diduino, les documents http://www.didel.com/coursera/LC.pdf vont plus en profondeur et offrent quantité d'exercices avec le shield Le.arnCbot

Cette 1ère partie ne va pas plus loin que le `if`, mais elle s'efforce de bien expliquer les bases en insistant sur l'utilisation de #define, qui évitent de polluer les programmes avec des digitalWrite peu explicites.
Pour tester les programmes, un poussoirs avec pull-up de 4k7 à 100k, une Led et sa résistance de 1 à 4k7 et un buzzer passif et sa résistance de 47 à 220 Ohm sont nécessaires.

1.1 Règle générale pour tous les programmes

On doit ''configurer'' l'application, c'est-à-dire mentionner les pins que l'on utilise, dire si ce sont des entrées ou sorties. C'est le **setup.**
On doit ensuite dire ce que l'on va faire en boucle (le processeur ne sait pas arrêter de travailler) C'est le **loop**. Avec des parenthèses, accolades, point-virgules au bon endroit , comme le veut le C!

1.2 Clignoter une Led

Sur toutes les cartes Arduino, on a sur la pin 13 une Led déjà câblée. Elle s'allume si la pin 13 est à l'état "1" (HIGH, tension de 5V). Pour clignoter cette Led, il faut écrire:

Le nom du fichier en commentaire
Un objet Led est branché sur la pin 13, à laquelle on donne un nom lié à l'objet.
 	
la pin13 appelée Led est une sortie
 	
on répète sans fin
allumer la Led
attendre une demi-seconde
eteindre la Led
attendre une demi-seconde
recommencer (fin partie répétée)
	`// Clignote.ino`
	`#define Led 13`
	
	`void setup () {`
	`  pinMode(Led, OUTPUT);`
	`}`
	
	`void loop () {`
	`  digitalWrite(Led,HIGH);`
	`  delay(500); // 500 millisecondes`
	`  digitalWrite(Led,LOW);`
	`  delay(500);`
	`}`

Il faut accepter sans bien comprendre pour le moment les mots réservés, la présence de () ; { } et respecter ces notations scrupuleusement , de même que les minuscules et majuscules.
Le compilateur qui fait la traduction est intolérant, et ses messages ne sont pas toujours faciles à comprendre.

1.2 Nommons les actions

Ecrire `digitalWrite(Led,HIGH);` pour allumer la Led, c'est parler le petit-nègre du processeur. On peut lui apprendre facilement un langage clair, simple, facile à taper, qui respecte les contraintes du C.
Par exemple `LedOn` pour allumer la led `LedOff` pour éteindre. Il suffit de définir des équivalences.

```
#define LedOn digitalWrite(Led,HIGH);
#define LedOff digitalWrite(Led,LOW);
```

Le programme devient plus clair

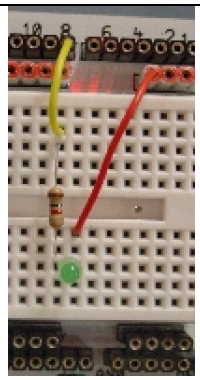
```
void loop () {
  LedOn;
  delay(500);
```

```
    LedOff;
    delay(500);
}
```

1.3 Exercice
Changer la durée de Led allumée. Réduire de moitié, un quart, plus. La diminution apparente de visibilité est-elle linéaire? On voit encore avec 1ms?

1.4 Exercice
Câbler une Led sur la pin 8, avec le courant qui vient du +5V et part dans la pin 8, ou il faudra un zero volt pour allumer. L'ordre résistance-Led n'a pas d'importance, mais la pin courte (cathode) doit être du côté du moins . Une résistance de 1 à 4k Ohm convient.

```
. . .
void loop () {
     LedOn;
     delay(500);
     LedOff;
     delay(1);
}
```



Il faut ajouter Led8 dans les définitions et modifier le programme pour clignoter la Led8.
Exercice: Modifier le programme pour que les deux Leds clignotent ensemble ou en opposition.On doit donc ajouter 3 définitions pour nommer clairement la fonctionnalité, et ajouter une ligne dans le set-up, puisqu'il y a un objet de plus à mettre en service.

```
#define Led8 8
#define Led8On digitalWrite(Led8,LOW);
#define Led8Off digitalWrite(Led8,HIGH);
pour le setup: pinMode(Led8, OUTPUT);
```

Le câblage n'est pas le même pour la Led8 que pour la Led13: un zéro allume, ce qui est exprimé dans le #define. Mais dans le programme, on peut penser Led8On Led8Off en oubliant cette différence de câblage.

