www.didel.com -- info@didel.com

Bureau d'ingénieur
Pierre-Yves Rochat
pyr@pyr.ch

www.didel.com/coursera/LC2.pdf

Apprendre à programmer avec le LearnCbot

Le texte équivalent pour Energia/Msp430G se trouve sous www.pyr.ch/coursera/LC2-msp.pdf

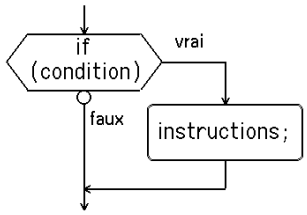
2 Instructions `if`, `for`, `while`, `switch-case`. Terminal série

2.1 Avertissement

Ce document correspond aux exercice libres de la semaine 2 du MOOC EPFL "Comprendre les Microcontrôleurs". Il ne recouvre qu'une partie des notions présentées dans les vidéos, le but de nos exemples étant d'aller en profondeur avec chaque notion.
Le document www.didel.com/coursera/LC1.pdf doit être étudié auparavant pour s'habituer à notre approche "C", différente des tutoriels Arduino.
Si vous êtes débutants, vous devez savoir néanmoins reconnaître les touches spéciales sur votre clavier, créer des fichiers, les sauver sur le disque, les compiler et les exécuter.
Pour se concentrer sur la fonctionnalité des programmes, les sources des exemples sont à disposition sous www.didel.com/coursera/LC2ino.zip.
Gérer vos dossiers et programmes est important, voir www.didel.com/coursera/GestionCroquis.pdf

Vous avez vu les vidéos. En quelques minutes on a parcouru beaucoup d'instructions et on vous a fait croire que c'est simple. C'est effectivement simple quand on a assimilé !
Notre approche pour apprendre le C, les facilités d'Arduino et se familiariser avec les problèmes dits "temps réel", passe par beaucoup d'exercices-exemples simples à modifier.

2.2 Instruction if

`if (condition) {`
`  instructions;`
`}`
	Si la condition est vraie, les instructions sont exécutées	

Exemple 2.21

Reprenons l'exemple LcCopy.ino vu précédemment, renommé A221,ino et que vous trouvez dans le dossier LC2ino que vous venez de créer.

Ajoutez un délai de 1 seconde avant ou après Led1On. (`delay (1000);` vu dans LC1)
Que va-t-il se passer ?
Vérifiez.

```
//A221.ino  Copie P1 sur L1
  . . . définitions et set-up
  void loop ()  {
      if (Pous1On) {
        Led1On;
      }
      if (Pous1Off) {
        Led1Off;
      }
  }
```

Exemple 2.22

On a supprimé le 2e `if` de l'exemple précédent. L'intensité lumineuse est faible quand on presse. Pourquoi?
Si chaque instruction et le retour prend 1 microseconde, la led1 est allumée quel pourcentage du temps quand on pèse ?
Ajouter un délai de 0.1s à différents endroits, et prévoir ce qui va se passer avant d'exécuter.

```
//A222.ino  P1 → Led1, éteint automatique
  . . . définitions et set-up
  void loop ()  {
      if (Pous1On) {
        Led1On;
      }
      Led1Off;
  }
```

Exemple 2.23

Un poussoir allume, l'autre éteint.
Mettre un retard de 2 secondes après Led1On (`delay (2000);`). Quelle conséquence sur le poussoir 2 ?

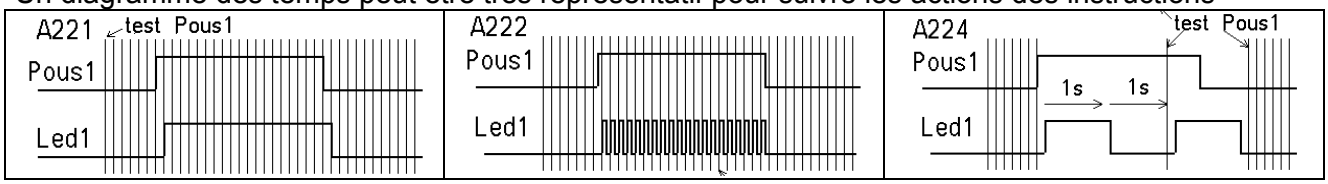
```
//A223.ino  P1 allume, P2 éteint
  . . . définitions et set-up
  void loop ()  {
       if (Pous1On) {
         Led1On;
       }
       if (Pous2On) {
         Led1Off;
       }
  }
```

