06/05/2020

TP08 Inductif Arduino

- Découverte du langage Arduino depuis Flowcode

MLK - ECST

TP08 Inductif Arduino

- Découverte du langage Arduino depuis Flowcode

Compétences spécifiques :	**Structures algorithmiques fondamentales**
Pré-requis :	Programmation par ordinogramme
Type :	TP inductif

Objectif

L'objectif est d'apprendre les bases du langage de programmation Arduino lui-même basé sur le langage C ANSI (American National Standards Institute) en faisant le lien avec ce que vous connaissez déjà : la programmation Flowcode. Nous découvrirons également dans un prochain TP quelques subtilités du C++ avec le paradigme de la POO (Programmation Orientée Objet).

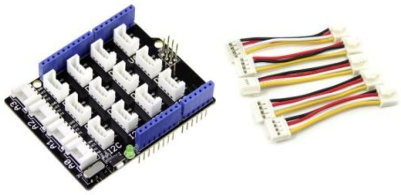
Présentation du matériel

Le succès des cartes Arduino vient du fait qu'elles sont très peu onéreuses et que l'IDE, ou EDI en français pour Espace de Développement Intégré, est open-source, donc gratuite.
Il existe plusieurs type de carte Arduino plus ou moins complexes dont le prix reste cependant très abordable : La plus simple et la plus distribuée est la carte Arduino Uno utilisée dans ce TP. Il existe également la Leonardo, plus puissante et avec un port USB pour interconnecter un clavier etc., la Méga qui à beaucoup plus de ports d'entrées/sorties (GPIO), la Yun qui permet de générer un réseau Wifi, etc.
La petite dernière Arduino Uno WiFi Rev2 (juin 2018), co-développée avec Microchip (microcontrôleurs PIC), intègre en plus du WiFi, une centrale inertielle… à suivre.

Bien qu'existant sur les microcontrôleurs des cartes Arduino, les ports de communication (PORTA, PORTB, etc.) sont rarement utilisés en tant que tels pour plusieurs raisons :

- La plupart des composants connectés utilisent peu de fils et très rarement le port complet de 8 bits,
- L'ensemble des bits des ports sont numérotés de 0 à 13 pour la carte Arduino UNO par exemple et jusqu'à 53 pour la Arduino Mega ce qui confère une plus grande portabilité des programmes pour passer d'un type de carte à un autre sans modification du code.

Matériel utilisé dans ce TP :

Arduino UNO	Interface grove[1]	LED, bouton et LCD RGB[2] I2C[3]
		

(1) La carte interface grove permet de connecter facilement sur la carte mère d'autres cartes filles : LED, boutons, etc.
(2) RGB : Pour le rétro éclairage le l'écran RVB (Rouge, Vert, Bleu)
(3) I2C (Inter-Integrated Circuit) : Interface de communication série qui permet de connecter une multitude de composants électroniques sur deux fils uniquement (bus informatique). Il est de ce fait également occasionnellement appelé TWI (Two Wire Interface) ou TWSI (Two Wire Serial Interface) chez certains constructeurs.

Les applications développées dans ce TP seront simulées en ligne sur Autodesk Tinkercad et/ou testé sur le matériel présenté ci-dessus avec l'IDE Arduino ou l'outils de développement en ligne.

Structure d'un programme Arduino - appelé croquis Arduino

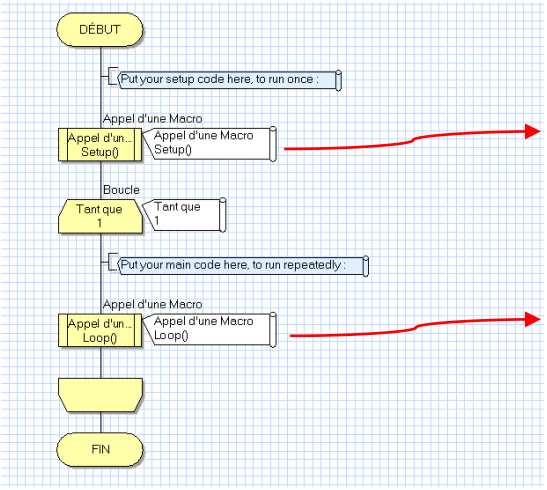
Dans les paragraphes suivants, nous allons présenter les structures algorithmiques de base. Pour chaque croquis Arduino et pour nous référer à quelque chose que nous connaissons déjà, nous présenterons l'équivalent en ordinogramme Flowcode.

Fonction *setup* et fonction *loop*

Un croquis Arduino est toujours composé d'au moins deux fonctions. Les fonctions sont équivalentes aux Macros dans Flowcode :

- La fonction *setup( )* qui s'exécute une seule fois au démarrage du programme et dans laquelle nous déclarons le sens de transfert des GPIO (entrée ou sortie), les vitesses de communication, l'initialisation des composants (LCD, etc.) et tout ce qui ne doit pas être exécuté dans la boucle de la fonction principale.
- La fonction *loop( )* qui représente le cœur du programme et qui, comme son nom l'indique, tourne en boucle.