Exercice 1.4 Faire clignorer la Led8, les deux en même temps ou en opposition de phase.

1.5 Résumé
Si un fil est câblé sur une pin (Led, moteur, capteur), il y a 3 choses préciser avec tous les processeurs. Avec Arduino on écrit

nommer le signal sur la pin	dire comment le processeur doit l'initialiser (setup)	nommer les actions utilisées dans le programme: la Led8 est allumée si la sortie est HIGH
`#define Led 13`	`pinMode (Led8,OUTPUT) ;`	`#define LedOn digitalWrite (Led8,HIGH) ;`

Tapez `led` à la place de `Led`. `Delay` à la place de `delay`. Oubliez un `;` une { . Cela vous arrivera! Alors testez pour voir ce que dit le compilateur et ne pas être trop perdu au prochains messages d'erreur.

1.6 Comprenons bien et souvenons-nous
`#define Led 8` est l'une des notations (on voit souvent `int Led=8; ou const int Led=8;`) qui pemet au compilateur de remplacer le nom choisi par le numéro 8 défini par Arduino, lui-même transformé dans des instructions qui agissent sur une patte du microcontrôleur AtMega328 .

`pinMode (Led8,OUTPUT) ;` Arduino dit à l'AtMega328 de mettre cette pin en sortie

`digitalWrite (Led,HIGH) ;` est une fonction (une opération) avec deux paramètres:
1) le nom de la pin que l'on a renommé pour que cela soit clair
2) une valeur binaire appelée `HIGH` et `LOW` (on peut écrire `1` et `0` à la place) qui fait que le programme active la pin à une tension proche de 5V ou proche de 0V.
Attention, `HIGH` ne veut pas dire allumé. Cela dépend du câblage.
Un #define nomme l'action câblée, plus besoin dans le programme de réfléchir au câblage..

`delay (500) ;` est aussi une fonction avec comme paramètre la durée en millisecondes.
Cette durée est un mot de 16 bits, donc la valeur max est 2^{16}-1= 65535 ms (~1minute).

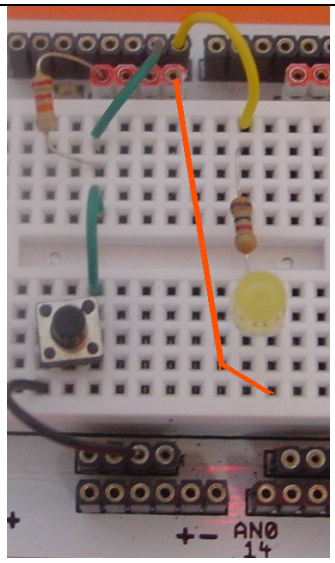
Les paramètres sont entre parenthèses `( )` , les blocs d'instructions entre accolades { }
et les instructions se terminent par un `;`

Dans un #define, il n'y a pas de ; à la fin. Il est ajouté en tapant le mot défini.

1.7 Lire le poussoir

Un poussoir peut comme la Led être branché électriquement de deux façons, suivant que l'on veut que le processeur lise HIGH (5V) ou LOW (0V) quand on pèse. Ne documentons que la plus usuelle.
Si on ne pèse pas, il faut que la tension d'entrée sur le processeur soit définie, donc on rajoute une résistance "pull-up" vers le 5V, donc un HIGH tant que l'on ne pèse pas (22 kOhm – rouge-rouge-orange convient bien). Si on pèse, on envoie le courant de la résistance dans le 0V (Gnd) et le processeur lit un LOW.
Le poussoir est branché sur une pin qui doit être nommée et déclarée en entrée.PousOn est le nom choisi pour dire que l'on agit sur le poussoir.

```
#define Pous9 9
#define PousOn  !digitalRead(Pous9)
#define PousOff digitalRead(Pous9)
void setup () {
  pinMode (Pous9, INPUT);
}
```



Attention! `digitalRead(Pous9)` lit l'état de la pin 9. Si on ne presse pas, c'est l'état actif, état 1, 5V. Pour exprimer l'état inverse, C utilise le signe !, appelé inversion logique.