Exemple 2.24

Clignoter L1 si on pèse sur P1
En pesant plus ou moins fréquemment, l'impulsion ne vient pas toujours. Pourquoi ?

```
//A224.ino  Clignote si P1
  . . . définitions et set-up
  void loop ()  {
     if (Pous1On) {
        Led1On; delay (1000);
        Led1Off; delay (1000);
     }
  }
```

Un diagramme des temps peut être très représentatif pour suivre les actions des instructions



2.25 Complément

Ecrivons l'instruction qui inverse l'état de la Led L1. Cela nous évitera, pour simplement clignoter, d'écrire deux lignes on-off.
Il faut lire l'état de la led, ce que l'on sait faire, c'est comme lire un poussoir :

```
digitalRead (L1)
```

Il faut ensuite inverser cet état avec le signe `!` qui en C signifie inverser la valeur booléenne qui suit :

```
!(digitalRead (L1))
```

Cette expression a une valeur qui vaut 0 ou 1, que l'on peut utiliser pour écrire sur la led :

```
digitalWrite (L1,!digitalRead(L1))
```

On peut alors ajouter la macro Led1Toggle :

```
#define   Led1Toggle  digitalWrite (L1,!digitalRead(L1))
```

et économiser une ligne dans le programme A224 :

```
if (Pous1On) {
      Led1Toggle; delay (1000);
}
```

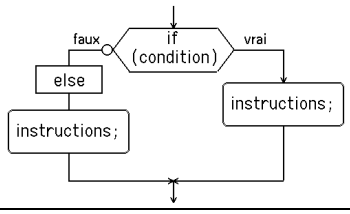
Ce programme n'est toutefois pas le même qu'avant. Observez la Led après chaque relâchement.

A noter que maintenant que le signe d'inversion booléen est connu, on n'a plus besoin de la macro `Pous1Off`. On écrira `!Pous1On`.

2.3 Instruction if - else

```
if (condition) {
   instructions;
}
else {
   instructions;
}
```

Si la condition est vraie, des instructions sont exécutées, autrement c'est un autre groupe d'instructions qui est exécuté.



Exemple 2.31

Ce programme A231 est-il équivalent au programme A221 ?
Comparez les tailles des programmes.
(Arduino prend 500-600 bytes pour se lancer et utiliser les fonctions pinMode delay, etc).

```
//A231.ino  Copie P1 sur L1
  . . . définitions et set-up
  void loop ()  {
       if (Pous1On) {
         Led1On;
       }  else {
         Led1Off;
       }
  }
```

Exemple 2.32

On veut que la Led1 s'allume seulement si on presse sur P1 et P2 simultanément.
Pour exprimer un ET logique entre deux états, le C utilise le signe &&.

Tester le signe || (OU logique) que l'on verra plus tard.

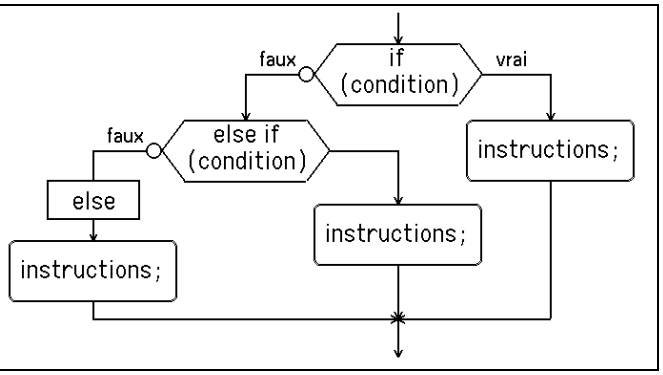
```
//A232.ino  L1 si P1 -ET- P2
. . . définitions et set-up
  void loop ()  {
       if (Pous1On&&Pous2On) {
         Led1On;
       }
       else {
         Led1Off;
       }
  }
```

2.4 Instruction if – else if – else if – else

```
if (condition) {
  instructions;
}
else if (condition) {
  instructions;
}
else if { ... }

else { ... }
```

Cette instruction est peu utilisée. Dans les cas ou il y a plusieurs choix, le "switch case" que l'on verra est plus élégant.



