C pour ARDUINO

Editeur de .ino

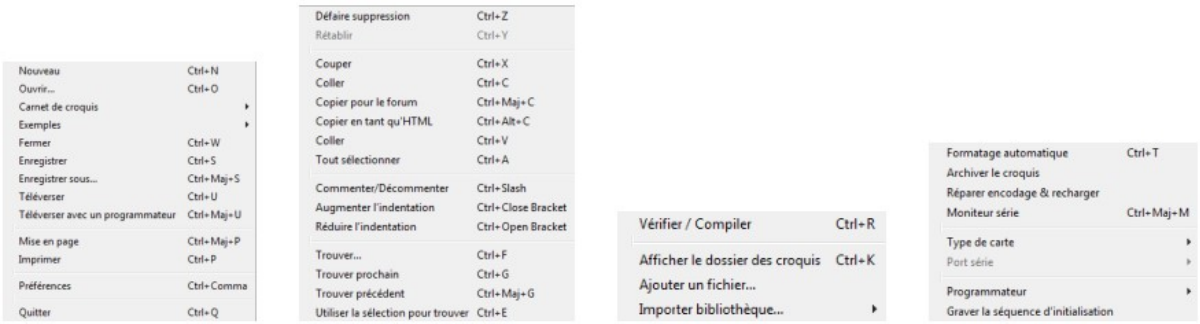

Figure 5: Menus “Fichier”, “Édition”, “Croquis” et “Outils” de l’environnement de développement ARDUINO.

Raccourcis Ctrl+U : Upload Ctrl+T : Auto Format

8 Types :

- **boolean**, qui correspond à une variable logique[19], qui peut valoir soit 0 soit 1 ;
- **char**, qui correspond à un nombre entier *relatif* codé en notation en complément à 2 sur 8 bits, donc compris entre -128 et $+127$;
- **byte** (ou son équivalent `unsigned char`), qui correspond à un nombre entier *naturel* codé sur 8 bits, donc compris entre 0 et $+255$;
- **int**, qui correspond à un nombre entier *relatif* codé en notation en complément à 2 sur 16 bits, donc compris entre -32768 et $+32767$;
- **word**, (ou son équivalent **unsigned int**), qui correspond à un nombre entier *naturel* codé sur 16 bits, donc compris entre 0 et 65535 ;
- **long**, qui correspond à un nombre entier *relatif* codé en notation en complément à 2 sur 32 bits, donc compris entre $-2\,147\,483\,648$ et $+2\,147\,483\,647$;
- **unsigned long**, qui correspond à un nombre entier *naturel* codé sur 32 bits, donc compris entre 0 et $+4\,294\,967\,295$;
- **float**, qui correspond à un nombre réel (ou nombre “à virgule”) au format IEEE 754 simple précision sur 32 bits. Des variables de type `float` peuvent être déclarées, affectées et utilisées dans des opérations de calcul arithmétique. Il est cependant important de retenir que le microcontrôleur met beaucoup plus de

Type NomDeTableau[n] ;

constante Type NomDeConstante=valeur ;

Ex tableau de constantes

```
const float Coefficients[5]= {1.6, 2.4, 3.2, 2.8, 1.7};
```

Affectation :

`x=17;`, `x=B10001;` et `x=0x11;`.

```
int Tableau[3];
Tableau[0]=20; Tableau[1]=1019; Tableau[2]=3*Tableau[0]+7;
```

- Les affectations multiples sont possibles : `x1=x2=x3=10;` met la valeur 10 dans les trois variables `x1`, `x2` et `x3` ;
- Des opérateurs d’affectation composés peuvent être utilisés pour écrire de manière abrégée des instructions du type “`x=x+1;`” sous la forme “`x+=1;`”. Les opérateurs composés utilisables sont `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=` et `>>=`, Ils permettent d’éviter d’avoir à écrire le nom d’une variable de chaque coté du signe égal, ce qui se justifie principalement lorsque ce nom de variable est compliqué ;
- Il existe un opérateur d’affectation conditionnelle : par exemple, l’instruction “`max=(a>b)?a:b;`” met dans la variable `max` la valeur de `a` si `a>b`, et la valeur de `b` dans le cas contraire.

