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

www.didel.com/diduino/Cours03.pdf

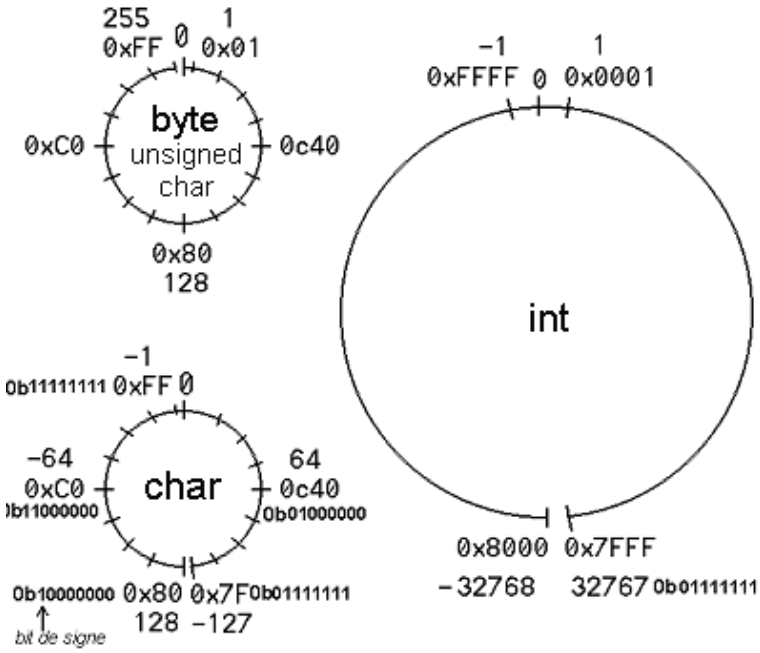
Cours Arduino/C 3ème partie – Variables et tableaux

Les types de données des variables ont été introduits en 1.14. Il est temps de préciser les choses.

3.1 Nombres positifs et négatifs

Les mots binaires dans les ordinateurs ont une taille limitée. Si on dépasse la capacité suite à une opération, on se retrouve au début (comme pour un compteur de kilomètres d'une voiture). La représentation sur un cercle aide à bien comprendre.

Codés en 8 bits comme avec les premiers microprocesseurs, on peut représenter des nombres de 0 à 255, notés en hexa 0x00 à 0xFF. Si on ajoute 1 à 255, le résultat est zéro, et un flag de dépassement de capacité s'active.
On peut aussi raisonner autour du zéro, et voir des valeurs positives et négatives. Le cercle est ouvert en bas, tous les positifs à droite, dont le bit de poids fort est nul, tous les négatifs à gauche. Les nombres sont signés, et c'est le type par défaut, qui correspond à nos habitudes scolaires.
Le type `unsigned int` (16 bits non signés, de 0 à 65535) n'est pas représenté dans la figure.
Il existe aussi des `long int` de 32 bits et des nombres flottants (avec codage d'un exposant). En temps réel, on évite ces nombres qui saturent le processeur !



Les types `int` (signed int) et `unsigned int` ont le même comportement jusqu'à 32767 si on ne fait pas de soustractions. En déclarant comme on l'a fait `int duree;` pour utiliser ensuite `delay(duree);` ce n'était en fait pas correct, il fallait déclarer `unsigned int duree;`. Le compilateur accepte l'instruction `delay(-1);` il prépare la valeur 0xFFFF égale à 65535 et le délai sera maximum, car la routine delay considère que les nombres donnés sont de type unsigned.
Ces notions de types et conversion de type sont délicates. A vous de les creuser lorsque cela sera nécessaire.
Vous l'avez deviné en regardant la figure. 0x annonce un nombre hexadécimal et 0b un nombre binaire parfois B avec Arduino). Dans la mémoire, même les nombres décimaux sont convertis en binaire par le compilateur.

3.2 const int ou #define

Arduino définit les numéros de pins avec par exemple `int Led8=8;` ou `const int Led8=8;`
On préfère ici écrire `#define Led8 8` sans `=` et sans `;` à la fin.
Le `#define` déclare un nom à une suite de caractères que le compilateur substituera.

On peut définir plusieurs instructions, par exemple dire au robot d'avancer, c'est à dire activer un bit et désactiver l'autre

```
#define  Avancer  Mot0Pos=On; Mot0Neg=Off
```

Mais cette écriture compacte n'est pas recommandée, il est préférable d'écrire une fonction (voir Cours04 ou http://www.didel.com/coursera/LC4.pdf).

3.3 Niveaux dans la fonctionnalité

Prenons l'exemple de la musique. On a vu (1.12) que pour générer une note, il faut répéter un certain nombre de fois une oscillation de période adéquate.

On verra dans Cours04 comment transformer la boucle de programme en une fonction et écrire plus simplement Bip (periode,duree) ;
Pour chaque note, il faudra définir ces deux paramètres.
Ce serait plus simple d'écrire note (DO): ce qui suppose une table qui donne les périodes et durées pour DO, RE, etc.
Si on veut jouer un morceau il faudrait pouvoir écrire Joue(AuClairDeLaLune); en se référant à une liste de notes préparée en mémoire.
Le processeur ne connaît que des nombres, c'est le compilateur qui doit gérer les noms donnés aux listes et tableaux.