Exemple 2.41

Modifier le programme A241 pour que P1 joue le rôle d'alarme, c'est-à-dire une fois L1 allumé par P1, il faut faire un reset pour éteindre L1
Sauver ce programme sous le nom A241b.ino

Variante: P2 éteint L1 (quittance l'alarme).

Peut-on enlever le else final ?

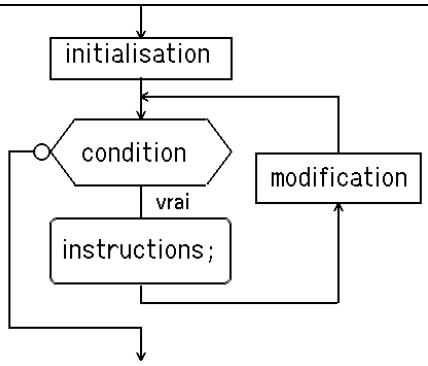
```
//A241.ino  P1 allume L1, P2 allume L2
  . . . définitions et set-up
  void loop ()  {
       if (Pous1On) {
         Led1On;
       }
       else if (Pous2On) {
         Led2On;
       }
       else  {
         Led1Off;
         Led2Off;
       }
  }
```

2.5 Boucle for

```
for (init ; cond ; modif) {
   instructions;
}
```

Exemple: on veut clignoter 20 fois

```
for (int i=0; i<20; i++;) {
   Led1On;  delay (200);
   Led1Off; delay (200);
}
delay (4000);// avant de recommencer
```



La boucle `for` est pratique pour répéter une action un certain nombre de fois. Dans sa forme simple elle utilise une variable locale `i` (par exemple un compteur 16 bits de type int).
Pour répéter une période de 1 ms 100 fois, donc jouer une note, on écrit :

```
    for (int i=0 ; i<100; i++) {
       HPOn; delayMicroseconds(500);
       HPOff; delayMicroseconds(500);
    }
```

Que penser de cette instruction que l'on voit dans certains programme?

```
  for(;;);
```

On initialise rien, la condition est toujours vraie, pas de modification, une instruction vide.
On tourne en boucle indéfiniment sans rien faire ! Comme un `while (1) {}`.

Exemple 2.51

On veut entendre un La normal (440 Hz) sur le Haut-parleur.
On joue le son 2 secondes, et on s'arrête 2 secondes.
On calcule que la demi-période est de 1000000/440/2 = 1136 microsecondes. Il faut répéter 440*2*2 = 1760 fois la demi-période pour que cela dure 2 secondes.
La taille du code est 1282 octets (sur mon PC).

```
//A251.ino
  . . . définitions et set-up
  void loop ()  {
    for (int i; i<1760; i++) {
      HPToggle;
      delayMicroseconds (1136);
    }
    delay (2000);
  }
```

On doit demander au compilateur de faire ces calculs, il fera moins de fautes si le calcul est bien posé!

Dans ce premier cas, le compilateur calcule les constantes qui seront utilisées.
Modifier le programme (A251b n'est pas sur le zip)
Cette écriture est naturellement la seule correcte; on ne doit pas voir des valeurs numériques dans un programme ! (sauf les délais qui sont de l'aide à la mise au point).