Transmission et affichage de variables

`Serial.print(val)` ou `Serial.print(val, format)`

Serial.begin (9600) 300, 1200,2400, 4800, 9600,14400, 19200,28800, 57600,115200 baud ds setup

Serial.println(x)

instruction	résultat à l'écran
affichage de nombres entiers	
`Serial.print(78);`	78
`Serial.print(78, DEC);`	78
`Serial.print(78, BIN);`	1001110
`Serial.print(78, OCT);`	116
`Serial.print(78, HEX);`	4E
affichage de nombres réels	
`Serial.print(1.23456);`	1.23
`Serial.print(1.23456, 0);`	1
`Serial.print(1.23456, 2);`	1.23
`Serial.print(1.23456, 4);`	1.2346
affichage de caractères et de chaînes	
`Serial.print(78);`	N
`Serial.print('N');`	N
`Serial.print("Hello world.")`	Hello world

EEPROM

La mémoire EEPROM de l'ARDUINO Nano permet de stocker 1024 octets, chaque octet étant stocké à une adresse comprise entre 0 et 1023. Les deux instructions de lecture et d'écriture dans l'EEPROM sont regroupées dans la bibliothèque appelée `EEPROM`. Pour utiliser cette bibliothèque, il est nécessaire d'écrire une directive `#include <EEPROM.h>` au début du programme. Ces deux fonctions sont :

- la fonction **EEPROM.read(address)**, qui renvoie la donnée contenue à l'adresse fournie en paramètre ;
- l'instruction **EEPROM.write(address,data)**, qui écrit à l'adresse indiquée la donnée (de type `byte`) fournie en second paramètre.

```
#include <EEPROM.h>                  // utilisation de la bibliotheque EEPROM

unsigned int annee;

// setup ***************************************************

void setup()
{// initialisation de la liaison serie avec le pc
 Serial.begin(9600); Serial.println("Demarrage du programme");
 // ecriture dans 4 octets de la memoire EEPROM
 EEPROM.write(0,14); EEPROM.write(1,7); EEPROM.write(2,6); EEPROM.write(3,253);
}

// loop ****************************************************

void loop()
{
 Serial.print( EEPROM.read(0), DEC); Serial.print("/");    // affichage des 2 premiers octets
 Serial.print( EEPROM.read(1), DEC); Serial.print("/");
 Serial.println( EEPROM.read(2)*256+EEPROM.read(3), DEC); // affichage d'un entier 16 bits
 delay(2000);                                             // attend 2 s avant la reprise
}
```

Opérateurs et fonctions prédéfinies

opérateurs logiques		
~	négation de chaque bit (complément à un)	~B10011010 renvoie B01100101
&	et logique	B10011010 & B10100011 renvoie B10000010
|	ou logique	B10011010 | B10100011 renvoie B10111011
^	ou exclusif	B10011010 ^ B10100011 renvoie B00111001
<<	décalage à gauche	B10011010 << 1 renvoie B00110100
>>	décalage à droite	B10011010 >> 1 renvoie B01001101

Figure 8: opérateurs logiques disponibles.

fonctions et opérateurs applicables aux nombres de tout type	
-	négation (complément à deux)
abs	valeur absolue (d'un nombre signé 16 bits)
+	addition
-	soustraction
*	multiplication
/	division exacte ou euclidienne (ou euclidienne), suivant le types des variables
max	calcule le maximum de 2 nombres : y=max(x,100) sera égal à x si celui-ci est supérieur à 100, et égal à 100 sinon.
min	calcule le minimum de 2 nombres : y=min(x,100) sera égal à x si celui-ci est inférieur à 100, et égal à 100 sinon.
constrain	constrain(x, a, b) contraint x à rester entre a et b.
fonctions applicables seulement aux nombres entiers	
%	reste de la division euclidienne de deux nombres entiers : 17 % 5 renvoie 2
map	map(x, x1, x2, y1, y2) calcule une transformation linéaire $y = a\,x + b$, où a et b sont tels que $y_1 = a\,x_1 + b$ et $y_2 = a\,x_2 + b$.
bitRead	bitRead(x, n) renvoie le niveau logique du bit n de la variable x.
bitWrite	bitWrite(x, n, level) met le bit n de la variable x au niveau logique level.
fonctions qui renvoient un résultat de type double	
sqrt	racine carrée
square	carré
log	fonction logarithme népérien
log10	fonction logarithme décimal
exp	fonction exponentielle
pow	pow(x,n) élève x à la puissance n.
sin	sinus
cos	cosinus
tan	tangente
atan	arctangente
atan2	atan2(y,x) arctangente de y/x entre 0 et 2π.

Figure 9: opérateurs arithmétiques et fonctions disponibles.

