[Arduino 2] Gestion des entrées / sorties

Maintenant que vous avez acquis assez de connaissances en programmation et quelques notions d'électronique, on va se pencher sur l'utilisation de la carte Arduino. Je vais vous parler des entrées et des sorties de la carte et vous aider à créer votre premier programme !

[Arduino 201] Notre premier programme !

Vous voilà enfin arrivé au moment fatidique où vous allez devoir programmer ! Mais avant cela, je vais vous montrer ce qui va nous servir pour ce chapitre. En l'occurrence, apprendre à utiliser une LED et la référence, présente sur le site arduino.cc qui vous sera très utile lorsque vous aurez besoin de faire un programme utilisant une notion qui n'est pas traitée dans ce cours.

La diode électroluminescente

DEL / LED ?

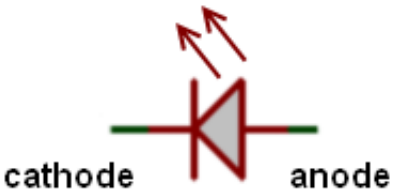
La question n'est pas de savoir quelle abréviation choisir mais plutôt de savoir qu'est ce que c'est. Une **DEL** / **LED** : **D**iode **E**lectro-**L**uminescente, ou bien "**L**ight **E**mitting **D**iode" en anglais. C'est un composant électronique qui crée de la lumière quand il est parcouru par un courant électrique. Je vous en ai fait acheter de différentes couleurs. Vous pouvez, pour ce chapitre, utiliser celle que vous voudrez, cela m'est égal. Vous voyez, sur votre droite, la photo d'une DEL de couleur rouge. La taille n'est pas réelle, sa "tête" (en rouge) ne fait que 5mm de diamètre. C'est ce composant que nous allons essayer d'allumer avec notre carte Arduino. Mais avant, voyons un peu comment il fonctionne.

J'appellerai la diode électroluminescente, tout au long du cours, une LED. Une LED est en fait une **diode** qui émet de la lumière. Je vais donc vous parler du fonctionnement des diodes en même temps que celui des LED.

Symbole

Sur un schéma électronique, chaque composant est repéré par un symbole qui lui est propre. Celui de la diode est celui-ci :

Celui de la LED est :



Il y a donc très peu de différence entre les deux. La LED est simplement une diode qui émet de la lumière, d'où les flèches sur son symbole.

Astuce mnémotechnique

Pour ce souvenir de quel côté est l'anode ou la cathode, voici une toute simple et en image …

K comme K-thode A comme A-node

Fonctionnement

Polarisation directe

On parle de **polarisation** lorsqu'un composant électronique est utilisé dans un circuit électronique de la "bonne manière". En fait lorsqu'il est polarisé, c'est qu'on l'utilise de la façon souhaitée. Pour polariser la diode, on doit faire en sorte que le courant doit la parcourir de l'anode vers la cathode. Autrement dit, la tension doit être plus élevée à l'anode qu'à la cathode.

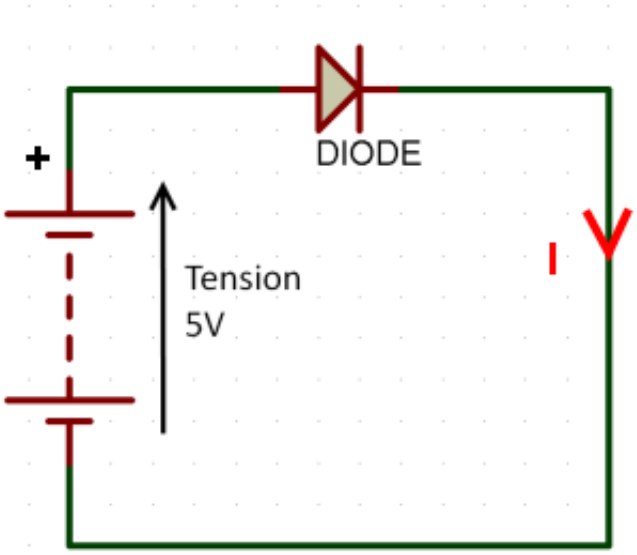


Figure 1 : diode polarisée directement

Polarisation inverse

La polarisation inverse d'une diode est l'opposé de la polarisation directe. Pour créer ce type de montage, il suffit simplement, dans notre cas, de "retourner" la diode enfin la brancher "à l'envers". Dans ce cas, le courant ne passe pas.

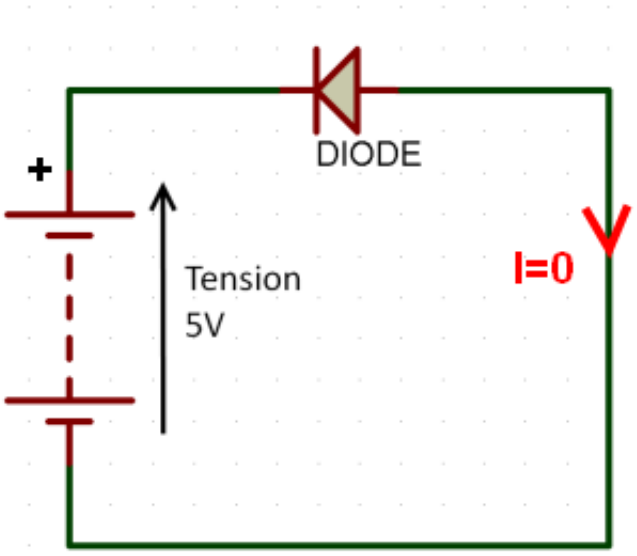


Figure 2 : diode polarisée en inverse

Note : une diode polarisée en inverse ne grillera pas si elle est utilisée dans de bonnes conditions. En fait, elle fonctionne de "la même façon" pour le courant positif et négatif.

Utilisation

Si vous ne voulez pas faire partir votre première diode en fumée, je vous conseille de lire les prochaines lignes attentivement

En électronique, deux paramètres sont à prendre en compte: le courant et la tension.

Pour une diode, deux tensions sont importantes. Il s'agit de la tension maximum en polarisation directe, et la tension maximum en polarisation inverse. Ensuite, pour un bon fonctionnement des LED, le courant à lui aussi son importance.

La tension maximum directe

Lorsque l'on utilise un composant, on doit prendre l'habitude d'utiliser la "datasheet" ("documentation technique" en anglais) qui nous donne toutes les caractéristiques sur le composant. Dans cette datasheet, on retrouvera quelque chose appelé "Forward Voltage", pour la diode. Cette indication représente la chute de tension aux bornes de la diode lorsque du courant la traverse en sens direct. Pour une diode classique (type 1N4148), cette tension sera d'environ 1V. Pour une led, on considérera plutôt une tension de 1,2 à 1,6V.

Bon, pour faire nos petits montages, on ne va pas chipoter, mais c'est la démarche à faire lorsque l'on conçoit un schéma électrique et que l'on dimensionne ses composants.

La tension maximum inverse

Cette tension représente la différence maximum admissible entre l'anode et la cathode lorsque celle-ci est branchée "à l'envers". En effet, si vous mettez une tension trop importante à ces bornes, la jonction ne pourra pas le supporter et partira en fumée. En anglais, on retrouve cette tension sous le nom de "Reverse Voltage" (ou même "Breakdown Voltage"). Si l'on reprend la diode 1N4148, elle sera comprise entre 75 et 100V. Au-delà de cette tension, la jonction casse et la diode devient inutilisable. Dans ce cas, la diode devient soit un court-circuit, soit un circuit ouvert. Parfois cela peu causer des dommages importants dans nos appareils électroniques ! Quoi qu'il en soit, on ne manipulera jamais du 75V ! 

Le courant de passage

Pour une LED, le courant qui la traverse à son importance. Si l'on branche directement la led sur une pile, elle va s'allumer, puis tôt ou tard finira par s'éteindre… définitivement. En effet, si on ne limite pas le courant traversant la LED, elle prendra le courant maximum, et ça c'est pas bon car ce n'est pas le courant maximum qu'elle peut supporter. Pour limiter le courant, on place une résistance avant (ou après) la LED. Cette résistance, savamment calculée, lui permettra d'assurer un fonctionnement optimal.

Mais comment on la calcule cette résistance ?

Simplement avec la formule de base, la loi d'ohm.  Petit rappel:

$$U = R * I$$

Dans le cas d'une LED, on considère, en général, que l'intensité la traversant doit-être de 20 mA. Si on veut être rigoureux, il faut aller chercher cette valeur dans le datasheet. On a donc $I = 20mA$. Ensuite, on prendra pour l'exemple une tension d'alimentation de 5V (en sortie de l'Arduino, par exemple) et une tension aux bornes de la LED de

1,2V en fonctionnement normal. On peut donc calculer la tension qui sera aux bornes de la résistance : $Ur = 5 - 1,2 = 3,8V$ Enfin, on peut calculer la valeur de la résistance à utiliser : Soit : $R = \frac{U}{I}$ $R = \frac{3,8}{0,02}$ $R = 190\Omega$ Et voila, vous connaissez la valeur de la résistance à utiliser pour être sur de ne pas griller des LED à tour de bras. A votre avis, vaut-il mieux utiliser une résistance de plus forte valeur ou de plus faible valeur ?

Si on veut être sûr de ne pas détériorer la LED à cause d'un courant trop fort, on doit placer une résistance dont la valeur est plus grande que celle calculée. Autrement, la diode admettrait un courant plus intense qui circulerait en elle et cela pourrait la détruire.

Par quoi on commence ?

Le but

Le but de ce premier programme est… de vous faire programmer ! Non, je ne rigole pas ! Car c'est en pratiquant la programmation que l'on retient le mieux les commandes utilisées. De plus, en faisant des erreurs, vous vous forgerez de bonnes bases qui vous seront très utiles ensuite, lorsqu'il s'agira de gagner du temps. Mais avant tout, c'est aussi parce que ce tuto est centré sur la programmation que l'on va programmer !

Objectif

L'objectif de ce premier programme va consister à allumer une LED. C'est nul me direz vous. J'en conviens. Cependant, vous verrez que ce n'est pas très simple. Bien entendu, je n'allais pas créer un chapitre entier dont le but ultime aurait été d'allumer une LED ! Non. Alors j'ai prévu de vous montrer deux trois trucs qui pourront vous aider dès lors que vous voudrez sortir du nid et prendre votre envol vers de nouveaux cieux !

Matériel

Pour pouvoir programmer, il vous faut, bien évidemment, une carte Arduino et un câble USB pour relier la carte au PC. Mais pour voir le résultat de votre programme, vous aurez besoin d'éléments supplémentaires. Notamment, une LED et une résistance.

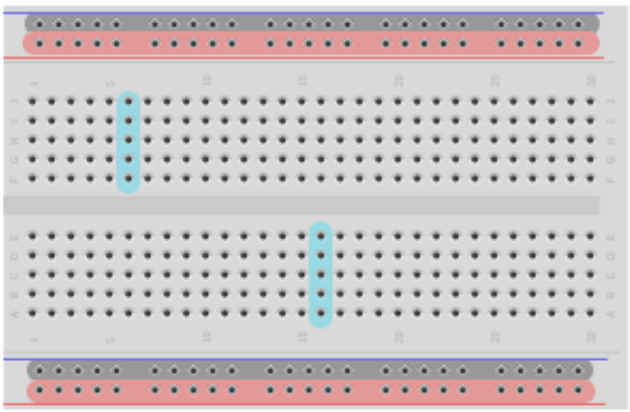
Un outil formidable : la breadboard !

Je vais maintenant vous présenter un outil très pratique lorsque l'on fait ses débuts en électronique ou lorsque l'on veut tester rapidement/facilement un montage. Cet accessoire s'appelle une **breadboard** (littéralement : Planche à pain, techniquement : plaque d'essai sans soudure). Pour faire simple, c'est une plaque pleine de trous !

Principe de la breadboard

Certes la plaque est pleine de trous, mais pas de manière innocente ! En effet, la plupart d'entre eux sont reliés. Voici un petit schéma rapide qui va aider à la

compréhension.



Comme vous pouvez le voir sur l'image, j'ai dessiné des zones. Les zones rouges et noires correspondent à l'alimentation. Souvent, on retrouve deux lignes comme celles-ci permettant de relier vos composants aux alimentations nécessaires. Par convention, le noir représente la masse et le rouge est l'alimentation (+5V, +12V, -5V… ce que vous voulez y amener). Habituellement tous les trous d'une même **ligne** sont reliés sur cette zone. Ainsi, vous avez une ligne d'alimentation parcourant tout le long de la carte. Ensuite, on peut voir des zones en bleu. Ces zones sont reliées entre elles par **colonne**. Ainsi, tous les trous sur une même colonne sont reliés entre eux. En revanche, chaque colonne est distincte. En faisant chevaucher des composants sur plusieurs colonnes vous pouvez les connecter entre eux. Dernier point, vous pouvez remarquer un espace coupant la carte en deux de manière symétrique. Cette espace coupe aussi la liaison des colonnes. Ainsi, sur le dessin ci-dessus on peut voir que chaque colonne possède 5 trous reliés entre eux. Cet espace au milieu est normalisé et doit faire la largeur des circuits intégrés standards. En posant un circuit intégré à cheval au milieu, chaque patte de ce dernier se retrouve donc sur une colonne, isolée de la précédente et de la suivante.

Si vous voulez voir plus concrètement ce fonctionnement, je vous conseille d'essayer le logiciel Fritzing, qui permet de faire des circuits de manière assez simple et intuitive. Vous verrez ainsi comment les colonnes sont séparées les unes des autres. De plus, ce logiciel sera utilisé pour le reste du tuto pour les captures d'écrans des schémas électroniques.

Réalisation

Avec le brochage de la carte Arduino, vous devrez connecter la plus grande patte au +5V (broche *5V*). La plus petite patte étant reliée à la résistance, elle-même reliée à la broche numéro 2 de la carte. Tout ceci a une importance. En effet, on pourrait faire le contraire, brancher la LED vers la masse et l'allumer en fournissant le 5V depuis la broche de signal. Cependant, les composants comme les microcontrôleurs n'aiment pas trop délivrer du courant, ils préfèrent l'absorber. Pour cela, on préférera donc alimenter la LED en la placant au +5V et en mettant la broche de Arduino à la masse pour faire passer le courant. Si on met la broche à 5V, dans ce cas le potentiel est le même de chaque côté de la LED et elle ne s'allume pas ! Ce n'est pas plus compliqué que ça ! 😉 Schéma de la réalisation (un exemple de branchement sans breadboard et deux exemples avec) :

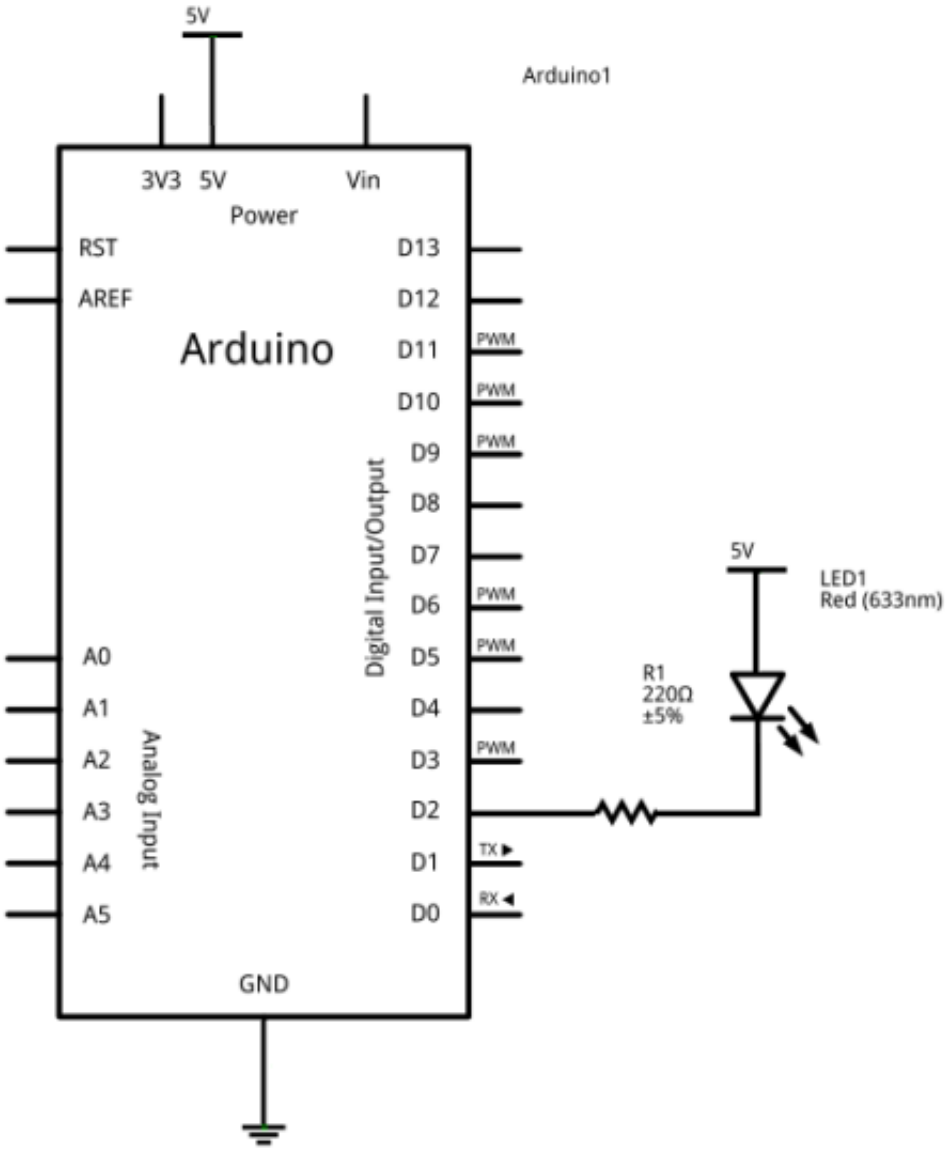
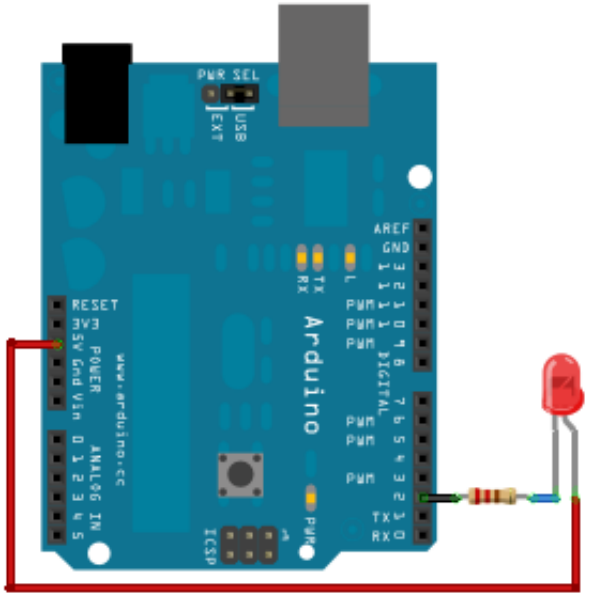
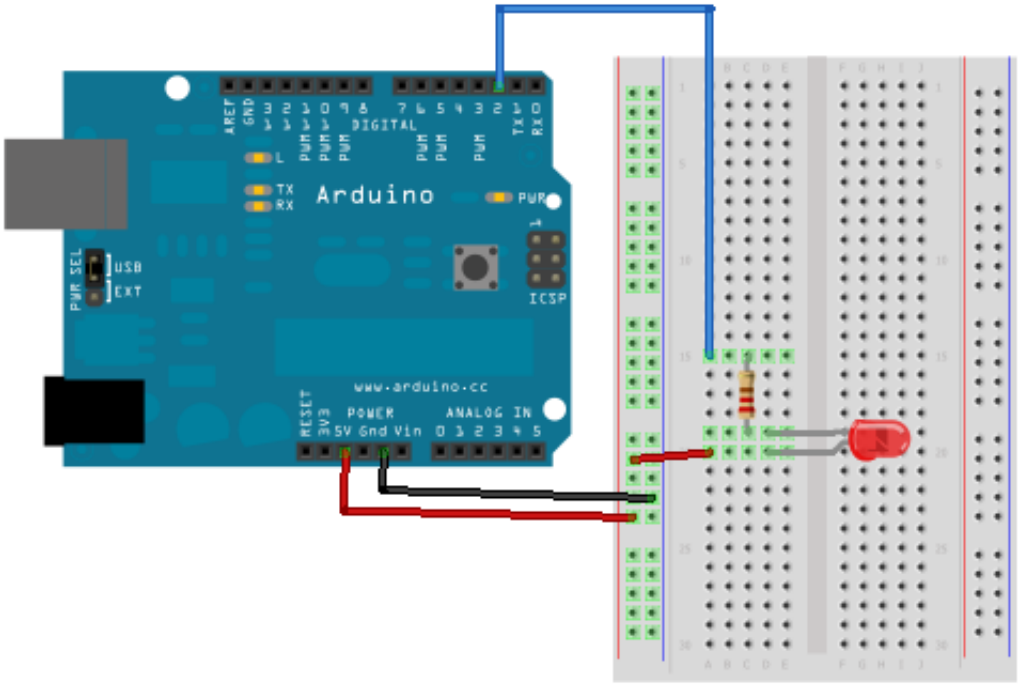
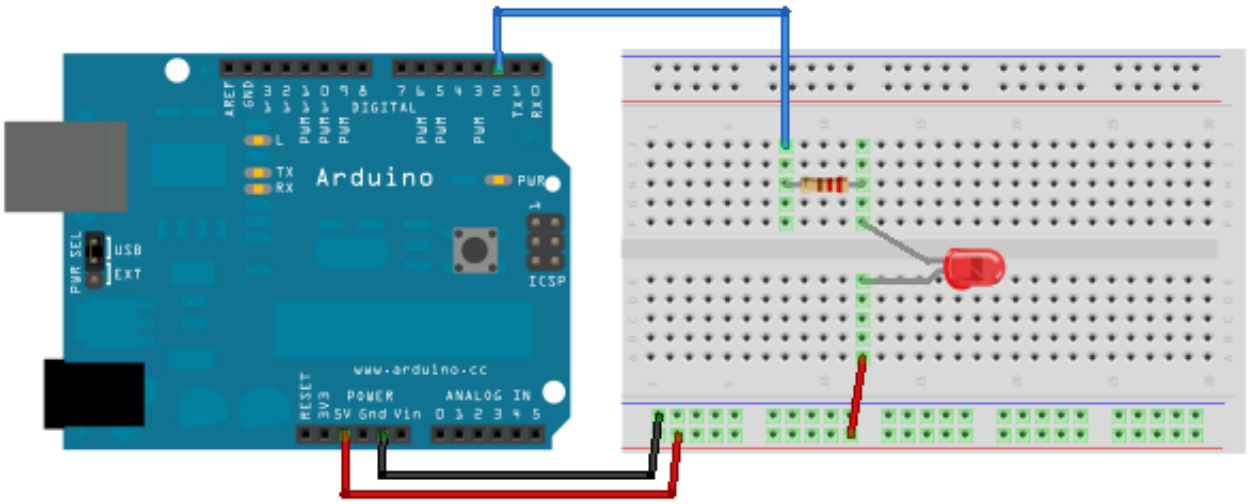


