Apprendre à programmer avec le LearnCbot

Le texte équivalent pour Energia/Msp430G se trouve sous www.pyr.ch/coursera/LC3-msp.pdf

3 Variables, tableaux, ports, opérations logiques

3.1 Préalable

Ce document correspond aux exercice libres de la semaine 3 du Coursera "Microcontrôleurs" Il complète en profondeur une partie des vidéos. Les sources des programmes sont à disposition sous www.didel.com/coursera/LC3ino.zip

3.2 Variables simples

Une variables est une case de 8, 16 ou 32 bits en mémoire, qui représente des mots positif ou signés. L'effet de certaines instructions n'est pas le même pour chaque type, il faut donc bien choisir le type de la variable et en tenir compte. Il y a plusieurs noms pour chaque type, ce qui est troublant. Nous utiliserons les noms en gras. La norme C recommande les noms de la première colonne. On peut assigner un autre nom à un type (typedef), la portabilité de nos programmes n'est donc pas un problème; si `byte` n'est pas accepté par un compilateur, il suffit d'ajouter `typedef unsigned char byte;` au début du programme.

uint8_t	8 bits entier positifs 0..255	**byte** unsigned char
int8_t	8 bits entier signés -128..0..127	**char** signed short int
uint16_t	16 bits entier positifs 0..65535	**unsigned int**
int16_t	16 bits entier signés -32768..0..32767	**int**
uint32_t	32 bits entier positifs 0 .. 1069 millions	unsigned long
int32_t	32 bits entier signés	**long**
float	nombre flottant 32 bits signés < $3.4*10^{38}$	float
	1 bit vrai/faux	bool

Exemples : `byte var1 ; int vit= -4 ; char i, c= 'a' ;`

Une variable doit être déclarée avant d'être utilisée. On peut lui donner une valeur initiale. Par défaut, la valeur est zéro.
On peut lister plusieurs variables de même type sans répéter le type et le `;` final.

3.3 Tableaux

Un tableau de variables se déclare comme une variable: il suffit de dire entre `[]` combien de cases mémoire réserver. Les numéros des cases (index) commencent par 0. Par exemple, `int table [3];` réserve 3 mots de 16 bits en mémoire. On peut ensuite écrire par exemple `table[2]++;` pour incrémenter la 3e variable du tableau.
Attention, dans `int table [3];` 3 est une taille exprimée dans le langage courant. Les 3 cases de la table ont comme nom `table [0]; table [1]; table [2];`. Si par erreur on écrit `table [3];`, le compilateur accepte, mais la place n'a pas été réservée en mémoire, une autre variable peut s'y trouver et on a un méchant bug !
Si on n'a pas défini la longueur de la table, on peut l'obtenir avec `sizeof (nomDeLaTable);`

Exemple 3.31

On veut calculer la moyenne des valeurs dans une table. On fait la somme et on divise.
En faisant la somme, il ne faut pas qu'il y ait dépassement de capacité. En divisant, il faut se souvenir que c'est une opération longue, sauf si on divise par 2,4,8,16,. . .
Dans le programme 331, on voit comment initialiser une table. Il n'est pas nécessaire de dire la taille; le compilateur peut la calculer (pratique

```
//A331.ino Moyenne
void setup()    {
  Serial.begin(9600);
}
byte table [] =
{23,43,66,51,11,24,92,7,33,15};
longTable = sizeof (table);
unsigned int total = 0;
unsigned int moyenne;
void loop () {
```

pour des grandes tables).
Modifiez des valeurs, le nombre d'éléments.

On pourrait additionner combien d'éléments avant dépassement et erreur du résultat?

```
    for (int i=0; i<longTable; i++) {
      total += table [i] ;
    }
    moyenne = total/longTable ;
    if (total%longTable) >= longTable/2)
  { moyenne++ }
    Serial.print (moyenne);
    while (1) {}
  }
```

(Le programme sur le zip est plus complet)

Exemple 3.32

Dans une table similaire, on veut connaître le minimum et sa position (son index dans la table).
On part d'une valeur minimum la plus grande possible, selon le type. Si la première valeur est plus petite, on substitue et mémorise l'index.