Delay (n) ; en ms

Structure conditionnelles et de répétition

Structure conditionnelle (ex : structures imbriquées)

```
if (condition1)
 {instructions executees si condition1 est vraie}
else if (condition2)
 {instructions executees si condition1 est fausse et si condition2 est vraie}
else
 {instructions executees si condition1 et condition2 sont fausses}
```

opérateurs de comparaison	
==	égal
!=	différent
<	inférieur
<=	inférieur ou égal
>	supérieur
>=	supérieur ou égal

Figure 10: opérateurs de comparaison.

association de comparaisons	
!	négation
&&	et logique
||	ou logique

Figure 11: opérateurs d'association de comparaisons.

Structure de répétition

- Il existe plusieurs structures de répétition (ou boucles). La première est la structure de répétition avec pré-condition, qui est utilisée pour répéter un ensemble d'instructions tant qu'une condition (appelée *condition de poursuite*) est vraie. La syntaxe est :

```
while (condition)
 {instructions repetees}
```

- La deuxième est la structure de répétition avec post-condition, utilisée lorsque la condition de poursuite doit être placée après le groupe d'instructions répétées. La syntaxe est :

```
do
 {instructions repetees}
while (condition);
```

- La dernière est la structure de répétition à cardinal fini, également appelée *structure de répétition avec compteur* ou tout simplement *boucle for*, utilisée principalement lorsque l'on sait d'avance combien de fois on doit répéter un ensemble d'instructions. La syntaxe est :

```
for (Initialisation;ConditionPoursuite;Progression)
 {instructions repetees}
```

Cette structure est équivalente à

```
 Initialisation;
 while (ConditionPoursuite)
{
 instructions repetees;
 Progression;
}
```

Fonction, procédure

Fonction

```
typeSortie NouvelleFonction(type1 Entree1, type2 Entree2, typeN EntreeN)
{
 instructions
 return expression; // expression doit renvoyer une donnee de type typeSortie
}
```

Procédure : pas de résultat donc pas de « return »

Dans certains cas, l'utilisateur peut souhaiter définir une fonction qui ne renvoie pas de résultat. C'est ce qu'on appelle une **procédure**. Le type du résultat renvoyé par la fonction sera alors déclaré comme **void** (vide), et la fonction ne comportera alors pas d'instruction `return`. Si l'utilisateur souhaite définir une fonction sans arguments, la liste des paramètres d'entrée sera déclarée comme `(void)`, ou plus simplement comme `()`. C'est ce qui est utilisé dans l'exemple ci-dessous :

Les deux éléments essentiels d'un programme ARDUINO , `setup` et `loop`, sont elles aussi des procédures.

Procédure avec sorties multiples par Pointeurs

que d'une copie du contenu des variables utilisées par le programme appelant. Pour qu'une fonction modifie le contenu d'une variable utilisée par le programme appelant, il faut que ce dernier lui communique l'adresse de cette variable, ce que l'on appelle en informatique un **pointeur** sur cette variable. Une fonction à deux entrées et deux sorties sera par exemple définie par

```
void AutreFonction(int e1, int e2, int *s1, int *s2)
```

où `s1` est l'adresse d'une variable de type int, dont le contenu est[23] `*s1`. Cette fonction peut alors être appelée en écrivant

```
AutreFonction(entree1, entree2, &sortie1, &sortie2); // syntaxe correcte
```

où `sortie1` est une variable de type int, dont l'adresse[24] est `&sortie1`. À titre d'exemple, la fonction ci-

Ex

```
void PgcdEtPpcm(unsigned int a, unsigned int b, unsigned int *pgcd, unsigned long *ppcm)
{
 unsigned long ma, copie_ma, mb;          // variables internes a la fonction

 ma=a ; mb=b;                             // premiere partie : calcul du pgcd
 do
  {
   if (ma<mb)
    {copie_ma=ma ; ma=mb ; mb=copie_ma ;} // interversion de a et b
   ma = ma % mb;                          // on remplace a par le reste de
  }                                       // la division euclidienne de a par b
 while (ma>0);                            // tant que ma n'est pas nul
 *pgcd=mb;                                // le resultat est stocke a l'adresse pgcd

 ma=a ; mb=b;                             // deuxieme partie : calcul du ppcm
 while (ma != mb)                         // tant que ma et mb ne sont pas egaux
  {
   if (ma < mb)                           // on augmente le plus petit des deux
    {ma=ma+a;}
   else
    {mb=mb+b;}
  }
 *ppcm=ma;                                // le resultat est stocke a l'adresse ppcm
}
```