Vous avez bien pris l'habitude de sauver sous un nouveau nom chaque variante de programme?

```
//A251b.ino
  . . . définitions et set-up
int dureePeriode = 1000000/440/2;
int nbPeriodes = 440*2*2;
  void loop ()  {
    for (int i; i<nbPeriodes; i++) {
      HPToggle;
      delayMicroseconds (dureePeriode);
    }
    delay (2000);
  }
```

Modifions encore en demandant au processeur (et pas au compilateur) de calculer la période et la durée.
Le compilateur est assez malin pour ne pas faire le calcul chaque fois, se sont des constantes.
Mais si on déclare `int la=440;` et on écrit `for (int i; i<la*2*2; i++) { ..` le calcul se fait chaque fois (la variable pourrait changer) et la durée est augmentée de 50 microsecondes. Faites suivre les deux variantes de boucles for, et vous entendrez la différence!

```
//A251c.ino  2 secondes de "La normal"
  . . . définitions et set-up
  void loop ()  {
    for (int i; i<440*2*2; i++) {
      HPToggle;
      delayMicroseconds (1000000/440/2);
    }
    delay (2000);
  }
```

Exemple 2.52

Programmons une sirène dont le son monte.
Il faut deux boucles `for` imbriquées, pour jouer plusieurs fois la même période avant de la modifier.
NbPeriodes fixe la vitesse d'évolution.
PerMin PerMax les périodes extrêmes.
Modifiez ces valeurs pour écouter des infra et ultrasons.
Ajouter une 2e boucle for pour diminuer la période (`p--` pour diminuer de 1).
Vous pouvez ajouter/soustraire des valeur >1 pour entendre les escaliers de fréquence (augmenter aussi le nombre de périodes répétées)

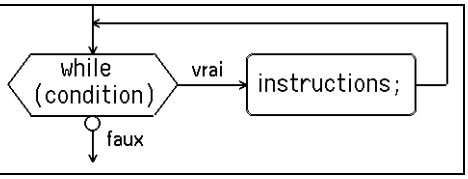
```
//A252.ino  sirene
  . . . définitions et set-up
  void loop ()  {
#define PerMin 500
#define PerMax 1000
#define NbPeriodes 10

void loop ()  {
   for (int p=900; p<1000; p++) {
       for (int i=0; i<NbPeriodes; i++)  {
           HPToggle; delayMicroseconds (p);
       }
   }
}
```

2.6 Instruction while

```
while (condition) {
   instructions;
}
```

Tant que la condition est vraie, les instructions sont exécutées

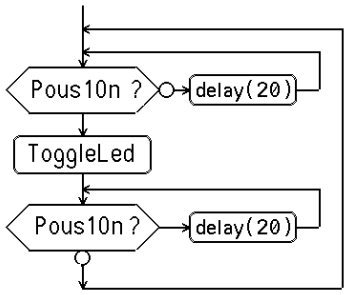


Il faut bien voire la différence entre le `if`, qui teste et passe plus loin, et le `while`, qui boucle tant que la condition n'est pas vraie. On dit que le `while` est "bloquant", car le processeur est bloqué dans la boucle `while()` à attendre que la condition soit vraie.
Si le processeur n'a rien d'autre à faire qu'attendre que l'on presse sur un bouton, ce n'est pas gênant.

Rappelons que `while(1){}` est utilisé pour terminer un programme : le processeur tourne en boucle à ne rien faire d'utile. Il faut faire un reset pour redémarrer le programme.
`for(;;);` a le même effet!

Exemple 2.61

On veut allumer et éteindre une Led avec un seul poussoir. Il faut attendre que l'on presse, puis que l'on relâche.
On remarque le rond d'inversion dans l'organigramme qui veut dire "faux" et qui se traduit par le `!` du C.
Le délai de 20ms évite les rebonds de contact.




```
//A261.ino
 . . . définitions et set-up
void loop ()  {
  while (!Pous1On) {
     delay (20);
  }
  Led1Toggle;
  while (Pous1On) {
     delay (20);
  }
}
```

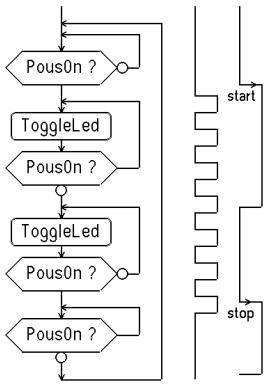
Modifier ce programme pour avoir une minuterie d'escalier : la led s'allume pour 10 secondes à chaque pression.
Remarque: Ce programme a un risque de mauvais fonctionnement. Il peut arriver exceptionnellement que `!Pous1On` soit échantillonné dans un rebond très court qui fait que le `Pous1On` qui suit est aussi vrai. Il n'y a pas d'attente, donc un basculement de trop (programme correct en 2.72).