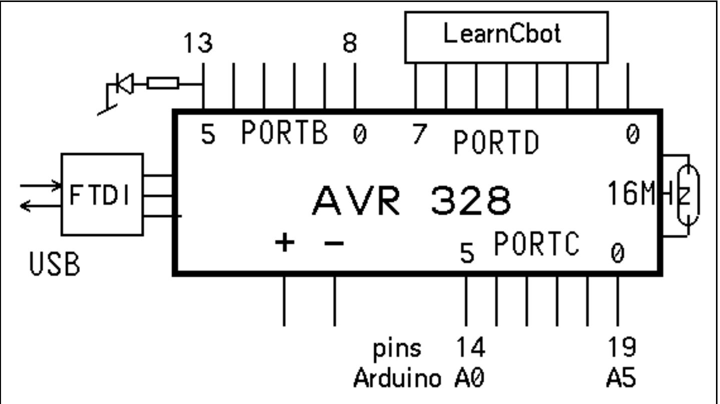
Modifier ce programme pour afficher le maximum et son index.

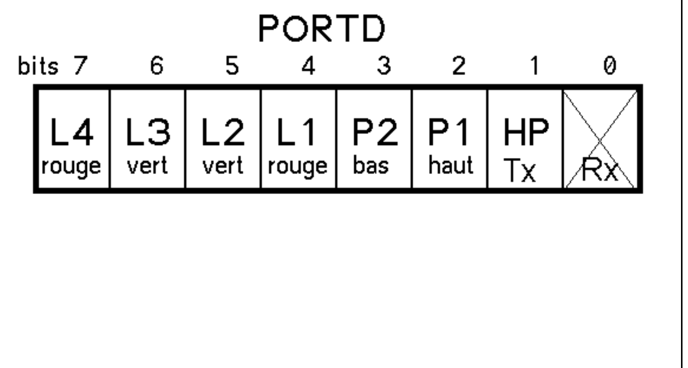
```
//A332.ino Minimum et sa position
void setup()   {
  Serial.begin(9600);
}
byte table [] =
{23,43,66,51,11,24,92,7,33,15};
int longTable = sizeof (table);
byte minimum = 255;
byte indexMinimum ;
void loop() {
  for (byte i = 0; i<longTable); i++) {
    if (minimum > table [i]) {
      minimum = table [i];
      indexMinimum = i;
    }
  }
}
```

3.4 Ports

Comprenons comment le processeur interagit avec les pins, et en particulier avec celles du LearnCbot.

La carte Arduino/Diduino a un processeur AVR Atmega328 avec son quartz 16 MHz et son interface USB. Le processeur a trois ports:B,C,D, accédés comme des variables 8 bits.
Dans chaque port, les bits sont numérotés de 0 à 7, 0 à 5 pour les ports B et C, qui sont incomplets.
Arduino a numéroté ces pins de 0 à 19.
Le LCbot est câblé sur le port D. Sur ce port, les numéros de pins et de bit sont identiques.





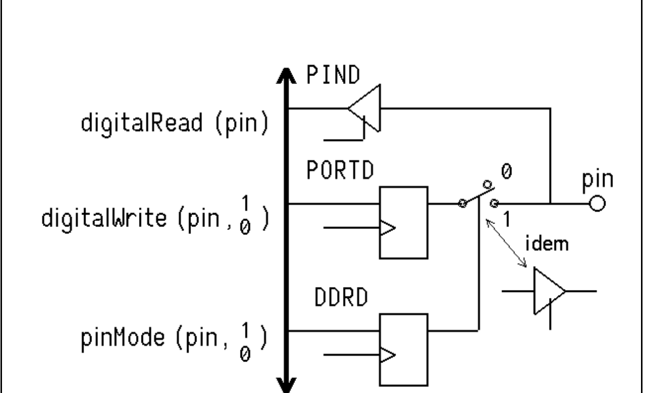
PORTD

bits 7	6	5	4	3	2	1	0
L4 rouge	L3 vert	L2 vert	L1 rouge	P2 bas	P1 haut	HP Tx	Rx

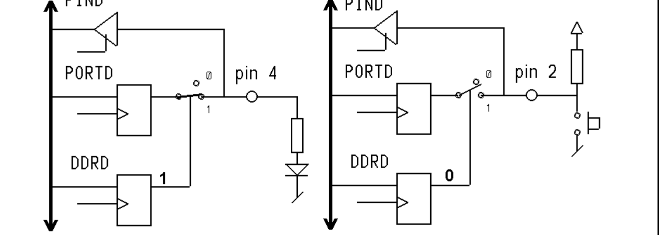
Pour le LearnCbot, ce port D doit être programmé avec 5 pins en sorties et 2 pins en entrées. Les pins 0 et 1 sont réservées pour la communication série avec le circuit FTDI. Le HP partage la pin Tx, ce qui n'est pas gênant pour les exercices. Juste un peu bruyant quand la communication série travaille avec le haut-parleur activé.

