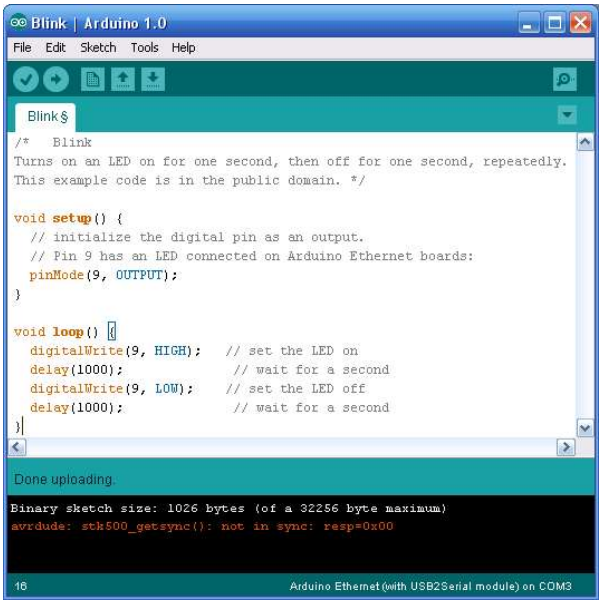
1. Présentation ARDUINO

ARDUINO est une **plateforme matérielle et logicielle de développement** d'applications embarquées.

Côté matériel, elle se compose d'une **carte électronique** basée autour d'un **microcontrôleur** (ATMEL AVR) comportant un certain nombre d'entrées et de sorties (les ports) permettant la connexion de capteurs, ou d'actionneurs.

Le logiciel de programmation des modules ARDUINO est une **application Java**, libre et multi- plateformes, servant **d'éditeur de code et de compilateur**, et qui peut transférer le programme au travers de la liaison USB. Le langage de programmation utilisé est un **mélange de C et de C++**, restreint et adapté aux possibilités de la carte.



Les principaux avantages du système ARDUINO sont :

- **Matériel peu onéreux** : Les cartes ARDUINO sont relativement peu coûteuses comparativement aux autres plateformes de développement. La moins chère des versions du module ARDUINO peut être assemblée à la main, et même les cartes ARDUINO pré-assemblées coûtent moins de 25 €.

- **Matériel** : Les schémas des modules sont publiés sous une licence Creative Commons, et il est possible de réaliser nos propres versions des cartes ARDUINO, en les complétant et en les améliorant.

- **Logiciel gratuit** : Le logiciel ARDUINO et le langage ARDUINO sont disponibles pour être complétés par des programmeurs expérimentés. Le langage peut être aussi étendu à l'aide de librairies C++.

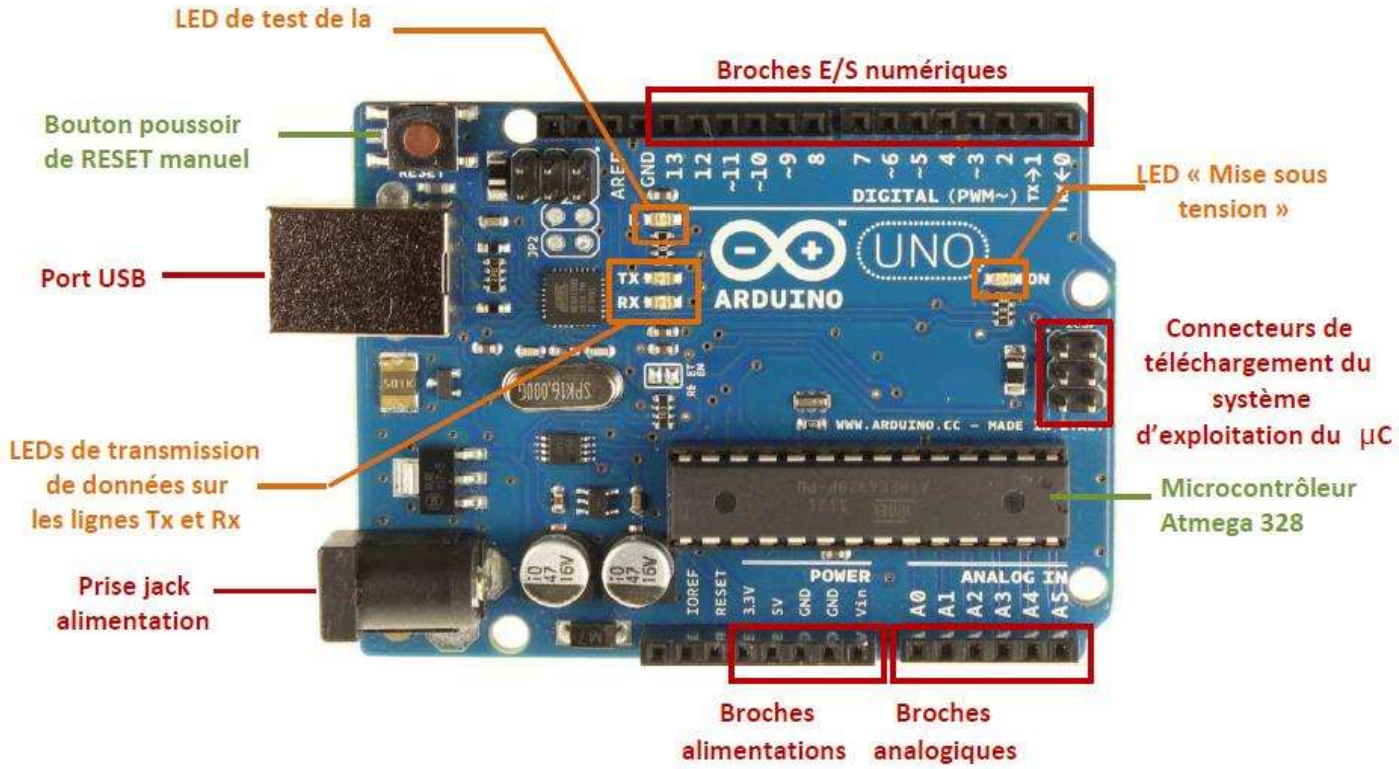
- **Logiciel multi-plateforme** : Le logiciel ARDUINO, écrit en Java, tourne sous les systèmes d'exploitation Windows, Macintosh et Linux.