Le programme ci-dessous ne fait rien. Il permet juste de présenter la structure du programme.




```
/* structure d'un programme */

void setup()
{
  // put your setup code here, to run once:

}

void loop()
{
  // put your main code here, to run repeatedly:

}
```

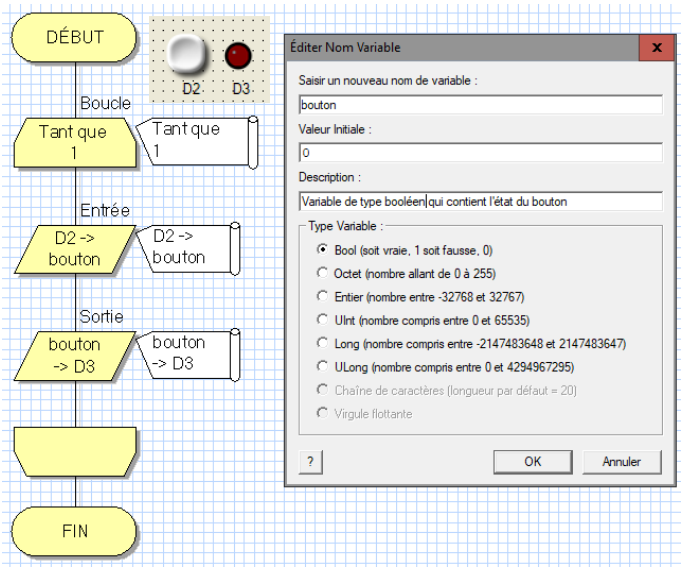
Remarque :

Le texte qui suit les caractères // est du commentaire. De même que celui qui se trouve entre /* et */

Les structures algorithmiques de base

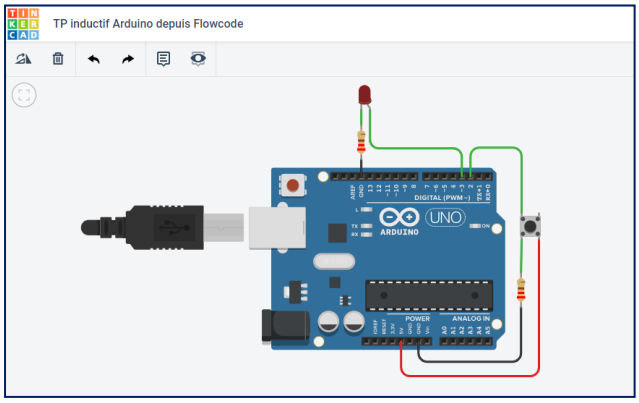
1. Structure Linéaire - FAIRE… *(inconditionnellement)*

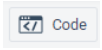
exemple : Lecture et écriture




```
// Création et initialisation à 0 d'une
// variable booléenne nommée bouton
bool bouton=0;

void setup()
{
  // Pin 2 en entrée
  pinMode(2, INPUT);
  // Pin 3 en sortie
  pinMode(3, OUTPUT);
}

void loop()
{
  // Lecture d'un bouton
  bouton=digitalRead(2);
  // Écriture de l'état du bouton sur la LED
  // connectée sur la sortie 3
  digitalWrite(3,bouton);
}
```

- Nous allons simuler le code ci-dessus. Connectez-vous au simulateur Autodesk Tinkercad avec votre identifiant Google Education (si vous n'avez pas encore créez de compte).
- Dans **Circuit** cliquez sur **Créer un circuit**
- Reproduisez fidèlement le circuit présenté ci-dessous en prenant soin de bien respecter le câblage, la polarité de la LED ainsi que la valeur des résistances (220 Ω) :

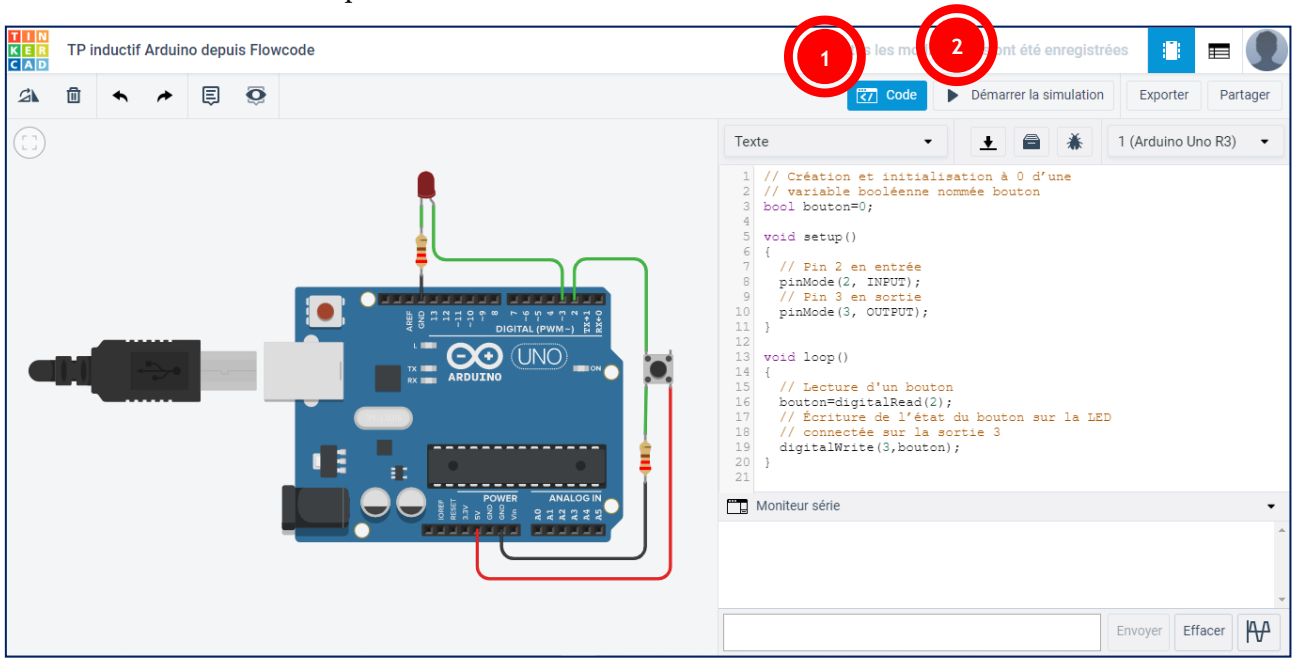


- Cliquez dans la partie code 

de Tinkercad (voir capture d'écran ci-dessous), sortez du mode *Blocs* pour passer le code en *mode texte*.