Pour lire le poussoir et copier son état sur le Led ou faire un autre action, on teste sa valeur

```
if(PousOn)  { LedOn; }
else { LedOff; }
```

1.8 Copier l'état du poussoir dans une variable

Une autre solution, qui donne plus de possibilités d'actions, est de copier l'état dans une variable. Appellons la variable etatPoussoir. C'est une position mémoire, un tiroir qui contient un nombre, que l'on doit réserver en déclarant

```
byte etatPoussoir ;
```
(`byte` est 8 bits, `int` est 16 bits)

On peut alors écrire dans le programme

```
etatPoussoir = PousOn ;
```

1.9 Allumer selon une variable

La conditions allumer/éteindre peut dépendre d'une condition extérieure ou d'une variable.
La fonction `AllumeLed8(variable)` allume si la variable est à 1, eteint si zéro.

```
void AllumeLed8(byte vv) {
  if(PousOn)  { LedOn; }
  else { LedOff; }
}
```

Le programme complet est

```
//Copy1.ino  Copie un poussoir sur une Led
//Le poussoir sur la pin9 est câblé vers le - avec une pull-up. La pin est LOW si actif
//La Led sur pin8 est câblée vers le +5V.  pin est LOW si allumé
#define Led8 8
#define Led8On digitalWrite(Led8,LOW);
#define Led8Off digitalWrite(Led8,HIGH);
#define Pous9 9
#define PousOn  !digitalRead(Pous9)
#define PousOff digitalRead(Pous9)
void setup () {
pinMode (Led8,OUTPUT);
pinMode (Pous9,INPUT);
}
// fin du bloc de définitions, début de l'application
byte etatPous9 ;
void loop () {
etatPous9 = Pous9On ; // on lit le poussoir
AllumeLed8 (etatPous9);   // on copie ce que l'on a lu
}
```

On remarque que l'utilisation de la variable etatPous9 n'est pas nécessaire ici.

1.9 Soyons critiques

Ce programme est-il bien écrit?
Le nom du fichier est clair ? – pas trop
L'objectif du programme est bien résumé? – oui
On peut câbler le montage sans une figure explicative? – non
Les noms des définitions et variables sont simples et non ambigus? – oui
Le groupes fonctionnels sont bien séparés? – oui
Les accolades sont bien visibles, les instructions tabulées et alignées ? - oui

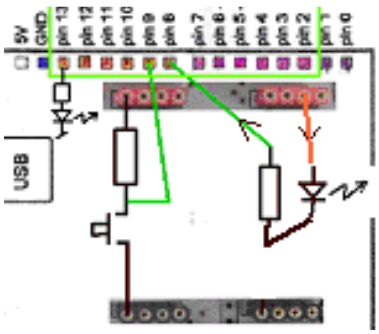
On reviendra sur les règles d'écriture, très importantes en C (minuscules, majuscules, signes).
Pour les commentaires, un commentaire par ligne est absurde. Il faut commenter la structure, pas les instructions qui doivent être familières. Les noms choisis doivent être mnémotechniques.

1.10 Instruction if (condition) { }

La condition est une valeur booléenne, vrai ou faux. Si la condition est vraie, on exécute le contenu de l'accolade. La condition est souvent une comparaison if (a<b), if (a<=b), if (a==b).
Attention, le `==` n'est pas une erreur ! a=b veut dire que l'on copie la valeur de la variable b dans a. `a==b` veut dire ''est-ce que `a` a la même valeur que `b`" ?
La réponse est *vrai* ou *faux* (codés 1 ou 0). Si vous êtes distrait et que vous mettez un `= à la place du ==`, le compilateur ne dira rien (cela a un sens spécial pour lui), et en fera à sa tête.
Souvent, on teste des nombres entiers 8 ou 16 bits ou plus. ***faux*** c'est la valeur zéro. ***vrai*** c'est tout le reste, différent de zéro.

Dans in `if` a partie `else` n'est écrite que si elle est nécessaire.

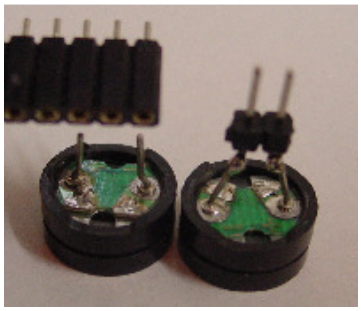
1.11 Exercices

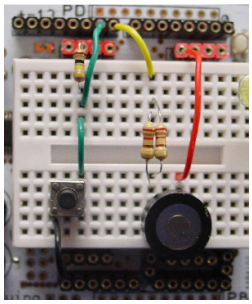


On continue avec le câblage précédent, un poussoir en 9 et les deux Leds en 13 et 8.
Si on presse, une Led clignote à une vitesse donnée, si on relâche, c'est l'autre qui clignote à une vitesse différente.