- **Modules « Shield »** : Cartes supplémentaires se connectant sur le module ARDUINO pour augmenter les possibilités : afficheur graphique couleur, interface Ethernet, GPS, etc.…

2. La carte ARDUINO UNO

Le **modèle UNO** de la société ARDUINO est une carte électronique dont le cœur est un microcontrôleur ATMEL de référence ATMega328. Le **microcontrôleur ATMega328** est un microcontrôleur 8bits de la famille AVR dont la programmation peut être réalisée en langage C.

2.1. Les différents éléments



2.2. Les principales caractéristiques

Microcontrôleur	ATMEGA 328
Tension d'alimentation interne	5V
Tension d'alimentation (recommandée)	7 à 12V
Tension d'alimentation (limites)	6 à 20 V
Entrées/sorties numériques	14 dont 6 sorties PWM (configurables)
Entrées analogiques	6
Courant max par broches E/S	40 mA (200mA cumulé pour l'ensemble des broches)
Mémoire programme Flash	32 Ko dont 0,5 Ko sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 KB
Mémoire EEPROM (mémoire non volatile)	1 KB
Fréquence horloge	16 MHz
Dimensions	68.6mm x 53.3mm

2.3. L'alimentation

La carte ARDUINO UNO peut être alimentée par le **câble USB**, par un **bloc secteur externe** connecté grâce à une **prise « jack » de 2,1mm** ou bien par un bloc de piles dont le raccordement est réalisé par l'intermédiaire des **« GND »** et **« Vin »** du connecteur d'alimentation. L'alimentation extérieure doit présenter une tension comprise entre **7 à 12V**.

La carte génère, par l'intermédiaire de régulateurs intégrés, deux tensions stabilisées : **5 V et 3,3 V**. Ces deux tensions permettent l'alimentation des composants électroniques de la carte ARDUINO. Étant disponibles sur connecteurs placés sur le pourtour des cartes, elles permettent également l'alimentation des modules Shields.

2.4. Les entrées-sorties

La carte « ARDUINO UNO » dispose de **14 E/S numériques** et de **6 entrées analogiques**.

2.4.1. Les entrées-sorties numériques

Chacune des **14 broches numériques (repérées 0 à 13)** peut être utilisée en entrée (input) ou en sortie (output) sous le contrôle du programme. Le sens de fonctionnement pouvant même changer de manière dynamique pendant son exécution.

Elles fonctionnent en logique TTL (0V-5V) ; chacune pouvant fournir (source) ou recevoir un courant maximal de 40 mA et dispose si besoin est d'une résistance interne de 'pull-up'.

Certaines broches peuvent avoir **plusieurs fonctions différentes** choisies par programmation décrites ci-dessous :

N° E/S	N° ligne de port	Fonction
0	**PD0**	**Rx** : Entrée liaison série synchrone
1	**PD1**	**Tx** : Sortie liaison série synchrone
2	**PD2**	**INT0** : Entrée interruption externe
3	**PD3**	**INT1** : Entrée interruption externe
		OC2B : PWM modulation à largeur d'impulsion
4	**PD4**	**T0** : Entrée Timer/compteur 0
		XCK : Entrée horloge
5	**PD5**	**T1** : Entrée Timer/compteur 1
		OC0B : Sortie module PWM modulation à largeur d'impulsion
6	**PD6**	**OC0A** : Sortie module PWM modulation à largeur d'impulsion
		AIN0 : Entrée comparateur analogique
7	**PD7**	**AIN1** : Entrée comparateur analogique
8	**PB0**	**CLKO** : Sortie de l'horloge de fonctionnement
		ICP1 : Entrée de capture Timer/compteur 1
9	**PB1**	**OC1A** : Sortie module PWM modulation à largeur d'impulsion
10	**PB2**	**SS** : Sélect Slave liaison SPI
		OC1B : Sortie module PWM modulation à largeur d'impulsion
11	**PB3**	**MOSI** : Sortie liaison SPI
		OC2A : Sortie module PWM modulation à largeur d'impulsion
12	**PB4**	**MISO** : Entrée liaison SPI
13	**PB5**	**SCK** : Horloge liaison SPI

2.4.2. Les entrées analogiques

Les six entrées analogiques, repérées **A0 à A5 (PC0 à PC5)**, peuvent admettre toute une **tension analogique** comprise entre **0 et 5V** (par défaut mais cela peut être modifié). Ces entrées analogiques sont gérées par un **convertisseur analogique/numérique de 10 bits**, dont la sortie peut varier de 0 à 1023.