3.4 Enumération

Associe des noms à une valeur mémorisée dans une variable. enum est un nom réservé suivi entre accolade d'une suite de symboles qui se voient attribuer les valeurs 0, 1, 2.. qui sont rangées dans la variable dont le type est unique, donc pas spécifié.
Ceci est donc une aide à la documentation.

```
enum { DO, RE, MI } notes ;
```

Le compilateur mémorise que `notes` a 3 valeurs (0,1,2) et s'il voit dans le programme le nom MI, il établit la correspondance avec la valeur 2 et le nom `notes`.

3.5 Tableau

Un tableau de variables se déclare comme une variable: il suffit de dire combien de cases mémoire réserver. Par exemple, `int table [3] ;` réserve la place en mémoire et on peut ensuite écrire par exemple `table[2]++;` pour incrémenter la 3e variable du tableau (ah, ces mauvaises habitudes prises à l'école).
Si notre table contient des périodes de notes, et que l'on a énuméré ces notes pour le compilateur, on peut écrire `periode = table[MI];` et utiliser cette période dans une boucle for pour jouer la note. La lisibilité du programme y gagne énormément.
Ce serait pratique d'initialiser cette table de périodes de notes quand on la déclare. Il suffit d'écrire `int table [3] = {PerDO,PerRE,PerMI };`, les valeurs `PerDO` etc sont définies au début avec des `#define PerDO 347`

3.6 Variables locales et globales

Une variable dont on a besoin tout au long du programme est une variable globale et elle doir être déclarée au début.
Mais on a vu des boucles `for` qui ont besoin d'un compteur, traditionnellement appelé `i` et le `i` n'est pas utilisé ailleurs. Il peut être déclaré localement avant de l'utiliser, voir même dans la boucle qui l'itilise:

```
for (byte i=0 ; i< 10 ; i++) {. . . }
```

Ceci gagne de la place en mémoire, et prépare à une programmation modulaire, avec des modules que l'on peut facilement passer d'un programme à l'autre.

3.7 Exemple: un peu de musique

Les fonctions seront décrites en détail dans Cours04.pdf. On va toutefois en utiliser du format le plus simple, avec une utilisation aussi naturelle que `delay(duree);` . Procédons par étapes..

Le DO (ne pas écrire do qui est réservé) de la 4e octave a une fréquence de 523 Hz (wiki/Note_de_musique) La période en microsecondes est de (1000000/523).
On peut laisser le compilateur calculer, ou sortir sa calculette: 1912
Pour simplifier, donnons à chaque note la même durée de 300 oscillations (en fait, le produit de la période et du nombre de périodes devrait être constant). La fonction note pour jouer ce DO utilise un paramètre local p pour la période, c'est dans la partie loop exécutée que la durée sera attribuée.

```
// note1.ino
#define HP 13
#define PerDO 1912
int per ;

void setup()
{
  pinMode(HP, OUTPUT);
}
```

```
void note (int p)
{
  for (int i=0 ; i<300 ; i++ )
  {
    digitalWrite(HP, HIGH);
    delayMicroseconds(p/2);
    digitalWrite(HP, LOW);
    delayMicroseconds(p/2);
  }
}
```

```
void loop ()
{
  per = PerDO;
  note (per) ;
  delay (1000);
}
//joue la note toutes les secondes
```

Formons maintenant une table avec les demi-périodes des notes.
Commençons par énumérer les notes `enum { DO, RE, MI } notes ;`
avant de former le tableau des périodes `int taPer[3] = {1912, 1702, 1518} ;`
Donc `taPer[DO]` contient la période du DO et le programme de test note2.ino joue 3 notes

```
// note2.ino
#define HP 13
enum { DO, RE, MI } notes ;
int taPer [3] = {1912, 1702, 1518};

void setup()
{
  pinMode(HP, OUTPUT);
}
```

```
void note (int j)
{
  for (int i=0 ; i<300 ; i++ )
  {
    digitalWrite(HP, HIGH);
    delayMicroseconds(taPer[j]/2);
    digitalWrite(HP, LOW);
    delayMicroseconds(taPer[j]/2);
  }
}
```

```
void loop ()
{
   note (DO);
   delay (100);
   note (RE);
   delay (100);
   note (MI);
   delay (1000);
}
```

Pour tester la gamme, on peut aussi jouer les 3 notes dans une boucle for: (programme note3.ino)

Note: avec le Diduino, pour éviter d'entendre le son pendant que l'on programme, on câble un poussoir en série avec le buzzer.
Avec le DdRobot, le buzzer est câblé par défaut sur la pin 13 et on peut tester une moustache par exemple pour décider d'écouter le son.

```
void loop ()
{
   for (int j=0 ; j<3 ; j++ )
   {
       note (j);
       delay (100);
   }
}
```