Un peu plus difficile, une Led clignote deux fois plus vite que l'autre quand on pèse, mais l'autre continue à clignoter sans changement.

1.12 Un peu de bruit





Branchons le buzzer à la place de la Led commandée par la pin 8, avec une résistance de 220 Ohm (rouge-rouge-brun) qui limite le courant, donc la puissance. La photo montre 2 résistances de 220 Ohm en parallèle pour avoir 100 Ohm et un son un peu plus fort.
Les pattes du buzzer sont au pas de 6.2mm. Il faut les plier parallèles à 5mm, ou mieux, souder un connecteur M07-2 au pas de la grille du bloc d'expérimentation.
Ce buzzer, que l'on devrait appeler transducteur électromécanique, est une simple membrane métallique devant une bobine, comme un haut-parleur. Vous faites passer du courant, la membrane colle et vibre un peu, on entend un tic.
Vous faites vibrer à 1 kHz (donc coller la membrane 0.5ms et décoller 0.5ms) le son est continu, de mauvaise qualité avec des harmoniques. A certaines fréquences, le son est plus fort ou plus faible à cause de phénomènes de résonance. Passons, ce n'est pas un cours de physique! Et il y a plus d'explications sous www.didel.com/diduino/Composants.pdf , section C9.

L'instruction `delay(0.5)` ; pour 0.5 millisecondes ne passe pas! Seul un paramètre de 1 à 2^{16}-1 est documenté. Pour les délais courts, il faut donc une fonction supplémentaire d'Arduino, `delayMicroseconds (xx)` ; qui semble documentée sur 15 bits (max 16383). 0.5 ms est égal à 500 microsecondes.

```
HpOn; // active
delayMicroseconds (500) ;
HpOff; // désactive (pas de courant)
delayMicroseconds (500) ;
```

Avec ces instuctions on entend un son continu qui va vite nous casser les oreilles. Plutôt que de déconnecter la carte, utilisons un `if` . Pour entendre le son il faut agir sur le poussoir.

```
// Son.ino
#define Hp8 8
#define HpOn digitalWrite(Hp8,LOW);
#define HpOff digitalWrite(Hp8,HIGH);
#define Pous9 9
#define PousOn  !digitalRead(Pous9)
void setup () {
  pinMode (HP8, OUTPUT);
  pinMode (Pous9, INPUT);
}

void loop () {
 if ( PousOn ){
   HpOn;
   delayMicroseconds (500) ;
   HpOff;
   delayMicroseconds (500) ;
 }
}
```

Si on ne presse pas sur la touche, le processeur tourne en boucle à lire la pin 9 à toute vitesse (un million de fois par seconde).

A noter encore que l'on peut déclarer HpToggle (bacule) en écrivant

```
#define HpToggle digitalWrite(Hp8,!digitalRead(Hp8));
```

et simplifier le programme

```
void loop () {
 if ( PousOn ){
   HpToggle;
   delayMicroseconds (500) ;
 }
}
```

1.13 Durée variable

Comment varier la durée? En définissant une variable et en la modifiant.
au début `int demiPeriode = 500 ;  max`
dans la boucle

```
HpToggle;
delayMicroseconds (demiPeriode) ;
```

On fait la même chose qu'avant si on en reste là. Mais dans la boucle on peut ajouter

```
demiPeriode = demiPeriode-2;
```

ou `demiPeriode = demiPeriode+2; (ou +100)`

Essayez! (créez Son1.ino avant de modifier Son.ino)
Que se passe-t-il? A chaque boucle, qui dure 1ms la première fois, on modifie la période, donc la fréquence.
Avec –2, on passe rapidement dans les ultra-sons. Que se passe-t-il quand on arrive à zéro ?
Avec +2, la variation est toujours plus lente. Vous pouvez calculer combien de temps jusqu'à 1 Hz?