On est toujours tenté de copier-coller les codes présentés. Je vous conseille fortement de les retaper pour faire fonctionner votre mémoire visuelle et aussi pour faire des erreurs. C'est en corrigeant vos erreurs que vous progresserez.

- Recopiez le code de la page précédente comme indiqué dans la capture d'écran ci-dessous.
- Lancez la simulation en cliquant sur 



- Dans le code, quelle est la syntaxe pour lire une entrée ?

_____

- Quelle est la syntaxe pour écrire sur une sortie ?

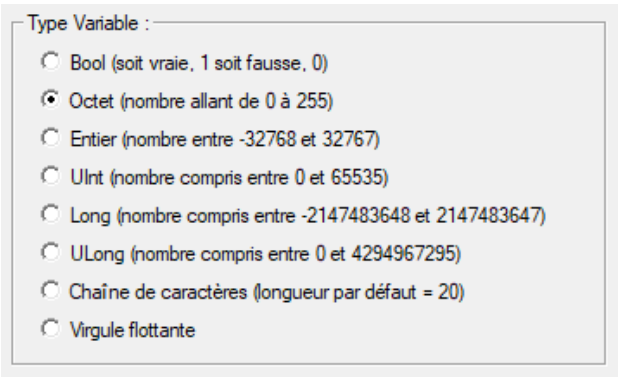
_____

- Si vous avez le matériel à disposition, créez le croquis *P:\Devoirs\sin\lectureEcriture\lectureEcriture.ino* (le sous-dossier *lectureEcriture* se crée automatiquement) avec l'IDE Arduino. Connectez une carte Arduino avec le bouton et la LED sur le bon GPIO. Vérifiez le port de communication USB dans le menu Outil. Testez votre programme.

Faites vérifier le fonctionnement par le professeur

Remarques :

- Le langage Arduino privilégie le camel case. Pratique qui consiste à écrire sans espace ni ponctuation et en mettant en capitale la première lettre de chaque nouveau mot. Ex : digitalWrite, digitalRead, pinMode, etc.
- Tout comme dans Flowcode, il existe un grand nombre de types de variables. Il faut toujours prendre le plus adapté : *bool* pour l'état d'un bit vrai ou faux, *char* pour contenir le code ASCII d'un caractère, *unsigned char* (ou *byte* qui revient au même) pour contenir une variable de type octet de 0 à 255, etc.




```
// création et initialisation de variables
// Type booléen Bool 0 ou 1
bool bouton=0;

// Type octet unsigned char ou bien byte de 0 à 255
unsigned char valeur = 255 ;

// Type entier signé int de -32768 à 32767
int variable = -100 ;

// Type entier non signé unsigned int de 0 à 65535
unsigned int variable2 = 0 ;

// Type nombre long signé de -2147383648 à 2147483647
long resultat = 0 ;

// Type nombre long non signé unsigned long de 0 à 4294967295
unsigned long total = 0 ;

// char chaine[20] pour contenir une chaîne de 20 caractères
char chaine[8] = "Arduino";
char texte[8] = {'A','r','d','u','i','n','o','\0'};

// Type nombre à virgule flottante ex : 2,3467 ou 2346,7
// de 3.4028235E+38 à - 3.4028235E+38
float sensorCalibrate = 1.117;
```

Comme dans tous les langages de programmation, les variables sont stockées en RAM (Random Acces Memory). Le contenu de la RAM à l'avantage d'être modifiable à volonté mais sa capacité est limité par rapport à la mémoire programme ROM (Read Only Memory).

Pour stocker en ROM une valeur qui ne change pas durant l'exécution du programme on ajoute à la syntaxe précédente le préfixe *const*.

```
// Exemples de création et initialisation de constantes

const float pi = 3.14;
const float R = 10.0;
const float D = 2*pi*R ; // Peut être tirée d'un calculs
```

Pour n'occuper ni la RAM ni la ROM on utilise la directive d'assemblage *#define*.
Ceci est particulièrement intéressant pour modifier le câblage d'une carte et faire correspondre très rapidement le programme correspondant.

```
// Câblage

#define pinBouton 2
#define pinDEL 3
```

Ainsi, le code vu précédemment : ---> *Devient celui-ci :*

```
// Création et initialisation à 0 d'une variable
// booléenne nommée bouton
bool bouton=0;

void setup()
{
  // Pin 2 en entrée
  pinMode(2, INPUT);
  // Pin 3 en sortie
  pinMode(3, OUTPUT);
}

void loop()
{
  // Lecture d'un bouton
  bouton=digitalRead(2);
  // Écriture de l'état du bouton sur la LED connectée
  // sur la sortie 3
  digitalWrite(3,bouton);
}
```

```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
bool bouton=0;

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
}

void loop()
{
  // Lecture d'un bouton
  bouton=digitalRead(pinBouton);
  // Écriture de l'état du bouton sur la LED
  digitalWrite(pinDEL,bouton);
}
```

