

Mettre en pause les interruptions pour les codes critiques :

```
noInterrupts();  
    //Code that MUST be not interrupt  
interrupts();
```

Lire dans l'EEPROM, le *get()* ne nécessite pas de cast :

```
byte value1;  MyObject value2;  
EEPROM.read(address, value1); // 0-255  
EEPROM.get(address, value2); // all
```

Mettre à jour une valeur que si besoin :

```
EEPROM.update(address, value); // 0-255
```

Vider l'EEPROM :

```
for (int i = 0 ; i < EEPROM.length() ; i++) {  
    EEPROM.write(i, 0);  
}
```

## Autres

S'il manque de la mémoire pour une chaîne de caractères, la macro *F(string)* permet de la stocker dans la flash :

```
Serial.print(F("Very long string"));
```

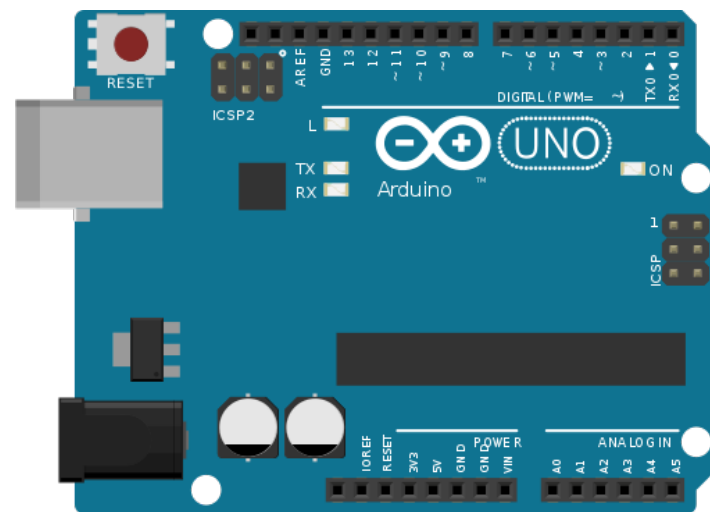
Taille d'une variable, un type, un tableau en bytes :


```
sizeof(int); int a = 1; sizeof(a);
```

Stocker dans la flash, à la place de la SRAM :

Suivant le compilateur, une des deux versions fonctionnent !

```
#include <avr/pgmspace.h>  
const type varName[] PROGMEM = {d0, d1, ...};  
const PROGMEM type varName[] = {d0, d1, ...};
```



<http://github.com/Servuc/jaizappe>        Licence GPLv3

# #Jaizappé ... ... le C++ Arduino

## Histoire

L'Arduino, ou Genuino est une plateforme de prototypage basée sur un micro-contrôleur *Atmel* en général.

Il existe plusieurs boards avec chacune des spécificités.

*Ce mémo est inspiré de la documentation Arduino.*

## Base

### Code de base

Le code d'un projet Arduino de base :

```
void setup() {  
    //Exécuté en premier  
}  
void loop() {  
    //Exécuté après setup() en boucle  
}
```

### Calcul binaire

Récupérer les parties *big* et *low* d'une valeur :

```
highByte(value);  
lowByte(value);
```

Lire, écrire les bits à partir de la droite, *value* : valeur numérique

```
bitRead(value, position);  
bitWrite(value, position, bit); // bit = 0 ou 1
```

Quelques raccourcis (*value* et *position*) :

```
bitSet(val, pos); // bitWrite(val, pos, 1);  
bitClear(val, pos); // bitWrite(val, pos, 0);
```

### Analyse des caractères

```
isAlphaNumeric(thisChar); // [a-zA-Z0-9]  
isAlpha(thisChar); // [a-zA-Z]  
isAscii(thisChar);  
isWhitespace(thisChar);  
isControl(thisChar); // \n \r ...  
isDigit(thisChar); // [0-9]  
isLowerCase(thisChar); // [a-z]  
isUpperCase(thisChar); // [A-Z]  
isPrintable(thisChar); // Affichable  
isPunct(thisChar);  
isHexadecimalDigit(thisChar);
```