Pour dire que certaines pins sont en entrées et d'autre en sortie, il faut configurer le port et cela se fait dans le set-up. Le port D est un ensemble de trois "registres" qu'Arduino a caché, mais qu'il est temps de comprendre. Analysons le schéma logique pour une pin.

Le PORTD mémorise l'information que l'on veut écrire sur un pin. Mais cette information ne vas pas nécessairement passer sur la pin.
La fonction Arduino `digitalWrite()` écrit dans le PORTD, mais pour que l'information passe sur la pin, il faut avoir fermé l'interrupteur en écrivant un "1" dans le registre de direction DDRD (Data Direction Register D). La fonction Arduino `pinMode()` agit sur ce registre DDRD.
Pour lire une pin, on passe par le registre PIND, qui lit toutes les pins. La fonction Arduino `digitalRead()` lit la pin indiquée.



Donc pour allumer la Led de la pin 4 il faut mettre un 1 dans les bascules correspondantes des PORTD et DDRD. La lecture sur PIND doit relire un "1". Si on relit un "0", il y a court-circuit (cc) !
Pour lire la pin2 il faut mettre un 0 dans DDRD.
Que se passe-il si on met un"1" ?



1er cas: la bascule PORTD bit 2 est à "0". Le courant de la résistance est absorbé par cette bascule, comme quand on agit sur le poussoir. On lit toujours un zéro.
2e cas: la bascule PORTD bit 2 est à "1". On lit un "1", mais ce n'est pas la résistance qui fournit le courant, c'est la bascule. Attention, si on presse sur le poussoir, on court-circuite la sortie de la bascule qui fournit du courant avec une résistance interne de 100 Ohm environ. On lit un zéro, mais un courant de 50 mA circule dans la bascule et le poussoir; le transistor de sortie chauffe et peut se détruire si le court-circuit dure quelques secondes. On pourrait ajouter une résistance de limitation de courant, mais elle dégraderait d'autres caractéristiques. Alors, apprenons à programmer plutôt que de compliquer les schémas.

Exemple 3.41

On veut vérifier qu'il n'y a pas de cc sur la sortie qui commande la led L1.
On écrit "1" – on doit relire "1"
On écrit "0" – doit relire "0"
Si c'est mauvais on allume la led13

Compléter l'exemple pour afficher sur le terminal ce que l'on écrit et ce que l'on relit.

Pour tester, il est permis d'utiliser un fil et de faire court-circuit de la pin L1 avec le Gnd/0V ou le 5V pendant une seconde au plus.

```
//A341.ino Test court-circuit
#define  Led13On digitalWrite (13,1)
#define   L1 4
#define   L1On   digitalWrite (L1,1)
#define   L1Off  digitalWrite (L1,0)
#define   L1Etat digitalRead(L1)
void setup() {
  pinMode (13,1); // Led13 sortie
  pinMode (L1,1); // sortie
  L1On;
  if (L1Etat != 1) { Led13On; }
  L1Off;
  if (L1Etat != 0) { Led13On; }
}
void loop () {
  L1On;
  if (L1Etat != 1) { Led13On; }
  L1Off;
  if (L1Etat != 0) { Led13On; }
  delay (100);
}
```

Exemple 3.42

Clignoter les 4 leds en écrivant directement dans le portD est facile et garanti la simultanéité.

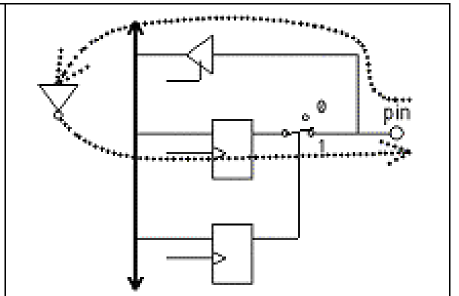
Modifier ce programme pour faire clignoter alternativement les Leds vertes et rouge.

```
//A342.ino Clignote les 4 leds
void setup() {
  DDRD = 0b11110000;  // bits 7..4 en sortie
}
void loop () {
  PORTD = 0b11110000; delay (500);
  PORTD = 0b00000000; delay (500);
}
```

Ce que l'on vient de programmer est brutal. On a forcé en entrée les pins 0 à 3 sans se demander s'il y avait un effet secondaire sur Tx Rx en particulier. Pour n'agir que sur certains bits, il faut avoir bien compris les opérations logiques.