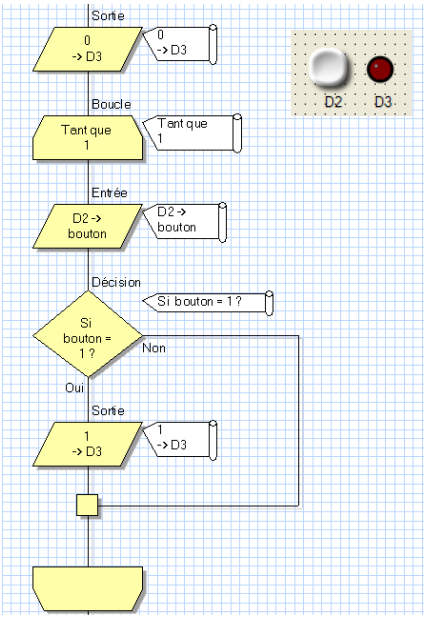
- Comptez les occurrences des chiffres 2 et 3 dans le code de gauche ? _____
- Comptez les occurrences des chiffres 2 et 3 dans le code de droite ? _____
- Quel code (de gauche ou de droite) demandera le moins de modifications si on doit modifier par exemple le câblage du bouton sur la broche 4 et de la DEL sur la broche 5 de l'Arduino ? _____
- Simulez et/ou tester la solution de droite, modifiez le câblage et faire correspondre le code. Testez à nouveau.

Faites vérifier le fonctionnement par le professeur

2. Structure Alternative

Structure Alternative Réduite - SI (Condition)… ALORS FAIRE…

- Simulez et/ou testez le croquis *P:\Devoirs\sin\alternativeReduite\alternativeReduite.ino*

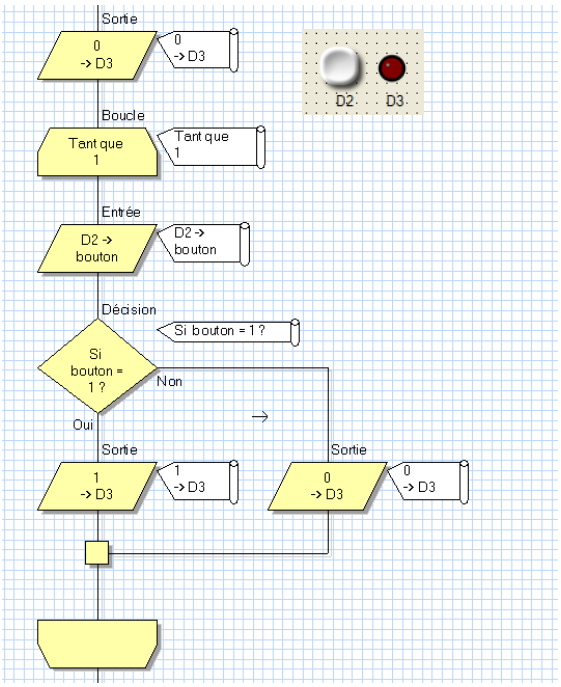



```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
bool bouton=0;

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
  // On éteint la LED à la mise sous tension :
  digitalWrite(pinDEL,0);
}

void loop()
{
  bouton=digitalRead(pinBouton);
  if(bouton)    // équivalent à if(bouton==1)
  {
     digitalWrite(pinDEL,1);
  }
}
```

Faites vérifier le fonctionnement par le professeur

Structure Alternative Complète - SI (Condition)… ALORS FAIRE… SINON FAIRE…

- Complétez, simulez et/ou testez le croquis *P:\Devoirs\sin\alternativeComplete\alternativeComplete.ino*




```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
bool bouton=0;

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
  // On éteint la LED à la mise sous tension :
  digitalWrite(pinDEL,0);
}
void loop()
{
  bouton=digitalRead(pinBouton);
  if(bouton)
  {
```

_____

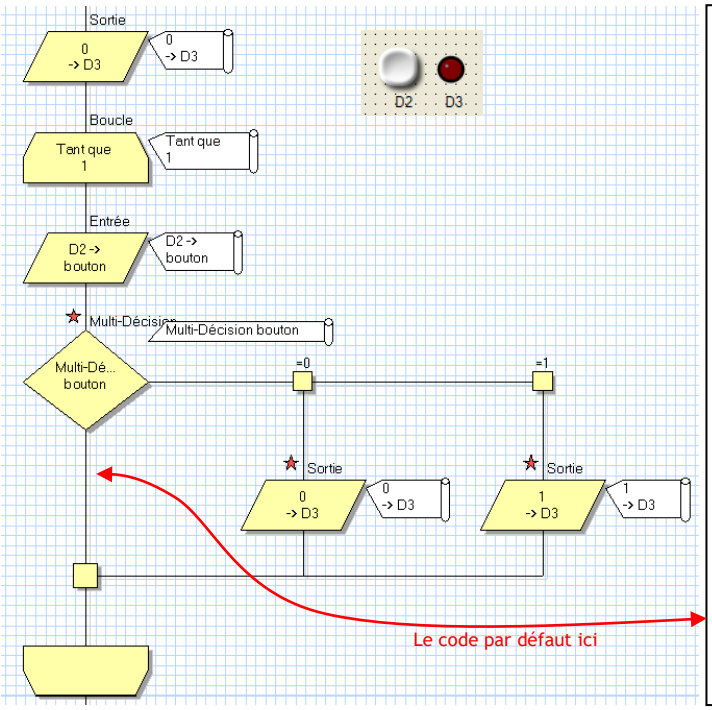
```
  }
  else
  {
```

_____

```
  }
}
```

Faites vérifier le fonctionnement par le professeur

Structure Alternative à Choix Multiple - SI (Condition) ALORS CAS 1 FAIRE… OU CAS N FAIRE…




```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
char bouton=0;    // attention type char ou int

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
  // On éteint la LED à la mise sous tension :
  digitalWrite(pinDEL,0);
}

void loop()
{
  bouton=digitalRead(pinBouton);
  switch(bouton)    // attention type char ou int
  {
     case 0:
       digitalWrite(pinDEL,0);
       break;
     case 1:
       digitalWrite(pinDEL,1);
       break;
     default:
       // on peut mettre ici un code par défaut
       break;
  }
}
```

