

[Arduino 7] L'affichage LCD, une autre manière d'interagir

Vous souhaitez rendre votre projet un peu plus autonome, en le disloquant de son attachement à votre ordinateur parce que vous voulez afficher du texte ? Eh bien grâce aux afficheurs LCD, cela va devenir possible ! Vous allez apprendre à utiliser ces afficheurs d'une certaine catégorie pour pouvoir réaliser vos projet les plus fous. Il est courant d'utiliser ces écrans permettant l'affichage du texte en domotique, robotique, voir même pour déboguer un programme ! Avec eux, vos projet n'aurons plus la même allure !

[Arduino 701] Les écrans LCD

Vous avez appris plus tôt comment interagir avec l'ordinateur, lui envoyer de l'information. Mais maintenant, vous voudrez sûrement pouvoir afficher de l'information sans avoir besoin d'un ordinateur. Avec les écrans LCD, nous allons pouvoir afficher du texte sur un écran qui n'est pas très coûteux et ainsi faire des projets sensationnels !

Un écran LCD c'est quoi ?

Mettons tout de suite au clair les termes : LCD signifie "Liquid Crystal Display" et se traduit, en français, par "Écran à Cristaux Liquides" (mais on a pas d'acronymes classe en français donc on parlera toujours de LCD). Ces écrans sont PARTOUT ! Vous en trouverez dans plein d'appareils électroniques disposant d'afficheur : les montres, le tableau de bord de votre voiture, les calculatrices, etc. Cette utilisation intensive est due à leur faible consommation et coût. Mais ce n'est pas tout ! En effet, les écrans LCD sont aussi sous des formes plus complexes telles que la plupart des écrans d'ordinateur ainsi que les téléviseurs à écran plat. Cette technologie est bien maîtrisée et donc le coût de production est assez bas. Dans les années à venir, ils vont avoir tendance à être remplacés par les écrans à affichage LED qui sont pour le moment trop chers.

J'en profite pour mettre l'alerte sur la différence des écrans à LED. Il en existe deux types :

- les écrans à rétro-éclairage LED : ceux sont des écrans LCD tout à fait ordinaires qui ont simplement la particularité d'avoir un rétro-éclairage à LED à la place des tubes néons. Leur prix est du même ordre de grandeur que les LCD "normaux". En revanche, la qualité d'affichage des couleurs semble meilleure comparés aux LCD "normaux".
- les écrans à affichage LED : ceux si ne disposent pas de rétro-éclairage et ne sont ni des écrans LCD, ni des plasma. Ce sont des écrans qui, en lieu et place des pixels, se trouvent des LED de très très petite taille. Leur coût est prohibitif pour le moment, mais la qualité de contraste et de couleur inégale tous les écrans existants !

Les deux catégories précédentes (écran LCD d'une montre par exemple et celui d'un moniteur d'ordinateur) peuvent être différenciées assez rapidement par une caractéristique simple : *la couleur*. En effet, les premiers sont monochromes (une seule couleur) tandis que les seconds sont colorés (rouge, vert et bleu). Dans cette partie, nous utiliserons uniquement le premier type pour des raisons de simplicité et de coût.

Fonctionnement de l'écran

N'étant pas un spécialiste de l'optique ni de l'électronique "bas-niveau" (jonction et tout le tralala) je ne vais pas vous faire un cours détaillé sur le "comment ça marche ?" mais plutôt aller à l'essentiel, vers le "pourquoi ça s'allume ?". Comme son nom l'indique, un écran LCD possède des cristaux liquides. Mais ce n'est pas tout ! En effet, pour fonctionner il faut plusieurs choses. Si vous regardez de très près votre écran (éteint pour pas vous bousiller les yeux) vous pouvez voir une grille de carré. Ces carrés sont appelés des pixels (de l'anglais "Picture Element", soit "Élément d'image" en français, encore une fois c'est moins classe). 😊 Chaque pixel est un cristal liquide. Lorsque aucun courant ne le traverse, ses molécules sont orientées dans un sens (admettons, 0°). En revanche lorsqu'un courant le traverse, ses molécules vont se tourner dans la même direction (90°). Voilà pour la base.

Mais pourquoi il y a de la lumière dans un cas et pas dans l'autre ?

Tout simplement parce que cette lumière est **polarisée**. Cela signifie que la lumière est orientée dans une direction (c'est un peu compliqué à démontrer, je vous demanderais donc de l'admettre). En effet, entre les cristaux liquides et la source lumineuse se trouve un filtre polariseur de lumière. Ce filtre va orienter la lumière dans une direction précise. Entre vos yeux et les cristaux se trouve un autre écran polariseur, qui est perpendiculaire au premier. Ainsi, il faut que les cristaux liquides soient dans la bonne direction pour que la lumière passe de bout en bout et revienne à vos yeux. Un schéma vaut souvent mieux qu'un long discours, je vous conseille donc de regarder celui sur la droite de l'explication pour mieux comprendre (source : Wikipédia). Enfin, vient le rétro-éclairage (fait avec des LED) qui vous permettra de lire l'écran même en pleine nuit (sinon il vous faudrait l'éclairer pour voir le contraste).



Si vous voulez plus d'informations sur les écrans LCD, j'invite votre curiosité à se diriger vers ce lien [Wikipédia](https://fr.wikipedia.org/wiki/Écran_à_cristaux_liquides) ou d'autres sources. 😊

Commande du LCD

Normalement, pour pouvoir afficher des caractères sur l'écran il nous faudrait activer individuellement chaque pixel de l'écran. Un caractère est représenté par un bloc de 7*5 pixels. Ce qui fait qu'un écran de 16 colonnes et 2 lignes représente un total de $16*2*7*5 = 1120$ pixels ! 😊 Heureusement pour nous, des ingénieurs sont passés par là et nous ont simplifié la tâche.

Le décodeur de caractères

Tout comme il existe un driver vidéo pour votre carte graphique d'ordinateur, il existe un driver "LCD" pour votre afficheur. Rassurez-vous, aucun composant ne s'ajoute à votre liste d'achat puisqu'il est intégré dans votre écran. Ce composant va servir à décoder un ensemble "simple" de bits pour afficher un caractère à une position précise ou exécuter des commandes comme déplacer le curseur par exemple. Ce composant est fabriqué principalement par *Hitachi* et se nomme le **HC44780**. Il sert de **décodeur de caractères**. Ainsi, plutôt que de devoir multiplier les signaux pour commander les pixels un à un, il nous suffira d'envoyer des octets de commandes pour lui dire "écris moi 'zéros' à partir de la colonne 3 sur la ligne 1". Ce composant possède 16 broches que je vais brièvement décrire :

N°	Nom	Rôle
1	VSS	Masse
2	Vdd	+5V
3	V0	Réglage du contraste
4	RS	Sélection du registre (commande ou donnée)
5	R/W	Lecture ou écriture
6	E	Entrée de validation
7 à 14	D0 à D7	Bits de données
15	A	Anode du rétroéclairage (+5V)
16	K	Cathode du rétroéclairage (masse)

Normalement, pour tous les écrans LCD (non graphiques) ce brochage est le même. Donc pas d'inquiétude lors des branchements, il vous suffira de vous rendre sur cette page pour consulter le tableau. 😊

Par la suite, les broches utiles qu'il faudra relier à l'Arduino sont les broches 4, 5 (facultatives), 6 et les données (7 à 14 pouvant être réduite à 8 à 14) en oubliant pas l'alimentation et la broche de réglage du contraste. Ce composant possède tout le système de traitement pour afficher les caractères. Il contient dans sa mémoire le schéma d'allumage des pixels pour afficher chacun d'entre eux. Voici la table des caractères affichables :

Higher 4bit Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000		0	a	P	`	F		-	9	3	o	p	
xxxx0001		!	1	A	Q	a	9	.	7	*	4	8	q
xxxx0010		"	2	B	R	b	r	「	イ	ウ	×	β	θ
xxxx0011		#	3	C	S	c	s	」	ウ	テ	E	ε	×
xxxx0100		\$	4	D	T	d	t	、	工	ト	μ	α	
xxxx0101		%	5	E	U	e	u	・	オ	★	1	ε	0
xxxx0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111		'	7	G	W	g	w	フ	十	又	う	q	π
xxxx1000		(8	H	X	h	x	イ	ウ	本	リ	フ	又
xxxx1001)	9	I	Y	i	y	ウ	テ	ル	リ	ウ	
xxxx1010		*	:	J	Z	j	z	エ	コ	ン	レ	i	チ
xxxx1011		+	:	K	L	k	l	オ	ウ	ロ	ロ	×	カ
xxxx1100		,	<	L	*	1	l	カ	ニ	フ	フ	φ	π
xxxx1101		-	=	M	I	m	>	ユ	ズ	ハ	コ	ト	÷
xxxx1110		.	>	N	^	n	+	ヨ	ロ	ホ	リ	カ	
xxxx1111		/	?	O	_	o	+	ウ	ウ	マ	"	δ	■

Quel écran choisir ?

Les caractéristiques

Texte ou Graphique ?