Figure 3 : réalisation montage, schéma de la carte







Créer un nouveau projet

Pour pouvoir programmer notre carte, il faut que l'on créer un nouveau programme. Ouvrez votre logiciel Arduino. Allez dans le menu *File* Et choisissez l'option *Save as…* :

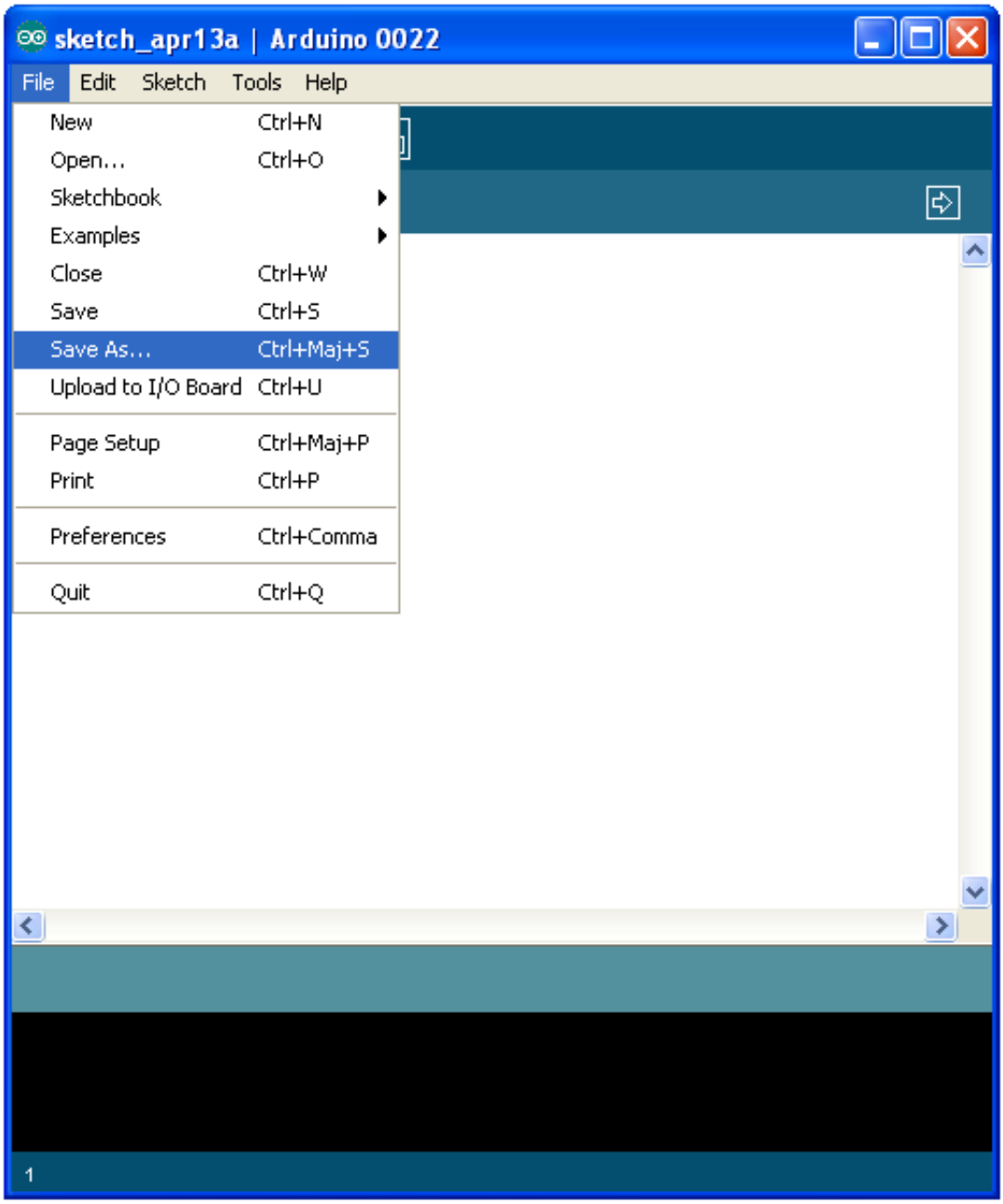


Figure 4 : Enregistrer sous…

Vous arrivez dans cette nouvelle fenêtre :

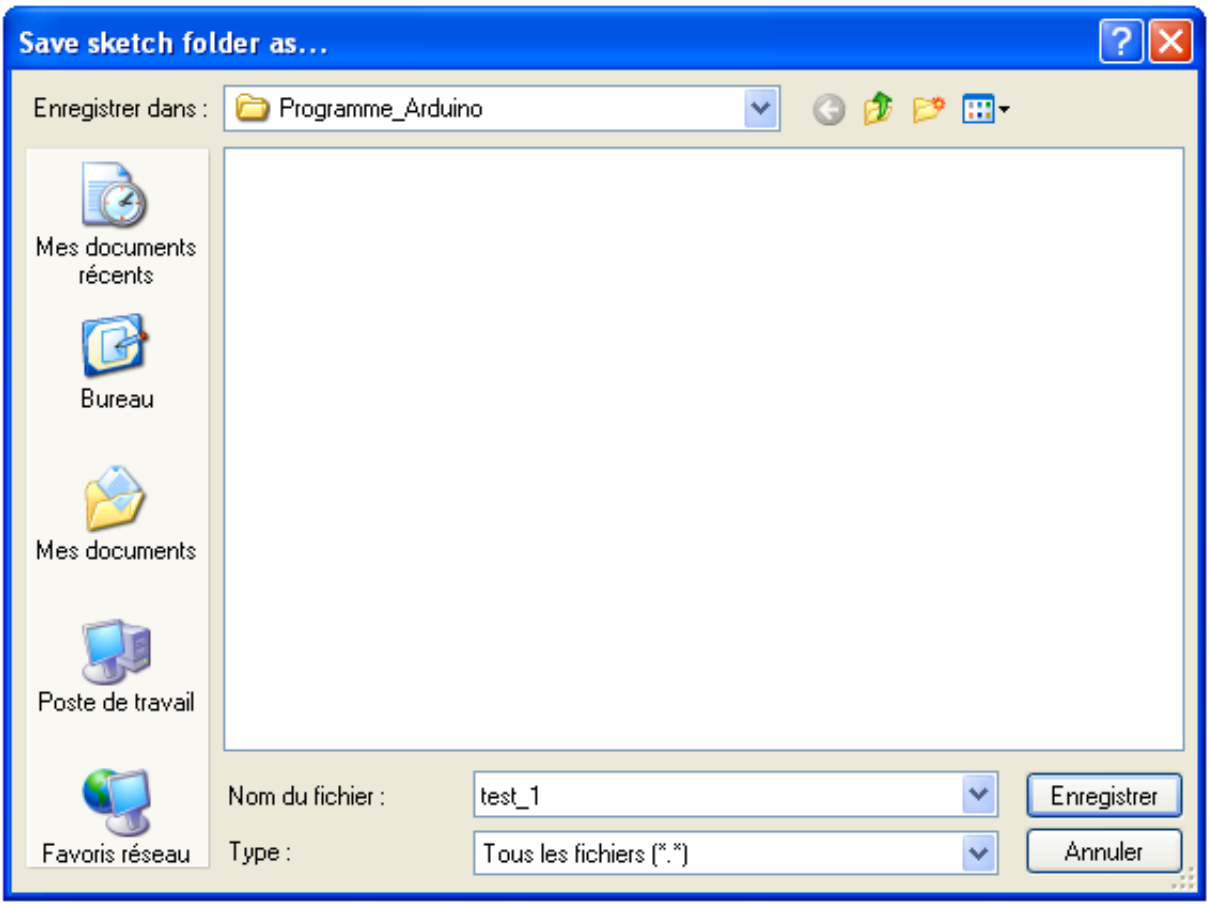


Figure 5 : Enregistrer

Tapez le nom du programme, dans mon cas, je l'ai appelé *test_1* . Enregistrez. vous arriver dans votre nouveau programme, qui est vide pour l'instant, et dont le nom s'affiche en Haut de la fenêtre et dans un petit onglet :

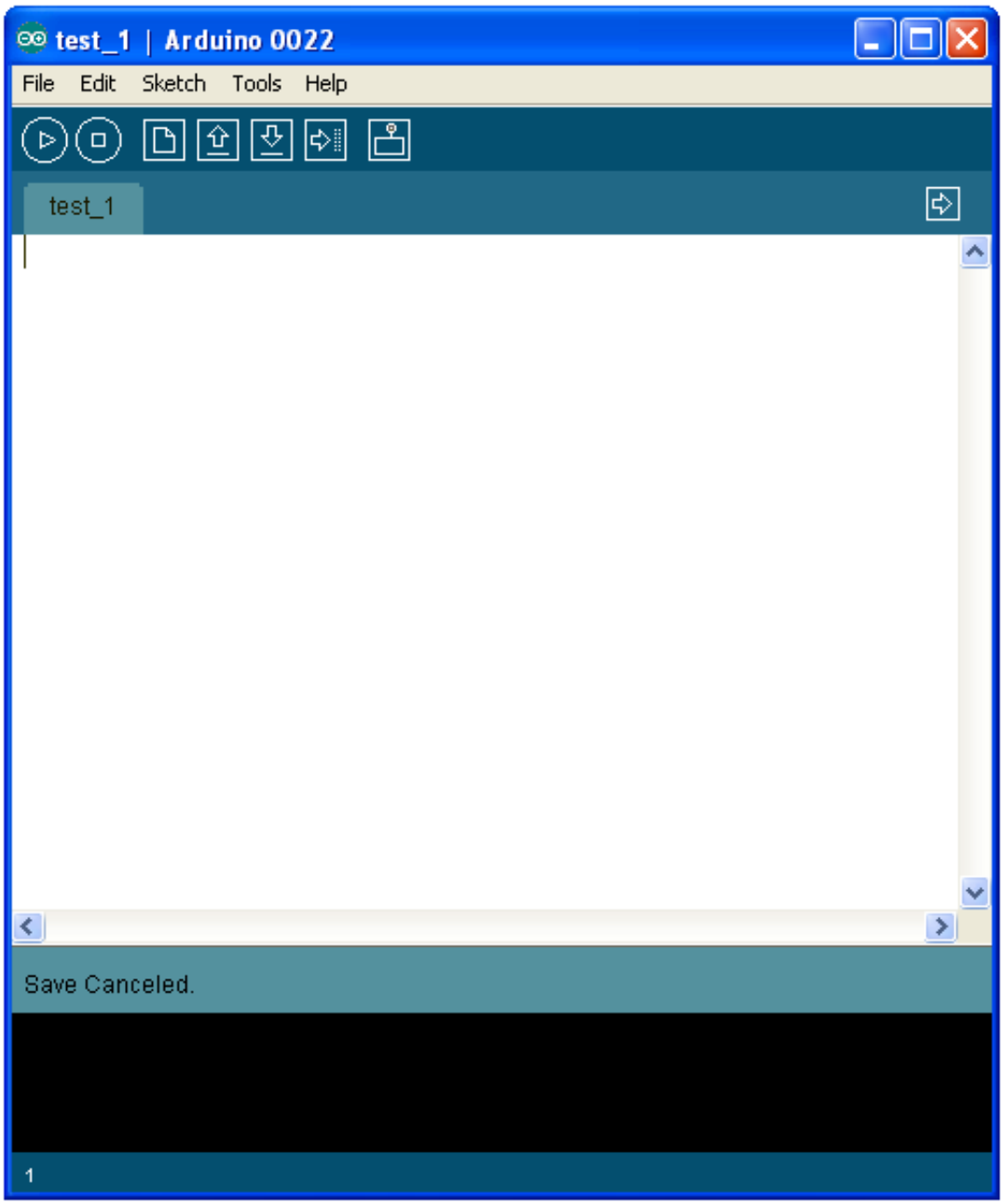


Figure 6 : Votre nouveau programme !

Le code minimal

Pour commencer le programme, il nous faut un code minimal. Ce code va nous permettre d'initialiser la carte et va servir à écrire notre propre programme. Ce code, le voici :

```
1  //fonction d'initialisation de la carte
2  void setup()
3  {
4      //contenu de l'initialisation
5  }
6
7  //fonction principale, elle se répète (s'exécute) à l'infini
8  void loop()
9  {
10     //contenu de votre programme
11 }
```

Créer le programme : les bons outils !

La référence Arduino

Qu'est ce que c'est ?

L'Arduino étant un projet dont la communauté est très active, nous offre sur son site internet une **référence**. Mais qu'est ce que c'est ? Et bien il s'agit simplement de "la notice d'utilisation" du langage Arduino. Plus exactement, une page internet de leur site est dédiée au référencement de chaque code que l'on peut utiliser pour faire un programme.

Comment l'utiliser ?

Pour l'utiliser, il suffit d'aller sur la page de leur site, malheureusement en anglais, mais dont il existe une traduction pas tout à fait complète sur le site Français Arduino. Ce que l'on voit en arrivant sur la page : trois colonnes avec chacune un type d'éléments qui forment les langages Arduino.

- *Structure* : cette colonne référence les éléments de la structure du langage Arduino. On y retrouve les conditions, les opérations, etc.
- *Variables* : Comme son nom l'indique, elle regroupe les différents types de variables utilisables, ainsi que certaines opérations particulières
- *Functions* : Ici c'est tout le reste, mais surtout les fonctions de lecture/écriture des broches du microcontrôleur (ainsi que d'autres fonctions bien utiles)

Il est très important de savoir utiliser la documentation que nous offre Arduino ! Car en sachant cela, vous pourrez faire des programmes sans avoir appris préalablement à utiliser telle fonction ou telle autre. Vous pourrez devenir les maitres du monde !!! Euh, non, je crois pas en fait… 

Allumer notre LED

1ère étape

Il faut avant tout définir les broches du micro-contrôleur. Cette étape constitue elle-même deux sous étapes. La première étant de créer une variable définissant la broche utilisée, ensuite, définir si la broche utilisée doit être une entrée du micro-contrôleur ou une sortie. Premièrement, donc, définissons la broche utilisée du micro-contrôleur :

```
1 const int led_rouge = 2;    //définition de la broche 2 de la carte en tant que variab
```

Le terme **const** signifie que l'on définit la variable comme étant constante. Par conséquent, on change la nature de la variable qui devient alors constante. Le terme **int** correspond à un type de variable. En définissant une variable de ce type, elle peut stocker un nombre allant de -2147483648 à +2147483647 ! Cela nous suffit amplement !  Nous sommes donc en présence d'une variable, nommée *led_rouge*, qui est en fait une constante, qui peut prendre une valeur allant de -2147483648 à +2147483647. Dans notre cas, cette variable, pardon constante, est assignée à 2. Le chiffre 2.

Lorsque votre code sera compilé, le micro-contrôleur saura ainsi que sur sa broche numéro 2, il y a un élément connecté.

Bon, cela ne suffit pas de définir la broche utilisée. Il faut maintenant dire si cette broche est une **entrée** ou une **sortie**. Oui, car le micro-contrôleur a la capacité d'utiliser certaines de ses broches en entrée ou en sortie. C'est fabuleux ! En effet, il suffit simplement d'interchanger UNE ligne de code pour dire qu'il faut utiliser une broche en entrée (récupération de donnée) ou en sortie (envoi de donnée). Cette ligne de code justement, parlons-en ! Elle doit se trouver dans la fonction **setup()**. Dans la référence, ce dont nous avons besoin se trouve dans la catégorie **Functions**, puis dans **Digital I/O**. I/O pour Input/Output, ce qui signifie dans la langue de Molière : Entrée/Sortie. La fonction se trouve être **pinMode()**. Pour utiliser cette fonction, il faut lui envoyer deux paramètres :

- Le nom de la variable que l'on a défini à la broche
- Le type de broche que cela va être (entrée ou sortie)

```
1  //fonction d'initialisation de la carte
2  void setup()
3  {
4      //initialisation de la broche 2 comme étant une sortie
5      pinMode(led_rouge, OUTPUT);
6  }
```

Ce code va donc définir la led_rouge (qui est la broche numéro 2 du micro-contrôleur) en sortie, car OUTPUT signifie en français : *sortie*. Maintenant, tout est prêt pour créer notre programme. Voici le code quasiment complet :

```
1  //définition de la broche 2 de la carte en tant que variable
2  const int led_rouge = 2;
3
4  //fonction d'initialisation de la carte
5  void setup()
6  {
7      //initialisation de la broche 2 comme étant une sortie
8      pinMode(led_rouge, OUTPUT);
9  }
10
11 //fonction principale, elle se répète (s'exécute) à l'infini
12 void loop()
13 {
14     //contenu de votre programme
15 }
```

2e étape

Cette deuxième étape consiste à créer le contenu de notre programme. Celui qui va aller remplacer le commentaire dans la fonction **loop()**, pour réaliser notre objectif : allumer la LED ! Là encore, on ne claque pas des doigts pour avoir le programme tout prêt ! Il faut retourner chercher dans la référence Arduino ce dont on a besoin.

Oui, mais là, on ne sait pas ce que l'on veut ? o_O

On cherche une fonction qui va nous permettre d'allumer cette LED. Il faut donc que l'on

se débrouille pour la trouver. Et avec notre niveau d'anglais, on va facilement trouver. Soyons un peu logique, si vous le voulez bien. Nous savons que c'est une fonction qu'il nous faut (je l'ai dis il y a un instant), on regarde donc dans la catégorie **Functions** de la référence. Si on garde notre esprit logique, on va s'occuper d'allumer une LED, donc de dire quel est l'état de sortie de la broche numéro 2 où laquelle est connectée notre LED. Donc, il est fort à parier que cela se trouve dans **Digital I/O**. Tiens, il y a une fonction suspecte qui se prénomme **digitalWrite()**. En français, cela signifie "écriture numérique". C'est donc l'écriture d'un état logique (0 ou 1). Quel se trouve être la première phrase dans la description de cette fonction ? Celle-ci : "Write a HIGH or a LOW value to a digital pin". D'après notre niveau bilingue, on peut traduire par : *Ecriture d'une valeur HAUTE ou une valeur BASSE sur une sortie numérique*. Bingo ! C'est ce que l'on recherchait ! Il faut dire que je vous ai un peu aidé.

Ce signifie quoi "valeur HAUTE ou valeur BASSE" ?

En électronique numérique, un niveau haut correspondra à une tension de +5V et un niveau dit bas sera une tension de 0V (généralement la masse). Sauf qu'on a connecté la LED au pôle positif de l'alimentation, donc pour qu'elle s'allume, il faut qu'elle soit reliée au 0V. Par conséquent, on doit mettre un état bas sur la broche du microcontrôleur. Ainsi, la différence de potentiel aux bornes de la LED permettra à celle-ci de s'allumer Voyons un peu le fonctionnement de **digitalWrite()** en regardant dans sa syntaxe. Elle requiert deux paramètres. Le nom de la broche que l'on veut mettre à un état logique et la valeur de cet état logique. Nous allons donc écrire le code qui suit, d'après cette syntaxe :

```
1 digitalWrite(led_rouge, LOW); // écriture en sortie (broche 2) d'un état BAS
```

Si on teste le code entier :

```
1  //définition de la broche 2 de la carte en tant que variable
2  const int led_rouge = 2;
3
4  //fonction d'initialisation de la carte
5  void setup()
6  {
7      //initialisation de la broche 2 comme étant une sortie
8      pinMode(led_rouge, OUTPUT);
9  }
10
11 //fonction principale, elle se répète (s'exécute) à l'infini
12 void loop()
13 {
14     // écriture en sortie (broche 2) d'un état BAS
15     digitalWrite(led_rouge, LOW);
16 }
```

On voit s'éclairer la LED !!! C'est fantastique !

Comment tout cela fonctionne ?

Et comment ça se passe à l'intérieur ?? o_O Je comprends pas comment le microcontrôleur y fait pour tout comprendre et tout faire. Je sais qu'il utilise les 0 et

les 1 du programme qu'on lui a envoyé, mais comment il sait qu'il doit aller chercher le programme, le lire, l'exécuter, etc. ?

Eh bien, eh bien ! En voilà des questions ! Je vais essayer d'y répondre simplement, sans entrer dans le détail qui est quand même très compliqué. Bon, si vous êtes prêt, c'est partit ! D'abord, tout se passe dans le cerveau du microcontrôleur…

Le démarrage

Un peu comme vous démarreriez un ordinateur, la carte Arduino aussi démarre. Alors c'est un peu transparent parce qu'elle démarre dans deux cas principaux : le premier c'est lorsque vous la branchez sur le port USB ou une sur autre source d'alimentation ; le deuxième c'est lorsque le compilateur a fini de charger le programme dans la carte, il la redémarre. Et au démarrage de la carte, il se passe des trucs.

Chargez !