Si cette fonction est utilisée dans la boucle principale suivante,

```
void loop()
{
 N1=14; N2 = 30; PgcdEtPpcm(N1,N2,&Pgcd_N1etN2,&Ppcm_N1etN2);
 Serial.print("N1= ");   Serial.print(N1, DEC);
 Serial.print(", N2= "); Serial.print(N2, DEC);
 Serial.print(", pgcd(N1,N2)= "); Serial.print(Pgcd_N1etN2, DEC);
 Serial.print(", ppcm(N1,N2)= "); Serial.print(Ppcm_N1etN2, DEC);
 Serial.println(); delay(1000);           // 1000 ms = 1 s
}
```

on obtiendra le résultat d'exécution suivant :

```
N1= 14, N2= 30, pgcd(N1,N2)= 2, ppcm(N1,N2)= 210
```

Gestion des broches d'entrée-sortie

E/S Digitale 14 broches D0 à D13

- la fonction **pinMode**(pin, direction), qui permet de configurer le sens de la broche pin. Le deuxième paramètre, direction, ne peut prendre que deux valeurs prédéfinies : INPUT et OUTPUT, pour configurer cette broche en entrée ou en sortie. Cette fonction est souvent utilisée lors de la phase d'initialisation (setup) du programme[40]. Par exemple, l'instruction

  ```
  pinMode(SmokeSensorPin, INPUT);
  ```

- la fonction **digitalWrite**(pin, level), qui permet de produire un niveau logique sur la broche pin, qui doit auparavant avoir été configurée en sortie. Le deuxième paramètre, level, est une variable logique contenant le niveau logique à envoyer sur la broche. Deux constantes prédéfinies peuvent être utilisées : HIGH et LOW. Par exemple, l'instruction

  ```
  digitalWrite(LedPin, HIGH);
  ```

  produit un niveau logique haut à la sortie de la broche LedPin, ce qui correspond à une tension de 5 V. L'instruction

  ```
  digitalWrite(LedPin, LOW);
  ```

  génère un niveau logique bas à la sortie de la broche LedPin, ce qui correspond à une tension nulle.

- la fonction **digitalRead**(pin), qui permet de connaître le niveau logique présent sur la borche pin, qui doit auparavant avoir été configurée en entrée. Cette fonction ne renvoie que deux valeurs possibles, qui correspondent aux constantes HIGH et LOW. Par exemple, l'instruction

  ```
  SmokeSensorLevel=digitalRead(SmokeSensorPin);
  ```

  met dans la variable SmokeSensorLevel le niveau logique présent sur la broche SmokeSensorPin.

Il est cependant possible de générer sur les broches 3, 5, 6, 9, 10 et 11 du port d'entrées-sorties logiques un signal logique périodique de fréquence fondamentale égale à 488 Hz environ[44] et dont le rapport cyclique r (le rapport entre la durée pendant laquelle le signal est au niveau logique haut et la période du signal, que l'on appelle *duty cycle* en anglais) varie entre 0 et 255. Ceci peut être facilement obtenu en utilisant la fonction **analogWrite**(pin,r). Si on filtre le signal obtenu à la sortie de la broche pin à l'aide d'un filtre passe-bas (en général, un simple circuit RC suffira), on obtiendra une tension pratiquement constante égale à la valeur moyenne du signal logique périodique, $5\,r/255 = r/51$, donc proportionnelle au rapport cyclique. C'est ce que l'on appelle le principe de conversion numérique-analogique par modulation de largeur d'impulsion (en anglais *pulse width modulation*, *PWM*, voir figure 19).

E/S Analogique tension 0 < u < 5 V

Le convertisseur analogique-numérique intégré dans le microcontrôleur ATMega328 de l'ARDUINO Nano permet d'acquérir des tensions qui proviennent par exemple d'un capteur. C'est un convertisseur unipolaire[43] qui posséde 8 voies (ou canaux) d'entrée, numérotées de 0 à 7, une résolution de 10 bits et un temps de conversion de 100 μs. Son utilisation s'effectue simplement à l'aide de la fonction **analogRead**(pin), qui fournit le résultat de la conversion de la tension appliquée sur la broche pin. Ce résultat est un nombre compris entre 0 et 1023, que l'on peut stocker dans une variable de type int ou unsigned int. Par exemple, l'instruction

```
Tension=5.0*analogRead(HumiditySensorChannel)/1024.0;
```

```
Vcc     = analogReadVcc();
Tension = Vcc * analogRead(HumiditySensorChannel) / 1024.0;
```