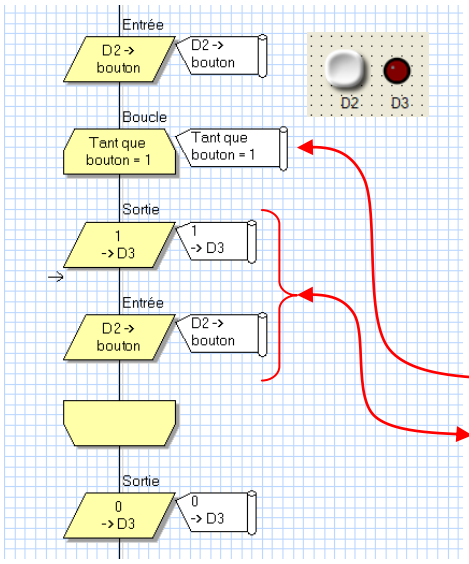
Remarquez que le nombre de choix ne se limite pas à deux. Dans notre exemple, la variable *bouton* est de type *char* donc il peut y avoir **256** cas en plus du cas par défaut.

- Simulez et/ou testez le croquis *P:\Devoirs\sin\choixMultiple\choixMultiple.ino*

Faites vérifier le fonctionnement par le professeur

3. Structure Répétitive

Boucle TANT QUE (Condition)… FAIRE…




```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
bool bouton=0;

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
}

void loop()
{
  bouton=digitalRead(pinBouton);
  while(bouton)    // équivalent à while(bouton==1)
  {
     digitalWrite(pinDEL,1);
     bouton=digitalRead(pinBouton);
  }
  digitalWrite(pinDEL,0);

}
```

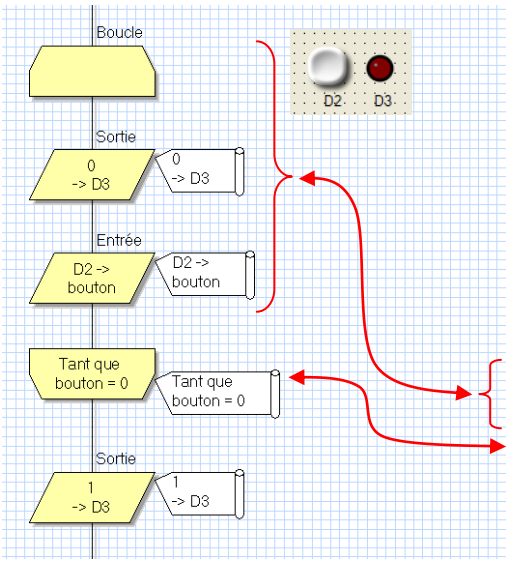
Remarquez la variable (*bouton)* dans le *Flowcode* et dans le *croquis Arduino* :

- Elle doit évoluer avant le test pour pouvoir rentrer dans la boucle. Pour ce type de boucle (*while*) Si la condition est fausse, la boucle n'est jamais exécutée.
- Elle doit surtout évoluer (et être rafraichie) dans la boucle pour ne pas que la boucle soit infinie.

- Simulez et/ou testez le croquis *P:\Devoirs\sin\tantQueFaire\tantQueFaire.ino*

Faites vérifier le fonctionnement par le professeur

Boucle FAIRE… TANT QUE (Condition)




```
// Câblage à adapter selon les connexions de la carte
#define pinBouton 2
#define pinDEL 3

// Création et initialisation à 0 d'une variable
bool bouton=0;

void setup()
{
  pinMode(pinBouton, INPUT);
  pinMode(pinDEL, OUTPUT);
}

void loop()
{
  do
  {
     digitalWrite(pinDEL,0);
     bouton=digitalRead(pinBouton);
  } while(bouton==0);
  digitalWrite(pinDEL,1);

}
```

Remarquez la variable (*bouton)* dans le *Flowcode* et dans le *croquis Arduino* :

- Même si elle est fausse départ, le corps de la boucle est exécutée au moins une fois.
- Elle doit évoluer à l'intérieur la boucle pour ne pas que la boucle soit infinie.

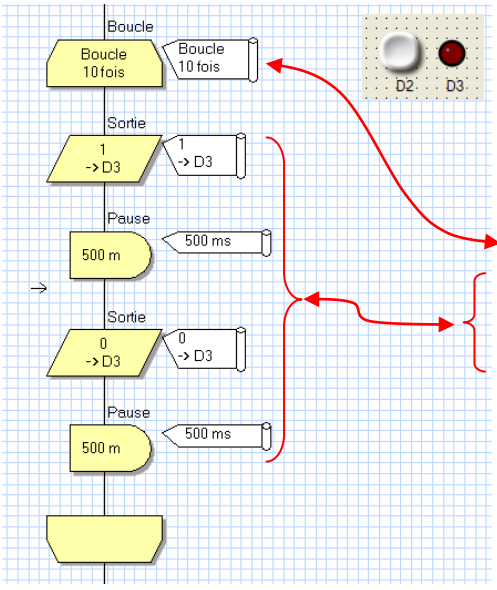
- Simulez et/ou testez le croquis *P:\Devoirs\sin\faireTantQue\faireTantQue.ino*
- La DEL est-elle réellement toujours allumée lorsqu'on appuie sur le bouton ? Justifiez votre réponse :

_____

Faites vérifier le fonctionnement par le professeur

4. Structure Répétition Contrôlée

Boucle POUR (Condition Initiale) TANT QUE (Condition Finale) FAIRE…




```
// Câblage à adapter selon les connexions de la carte

#define pinDEL 3

void setup()
{
  pinMode(pinDEL, OUTPUT);
}
void loop()  // Attention la boucle tourne en boucle !
{
  for(int n=0 ; n<10 ; n++)
  {
     digitalWrite(pinDEL,1);
     delay(500);
     digitalWrite(pinDEL,0);
     delay(500);
  }
}
```

Dans la boucle *pour*, remarquez que la variable *n* est initialisée à 0 une seule fois au début. Le test *n<10* est effectué : Si le résultat est *vrai* alors la boucle est exécutée puis la variable *n* est incrémentée (+1), sinon on sort de la boucle.
Cet ordre peut changer dans d'autres langages informatiques.