Dans la grande famille afficheur LCD, on distingue plusieurs catégories :

- Les afficheurs alphanumériques
- Les afficheurs graphiques monochromes
- Les afficheurs graphiques couleur

Les premiers sont les plus courants. Ils permettent d'afficher des lettres, des chiffres et quelques caractères spéciaux. Les caractères sont prédéfinis (voir table juste au-dessus) et on a donc aucunement besoin de gérer chaque pixel de l'écran. Les seconds sont déjà plus avancés. On a accès à chacun des pixels et on peut donc produire des dessins beaucoup plus évolués. Ils sont cependant légèrement plus onéreux que les premiers. Les derniers sont l'évolution des précédents, la couleur en plus (soit 3 fois plus de pixels à gérer : un sous-pixel pour le rouge, un autre pour le bleu et un dernier pour le vert, le tout forme la couleur d'un seul pixel). Pour le TP on se servira d'afficheur de la première catégorie car ils suffisent à faire de nombreux montages et restent accessibles pour des zéros. 😊

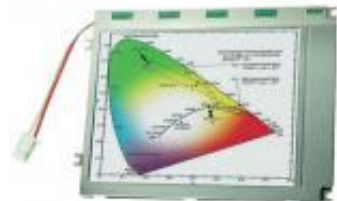
**Afficheur
alphanumérique**



**Afficheur graphique
(monochrome)**



**Afficheur graphique
(couleur)**



Ce n'est pas la taille qui compte !

Les afficheurs existent dans de nombreuses tailles. Pour les afficheurs de type textes, on retrouve le plus fréquemment le format 2 lignes par 16 colonnes. Il en existe cependant de nombreux autres avec une seule ligne, ou 4 (ou plus) et 8 colonnes, ou 16, ou 20 ou encore plus ! Libre à vous de choisir la taille qui vous plait le plus, sachant que le TP devrait s'adapter sans souci à toute taille d'écran (pour ma part ce sera un 2 lignes 16 colonnes) !

La couleur, c'est important

Nan je blague ! Prenez la couleur qui vous plait ! Vert, blanc, bleu, jaune, amusez-vous ! (moi c'est écriture blanche sur fond bleu, mais je rêve d'un afficheur à la matrix, noir avec des écritures vertes !)

Communication avec l'écran

La communication parallèle

De manière classique, on communique avec l'écran de manière **parallèle**. Cela signifie que l'on envoie des bits par blocs, en utilisant plusieurs broches en même temps (opposée à une transmission série où les bits sont envoyés un par un sur une seule

broche). Comme expliqué plus tôt dans ce chapitre, nous utilisons 10 broches différentes, 8 pour les données (en parallèle donc) et 2 pour de la commande (E : Enable et RS : Register Selector). La ligne R/W peut être connecté à la masse si l'on souhaite uniquement faire de l'écriture. Pour envoyer des données sur l'écran, c'est en fait assez simple. Il suffit de suivre un ordre logique et un certain timing pour que tout se passe bien. Tout d'abord, il nous faut placer la broche RS à 1 ou 0 selon que l'on veut envoyer une commande, comme par exemple "déplacer le curseur à la position (1;1)" ou que l'on veut envoyer une donnée : "écris le caractère 'a' ". Ensuite, on place sur les 8 broches de données (D0 à D7) la valeur de la donnée à afficher. Enfin, il suffit de faire une impulsion d'au moins 450 ns pour indiquer à l'écran que les données sont prêtes. C'est aussi simple que ça ! Cependant, comme les ingénieurs d'écrans sont conscients que la communication parallèle prend beaucoup de broches, ils ont inventé un autre mode que j'appellerai "semi-parallèle". Ce dernier se contente de travailler avec seulement les broches de données D4 à D7 (en plus de RS et E) et il faudra mettre les quatre autres (D0 à D3) à la masse. Il libère donc quatre broches. Dans ce mode, on fera donc deux fois le cycle "envoi des données puis impulsion sur E" pour envoyer un octet complet.

Ne vous inquiétez pas à l'idée de tout cela. Pour la suite du chapitre nous utiliserons une librairie nommée **LiquidCrystal** qui se chargera de gérer les timings et l'ensemble du protocole.

Pour continuer ce chapitre, le mode "semi-parallèle" sera choisi. Il nous permettra de garder plus de broches disponibles pour de futurs montages et est souvent câblé par défaut dans de nombreux shields (dont le mien). La partie suivante vous montrera ce type de branchement. Et pas de panique, je vous indiquerai également la modification à faire pour connecter un écran en mode "parallèle complet".

La communication série

Lorsque l'on ne possède que très peu de broches disponibles sur notre Arduino, il peut être intéressant de faire appel à un composant permettant de communiquer par voie série avec l'écran. Un tel composant se chargera de faire la conversion entre les données envoyées sur la voie série et ce qu'il faut afficher sur l'écran. Le gros avantage de cette solution est qu'elle nécessite seulement un seul fil de donnée (avec une masse et le VCC) pour fonctionner là où les autres méthodes ont besoin de presque une dizaine de broches. Toujours dans le cadre du prochain TP, nous resterons dans le classique en utilisant une connexion parallèle. En effet, elle nous permet de garder l'approche "standard" de l'écran et nous permet de garder la liaison série pour autre chose (encore que l'on pourrait en émuler une sans trop de difficulté). Ce composant de conversion "Série -> parallèle" peut-être réalisé simplement avec un 74h595 😊 (je vous laisse coder le driver comme exercice si vous voulez 😊)

Et par liaison I²C

Un dernier point à voir, c'est la communication de la carte Arduino vers l'écran par la liaison I²C. Cette liaison est utilisable avec seulement 2 broches (une broche de donnée et une broche d'horloge) et nécessite l'utilisation de deux broches analogiques de l'Arduino (broche 4 et 5).

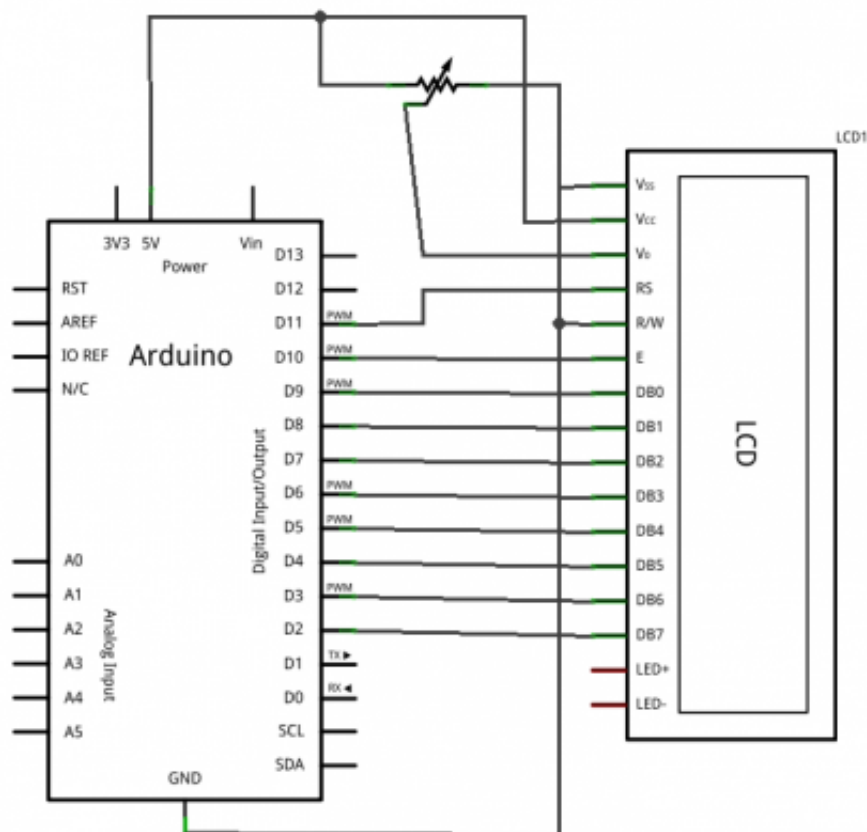
Comment on s'en sert ?

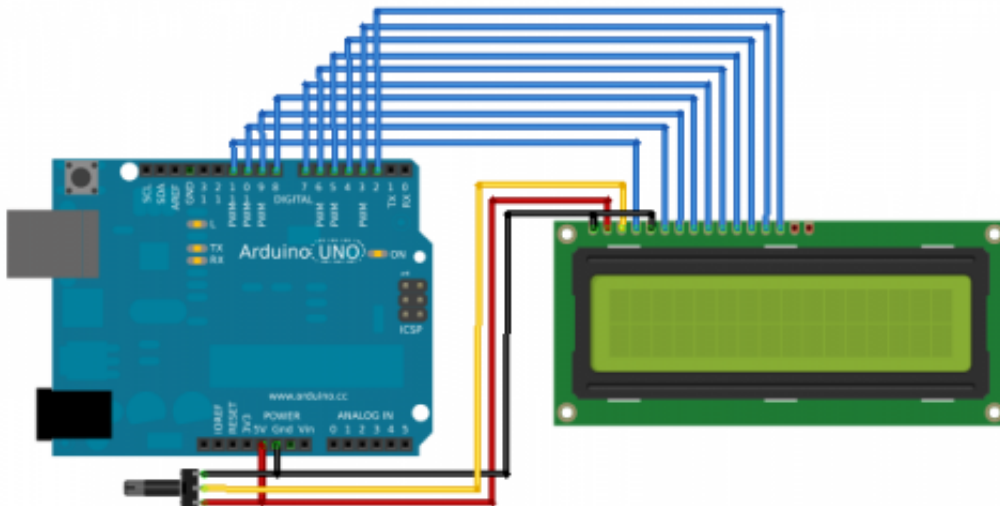
Comme expliqué précédemment, je vous propose de travailler avec un écran dont seulement quatre broches de données sont utilisées. Pour le bien de tous je vais présenter ici les deux montages, mais ne soyez pas surpris si dans les autres montages ou les vidéos vous voyez seulement un des deux. 😊