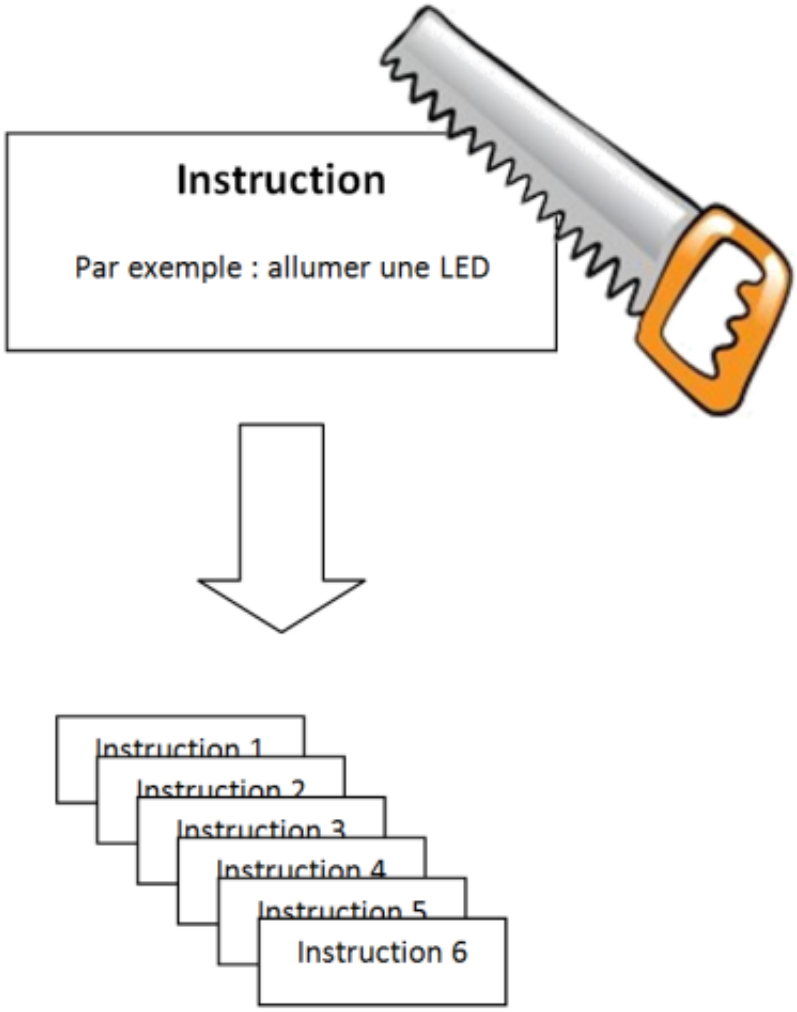
Vous vous souvenez du chapitre où je vous présentais un peu le fonctionnement global de la carte ? Oui, celui-là. Je vous parlais alors de l'exécution du programme. Au démarrage, la carte (après un petit temps de vérification pour voir si le compilateur ne lui charge pas un nouveau programme) commence par aller charger les variables en mémoire de données. C'est un petit mécanisme électronique qui va simplement faire en sorte de copier les variables inscrites dans le programme vers la mémoire de données. En l'occurrence, dans le programme que l'on vient de créer, il n'y a qu'une variable et elle est constante en plus. Ce ne sera donc pas bien long à mettre ça en mémoire ! Ensuite, vient la lecture du programme. Et là, que peut-il bien se passer à l'intérieur du microcontrôleur ? En fait, ce n'est pas très compliqué (sur le principe).

La vraie forme du programme

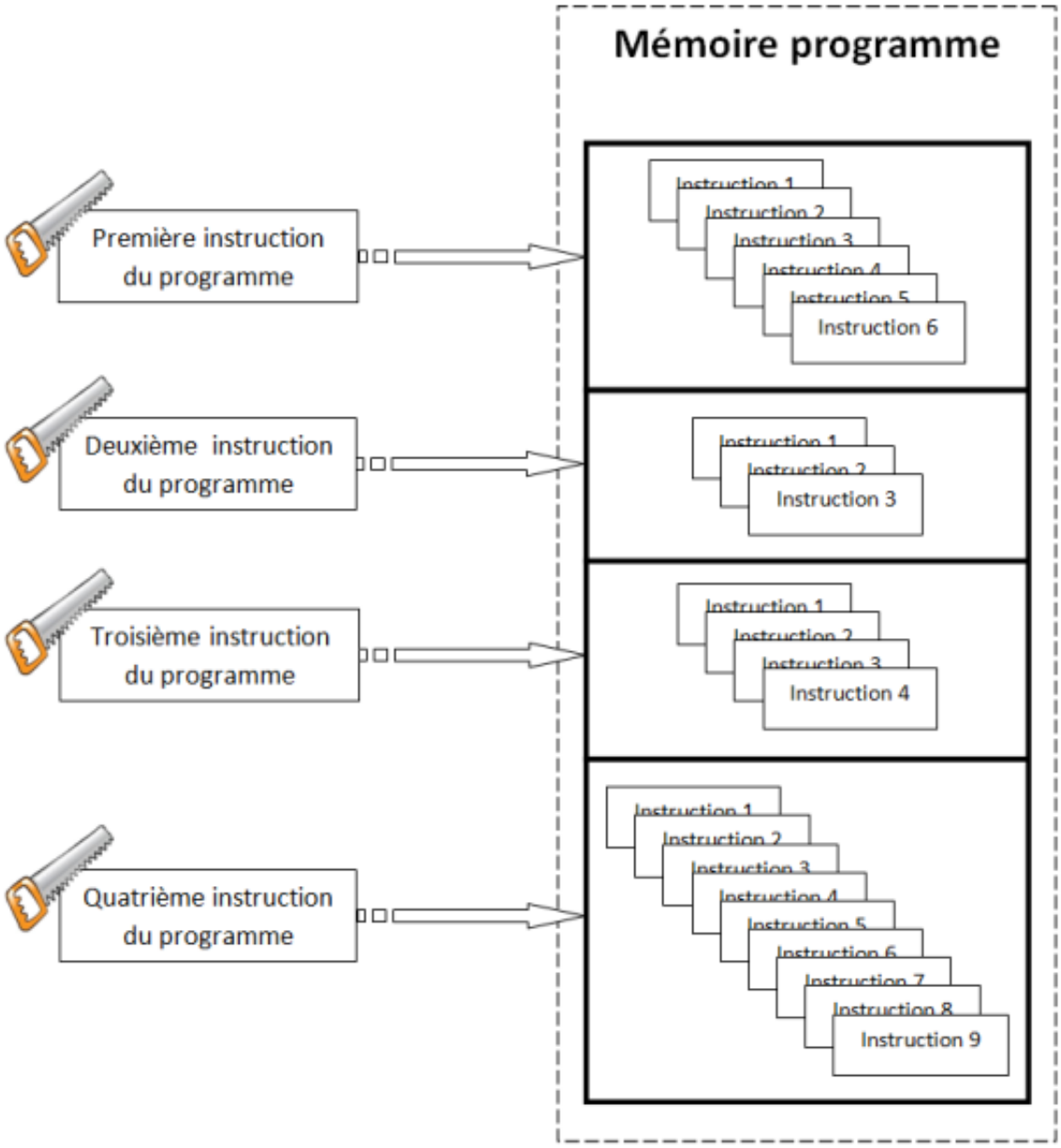
A présent, le cerveau du microcontrôleur va aller lire la première instruction du programme. Celle qui se trouve dans la fonction `setup()`. Sauf que, l'instruction n'est plus sous la même forme. Non cette fois-ci je ne parle pas des 0 et des 1, mais bien d'une transformation de l'instruction. C'est le compilateur qui a découpé chaque instruction du programme en plusieurs petites instructions beaucoup plus simples.

Et pourquoi cela ? Le microcontrôleur ne sais pas faire une instruction aussi simple que de déclarer une broche en sortie ou allumer une LED ? o_O

Oui. C'est pourquoi il a besoin que le programme soit non plus sous forme de "grandes instructions" comme on l'a écrit, mais bien sous forme de plusieurs petites instructions. Et cela est du au fait qu'il ne sait exécuter que des instructions très simples !



Bien entendu, il n'y a pas de limite à 6 instructions, il peut y en avoir beaucoup plus ou beaucoup moins ! Donc, en mémoire de programme, là où le programme de la carte est stocké, on va avoir plutôt quelque chose qui ressemble à ça :



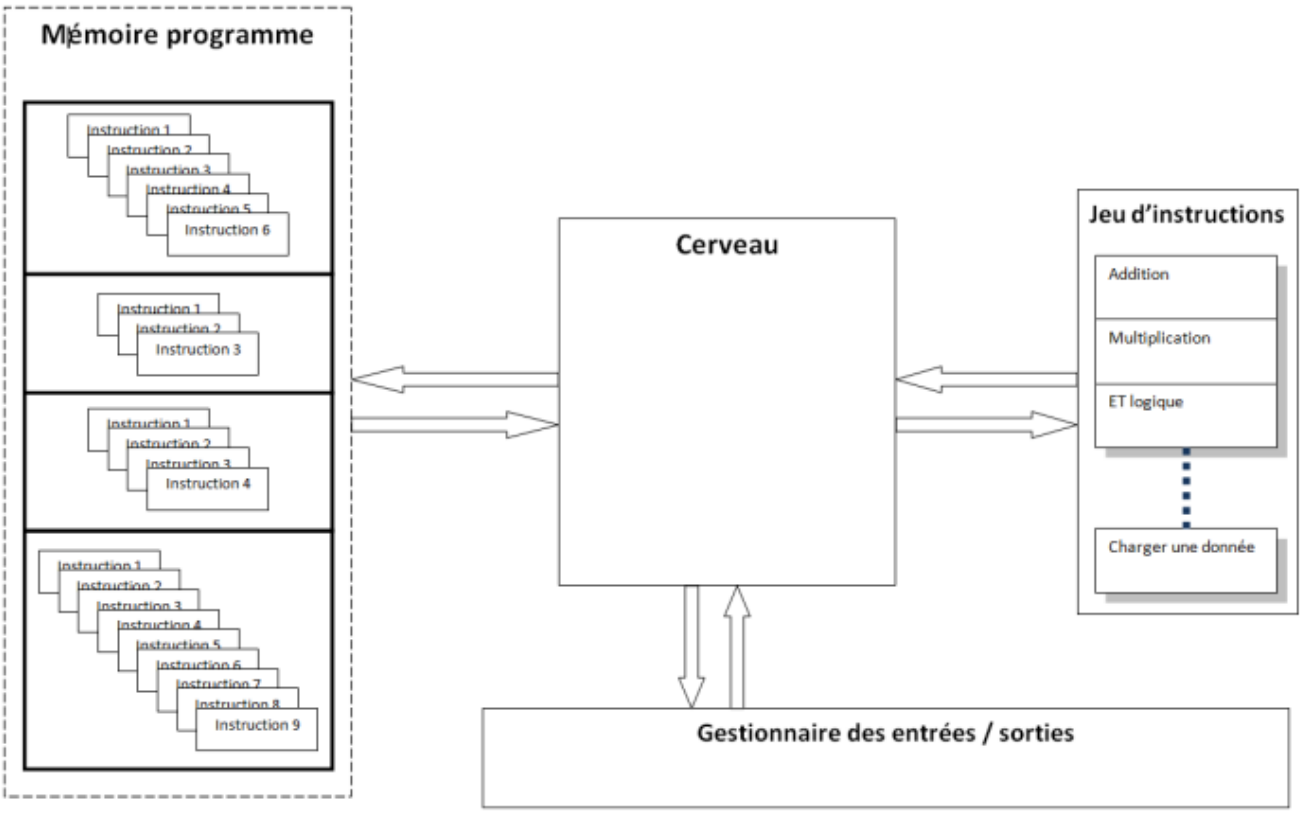
Chaque grande instruction est découpée en petite instructions par le compilateur et est ensuite stockée dans la mémoire de programme. Pour être encore plus détaillé, chaque instruction agit sur un *registre*. Un registre, c'est la forme la plus simplifié de la mémoire en terme de programmation. On en trouve plusieurs, par exemple le registre des timers ou celui des entrées/sorties du port A (ou B, ou C) ou encore des registres généraux pour manipuler les variables. Par exemple, pour additionner 3 à la variable 'a' le microcontrôleur fera les opérations suivantes :

- - chargement de la variable 'a' dans le registre général 8 (par exemple) depuis la RAM
- - chargement de la valeur 3 dans le registre général 9
- - mise du résultat de "registre 8 + registre 9″ dans le registre 8
- - changement de la valeur de 'a' en RAM depuis le registre 8

Et l'exécution du programme

A présent que l'on a plein de petites instructions, qu'avons nous de plus ? Pas grand

chose me direz-vous. Le Schmilblick n'a guère avancé… Pour comprendre, il faut savoir que le microcontrôleur ne sais faire que quelques instructions. Ces instructions sont encore plus simple que d'allumer une LED ! Il peut par exemple faire des opérations logique (ET, OU, NON, décalage de bits, …), des opérations numérique (addition et soustraction, les multiplication et division sont fait avec des opérations de types décalage de bits) ou encore copier et stocker des données. Il sait en faire, donc, mais pas tant que ça. Tout ce qu'il sait faire est régie par son **jeu d'instructions**. C'est à dire qu'il a une liste des instructions possible qu'il sait exécuter et il s'y tient. Le compilateur doit donc absolument découper chaque instruction du programme en instructions que le microcontrôleur sait exécuter.



Le cerveau du microcontrôleur va aller lire le programme, il compare ensuite chaque instruction à son registre d'instruction et les exécute. Pour allumer une LED, il fera peut-être un ET logique, chargera une donnée, fera une soustraction, … on ne sait pas mais il va y arriver. Et pour terminer, il communiquera à son gestionnaire d'entrées/sortie pour lui informer qu'il faut activer tel transistor interne pour mettre une tension sur telle broche de la carte pour ainsi allumer la LED qui y est connectée.

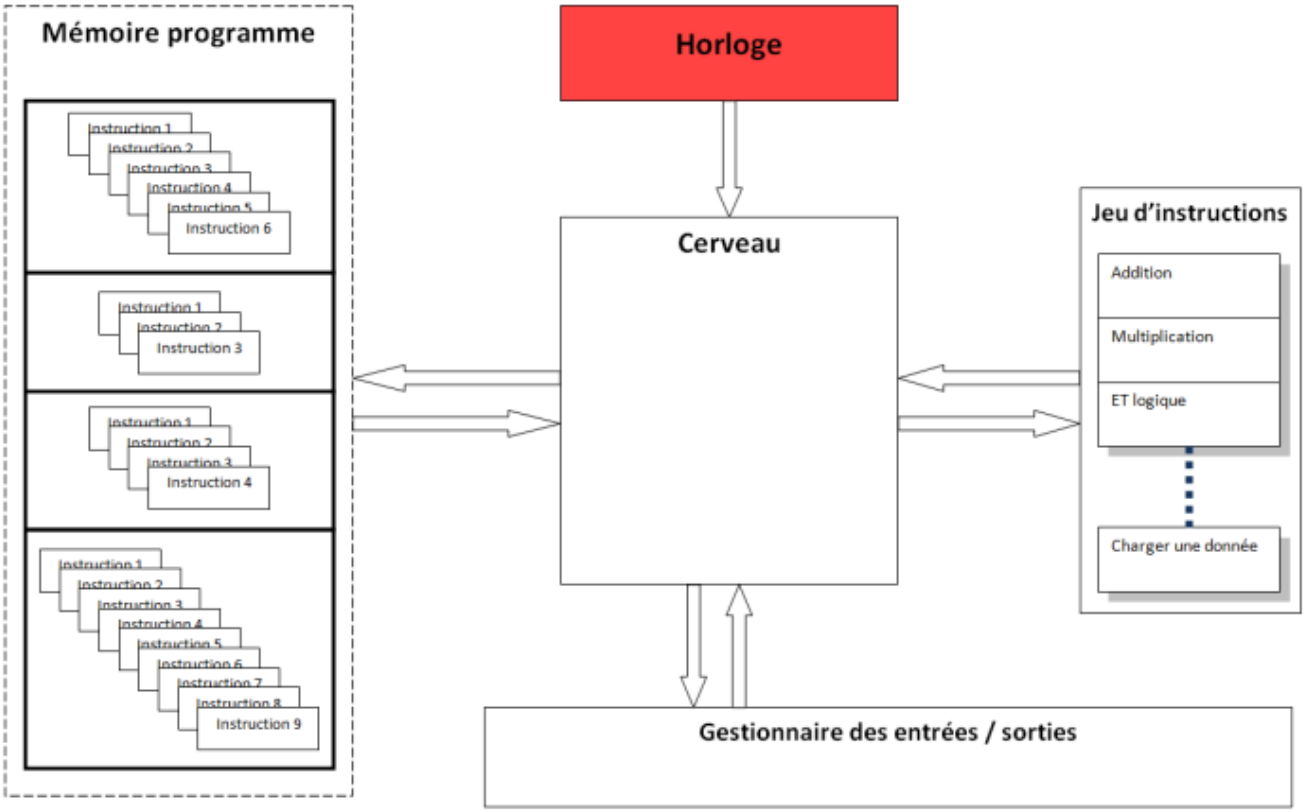
Waoou ! Et ça va vite tout ça ?

Extrêmement vite ! Cela dit, tout dépend de sa vitesse d'exécution…

La vitesse d'exécution

Le microcontrôleur est capable de faire un très grand nombre d'opérations par seconde. Ce nombre est défini par sa vitesse, entre autre. Sur la carte Arduino Duemilanove ou Uno, il y a un composant, que l'on appel un **quartz**, qui va définir à quelle vitesse va aller le microcontrôleur. Ce quartz permet de **cadencer** le microcontrôleur. C'est en fait

une **horloge** qui permet au microcontrôleur de se repérer. A chaque top de l'horloge, le microcontrôleur va faire quelque chose. Ce quelque chose peut, par exemple, être l'exécution d'une instruction, ou une lecture en mémoire. Cependant, chaque action ne dure pas qu'un seul top d'horloge. Suivant l'action réalisée, cela peut prendre plus ou moins de temps, enfin de top d'horloge.



La carte Arduino atteint au moins le million d'instructions par secondes ! Cela peut paraître énorme, mais comme je le disais, si il y a des instructions qui prennent beaucoup de temps, eh bien il se peut qu'elle n'exécute qu'une centaine d'instruction en une seconde. Tout dépend du temps pris par une instruction à être exécuté. Certaines opérations sont aussi parallélisées. Par exemple, le microcontrôleur peut faire une addition d'un côté pour une variable et en même temps il va mesurer le nombre de coup d'horloge pour faire s'incrémenter un compteur pour gérer un timer. Ces opération sont **réellement** faite en parrallèle, ce n'est pas un faux multi-tâche comme sur un ordinateur. Ici les deux registres travaille en même temps. Le nombre de la fin ? 62.5 nanoSecondes. C'est le temps qu'il faut au microcontrôleur d'Arduino pour faire une instruction la plus simple possible. (En prenant en compte l'Arduino Uno et son quartz à 16MHz).

A présent, vous savez utiliser les sorties du micro-contrôleur, nous allons donc pouvoir passer aux choses sérieuses et faire clignoter notre LED !

[Arduino 202] Introduire le temps

C'est bien beau d'allumer une LED, mais si elle ne fait rien d'autre, ce n'est pas très utile. Autant la brancher directement sur une pile (avec une résistance tout de même !). Alors voyons comment rendre intéressante cette LED en la faisant clignoter ! Ce que

ne sait pas faire une pile… Pour cela il va nous falloir introduire la notion de temps. Et bien devinez quoi ? Il existe une fonction toute prête là encore ! Je ne vous en dis pas plus, passons à la pratique !

Comment faire ?

Trouver la commande…

Je vous laisse cherche un peu par vous même, cela vous entrainera ! :Pirate: … Pour ceux qui ont fait l'effort de chercher et n'ont pas trouvé (à cause de l'anglais ?), je vous donne la fonction qui va bien : On va utiliser : **delay()** Petite description de la fonction, elle va servir à mettre en pause le programme pendant un temps prédéterminé.

Utiliser la commande

La fonction admet un paramètre qui est le temps pendant lequel on veut mettre en pause le programme. Ce temps doit être donné en millisecondes. C'est-à-dire que si vous voulez arrêter le programme pendant 1 seconde, il va falloir donner à la fonction ce même temps, écrit en millisecondes, soit 1000ms. Le code est simple à utiliser, il est le suivant :

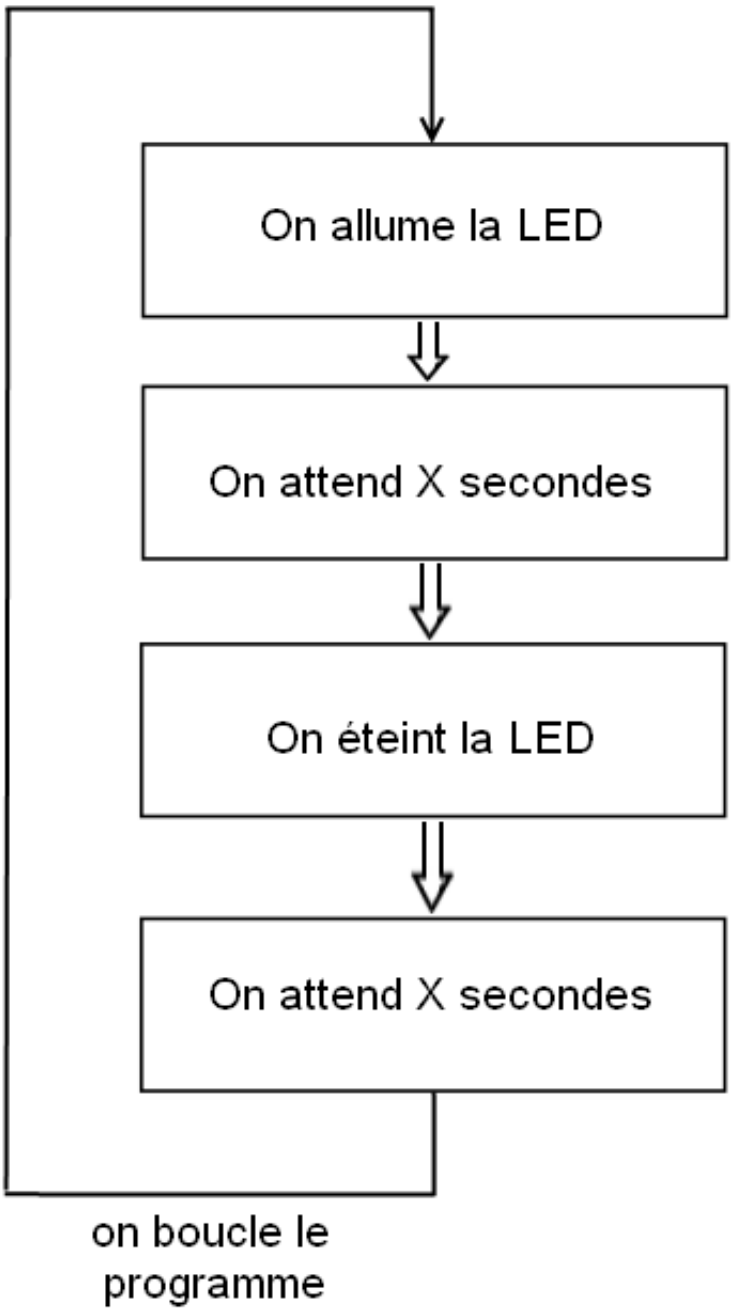
```
1 // on fait une pause du programme pendant 1000ms, soit 1 seconde
2 delay(1000);
```

Rien de plus simple donc. Pour 20 secondes de pause, il aurait fallu écrire :

```
1 // on fait une pause du programme pendant 20000ms, soit 20 secondes
2 delay(20000);
```

Mettre en pratique : faire clignoter une LED

Du coup, si on veut faire clignoter notre LED, il va falloir utiliser cette fonction. Voyons un peu le schéma de principe du clignotement d'une LED :



Vous le voyez, la LED s'allume. Puis, on fait intervenir la fonction **delay()**, qui va mettre le programme en pause pendant un certain temps. Ensuite, on éteint la LED. On met en pause le programme. Puis on revient au début du programme. On recommence et ainsi de suite. C'est cette somme de commande, qui forme le processus qui fait clignoter la LED.