- Trouvez une façon de stopper l'effet de la boucle loop pour que la LED ne clignote réellement que 10 fois.

  Code ajouté : _____

- Simulez et/ou testez le croquis *P:\Devoirs\sin\bouclePour\bouclePour.ino*

Faites vérifier le fonctionnement par le professeur

Débogage des programmes

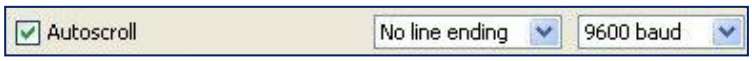
1. Moniteur série

L'interface de développement Arduino ne permet pas de faire du débogage pas à pas. Il faut donc utiliser le moniteur série qui permet la communication entre le PC et la carte Arduino. En plaçant judicieusement du texte dans le programme, on peut savoir s'il s'exécute correctement. Le moniteur série permet d'afficher du texte ou des variables sur le PC. Pour fonctionner correctement, la vitesse du moniteur série du PC doit être la même que celle paramétrée dans l'Arduino. Par exemple 9600 baud :

Code à ajouter dans setup( ) :

```
Serial.begin(9600);
```

Configuration de l'IDE : (inutile dans le simulateur)



Communication série dans un sens : Arduino -> PC

Le croquis suivant permet d'envoyer des textes sous forme de *"chaînes de caractères"*. Ces textes sont transmis de la carte Arduino vers le PC. Ils doivent s'afficher sur le moniteur série de l'IDE Arduino ou du simulateur en ligne Tinkercad. Prenez soin de bien lire et comprendre le code et ainsi que ses commentaires.

- Simulez et/ou testez le croquis *P:\Devoirs\sin\ArduinoVersPC\ArduinoVersPC.ino*

```
// // Communication Arduino -> PC sur la liaison série


void setup()
{
  Serial.begin(9600);               // Paramétrage de la liaison série à 9600 bps
}

void loop()
{
  Serial.print("Bonjour");          // Serial.print() écrit sans saut de ligne
  Serial.println("Bonne journée");  // Serial.println() écrit puis saute la ligne (retour chariot)
}
```

- Appropriez vous la syntaxe en modifiant le texte, ajoutant des lignes avec et sans « retour chariot ».

Faites vérifier le fonctionnement par le professeur

Communication série dans les deux sens : PC -> Arduino -> PC

Il est possible d'envoyer des nombres du PC vers l'Arduino avec le champ suivant (dans le moniteur série) :

Le croquis suivant permet d'envoyer des nombres du simulateur ou du *PC* vers la carte *Arduino*. Une fois reçus par la carte Arduino, cette dernière les renvoie au PC comme dans l'exemple précédent. Une fois encore, prenez soin de bien lire et comprendre le code et sa syntaxe ainsi que les commentaires.

```
// Communication PC -> Arduino -> PC sur la liaison série
int nombre = 0;   // création et initialisation à 0 d'une variable pour la communication série

void setup()
{
  Serial.begin(9600);     // Paramétrage de la liaison série à 9600 bps
}

void loop()
{
    // Attente de réception de données sur la liaison série
    if (Serial.available()>0)  // la fonction Serial.available() retourne le nombre de caractères (ou d'octet)
    {                          // présents dans le buffer de la liaison série
       nombre = Serial.read();          // Lecture du nombre entrant par la liaison série de l'Arduino
       Serial.print("Nombre recu : ");  // Écriture d'un texte sans passage à la ligne
       Serial.println(nombre);          // Écriture avec passage à la ligne. La base par défaut est décimale
       Serial.print("Soit en base decimale : ");
       Serial.println(nombre, DEC);     // Autre façon d'avoir la base décimale
       Serial.print("Soit en base binaire : ");
       Serial.println(nombre, BIN);     // Pour avoir une écriture dans la base binaire
       Serial.print("Soit en base hexadecimale: ");
       Serial.println(nombre, HEX);     // Pour avoir une écriture dans la base hexadécimale
       Serial.print('\n');              // Autre façon passer à la ligne
    }
}
```

- Simulez et/ou testez le croquis suivant *P:\Devoirs\sin\pcVersArduino\pcVersArduino.ino*

- Le résultat n'est pas celui que l'on pouvait espérer ; commentez en tenant compte du codage ASCII des caractères de la table ci-contre :

_____

_____

_____

_____

Caractère	Décimale	Binaire	Hexadécimale
'0'	48	0011 0000	30
'1'	49	0011 0001	31
'2'	50	0011 0010	32
'3'	51	0011 0011	33
'4'	52	0011 0100	34
'5'	53	0011 0101	35
'6'	54	0011 0110	36
'7'	55	0011 0111	37
'8'	56	0011 1000	38
'9'	57	0011 1001	39

- Proposez une opération mathématique à effectuer sur la variable nombre (code ASCII) pour que les valeurs s'affichent dans les bonnes bases (décimale, binaire ou hexadécimale) :

_____

Faites vérifier le fonctionnement par le professeur

- Que se passe t-il lorsqu'on envoie un nombre supérieur à 9 ?