Le branchement

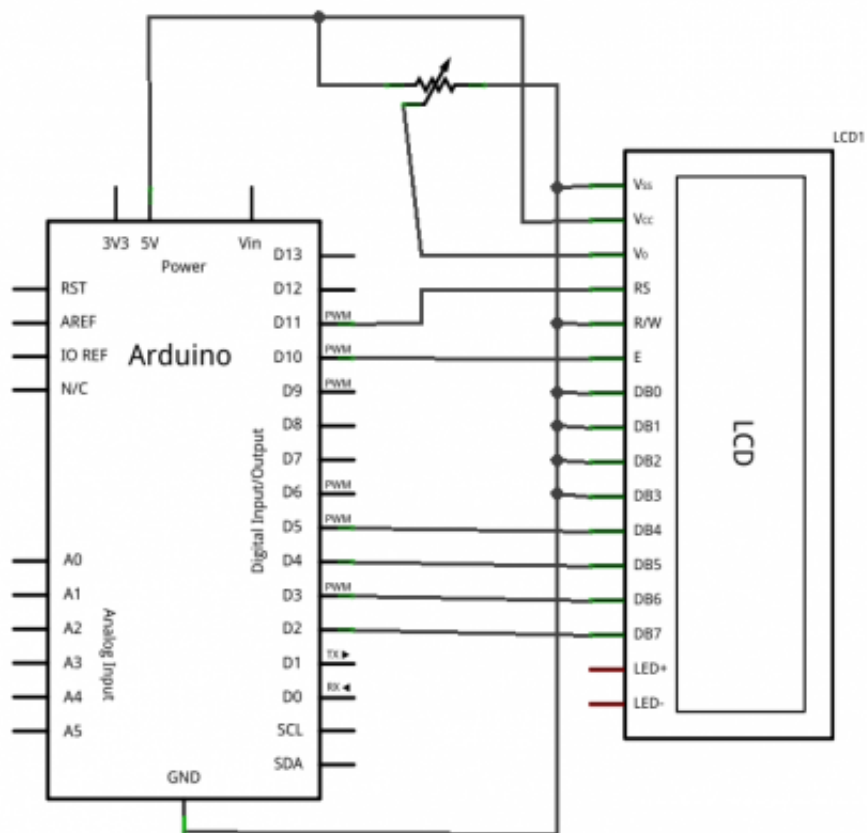
L'afficheur LCD utilise 6 à 10 broches de données ((D0 à D7) ou (D4 à D7) + RS + E) et deux d'alimentations (+5V et masse). La plupart des écrans possèdent aussi une entrée analogique pour régler le contraste des caractères. Nous brancherons dessus un potentiomètre de 10 kOhms. Les 10 broches de données peuvent être placées sur n'importe quelles entrées/sorties numériques de l'Arduino. En effet, nous indiquerons ensuite à la librairie LiquidCrystal qui est branché où.

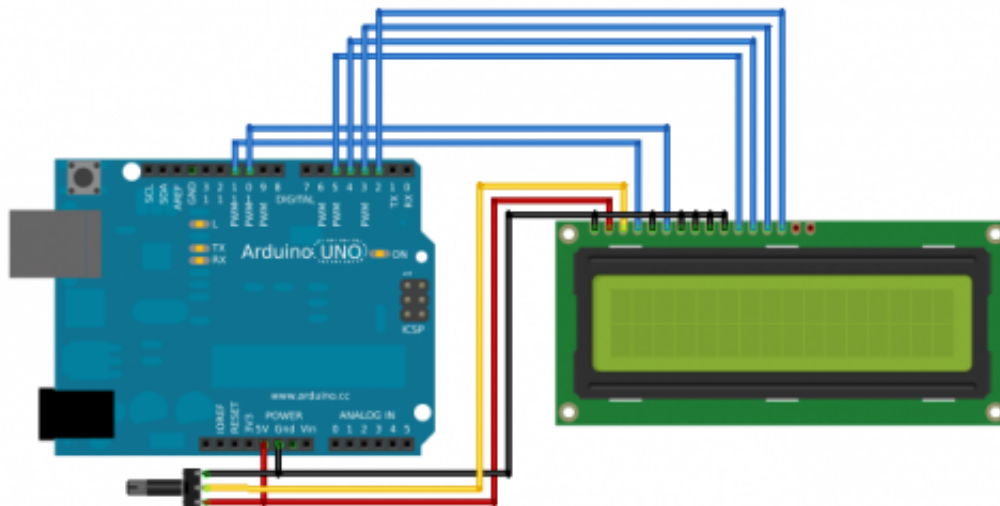
Le montage à 8 broches de données





Le montage à 4 broches de données





Le démarrage de l'écran avec Arduino

Comme écrit plus tôt, nous allons utiliser la librairie "LiquidCrystal". Pour l'intégrer c'est très simple, il suffit de cliquer sur le menu "Import Library" et d'aller chercher la bonne. Une ligne `#include "LiquidCrystal.h"` doit apparaître en haut de la page de code (les prochaines fois vous pourrez aussi taper cette ligne à la main directement, ça aura le même effet). Ensuite, il ne nous reste plus qu'à dire à notre carte Arduino où est branché l'écran (sur quelles broches) et quelle est la taille de ce dernier (nombre de lignes et de colonnes). Nous allons donc commencer par déclarer un objet (c'est en fait une variable évoluée, plus de détails dans la prochaine partie) `lcd`, de type `LiquidCrystal` et qui sera global à notre projet. La déclaration de cette variable possède plusieurs formes ([lien vers la doc.](#)) :

- `LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)` où `rs` est le numéro de la broche où est branché "RS", "enable" est la broche "E" et ainsi de suite pour les données.
- `LiquidCrystal(rs, enable, d4, d5, d6, d7)` (même commentaires que précédemment

Ensuite, dans le `setup()` il nous faut démarrer l'écran en spécifiant son nombre de **colonnes** puis de **lignes**. Cela se fait grâce à la fonction `begin(cols,rows)`. Voici un exemple complet de code correspondant aux deux branchements précédents (commentez la ligne qui ne vous concerne pas) :

```
1 #include "LiquidCrystal.h" //ajout de la librairie
2
3 //Vérifier les broches !
4 LiquidCrystal lcd(11,10,9,8,7,6,5,4,3,2); //liaison 8 bits de données
5 LiquidCrystal lcd(11,10,5,4,3,2); //liaison 4 bits de données
6
7 void setup()
8 {
9     lcd.begin(16,2); //utilisation d'un écran 16 colonnes et 2 lignes
10    lcd.write("Salut les Zeros !"); //petit test pour vérifier que tout marche
11 }
12
13 void loop() {}
```

Surtout ne mettez **pas d'accents** ! L'afficheur ne les accepte pas par défaut et

affichera du grand n'importe quoi à la place.

Vous remarquez que j'ai rajouté une ligne dont je n'ai pas parlé encore. Je l'ai juste mise pour vérifier que tout fonctionne bien avec votre écran, nous reviendrons dessus plus tard. Si tout se passe bien, vous devriez obtenir l'écran suivant :



Si jamais rien ne s'affiche, essayez de tourner votre potentiomètre de contraste. Si cela ne marche toujours pas, vérifiez les bonnes attributions des broches (surtout si vous utilisez un shield).

Maintenant que nous maîtrisons les subtilités concernant l'écran, nous allons pouvoir commencer à jouer avec... En avant !

[Arduino 702] Votre premier texte sur le LCD !

Ça y est, on va pouvoir commencer à apprendre des trucs avec notre écran LCD ! Alors, au programme : afficher des variables, des tableaux, déplacer le curseur, etc. Après toutes ces explications, vous serez devenu un pro du LCD, du moins du LCD alphanumérique 😊. Aller, en route ! Après ça vous ferez un petit TP plutôt intéressant, notamment au niveau de l'utilisation pour l'affichage des mesures sans avoir besoin d'un ordinateur. De plus, pensez au fait que vous pouvez vous aider des afficheurs pour déboguer votre programme !

Ecrire du texte sur le LCD

Afficher du texte

Vous vous rappelez comme je vous disais il y a longtemps "Les développeurs Arduino sont des gens sympas, ils font les choses clairement et logiquement !" ? Eh bien ce constat se reproduit (encore) pour la bibliothèque LiquidCrystal ! En effet, une fois que votre écran LCD est bien paramétré, il nous suffira d'utiliser qu'une seule fonction pour afficher du texte ! Allez je vous laisse 10 secondes pour deviner le nom de la fonction que nous allons utiliser. Un indice, ça a un lien avec la voie série... C'est trouvé ? Félicitations à tous ceux qui auraient dit `print()`. En effet, une fois de plus nous retrouvons une fonction `print()`, comme pour l'objet `Serial`, pour envoyer du texte. Ainsi, pour saluer tous les zéros de la terre nous aurons juste à écrire :

```
1 lcd.print("Salut les Zer0s!");
```

et pour code complet avec les déclarations on obtient :

```
1 #include //on inclut la librairie
2
3 // initialise l'écran avec les bonnes broches
4
5 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
6
7 LiquidCrystal lcd(8,9,4,5,6,7);
8
9 void setup() {
10     lcd.begin(16, 2);
11     lcd.print("Salut les Zer0s!");
12 }
13
14 void loop() {
15
16 }
```

Mais c'est nul ton truc on affiche toujours au même endroit, en haut à gauche !