Dorénavant, prenez l'habitude de faire ce genre de schéma lorsque vous faites un programme. Cela aide grandement la réflexion, croyez moi ! C'est le principe de perdre du temps pour en gagner. Autrement dit : l'**organisation** !

Maintenant, il faut que l'on traduise ce schéma, portant le nom d'**organigramme**, en code. Il suffit pour cela de remplacer les phrases dans chaque cadre par une ligne de code. Par exemple, "on allume la LED", va être traduis par l'instruction que l'on a vue dans le chapitre précédent :

```
digitalWrite(led_rouge, LOW);   // allume la LED
```

Ensuite, on traduit le cadre suivant, ce qui donne :

```
// fait une pause de 1 seconde (= 1000ms)
delay(1000);
```

Puis, on traduit la ligne suivante :

```
// éteint la LED
digitalWrite(led_rouge, HIGH);
```

Enfin, la dernière ligne est identique à la deuxième, soit :

```
// fait une pause de 1 seconde
delay(1000);
```

On se retrouve avec le code suivant :

```
// allume la LED
digitalWrite(led_rouge, LOW);
// fait une pause de 1 seconde
delay(1000);
// éteint la LED
digitalWrite(led_rouge, HIGH);
// fait une pause de 1 seconde
delay(1000);
```

La fonction qui va boucler à l'infini le code précédent est la fonction **loop()**. On inscrit donc le code précédent dans cette fonction :

```
void loop()
{
    // allume la LED
    digitalWrite(led_rouge, LOW);
    // fait une pause de 1 seconde
    delay(1000);
    // éteint la LED
    digitalWrite(led_rouge, HIGH);
    // fait une pause de 1 seconde
    delay(1000);
}
```

Et on n'oublie pas de définir la broche utilisée par la LED, ainsi que d'initialiser cette broche en tant que sortie. Cette fois, le code est terminé !

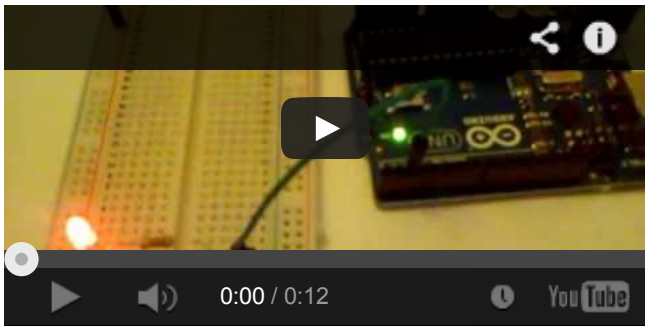
```
//définition de la broche 2 de la carte en tant que variable
const int led_rouge = 2;

//fonction d'initialisation de la carte
void setup()
{
    //initialisation de la broche 2 comme étant une sortie
    pinMode(led_rouge, OUTPUT);
}

void loop()
{
```

```
    // allume la LED
    digitalWrite(led_rouge, LOW);
    // fait une pause de 1 seconde
    delay(1000);
    // éteint la LED
    digitalWrite(led_rouge, HIGH);
    // fait une pause de 1 seconde
    delay(1000);
}
```

Vous n'avez plus qu'à charger le code dans la carte et admirer ~~mon~~ votre travail ! La LED clignote ! Libre à vous de changer le temps de clignotement : vous pouvez par exemple éteindre la LED pendant 40ms et l'allumer pendant 600ms :

```
//définition de la broche 2 de la carte en tant que variable
const int led_rouge = 2;

//fonction d'initialisation de la carte
void setup()
{
    //initialisation de la broche 2 comme étant une sortie
    pinMode(led_rouge, OUTPUT);
}

void loop()
{
    // allume la LED
    digitalWrite(led_rouge, LOW);
    // fait une pause de 600 ms
    delay(600);
    // éteint la LED
    digitalWrite(led_rouge, HIGH);
    // fait une pause de 40 ms
    delay(40);
}
```

Et Hop, une petite vidéo d'illustration !

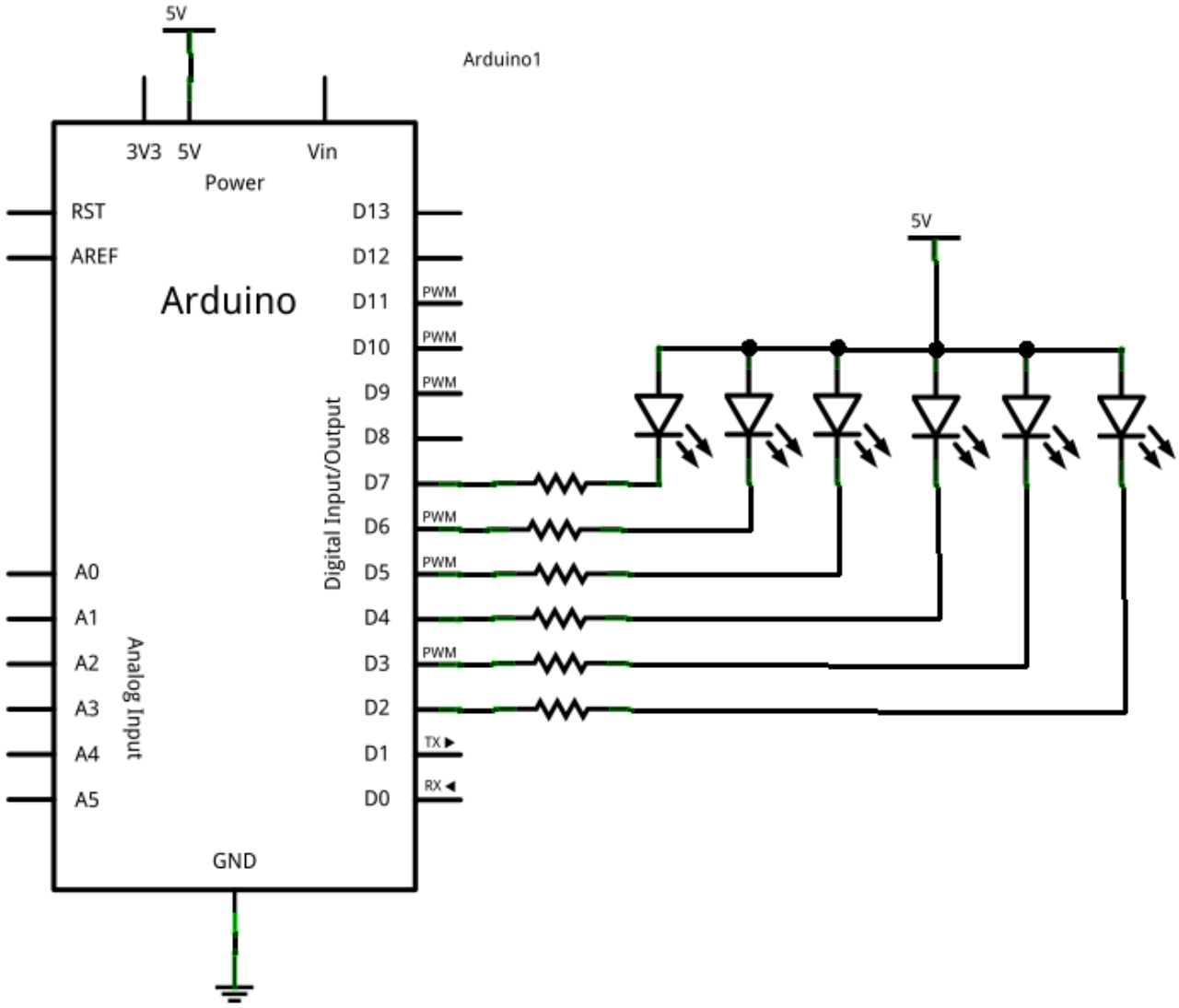


Faire clignoter un groupe de LED

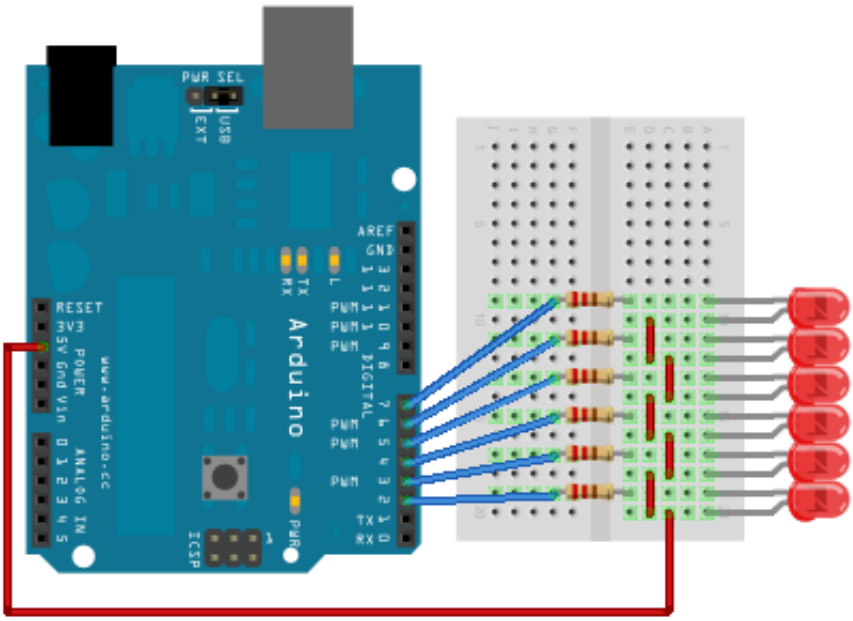
Vous avouerez facilement que ce n'était pas bien difficile d'arriver jusque-là. Alors, à présent, accentuons la difficulté. Notre but : faire clignoter un groupe de LED.

Le matériel et les schémas

Ce groupe de LED sera composé de six LED, nommées L1, L2, L3, L4, L5 et L6. Vous aurez par conséquent besoin d'un nombre semblable de résistances. Le schéma de la réalisation :

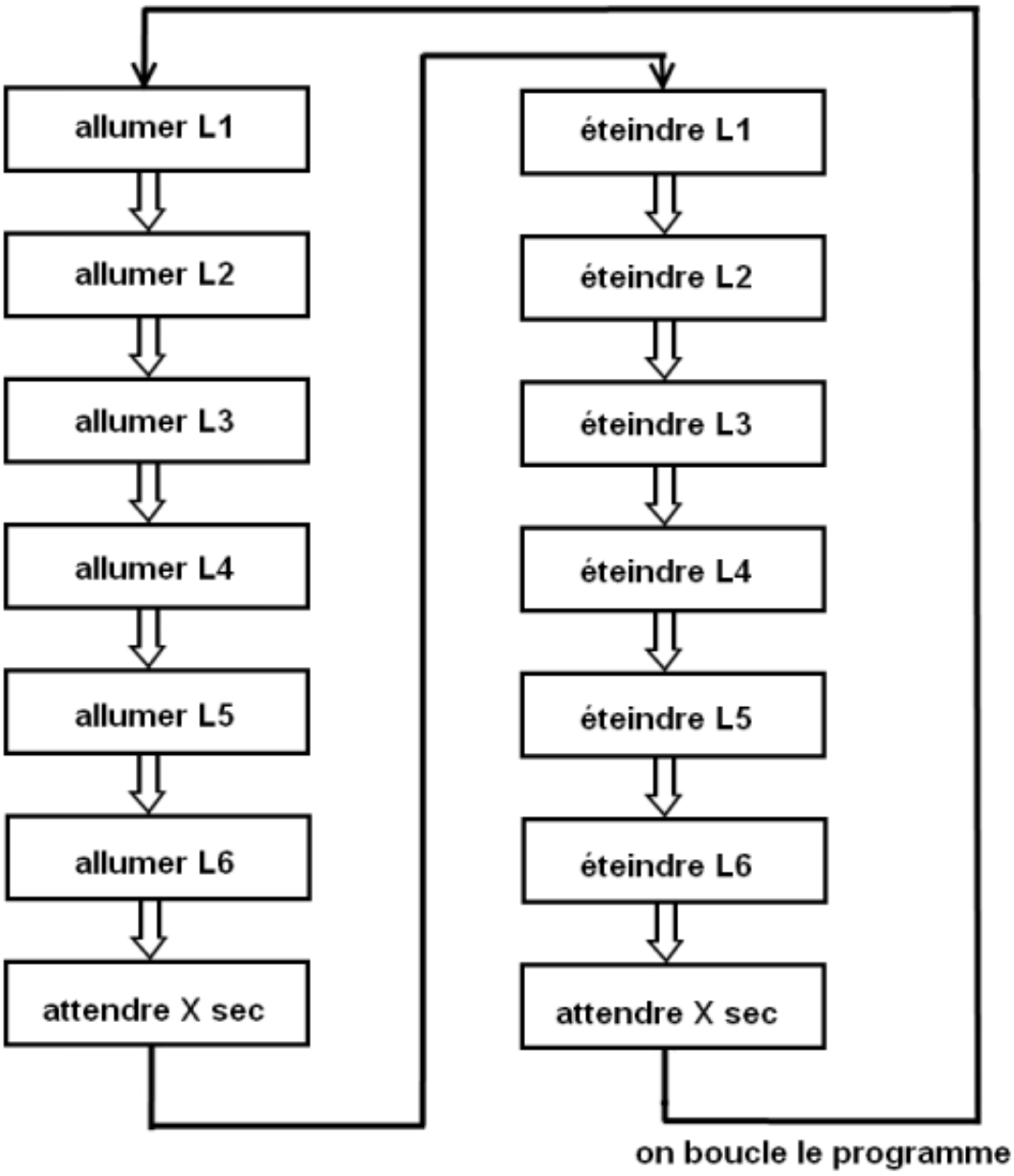


La photo de la réalisation :



Le programme

Le programme est un peu plus long que le précédent, car il ne s'agit plus d'allumer 1 seule LED, mais 6 ! Voilà l'organigramme que va suivre notre programme :



Cet organigramme n'est pas très beau, mais il a le mérite d'être assez lisible. Nous allons essayer de le suivre pour créer notre programme. Traduction des six premières instructions :

```
digitalWrite(L1, LOW); //notez que le nom de la broche à changé
digitalWrite(L2, LOW); //et ce pour toutes les LED connectées
digitalWrite(L3, LOW); //au micro-controleur
digitalWrite(L4, LOW);
digitalWrite(L5, LOW);
digitalWrite(L6, LOW);
```

Ensuite, on attend 1,5 seconde :

```
delay(1500);
```

Puis on traduis les six autres instructions :

```
digitalWrite(L1, HIGH); //on éteint les LED
digitalWrite(L2, HIGH);
```

```
digitalWrite(L3, HIGH);
digitalWrite(L4, HIGH);
digitalWrite(L5, HIGH);
digitalWrite(L6, HIGH);
```

Enfin, la dernière ligne de code, disons que nous attendrons 4,32 secondes :

```
delay(4320);
```

Tous ces bouts de code sont à mettre à la suite et dans la fonction **loop()** pour qu'ils se répètent.

```
void loop()
{
    digitalWrite(L1, LOW);  //allumer les LED
    digitalWrite(L2, LOW);
    digitalWrite(L3, LOW);
    digitalWrite(L4, LOW);
    digitalWrite(L5, LOW);
    digitalWrite(L6, LOW);

    delay(1500);              //attente du programme de 1,5 secondes

    digitalWrite(L1, HIGH);   //on éteint les LED
    digitalWrite(L2, HIGH);
    digitalWrite(L3, HIGH);
    digitalWrite(L4, HIGH);
    digitalWrite(L5, HIGH);
    digitalWrite(L6, HIGH);

    delay(4320);              //attente du programme de 4,32 secondes
}
```

Je l'ai mentionné dans un de mes commentaires entre les lignes du programme, les noms attribués aux broches sont à changer. En effet, car si on définit des noms de variables identiques, le compilateur n'aimera pas ça et vous affichera une erreur. En plus, le micro-contrôleur ne pourrait pas exécuter le programme car il ne saurait pas quelle broche mettre à l'état HAUT ou BAS. Pour définir les broches, on fait la même chose qu'à notre premier programme :

```
const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1
const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2
const int L3 = 4; // ...
const int L4 = 5;
const int L5 = 6;
const int L6 = 7;
```

Maintenant que les broches utilisées sont définies, il faut dire si ce sont des entrées ou des sorties :

```
pinMode(L1, OUTPUT); //L1 est une broche de sortie
pinMode(L2, OUTPUT); //L2 est une broche de sortie
pinMode(L3, OUTPUT); // ...
pinMode(L4, OUTPUT);
pinMode(L5, OUTPUT);
pinMode(L6, OUTPUT);
```

Le programme final

Il n'est certes pas très beau, mais il fonctionne :

```
const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1
const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2
const int L3 = 4; // ...
const int L4 = 5;
const int L5 = 6;
const int L6 = 7;

void setup()
{
    pinMode(L1, OUTPUT); //L1 est une broche de sortie
    pinMode(L2, OUTPUT); //L2 est une broche de sortie
    pinMode(L3, OUTPUT); // ...
    pinMode(L4, OUTPUT);
    pinMode(L5, OUTPUT);
    pinMode(L6, OUTPUT);
}

void loop()
{
    //allumer les LED
    digitalWrite(L1, LOW);
    digitalWrite(L2, LOW);
    digitalWrite(L3, LOW);
    digitalWrite(L4, LOW);
    digitalWrite(L5, LOW);
    digitalWrite(L6, LOW);

    //attente du programme de 1,5 secondes
    delay(1500);

    //on éteint les LED
    digitalWrite(L1, HIGH);
    digitalWrite(L2, HIGH);
    digitalWrite(L3, HIGH);
    digitalWrite(L4, HIGH);
    digitalWrite(L5, HIGH);
    digitalWrite(L6, HIGH);

    //attente du programme de 4,32 secondes
    delay(4320);
}
```

Voilà, vous avez en votre possession un magnifique clignotant, que vous pouvez attacher à votre vélo ! 😀

> Une question me chiffonne. Doit-on toujours écrire l'état d'une sortie, ou peut-on faire plus simple ?

Tu soulèves un point intéressant. Si je comprends bien, tu te demandes comment faire pour remplacer l'intérieur de la fonction **loop()**? C'est vrai que c'est très lourd à écrire et à lire ! Il faut en effet s'occuper de définir l'état de chaque LED. C'est rébarbatif, surtout si vous en aviez mis autant qu'il y a de broches disponibles sur la carte ! Il y a une solution pour faire ce que tu dis. Nous allons la voir dans quelques chapitres, ne sois

pas impatient ! En attendant, voici une vidéo d'illustration du clignotement :

Réaliser un chenillard

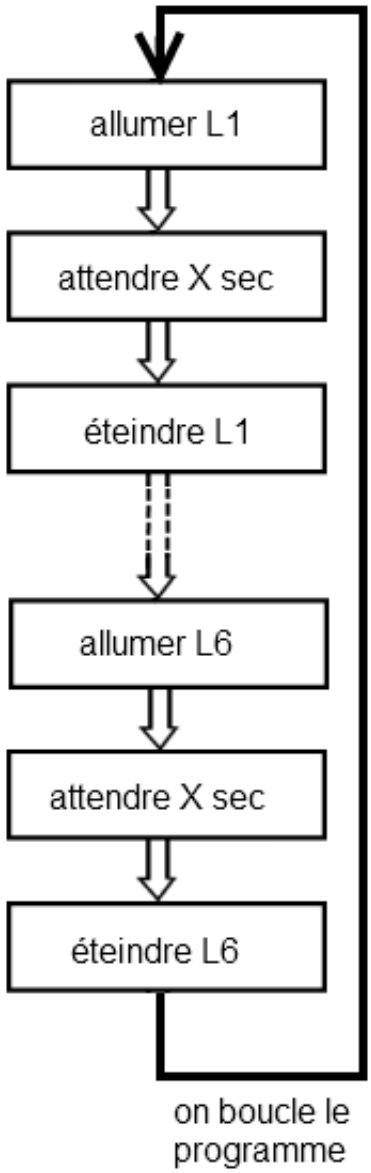
Le but du programme

Le but du programme que nous allons créer va consister à réaliser un chenillard. Pour ceux qui ne savent pas ce qu'est un chenillard, je vous ai préparé une petite image .gif animée :

Comme on dit souvent, une image vaut mieux qu'un long discours ! Voilà donc ce qu'est un chenillard. Chaque LED s'allume alternativement et dans l'ordre chronologique. De la gauche vers la droite ou l'inverse, c'est au choix.

Organigramme

Comme j'en ai marre de faire des dessins avec paint.net, je vous laisse réfléchir tout seul comme des grands à l'organigramme du programme. … Bon, aller, le voilà cet organigramme ! Attention, il n'est pas complet, mais si vous avez compris le principe, le compléter ne vous posera pas de problèmes :



A vous de jouer !

Le programme

Normalement, sa conception ne devrait pas vous poser de problèmes. Il suffit en effet de récupérer le code du programme précédent (“allumer un groupe de LED”) et de le modifier en fonction de notre besoin. Ce code, je vous le donne, avec les commentaires qui vont bien :

```
1  const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1
2  const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2
3  const int L3 = 4; // ...
4  const int L4 = 5;
5  const int L5 = 6;
6  const int L6 = 7;
7
8  void setup()
9  {
10     pinMode(L1, OUTPUT); //L1 est une broche de sortie
11     pinMode(L2, OUTPUT); //L2 est une broche de sortie
12     pinMode(L3, OUTPUT); // ...
```

```
13      pinMode(L4, OUTPUT);
14      pinMode(L5, OUTPUT);
15      pinMode(L6, OUTPUT);
16  }
17
18  // on change simplement l'intérieur de la boucle pour atteindre notre objectif
19
20  void loop() //la fonction loop() exécute le code qui suit en le répétant en boucle
21  {
22      digitalWrite(L1, LOW);   //allumer L1
23      delay(1000);             //attendre 1 seconde
24      digitalWrite(L1, HIGH);  //on éteint L1
25      digitalWrite(L2, LOW);   //on allume L2 en même temps que l'on éteint L1
26      delay(1000);             //on attend 1 seconde
27      digitalWrite(L2, HIGH);  //on éteint L2 et
28      digitalWrite(L3, LOW);   //on allume immédiatement L3
29      delay(1000);             // ...
30      digitalWrite(L3, HIGH);
31      digitalWrite(L4, LOW);
32      delay(1000);
33      digitalWrite(L4, HIGH);
34      digitalWrite(L5, LOW);
35      delay(1000);
36      digitalWrite(L5, HIGH);
37      digitalWrite(L6, LOW);
38      delay(1000);
39      digitalWrite(L6, HIGH);
40  }
```

Vous le voyez, ce code est très lourd et n'est pas pratique. Nous verrons plus loin comment faire en sorte de l'alléger. Mais avant cela, un TP arrive… Au fait, voici un exemple de ce que vous pouvez obtenir !