_____

Pour remédier à ces problème, il suffit de remplacer la commande `nombre = Serial.read()` ; par celle-ci :

`nombre = Serial.parseInt()` ;

- Modifier le programme, envoyer les données avec le moniteur série et complétez le tableau suivant :

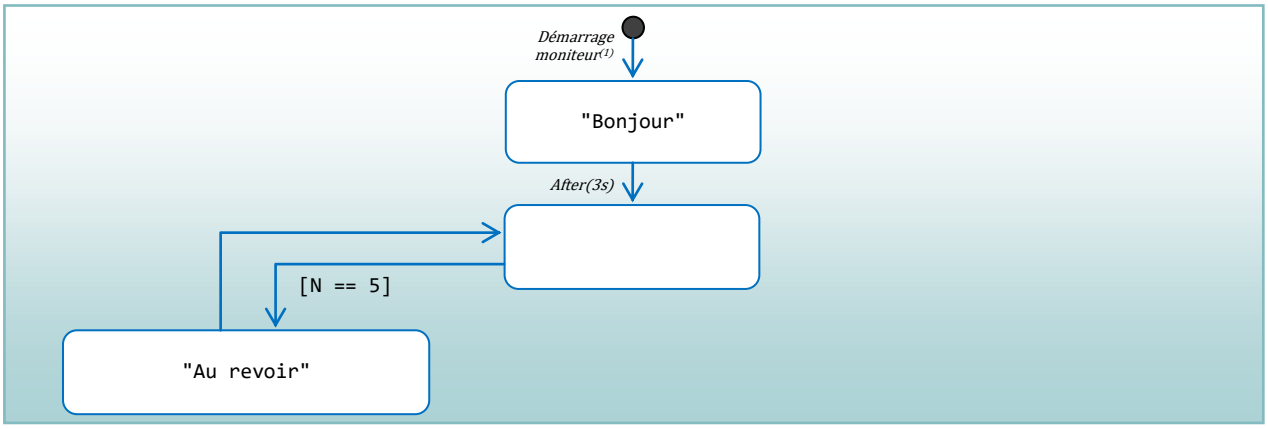
	Données reçues		
Données envoyées	DEC	BIN	HEX
-32768			
-1			
0			
32767			

Remarquez que les *int* sont codés sur 16 bits dans les Arduino Uno. Les 1 supplémentaires pour le signe négatif sont non significatifs car les *int* sont usuellement codés sur 32 bits.

2. Applications

Dans les applications suivantes, les états représentent les textes affichés sur le moniteur série et les transitions représentent les nombres N envoyés par le moniteur série.

If… Then…



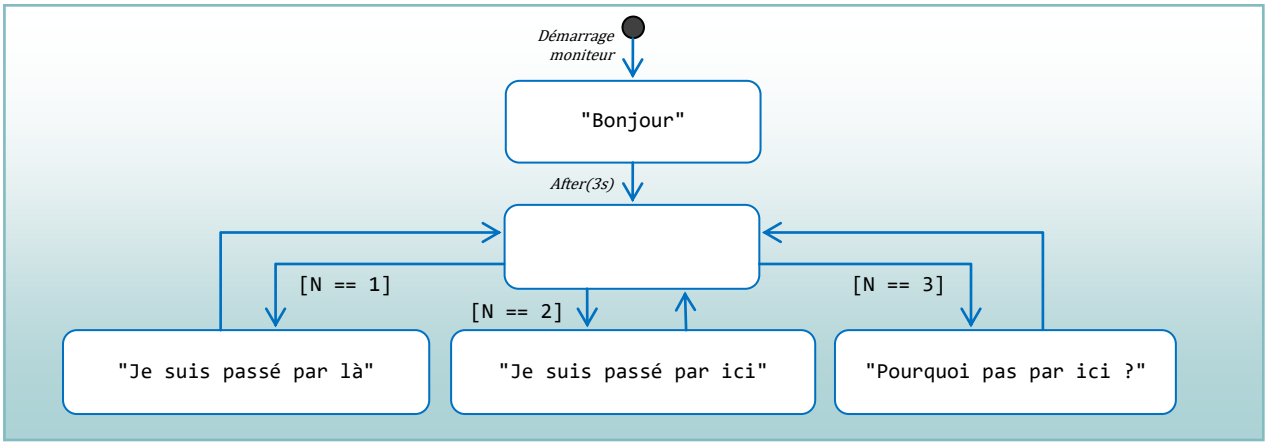
(1) Dans l'IDE Arduino, le démarrage du moniteur série effectue un *Reset* du programme.

Remarquez que du fait de la fonction *loop*, la structure if réalise en fait une boucle *tant que*.

- Simulez et/ou testez le croquis *P:\Devoirs\sin\appliIfThen\appliIfThen.ino*

Faites vérifier le fonctionnement par le professeur

Switch... Case...



- Simulez et/ou testez le croquis *P:\Devoirs\sin\appliSwitchCase\appliSwitchCase.ino*

Faites vérifier le fonctionnement par le professeur

Boucle For

- Simuler et ou testez la structure suivante dans un nouveau croquis *P:\Devoirs\sin\appliFor\appliFor.ino* :
- Recopiez la suite de nombres générés. Justifiez le résultat.

