Apprendre le C avec le LearnCbot

Le texte équivalent pour Energia/Msp430G se trouve sous www.pyr.ch/coursera/LC5-msp.pdf

Chap 5 – Tables, moyennes, timers, EEPROM

Les sources des exemples sont à disposition sous www.didel.com/coursera/LC5ino.zip

5.1 Tableaux

Un tableau de variables (array) correspond à une suites de positions mémoires de même dimension numérotées à partir de 0. La dimension du tableau ou table est le nombre d'éléments

`int table [4];` réserve la place pour 4 éléments dont le numéro (index) est 0,1,2,3

On peut définir le contenu de ces élément s'ils doivent être différents de 0:

`int table [4] = {57, -3, , 4};  // ` c'est mieux d'écrire le 0 pour l'index 2!

Si le type est char, le tableau peut être formé de caractères ASCII

```
char message[] = "hello";
```

On voit qu'il n'a pas été nécessaire de compter les lettres, le compilateur sait le faire et peut vous donner son résultat avec l'instruction `sizeof (message)`.

On peut déclarer des tableaux multidimensionnels. A 2 dimension, on visualise les données en ligne et colonne. Le premier index est la ligne, le second la colonne.

```
int table [3][2] = { {1,2},
                     {10,20},
                     {100,200}
                   };
```

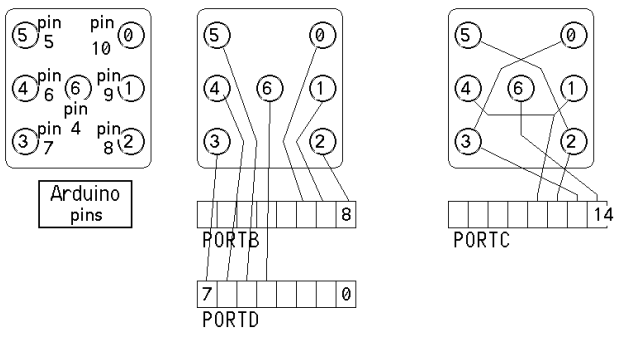
Exemple 5.11

Pour calculer la somme des nombres de la table précédente, il faut deux boucles `for`.
On remarque que compactifier le tableau 2D sur une ligne n'est pas une bonne idée.

```
//A511 Tableau 2D
void setup() {
  Serial.begin(9600);
}
int table [3][2] = {{1,2},{10,20},{100,200}} ;
int result =0;
void loop () {
  for (byte i=0; i<3; i++) {
    for (byte j=0; j<2; j++) {
      result += table [i][j];
    }
  }
  Serial.println(result);
  while (1) {}
}
```

5.2 Tableaux de constantes

Comme pour les variables, le prefix `const` dit au compilateur de mettre les valeurs dans les instructions qui utilisent ces valeurs. Il n'y a pas de place perdue en mémoire.
La table pour un dé électronique par exemple dépend du câblage. On préfère optimiser le câblage plutôt que la facilité d'écriture de la table.



On peut faire une table avec des numéro de pins Arduino.

Exemple 5.21

Pour initialiser les 4 leds du LCbot et la led 13, formons un tableau avec leurs numéros.
Une boucle `for` balance ces numéros dans des pinMode qui interprètent les

```
//A521 Initialisation à la Arduino
  byte pinsOut [] = {4,5,6,7,13};
void setup () {
  for (byte i=0; i<sizeof(pinsOut); i++) {
     pinMode (pinsOut [i],1);
  }
}
```

constantes comme des numéros de pins. Exécuter 10000 fois ces instructions du setup avec un ToggleLed en fin de boucle. Quel est le temps d'exécution du setup?. En agissant directement sur le registre, cela prend 0.4 microsecondes.

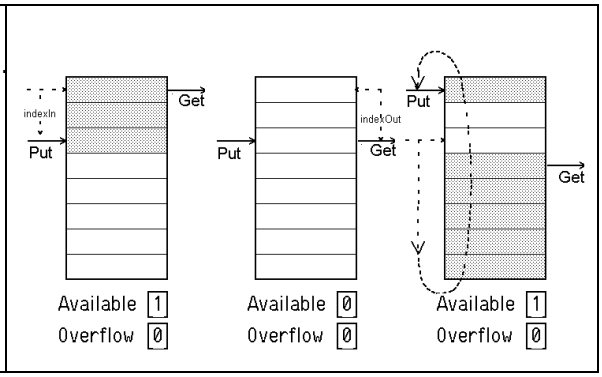
```
void loop () {
 for (int i=0; i<10000, i++)  {
    for (byte i=0; i<sizeof(pinsOut); i++) {
    pinMode (pinsOut [i],1);
 }
  Led13Toggle;
}
```

