Aide-mémoire Arduino

Auteur : Frédéric Perrenoud
<frederic@perrenoud.com>
D'après la « Arduino cheat sheet » de Gavin Smith et la référence du langage à :
http://arduino.cc/en/Reference/Extended

Structure :

```
void setup(){ }   void loop(){ }
type fonction(type parametre1, type parametre2...)
{ }
return x; // Ou "return;" pour les voids
struct Structure { }; // majuscule!
enum chiffres {un=1, deux, trois}; //comme
#define un 1
class Classe { }; // majuscule!
class Enfant : public Parent {
   // constructeur
   Enfant() : Parent(){ };
};
```

Précompilateur :

```
// (commentaire)
/*  (commentaire multi ligne) */
#define DOUZAINE 12 // pas de point virgule!
#define SOMME(x,y) ((x) + (y))
#include <avr/pgmspace.h>
#define _DEBUG(x) Serial.println("DEBUG :
"+String((x)));
#ifdef _DEBUG
_DEBUG(message);
#endif
```

Opérateurs :

```
= (assignement)
+ (addition)
- (soustraction)
* (multiplication)
/ (division)
% (modulo, reste de la division)
== (égal à)
!= (non égal à, différent de)
< (plus petit que)
> (plus grand que)
<= (plus petit que ou égal à)
>= (plus grand que ou égal à)
&& (et)
|| (ou)
! (non)
```

Pointeurs :

```
& référence;
* déréférence
int a; // déclaration d'une variable
int *pa; // déclaration d'un pointeur sur entier
pa = &a; // on récupère l'adresse de a
```

Opérations sur les bits :

```
& (et binaire)
| (ou binaire)
^ (ou exclusif binaire)
~ (non binaire)
<< (décalage à gauche)
>> (décalage à droite)
```

Opérateurs unaires :

```
++ (incrémentation)
-- (décrémentation)
+= (addition et affectation)
-= (soustraction et affectation)
*= (multiplication et affectation)
/= (division et affectation)
&= (et binaire et affectation)
|= (ou binaire et affectation)
```

Conditions :

```
if (x<5){  } else {  }
switch (variable) {
   case 1: instruction; break;
   case 2: instruction; break;
   default: instruction;
}
```

Boucles :

```
for (int i=0; i <= 255; i++){   }
while (x<5){  }
do {  } while (x<5);
```

Contrôle de boucles :

```
break; // termine la boucle
continue; // force l'itération suivante
goto label; label:  // existe toujours
```

Variables

Constantes :

```
HIGH | LOW
OUTPUT | INPUT
true | false
143 // Décimal
0173 // Octal
0b11011111 //Binaire
0x7B // Hexadécimal
7U // Force unsigned
10L // Force long
15UL // Force long unsigned
10.0 // Force float
2.4e5 // 240000
```

Types de données :

```
void
boolean(0, 1, false, true)
char(e.g. 'a' -128 à 127)
unsigned char (0 à 255)
byte (0 à 255) (octet)
int(-32768 à 32767)
unsigned int (0 à 65535)
word (0 à 65535)
long(-2147483648 à 2147483647)
unsigned long (0 à 4294967295)
float(-3.4028235E+38 à 3.4028235E+38)
double (comme float)
sizeof(myint) // retourne 2 octets
```

int, word, long peuvent être remplacés par :

```
[u]int{8|16|32|64}_t
[u] pour unsigned, et {8|16|32|64} pour la taille en bits.
```

Tableaux :

```
int myInts[6]; //6 entiers
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
int myMatrice[3][3]; //9 entiers
```

Chaînes de caractères :

```
char S1[15];
char S2[8]={'a','r','d','u','i','n','o'};
char S3[8]={'a','r','d','u','i','n','o','\0'};
//inclus \0 null de fin de chaîne
char S4[ ] = "arduino";
char S5[8] = "arduino";
String S6 = "arduino"; //classe
```

Conversion :

```
char()
byte()
int()
word()
long()
float()
```

Portée :

```
variables globales / locales
public: private: // dans les classes
static // persistante
volatile // en  RAM
const // constante
PROGMEM // en flash
```

Fonctions

Entrées/sorties :

```
pinMode(pin,
[INPUT_PULLUP,INPUT,OUTPUT])
digitalWrite(pin, value)
boolean digitalRead(pin)
```

Analogique :

```
analogReference([DEFAULT,INTERNAL,EXTERNAL])
int analogRead(pin) // de 0 à 1023
analogWrite(pin, value) // PWM
```