Fonction millis()

Nous allons terminer ce chapitre par un point qui peutêtre utile, notamment dans certaines situations où l'on veut ne pas arrêter le programme. En effet, si on veut faire clignoter une LED sans arrêter l'exécution du programme, on ne peut pas utiliser la fonction `delay()` qui met en pause le programme durant le temps défini.

Les limites de la fonction delay()

Vous avez probablement remarqué, lorsque vous utilisez la fonction "delay()" tout notre programme s'arrête le temps d'attendre. Dans certains cas ce n'est pas un problème mais dans certains cas ça peut être plus gênant. Imaginons, vous êtes en train de faire

avancer un robot. Vous mettez vos moteurs à une vitesse moyenne, tranquille, jusqu'à ce qu'un petit bouton sur l'avant soit appuyé (il clic lorsqu'on touche un mur par exemple). Pendant ce temps-là, vous décidez de faire des signaux en faisant clignoter vos LED. Pour faire un joli clignotement, vous allumez une LED rouge pendant une seconde puis l'éteignez pendant une autre seconde. Voilà par exemple ce qu'on pourrait faire comme code

```
1  void setup()
2  {
3      pinMode(moteur, OUTPUT);
4      pinMode(led, OUTPUT);
5      pinMode(bouton, INPUT);
6      //on met le moteur en marche (en admettant qu'il soit en marche à HIGH)
7      digitalWrite(moteur, HIGH);
8      //on allume la LED
9      digitalWrite(led, LOW);
10 }
11
12 void loop()
13 {
14     //si le bouton est cliqué (on rentre dans un mur)
15     if(digitalRead(bouton)==HIGH)
16     {
17         //on arrête le moteur
18         digitalWrite(moteur, LOW);
19     }
20     else //sinon on clignote
21     {
22         digitalWrite(led, HIGH);
23         delay(1000);
24         digitalWrite(led, LOW);
25         delay(1000);
26     }
27 }
```

Attention ce code n'est pas du tout rigoureux voire faux dans son écriture, il sert juste à comprendre le principe !

Maintenant imaginez. Vous roulez, tester que le bouton n'est pas appuyé, donc faites clignoter les LED (cas du *else*). Le temps que vous fassiez l'affichage en entier s'écoule 2 longues secondes ! Le robot a pu pendant cette éternité se prendre le mur en pleine poire et les moteurs continuent à avancer tête baissée jusqu'à fumer ! Ce n'est pas bon du tout ! Voici pourquoi la fonction millis() peut nous sauver.

Découvrons et utilisons millis()

Tout d'abord, quelques précisions à son sujet, avant d'aller s'en servir. A l'intérieur du cœur de la carte Arduino se trouve un chronomètre. Ce chrono mesure l'écoulement du temps depuis le lancement de l'application. Sa granularité (la précision de son temps) est la milliseconde. La fonction millis() nous sert à savoir quelle est la valeur courante de ce compteur. Attention, comme ce compteur est capable de mesurer une durée allant jusqu'à 50 jours, la valeur retournée doit être stockée dans une variable de type "long".

C'est bien gentil mais concrètement on l'utilise comment ?

Et bien c'est très simple. On sait maintenant "lire l'heure". Maintenant, au lieu de dire "allume-toi pendant une seconde et ne fais surtout rien pendant ce temps", on va faire un truc du genre "Allume-toi, fais tes petites affaires, vérifie l'heure de temps en temps et si une seconde est écoulée, alors réagis !". Voici le code précédent transformé selon la nouvelle philosophie :

```
1  long temps; //variable qui stocke la mesure du temps
2  boolean etat_led;
3
4  void setup()
5  {
6      pinMode(moteur, OUTPUT);
7      pinMode(led, OUTPUT);
8      pinMode(bouton, INPUT);
9      //on met le moteur en marche
10     digitalWrite(moteur, HIGH);
11     //par défaut la LED sera éteinte
12     etat_led = 0;
13     //on éteint la LED
14     digitalWrite(led, etat_led);
15 }
16
17 void loop()
18 {
19     //si le bouton est cliqué (on rentre dans un mur)
20     if(digitalRead(bouton)==HIGH)
21     {
22         //on arrête le moteur
23         digitalWrite(moteur, LOW);
24     }
25     else //sinon on clignote
26     {
27         //on compare l'ancienne valeur du temps et la valeur sauvée
28         //si la comparaison (l'un moins l'autre) dépasse 1000...
29         //...cela signifie qu'au moins une seconde s'est écoulée
30         if((millis() - temps) > 1000)
31         {
32             etat_led = !etat_led; //on inverse l'état de la LED
33             digitalWrite(led, etat_led); //on allume ou éteint
34             temps = millis(); //on stocke la nouvelle heure
35         }
36     }
37 }
```

Et voilà, grâce à cette astuce plus de fonction bloquante. L'état du bouton est vérifié très fréquemment ce qui permet de s'assurer que si jamais on rentre dans un mur, on coupe les moteurs très vite. Dans ce code, tout s'effectue de manière fréquente. En effet, on ne reste jamais bloqué à attendre que le temps passe. A la place, on avance dans le programme et test souvent la valeur du chronomètre. Si cette valeur est de 1000 itérations supérieures à la dernière valeur mesurée, alors cela signifie qu'une seconde est passée.

Attention, au "if" de la ligne 25 ne faites surtout pas "millis() – temp == 1000". Cela signifierait que vous voulez vérifier que 1000 millisecondes EXACTEMENT se sont écoulées, ce qui est très peu probable (vous pourrez plus probablement mesurer plus ou moins mais rarement exactement)

Maintenant que vous savez maîtriser le temps, vos programmes/animations vont pouvoir posséder un peu plus de “vie” en faisant des pauses, des motifs, etc. Impressionnez-moi !

[Arduino 203] [TP] Feux de signalisation routière

Vous voilà arrivé pour votre premier TP, que vous ferez seul ! 😈 Je vous aiderai quand même un peu. Le but de ce TP va être de réaliser un feu de signalisation routière. Je vous donne en détail tout ce qu’il vous faut pour mener à bien cet objectif.

Préparation

Ce dont nous avons besoin pour réaliser ces feux.

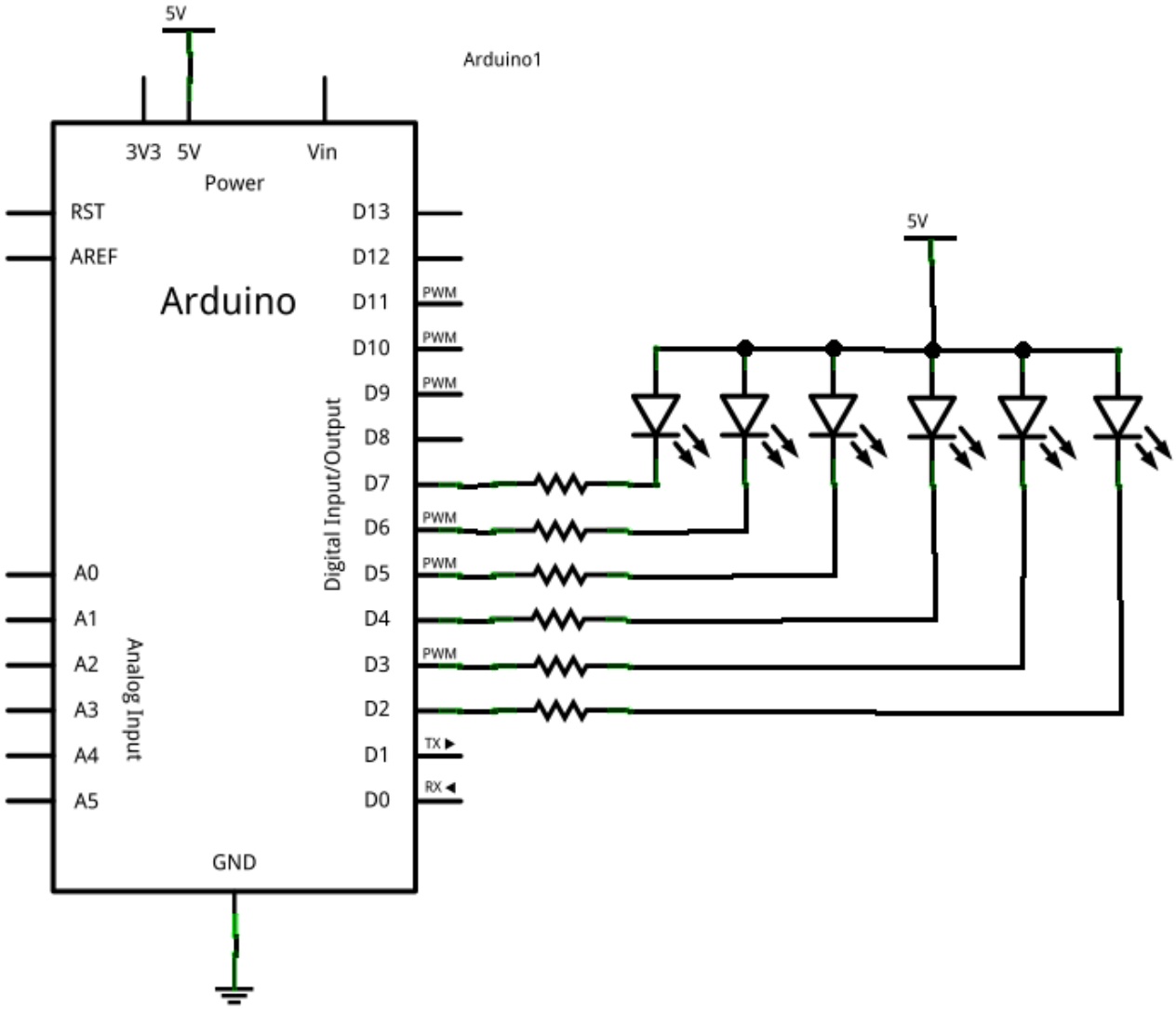
Le matériel

Le matériel est la base de notre besoin. On a déjà utilisé 6 LED et résistances, mais elles étaient pour moi en l’occurrence toutes rouges. Pour faire un feu routier, il va nous falloir 6 LED, mais dont les couleurs ne sont plus les mêmes.

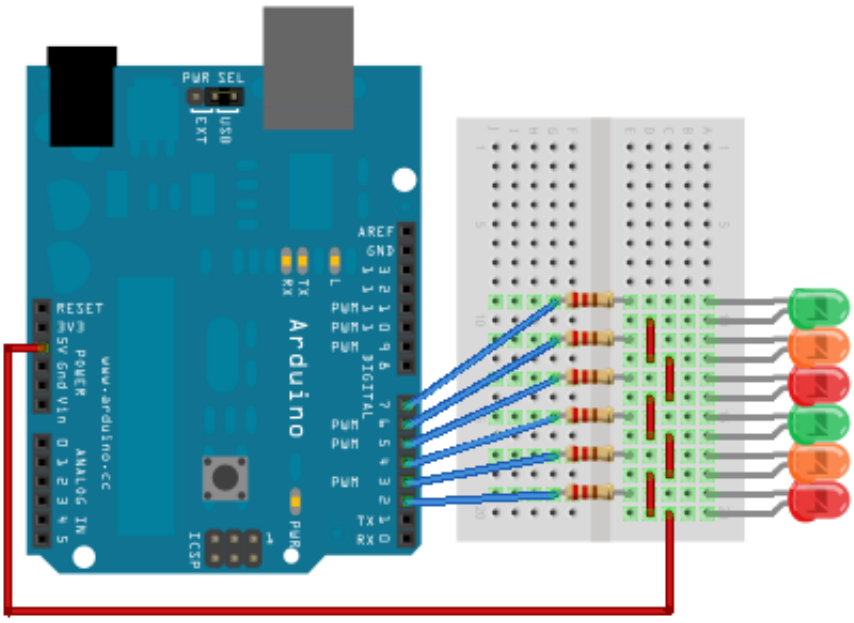
- LED : un nombre de 6, dont 2 **rouges**, 2 **jaune**/**orange** et 2 **vertes**
- Résistors : 6 également, de la même valeur que ceux que vous avez utilisés.
- Arduino : une carte Arduino évidemment !

Le schéma

C’est le même que pour le montage précédent, seul la couleur des LED change, comme ceci :



Vous n'avez donc plus qu'à reprendre le dernier montage et changer la couleur de 4 LED, pour obtenir ceci :

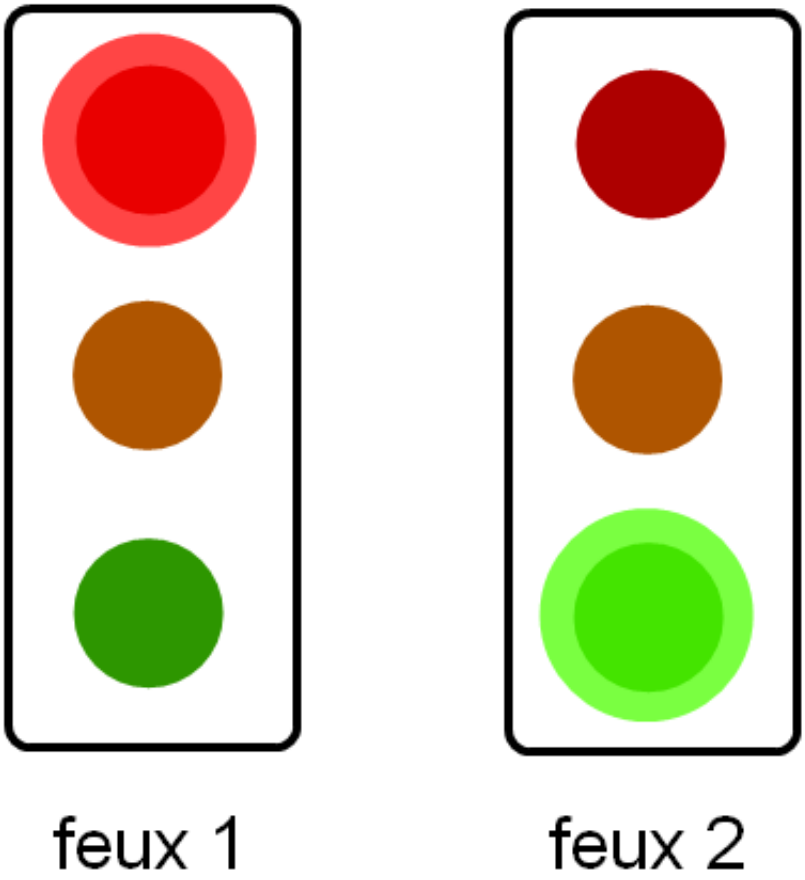


N’oubliez pas de tester votre matériel en chargeant un programme qui fonctionne ! Cela évite de s’acharner à faire un nouveau programme qui ne fonctionne pas à cause d’un matériel défectueux. On est jamais sur de rien, croyez-moi ! 😉

Énoncé de l’exercice

Le but

Je l’ai dit, c’est de réaliser des feux de signalisation. Alors, vu le nombre de LED, vous vous doutez bien qu’il faut réaliser 2 feux. Ces feux devront être synchronisés. Là encore, je vous ai préparé une belle image animée :



Le temps de la séquence

Vous allez mettre un délai de 3 secondes entre le feu vert et le feu orange. Un délai de 1 seconde entre le feu orange et le feu rouge. Et un délai de 3 secondes entre le feu rouge et le feu vert.

Par où commencer ?

D’abord, vous devez faire l’organigramme. Oui je ne vous le donne pas ! Ensuite, vous commencez un nouveau programme. Dans ce programme, vous devez définir quelles sont les broches du micro-contrôleur que vous utilisez. Puis définir si ce sont des entrées, des sorties, ou s’il y a des deux. Pour terminer, vous allez faire le programme

complet dans la fonction qui réalise une boucle.

C'est parti !

Allez, c'est parti ! A vous de m'épater. Vous avez théoriquement toutes les bases nécessaires pour réaliser ce TP. En plus on a presque déjà tout fait. Mince ,j'en ai trop dit… Pendant ce temps, moi je vais me faire une raclette. Et voici un résultat possible :

Correction !

Fini !

Vous avez fini ? Votre code ne fonctionne pas, mais vous avez eu beau cherché pourquoi, vous n'avez pas trouvé ? Très bien. Dans ce cas, vous pouvez lire la correction. Ceux qui n'ont pas cherché ne sont pas les bienvenus ici !

L'organigramme

Cette fois, l'organigramme a changé de forme, c'est une liste. Comment le lire ? De haut en bas ! Le premier élément du programme commence après le début, le deuxième élément, après le premier, etc.

- **DEBUT**
- //première partie du programme, on s'occupe principalement du deuxième feu
- Allumer led_rouge_feux_1
- Allumer led_verte_feux_2
- Attendre 3 secondes
- Éteindre led_verte_feux_2
- Allumer led_jaune_feux_2
- Attendre 1 seconde
- Éteindre led_jaune_feux_2
- Allumer led_rouge_feux_2
- /*deuxième partie du programme, pour l'instant : led_rouge_feux_1 et led_rouge_feux_2 sont allumées; on éteint donc la led_rouge_feux_1 pour allumer la led_verte_feux_1*/
- Attendre 3 secondes
- Éteindre led_rouge_feux_1
- Allumer led_verte_feux_1
- Attendre 3 secondes

- Éteindre led_verte_feux_1
- Allumer led_jaune_feux_1
- Attendre 1 seconde
- Éteindre led_jaune_feux_1
- Allumer led_rouge_feux_1
- **FIN**

Voilà donc ce qu'il faut suivre pour faire le programme. Si vous avez trouvé comme ceci, c'est très bien, sinon il faut s'entraîner car c'est très important d'organiser son code et en plus cela permet d'éviter certaines erreurs !

La correction, enfin !

Voilà le moment que vous attendez tous : la correction ! Alors, je préviens tout de suite, le code que je vais vous montrer n'est pas absolu, on peut le faire de différentes manières

La fonction setup

Normalement ici aucune difficulté, on va nommer les broches, puis les placer en sortie et les mettre dans leur état de départ.

```
1  //définition des broches
2  const int led_rouge_feux_1 = 2;
3  const int led_jaune_feux_1 = 3;
4  const int led_verte_feux_1 = 4;
5  const int led_rouge_feux_2 = 5;
6  const int led_jaune_feux_2 = 6;
7  const int led_verte_feux_2 = 7;
8
9  void setup()
10 {
11     //initialisation en sortie de toutes les broches
12     pinMode(led_rouge_feux_1, OUTPUT);
13     pinMode(led_jaune_feux_1, OUTPUT);
14     pinMode(led_verte_feux_1, OUTPUT);
15     pinMode(led_rouge_feux_2, OUTPUT);
16     pinMode(led_jaune_feux_2, OUTPUT);
17     pinMode(led_verte_feux_2, OUTPUT);
18
19     //on initialise toutes les LED éteintes au début du programme (sauf les deux feu
20     digitalWrite(led_rouge_feux_1, LOW);
21     digitalWrite(led_jaune_feux_1, HIGH);
22     digitalWrite(led_verte_feux_1, HIGH);
23     digitalWrite(led_rouge_feux_2, LOW);
24     digitalWrite(led_jaune_feux_2, HIGH);
25     digitalWrite(led_verte_feux_2, HIGH);
26 }
```

Vous remarquerez l'utilité d'avoir des variables bien nommées.

Le code principal

Si vous êtes bien organisé, vous ne devriez pas avoir de problème ici non plus! Point trop de paroles, la solution arrive

```
void loop()
{
    // première séquence
    digitalWrite(led_rouge_feux_1, HIGH);
    digitalWrite(led_verte_feux_1, LOW);

    delay(3000);

    // deuxième séquence
    digitalWrite(led_verte_feux_1, HIGH);
    digitalWrite(led_jaune_feux_1, LOW);

    delay(1000);

    // troisième séquence
    digitalWrite(led_jaune_feux_1, HIGH);
    digitalWrite(led_rouge_feux_1, LOW);

    delay(1000);

    /* ---------- deuxième partie du programme, on s'occupe du feux numéro 2 -------

    // première séquence
    digitalWrite(led_rouge_feux_2, HIGH);
    digitalWrite(led_verte_feux_2, LOW);

    delay(3000);

    // deuxième séquence
    digitalWrite(led_verte_feux_2, HIGH);
    digitalWrite(led_jaune_feux_2, LOW);

    delay(1000);

    // deuxième séquence
    digitalWrite(led_jaune_feux_2, HIGH);
    digitalWrite(led_rouge_feux_2, LOW);

    delay(1000);

    /* --------------- le programme va reboucler et revenir au début ---------------
}
```

Si ça marche, tant mieux, sinon référez vous à la résolution des problèmes en annexe du cours. Ce TP est donc terminé, vous pouvez modifier le code pour par exemple changer les temps entre chaque séquence, ou bien même modifier les séquences elles-mêmes, …

Bon, c'était un TP gentillet. L'intérêt est seulement de vous faire pratiquer pour vous "enfoncer dans le crâne" ce que l'on a vu jusqu'à présent.

[Arduino 204] Un simple bouton

Dans cette partie, vous allez pouvoir interagir de manière simple avec votre carte. A la fin de ce chapitre, vous serez capable d'utiliser des boutons ou des interrupteurs pour interagir avec votre programme.

Qu'est-ce qu'un bouton ?

Derrière ce titre trivial se cache un composant de base très utile, possédant de nombreux détails que vous ignorez peut-être. Commençons donc dès maintenant l'autopsie de ce dernier.

Mécanique du bouton

Vous le savez sûrement déjà, un bouton n'est jamais qu'un fil qui est connecté ou non selon sa position. En pratique, on en repère plusieurs, qui diffèrent selon leur taille, leurs caractéristiques électriques, les positions mécaniques possibles, etc.

Le bouton poussoir normalement ouvert (NO)

Dans cette partie du tutoriel, nous allons utiliser ce type de boutons poussoirs (ou BP). Ces derniers ont deux positions :

- - **Relâché** : le courant ne passe pas, le circuit est déconnecté ; on dit que le circuit est "**ouvert**".
- - **Appuyé** : le courant passe, on dit que le circuit est **fermé**.

Retenez bien ces mots de vocabulaire !

Habituellement le bouton poussoir a deux broches, mais en général ils en ont 4 reliées deux à deux.

Le bouton poussoir normalement fermé (NF)

Ce type de bouton est l'opposé du type précédent, c'est-à-dire que lorsque le bouton est relâché, il laisse passer le courant. Et inversement :

- - **Relâché** : le courant passe, le circuit est connecté ; on dit que le circuit est "**fermé**".
- - **Appuyé** : le courant ne passe pas, on dit que le circuit est **ouvert**.