Ce type de programmation est intéressant dans le monde Arduino; il permet de passer d'une carte à l'autre sans se préoccuper du microcontrôleur. La correspondance entre le no de pin et le bit sur un port est faite par une fonction dans la librairie, d'où le choix "Type de carte" à sélectionner dans un menu Arduino.
Avec un autre microcontrôleur que le Atmega328, il faut pour programmer en C retrouver le schéma de câblage pour agir sur les bons registres. C'est 20 fois plus rapide (vérifiez, on a vu les instructions) mais ce n'est plus portable dans le monde Arduino, où on vous conseillera d'acheter un processeur plus rapide avec plus de mémoire.

5.3 Mémoire tampon

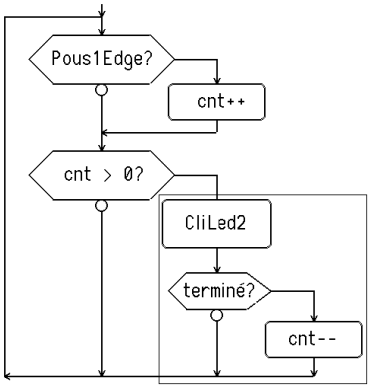
Une mémoire tampon reçoit de l'information et livre cette informations dans le même ordre, mais pas à la même vitesse. On parle de premier entré-premier sorti (FIFO FirstInFirstOut), à ne pas confondre avec une pile (stack) qui est un LIFO (LastInFirstOut).
On a vu un exemple avec le terminal: les caractères tapés sont mémorisés dans un tampon et transféré à une vitesse différente sur l'écran.
Un tampon est un tableau. L'astuce est dans les index qui pointent l'information qui entre et l'information qui sort. On parle de tampon circulaire, car le bas de la zone mémoire continue en haut: si le pointeur arrive en bas, on le mets en haut. On a deux index, en face l'un de l'autre si le tampon est vide. On ne s'intéresse ici pas au nombre de mots en attente, mais on peut l'obtenir par différence circulaire.

On déclare un tableau tampon de longueur `LongTampon`. et deux pointeurs `indexIn indexOut`.
La fonction `PutData()` écrit et incrémente circulairement le pointeur. La fonction `GetData()` lit s'il y a quelque chose à lire. Deux flags `Available` et `Overflow` dépendent de la distance des pointeurs, qui sont des bytes utilisés en booléens (0 faux 1 vrai).
On peut écrire si `Overflow ==0;`
On peut lire si `Available ==1;`



Exemple 5.31

Choisissons un exemple simple pour comprendre la synchronisation entre deux tâches asynchrones.
On presse irreguliérement sur un poussoir.
Chaque pression doit générer un clignotement de 1 seconde (500ms allumé, 500 ms éteint).
Une solution est de faire communiquer les 2 tâches, non bloquantes évidemment, par une variable globale `cnt`.
Chaque pression augmente le compteur. Chaque clignotement complet décompte.
La fonction `PousEdge`() est dérivée de ce qui a été vu en 4.64, `flagPous` est remplacé par le compteur qui joue le même rôle.



Le clignotement non bloquant a été vu en 4.61. C'est un peu plus compliqué ici parce qu'il faut maintenant voir 3 états: allumé 0.5s, éteint 0.5s, terminé. Un switch-case résout le problème.

On voudrait envoyer `cnt` sur le terminal. Où faut-il le faire? Cela prend 5ms, acceptable.

```
//A531 Cligote asynchrone
```

```
void CliLed2 () {  // appelé toutes les 20ms 25+25 cycles
  byte etat;
  switch (etat) {
    static byte cntLed2; {
    case 0:
      Led2On;
      if (cntLed2++ > DurOn) {
         cntLed2 = 0;
         etat = 1;
      }
      break;
    case 1:
```