Les entrées A4 et A5 peuvent également être utilisées respectivement comme la ligne de donnée SDA et la ligne d'horloge SCL de l'interface série I2C.

2.5. Les mémoires

Le microcontrôleur ATmega 328 dispose de **32ko de mémoire Flash** permettant de stocker le programme à exécuter.

Il contient aussi de **2ko de mémoire vive (SRAM)**. Cette mémoire est généralement utilisée pour stocker les résultats temporaires lors de calculs. Elle peut être lue et écrite à tout instant par le microcontrôleur mais son contenu est perdu dès que la n'est plus alimentée.

L'ATmega 328 dispose également **1ko mémoire EEPROM** pour permettre au programme de stocker des données persistantes. Le contenu de cette mémoire est accessible grâce aux fonctions de la librairie « EEPROM ».

2.6. L'horloge

L'horloge est pilotée par **quartz** et fonctionne à la fréquence de **16 MHz**.

2.7. La communication

La carte « ARDUINO UNO » a de nombreuses possibilités de communications avec l'extérieur. L'Atmega328 possède une **interface de communication série** accessible, grâce aux **broches numériques 0 (Rx) et 1 (Tx)**.

D'autre part elle supporte **le bus I2C** accessible, grâce aux **broches analogiques 4 (SDA) et 5 (SCL)** et la **liaison série synchrone SPI** grâce aux **broches numériques 10 (SS), 11 (MOSI), 12(MISO) et 13 (SCX)**.

2.8. Le reset

A la mise sous tension un **reset automatique** permet au programme contenu en mémoire du microcontrôleur de démarrer automatiquement dès que la carte ARDUINO est alimentée.

La carte « Arduino UNO » est également équipée d'un bouton poussoir de **reset manuel**. Un appui sur celui-ci permet de relancer l'exécution d'un programme si nécessaire, soit parce qu'il s'est « planté » soit tout simplement parce que l'on souhaite le faire repartir de son début.

2.9. La protection de surintensité USB

Par mesure de sécurité pour l'ordinateur auquel sera relié l'ARDUINO, **un fusible** est présent sur la connexion d'alimentation 5 V de la prise USB. Toute consommation **supérieure à 500mA**, provoque le **déclenchement de ce fusible**, protégeant ainsi le port USB de l'ordinateur auquel la carte est reliée.

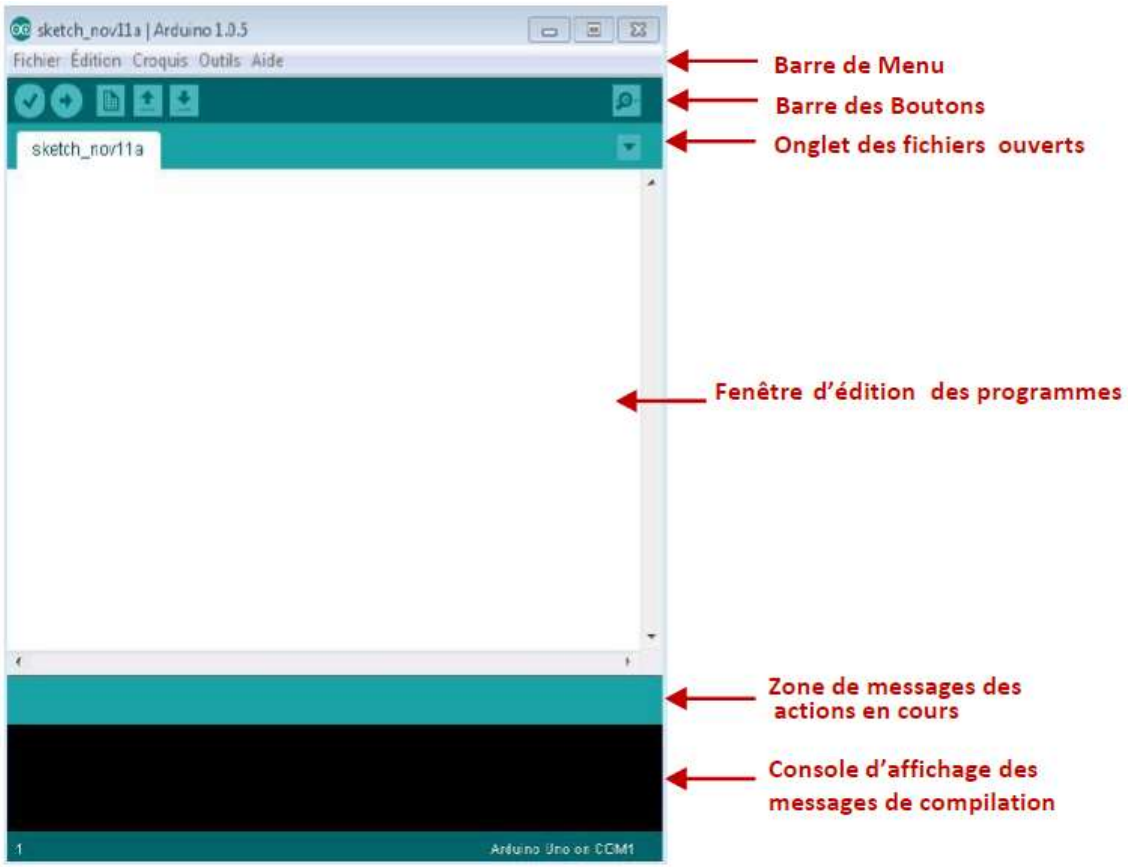
3. La programmation de la carte ARDUINO

Le **logiciel ARDUINO** a pour fonctions principales :