Oui je sais, mais chaque chose en son temps, on s'occupera du positionnement du texte bientôt, promis !

Afficher une variable

Afficher du texte c'est bien, mais afficher du contenu dynamique c'est mieux ! Nous allons maintenant voir comment afficher une variable sur l'écran. Là encore, rien de difficile. Je ne vais donc pas faire un long discours pour vous dire qu'il n'y a qu'une seule fonction à retenir... le suspense est terrible... OUI évidemment cette fonction c'est **`print()`** ! Décidément elle est vraiment tout-terrain (et rédacteur du tutoriel Arduino devient un vrai boulot de feignant, je vais finir par me copier-coller à chaque fois !) Allez zou, un petit code, une petite photo et en avant Guingamp !

```
1 int mavariable = 42;
2 lcd.print(mavariable);
```

Combo ! Afficher du texte ET une variable

Bon vous aurez remarqué que notre code possède une certaine faiblesse... On n'affiche au choix un texte ou un nombre, mais pas les deux en même temps ! Nous allons donc voir maintenant une manière d'y remédier.

La fonction solution

La solution se trouve dans les bases du langage C , grâce à une fonction qui s'appelle **sprintf()** (aussi appelé "string printf"). Les personnes qui ont fait du C doivent la connaître, ou connaître sa cousine "printf". Cette fonction est un peu particulière car elle ne prend pas un nombre d'argument fini. En effet, si vous voulez afficher 2 variables vous ne lui donnerez pas autant d'arguments que pour en afficher 4 (ce qui paraît logique d'une certaine manière). Pour utiliser cette dernière, il va falloir utiliser un tableau de char qui nous servira de *buffer*. Ce tableau sera celui dans lequel nous allons écrire notre chaîne de caractère. Une fois que nous aurons écrit dedans, il nous suffira de l'envoyer sur l'écran en utilisant... `printf()` !

Son fonctionnement

Comme dit rapidement plus tôt, `sprintf()` n'a pas un nombre d'arguments fini. Cependant, elle en aura au minimum deux qui sont le tableau de la chaîne de caractère et une chaîne à écrire. Un exemple simple serait d'écrire :

```
1 char message[16] = "";
2 sprintf(message, "J'ai 42 ans");
```

Au début, le tableau `message` ne contient rien. Après la fonction `sprintf()`, il possédera le texte "J'ai 42 ans". Simple non ?

J'utilise un tableau de 16 cases car mon écran fait 16 caractères de large au maximum, et donc inutile de gaspiller de la mémoire en prenant un tableau plus grand que nécessaire.

Nous allons maintenant voir comment changer mon âge en le mettant en dynamique dans la chaîne grâce à une variable. Pour cela, nous allons utiliser des **marqueurs de format**. Le plus connu est `%d` pour indiquer un nombre entier (nous verrons les autres ensuite). Dans le contenu à écrire (le deuxième argument), nous placerons ces marqueurs à chaque endroit où l'on voudra mettre une variable. Nous pouvons en placer autant que nous voulons. Ensuite, il nous suffira de mettre dans le même ordre que les marqueurs les différentes variables en argument de `sprintf()`. Tout va être plus clair avec un exemple !

```
1 char message[16] = "";
2 int nbA = 3;
3 int nbB = 5;
4 sprintf(message, "%d + %d = %d", nbA, nbB, nbA+nbB);
```

Cela affichera :

```
1 3 + 5 = 8
```

Les marqueurs

Comme je vous le disais, il existe plusieurs marqueurs. Je vais vous présenter ceux qui vous serviront le plus, et différentes astuces pour les utiliser à bon escient :

- **%d** qui sera remplacé par un int (signé)
- **%s** sera remplacé par une chaîne (un tableau de char)
- **%u** pour un entier non signé (similaire à %d)
- **%%** pour afficher le symbole '%' 😊

Malheureusement, Arduino ne les supporte pas tous. En effet, le %f des float ne fonctionne pas. 😞 Il vous faudra donc bricoler si vous désirez l'afficher en entier (je vous laisse deviner comment). Si jamais vous désirez forcer l'affichage d'un marqueur sur un certain nombre de caractères, vous pouvez utiliser un indicateur de taille de ce nombre entre le '%' et la lettre du marqueur. Par exemple, utiliser "%3d" forcera l'affichage du nombre en paramètre (quel qu'il soit) **sur trois caractères au minimum**. La variable ne sera pas tronquée s'il est plus grande que l'emplacement prévu. Ce paramètre prendra donc toujours autant de place *au minimum* sur l'écran (utile pour maîtriser la disposition des caractères). Exemple :

```
1 int age1 = 42;
2 int age2 = 5;
3 char prenom1[10] = "Ben";
4 char prenom2[10] = "Luc";
5 char message[16] = "";
6 sprintf(message, "%s:%2d,%s:%2d", prenom1, age1, prenom2, age2);
```

À l'écran, on aura un texte tel que :

```
1 Ben:42, Luc: 5
```

On note l'espace avant le 5 grâce au forçage de l'écriture de la variable sur 2 caractères induit par %2d.

Exercice, faire une horloge

Consigne

Afin de conclure cette partie, je vous propose un petit exercice. Comme le titre l'indique, je vous propose de réaliser une petite horloge. Bien entendu elle ne sera pas fiable du tout car nous n'avons aucun repère réel dans le temps, mais ça reste un bon exercice. L'objectif sera donc d'afficher le message suivant : "Il est hh:mm:ss" avec 'hh' pour les heures, 'mm' pour les minutes et 'ss' pour les secondes. Ça vous ira ? Ouais, enfin je vois pas pourquoi je pose la question puisque de toute manière vous n'avez pas le choix ! 😡 Une dernière chose avant de commencer. Si vous tentez de faire plusieurs affichages successifs, le curseur ne se replacera pas et votre écriture sera vite chaotique. Je vous donne donc rapidement une fonction qui vous permet de revenir à la position en haut à gauche de l'écran : home(). Il vous suffira de faire un lcd.home() pour replacer le curseur en haut à gauche. Nous reparlerons de la position curseur dans le chapitre suivant !

Solution

Je vais directement vous parachuter le code, sans vraiment d'explications car je pense l'avoir suffisamment commenté (et entre nous l'exercice est sympa et pas trop dur). 😊

```
1 #include //on inclut la librairie
2
3 // initialise l'écran avec les bonnes broches
4 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
5 LiquidCrystal lcd(8,9,4,5,6,7);
6
7 int heures,minutes,secondes;
8 char message[16] = "";
9
10 void setup()
11 {
12     lcd.begin(16, 2); // règle la taille du LCD : 16 colonnes et 2 lignes
13
14     //changer les valeurs pour démarrer à l'heure souhaitée !
15     heures = 0;
16     minutes = 0;
17     secondes = 0;
18 }
19
20 void loop()
21 {
22     //on commence par gérer le temps qui passe...
23     if(secondes == 60) //une minutes est atteinte ?
24     {
25         secondes = 0; //on recompte à partir de 0
26         minutes++;
27     }
28     if(minutes == 60) //une heure est atteinte ?
29     {
30         minutes = 0;
31         heures++;
32     }
33     if(heures == 24) //une journée est atteinte ?
34     {
35         heures = 0;
36     }
37
38     //met le message dans la chaine à transmettre
39     sprintf(message, "Il est %2d:%2d:%2d", heures, minutes, secondes);
40
41     lcd.home(); //met le curseur en position (0;0) sur l'écran
42
43     lcd.write(message); //envoi le message sur l'écran
44
45     delay(1000); //attend une seconde
46     //une seconde s'écoule...
47     secondes++;
48 }
```

Se déplacer sur l'écran

Bon, autant vous prévenir d'avance, ce morceau de chapitre ne sera pas digne du nom de "tutoriel". Malheureusement, pour se déplacer sur l'écran (que ce soit le curseur ou du texte) il n'y a pas 36 solutions, juste quelques appels relativement simples à des

fonctions. Désolé d'avance pour le "pseudo-listing" de fonctions que je vais faire tout en essayant de le garder intéressant...

Gérer l'affichage

Les premières fonctions que nous allons voir concernent l'écran dans son ensemble. Nous allons apprendre à enlever le texte de l'écran mais le garder dans la mémoire pour le ré-afficher ensuite. En d'autres termes, vous allez pouvoir faire un mode "invisible" où le texte est bien stocké en mémoire mais pas affiché sur l'écran. Les deux fonctions permettant ce genre d'action sont les suivantes :

- `noDisplay()` : fait disparaître le texte
- `display()` : fait apparaître le texte (s'il y en a évidemment)

Si vous tapez le code suivant, vous verrez le texte clignoter toutes les secondes :

```
1 #include //on inclut la librairie
2
3 // initialise l'écran avec les bonnes broches
4 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
5 LiquidCrystal lcd(8,9,4,5,6,7);
6
7 void setup() {
8     // règle la taille du LCD
9     lcd.begin(16, 2);
10    lcd.print("Hello World !");
11 }
12
13 void loop() {
14    lcd.noDisplay();
15    delay(500);
16    lcd.display();
17    delay(500);
18 }
```

Utile si vous voulez attirer l'attention de l'utilisateur ! Une autre fonction utile est celle vous permettant de nettoyer l'écran. Contrairement à la précédente, cette fonction va supprimer le texte de manière permanente. Pour le ré-afficher il faudra le renvoyer à l'afficheur. Cette fonction au nom évident est : `clear()`. Le code suivant vous permettra ainsi d'afficher un texte puis, au bout de 2 secondes, il disparaîtra (pas de `loop()`, pas nécessaire) :

```
1 #include //on inclut la librairie
2
3 // initialise l'écran avec les bonnes broches
4 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
5 LiquidCrystal lcd(8,9,4,5,6,7);
6
7 void setup() {
8     // règle la taille du LCD
9     lcd.begin(16, 2);
10    lcd.print("Hello World !");
11    delay(2000);
12    lcd.clear();
13 }
```

Cette fonction est très utile lorsque l'on fait des menus sur l'écran, pour pouvoir changer de page. Si on ne fait pas un `clear()`, il risque d'ailleurs de subsister des caractères de la page précédente. Ce n'est pas très joli.