Les interrupteurs

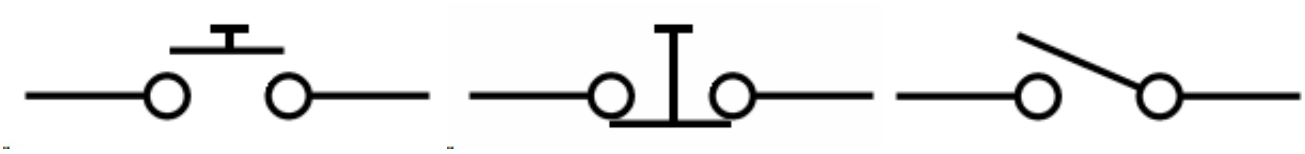
A la différence d'un bouton poussoir, l'interrupteur agit comme une bascule. Un appui ferme le circuit et il faut un second appui pour l'ouvrir de nouveau. Il possède donc des états stables (ouvert ou fermé). On dit qu'un interrupteur est **bistable**. Vous en rencontrez tous les jours lorsque vous allumez la lumière .

L'électronique du bouton

Symbole

Le BP et l'interrupteur ne possèdent pas le même symbole pour les schémas électroniques. Pour le premier, il est représenté par une barre qui doit venir faire contact pour fermer le circuit ou défaire le contact pour ouvrir le circuit. Le second est représenté par un fil qui ouvre un circuit et qui peut bouger pour le fermer. Voici leurs

symboles, il est important de s'en rappeler :



Bouton Poussoir NO | Bouton Poussoir NF | Interrupteur

Tension et courant

Voici maintenant quelques petites précisions sur les boutons :

- Lorsqu'il est ouvert, la tension à ses bornes ne peut être nulle (ou alors c'est que le circuit n'est pas alimenté). En revanche, lorsqu'il est fermé cette même tension doit être nulle. En effet, aux bornes d'un fil la tension est de 0V.
- Ensuite, lorsque le bouton est ouvert, aucun courant ne peut passer, le circuit est donc déconnecté. Par contre, lorsqu'il est fermé, le courant nécessaire au bon fonctionnement des différents composants le traverse. Il est donc important de prendre en compte cet aspect. Un bouton devant supporter deux ampères ne sera pas aussi gros qu'un bouton tolérant 100 ampères (et pas aussi cher ☺)

Il est très fréquent de trouver des boutons dans les starters kit.

Souvent ils ont 4 pattes (comme sur l'image ci-dessous). Si c'est le cas, sachez que les broches sont reliées deux à deux. Cela signifie quelles fonctionnent par paire. Il faut donc se méfier lorsque vous le brancher sinon vous obtiendrez le même comportement qu'un fil (si vous connectez deux broches reliés). Utilisez un multimètre pour déterminer quels broches sont distinctes. **Pour ne pas se tromper, on utilise en général deux broches qui sont opposées sur la diagonale du bouton.**

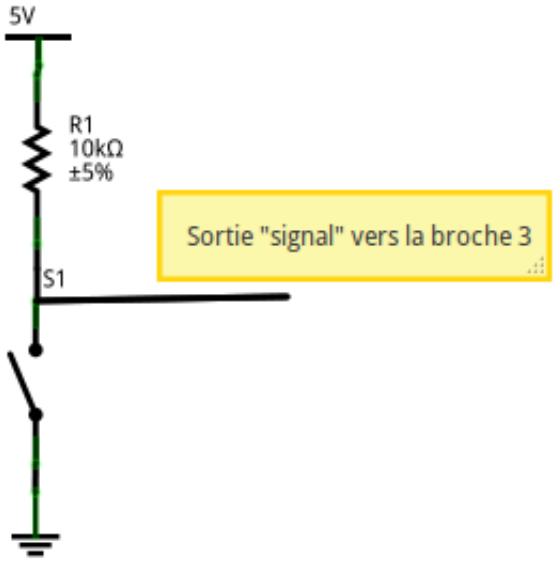
Contrainte pour les montages

Voici maintenant un point très important, soyez donc attentif car je vais vous expliquer le rôle d'une résistance de pull-up !

C'est quoi st'animal, le poule-eup ?

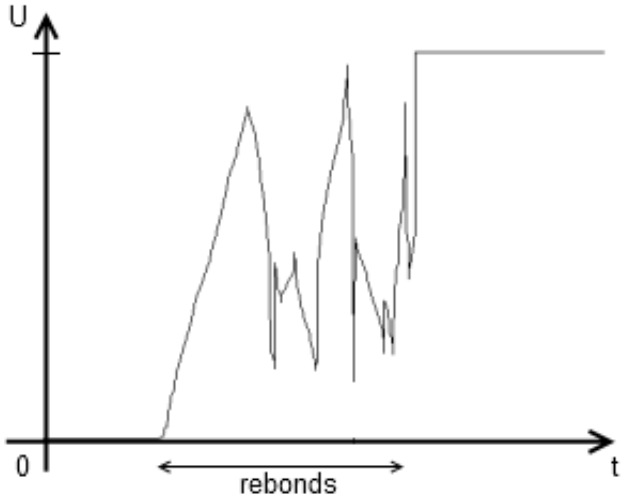
Lorsque l'on fait de l'électronique, on a toujours peur des perturbations (générées par plein de choses : des lampes à proximité, un téléphone portable, un doigt sur le circuit, l'électricité statique, …). On appelle ça des contraintes de **CEM**. Ces perturbations sont souvent inoffensives, mais perturbent beaucoup les montages électroniques. Il est alors nécessaire d'en prendre compte lorsque l'on fait de l'électronique de signal. Par exemple, dans certains cas on peut se retrouver avec un bit de signal qui vaut 1 à la place de 0, les données reçues sont donc fausses. Pour contrer ces effets nuisibles, ont place en série avec le bouton une résistance de pull-up. Cette résistance sert à "tirer" ("to pull" in english) le potentiel vers le haut (up) afin d'avoir un signal clair sur la broche étudiée. Sur le schéma suivant, on voit ainsi qu'en temps normal le "signal" à un potentiel de 5V. Ensuite, lorsque l'utilisateur appuiera sur le bouton une connexion sera faite avec la masse. On lira alors une valeur de 0V pour le signal. Voici donc un deuxième intérêt de la résistance de pull-up, éviter le court-circuit qui serait généré à

l'appui !



Filtrer les rebonds

Les boutons ne sont pas des systèmes mécaniques parfaits. Du coup, lorsqu'un appui est fait dessus, le signal ne passe pas immédiatement et proprement de 5V à 0V. En l'espace de quelques millisecondes, le signal va "sauter" entre 5V et 0V plusieurs fois avant de se stabiliser. Il se passe le même phénomène lorsque l'utilisateur relâche le bouton. Ce genre d'effet n'est pas désirable, car il peut engendrer des parasites au sein de votre programme (si vous voulez détecter un appui, les rebonds vont vous en générer une dizaine en quelques millisecondes, ce qui peut-être très gênant dans le cas d'un compteur par exemple). Voilà un exemple de chronogramme relevé lors du relâchement d'un bouton poussoir :



Pour atténuer ce phénomène, nous allons utiliser un condensateur en parallèle avec le bouton. Ce composant servira ici "d'amortisseur" qui absorbera les rebonds (comme sur une voiture avec les cahots de la route). Le condensateur, initialement chargé, va se décharger lors de l'appui sur le bouton. S'il y a des rebonds, ils seront encaissés par le condensateur durant cette décharge. Il se passera le phénomène inverse (charge du condensateur) lors du relâchement du bouton. Ce principe est illustré à la figure suivante :

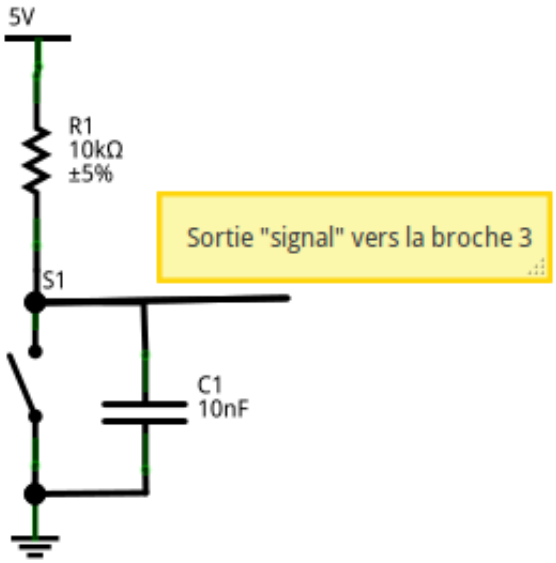
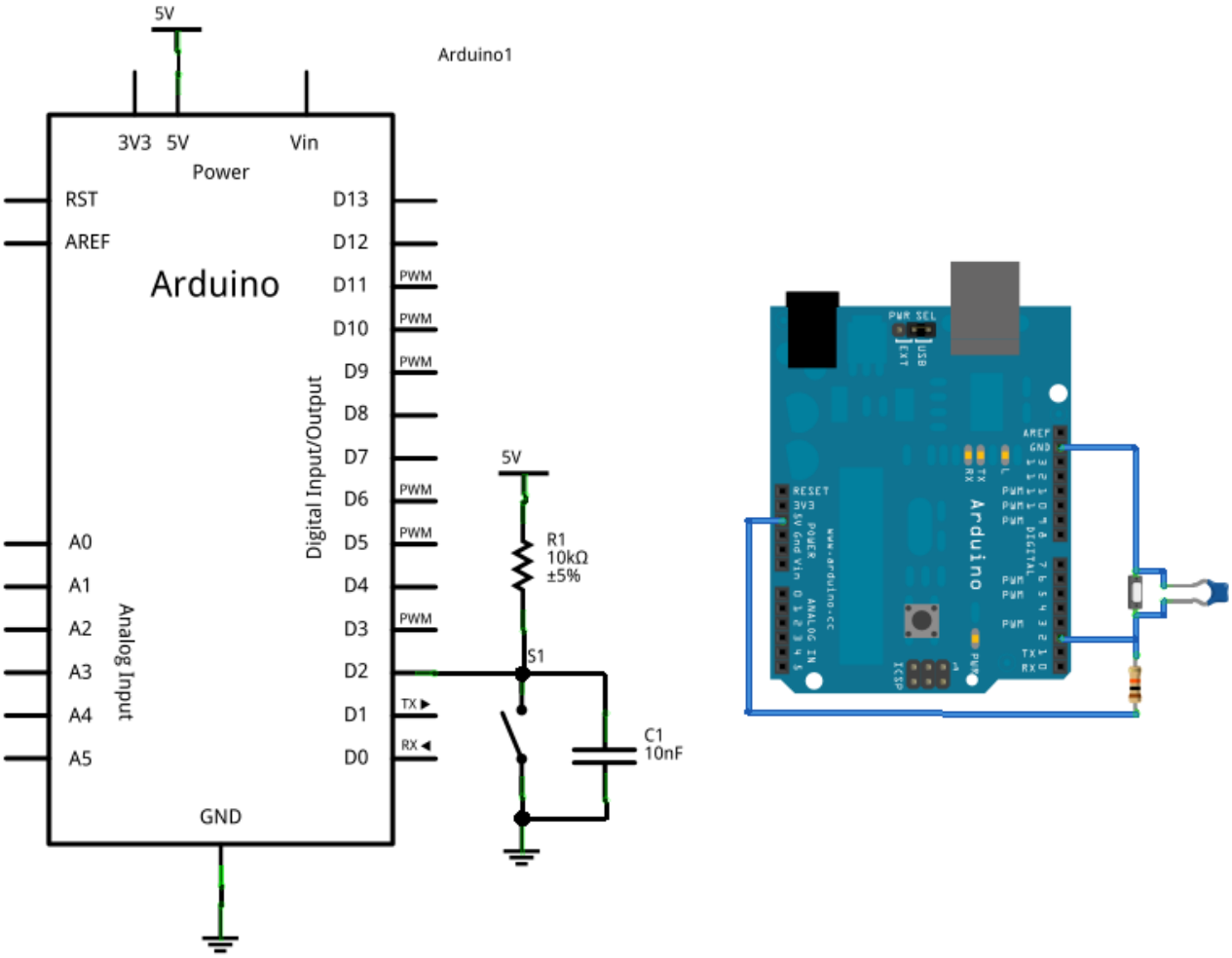


Schéma résumé

En résumé, voilà un montage que vous pourriez obtenir avec un bouton, sa résistance de pull-up et son filtre anti-rebond sur votre carte Arduino :



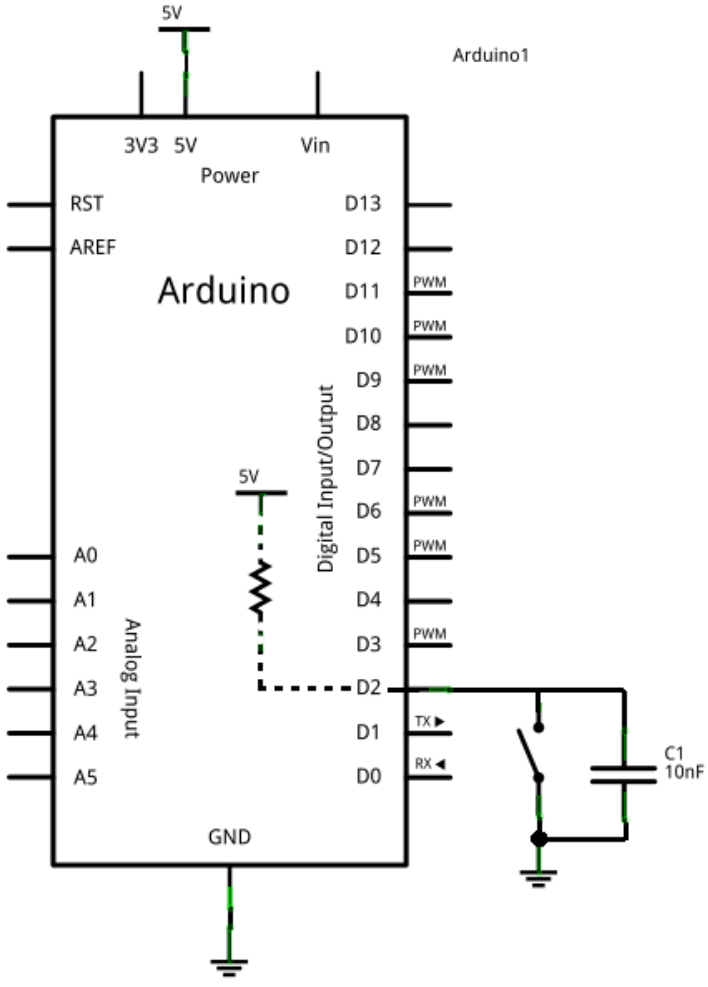
Les pull-ups internes

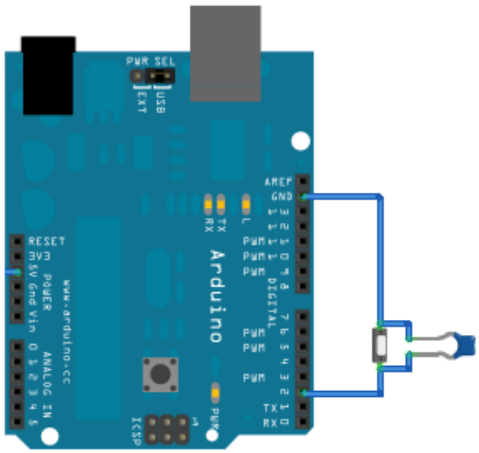
Comme expliqué précédemment, pour obtenir des signaux clairs et éviter les courts-

circuits, on utilise des résistances de pull-up. Cependant, ces dernières existent aussi en interne du microcontrôleur de l'Arduino, ce qui évite d'avoir à les rajouter par nous mêmes par la suite. Ces dernières ont une valeur de 20 kilo-Ohms. Elles peuvent être utilisés sans aucune contraintes techniques. Cependant, si vous les mettez en marche, il faut se souvenir que cela équivaut à mettre la broche à l'état haut (et en entrée évidemment). Donc si vous repassez à un état de sortie ensuite, rappelez vous bien que tant que vous ne l'avez pas changée elle sera à l'état haut. Ce que je vient de dire permet de mettre en place ces dernières dans le logiciel :

```
1  const int unBouton = 2; //un bouton sur la broche 2
2
3  void setup()
4  {
5      //on met le bouton en entrée
6      pinMode(unBouton, INPUT);
7      //on active la résistance de pull-up en mettant la broche à l'état haut (mais ce
8      digitalWrite(unBouton, HIGH);
9  }
10
11 void loop()
12 {
13     //votre programme
14 }
```

Schéma résumé





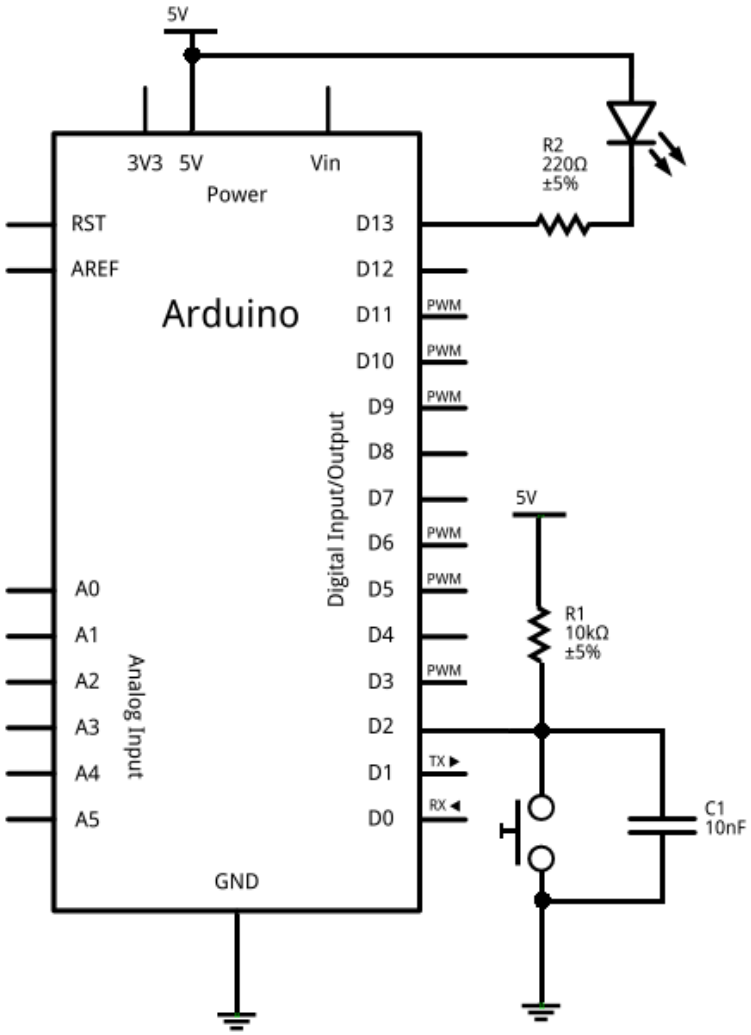
Récupérer l'appui du bouton

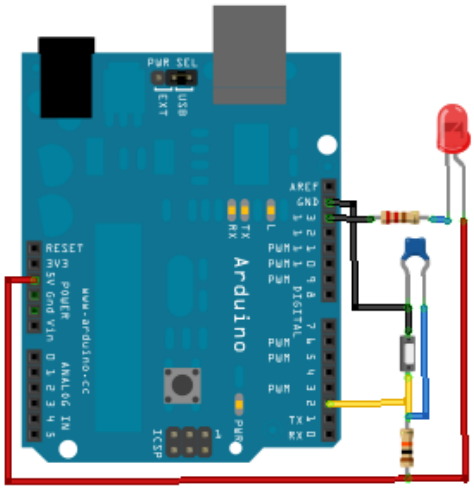
Montage de base

Pour cette partie, nous allons apprendre à lire l'état d'une entrée numérique. Tout d'abord, il faut savoir qu'une entrée numérique ne peut prendre que deux états, HAUT (HIGH) ou BAS (LOW). L'état haut correspond à une tension de +5V sur la broche, tandis que l'état bas est une tension de 0V. Dans notre exemple, nous allons utiliser un simple bouton. Dans la réalité, vous pourriez utiliser n'importe quel capteur qui possède une sortie numérique. Nous allons donc utiliser :

- Un bouton poussoir (et une résistance de 10k de pull-up et un condensateur anti-rebond de 10nF)
- Une LED (et sa résistance de limitation de courant)
- La carte Arduino

Voici maintenant le schéma à réaliser :





Montage simple avec un bouton et une LED

Paramétrer la carte

Afin de pouvoir utiliser le bouton, il faut spécifier à Arduino qu'il y a un bouton de connecté sur une de ses broches. Cette broche sera donc une **entrée**. Bien entendu, comme vous êtes de bons élèves, vous vous souvenez que tous les paramétrages initiaux se font dans la fonction `setup()`. Vous vous souvenez également que pour définir le type (entrée ou sortie) d'une broche, on utilise la fonction : pinMode(). Notre bouton étant branché sur la pin 2, on écrira :

```
pinMode(2, INPUT);
```

Pour plus de clarté dans les futurs codes, on considérera que l'on a déclaré une variable globale nommée "bouton" et ayant la valeur 2. Comme ceci :

```
const int bouton = 2;

void setup()
{
    pinMode(bouton, INPUT);
}
```

Voilà, maintenant notre carte Arduino sait qu'il y a quelque chose de connecté sur sa broche 2 et que cette broche est configurée en entrée.

Récupérer l'état du bouton

Maintenant que le bouton est paramétré, nous allons chercher à savoir quel est son état (appuyé ou relâché).

- S'il est relâché, la tension à ses bornes sera de +5V, donc un état logique HIGH.
- S'il est appuyé, elle sera de 0V, donc LOW.

Un petit tour sur la référence et nous apprenons qu'il faut utiliser la fonction digitalRead() pour lire l'état logique d'une entrée logique. Cette fonction prend un paramètre qui est la broche à tester et elle retourne une variable de type int. Pour lire l'état de la broche 2 nous ferons donc :

```
int etat;

void loop()
{
    etat = digitalRead(bouton); //Rappel : bouton = 2

    if(etat == HIGH)
        actionRelache(); //le bouton est relaché
    else
        actionAppui(); //le bouton est appuyé
}
```

Observez dans ce code, on appelle deux fonctions qui dépendent de l'état du bouton. Ces fonctions ne sont pas présentes dans ce code, si vous le testez ainsi, il ne fonctionnera pas. Pour ce faire, vous devrez créer les fonctions `actionAppui()`.

Test simple

Nous allons passer à un petit test, que vous allez faire. Moi je regarde ! 😈

But

L'objectif de ce test est assez simple : lorsque l'on appuie sur le bouton, la LED doit s'allumer. Lorsque l'on relâche le bouton, la LED doit s'éteindre. Autrement dit, tant que le bouton est appuyé, la LED est allumée.