1.14 Les variables

Dans la mémoire de l'ordinateur, il y a une zone pour le programme que l'on a téléchargé et qui ne se modifie pas à l'exécution.
Il y a en plus de la mémoire vive (RAM) que le processeur (via votre programme) peut lire et écrire. Il faut voir cette zone comme une commode avec des tiroirs. le fabricant a numéroté les tiroirs, pas très pratique. On leur donne des noms : pullover, chaussettes, etc. Il y a deux types de tiroirs, des larges et des demis (pas besoin d'un grand pour les

chaussettes). Pour savoir ce qu'il y a dans un tiroir, il faut l'ouvrir. Pour ajouter un pull, on prend toute la pile, on "additionne" le nouveau et on remet dans le tiroir. Quand on regarde un tiroir, on ne sait pas ce qu'il contient. Quand on installe une nouvelle commode, il peut déjà y avoir des choses dans les tiroirs.

Les inventeurs de l'ordinateur n'ont rien inventé. La RAM est une commode.
Sur les microcontrôleurs, les tiroirs ont en général 16 bits de large, ce qui convient bien on l'a vu pour une durée de 1 ms à plus d'une minute. Pour coder une lettre de l'alphabet, 8 bits (256 possibilités) est bien suffisant. Le C définit une dizaine de types de données pour optimiser différents cas, le catalogue nécessite beaucoup d'explications.
Pour le moment contentons-nous du type 16 bits, appelé int (pour integer). On sera parfois limité, et souvent on va gaspiller de la mémoire en mettant dans ce mot (tiroir) de 16 bits, 8 bits ou même un seul bit: 0 ou 1, vrai ou faux, ce que l'on a fait en 1.8 en définissant `int etatPoussoir ;`
qui n'a que deux valeurs: 0 (pressé) et 1 (relâché – avec le câblage choisi)
En écrivant en 1.13

```
int demiPeriode = 500 ;
```

on a à la fois donné un nom à la variable et dit ce qu'elle contient. Le délai est un nombre de 0 à 2^{16}-1 (= 65535).
`int aa = 70000 ;` est refusé, mais attention, il faut voir ces nombres 16 bits comme disposés sur un grand cercle; ils se mordent la queue: 2^{16} = 0
Il n'y a jamais de débordements dans les opérations: si on a dans un programme
`int aa = 70000 ;` et plus loin
`aa = aa+10000` laisse dans la variable la valeur 70000-65535 = 4475. OK?

`int` est aussi utilisé par Arduino pour donner un nom à une pin. En précisant `const int` on évite d'utiliser une position RAM pour une simple constante.

Encore un point avant de revenir aux sons. Le C vous permet de calculer selon vos habitudes, pour ce qui est simple. Mentionnons juste le truc pour paresseux:
Au lieu de `aa = aa+10000 ,`
C permet d'écrire `aa += 10000` . On a aussi `-=` et d'autres opérations vues plus loin.

1.15 Des bips

On ne connaît pour l'instant que la commande `if` . Exerçons cette instruction avant de passer au `for` et autres instructions du C. Le `if` n'est pas dans les prochains exemples la meilleure solution, mais il nous faut faire un pas à la fois pour ne pas risquer de s'encoubler.

Pour générer un bip, il faut compter un certain nombre d'oscillations de la membrane du haut-parleur.
On peut compter de zéro jusqu'à la valeur limite, ou décompter depuis cette valeur.
Les constantes sont DureeBip et DemiPeriodeBip.
Si la période est 1ms, et la durée du bip 1 seconde, il faut répéter 1000 fois la période.
Cela définit la valeur des deux constantes.
La variable est cntPeriode; on l'initialise à zéro.

A la fin du bip, on attend une seconde pour recommencer (Bip.ino)

```
void loop () {
      cntPeriode += 1 ;
      if ( cntPeriode < DureeBip ) {
            // les 4 instructions pour jouer une période
      } else {
            delay (1000) ;
            cntPeriode = 0 ;
      }
}
```

On voit bien ce que fait le processeur avec ces instructions: soit on joue DureeBip périodes, soit on attend et on se prépare pour recommencer.
A noter que pour +=1 on a le raccourci ++. Un geek aurait écrit `cntPeriode++;`