Attention à ne pas appeler cette fonction plusieurs fois de suite, par exemple en la mettant dans la fonction `loop()`, vous verrez le texte ne s'affichera que très rapidement puis disparaîtra et ainsi de suite.

Gérer le curseur

Se déplacer sur l'écran

Voici maintenant d'autres fonctions que vous attendez certainement, celles permettant de déplacer le curseur sur l'écran. En déplaçant le curseur, vous pourrez écrire à n'importe quel endroit sur l'écran (attention cependant à ce qu'il y ait suffisamment de place pour votre texte). 😊 Nous allons commencer par quelque chose de facile que nous avons vu très rapidement dans le chapitre précédent. Je parle bien sûr de la fonction `home()` ! Souvenez-vous, cette fonction permet de replacer le curseur au début de l'écran.

Mais au fait, savez-vous comment est organisé le repère de l'écran ?

C'est assez simple, mais il faut être vigilant quand même. Tout d'abord, sachez que les coordonnées s'expriment de la manière suivante (x, y) . x représente les abscisses, donc les pixels horizontaux et y les ordonnées, les pixels verticaux. L'origine du repère sera logiquement le pixel le plus en haut à gauche (comme la lecture classique d'un livre, on commence en haut à gauche) et à pour coordonnées ... $(0,0)$! Eh oui, on ne commence pas aux pixels $(1,1)$ mais bien $(0,0)$. Quand on y réfléchit, c'est assez logique. Les caractères sont rangés dans des chaînes de caractères, donc des tableaux, qui eux sont adressés à partir de la case 0. Il paraît donc au final logique que les développeurs aient gardé une cohérence entre les deux. Puisque nous commençons à 0, un écran de 16×2 caractères pourra donc avoir comme coordonnées de 0 à 15 pour x et 0 ou 1 pour y . Ceci étant dit, nous pouvons passer à la suite. La prochaine fonction que nous allons voir prend directement en compte ce que je viens de vous dire. Cette fonction nommée `setCursor()` vous permet de positionner le curseur sur l'écran. On pourra donc faire `setCursor(0,0)` pour se placer en haut à gauche (équivalent à la fonction "home()") et en faisant `setCursor(15,1)` on se placera tout en bas à droite (toujours pour un écran de 16×2 caractères). Un exemple :

```
1 #include
2
3 // initialise l'écran avec les bonnes broches
4 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
5 LiquidCrystal lcd(8,9,4,5,6,7);
6
7 void setup()
8 {
9     lcd.begin(16, 2);
10    lcd.setCursor(2,1);           //place le curseur aux coordonnées (2,1)
11    lcd.print("Texte centré");    //texte centré sur la ligne 2
12 }
```

Animer le curseur

Tout comme nous pouvons faire disparaître le texte, nous pouvons aussi faire disparaître le curseur (comportement par défaut). La fonction `noCursor()` va donc l'effacer. La fonction antagoniste `cursor()` de son côté permettra de l'afficher (vous verrez alors un petit trait en bas du carré (5*8 pixels) où il est placé, comme lorsque vous appuyez sur la touche Insér. de votre clavier). Une dernière chose sympa à faire avec le curseur est de le faire clignoter. En anglais clignoter se dit "blink" et donc tout logiquement la fonction à appeler pour activer le clignotement est `blink()`. Vous verrez alors le curseur remplir le carré concerné en blanc puis s'effacer (juste le trait) et revenir. S'il y a un caractère en dessous, vous verrez alternativement un carré tout blanc puis le caractère. Pour désactiver le clignotement il suffit de faire appel à la fonction `noBlink()`.

```
1 #include
2
3 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
4 LiquidCrystal lcd(8,9,4,5,6,7);
5
6 void setup()
7 {
8     lcd.begin(16, 2);
9     lcd.home();           //place le curseur aux coordonnées (0,0)
10    lcd.setCursor();       //affiche le curseur
11    lcd.blink();           //et le fait clignoter
12    lcd.print("Curseur clignotant"); //texte centré sur la ligne 2
13 }
```

Si vous faites appel à `blink()` puis à `noCursor()` le carré blanc continuera de clignoter. En revanche, quand le curseur est dans sa phase "éteinte" vous ne verrez plus le trait du bas.

Jouer avec le texte

Nous allons maintenant nous amuser avec le texte. Ne vous attendez pas non plus à des miracles, il s'agira juste de déplacer le texte automatiquement ou non.

Déplacer le texte à la main

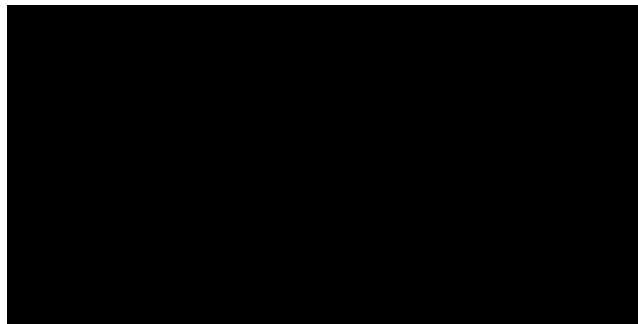
Pour commencer, nous allons déplacer le texte manuellement, vers la droite ou vers la gauche. N'essayez pas de produire l'expérience avec votre main, ce n'est pas un écran tactile, hein ! 😊 Le comportement est simple à comprendre. Après avoir écrit du texte sur l'écran, on peut faire appel aux fonctions `scrollDisplayRight()` et `scrollDisplayLeft()` vous pourrez déplacer le texte d'un carré vers la droite ou vers la gauche. S'il y a du texte sur chacune des lignes avant de faire appel aux fonctions, c'est le texte de chaque ligne qui sera déplacé par la fonction. Utilisez deux petits boutons poussoirs pour utiliser le code suivant. Vous pourrez déplacer le texte en appuyant sur chacun des poussoirs !

```
1 #include //on inclut la librairie
2
3 //les branchements
4 const int boutonGauche = 2; //le bouton de gauche
```

```

5  const int boutonDroite = 3; //le bouton de droite
6
7  // initialise l'écran avec les bonnes broches
8  // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
9  LiquidCrystal lcd(8,9,4,5,6,7);
10
11  //-----
12
13  void setup() {
14      //règlage des entrées/sorties
15      pinMode(boutonGauche, INPUT);
16      pinMode(boutonDroite, INPUT);
17
18      //on attache des fonctions aux deux interruptions externes (les boutons)
19      attachInterrupt(0, aDroite, RISING);
20      attachInterrupt(1, aGauche, RISING);
21
22      //paramétrage du LCD
23      lcd.begin(16, 2); // règle la taille du LCD
24      lcd.print("Hello les Zeros !");
25  }
26
27  void loop() {
28      //pas besoin de loop pour le moment
29  }
30
31  //fonction appelée par l'interruption du premier bouton
32  void aGauche() {
33      lcd.scrollDisplayLeft(); //on va à gauche !
34  }
35
36  //fonction appelé par l'interruption du deuxième bouton
37  void aDroite() {
38      lcd.scrollDisplayRight(); //on va à droite !
39  }

```



Déplacer le texte automatiquement

De temps en temps, il peut être utile d'écrire toujours sur le même pixel et de faire en sorte que le texte se décale tout seul (pour faire des effets zolis par exemple). 🇵🇷 Un couple de fonctions va nous aider dans cette tâche. La première sert à définir la direction du défilement. Elle s'appelle `leftToRight()` pour aller de la gauche vers la droite et `rightToLeft()` pour l'autre sens. Ensuite, il suffit d'activer (ou pas si vous voulez arrêter l'effet) avec la fonction `autoScroll()` (et `noAutoScroll()` pour l'arrêter). Pour mieux voir cet effet, je vous propose d'essayer le code qui suit. Vous verrez ainsi les chiffres de 0 à 9 apparaître et se "pousser" les uns après les autres :

```

1  #include //on inclut la librairie
2

```

```

3 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
4 LiquidCrystal lcd(8,9,4,5,6,7);
5
6 void setup()
7 {
8     lcd.begin(16, 2);
9     lcd.setCursor(14,0);
10    lcd.leftToRight(); //indique que le texte doit être déplacé vers la gauche
11    lcd.autoscroll(); //rend automatique ce déplacement
12    lcd.print("{");
13    int i=0;
14    for(i=0; i<10; i++)
15    {
16        lcd.print(i);
17        delay(1000);
18    }
19    lcd.print("}");
20 }

```

Créer un caractère

Dernière partie avant la pratique, on s'accroche vous serez bientôt incollable sur les écrans LCD ! En plus réjouissez-vous je vous ai gardé un petit truc sympa pour la fin. En effet, dans ce dernier morceau toute votre âme créatrice va pouvoir s'exprimer ! Nous allons créer des caractères !