3.43 Remarque

On a vu que pour inverser une pin, il faut écrire
`digitalWrite (pin,!digitalRead(pin)).`
La figure montre ce que fait le processeur: lire, inverser, écrire.
Si on veut inverser tout le port, on code `PORTD = ~PIND;`
On verra plus loin comment agir sur une seule pin.
On remarque la différence de notation entre l'inversion booleenne, signe `!`, et l'inversion bit à bit, complément d'un mot binaire (variable ou constante) qui utilise le signe `~`. ***Ne pas confondre !***



Pour ceux qui maîtrisent les opérations logique, le document
www.didel.com/C/OperationsLogiques.pdf
est une bonne révision, le document est plus compact que ce qui suit (il n'a pas d'exemples).

3.5 Opération logique booléennes

Les fonctions logiques ou booléennes agissent sur un ou deux bits et sont résumées ci-dessous. Des symboles logiques sont utilisés dans les schémas électroniques.

Not !	And &&	Or ||	Xor (^^ pas dispo)
Le résultat a la valeur inverse de l'entrée.	Le résultat vaut 1 si les deux entrées sont à 1	Le résultat vaut 1 si l'une des deux entrées est à 1	Le résultat vaut 1 si les deux entrées sont différentes
Inverseur NOT NOT Inverseur	Porte ET AND	Porte OU OR	Ou exclusif XOR

Ceci s'applique dans des expressions conditionnelle, dans un `if()`,un `while()`, etc. On peut écrire `if (toto || titi) {instr;}`. L'instruction sera exécuté s'il y a au moins un bit à 1 entre les 2 mots `toto` et `titi`. Donc elle n'est pas exécutée que si les deux mots sont à zéro. Si la valeur du mot est 0 (booléen faux), le bit est 0. Si la valeur est différente de zéro (vrai), le bit vaut 1.

Exemple 3.51

L'exemple A292.ino compte et décompte. On veut maintenant que la Led1 s'allume seulement si on pèse sur P2 quand on atteint la valeur zéro.

Le programme ci-contre est faux! L'idée est juste, mais on s'est bloqué dans le précédent `if` avec le `while (PousOn)`. Vérifiez ce qui se passe.

C'est dans cette attente qu'il faut intervenir.
A vous de corriger et tester le programme

```
//A351.ino  P1 compte, P2 décompte saturé
  . . . définitions et set-up
short int compteur;
void loop () {    (ce programme doit être corrigé)
  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); }
  }
  if (Pous2On && (compteur==0)) {
     Led1On;
  }
  else {
     Led1Off
  }
}
```

3.6 Opération logique bit par bit (bitwise)

Ces opération sont effectuées par le processeur sur des mots de 8 bits. Dans l'unité arithmétique et logique (ALU) on trouve 8 portes ET en parallèle, etc.
L'opération et dite "bit-à-bit" (bitwise). Les opérateurs sont
`~` (not) `&` (and) `|` (or) `^` (xor). On met aussi dans cette catégorie les décalages `>>` `<<` .

Not ~	And &	Or |	Xor ^
a = 0b10110101 ~a = 0b01001010	a = 0b10100101 m = 0b00011111 a&m = 0b00000101	a = 0b10100101 m = 0b00011111 a|m = 0b10111111	a = 0b10100101 m = 0b00011111 a^m = 0b10111010

Attention, `-a` complément arithmétique de `a`, n'es pas égal à `~a`, complément bit à bit de a

Dans les exemples ci-dessus, la lettre `m` évoque la notion de masque. Souvent, une partie des bits dans une variable, sur un port, doit être masquée, seuls d'autres bits sont significatifs, ou le masque est vu comme une passoire pour garder certains bits.

Exemple 3.61

On veut faire le XOR de deux mots de 8 bits sans utiliser le `XOR`. Pour chaque bit la règle dit l'un ou l'autre, pas les deux. Exemple en 4 bits:

```
        1100 a          1100 a
        1001 b          1001 b
   OR   1101 x   AND   1000 y
```

Le AND du 1er résultat avec l'inverse du 2e donne

```
               1101   x
               0111  ~y
          AND  0101   a^b
```