1.16 Exercice
Faire 3 bips rapides distincts avant d'attendre une seconde.
Il faut ajouter un compteur de bips.

1.17 Une sirène
Pour une sirène montante, on part d'une période initiale et on diminue jusqu'à une valeur limite. Si la période est supérieure à cette valeur limite, on passe à une deuxième partie de programme très similaire. Et on recommence. Non, pour ne pas en avoir plein les oreilles quand on préparera une variante ou lira la suite de la doc, on ajoute un `if` : si le poussoir est pressé, sirène, autrement silence!
On a enfin un programme un peu plus long intéressant à analyser. Il faut choisir les périodes limites. On donne des noms à ces valeurs et on déclare la valeur au début. Fini ces chiffres qui se promènent dans le programme, et on ne sait plus à quoi est associé la valeur 500, par exemple.
Une variable définit quand la sirène monte ou descend. Selon sa valeur 0 ou 1, on saute par dessus les instructions que l'on ne veut pas exécuter.

```
//Sirene.ino  Si on presse, la sirène fait un tour
// Le poussoir sur pin9 est câblé vers le - avec une pull-up
//La haut-parleur sur la pin8 est câblé vers le + avec une résistance de 47-470 Ohm
#define Hp8 8
#define HpOn digitalWrite(Hp8,LOW);
#define HpOff digitalWrite(Hp8,HIGH);
#define Pous9 9
#define Pous9On  !digitalRead(Pous9)
void setup () {
  pinMode (HP8, OUTPUT);
  pinMode (Pous9, INPUT);
}

#define DemiPerMax  2000  // 250 Hz
#define DemiPerMin  100    // 10 kHz
#define Increment   4      // fixe la vitesse d'évolution
int demiPeriode = DemiPerMax ;  // notre variable 16 bits
int descend = 0 ;  // au début la sirène monte

void loop () {
    if (Pous9On) {  // on lance la sirène
         if (descend == 0 ) { // on monte
               digitalWrite(HP8,LOW);
               delayMicroseconds (demiPeriode) ;
               digitalWrite(HP8,HIGH);
               delayMicroseconds (demiPeriode) ;
               demiPeriode -= Increment ;
               if  (demiPeriode < DemiPerMin){
                     descend = 1 ;
               }
         } else { // on monte
               digitalWrite(HP8,LOW);
               delayMicroseconds (demiPeriode) ;
               digitalWrite(HP8,HIGH);
               delayMicroseconds (demiPeriode) ;
               demiPeriode += Increment ;
               if  (demiPeriode > DemiPerMax) {
                     descend = 0 ;
               }
         }
    }
}
```

1.17 Une facilité d'Arduino
Arduino connaît la fonction `tone` avec 2 ou trois paramètres pour osciller une sortie:
`tone (`pin, fréquence`) ;` fréquence en Hz. On arrête le son par `notone();`
`tone (`pin, fréquence, durée en ms`) ;`

A noter que l'instruction lance le son par interruption avec un timer. Si on donne la durée en paramètre, on passe à l'instruction suivante au bout de cette attente, sans action sur le son. Il faut donc soit modifier le son, soit le couper avec un notone(); .
Vous pouvez réécrire les programmes précédents avec ces fonctions?

1.18 Résumé

Une variable est un mot en mémoire que l'on doit nommer avant de l'utiliser par une instruction
`int nom ;` ou `int nom=valeur ;` (pour lui donner une valeur initiale)
On peut modifier le contenu d'une variable (sa valeur) avec une expression arithmétique comme
`nom=33; nom++; nom+=33; vit=4*cpulse;` ou `periode = 1000/frequences`
Le calcul se fait nombres entiers 16 bits si le type est `int`.
Une condition est une valeur binaire égale à 0 (faux) ou 1 (vrai). C'est le plus souvent le résultat d'une comparaison entre 2 valeurs utilisant les signes `== > >= < >=`
Si on écrit `if (0)` ou `if (3==4)` les instructions suivantes ne sont jamais exécutées
Si on écrit `if (compteur < 10)` le bloc d'instructions qui suit sera exécuté si la valeur de la variable compteur de type `int` est de 0,1,2,3,4,5,6,7,8,9

Nous n'avons pas encore parlé de nombres négatifs et il ne faut pas les utiliser sans autre. Ce que nous avons dit sur le type `int` devra être affiné. C'est le problème du C et de tous les langages, il y a beaucoup de richesse dans les concepts et le débutant se perd facilement si on explique tout d'un coup.

L'éditeur d'Arduino est lamentable. Pour modifier, imprimer les programmes, utiliser par exemple Notepad2.

Vous trouvez la suite de ce cours sous www.didel.com/diduino/Cours02.pdf

jdn 120204/0305130123/170210/181012