## Mathématiques et temps

### Général

Sinus	$\sin(x, y)$	Cosinus	$\cos(x)$	Tang.	$\tan(x, y)$
Mini.	$\min(x, y)$	Racine	$\sqrt{x}$	Absolu	$\text{abs}(x, y)$
Maxi.	$\max(x, y)$	Puiss.	$\text{pow}(x, y)$		

*constrain(x, min, max)* : Retourne *x* si  $\min < x < \max$  sinon une valeur extrême.

*map(x, minSource, maxSource, minDest, maxDest)* :

Retourne *x* adapté aux extrêmes de destination en fonction des extrêmes sources.

### Aléatoire

Changer la *seed* de l'aléatoire :

```
randomSeed(analogRead(0));  
randomSeed(numberValue);
```

Avoir un nombre aléatoire, *min* inclusif, *max* exclusif

```
long random(max);  
long random(min, max);
```

### Temps

Temps écoulé depuis le démarrage :

```
unsigned long millis(); unsigned long micros();
```

*Overflow* de *millis()* et *micros()* à 50 jours et 70 minutes.

Faire patienter quelques instants :

```
delay(milliSeconds);  
delayMicroseconds(microSeconds);
```

## Mémoire

### EEPROM

Un Arduino a une mémoire flash pour les variables, l'EEPROM.  
Attention, le nombre d'écriture est limitée ! (~ 100 000)

```
#include <EEPROM.h>
```

Écrire dans l'EEPROM :

```
EEPROM.write(address, value); // 0-255  
EEPROM.put(address, value); // all
```

# Débugage et série

## Serial

Initialisation (une seule fois!) :

```
Serial.begin( SPEED );
```

**SPEED** vaut **9600**, **57600** ou **115200** en général. Vitesse en baud.  
Écrire sur le moniteur de debug :

```
Serial.print( VARIABLE ); //Pas de \n  
Serial.println( VARIABLE ); //Avec \n
```

Dans le cas d'un *float* :

```
Serial.print( VARIABLE, precision );
```

Lire le port série (byte par byte), utiliser dans *loop()* :

```
if ( Serial.available() > 0 ) {  
    byte incomingByte = Serial.read();  
}
```

## SoftwareSerial

Fonctionne comme *Serial*. Nécessite :

```
#include <SoftwareSerial.h>
```

Initialisation (inutile pour *Serial*) :

```
SoftwareSerial mSerial(PIN_RX, PIN_TX);
```

**IMPORTANT!** Pour communiquer entre 2 supports :

- Support 1 : *RX* ↔ *TX* : Support 2
- Support 1 : *TX* ↔ *RX* : Support 2

Si plus d'un *SoftwareSerial* est initialisé, un seul peut être en écoute grâce à la première fonction :

```
mSerial.listen();  
mSerial.isListening();
```

## Les broches

Il est conseillé de définir les broches :

```
#define PIN_FCT_ABCD 5
```

Indiquer le mode de la broche (dans *setup()*) :

```
pinMode( PIN_ID, INPUT ); //Lecture  
pinMode( PIN_ID, OUTPUT ); //Ecriture
```

## Analogiques

Notées **AX** (*X* un nombre). **PIN\_ID** notée **AX** aussi.