Principe de la création

Créer un caractère n'est pas très difficile, il suffit d'avoir un peu d'imagination. Sur l'écran les pixels sont en réalité divisés en grille de 5×8 (5 en largeur et 8 en hauteur). C'est parce que le contrôleur de l'écran connaît l'alphabet qu'il peut dessiner sur ces petites grilles les caractères et les chiffres. Comme je viens de le dire, les caractères sont une grille de 5×8. Cette grille sera symbolisée en mémoire par un tableau de huit octets (type byte). Les 5 bits de poids faible de chaque octet représenteront une ligne du nouveau caractère. Pour faire simple, prenons un exemple. Nous allons dessiner un smiley, avec ses deux yeux et sa bouche pour avoir le rendu suivant :

```

0 0 0 0 0
X 0 0 0 X
0 0 0 0 0
0 0 0 0 0
X 0 0 0 X
0 X X X 0
0 0 0 0 0
0 0 0 0 0

```

Ce dessin se traduira en mémoire par un tableau d'octet que l'on pourra coder de la manière suivante :

```

1 byte smiley[8] = {
2     B000000,
3     B10001,
4     B000000,
5     B000000,

```

```

6      B10001,
7      B01110,
8      B00000,
9      B00000
10 };

```

La lettre 'B' avant l'écriture des octets veut dire "Je t'écris la valeur en binaire". Cela nous permet d'avoir un rendu plus facile et rapide.



L'envoyer à l'écran et l'utiliser

Une fois que votre caractère est créé, il faut l'envoyer à l'écran, pour que ce dernier puisse le connaître, avant toute communication avec l'écran (oui oui avant le begin()). La fonction pour apprendre notre caractère à l'écran se nomme createChar() signifiant "créer caractère". Cette fonction prend deux paramètres : "l'adresse" du caractère dans la mémoire de l'écran (de 0 à 7) et le tableau de byte représentant le caractère. Ensuite, l'étape de départ de communication avec l'écran peut-être faite (le begin). Ensuite, si vous voulez écrire ce nouveau caractère sur votre bel écran, nous allons utiliser une nouvelle (la dernière fonction) qui s'appelle write(). En paramètre sera passé un int représentant le numéro (adresse) du caractère que l'on veut afficher. Cependant, il y a là une faille dans le code Arduino. En effet, la fonction write() existe aussi dans une librairie standard d'Arduino et prend un pointeur sur un char. Le seul moyen de les différencier pour le compilateur sera donc de regarder le paramètre de la fonction pour savoir ce que vous voulez faire. Dans notre cas, il faut passer un int. On va donc forcer (on dit "caster") le paramètre dans le type "uint8_t" en écrivant la fonction de la manière suivante : write(uint8_t param). Le code complet sera ainsi le suivant :

```

1  #include  //on inclut la librairie
2
3  // initialise l'écran avec les bonnes broches
4  // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
5  LiquidCrystal lcd(8,9,4,5,6,7);
6
7  //notre nouveau caractère
8  byte smiley[8] = {
9      B00000,
10     B10001,
11     B00000,

```



```

12     B00000,
13     B10001,
14     B01110,
15     B00000,
16 };
17
18 void setup()
19 {
20     lcd.createChar(0, smiley); //apprend le caractère à l'écran LCD
21     lcd.begin(16, 2);
22     lcd.write((uint8_t) 0); //affiche le caractère de l'adresse 0
23 }

```

Désormais, vous savez l'essentiel sur les LCD alphanumériques, vous êtes donc aptes pour passer au TP. 😊

[Arduino 703] [TP] Supervision avec un LCD

Cher(e)s lecteur(e)s, savez-vous qu'il est toujours aussi difficile de faire une introduction et une conclusion pour chaque chapitre ? C'est pourquoi je n'ai choisi ici que de dire ceci : **amusez-vous bien avec les LCD !** 😊

Consigne

Dans ce TP, on se propose de mettre en place un système de supervision, comme on pourrait en retrouver dans un milieu industriel (en plus simple ici bien sur) ou dans d'autres applications. Le but sera d'afficher des informations sur l'écran LCD en fonction d'évènements qui se passent dans le milieu extérieur. Ce monde extérieur sera représenté par les composants suivants :

- Deux boutons, qui pourraient représenter par exemple deux barrières infrarouges donc le signal passe de 1 à 0 lorsque un objet passe devant.
- Deux potentiomètres. Un sert de "consigne" et est réglé par l'utilisateur. L'autre représentera un capteur (mais comme vous n'avez pas forcément lu la partie sur les capteurs (et qu'elle n'est pas rédigée à l'heure de la validation de cette partie), ce capteur sera représenté par un autre potentiomètre). A titre d'exemple, sur la vidéo à la suite vous verrez un potentiomètre rotatif qui représentera la consigne et un autre sous forme de glissière qui sera le capteur.
- Enfin, une LED rouge nous permettra de faire une alarme visuelle. Elle sera normalement éteinte mais si la valeur du capteur dépasse celle de la consigne alors elle s'allumera.

Comportement de l'écran

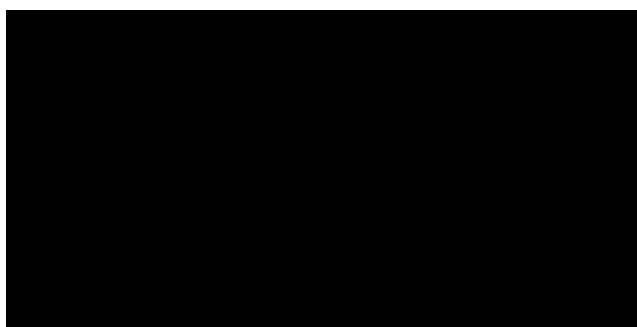
L'écran que j'utilise ne propose que 2 lignes et 16 colonnes. Il n'est donc pas possible d'afficher toute les informations de manière lisible en même temps. On se propose donc de faire un affichage alterné entre deux interface. Chaque interface sera affiché pendant cinq secondes à tour de rôle. La première affichera l'état des boutons. On pourra par exemple lire :

1	Bouton G : ON
2	Bouton D : OFF

La seconde interface affichera la valeur de la consigne et celle du capteur. On aura par exemple :

1	Consigne : 287
2	Capteur : 115

(Sur la vidéo vous verrez "gauche / droite" pour symboliser les deux potentiomètres, chacun fait comme il veut). 😊 Enfin, bien que l'information "consigne/capteur" ne s'affiche que toutes les 5 secondes, l'alarme (la LED rouge), elle, doit-être visible à tout moment si la valeur du capteur dépasse celle de la consigne. En effet, imaginez que cette alarme représentera une pression trop élevée, ce serait dommage que tout explose à cause d'un affichage 5 secondes sur 10 ! 😊 Je pense avoir fait le tour de mes attentes ! Je vous souhaite un bon courage, prenez votre temps, faites un beau schéma/montage/code et à bientôt pour la correction !



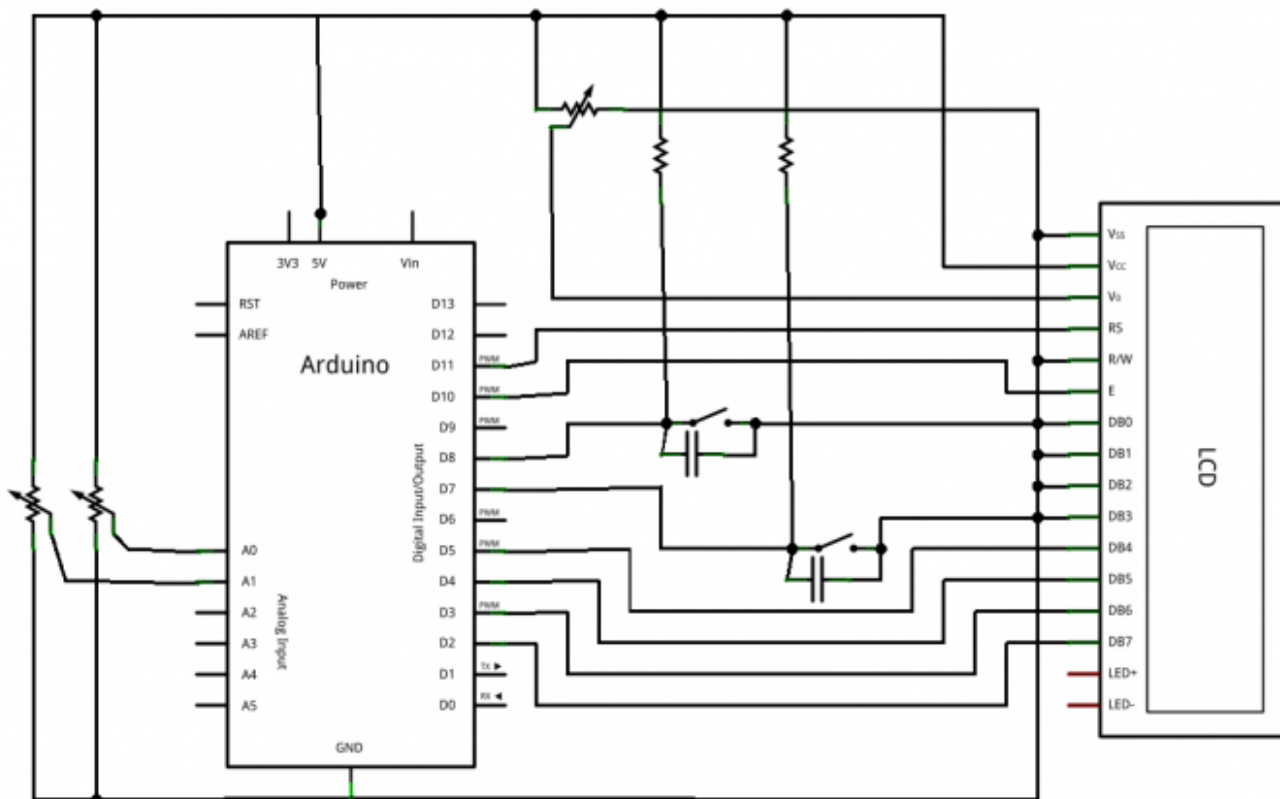
Correction !