Modifier le programme pour ne pas utiliser la variable `vXor`, puis ne pas utiliser la variable `vAnd`, puis aucune variables (cela fera plusieurs parenthèses imbriquées dans le Serial.println!).

```
//A361.ino  Xor sans Xor
  . . . définitions et set-up
#define Mot1 0b10100101
#define Mot2 0b11110000

byte vAnd,vOr,vXor;

void loop () {
  Serial.println (Mot1,BIN);
  Serial.println (Mot2,BIN);
  vOr  = Mot1 | Mot2;
  vAnd = Mot1 & Mot2;
  vXor = vOr & ~vAnd;
  Serial.println (vXor,BIN);
  for (;;);
}
```

! serial.Print enlève les zéros non significatifs

3.7 Décalages

On peut décaler un mot binaire de type entier 8,16,32 bits à droite avec les opérations "shift right", signe `>>` ou à gauche avec "shift left", signe `<<`.

a = 0b10110101 a>>3 = 0b00011010	a = 0b10110101 a<<4 = 0b01010000	a = 0b10110101 a>>1 = 0b01101010	a = 0b10110101 a>>10 = 0b00000000

Exemple 3.71

Ce programme décale à droite de 4, puis à gauche de 4. Quel est le résultat?

Ecrire le programme plus simplement avec une seule opération logique.

```
//A371.ino  Décalages
void setup () {
  Serial.begin (9600);
}
#define Mot 0b10100101
byte var = Mot;
void loop () {
  Serial.println (var,BIN);
  var >>= 4; //  idem à var = var>>4;
  var <<= 4;
  Serial.println (var,BIN);
  while {};
}
```

Exemple 3.72

On veut allumer les Leds L1 .. L2 dans l'ordres en agissant directement sur le PORTD, sans faire une boucle for avec réinitialisation tous les 4 décalages.
Ici, la boucle teste après chaque décalage si la partie Leds du PORTD est vide, et effectue la réinitialisation.

Ecrire `PORTD = etatLed;` fonctionne ici, mais cela forcerait les 4 bits de poids faibles

```
//A372.ino  Chenillard L1 .. L4

void setup () {
  DDRD = 0b11110000;
}
#define MaskLeds 0b11110000
#define EtatInitial 0b00010000 //L1 allumé
byte etatLeds = EtatInitial;
void loop () {
 delay (500);
 PORTD |= (etatLeds&MaskLeds); // force les "1"
```

à zéro sans se préoccuper si cela a une conséquence sur ces pins.
La nécessité de ces deux instructions de masquage est bien expliquée dans .www.didel.com/C/OperationsLogiques.pdf

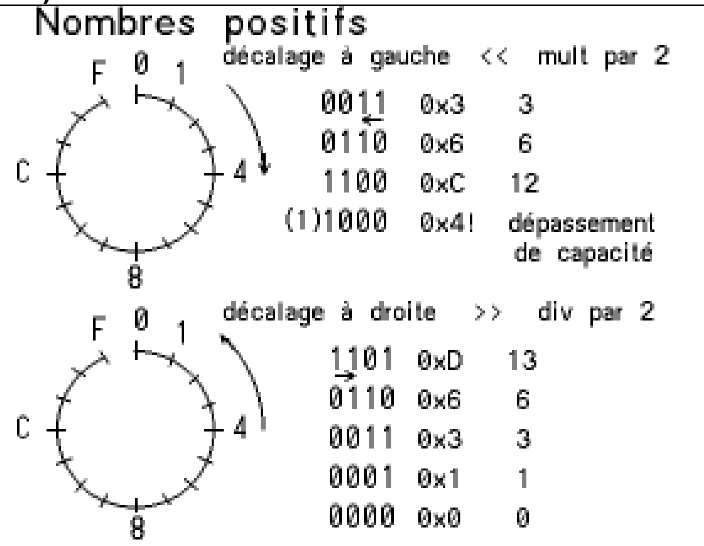
```
PORTD &= ~(etatLeds&MaskLeds); // force les "0"
  etatLeds <<= 1;  // décale à droite de 1 position
  if ((etatLeds & MaskLeds) == 0) {
     etatLeds = EtatInitial ;
  }
}
```

3.8 Décalage arithmétique

Décaler à droite revient à diviser par 2, mais on va voir qu'il ne faut pas utiliser les décalages pour faire des opérations arithmétiques. Mais c'est une bonne occasion de mieux comprendre les problèmes de dépassement de capacité que le compilateur C ne peut pas signaler.