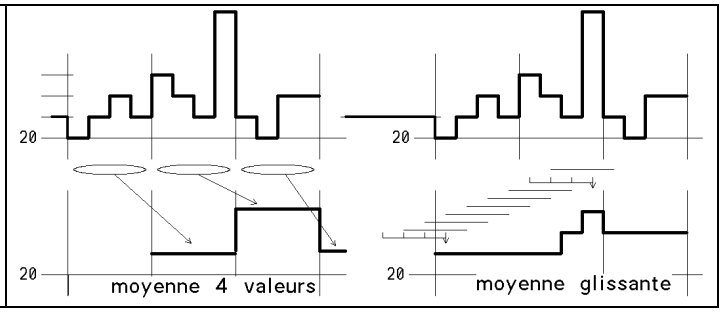
```
#include "LcDef.h"
void setup()  {
  LcSetup ();
}
int cnt=0 ; // variable globale
void Pous1Edge () {
  volatile byte prevPous1;
  if (Pous1On && (prevPous1==0)) {
    cnt++;
    Led1Toggle;
  }
  prevPous1=Pous1On;
}
#define DurOn 1000/20
#define DurOff 1000/20
```

suite colonne suivante page précédente

```
      Led2Off;
      if (cntLed2++ > DurOff) {
         cntLed2 = 0;
         etat = 2;
      }
      break;
    case 2:
        cnt--;
        etat = 2;  break;
  }
}
void loop () {
  delay (20);
   Pous1Edge ();
  if (cnt>0) {
    CliLed2();
  }
}
```

5.4 Moyenne glissante

On a vu (3.31) comment moyenner les valeurs d'un capteur. La fréquence des mesures utilisables est divisée par le nombre d'échantillons intervenant dans la moyenne. La moyenne glissante se calcule à chaque mesure. A l'initialisation, on fait comme si les dernières mesures étaient identiques à la première mesure.



Exemple 5.42

A défaut de capteur, introduisons des valeurs au terminal (voir 4.51) et affichons la moyenne glissante. Sans initialisation, on voit que les 3 premières moyennes sont fausses.

Compléter le setup pour que la première moyenne soit égale à la première valeur entrée. On voit que les variables concernées ont déjà été déclarées avant le setup, puisqu'elle y sont utilisées.

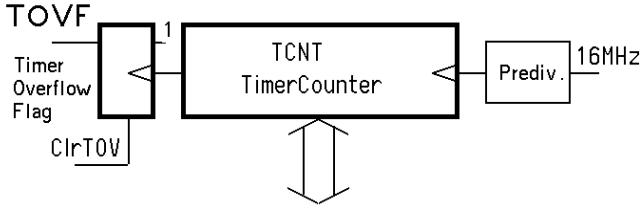
Comment faire pour que cette initialisation soit faites dans la fonction? Indication: ajouter un flag qui est à zéro au premier appel de `GetMoyGlis`().

```
//5.42 Moyenne glissante
#include "LcDef.h"

int taGlis [4];
byte iTag;  // index à taGlis
void setup()  {
  LcSetup ();
  Serial.begin(9600);
}

// fonction GetMoyGlis (valeur);
//Rend la moyenne des 4 valeurs précédentes
int GetMoyGlis (int vv) {
  int mm = 0;
  taGlis [iTag++] = vv; // le ++ se fait après l'écriture
  if (iTag > 3) {iTag = 0; }
  for (byte i=0; i<4; i++) {
    mm += taGlis [i];
  }
  return mm/4;
}
int valeur, moyenne;
void loop() {
  while (Serial.available() > 0) {
    valeur = Serial.parseInt();
    Serial.print(valeur); Serial.print("  ");
    moyenne = GetMoyGlis (valeur);
    Serial.println(moyenne);
  }
}
```

5.5 Timers

Dans sa forme la plus simple, un timer est un registre compteur que l'on peut lire et écrire comme une variable. Il compte à une fréquence qui est un diviseur de la fréquence du processeur. Le compteur, appelé `TCNT`, a une longueur limitée, 8 ou 16 bits, et est associé à une bascule appelée `TOVF` (Timer Overflow Flag).

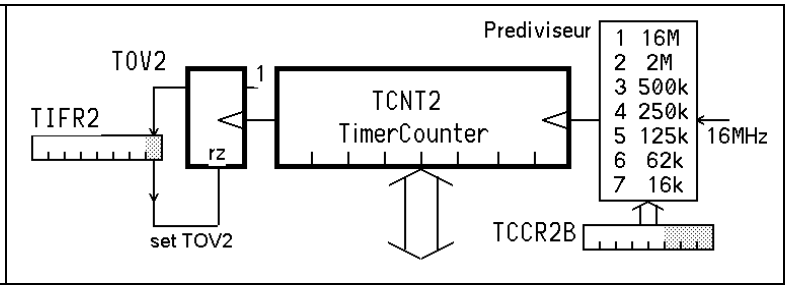
Le timer compte sans cesse. La bascule `TOVF` signale le dépassement de capacité (quand le compteur passe de 0xFF à 0) et permet d'étendre le comptage par logiciel. Si `TOVF` est à un, on ajoute 1 dans un compteur soft et on remet à zéro la bascule.