E/S spéciales :

```
tone(pin, freqhz)
tone(pin, freqhz, durée_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
[MSBFIRST,LSBFIRST], valeur)
unsigned long pulseIn(pin, [HIGH,LOW])
```

Temps :

```
unsigned long millis() // 50 jours
unsigned long micros() // 70 minutes
```

```
delay(ms)
delayMicroseconds(µs)
```

Maths :

```
min(x, y)    max(x, y)    abs(x)
constrain(x, minval, maxval )
map(val, deL, deH, àL, àH)
pow(x, exposant)    sqrt(x)
sin(rad)    cos(rad)    tan(rad)
```

Nombres aléatoires :

```
randomSeed(analogRead(0)) // dans le setup
long random(max)
long random(min, max)
```

Bits et octets :

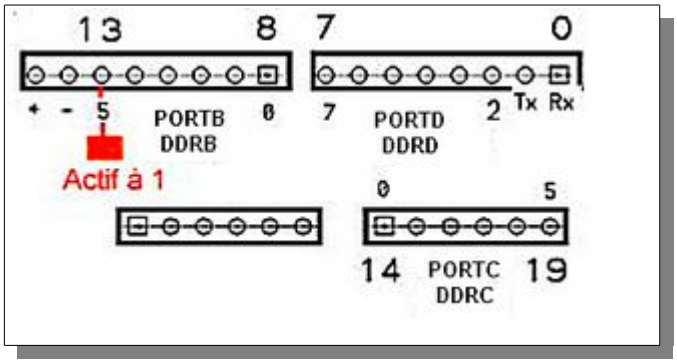
```
lowByte()    highByte()
bitRead(x,bitn)    bitWrite(x,bitn,bit)
bitSet(x,bitn)    bitClear(x,bitn)
bit(bitn) //bitn:  0-LSB 7-MSB
```

Interruptions :

```
// n° 0 sur pin 2 et n° 1 sur pin 3
// variables volatiles dans fonctions
attachInterrupt(n°, fonction,
[LOW,CHANGE,RISING,FALLING]) // dans le
setup
detachInterrupt(n°)
interrupts()
noInterrupts()
```

Acces direct aux ports :

```
PortD : pins 0 à 7 de l'Arduino
PortB : pins 8 à 13 (B6 et B7 non dispos)
bitSet(DDRD,2); ou DDRD = 1 << 2 // D2 (pin2)
en sortie
bitSet(PORTB,3); ou PORTB |= 1 << 3  // B3 (pin
11) à 1
ou PORTB = 0b111111 // B0 à B5 (pins 8 à 13) à
1
```



Bibliothèques

Chaînes de caractères (String) :

```
String chaine="arduino"; // guillemets doubles
String nombre=String(n,[BIN,DEC,HEX])
char charAt(n)
booleann equals(chaine2) // idem ==
booleann equalsIgnoreCase(chaine2);
int length()
String replace("ancien","nouveau")
String substring(n1,[n2])
toInt()
toLowerCase()
toUpperCase()
trim()
```

Port série (Serial) :

```
begin(vitesse) // dans le setup
end()
int available() // nbre d'octets reçus
byte read()
flush()
print() //envoi en ASCII
println() // ajout de \r\n
write() //envoi en binaire
```

EEPROM (#include <EEPROM.h>) :

```
byte read(intAddr)
write(intAddr, myByte)
```

Servo (#include <Servo.h>) :

```
attach(pin , [min_uS, max_uS])
write(angle) // 0-180
writeMicroseconds(uS) //1000-2000, milieu (90°)
à 1500
read() // 0-180
attached() //renvoie un booleen
detach()
```

Moteur pas à pas (#include <Stepper.h>) :

```
Stepper moteur(nbDePas, pinA,pinB,pinC,pinD);
setSpeed(60); // 60 tours/mn
step(100);  // avance de 100 pas
step(-100); // recule de 100 pas
```

I2C (#include <Wire.h>) :

```
begin()  // master
begin(adresse) // exclave @ adresse
beginTransmission(adresse) // étape 1
write(byte)          // étape 2
ou write(tableau, nbre d'octets)
endTransmission()       // étape 3
requestFrom(adresse, nbre d'octets)
int available() // nbre d'octets reçus
byte read()
onReceive(fonction) // fonction() si le master
envoie
onRequest(fonction) // si le master demande
```

SPI (#include <SPI.h>) :

```
// pins 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)
begin()
setBitOrder(MSBFIRST,LSBFIRST)
setDataMode(SPI_MODEm) // m=0,1,2,ou 3
setClockDivider(SPI_CLOCK_DIVn) //
n=2,4,8,16,32,64,ou 128
transfer(byte)
byte transfer(0x00)
end()
```