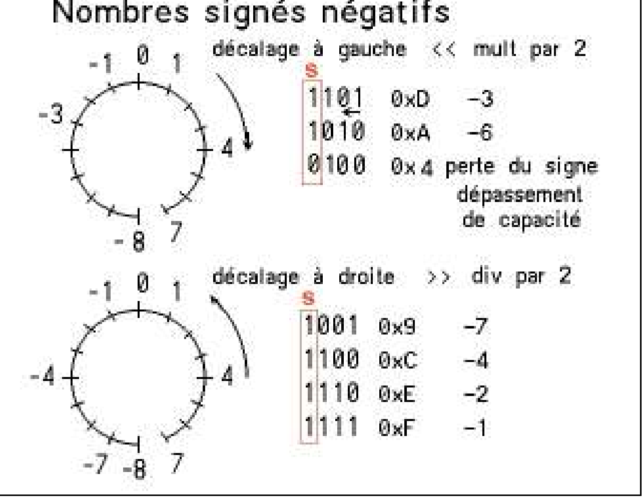
3.81 Décalage de nombres binaires (positifs)

Expliquons avec des nombres de 4 bits, qui peuvent représenter des nombres de 0 à 15, 0 à 0xF en hexadécimal.
Décaler à gauche multiplie par 2, mais il y a rapidement dépassement de capacité puis résultat nul.
Décaler à droite divise par 2, mais il y a rapidement un résultat nul



3.82 Décalages de nombres arithmétiques (signés)

La compréhension est plus délicate pour les nombres signés, lorsqu'ils sont négatifs.
Le bit de poids fort (MSB Most Significant Bit) est le bit de signe, toujours 0 si le nombre est positif, 1 si négatif..
Avec 4 bits, on peut représenter des nombres signés de –8 à +7.
Décaler à gauche multiplie par 2, mais pour les nombres négatifs, il y a extension du signe.
Décaler à droite divise par 2, pour les nombres négatifs, il y a conservation du signe.



Le compilateur gère les décalages selon le type de donnée et il faut éviter de décaler des types signés sans bien savoir ce que l'on fait. Si on multiplie ou divise par 2, 4, 8, . . le compilateur utilise des décalages, plus rapides que la division.

3.9 Positionner un bit 1<<n

L'opérateur de décalage est souvent préféré pour décrire un mot binaire.
Si on écrit "1" le compilateur voit le nombre binaire `0b00000001`.
Pour `1<<3` il calcule `0b00001000`. `1<<5|1<<0` est équivalent à `0b00100001` .
Pour décrire les masques associés à des périphériques ou des modes il faut donner des noms aux lignes (bits) et utiliser cette notation. Par exemple, on a 2 sorties sur un port : un moteur sur le bit 3 et un servo sur le bit 5. La valeur à donner au registre de direction (processeur AVR ou MSP) est `0b00101000`. C'est du charabia! `0x28` ou `40` (décimal) est pire. Il faut déclarer

```
#define bMot = 3
#define bServo = 5
mPort = 1<<bMot | 1<<bServo ; // mPort sera utilisé pour assigner le registre de direction
```

A noter qu'écrire `mPort = (1<<3)|(1<<5);` est aussi du charabia ! Autant écrire `00101000`.

Exemple 3.91

<table>
<tr>
<td>Reprenons l'exemple simple qui copie P1 sur L1, mais en utilisant des définitions C.
On remarque la notation <code>bP1</code> pour nommer le bit 2 dans le <code>PORTD</code>. On avait nommé <code>P1</code> la pin Arduino, qui "par hasard" a le même no.

Ajouter les définitions pour P2 et utiliser P1 pour allumer, P2 pour éteindre.
Recharger le programme A223.ino qui fait la même chose et comparer les tailles mémoire.</td>
<td>

```
//A391.ino Copie P1 sur L1  478 octets
#define  bP1  2
#define  bL1  4
#define  Pous1On  PIND & (1<<bP1)
#define  Led1On   PORTD |=  (1<<bL1)
#define  Led1Off  PORTD &= ~(1<<bL1)
void  setup () {
   DDRD  |= (1<<bL1) ; // led en sortie
}
void loop () {
   if (Pous1On) { Led1On; }
   else { Led1Off; }
}
```

</td>
</tr>
</table>

3.10 Action sur des bits

Arduino/C définit 3 fonctions qui modifient un bit, sans toucher les bits d'à côté et qui font donc ce que l'on vient de voir.

mettre le bit à1	`bitSet    (var,no);`	`var |=  (1<<no);`
mettre ce bit à 0	`bitClear  (var,no);`	`var &= ~(1<<no);`
tester si ce bit vaut 0 ou 1	`bitRead   (var,no)`	`var &  (1<<no)`

Attention, le `;` à la fin de la 3e expression n'a pas été oublié. Les deux premières lignes concernent des actions, des instructions. La troisième intervient dans une condition
`if (bitRead (var,no)) {  }` ou `if (var & (1<<no)) {  }`.
Ces fonction ne sont pas connues des "vrais" compilateurs C, mais elles sont faciles à définir pour que le programme transporté soit compatible.