On peut à chaque dépassement charger le compteurs à une valeur quelconque, pour que le TOVF revienne plus vite. En chargeant 200, l'overflow aura lieu 56 impulsions plus tard. Mais attention, tout ne se fait pas simultanément. Le microcontrôleur doit surveiller TOVF directement (ou par interruption, on verra plus tard). Quand il agit pour réinitialiser le timer, quelques impulsions de comptage ont passé, qu'il faut soustraire. On chargera 56-2 au lieu de 56 par exemple pour tenir compte de ce temps de réponse.

5.51 Timer 2 de l'Atmega328

Ce timer 8 bits est piloté par 3 registres. On peut lire et écrire le compteur 8 bits `TCNT2`. Il avance à la fréquence définie par le contenu des 3 bits de poids faible du registre `TCCR2B`. (Timer Control Register timer 2). Les autres bits sont à 0 dans ce mode simple.
Le registre TIFR2 contient le flag `TOV2`.



La bascule est remise à zéro en écrivant un 1 dans le bit correspondant de `TIFR2`: `bitSet(TIFR2,TOV2)`. Cela peut surprendre, mais le câblage des registres de flag est spécial: on ne peut par écrire dans TIFR2, on peut seulement agir sur le reset de ses flags.

Exemple 5.52

Clignotons de nouveau pour changer!
Si TOV2 est actif, on toggle la Led1 et le HP.
Le compteur est réinitialisé à 16 pour que la fréquence des TOV2 soit 1 kHz.
Un post-diviseur soft clignote la Led2 à 1Hz.

Corriger les paramètres pour avoir comme prédiviseur `TCCR2B = 0b00000100;` et garder le même timing pour les Leds et le HP.

Programmer une sirène en changeant la valeur initialisant le compteur-timer à chaque cycle. Il ne faut naturellement pas modifier le timer à chaque cycle, pour que l'évolution soit lente.

```
//A552 Clignote via le timer2
#include "LcDef.h"
void setup () {
  LcSetup ();
  TCCR2A = 0;
  TCCR2B = 0b00000111;   // clk/1024 16kHz 62,5us
}

int cntLed;
void loop () {
  if ( TIFR2 & (1<<TOV2) ) {
     TCNT2 = -16 ;     // pour 1 kHz
     bitSet (TIFR2,TOV2);
     Led1Toggle;
     HPToggle;

     if (cntLed++ > 500) {   // pour période 1Hz
        cntLed = 0;
        Led2Toggle;
     }
  }
}
```