Correction

Allez, c'est vraiment pas dur, en plus je vous donnais le montage dans la première partie… Voici la correction :

- - Les variables globales

```
//le bouton est connecté à la broche 2 de la carte Adruino
const int bouton = 2;
//la LED à la broche 13
const int led = 13;

//variable qui enregistre l'état du bouton
int etatBouton;
```

- - La fonction setup()

```
void setup()
{
    pinMode(led, OUTPUT); //la led est une sortie
    pinMode(bouton, INPUT); //le bouton est une entrée
    etatBouton = HIGH; //on initialise l'état du bouton comme "relaché"
}
```

- - La fonction loop()

```
void loop()
{
    etatBouton = digitalRead(bouton); //Rappel : bouton = 2

    if(etatBouton == HIGH) //test si le bouton a un niveau logique HAUT
    {
        digitalWrite(led,HIGH); //la LED reste éteinte
    }
    else  //test si le bouton a un niveau logique différent de HAUT (donc BAS)
    {
        digitalWrite(led,LOW); //le bouton est appuyé, la LED est allumée
    }
}
```

J'espère que vous y êtes parvenu sans trop de difficultés ! Si oui, passons à l'exercice suivant…

Interagir avec les LEDs

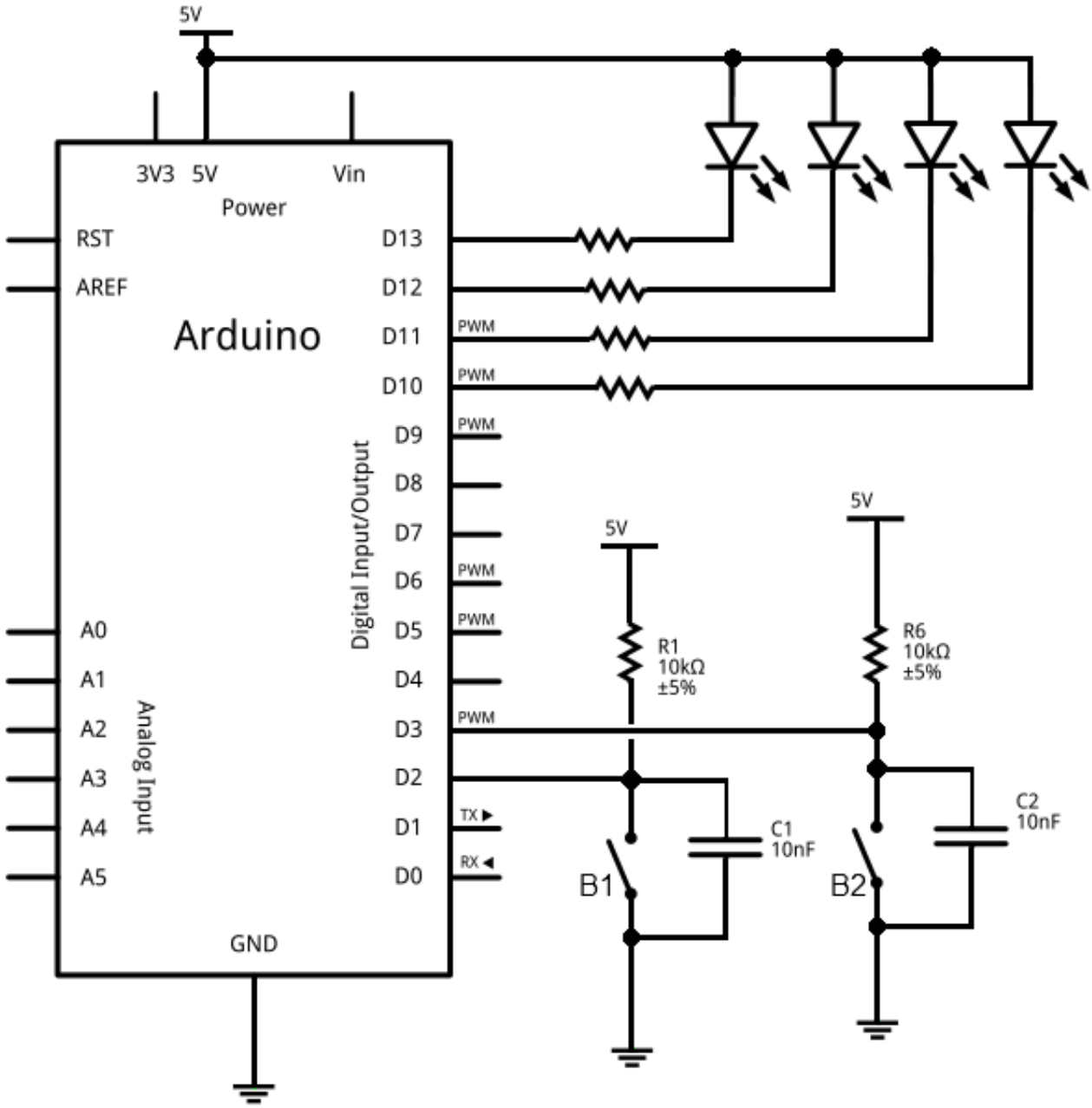
Nous allons maintenant faire un exemple d'application ensemble.

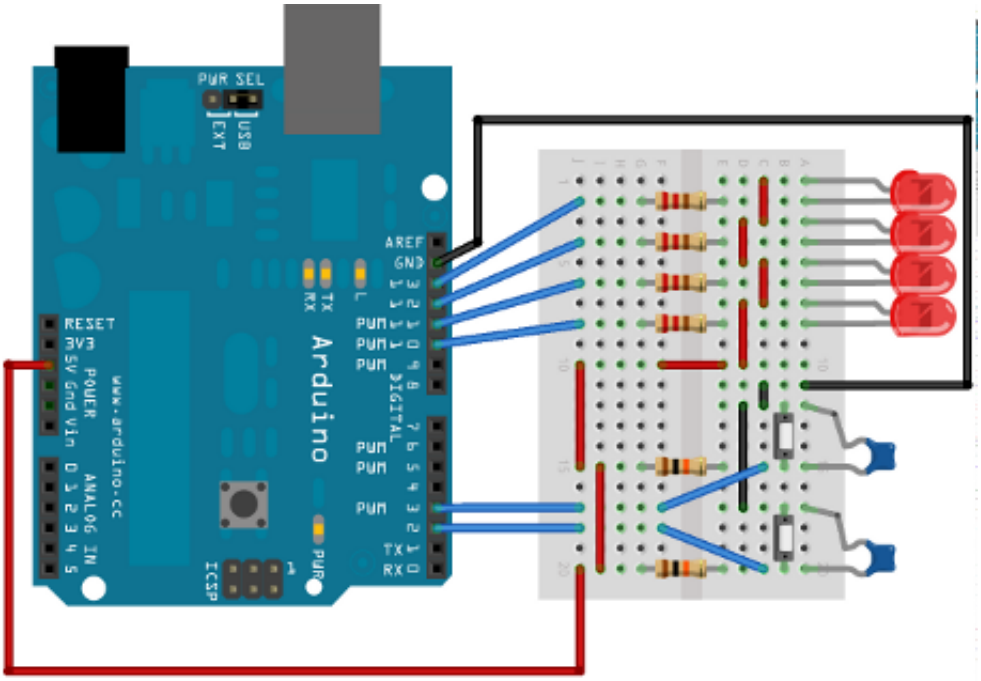
Montage à faire

Pour cet exercice, nous allons utiliser deux boutons et quatre LEDs de n'importe quelles couleurs.

- Les deux boutons seront considérés actifs (appuyés) à l'état bas (0V) comme dans la partie précédente. Ils seront connectés sur les broches 2 et 3 de l'Arduino.
- Ensuite, les 4 LEDs seront connectées sur les broches 10 à 13 de l'Arduino.

Voilà donc le montage à effectuer :


5V
3V3
5V
Vin
Power
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
PWM
PWM
PWM
PWM
TX
RX
Digital Input/Output
A0
A1
A2
A3
A4
A5
Analog Input
GND
5V
R1
10kΩ
±5%
5V
R6
10kΩ
±5%
B1
C1
10nF
B2
C2
10nF



Montage de l'exercice, avec deux boutons et quatre LEDs

Objectif : Barregraphe à LED

Dans cet exercice, nous allons faire un mini-barregraphe. Un barregraphe est un afficheur qui indique une quantité, provenant d'une information quelconque (niveau d'eau, puissance sonore, etc.), sous une forme lumineuse. Le plus souvent, on utilise des LEDs alignées en guise d'affichage. Chaque LED se verra allumée selon un niveau qui sera une fraction du niveau total. Par exemple, si je prends une information qui varie entre 0 et 100, chacune des 4 LED correspondra au quart du maximum de cette variation. Soit 100 / 4 = 25. En l'occurrence, l'information entrante c'est l'appui des boutons. Par conséquent un appui sur un bouton allume une LED, un appui sur un autre bouton éteint une LED. En fait ce n'est pas aussi direct, il faut incrémenter ou décrémenter la valeur d'une variable et en fonction de cette valeur, on allume telle quantité de LED.

Cahier des charges

La réalisation prévue devra :

- - posséder 4 LED (ou plus pour les plus téméraires)
- - posséder 2 boutons : un qui incrémentera le nombre de LED allumées, l'autre qui le décrémentera

Vous devrez utiliser une variable qui voit sa valeur augmenter ou diminuer entre 1 et 4 selon l'appui du bouton d'incrémentation ou de décrémentation.

Vous pouvez maintenant vous lancer dans l'aventure. Pour ceux qui se sentiraient encore un peu mal à l'aise avec la programmation, je vous autorise à poursuivre la lecture qui vous expliquera pas à pas comment procéder pour arriver au résultat final. 😉

Correction

Initialisation

Pour commencer, on créer et on initialise toutes les variables dont on a besoin dans notre programme :

```
/* déclaration des constantes pour les noms des broches ; ceci selon le schéma*/
const int btn_minus = 2;
const int btn_plus = 3;
const int led_0 = 10;
const int led_1 = 11;
const int led_2 = 12;
const int led_3 = 13;


/* déclaration des variables utilisées pour le comptage et le décomptage */

int nombre_led = 0; //le nombre qui sera incrémenté et décrémenté
int etat_bouton; //lecture de l'état des boutons (un seul à la fois mais une variabl

/* initilisation des broches en entrée/sortie */
void setup()
{
    pinMode(btn_plus, INPUT);
    pinMode(btn_minus, INPUT);
    pinMode(led_0, OUTPUT);
    pinMode(led_1, OUTPUT);
    pinMode(led_2, OUTPUT);
    pinMode(led_3, OUTPUT);
}

void loop()
{
    //les instructions de votre programme
}
```

Détection des différences appuyé/relâché

Afin de détecter un appui sur un bouton, nous devons comparer son état **courant** avec son état **précédent**. C'est-à-dire qu'avant qu'il soit appuyé ou relâché, on lit son état et on l'inscrit dans une variable. Ensuite, on relit si son état à changé. Si c'est le cas alors on incrémente la variable `nombre_led`. Pour faire cela, on va utiliser une variable de plus par bouton :

```
int memoire_plus = HIGH; //état relâché par défaut
int memoire_minus = HIGH;
```

Détection du changement d'état

Comme dit précédemment, nous devons détecter le changement de position du bouton, sinon on ne verra rien car tout se passera trop vite. Voilà le programme de la boucle principale :

```
void loop()
{
    //lecture de l'état du bouton d'incrémentation
    etat_bouton = digitalRead(btn_plus);

```

```
    //Si le bouton a un état différent que celui enregistré ET que cet état est "app
    if((etat_bouton != memoire_plus) && (etat_bouton == LOW))
    {
        nombre_led++; //on incrémente la variable qui indique combien de LED devrons
    }

    memoire_plus = etat_bouton; //on enregistre l'état du bouton pour le tour suivan


    //et maintenant pareil pour le bouton qui décrémente
    etat_bouton = digitalRead(btn_minus); //lecture de son état

    //Si le bouton a un état différent que celui enregistré ET que cet état est "app
    if((etat_bouton != memoire_minus) && (etat_bouton == LOW))
    {
        nombre_led--; //on décrémente la valeur de nombre_led
    }
    memoire_minus = etat_bouton; //on enregistre l'état du bouton pour le tour suiva

    //on applique des limites au nombre pour ne pas dépasser 4 ou 0
    if(nombre_led > 4)
    {
        nombre_led = 4;
    }
    if(nombre_led < 0)
    {
        nombre_led = 0;
    }

    //appel de la fonction affiche() que l'on aura créée
    //on lui passe en paramètre la valeur du nombre de LED à éclairer
    affiche(nombre_led);
}
```

Nous avons terminé de créer le squelette du programme et la détection d'évènement, il ne reste plus qu'à afficher le résultat du nombre !

L'affichage

Pour éviter de se compliquer la vie et d'alourdir le code, on va créer une fonction d'affichage. Celle dont je viens de vous parler : `affiche(int le_parametre)`. Cette fonction reçoit un paramètre représentant le nombre à afficher. A présent, nous devons allumer les LEDs selon la valeur reçue. On sait que l'on doit afficher une LED lorsque l'on reçoit le nombre 1, 2 LEDs lorsqu'on reçoit le nombre 2, …

```
void affiche(int valeur_recue)
{
    //on éteint toutes les LEDs
    digitalWrite(led_0, HIGH);
    digitalWrite(led_1, HIGH);
    digitalWrite(led_2, HIGH);
    digitalWrite(led_3, HIGH);

    //Puis on les allume une à une
    if(valeur_recue >= 1)
    {
        digitalWrite(led_0, LOW);
    }
    if(valeur_recue >= 2)
```

```
    {
        digitalWrite(led_1, LOW);
    }
    if(valeur_recue >= 3)
    {
        digitalWrite(led_2, LOW);
    }
    if(valeur_recue >= 4)
    {
        digitalWrite(led_3, LOW);
    }
}
```

Donc, si la fonction reçoit le nombre 1, on allume la LED 1. Si elle reçoit le nombre 2, elle allume la LED 1 et 2. Si elle reçoit 3, elle allume la LED 1, 2 et 3. Enfin, si elle reçoit 4, alors elle allume toutes les LEDs. Le code au grand complet :

```
/* déclaration des constantes pour les nom des broches ; ceci selon le schéma*/
const int btn_minus = 2;
const int btn_plus = 3;
const int led_0 = 10;
const int led_1 = 11;
const int led_2 = 12;
const int led_3 = 13;


/* déclaration des variables utilisées pour le comptage et le décomptage */

int nombre_led = 0; //le nombre qui sera incrémenté et décrémenté
int etat_bouton; //lecture de l'état des boutons (un seul à la fois mais une variabl

int memoire_plus = HIGH; //état relâché par défaut
int memoire_minus = HIGH;


/* initilisation des broches en entrée/sortie */
void setup()
{
    pinMode(btn_plus, INPUT);
    pinMode(btn_minus, INPUT);
    pinMode(led_0, OUTPUT);
    pinMode(led_1, OUTPUT);
    pinMode(led_2, OUTPUT);
    pinMode(led_3, OUTPUT);
}

void loop()
{
    //lecture de l'état du bouton d'incrémentation
    etat_bouton = digitalRead(btn_plus);

    //Si le bouton a un état différent que celui enregistré ET que cet état est "app
    if((etat_bouton != memoire_plus) && (etat_bouton == LOW))
    {
        nombre_led++; //on incrémente la variable qui indique combien de LED devrons
    }

    memoire_plus = etat_bouton; //on enregistre l'état du bouton pour le tour suivan


```

```
    //et maintenant pareil pour le bouton qui décrémente
    etat_bouton = digitalRead(btn_minus); //lecture de son état

    //Si le bouton a un état différent que celui enregistré ET que cet état est "app
    if((etat_bouton != memoire_minus) && (etat_bouton == LOW))
    {
        nombre_led--; //on décrémente la valeur de nombre_led
    }
    memoire_minus = etat_bouton; //on enregistre l'état du bouton pour le tour suiva

    //on applique des limites au nombre pour ne pas dépasser 4 ou 0
    if(nombre_led > 4)
    {
        nombre_led = 4;
    }
    if(nombre_led < 0)
    {
        nombre_led = 0;
    }

    //appel de la fonction affiche() que l'on aura créée
    //on lui passe en paramètre la valeur du nombre de LED à éclairer
    affiche(nombre_led);
}

void affiche(int valeur_recue)
{
    //on éteint toutes les leds
    digitalWrite(led_0, HIGH);
    digitalWrite(led_1, HIGH);
    digitalWrite(led_2, HIGH);
    digitalWrite(led_3, HIGH);

    //Puis on les allume une à une
    if(valeur_recue >= 1)
    {
        digitalWrite(led_0, LOW);
    }
    if(valeur_recue >= 2)
    {
        digitalWrite(led_1, LOW);
    }
    if(valeur_recue >= 3)
    {
        digitalWrite(led_2, LOW);
    }
    if(valeur_recue >= 4)
    {
        digitalWrite(led_3, LOW);
    }
}
```

Une petite vidéo du résultat que vous devriez obtenir, même si votre code est différent du mien :

Les interruptions matérielles

Voici maintenant un sujet plus délicat (mais pas tant que ça ! :ninja:) qui demande votre attention.

Comme vous l'avez remarqué dans la partie précédente, pour récupérer l'état du bouton il faut surveiller régulièrement l'état de ce dernier. Cependant, si le programme a quelque chose de long à traiter, par exemple s'occuper de l'allumage d'une LED et faire une pause avec delay() (bien que l'on puisse utiliser millis()), l'appui sur le bouton ne sera pas très réactif et lent à la détente. Pour certaines applications, cela peut gêner. Problème : si l'utilisateur appuie et relâche rapidement le bouton, vous pourriez ne pas détecter l'appui (si vous êtes dans un traitement long). Solution : Utiliser le mécanisme d'**interruption**.

Principe

Dans les parties précédentes de ce chapitre, la lecture d'un changement d'état se faisait en comparant régulièrement l'état du bouton à un moment avec son état précédent. Cette méthode fonctionne bien, mais pose un problème : l'appui ne peut pas être détecté s'il est trop court. Autre situation, si l'utilisateur fait un appui très long, mais que vous êtes déjà dans un traitement très long (calcul de la millième décimale de PI, soyons fous), le temps de réponse à l'appui ne sera pas du tout optimal, l'utilisateur aura une impression de lag (= pas réactif). Pour pallier ce genre de problème, les constructeurs de microcontrôleurs ont mis en place des systèmes qui permettent de détecter des évènements et d'exécuter des fonctions dès la détection de ces derniers. Par exemple, lorsqu'un pilote d'avion de chasse demande au siège de s'éjecter, le siège doit réagir au moment de l'appui, pas une minute plus tard (trop tard).

Qu'est-ce qu'une interruption ?

Une interruption est en fait un déclenchement qui arrête l'exécution du programme pour faire une tâche demandée. Par exemple, imaginons que le programme compte jusqu'à l'infinie. Moi, programmeur, je veux que le programme arrête de compter lorsque j'appuie sur un bouton. Or, il s'avère que la fonction qui compte est une boucle for(), dont on ne peut sortir sans avoir atteint l'infinie (autrement dit jamais, en théorie). Nous allons donc nous tourner vers les interruptions qui, dès que le bouton sera appuyé, interromprons le programme pour lui dire : "*Arrête de compter, c'est l'utilisateur qui le demande !*". Pour résumer : **une interruption du programme est générée lors d'un événement attendu. Ceci dans le but d'effectuer une tâche, puis de reprendre**

l'exécution du programme. Arduino propose aussi ce genre de gestion d'évènements. On les retrouvera sur certaines broches, sur des timers, des liaisons de communication, etc.

Mise en place

Nous allons illustrer ce mécanisme avec ce qui nous concerne ici, les boutons. Dans le cas d'une carte Arduino UNO, on trouve deux broches pour gérer des interruptions externes (qui ne sont pas dues au programme lui même), la 2 et la 3. Pour déclencher une interruption, plusieurs cas de figure sont possibles :

- **LOW** : Passage à l'état bas de la broche
- **FALLING** : Détection d'un front descendant (passage de l'état haut à l'état bas)
- **RISING** : Détection d'un front montant (pareil qu'avant, mais dans l'autre sens)
- **CHANGE** : Changement d'état de la broche

Autrement dit, s'il y a un changement d'un type énuméré au-dessus, alors le programme sera interrompu pour effectuer une action.

Créer une nouvelle interruption

Comme d'habitude, nous allons commencer par faire des réglages dans la fonction setup(). La fonction importante à utiliser est `attachInterrupt(interrupt, function, mode)`. Elle accepte trois paramètres :

- - interrupt : qui est le numéro de la broche utilisée pour l'interruption (0 pour la broche 2 et 1 pour la broche 3)
- - function : qui est le nom de la fonction à appeler lorsque l'interruption est déclenchée
- - mode : qui est le type de déclenchement (cf. ci-dessus)

Si l'on veut appeler une fonction nommée `Reagir()` lorsque l'utilisateur appuie sur un bouton branché sur la broche 2 on fera :

```
1                                   attachInterrupt(0, Reagir, FALLING);
```

Vous remarquerez l'absence des parenthèses après le nom de la fonction "Reagir"

Ensuite, il vous suffit de coder votre fonction Reagir() un peu plus loin.

Attention, cette fonction ne peut pas prendre d'argument et ne retournera aucun résultat.

Lorsque quelque chose déclenchera l'interruption, le programme principal sera mis en pause. Ensuite, lorsque l'interruption aura été exécutée et traitée, il reprendra comme si rien ne s'était produit (avec peut-être des variables mises à jour).

Mise en garde

Si je fais une partie entière sur les interruptions, ce n'est pas que c'est difficile mais

c'est surtout pour vous mettre en garde sur certains points. Tout d'abord, **les interruptions ne sont pas une solution miracle**. En effet, gardez bien en tête que leur utilisation répond à un besoin **justifié**. Elles mettent tout votre programme en pause, et une mauvaise programmation (ce qui n'arrivera pas, je vous fais confiance) peut entraîner une altération de l'état de vos variables. De plus, les fonctions delay() et millis() n'auront pas un comportement correct. En effet, pendant ce temps le programme principal est complètement stoppé, donc les fonctions gérant le temps ne fonctionneront plus, elles seront aussi en pause et laisseront la priorité à la fonction d'interruption. La fonction delay() est donc désactivée et la valeur retournée par millis() ne changera pas. Justifiez donc votre choix avant d'utiliser les interruptions.

Et voilà, vous savez maintenant comment donner de l'interactivité à l'expérience utilisateur. Vous avez pu voir quelques applications, mais nul doute que votre imagination fertile va en apporter de nouvelles !