On remarque que dans la fonction note, le paramètre est l'index dans la table de notes. Les notes ont été séparées par un silence de 0.1s, que l'on aurait pu mettre dans la fonction note.
C'est ce que l'on va faire pour continuer et jouer une mélodie.
Il faut définir la table de la mélodie qui contient les numéros, donc les noms de notes puisqu'ils ont été énumérés. int taMelo [ ] = { DO, MI, DO, DO, RE, DO, MI} ;
Il n'est pas nécessaire de compter les notes pour mettre 7 dans le crochet: le compilateur sait faire.

```
// note4.ino
#define HP 13
#define HpToggle digitalWrite(Hp8,!digitalRead(Hp8));

enum { DO, RE, MI } notes ;
int taPer [3] = {1912, 1702, 1518};
int  maMelo [] = {DO,RE,MI,DO,DO,RE,MI};

void setup() {
  pinMode(HP, OUTPUT);
}
```

```
void note (int j) {
  for (int i=0 ; i<300 ; i++ ) {
  HpToggle;
  delayMicroseconds(taPer[j]/2);   }
  delay (100); //espace entre notes
}

void loop (){
  for (int k=0; k< sizeof(maMelo)/2 ; k++ ) {
      note (maMelo [k]);
  }
  delay (1000);
}
```

On voit que le programme tient compte de la longueur de la mélodie. On peut ajouter des notes sans rien changer ailleurs. L'instruction sizeof () ne tient pas compte du type de la variable. Elle compte des bytes, et on a vu qu'il y a deux bytes dans le type int de 16 bits. Si on ne divise pas par 2, la boucle for va chercher des contenus mémoire qui n'ont pas été initialisés! (A essayer)

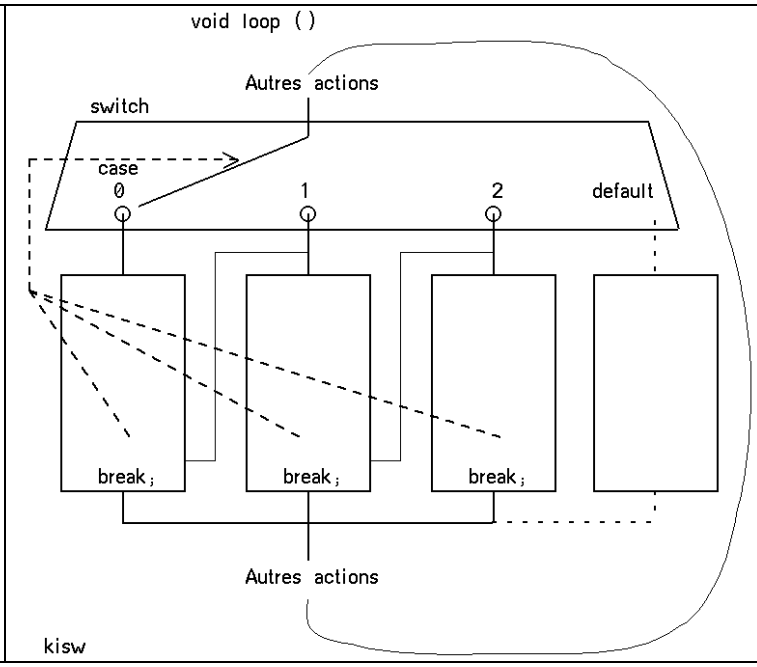
3.8 switch .. case

Il faut souvent exécuter diverses instructions selon la valeur d'une variable. Par exemple on presse sur des touches 1,2,3 pour commander un automate. Des `if ... else if ...` ne conviennent que pour des cas simples.
La commande `switch` observe une variable et examine différents cas selon sa valeur. Une nouvelle instruction, `break ;` permet de dire "j'ai trouvé, inutile de regarder plus loin". Si on n'a pas trouvé, on ajoute ce qu'il faut faire après l'instruction `default: .`

```
switch (variable) {
    case val1:
        instruction ;  break ;
    case val1:
        instruction ;  break ;
. . .
    default:
. . .
}
```

Il faut parenthéser le groupe `switch`, mais les `case` et `default` sont terminés par un `:` , sans accolade pour le groupe d'instructions.

`break ;` sort du groupe `switch`, pour ne pas exécuter les instructions des autres `case`.



L'exemple classique est de lire le clavier du PC en mode terminal et afficher une valeur, ou jouer une note puisqu'on vient d'apprendre à le faire. Voir note5.ino que nous ne commenterons pas.

L'application la plus fréquente en temps réel est de gérer un machine d'état, c'est-à-dire un programme qui a plusieurs états, et passe d'un état à l'autre selon des conditions extérieures.
Par exemple, une robot est dans un état "avance". La moustache gauche touche et on passe dans un état "tourne à droite", etc.
Un exemple bien détaillé est la commande de l'ascenseur kidule, voir www.didel.com/diduino/DikiAsc.pdf
Un autre exemple est l'encodeur présenté sous www.didel.com/diduino/Composants04.pdf

jdn 120304/131110/170211