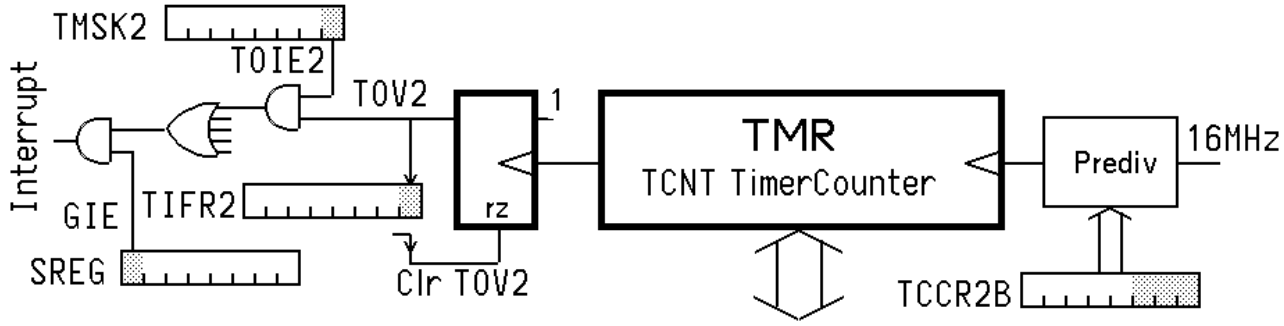
5.6 Principe des interruption

Une interruption ressemble à un appel de fonctions déclenché par un flag, lui-même activé par une entrée du microcontrôleur ou une condition sur un registre timer ou autre. C'est un peu plus compliqué parce que cela arrive n'importe quand dans le programme. Il faut sauver l'état des registres sur une pile et les rétablir en fin de routine d'interruption. La fonction (on utilise plus fréquemment le terme de routine) appelée par un signal d'interruption commence toujours par interdire les interruptions, et dans le cas des AVR, le flag qui a déclenché l'interruptions est remis à zéro automatiquement.
Un bit dans un registre, activé par la fonction Arduino `sei();` permet d'autoriser les interruptions, `cli();` permet de les ignorer quand le programme principal doit respecter des timings précis, ou que l'on change des conditions liées aux interruptions.
Il y a en général plusieurs sources d'interruption (timers, entrées du processeur, série, etc). Chaque source a un flag, et un vecteur d'interruption propre, c'est à dire une adresse pour la routine à exécuter. Arduino a des noms réservés pour les routines d'interruption, par exemple `ISR(TIMER2_OVF_vect)` (ISR signifie Interrupt Service Routine).

5.7 Interruptions du Timer2

On voit sur la figure que toutes les demandes d'interruption viennent sur une porte OU et sont éventuellement bloquées pas le signal `GIE`, activé par `sei();`
Les interruptions individuelles sont chacunes autorisée par un bit "interrupt enable" qui laisse passer la demande du flag actif. Pour le Timer2, `TOIE2` et `GIE` doivent être actifs pour que, au moment ou `TOV2` passe à 1, la routine `TIMER2_OVF_vect` soit appelée. Tous ces

noms sont connus du compilateur, il y a dans les fichiers du dossier Arduino des milliers de `#define` liés à tous les bits et registres du microcontrôleur.



Les timers ont une fonctionnalité très riche. On en reparlera dans le prochain chapitre.

Exemple 5.71

Pour clignoter par interruption, les instructions de l'exemple 5.52 sont reportées dans la routine d'interruption du timer. Le programme principal ne fait plus rien. S'il clignote une autre led à la vieille mode, il est ralenti de ~5 microsecondes à chaque interruption, ceci toutes les millisecondes avec ce qui a été programmé, soit 0.5% du temps.

Que se passe-t-il si le programme principal et l'interruption clignotent la même led à la même fréquence? Essayez. En physique, on parle d'interférences.

```
//A571.ino Clignote par interruption timer
#include "LcDef.h"
void setup () {
  LcSetup ();
  TCCR2A = 0; //default
  TCCR2B = 0b00000111;  // clk/1024
  TIMSK2 = 0b00000001;  // TOIE2
  sei();  // autorise les interruptions
}
volatile unsigned int cntLed;
ISR (TIMER2_OVF_vect) {
  TCNT2 = -16;  // pour 1 kHz
  if (cntLed++ > 500) {
    cntLed = 0;
    Led1Toggle;
  }
}
void loop () {
   Led2Toggle;
   delay (500);
}
```

Exemple 5.72

Dans le programme principal, on veut pouvoir lancer et arrêter le clignotement par interruption. On définit un flag qui agit comme un interrupteur on-off qui commanderait une lampe qui clignote.

Ajouter des instructions dans la routine d'interruption pour que la Led1 soit éteinte quand elle ne clignote pas.

```
//A572.ino Clignote par interruption timer
#include "LcDef.h"
void setup () {
... comme A571.ino
}
int cntLed;
volatile byte flagCligno;
ISR (TIMER2_OVF_vect)  {
  TCNT2 = -16;  // pour 1 kHz
  if (flagCligno) {
    if (cntLed++ > 500) {
      cntLed = 0;
      Led1Toggle;
    }
  }
}
void loop () {
   flagCligno = 1;  delay (2000);
   flagCligno = 0;  delay (3000);
}
```

5.8 Interruption directe (External interrupt)

Tous les microcontrôleurs peuvent créer une interruption si une ou plusieurs pins changent d'état. Des ou-exclusifs sont câblés sur des ports pour reconnaître un changement, ce qui facilite par exemple l'interface avec un clavier.
Le Atmega328 a deux entrées avec une logique sensible au changement, sur les pins 2 et 3 (P1 et P2). On retrouve un flag, un mask pour autoriser les interruptions et un registre de mode. Pour plus de détails, voir www.didel.com/C/ArduinoInterruptPins.pdf .