Pour que cette minuterie soit pratique, il faut pouvoir la réarmer avant les 10 secondes (bloquantes si on écrit delay (10000);. Il faut faire une boucle d'attentes avec 500 délais de 20ms, en testant chaque fois si on a pressé pour réarmer le délai.

Exemple 2.62

On veut maintenant démarrer et arrêter un clignotement avec le poussoir. Pendant le clignotement on ne peut pas se bloquer sur le poussoir. On teste à chaque `LedToggle` si on peut continuer.

Le cahier des charges demande que la Led1 soit éteinte si elle ne clignote pas et que cela ne clignote pas au démarrage.
Modifier le programme pour que ce soit le cas.




```
//A262.ino  clignote on/off
. . . définitions et set-up
void loop ()  {
  while (!Pous1On) {   // attend pression
     delay (10);
  }
  while (Pous1On) {    // attend relachement
     delay (100);
     Led1Toggle;
  }
  while (!Pous1On) {  // attend pression
     delay (100);
     Led1Toggle;
  }
  while (Pous1On) {   // attend relachement
     delay (10);
  }
}
```

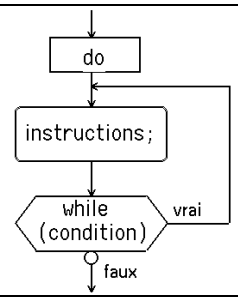
2.7 Instruction do while

```
do {
  instructions;
}
while (condition);
```

Avec le `while()`, si la condition est vraie, on continue sans avoir exécuté les instructions. C'est parfois gênant.

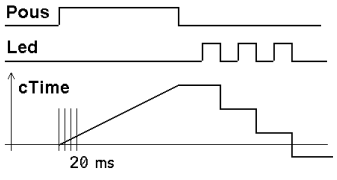
Les instructions sont d'abord exécutées, et ensuite la condition est testée.

Attention aux accolades et `;`



Exemple 2.71

On veut clignoter pour une durée proportionnelle à la durée de l'action sur un poussoir. On compte toute les 20ms pendant l'action.
On divise par 50 pour avoir le nombre d'impulsions. Il faut au moins une impulsion, même si l'action du poussoir est courte.



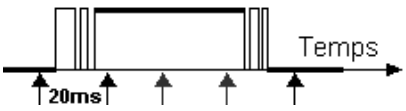
Modifier le programme pour utiliser l'instruction

```
//A271.ino  Cligno variable
 . . . définitions et set-up
int compteDuree;
void loop () {
   while (!Pous1On) {
      delay (20);
   }
   compteDuree = 0;
   while (Pous1On) {
      delay (20);
      compteDuree++;
   }
```

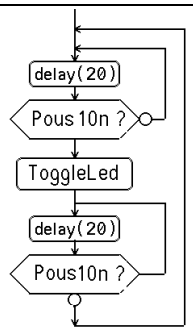
Une variante serait d'ajouter 1 et utiliser un `while()` au lieu du `do..while`.	Led1Toggle en gardant la même fréquence de clignotement.	`compteDuree /= 50;` // division
		`do {`
		`Led1On; delay (200);`
		`Led1Off; delay (200);`
		`compteDuree--;`
		`}`
		`while (compteDuree>0) ;`
		`}`

Exemple 2.72

L'exemple 2.61 a mis en évidence un problème assez subtil. Si on lit un signal avec rebonds, il faut s'assurer qu'il y ait toujours un délai entre 2 décisions.