3.11 Surveiller un changement

Le rôle fréquent d'un microcontrôleur est de réagir à un changement de l'environnement. Toutes les x millisecondes, le contrôleur peut mémoriser l'état et x millisecondes suivante il compare, active des variables qui déclenchent des actions.
Prenons comme exemple surveiller les 2 poussoirs P1 P2 pendant que l'on clignote L1. On peut faire des `if` sur chaque poussoir, ce qui n'est pas très élégant si on a un clavier.
On programme une boucle de 20 ms dans laquelle on lit l'état des 2 touches sur le PORTD et clignote par exemple tous les 10 fois.. 20 ms plus tard, on relit le PORTD et on compare avec la valeur précédente qui a été mémorisée. Il faut mettre en place un joli jeu de variables et de masques. A chaque ligne de l'exemple simple qui suit, vous verrez si vous avez bien assimilé ce qui précède! Accrochez-vous, c'est le B A BA du C !

Exemple 3.111

<table>
<tr>
<td>La variable prevPous mémorise l'état des 2 poussoirs. Un masque ne garde que les 2 bits qui nous intéresse. Il faut initialiser prevPous dans le setup, autrement le premier passage dans le programme pourrait signaler un changement.
Il faut immédiatement copier le nouvel l'état dans l'ancien, mais il faut aussi les comparer avant mise à jour. Cela oblige à créer une variable pour ne pas relire deux fois l'état des poussoirs, qui peut changer chaque microseconde.
S'il y a changement, on clignote une autre led en modifiant le masque qui sélectionne la led – admirez l'élégance de la solution !

.Modification: au démarrage, clignotez L1 et L4 une fois.</td>
<td>

```
//A3111.ino Surveille 2 poussoirs
#define bP1 2 // bit 2 sur PORTD
#define bP2 3
#define MaskPous ((1<<bP1) | (1<<bP2))
..#define bL1 bL2 bL3 bL4   (pas la place ici!)
byte maskLedOn =(1<<bL1);  // var masque initial, changera
void setup () {
  DDRD = (1<<bL1)|(1<<bL2)|(1<<bL1)|(1<<bL2);
  prevPous = PIND & MaskPous;
}
byte nowPous, prevPous, diffPous;  // variables globales
byte cntCli=10;  //demi-periode clignotement 10x20ms
void loop () {
  delay (20);
  if (cntCli-- == 0) {
    cntCli=10;
    PORTD ^= maskLedOn;  // inverse les leds sélectionnées
  }
  nowPous = PIND & MaskPous;
  diffPous = nowPous ^ prevPous ;  // bits différents?
  prevPous = nowPous;   //mise à jour pour le prochain cycle
  if (diffPous != 0) {   // si différence
    if (diffPous & (1<<bP1)) {
       maskLedOn =(1<<bL2);
    }
    if (diffPous & (1<<bP2)) {
       maskLedOn = (1<<bL3);
    }
    PORTD = 0;  // pourquoi?
  }
}
```

</td>
</tr>
</table>

Résumé
La compréhension des opérations logiques est très importante pour maîtriser le C temps réel et se préparer à des applications qui gèrent plusieurs capteurs.
Le débutant hésite souvent sur les bons signes à utiliser. Notre feuille résumée compacte se trouve sous www.didel.com/C/Resume.pdf