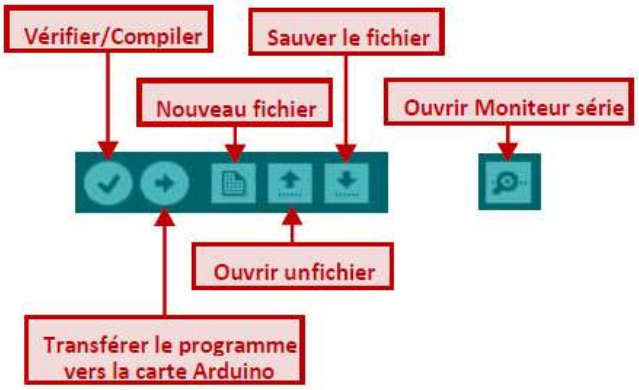
- De pouvoir écrire et compiler des programmes pour la carte ARDUINO
- De se connecter avec la carte ARDUINO pour y transférer les programmes
- De communiquer avec la carte ARDUINO

3.1. L'interface de programmation

L'interface de programmation du logiciel ARDUINO est la suivante :

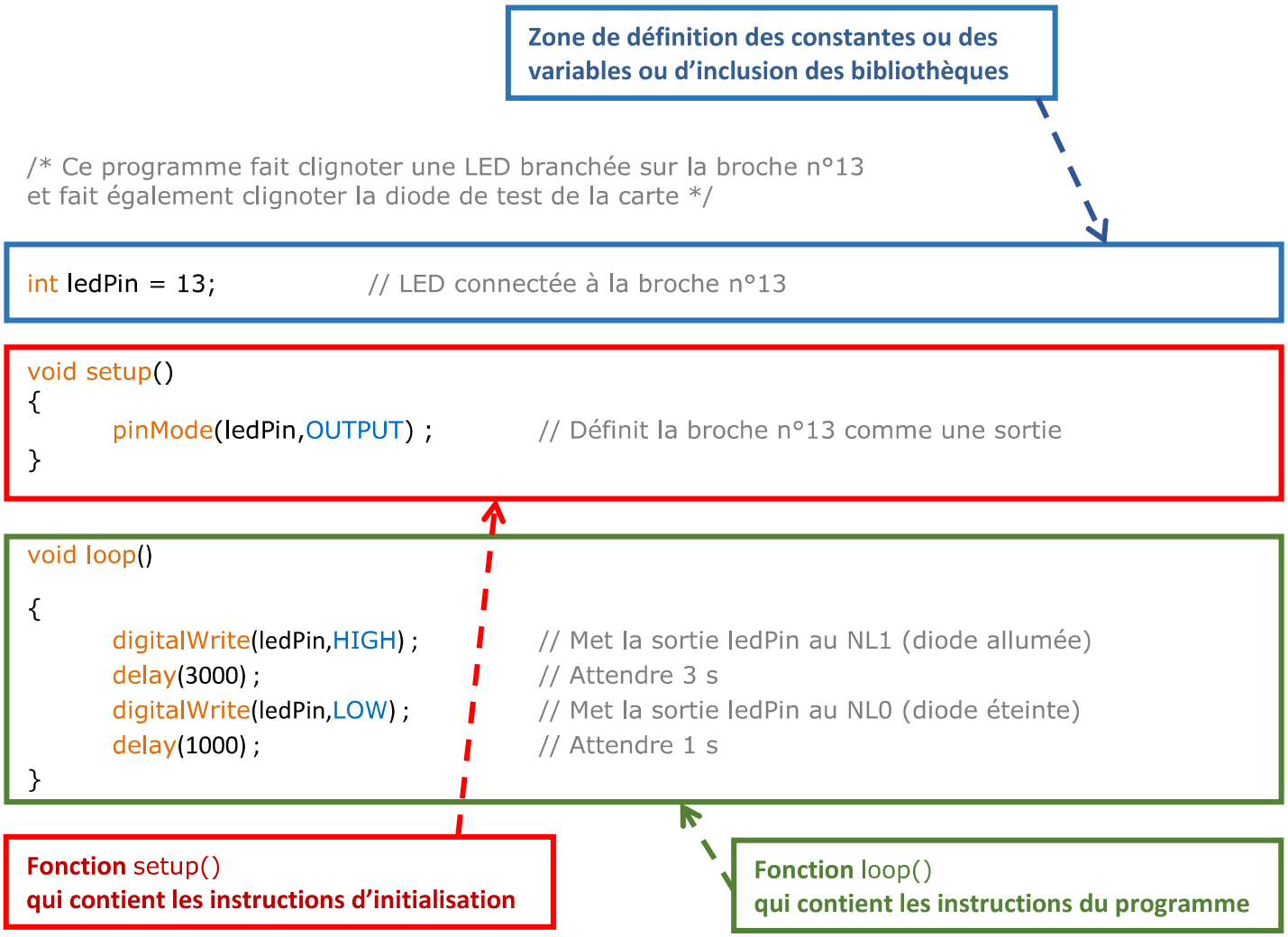


La **« Barre de Boutons »** qui donne un accès direct aux fonctions essentielles du logiciel. Les différents boutons sont :