Arduino propose une fonction qui fait tout et que l'on a avantage à utiliser si le signal n'a pas de rebonds..
La fonction `attachInterrupt(pin,function,mode)` a trois paramètres.

`pin` 0 ou 1 selon la pin 2 ou 3 (3 ou 2 selon la carte)

`function` le nom de la fonction appelée
`mode` `0,1,2,3 ou` LOW, CHANGE, RISING, FALLING

Ce qui est appelé pin n'est donc pas le numéro de la pin, mais le no du flag. La correspondance avec les pins dépends de l'AVR. Pour le 328, le paramètre 0 est associé à la pin 2, poussoir P1. C'est l'inverse sur d'autres cartes.
La fonction est le nom de la fonction à exécuter.
Le mode agit sur des inverseurs et ou-exclusif entre l'entrée et le flag d'interruption.

Exemple 5.81

Clignotons en passant par les interruptions créées par P1, à l'instant ou l'on presse, c'est à dire quand le signal sur processeur passe de 1 à 0 (le 0 est actif sur le poussoir).
Le mode est donc 3 ou `FALLING`.
L'interruption réagit dans la microseconde, le Toggle led aussi, donc s'il y a un nombre impair de rebonds avant le contact, il n'y aura pas de basculement. S'il y a un nombre impair de rebonds au relâchement, il y aura basculement.

```
//A581 Toggle par interruption P1
#include "LcDef.h"
void setup () {
  LcSetup ();
  attachInterrupt(0,CliUneFois,FALLING);
}
void CliUneFois() {
   Led1Toggle;
}
void loop () {
}
```

On voit que le poussoir a parfois des rebonds, qui sont reconnus même s'ils sont très courts (~5 us).

5.9 Entrées analogiques

Les microcontrôleurs ont des entrées analogiques multiplexées sur le port C dans le cas du Atmega328, donc sur les pins 14 à 19, connues aussi sous les noms A0 à A5.
A l'intérieur du 328, il faut préparer plusieurs registres avant de lancer la conversion et attendre qu'elle soit finie. Arduino cache cette complexité avec la fonction analog.Read () qui rend une valeur de 0 à 1023 (10 bits) pour une tension entre 0 et la tension d'alimentation.
Ce n'est pas nécessaire de déclarer la pin en entrée, Arduino force la pin en entrée et désactive la pull-up (sauf erreur) à chaque lecture. Cette lecture est bloquante (~120 us).

Si vous avez un breadboard, enlevez le LearnCbot et prenez l'un des innombrables tutoriaux sur Arduino pour jouer avec un pot ou une LDR.

5.10 EEPROM

On peut stocker des bytes en EEPROM et les retrouver après coupure de courant.
L'écriture est lente (10ms) mais la lecture est aussi rapide que la lecture d'une variable.

Exemple 5.101

Les instructions AVR 328 pour écrire en EEPROM sont compliquées, pour diminuer la probablilté d'une écriture intempestive si le programme déraille.
Arduino propose une librairie facile à utiliser.
`EEPROM.write (ad,data);` a deux paramètres, l'adresse de 0 à 255 et le contenu 8 bits, aussi entre 0 et 255.
`EEPROM.read (ad);` rend le contenu à l'adresse donnée.
Le programme ci-contre initialise un contenu dans le setup. Commentez cette partie à la 2e exécution et coupez l'alimentation.
La boucle demande de taper l'adresse 0 1 2..9 au terminal (suivi d'un retour à la ligne).
`Serial.read` rend le code ASCII du chiffre, il faut soustraire le code du 0 pour avoir l'adresse binaire.
Modification: afficher le contenu complet avec une boucle `for`.

```
//A5101.ino Eeprom   On écrit 00 11 22 33
//  en 0 1 2 3 . . . 9  et on relit en tapant l'adresse
#include "LcDef.h"
#include <EEPROM.h>

byte ad=0,data=0;
void setup () {
  LcSetup ();
  Serial.begin(9600);

  for (byte i=0; i<10; i++) {
    EEPROM.write (ad,data);
    ad++; data +=11;
  }
}

void loop () {
 if (Serial.available() > 0) {
   ad = Serial.read();
   ad -= '0';   // ou code du 0 = 48 décimal = 0x30
   data = EEPROM.read (ad);
   Serial.print(data); Serial.print("  ");
 }
}
```