3.12 Fichier de définitions

On s'embête à lister au début de chaque programme les définitions associées au matériel utilisé. Rassemblons toutes les définitions qui concernent le LearnCbot dans un seul fichier, que le concept de "croquis" d'Arduino permet de cacher dans une librairie personnelle.
Cette définition du matériel peut se faire a la "Arduino" en raisonnant sur les pins et en appelant des fonctions qui prennent du temps et de la place mémoire. Regrouper ces définitions qui dépendent de la carte microcontrôleur (ici Arduino ou MSP430G) rend les programmes portables, ce qui est le premier but du C.
Les définitions qui concernent le LearnCbot peuvent s'écrire en "Arduino" ou en "C", ce qui ne change rien dans le programme principal, à part le temps d'exécution et la taille du code.

```
//LCArduiDef.h LearnCbot  definitions Arduino
#include <Arduino.h>
#define  Led13  13
#define   Led13On  digitalWrite (Led13,1)
#define   Led13Off digitalWrite (Led13,0)
#define  Led13Toggle digitalWrite(Led13,!digitalRead(Led13))
#define HP   1
#define   HPOn    digitalWrite (HP,1)
#define   HPOff   digitalWrite (HP,0)
#define   HPToggle  digitalWrite (HP,!digitalRead(HP))
#define L1   4
#define L2   5
#define L3   6
#define L4   7
#define   Led1On   digitalWrite (L1,1)
#define   Led1Off  digitalWrite (L1,0)
#define   Led1Toggle  digitalWrite (L1,!digitalRead(L1))
#define   Led2On   digitalWrite (L2,1)
#define   Led2Off  digitalWrite (L2,0)
#define   Led2Toggle  digitalWrite (L2,!digitalRead(L2))
#define   Led3On   digitalWrite (L3,1)
#define   Led3Off  digitalWrite (L3,0)
#define   Led3Toggle  digitalWrite (L3,!digitalRead(L3))
#define   Led4On   digitalWrite (L4,1)
#define   Led4Off  digitalWrite (L4,0)
#define   Led4Toggle  digitalWrite (L4,!digitalRead(L4))
#define P1   2
#define P2   3
#define   Pous1On  (!digitalRead(P1))
#define   Pous2On  (!digitalRead(P2))

void LcSetup () {
  pinMode (1,1); pinMode (2,0); pinMode (3,0);
  pinMode (4,1); pinMode (5,1);
  pinMode (6,1); pinMode (7,1);
}
```

```
//LCDef.h LearnCbot  definitions C
#include <Arduino.h>
#define bLed13 5 // bit 5 PORTB (pin 13)
#define  Led13On  bitSet   (PORTB,bLed)
#define  Led13Off bitClear (PORTB,bLed)
#define  Led13Toggle PORTB ^= 1<<bLed
#define bHP 1 // PORTD
#define   HPOn    bitSet  (PORTD,bL1)
#define   HPOff   bitSet  (PORTD,bL1)
#define   HPToggle   PORTD ^= 1<<bL1
#define bL1   4 // PORTD actifs à 1
#define bL2   5
#define bL3   6
#define bL4   7
#define  Led1On   bitSet   (PORTD,bL1)
#define  Led1Off bitClear (PORTD,bL1)
#define  Led1Toggle  PORTD ^= 1<<bL1
#define  Led2On   bitSet   (PORTD,bL2)
#define  Led2Off bitClear (PORTD,bL2)
#define  Led2Toggle PORTD ^= 1<<bL2
#define  Led3On   bitSet   (PORTD,bL3)
#define  Led3Off bitClear (PORTD,bL3)
#define  Led3Toggle PORTD ^= 1<<bL3
#define  Led4On   bitSet   (PORTD,bL4)
#define  Led4Off bitClear (PORTD,bL4)
#define  Led4Toggle PORTD ^= 1<<bL4
#define bP1   2 // PORTD
#define bP2   3
#define   Pous1On   (!digitalRead(bP2))
#define   Pous2On   (!digitalRead(bP2))

void LcSetup () {
  DDRD |= 0b11110010; //HP L1 L2 L3 L4 out
}
```

3.12 Librairie et fichiers de définition

On ne doit garder dans le programme principal que la partie propre à l'application. Les définitions permettent d'oublier le câblage, caché dans une librairie.
Il faut distinguer les librairies locales, appelées par un `#include "MaLibrairie.h"`
et les librairies globales, mises à disposition de toute la collectivité et rassemblées dans le dossier librairie d'Arduino et appelées par un `#include <lib.h>`. Les librairies globales sont difficiles à créer. On utilisera seulement des librairies de définition locales, inclues chaque fois dans le croquis du programme. Les programmes deviennent très clairs.

```
TestLcArduiDef   LcArduiDef.h
//TestLcArduiDef.ino  Copie P1 sur L1
#include "LcArduiDef.h"

void setup () {
  LcSetup ();
}
void loop () {
   if (Pous1On) { Led3On; }
   else { Led3Off; }
}
```

```
TestLcDef   LcDef.h
//TestLcDef.ino  Copie P1 sur L1
#include "LcDef.h"

void setup () {
  LcSetup ();
}
void loop () {
   if (Pous1On) { Led3On; }
   else { Led3Off; }
}
```