Le programme 2.62 a-t-il le même problème potentiel? Si oui, il faut le corriger.




```
//A272.ino
 . . . définitions et set-up
void loop ()  {
 do {
    delay (20);
 } while (!Pous1On);
   Led1Toggle;
 do {
    delay (20);
 } while (Pous1On);
}
```

2.8 Switch case

L'idée du switch case est de faciliter la programmation d'une suite de tâche:
Je fais ceci. Quand c'est fini (compteur à zéro par exemple) ou s'il y a un signal qui s'active, je passe à la tâche suivante. Les cas sont numérotés, on verra plus tard comment les nommer. Il ne doit pas y avoir d'instruction bloquante dans un `switch-case` (sauf pour des temps courts).
La fonctionnalité est riche et la syntaxe apparaît compliquée. Dans la forme simplifiée présentée maintenant :

```
byte etat=1;       // on définit une variable cas et on décide que le premier cas est numéroté 1
switch (etat) {  // on se réfère à cette variable qui va prendre différentes valeurs
    case 1:      // pas d'accolade! On aligne les instruction et on termine avec break;
    case 2:      // on écrit  etat=1; par exemple pour retourner à l'état 1
}
```

Exemple 2.81

Reprenons l'exemple 2.61 qui a un état dormant et un état clignotant. Il faut dédoubler ces états pour décoder la transition du poussoir.
On voit que la lisibilité est bonne, les commentaires ne sont pas nécessaires.

Modifier le programme pour que l'on doive presser 2 fois sur le poussoir pour obtenir le clignotement.

5 fois ? – avec une variable compteur, pas avec une boucle `for` bloquante !

```
//A281.ino   clignote on/off avec switch case
 #define L1 4
 #define Led1Toggle  digitalWrite (L1,!digitalRead(L1))
 #define Led1Off digitalWrite (L1,0);
 #define P1 2  // actif à 0
 #define Pous1On (digitalRead (P1)==0) //  actif à 0V
 void setup() {
    pinMode (L1,OUTPUT);
    pinMode (P1,INPUT);
    DDRC = 0xFF;
 }
  int etat=1;
void loop ()  {
 delay (20);  // inutile de le répéter dans chaque cas!
 switch (etat) {
    case 1:       // on attend
       if (Pous1On) {
          etat = 2;
       }
       break;
  case 2:       // on commence tout de suite à clignoter
       Led1Toggle; delay (100);
       if (!Pous1On) {
          etat = 3;
       }
       break;
    case 3:       // on continue jusqu'à ce que l'on presse
       Led1Toggle; delay (100);
       if (Pous1On) {
          Led1Off;
          etat = 4;
       }
       break;
 case 4:       //on attend le relâchement
       if (!Pous1On) {
           etat = 1;
       }
  }
}
```

2.9 Terminal série

Le terminal série sera expliqué dans une vidéo de la semaine 4. On va se limiter ici à sa fonctionnalité couramment utilisée pour afficher des textes et variables, aider à la mise au point. Pour plus de détails voir **www.didel.com/C/Terminal.pdf**

Dans le set-up, il faut dire que l'on va communiquer avec l'écran et préciser la vitesse de transfert. Dans le programme, on lance les affichages.

```
Serial.begin (9600);      // débit standard de 9600 bits/s
Serial.print ("texte");  // affiche une chaine sans saut de ligne
Serial.print (valeur);   // affiche une valeur en décimal
Serial.print (valeur, DEC); Serial.print(valeur, HEX); Serial.print(valeur, BIN);
```

- attention, les zéros non significatifs sont effacés: (0b00010100,BIN) est affiché 10100

```
Serial.println();     // l'affichage est suivi d'un saut de ligne
```

Pour voir l'écran terminal, il faut cliquer sur l'icone "Moniteur série".
Le HP est connecté sur la ligne Tx, et chaque transfert excite le HP avec un bruit qui signale que le téléchargement se fait, mais est désagréable lorsqu'il y a des transferts régulier. Un commutateur permet de désactiver le haut-parleur..

Exemple 2.91 Compter et décompter en 8 bits.