_____

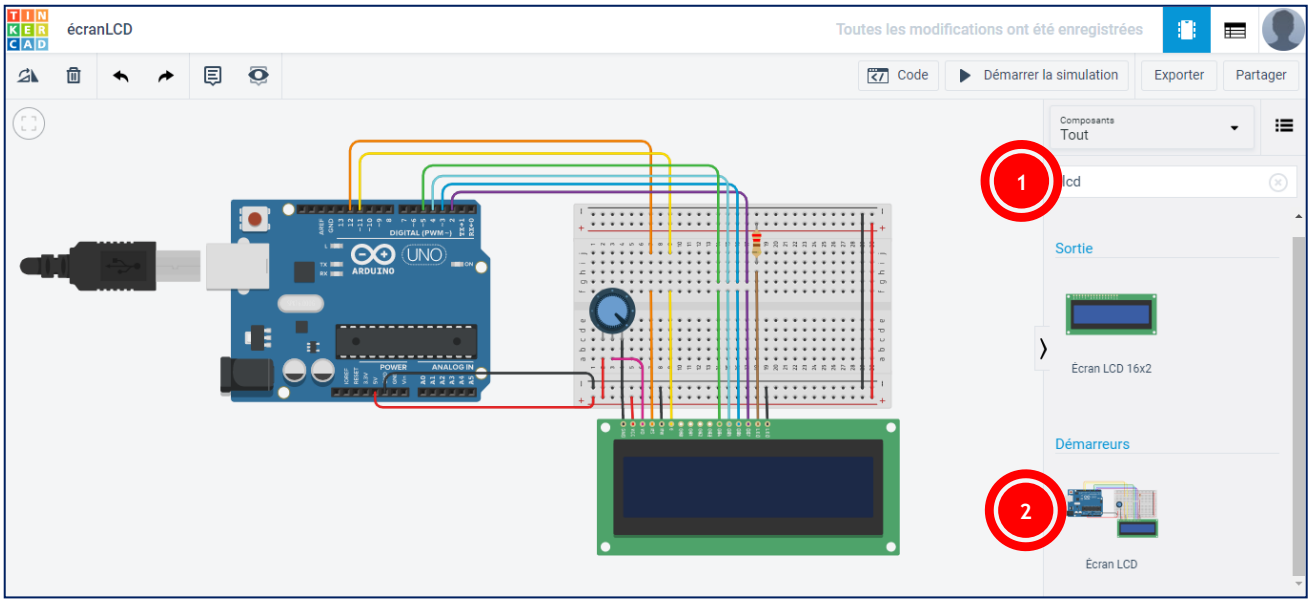
_____

_____

```
for(int x = 2; x < 100; x = x * 1.5)
{
  Serial.println(x);
}
```

Écran LCD avec le simulateur Tinkercad en ligne

- Réalisez le montage suivant. Pour cela vous pouvez utiliser l'assemblage (démarreurs) dans les composants en faisant une recherche avec le mot LCD comme indiqué ci-dessous :



Le potentiomètre permet d'ajuster le réglage du contraste de l'écran LCD. Pour voir le texte correctement, il doit être placé en butée à droite après avoir lancé la simulation.

- Rentrez le code suivant et lancer la simulation :

```
#include "LiquidCrystal.h"
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);;  //Instancie un objet lcd de la classe LiquidCrystal (C++)

void setup(){
    lcd.begin(16, 2);      // Configure le nombre de colonnes et de lignes du LCD
    Serial.begin(9600);    // Initialisation de la liaison série
    lcd.setCursor(0,0);
    lcd.print("Taper texte dans");
    lcd.setCursor(0,1);
    lcd.print(" moniteur serie");
}
void loop()
{
    if (Serial.available())             // Quand un caractère arrive dans la liaison série
    {
        delay(100);                     // délai pour que le message complet arrive
        lcd.clear();                    // effacement de l'écran LCD
        lcd.setCursor(0,0);          // début de première ligne
        lcd.print("Votre texte :");
        lcd.setCursor(0,1);          // début de deuxième ligne
        while (Serial.available() > 0)  // Tant qu'il y a des caractère dans le buffer série
        {
            lcd.write(Serial.read());   // Affiche chaque caractère du buffer sur le LCD
        }
    }
}
```

- Justifiez le résultat :

_____

_____

Faites vérifier le fonctionnement par le professeur

Écran LCD rétro éclairé avec l'IDE et une carte Arduino UNO (ne fonctionne pas en simulation)

- Connectez un afficheur LCD sur le bus I2C puis créez et testez le croquis suivant :

P:\Devoirs\sin\appliLCD\appliLCD.ino :

```
#include <Wire.h>
#include "rgb_lcd.h"
rgb_lcd lcd;  //zoomer pour bien voir la syntaxe
unsigned char rouge = 0 ;               // Variable composante rouge de 0 à 255
unsigned char vert  = 255               // Variable composante verte de 0 à 255
unsigned char bleu  = 0 ;               // Variable composante bleue de 0 à 255

void setup(){
    Serial.begin(9600);                 // Initialisation de la liaison série
    lcd.begin(16, 2);                   // Configure le nombre de colonnes et de lignes du LCD
    lcd.setRGB(rouge, vert, bleu) ;
    lcd.setCursor(0,0);
    lcd.print("Taper texte dans");
    lcd.setCursor(0,1);
    lcd.print(" moniteur serie");
}
void loop()
{
    if (Serial.available())             // Quand un caractère arrive dans la liaison série
    {
        delay(100);                     // délai pour que le message complet arrive
        lcd.clear();                    // effacement de l'écran LCD
        lcd.setCursor(0,0);             // début de première ligne
        lcd.print("Votre texte :");
        lcd.setCursor(0,1);             // début de deuxième ligne
        while (Serial.available() > 0)  // Tant qu'il y a des caractère dans le buffer série
        {
            lcd.write(Serial.read());   // Affiche chaque caractère du buffer sur le LCD
        }
    }
}
```

Faites vérifier le fonctionnement par le professeur