Communication par liaison série

Pour faire la tâche qui lui est demandée, un microcontrôleur peut être amené à échanger des informations avec d'autres circuits électroniques élaborés, voire avec d'autres microcontrôleurs. Dans de telles situations, le système de transmission de données par **liaison série** (où les chiffres binaires qui correspondent aux données sont envoyés les uns derrière les autres sur la même broche). Le langage de programmation de l'environnement propose plusieurs fonctions pour gérer ce mode de communication. Elles sont présentées en détail dans le manuel de référence de l'environnement ARDUINO, et on ne trouvera ici qu'une présentation sommaire dont le seul but est de faire connaître l'existence de ces instructions. Il existe deux types de liaison série :

- Les **liaisons synchrones**, qui sont des liaisons à 4 fils : un fil de communication pour chaque sens (émettre, *transmit*, et recevoir, *receive*), un signal d'horloge pour la synchronisation, et un fil de masse. Ces liaisons sont gérées par les deux fonctions

  ```
  shiftOut(dataPin, clockPin, bitOrder, value)
  ```

  et

  ```
  shiftIn(dataPin, clockPin, bitOrder)
  ```

  La première fonction envoie un octet de données bit par bit sur la broche `dataPin`, en commençant soit par le bit de poids fort soit par le bit de poids faible, suivant que `bitOrder` est égal à `MSBFIRST` ou à `LSBFIRST`. Après avoir envoyé chaque bit sur la broche `dataPin`, une impulsion est produite sur la broche `clockPin`, conformément au protocole SPI. Les deux broches `dataPin` et `clockPin` doivent auparavant avoir été configuées en sortie. Pour envoyer un mot de 16 bits par ce moyen, il faudra envoyer successivement son octet de poids fort et son octet de poids faible, par exemple en faisant

  ```
  shiftOut(dataPin, clockPin, MSBFIRST, (data >> 8));   // shift out highbyte
  shiftOut(dataPin, clockPin, MSBFIRST, data);          // shift out lowbyte
  ```

  La seconde fonction reçoit un octet par les différents niveaux logiques présents sur la broche `dataPin` (configurée en entrée) après chaque front montant appliqué sur la broche `clockPin` (configurée en sortie).

- Les **liaisons asynchrones**, qui n'utilisent pas de signal d'horloge pour la synchronisation de l'émetteur et du récepteur. C'est pour ce type de communication qu'a été conçue la bibliothèque `Serial`, présentée au paragraphe 3.3. Elle permet de gérer les échanges réalisés avec les broches 0 et 1 du port d'entrées-sorties logiques. Comme ces broches sont reliées au port USB, ce qui nous a permis d'afficher des variables sur l'écran de l'ordinateur, il n'est pas possible de connecter sur ces deux broches à la fois l'ordinateur de développement et un autre microcontrôleur. Le circuit ARDUINO Mega, en plus de ses très nombreuses entrées-sorties logiques et entrées analogiques supplémentaires, possède en tout quatre liaisons de communication en série, qui peuvent toutes être gérées par les fonctions de la bibliothèque `Serial`. Si on souhaite réaliser avec un ARDUINO nano des transmissions série asynchrones sur d'autres broches que les broches 0 et 1, on peut utiliser les fonctions de la bibliothèque `SoftwareSerial`. Mais les transmissions réalisées de cette manière seront beaucoup moins performantes, notamment en termes de vitesse de transmission, non pas à cause de défauts des fonctions de cette bibliothèque, mais à cause de l'absence de circuits électroniques (appelés UART, *Universal Asynchronous Receiver and Transmitter*) disponibles uniquement pour les broches 0 et 1.

Notes

[1] Le site `http://www.arduino.cc` fournit un très grand nombre d'informations sur cet environnement. Une page destinée aux utilisateurs francophones est également disponible à l'adresse `http://www.arduino.cc/fr`. Des informations générales sont également disponibles à l'adresse `http://fr.wikipedia.org/wiki/Arduino` et des articles sur cet environnement sont disponibles en anglais à l'adresse `http://tronixstuff.wordpress.com` et en français aux adresses `http://www.framablog.org/index.php/post/2010/06/02/arduino-materiel-libre` et `http://www.pobot.org/-L-univers-Arduino-.html`. En France, les cartes ARDUINO sont distribués par les sociétés *Lextronic* (`http://www.lextronic.fr`), *Zartronic* (`http://www.zartronic.fr`), *Alevita Électronique* (`http://www.alevita.com`), et RS Composants (`http://radiospares-fr.rs-online.com/web`).