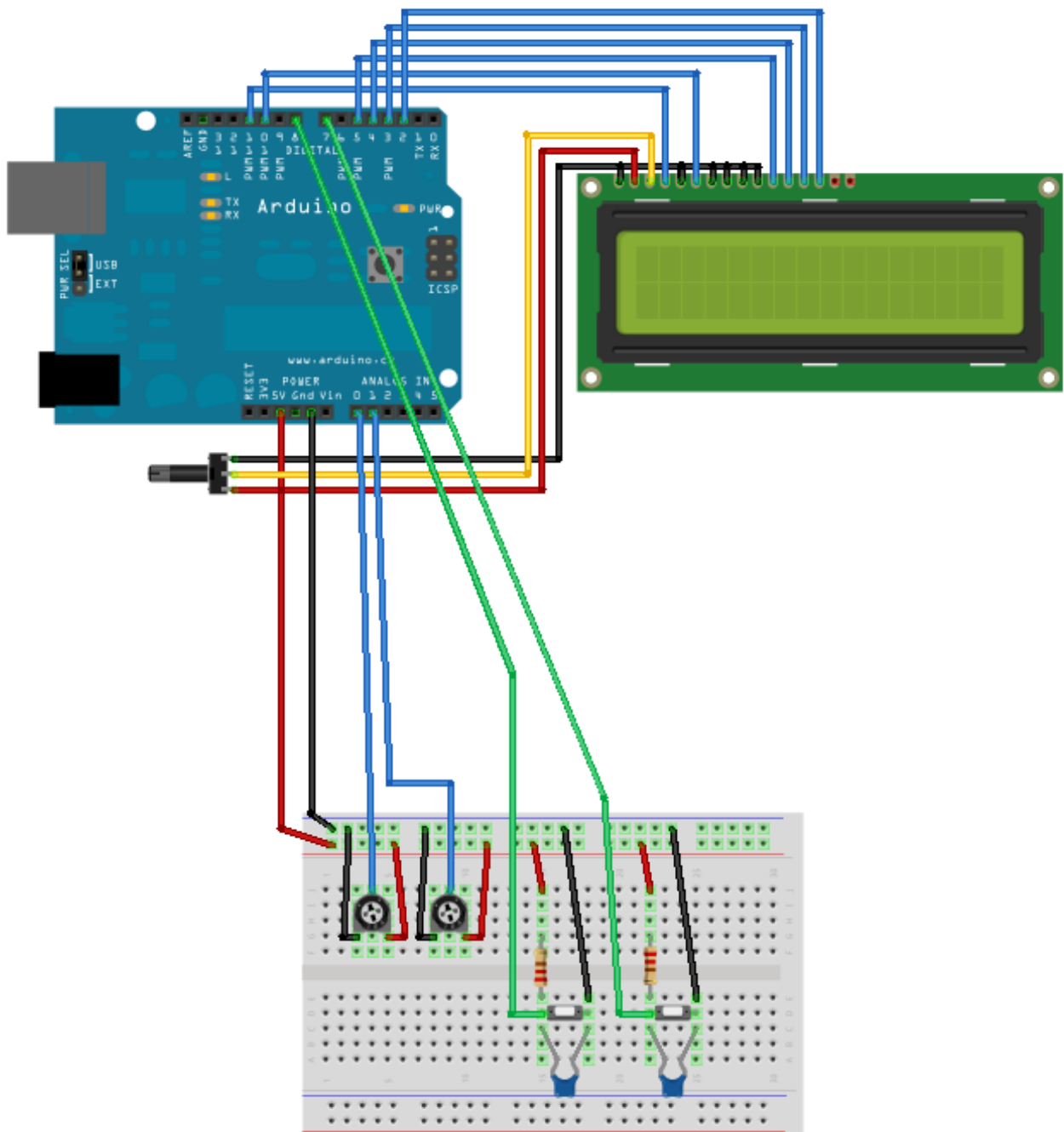
Le montage

Vous en avez l'habitude maintenant, je vais vous présenter le schéma puis ensuite le code. Pour le schéma, je n'ai pas des milliers de commentaires à faire. Parmi les choses sur lesquelles il faut être attentif se trouvent :

- Des condensateurs de filtrage pour éviter les rebonds parasites créés par les boutons
- Mettre les potentiomètres sur des entrées analogiques
- Brancher la LED dans le bon sens et ne pas oublier sa résistance de limitation de courant

Et en cas de doute, voici le schéma (qui est un peu fouillis par endroit, j'en suis désolé) !





Le code

Là encore, je vais reprendre le même schéma de fonctionnement que d'habitude en vous présentant tout d'abord les variables globales utilisées, puis les initialisations pour continuer avec quelques fonctions utiles et la boucle principale.

Les variables utilisées

Dans ce TP, beaucoup de variables vont être déclarées. En effet, il en faut déjà 5 pour les entrées/sorties (2 boutons, 2 potentiomètres, 1 LED), j'utilise aussi deux tableaux pour contenir et préparer les messages à afficher sur la première et deuxième ligne. Enfin, j'en utilise 4 pour contenir les mesures faites et 4 autres servant de mémoire pour ces mesures. Ah et j'oubliais, il me faut aussi une variable contenant le temps écoulé et une servant à savoir sur quel "interface" nous sommes en train d'écrire. Voici un petit

tableau résumant tout cela ainsi que le type des variables.

Nom	Type	Description
boutonGauche	const int	Broche du bouton de gauche
boutonDroite	const int	Broche du bouton de droite
potentiometreGauche	const int	Broche du potar "consigne"
potentiometreDroite	const int	Broche du potar "alarme"
ledAlarme	const int	Broche de la LED d'alarme
messageHaut[16]	char	Tableau représentant la ligne du haut
messageBas[16]	char	Tableau représentant la ligne du bas
etatGauche	int	État du bouton de gauche
etatDroite	int	État du bouton de droite
niveauGauche	int	Conversion du potar de gauche
niveauDroite	int	Conversion du potar de droite
etatGauche_old	int	Mémoire de l'état du bouton de gauche
etatDroite_old	int	Mémoire de l'état du bouton de droite
niveauGauche_old	int	Mémoire de la conversion du potar de gauche
niveauDroite_old	int	Mémoire de la conversion du potar de droite
temps	unsigned long	Pour mémoriser le temps écoulé
ecran	boolean	Pour savoir sur quelle interface on écrit

Le setup

Maintenant que les présentations sont faites, nous allons passer à toutes les initialisations. Le setup n'aura que peu de choses à faire puisqu'il suffira de régler les broches en entrées/sorties et de mettre en marche l'écran LCD.

```
1 void setup() {
2     //réglage des entrées/sorties
3     pinMode(boutonGauche, INPUT);
4     pinMode(boutonDroite, INPUT);
5     pinMode(ledAlarme, OUTPUT);
6
7     //paramétrage du LCD
8     lcd.begin(16, 2); // règle la taille du LCD
9     lcd.noBlink(); //pas de clignotement
10    lcd.noCursor(); //pas de curseur
11    lcd.noAutoscroll(); //pas de défilement
12 }
```

Quelques fonctions utiles

Afin de bien séparer notre code en morceaux logiques, nous allons écrire plusieurs fonctions, qui ont toutes un rôle particulier. La première d'entre toute sera celle chargée de faire le relevé des valeurs. Son objectif sera de faire les conversions analogiques et de regarder l'état des entrées numériques. Elle stockera bien entendu chacune des mesures dans la variable concernée.

```
1 void recupererDonnees()
2 {
3     //efface les anciens avec les "nouveaux anciens"
```

```

4   etatGauche_old = etatGauche;
5   etatDroite_old = etatDroite;
6   niveauGauche_old = niveauGauche;
7   niveauDroite_old = niveauDroite;
8
9   //effectue les mesures
10  etatGauche = digitalRead(boutonGauche);
11  etatDroite = digitalRead(boutonDroite);
12  niveauGauche = analogRead(potentiometreGauche);
13  niveauDroite = analogRead(potentiometreDroite);
14
15  delay(2); //pour s'assurer que les conversions analogiques sont terminées avant
16 }