3.2. La structure d'un programme

Un programme destiné à une carte ARDUINO est constitué de **3 parties** :



La première partie permet de **définir les constantes et les variables** en déclarant leur type. Elle permet également l'inclusion des bibliothèques utilisées dans le programme au moyen de #include.

La fonction setup() contient **les instructions d'initialisation ou de configuration des ressources** de la carte comme par exemple, la configuration en entrée ou sorties des broches d'E/S numériques, la définition de la vitesse de communication de l'interface série, etc.. Cette fonction **n'est exécutée qu'une seule fois** juste après le lancement du programme.

La fonction loop() contient **les instructions du programme** à proprement parlé. Cette fonction sera **répétée indéfiniment** tant que la carte ARDUINO restera sous tension.

3.3. La syntaxe du langage ARDUINO

La cinquantaine d'éléments de la syntaxe ARDUINO est visible ici http://www.arduino.cc/en/Reference/HomePage ainsi qu'à partir du document "index.html" (dans le dossier "Reference" que vous avez téléchargé avec ARDUINO), également accessible dans le menu "Aide" du logiciel.

3.4. Les commentaires

Pour placer des commentaires sur une ligne unique ou en fin de ligne, il faut utiliser la syntaxe suivante :

```
// Cette ligne est un commentaire sur UNE SEULE ligne
```

Pour placer des commentaires sur plusieurs lignes :

```
/* Commentaire, sur PLUSIEURS lignes qui sera ignoré par le programme, mais
pas par celui qui lit le code */
```

3.5. Les données, variables et constantes

Les différents types utilisés avec la programmation ARDUINO sont :

Nom	Description
boolean	Donnée logique (true ou false) sur 8 bits
byte	Entier non signé sur 8 bits
int	Entier signé sur 16 bits
unsigned int	Entier non signé sur 16 bits
long	Entier signé sur 32 bits
unsigned long	Entier non signé sur 32 bits
float	Nombre à virgule flottant sur 32 bits
char	Caractère sur 8 bits. Seuls les caractères ayant pour code ASCII une valeur 0 à 127 sont définis

Une constante peut être définie au moyen de const ou de #define :

```
#define N 2          //Toute variable N rencontrée dans le programme sera remplacée par la valeur 2
const byte N=2       // N est constante de type « byte » et de valeur 2
```

Un certain nombre de noms de constantes sont prédéfinis dans le langage « ARDUINO » :

Nom	Description
true	Donnée binaire égale à 1 ou donnée numérique différente de 0
false	Donnée binaire ou numérique égale à 0
HIGH	Génération d'un niveau de tension haut sur une des sorties numériques
LOW	Génération d'un niveau de tension bas sur une des sorties numériques
INPUT	Configuration d'une broche numérique en entrée
OUTPUT	Configuration d'une broche numérique en sortie

3.6. Les opérateurs

3.6.1. Les opérateurs arithmétiques

Symbole	Description	Exemple
+	Addition	c = a + b ;
-	Soustraction	c = a - b ;
*	Multiplication	c = a * b ;
/	Division	c = a / b ;
%	Modulo (reste de la division entière)	c = a % b ;

3.6.1. Les opérateurs logiques

Symbole	Description	Exemple
&	ET	c = a & b ;
|	OU	c = a | b ;
^	OU exclusif	c = a ^ b ;
~	NON	b = ~a ;
>>	Décalage à droite des bits	c = a >> b ; //a décalé b fois à droite
<<	Décalage à gauche des bits	c = a << b ; //a décalé b fois à gauche

3.6.2. Les affectations

Symbole	Description	Exemple
=	Affection ordinaire	a = b ;
+=	Additionner de	a += b ; → a = a + b ;
-=	Soustraire de	a -= b ; → a = a - b ;
*=	Multiplier par	a *= b ; → a = a * b ;
/=	Diviser par	a /= b ; → a = a / b ;
%=	Reste de la division entière par	a %= b ; → a = a % b ;
&=	ET avec	a &= b ; → a = a & b ;
|=	OU avec	a |= b ; → a = a | b ;
--	Soustraire de 1 (décrémentation)	a -- ; → a = a - 1 ;
++	Additionner de 1 (incrémentation)	a ++ ; → a = a + 1 ;

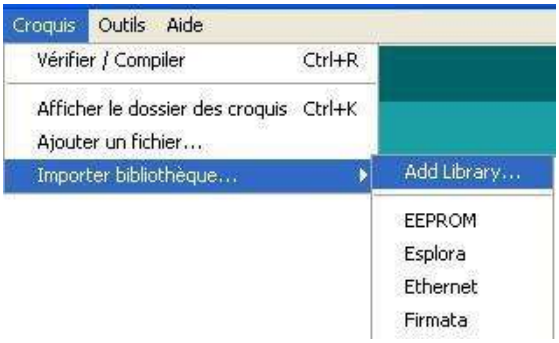
3.6.3. Les opérateurs de comparaisons

Symbole	Description	Exemple
&&	ET logique	if (a && b)
||	OU logique	if (a || b)
==	Égal à	if (a == b)
!=	Différent de	if (a != b)
>	Supérieur à	if (a > b)
<	Inférieur à	if (a < b)
>=	Supérieur ou égal à	if (a >= b)
<=	Inférieur ou égal à	if (a <= b)

3.7. Les bibliothèques de fonctions

Une bibliothèque est un ensemble de fonctions utilitaires mises à disposition des utilisateurs de l'environnement Arduino. Les fonctions sont regroupées en fonction de leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc).