Mettre la valeur d'une broche (mode **OUTPUT**) :

```
analogWrite( PIN_ID, value ); // 0 - 255
```

Récupérer la valeur d'une broche (pas de mode) :

```
int myVal = analogRead( PIN_ID );  
float myVoltage = myVal * (VOLT_MAX / 1023.0);
```

Fixer la valeur de lecture sur les broches analogiques :

```
analogReference( value );
```

Arduino	value	Description
Tous	<b>DEFAULT</b>	<b>5V</b> ou <b>3.3V</b>
ATMega 168/328	<b>INTERNAL</b>	<b>1.1V</b>
ATMega 8	<b>INTERNAL</b>	<b>2.56V</b>
Ard. Mega	<b>INTERNAL1V1</b>	<b>1.1V</b>
Ard. Mega	<b>INTERNAL2V56</b>	<b>2.56V</b>
Tous	<b>EXTERNAL</b>	Sur <i>AREF</i> ( <b>0V</b> - <b>5V</b> )

## Digitales

Notées **dX** ou **X** (*X* un nombre). Égale à **0** ou **1**.  
Correspondant respectivement à **LOW** et **HIGH**.

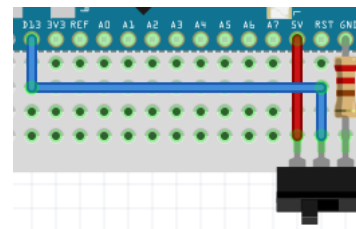
Mettre la valeur d'une broche (mode **OUTPUT**) :

```
digitalWrite( PIN_ID, HIGH ); //1 = 5V ou 3.3V  
digitalWrite( PIN_ID, LOW ); //0 = 0V
```

Récupérer la valeur d'une broche (mode **INPUT**) :

```
int myVal = digitalRead( PIN_ID );
```

Cependant, il faut stabiliser la broche à 0 ou 5/3.3V. Exemple :



## Usage avancé

Inutilisable avec les Arduino *Gemma* et *Due*.

Émettre une tonalité avec un Arduino avec un piezo, un haut parleur, **pin** doit être notée **PWM**. **time** en millisecondes.

Arduino	Fréq. min	Fréq. max
Tous	<b>31</b>	<b>65535</b>
Zero	<b>41</b>	<b>275000</b>

```
tone( pin, frequency ); //Ou  
tone( pin, frequency, time );
```

Pour arrêter la première instruction :

```
noTone();
```

Lire une impulsion (signal continu) sur une broche digitale de 10 microsecondes à 3 minutes,  
*value* : **HIGH** et **LOW**, *timeout* : Attente max. en microsecondes.

```
pulseIn( pin, value );  
pulseIn( pin, value, timeout );
```

## Veille et interruptions

```
delay( MILLI_SECONDS );  
delayMicroseconds( MICRO_SECONDS );
```

Un *delay()* peut être arrêté par une action sur une broche digit. :

Uno, Nano, base 328	<b>2,3</b>
Mega, Mega2560, MegaADK	<b>2,3,18,19,20,21</b>
Micro, Leonardo, base 32u4	<b>0,1,2,3,7</b>
Zero	Toutes sauf la <b>4</b>
MKR1000 Rev.1	<b>0,1,4,5,6,7,8,9,A1,A2</b>
Due, 101	Toutes

Attacher une fonction à l'interruption :

```
attachInterrupt(  
    digitalPinToInterrupt( PIN_ID ),  
    FUNCTION, MODE);
```

1. **FUNCTION** : Une fonction de type *void* sans paramètre;
2. **MODE** :
  - (a) **LOW** : Si le pin est à 0 (**LOW**);
  - (b) **CHANGE** : Tension qui change;
  - (c) **RISING** : **LOW** à **HIGH**;
  - (d) **FALLING** : **HIGH** à **LOW**;
  - (e) **HIGH** : Pin à 1 (Sur *Due*, *Zero* et *MKR1000*).

Enlever une interruption sur une broche :

```
detachInterrupt(  
    digitalPinToInterrupt( PIN_ID ));
```

**IMPORTANT** : Les variables partagées doivent être *volatile* :

```
volatile int mCount = 0;  
void buttonPressed () { //Fct interrup.  
    mCount++;  
}  
  
void loop() {  
    Serial.println(mCount); delay(7331);  
}
```