```

Ensuite, deux fonctions vont nous permettre de déterminer si oui ou non il faut mettre à jour l'écran. En effet, afin d'éviter un phénomène de scintillement qui se produit si on envoie des données sans arrêt, on préfère écrire sur l'écran que si nécessaire. Pour décider si l'on doit mettre à jour les "phrases" concernant les boutons, il suffit de vérifier l'état "ancien" et l'état courant de chaque bouton. Si l'état est différent, notre fonction renvoie true, sinon elle renvoie false. Une même fonction sera codée pour les valeurs analogiques. Cependant, comme les valeurs lues par le convertisseur de la carte Arduino ne sont pas toujours très stables (je rappelle que le convertisseur offre plus ou moins deux bits de précision, soit 20mV de précision totale), on va faire une petite opération. Cette opération consiste à regarder si la valeur absolue de la différence entre la valeur courante et la valeur ancienne est supérieure à deux unités. Si c'est le cas on renvoie true sinon false.

```

1  boolean boutonsChanged()
2  {
3      //si un bouton à changé d'état
4      if(etatGauche_old != etatGauche || etatDroite_old != etatDroite)
5          return true;
6      else
7          return false;
8  }
9
10 boolean potarChanged()
11 {
12     //si un potentiomètre affiche une différence entre ces deux valeurs de plus de 2
13     if(abs(niveauGauche_old-niveauGauche) > 2 || abs(niveauDroite_old-niveauDroite) > 2)
14         return true;
15     else
16         return false;
17 }

```

Une dernière fonction nous servira à faire la mise à jour de l'écran. Elle va préparer les deux chaînes de caractères (celle du haut et celle du bas) et va ensuite les envoyer successivement sur l'écran. Pour écrire dans les chaînes, on vérifiera la valeur de la variable `ecran` pour savoir si on doit écrire les valeurs des potentiomètres ou celles des boutons. L'envoi à l'écran se fait simplement avec `print()` comme vu antérieurement. On notera le `clear()` de l'écran avant de faire les mises à jour. En effet, sans cela les valeurs pourraient se chevaucher (essayer d'écrire un OFF puis un ON, sans `clear()`, cela vous fera un "ONF" à la fin). 😊

```

1  void updateEcran()
2  {

```



```

3     if(ecran)
4     {
5         //prépare les chaines à mettre sur l'écran : boutons
6         if(etatGauche)
7             sprintf(messageHaut, "Bouton G : ON");
8         else
9             sprintf(messageHaut, "Bouton G : OFF");
10        if(etatDroite)
11            sprintf(messageBas, "Bouton D : ON");
12        else
13            sprintf(messageBas, "Bouton D : OFF");
14        }
15    else
16    {
17        //prépare les chaines à mettre sur l'écran : potentiomètres
18        sprintf(messageHaut, "gauche = %4d", niveauGauche);
19        sprintf(messageBas, "droite = %4d", niveauDroite);
20    }
21
22    //on envoie le texte
23    lcd.clear();
24    lcd.setCursor(0,0);
25    lcd.print(messageHaut);
26    lcd.setCursor(0,1);
27    lcd.print(messageBas);
28 }

```

La boucle principale

Nous voici enfin au cœur du programme, la boucle principale. Cette dernière est relativement légère, grâce aux fonctions permettant de repartir le code en unité logique. La boucle principale n'a plus qu'à les utiliser à bon escient et dans le bon ordre (😁) pour faire son travail. Dans l'ordre il nous faudra donc :

- Récupérer toutes les données (faire les conversions etc...)
- Selon l'interface courante, afficher soit les états des boutons soit les valeurs des potentiomètres si ils/elles ont changé(e)s
- Tester les valeurs des potentiomètres pour déclencher l'alarme ou non
- Enfin, si 5 secondes se sont écoulées, changer d'interface et mettre à jour l'écran

Simple non ? On ne le dira jamais assez, un code bien séparé est toujours plus facile à comprendre et à retoucher si nécessaire ! 😊 Aller, comme vous êtes sages, voici le code de cette boucle (qui va de paire avec les fonctions expliquées précédemment) :

```

1 void loop() {
2
3     recupererDonnees(); //commence par récupérer les données des boutons et capteurs
4
5     if(ecran) //quel écran affiche t'on ? (bouton ou potentiomètre ?)
6     {
7         if(boutonsChanged()) //si un bouton a changé d'état
8             updateEcran();
9     }
10    else
11    {
12        if(potarChanged()) //si un potentiomètre a changé d'état
13            updateEcran();
14    }

```

```

15
16     if(niveauDroite > niveauGauche)
17         digitalWrite(ledAlarme, LOW); //RAPPEL : piloté à l'état bas donc on allume
18     else
19         digitalWrite(ledAlarme, HIGH);
20
21     if(millis() - temps > 5000) //si ça fait 5s qu'on affiche la même donnée
22     {
23         ecran = ~ecran;
24         lcd.clear();
25         updateEcran();
26         temps = millis();
27     }
28 }

```

Programme complet

Voici enfin le code complet. Vous pourrez le copier/coller et l'essayer pour comparer si vous voulez. **Attention cependant à déclarer les bonnes broches en fonction de votre montage (notamment pour le LCD).**

```

1  #include <LiquidCrystal.h> //on inclut la librairie
2
3  //les branchements
4  const int boutonGauche = 11; //le bouton de gauche
5  const int boutonDroite = 12; //le bouton de droite
6  const int potentiometreGauche = 0; //le potentiomètre de gauche sur l'entrée analog
7  const int potentiometreDroite = 1; //le potentiomètre de droite sur l'entrée analog
8  const int ledAlarme = 2; //la LED est branché sur la sortie 2
9
10 // initialise l'écran avec les bonne broche
11 // ATTENTION, REMPLACER LES NOMBRES PAR VOS BRANCHEMENTS À VOUS !
12 LiquidCrystal lcd(8,9,4,5,6,7);
13
14 char messageHaut[16] = ""; //Message sur la ligne du dessus
15 char messageBas[16] = ""; //Message sur la ligne du dessous
16
17 unsigned long temps = 0; //pour garder une trace du temps qui s'écoule et gérer les
18 boolean ecran = LOW; //pour savoir si on affiche les boutons ou les conversions
19
20 int etatGauche = LOW; //état du bouton de gauche
21 int etatDroite = LOW; //état du bouton de droite
22 int niveauGauche = 0; //conversion du potentiomètre de gauche
23 int niveauDroite = 0; //conversion du potentiomètre de droite
24
25 //les memes variables mais "old" servant de mémoire pour constater un changement
26 int etatGauche_old = LOW; //état du bouton de gauche
27 int etatDroite_old = LOW; //état du bouton de droite
28 int niveauGauche_old = 0; //conversion du potentiomètre de gauche
29 int niveauDroite_old = 0; //conversion du potentiomètre de droite
30
31 //-----
32
33 void setup() {
34     //réglage des entrées/sorties
35     pinMode(boutonGauche, INPUT);
36     pinMode(boutonDroite, INPUT);
37     pinMode(ledAlarme, OUTPUT);
38
39     //paramétrage du LCD

```

```

40     lcd.begin(16, 2); // règle la taille du LCD
41     lcd.noBlink(); //pas de clignotement
42     lcd.noCursor(); //pas de curseur
43     lcd.noAutoscroll(); //pas de défilement
44 }
45
46 void loop() {
47     recupererDonnees(); //commence par récupérer les données des boutons et capteurs
48
49     if(ecran) //quel écran affiche t'on ? (bouton ou potentiomètre ?)
50     {
51         if(boutonsChanged()) //si un bouton a changé d'état
52             updateEcran();
53     }
54     else
55     {
56         if(potarChanged()) //si un potentiomètre a changé d'état
57             updateEcran();
58     }
59
60     if(niveauDroite > niveauGauche)
61         digitalWrite(ledAlarme, LOW); //RAPPEL : piloté à l'état bas donc on allume
62     else
63         digitalWrite(ledAlarme, HIGH);
64
65     if(millis() - temps > 5000) //si ca fait 5s qu'on affiche la même donnée
66     {
67         ecran = ~ecran;
68         lcd.clear();
69         updateEcran();
70         temps = millis();
71     }
72 }
73
74 //-----
75
76 void recupererDonnees()
77 {
78     //efface les anciens avec les "nouveaux anciens"
79     etatGauche_old = etatGauche;
80     etatDroite_old = etatDroite;
81     niveauGauche_old = niveauGauche;
82     niveauDroite_old = niveauDroite;
83
84     etatGauche = digitalRead(boutonGauche);
85     etatDroite = digitalRead(boutonDroite);
86     niveauGauche = analogRead(potentiometreGauche);
87     niveauDroite = analogRead(potentiometreDroite);
88
89     delay(1); //pour s'assurer que les conversions analogiques sont terminées avant
90 }
91
92
93 boolean boutonsChanged()
94 {
95     if(etatGauche_old != etatGauche || etatDroite_old != etatDroite)
96         return true;
97     else
98         return false;
99 }
100

```

```
101 boolean potarChanged()
102 {
103     //si un potentiomètre affiche une différence entre ces deux valeurs de plus de
104     if(abs(niveauGauche_old-niveauGauche) > 2 || abs(niveauDroite_old-niveauDroite_old))
105         return true;
106     else
107         return false;
108 }
109
110 void updateEcran()
111 {
112     if(ecran)
113     {
114         //prépare les chaines à mettre sur l'écran
115         if(etatGauche)
116             sprintf(messageHaut, "Bouton G : ON");
117         else
118             sprintf(messageHaut, "Bouton G : OFF");
119         if(etatDroite)
120             sprintf(messageBas, "Bouton D : ON");
121         else
122             sprintf(messageBas, "Bouton D : OFF");
123     }
124     else
125     {
126         //prépare les chaines à mettre sur l'écran
127         sprintf(messageHaut, "gauche = %4d", niveauGauche);
128         sprintf(messageBas, "droite = %4d", niveauDroite);
129     }
130
131     //on envoie le texte
132     lcd.clear();
133     lcd.setCursor(0,0);
134     lcd.print(messageHaut);
135     lcd.setCursor(0,1);
136     lcd.print(messageBas);
137 }
```