L'IDE ARDUINO comporte par défaut plusieurs bibliothèques externes. Pour les importer dans votre programme, vous devez utiliser le menu suivant :



L'instruction suivante sera alors ajoutée au début de votre programme : #include <la_bibliothèque.h>

Exemple :

```
#include <SPI.h>              // Importation de la bibliothèque SPI
```

Certaines fonctions sont déjà présentes dans l'IDE ARDUINO, les principales sont expliquées ci-dessous.

3.7.1. Les fonctions de gestion du temps

La fonction « delay »

delay(tempo) ;

Cette fonction **génère une pause** dont la durée est égale à tempo × 1 ms.

La variable tempo doit être de type unsigned long ce qui peut autoriser des temporisations qui peuvent être très longues.

```
delay(1000) ;        // Pause de 1 s
```

La fonction « delayMicroseconds »

delayMicroseconds(tempo) ;

Cette fonction **génère une pause** dont la durée est égale à tempo × 1 µs.

La variable tempo doit être de type unsigned int.

⚠ La durée de la pause n'est assurée que des valeurs supérieures à 3 µs.

```
delayMicroseconds(100) ;          // Pause de 100 µs
```

La fonction « *millis()* »

Valeur = millis() ;

Cette fonction **retourne le nombre de millisecondes depuis le démarrage du programme** par la carte ARDUINO.

La valeur retournée est de type unsigned long. Le nombre de millisecondes retourne à zéro environ après 50 jours.

La fonction « *micros()* »

Valeur = micros() ;

Cette fonction **retourne le nombre de microsecondes depuis le démarrage du programme** par la carte ARDUINO.

La valeur retournée est de type unsigned long

⚠ Cette fonction a une résolution de de 4µs (avec une horloge de 16MHz), La valeur retournée est donc un multiple de 4.

Le nombre de millisecondes retourne à zéro environ après 70 minutes.

3.7.2. Les fonctions de gestion des E/S numériques

La fonction « pinMode »

pinMode(Numéro broche, Sens de fonctionnement) ;

Cette fonction permet de **configurer le sens de fonctionnement**, entrée ou sortie, **de la broche sélectionnée**.

```
pinMode (12,OUTPUT) ;    // La broche 12 est configurée comme une sortie
```

La fonction « digitalWrite »

digitalWrite(Numéro broche, Niveau logique) ;

Cette fonction permet **d'imposer un niveau logique** haut ou bas **sur la sortie numérique sélectionnée**.

```
digitalWrite(12,HIGH) ;    // La sortie 12 est placée au niveau logique haut (NL1)
```

La fonction « digitalRead »

Valeur = digitalRead(Numéro broche) ;

Cette fonction permet de **lire le niveau logique sur l'entrée numérique sélectionnée**. Cette fonction ne retourne que deux valeur HIGH ou LOW.

```
etat = digitalRead(11) ;          // La variable etat est égale au niveau logique de l'entrée 11
```

3.7.3. Les fonctions de gestion des sorties PWM

Les µC qui équipent les cartes Arduino sont capables de générer des signaux à Modulation à Largeur d'Impulsions (MLI) ou « Pulse Width Modulation » (PWM).

Il s'agit de signaux logiques rectangulaires de fréquence égale à 490 Hz et à rapport cyclique programmable.

La fonction « analogWrite »

analogWrite(Numéro sortie, Rapport cyclique) ;

Cette fonction permet de **générer, sur la sortie sélectionnée, un signal PWM avec le rapport cyclique désiré**. La variable « Rapport cyclique » doit être de type int, c'est-à-dire qu'elle peut être comprise entre 0 et 255 pour un rapport cyclique, pour le signal généré, compris entre 0 et 100 %.

Le signal PWM est généré à partir de l'instant de l'exécution de la fonction « analogWrite » et jusqu'à ce qu'une nouvelle instruction « analogWrite » ou une instruction « digitalRead » ou « digitalWrite » soit exécutée sur la même broche.

⚠ La valeur du rapport cyclique n'est garantie que pour des valeurs supérieures à 10 %.

```
analogWrite(2,64) ;                // Signal PWM de rapport cyclique de 25 % sur la sortie analogique 2
```

3.7.4. Les fonctions de gestion des entrées analogiques

Les entrées analogiques sont connectées à un convertisseur analogique numérique (CAN) 10 bits. La sortie du CAN génère une donnée égale à 0 lorsque la tension d'entrée est nulle et une donnée égale à 1023 lorsque la tension d'entrée est égale à la tension de référence du CAN.