On veut utiliser P1 pour compter et P2 pour décompter. On a vu comment attendre sur un poussoir. Il y en a deux maintenant. Avec des `if`, il faut balayer pour voir quel poussoir est actif, et se bloquer dessus pour attendre le relâchement dans un `while`. On affiche sur le terminal la valeur à chaque action. Comptez et décomptez, observez le passage par zéro.
Le type `short int` correspond à un mot de 8 bits signé. `byte` est un nom équivalent. On inventorie des types dans LC3.pdf.

```
//A291.ino  P1 compte, P2 décompte
  . . . définitions et set-up
short int compteur;

void loop () {
  if (Pous1On) {
     delay (10); compteur++;
     Serial.println (compteur);
     while (Pous1On) {delay (10); }
  }
  if (Pous2On) {
     delay (10); compteur--;
     Serial.println (compteur);
     while (Pous2On) {delay (10); }
  }
}
```

Changeons de type de donnée.
Si on déclare `byte compteur`; on passe de 0 à 255 en décomptant.
Si on déclare `int compteur;` on passe de 0 à -1.
Si on déclare `unsigned int compteur;` on passe de 0 à 65535.
Si on déclare `char compteur;` cela n'affiche rien! Le type char est traité spécialement.
Mais `short int compteur;` affiche –1 quand on décompte de zéro.

Note: C définit aussi des des noms cohérents pour ces types : `int8_t uint8_t int16_t uint16_t`

Exemple 2.92 Saturation

Le programme A291 est modifié pour que la valeur ne dépasse pas 5.
On aurait pu écrire

```
if (compteur > 5) {
   compteur=5;
}
```

Mais si on modifie il faut changer le 5 à 2 endroits !

Ajouter les instructions pour saturer à 0, pour saturer à –1.
Quelle est la réaction si on sature à –1 avec un type "unsigned".

```
//A292.ino  P1 compte, P2 décompte saturé
  . . . définitions et set-up
short int compteur;
void loop () {
  if (Pous1On) {
     delay (10); compteur++;
     if (compteur > 5) {
        compteur--;
     }
     Serial.println (compteur);
     while (Pous1On) {delay (10); }
  }
  if (Pous2On) {
     delay (10); compteur--;
     Serial.println (compteur);
     while (Pous2On) {delay (10); }
  }
}
```

Exemple 2.93

Serial.print supprime les zéros non significatifs selon notre habitude scolaire. Les nombres sont en binaire dans la mémoire. Par défaut, ils sont convertis et affichés en décimal.
Le programme montre comment afficher un texte et un nombre dans différents formats.

Pour afficher les zéros non significatifs, c'est facile d'écrire une boucle `for` .
Le signe << décale à gauche (LC3.pdf)
+= additionne le 2e terme,
<<=1 décale de 1 bit

Remplacer `int val2 = 0x0006;` par `int val2 = 0xC006;`
Bizarrement notre type `int` est transformé en type `long` avec extension du signe, puisque 0xC006 est négatif.

```
//A293.ino   Affichage de nombres différents formats
  . . . définitions et set-up
void setup() {
Serial.begin(9600);
}
int val = 45 ;
int val2 = 0x0006;
void loop() {
  Serial.print(val);
  Serial.print("  bin ");
  Serial.print(val,BIN);
  Serial.print("  hex");
  Serial.println(val,HEX);
  Serial.println("com\npte ");
  for (byte i=0;i<15;i++) {
     Serial.print(i);
  }
  Serial.print('\n');   // comme Serial.println();
  Serial.print(val2);
  Serial.print(" bin ");
  Serial.print(val2,BIN);
  Serial.print(" hex ");
  Serial.println(val2,HEX);
  for (int i=0; i<16;i++)  {
    if ((val2 & 0x8000) == 0) {
      Serial.print("0");
    }
    else {
      Serial.print("1");
    }
    val2<<=1;  // décale à gauche
  }
  while (1) {}  // reset pour recommencer
}
```

2.10 Résumé

On a vu les instructions

```
if (condition)  {  instructions;  }
for (init ; cond ; modif) { instructions; }
while (condition)  {  instructions;  }
do { instructions; }  while (condition) ;
switch (variable) case: . . .
```

Leur bonne utilisation nécessite de la pratique. Reprogrammez nos exemples et inventez-en d'autres.

jdn 140505/150428/151021