La fonction « analogReference »

analogReference(Type) ;

Cette fonction permet de **définir la tension de référence du CAN**.

La variable type peut prendre les valeurs suivantes :

- **DEFAUT** : La tension de référence est égale à la tension d'alimentation de la carte Arduino (5V pour la carte Arduino Uno) ;
- **INTERNAL** : La tension de référence est égale à 1,1 V pour la carte Arduino Uno ;
- **EXTERNAL** : La tension de référence est égale à la tension appliquée sur l'entrée externe AREF de la carte Arduino.

⚠ La tension appliquée sur une entrée analogique ne doit jamais être supérieure à la tension d'alimentation de la carte.

La fonction « analogRead »

Valeur = analogRead(Numéro entrée) ;

Cette fonction permet de **lire le résultat de la conversion analogique numérique de la tension présente sur l'entrée analogique sélectionnée**.

La donnée retournée étant comprise entre 0 et 1023, doit être de type int.

⚠ La durée de la conversion est d'environ 100 µs.

```
tension = analogRead(1) ;          // tension est égale au résultat de la CAN de l'entrée analogique 1
```

3.7.5. Les fonctions particulières de gestion des entrées-sorties

La fonction « tone »

tone(Numéro sortie, Fréquence, Durée) ;

Cette fonction permet de **générer, sur la sortie sélectionnée, un signal carré** (rapport cyclique de 50 %) de fréquence programmable et pendant la durée désirée.

La donnée « Fréquence » doit être donnée en Hz et la donnée « Durée » doit être fixée en ms. Si cette durée n'est pas précisée, le signal est généré jusqu'à ce que la fonction « notone » soit exécutée.

La fonction « notone »

notone(Numéro sortie) ;

Cette fonction permet de **stopper la génération du signal carré sur la sortie sélectionnée**.

La fonction « pulseIn »

durée = pulseIn(Numéro entrée, Type, Delai max) ;

Cette fonction permet de **mesurer la durée d'une impulsion sur l'entrée sélectionnée**. Le résultat retourné est du type unsigned long.

La valeur de la donnée « Type » peut être HIGH dans le cas d'une impulsion au NL1 ou LOW d'une impulsion au NL0.

La variable « Delai max » permet de définir le délai d'attente de l'impulsion. Lorsque ce délai est dépassé et qu'aucune impulsion ne s'est pas produite, la fonction retourne une valeur nulle. Cette donnée est du type unsigned long et est exprimé en µs. Si cette donnée n'est pas précisée, la valeur par défaut sera de 1 s.

 Le résultat fourni n'est considéré comme correct que pour des impulsions de durée comprise entre 10 µs et 3 min.

3.7.6. Les fonctions de manipulation de bits

La fonction « lowByte »

resultat = lowByte(Donnée) ;

Cette fonction permet **d'extraire les 8 bits de poids faible de la variable « Donnée ».**

La valeur retournée est de type byte alors que la variable « Donnée » peut être de n'importe quel type (donc n'importe quelle taille).

La fonction « highByte »

resultat = highByte(Donnée) ;

Cette fonction permet **d'extraire le deuxième octet en partant de la droite de la variable « Donnée ».**

La valeur retournée est de type byte alors que la variable « Donnée » peut être de n'importe quel type.

La fonction « bitRead »

resultat = bitRead(Donnée, n) ;

Cette fonction permet de **lire le n^{ième} bit de la variable « Donnée ».**

La fonction « bitWrite »

bitWrite(Donnée, n, Niveau) ;

Cette fonction permet **d'imposer un niveau logique sur le nième bit de la variable « Donnée ».**

La fonction « bitSet »

bitSet(Donnée, n) ;

Cette fonction permet **d'imposer un 1 logique sur le n^{ième} bit de la variable « Donnée ».**

La fonction « bitClear »

bitClear(Donnée, n) ;

Cette fonction permet **d'imposer un 0 logique sur le n^{ième} bit de la variable « Donnée ».**

3.7.7. Les fonctions de gestion du port série asynchrone

La carte ARDUINO UNO dispose d'un port série asynchrone accessible via les E/S numériques 0 (ligne Rx) et 1 (Ligne Tx).

Le port série asynchrone des microcontrôleurs qui équipent les cartes ARDUINO disposent d'une mémoire tampon capable de mémoriser jusqu'à 128 caractères reçus.

La fonction « Serial.begin »

Serial.begin(Vitesse) ;

Cette fonction qui doit être appelée au moins une fois, généralement dans la fonction setup(), permet de **définir la vitesse utilisée par la liaison série.**

La valeur prise par la variable « Vitesse » doit être une des vitesses définies par la norme RS232.

La fonction « Serial.end »

Serial.end() ;

Cette fonction permet de **désactiver la liaison série asynchrone** et donc de libérer les broches numériques 0 et 1.

La fonction « Serial.available »

Serial.available() ;

Cette fonction permet de **connaître le nombre de caractères reçus** et donc contenus dans la mémoire tampon en attente de lecture par le programme. La variable « nombre » est de type int.

La fonction « Serial.read »

Serial.available() ;

Cette fonction permet de **lire le premier caractère disponible** dans la mémoire tampon de réception. La variable « caractère » est de type int. La valeur retournée est -1 si aucun caractère n'a été reçu (mémoire tampon vide).

La fonction « Serial.flush »

Serial.flush() ;

Cette fonction permet de **vider la mémoire tampon**.

La fonction « Serial.write »

Serial.write(Donnée) ;

Cette fonction permet **d'émettre une ou plusieurs données sur la liaison série** sous la forme binaire brute. Si la donnée est de type chaîne de caractères, les caractères sont transmis, sous le format ASCII, les uns après les autres.

La fonction « Serial.print »

Serial.print(Donnée, format) ;

Cette fonction permet **d'émettre une ou plusieurs données sur la liaison série en précisant le codage utilisé**.

Si la variable « Format » n'est pas précisée, une donnée numérique sera considérée comme décimale, une chaîne de caractères sera transmise tel quelle et une donnée à virgule flottante sera transmise uniquement avec deux décimales.

Sinon la variable « Format » permet de préciser le codage ou la base de numération pour les données numériques. Elle peut prendre une des valeurs suivantes : BIN (binaire), OCT (Octal), HEX (hexadécimal), DEC (Décimal) ou BYTE (codage ASCII). Pour les données à virgule flottante, la variable « Format » peut prendre une valeur numérique comprise entre 0 et 7 correspondant au nombre de décimales à transmettre.

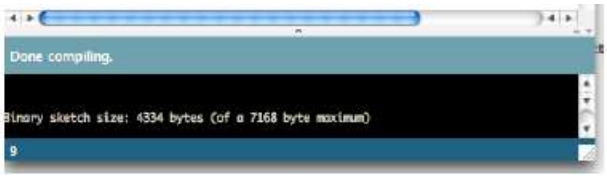
```
Serial.print(100101,BIN) ; // Transmission de la donnée (100101)2
Serial.print(4B,HEX) ;     // Transmission de la donnée (4B)16
Serial.print(K,BYTE) ;     // Transmission de la donnée (K)ASCII
Serial.print(1.23456,0) ;  // Transmission de la donnée 1
Serial.print(1.23456,5) ;  // Transmission de la donnée 1.23456
```

3.8. Les étapes de développement d'un programme

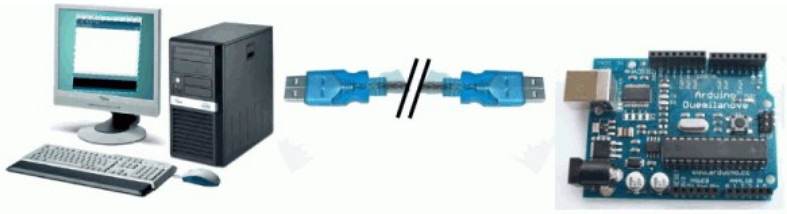
Éditer le programme à partir de ARDUINO

Vérifier et compiler le code à l'aide du bouton « Vérifier ».

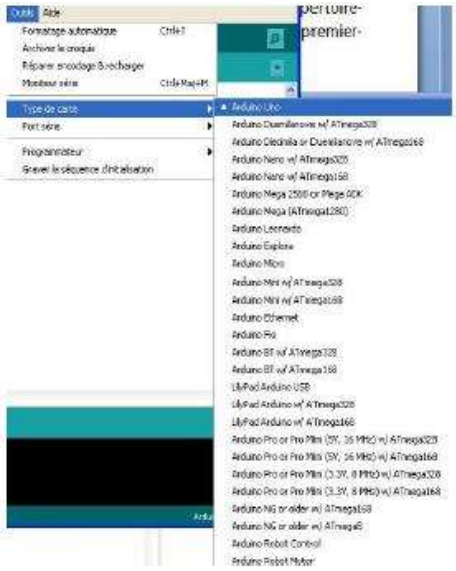
S'il n'y a pas d'erreurs on verra s'afficher : "Compilation terminée", suivi de la taille du programme.



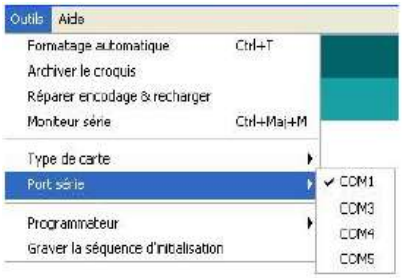
Connectez votre carte ARDUINO à votre ordinateur en utilisant votre câble USB. La LED verte d'alimentation (notée PWR) devrait s'allumer.



Sélectionner la carte ARDUINO utilisée à partir du menu Outil → Type de carte



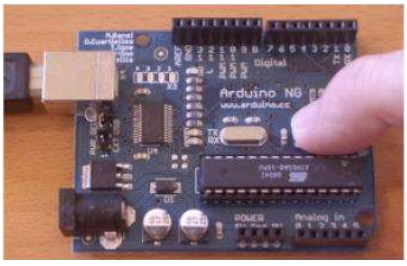
Sélectionner le bon port série à partir du menu Outils → Port série



Charger le programme dans la carte ARDUINO en cliquant sur le bouton « Téléverser ».

Une fois le Téléversement terminé, le logiciel ARDUINO doit afficher un message " Téléversement terminé"

Vérifier le fonctionnement du programme sur la carte ARDUINO



Si nécessaire, appuyer sur le bouton de réinitialisation de la carte pour redémarrer votre programme