[Arduino 205] Afficheurs 7 segments

Vous connaissez les afficheurs 7 segments ? Ou alors vous ne savez pas que ça s'appelle comme ça ? Il s'agit des petites lumières qui forment le chiffre 8 et qui sont de couleur rouge ou verte, la plupart du temps, mais peuvent aussi être bleus, blancs, etc. On en trouve beaucoup dans les radio-réveils, car ils servent principalement à afficher l'heure. Autre particularité, non seulement de pouvoir afficher des chiffres (0 à 9), ils peuvent également afficher certaines lettres de l'alphabet.

Matériel

Pour ce chapitre, vous aurez besoin de :

- Un (et plus) afficheur 7 segments (évidemment)
- 8 résistances de 330Ω
- Un (ou deux) décodeurs BCD 7 segments
- Une carte Arduino ! Mais dans un premier temps on va d'abord bien saisir le truc avant de faire du code

Nous allons commencer par une découverte de l'afficheur, comment il fonctionne et comment le branche-t-on. Ensuite nous verrons comment l'utiliser avec la carte Arduino. Enfin, le chapitre suivant amènera un TP résumant les différentes parties vues.

Première approche : côté électronique

Un peu (beaucoup) d'électronique

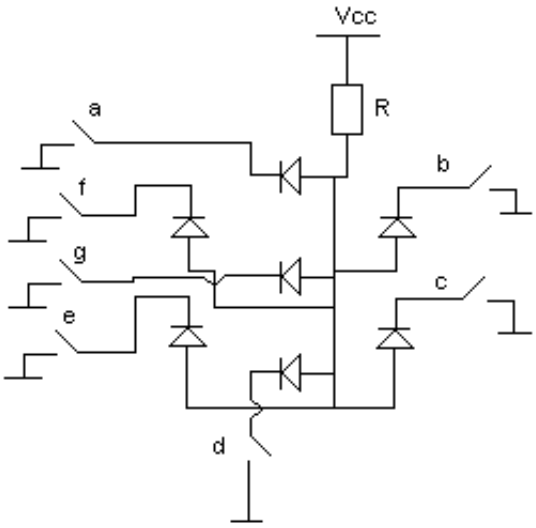
Comme son nom l'indique, l'afficheur 7 segments possède… 7 segments. Mais un segment c'est quoi au juste ? Et bien c'est une portion de l'afficheur, qui est allumée ou éteinte pour réaliser l'affichage. Cette portion n'est en fait rien d'autre qu'une LED qui au lieu d'être ronde comme d'habitude est plate et encastré dans un boiter. On dénombre donc 8 portions en comptant le point de l'afficheur (mais il ne compte pas en

tant que segment à part entière car il n'est pas toujours présent). Regardez à quoi ça ressemble :

Afficheur 7 segments

Des LED, encore des LED

Et des LED, il y en a ! Entre 7 et 8 selon les modèles (c'est ce que je viens d'expliquer), voir beaucoup plus, mais on ne s'y attardera pas dessus. Voici un schéma vous présentant un modèle d'afficheur sans le point (qui au final est juste une LED supplémentaire rappelez-vous) :



Les interrupteurs a,b,c,d,e,f,g représentent les signaux pilotant chaque segments

Comme vous le voyez sur ce schéma, toutes les LED possèdent une broche commune, reliée entre elle. Selon que cette broche est la cathode ou l'anode on parlera d'afficheur à cathode commune ou… anode commune (vous suivez ?). Dans l'absolu, ils fonctionnent de la même façon, seule la manière de les brancher diffère (actif sur état bas ou sur état haut).

Cathode commune ou Anode commune

Dans le cas d'un afficheur à cathode commune, toutes les cathodes sont reliées entre elles en un seul point lui-même connecté à la masse. Ensuite, chaque anode de chaque segment sera reliée à une broche de signal. Pour allumer chaque segment, le signal devra être une tension positive. En effet, si le signal est à 0, il n'y a pas de différence de potentiel entre les deux broches de la LED et donc elle ne s'allumera pas ! Si nous sommes dans le cas d'une anode commune, les anodes de toutes les LED sont reliées entre elles en un seul point qui sera connecté à l'alimentation. Les cathodes elles seront reliées une par une aux broches de signal. En mettant une broche de signal à 0, le courant passera et le segment en question s'allumera. Si la broche de signal est à l'état haut, le potentiel est le même de chaque côté de la LED, donc elle est bloquée et ne s'allume pas ! Que l'afficheur soit à anode ou à cathode commune, on doit toujours prendre en compte qu'il faut ajouter une résistance de limitation de courant entre la broche isolée et la broche de signal. Traditionnellement, on prendra une

résistance de 330 ohms pour une tension de +5V, mais cela se calcul (cf. chapitre 1, partie 2). Si vous voulez augmenter la luminosité, il suffit de diminuer cette valeur. Si au contraire vous voulez diminuer la luminosité, augmenter la résistance.

Choix de l'afficheur

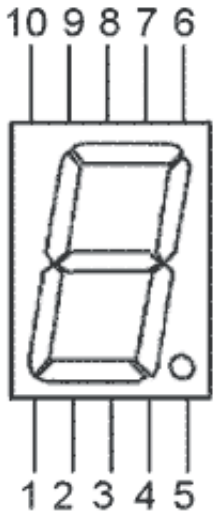
Pour la rédaction j'ai fait le choix d'utiliser des afficheurs à anode commune et ce n'est pas anodin. En effet et on l'a vu jusqu'à maintenant, on branche les LED du +5V vers la broche de la carte Arduino. Ainsi, dans le cas d'un afficheur à anode commune, les LED seront branchés d'un côté au +5V, et de l'autre côté aux broches de signaux. Ainsi, pour allumer un segment on mettra la broche de signal à 0 et on l'éteindra en mettant le signal à 1. On a toujours fait comme ça depuis le début, ça ne vous posera donc aucun problème.

Branchement "complet" de l'afficheur

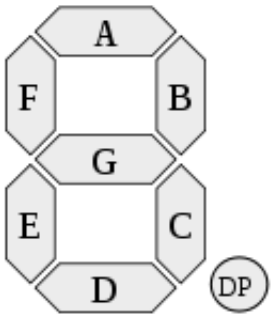
Nous allons maintenant voir comment brancher l'afficheur à anode commune.

Présentation du boîtier

Les afficheurs 7 segments se présentent sur un *boîtier* de type DIP 10. Le format DIP régie l'espacement entre les différentes broches du circuit intégré ainsi que d'autres contraintes (présence d'échangeur thermique etc…). Le chiffre 10 signifie qu'il possède 10 broches (5 de part et d'autre du boitier). Voici une représentation de ce dernier (à gauche) :



PIN NO.	E MAN6960
1	Cathode E
2	Cathode D
3	Com. Anode
4	Cathode C
5	Cathode D.P.
6	Cathode B
7	Cathode A
8	Com. Anode
9	Cathode F
10	Cathode G



Voici la signification des différentes broches :

1. LED de la cathode E
2. LED de la cathode D
3. Anode commune des LED
4. LED de la cathode C
5. (facultatif) le point décimal.
6. LED de la cathode B
7. LED de la cathode A
8. Anode commune des LED

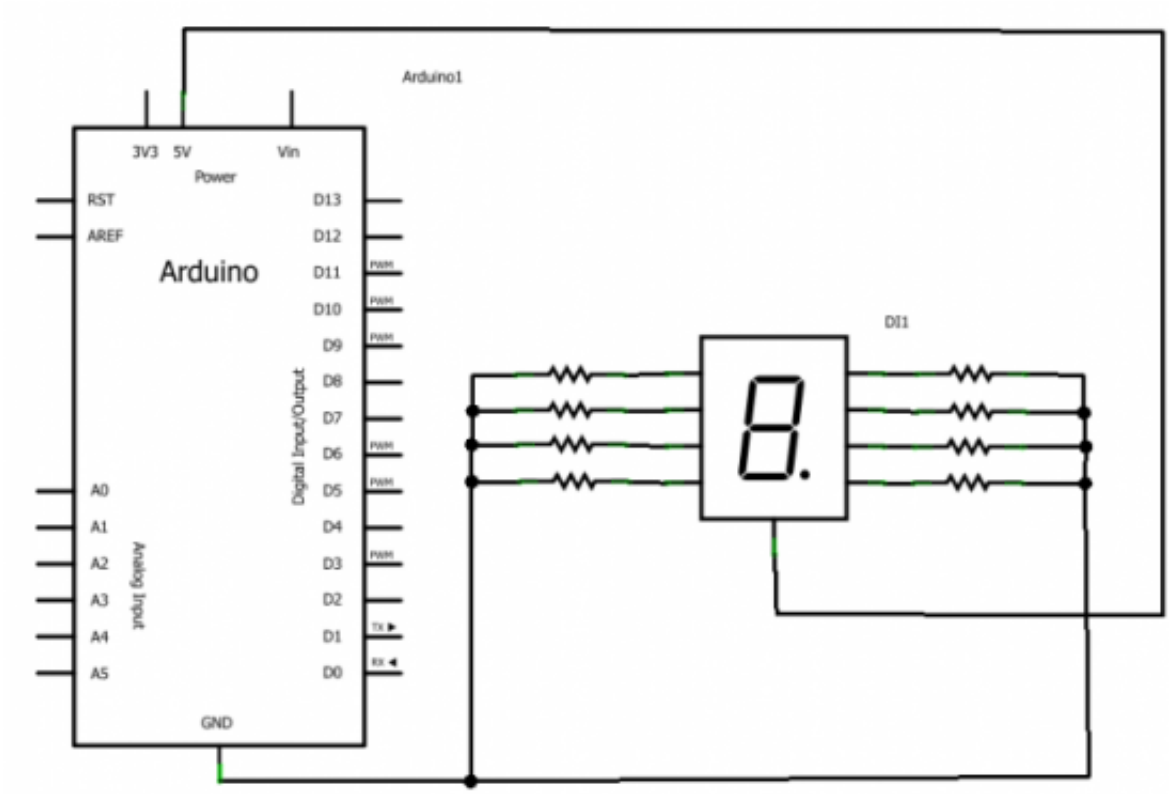
9. LED de la cathode F
10. LED de la cathode G

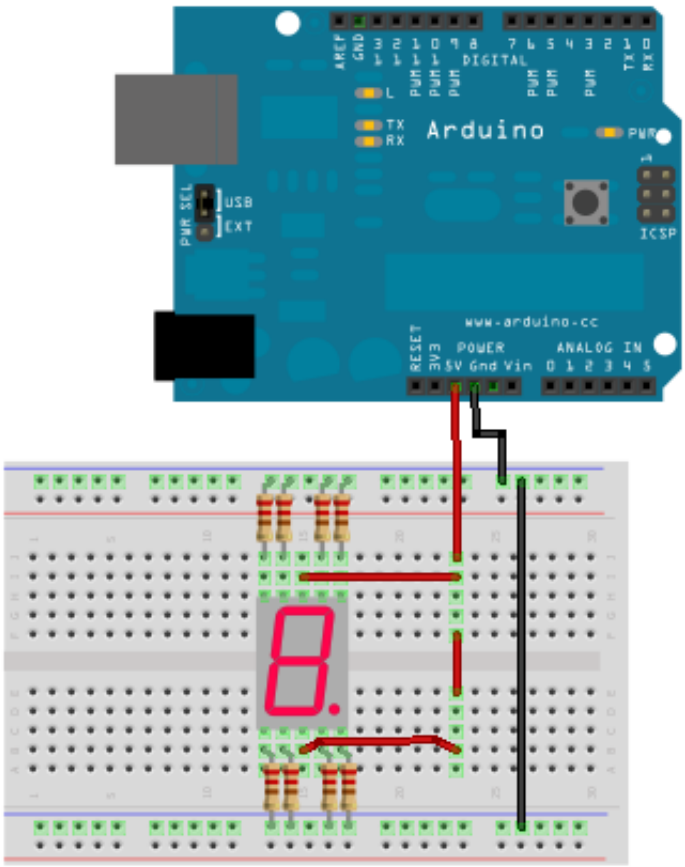
Pour allumer un segment c'est très simple, il suffit de le relier à la masse !

Nous cherchons à allumer les LED de l'afficheur, il est donc impératif de ne pas oubliez les résistances de limitations de courant !

Exemple

Pour commencer, vous allez tout d'abord mettre l'afficheur à cheval sur la plaque d'essai (breadboard). Ensuite, trouvez la broche représentant l'anode commune et reliez la à la future colonne du +5V. Prochaine étape, mettre une résistance de 330Ω sur chaque broche de signal. Enfin, reliez quelques une de ces résistances à la masse. Si tous se passe bien, les segments reliés à la masse via leur résistance doivent s'allumer lorsque vous alimentez le circuit. Voici un exemple de branchement :





Dans cet exemple de montage, vous verrez que tous les segment de l'afficheur s'allument ! Vous pouvez modifier le montage en déconnectant quelques unes des résistance de la masse et afficher de nombreux caractères.

> Pensez à couper l'alimentation lorsque vous changer des fils de place. Les composants n'aiment pas forcément être (dé)branchés lorsqu'ils sont alimentés. Vous pourriez éventuellement leur causer des dommages.

Seulement 7 segments mais plein de caractère(s) !

Vous l'avez peut-être remarqué avec "l'exercice" précédent, un afficheurs 7 segments ne se limite pas à afficher juste des chiffres. Voici un tableau illustrant les caractères possibles et quels segments allumés. Attention, il est possible qu'il manque certains caractères !

Caractère	seg. A	seg. B	seg. C	seg. D	seg. E	seg. F	seg. G
0	x	x	x	x	x	x	
1		x	x				
2	x	x		x	x		x
3	x	x	x	x			x
4		x	x			x	x
5	x		x	x		x	x
6	x		x	x	x	x	x

7	x	x	x				
8	x	x	x	x	x	x	x
9	x	x	x	x		x	x
A	x	x	x		x	x	x
b			x	x	x	x	x
C	x			x	x	x	
d		x	x	x	x	x	
E	x			x	x	x	x
F	x				x	x	x
H		x	x		x	x	x
I		x	x				
J		x	x	x	x		
L				x	x	x	
o			x	x	x		x
P	x	x			x	x	x
S	x		x	x		x	x
t					x	x	x
U		x	x	x	x	x	
y		x	x	x		x	x
°	x	x				x	x

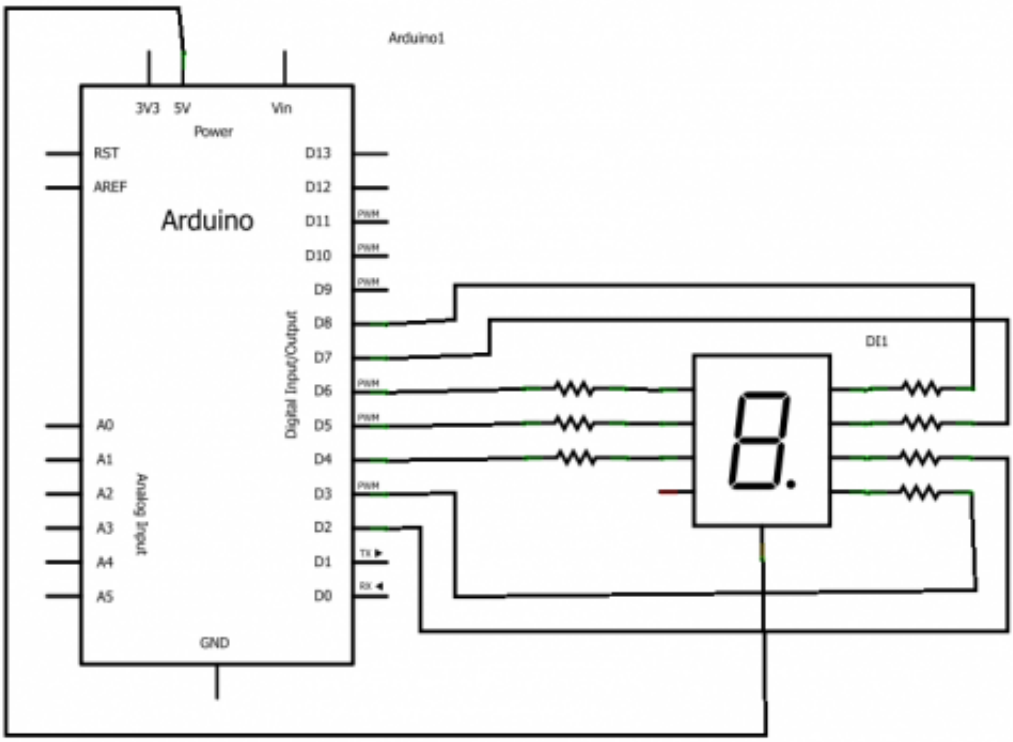
Aidez-vous de ce tableau lorsque vous aurez à coder l'affichage de caractères ! 😉

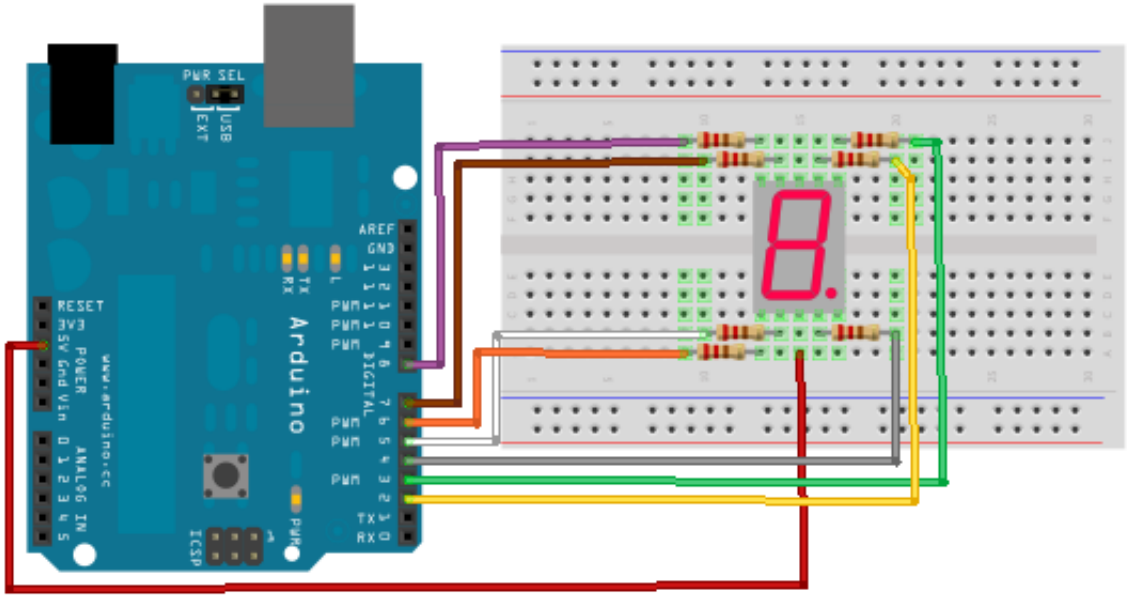
Afficher son premier chiffre !

Pour commencer, nous allons prendre en main un afficheur et lui faire s'afficher notre premier chiffre ! C'est assez simple et ne requiert qu'un programme très simple, mais un peu rébarbatif.

Schéma de connexion

Je vais reprendre le schéma précédent, mais je vais connecter chaque broche de l'afficheur à une sortie de la carte Arduino. Comme ceci :





Vous voyez donc que chaque LED de l'afficheur va être commandée séparément les unes des autres. Il n'y a rien de plus à faire, si ce n'est qu'à programmer…

Le programme

L'objectif du programme va être d'afficher un chiffre. Eh bien… c'est partit ! Quoi ?! Vous voulez de l'aide ? o_O Ben je vous ai déjà tout dit y'a plus qu'à faire. En plus vous avez un tableau avec lequel vous pouvez vous aider pour afficher votre chiffre. Cherchez, je vous donnerais la solution ensuite.

```
1  /* On assigne chaque LED à une broche de l'arduino */
2  const int A = 2;
3  const int B = 3;
4  const int C = 4;
```

```
5  const int D = 5;
6  const int E = 6;
7  const int F = 7;
8  const int G = 8;
9  //notez que l'on ne gère pas l'affichage du point, mais vous pouvez le rajouter si c
10
11 void setup()
12 {
13     //définition des broches en sortie
14     pinMode(A, OUTPUT);
15     pinMode(B, OUTPUT);
16     pinMode(C, OUTPUT);
17     pinMode(D, OUTPUT);
18     pinMode(E, OUTPUT);
19     pinMode(F, OUTPUT);
20     pinMode(G, OUTPUT);
21
22     //mise à l'état HAUT de ces sorties pour éteindre les LED de l'afficheur
23     digitalWrite(A, HIGH);
24     digitalWrite(B, HIGH);
25     digitalWrite(C, HIGH);
26     digitalWrite(D, HIGH);
27     digitalWrite(E, HIGH);
28     digitalWrite(F, HIGH);
29     digitalWrite(G, HIGH);
30 }
31
32 void loop()
33 {
34     //affichage du chiffre 5, d'après le tableau précédent
35     digitalWrite(A, LOW);
36     digitalWrite(B, HIGH);
37     digitalWrite(C, LOW);
38     digitalWrite(D, LOW);
39     digitalWrite(E, HIGH);
40     digitalWrite(F, LOW);
41     digitalWrite(G, LOW);
42 }
```

Vous le voyez par vous-même, c'est un code hyper simple. Essayez de le bidouiller pour afficher des messages, par exemple, en utilisant les fonctions introduisant le temps. Ou bien compléter ce code pour afficher tous les chiffres, en fonction d'une variable définie au départ (ex: var = 1, affiche le chiffre 1 ; etc.).

Techniques d'affichage

Vous vous en doutez peut-être, lorsque l'on veut utiliser plusieurs afficheur il va nous falloir beaucoup de broches. Imaginons, nous voulons afficher un nombre entre 0 et 99, il nous faudra utiliser deux afficheurs avec $2 * 7 = 14$ broches connectées sur la carte Arduino. Rappel : une carte Arduino UNO possède… 14 broches entrées/sorties classiques. Si on ne fais rien d'autre que d'utiliser les afficheurs, cela ne nous gène pas, cependant, il est fort probable que vous serez amener à utiliser d'autres entrées avec votre carte Arduino. Mais si on ne libère pas de place vous serez embêté. Nous allons donc voir deux techniques qui, une fois cumulées, vont nous permettre d'utiliser seulement 4 broches pour obtenir le même résultat qu'avec 14 broches !

Les décodeurs “4 bits -> 7 segments”

La première technique que nous allons utiliser met en œuvre un circuit intégré. Vous vous souvenez quand je vous ai parlé de ces bêtes là ? Oui, c’est le même type que le microcontrôleur de la carte Arduino. Cependant, le circuit que nous allons utiliser ne fait pas autant de choses que celui sur votre carte Arduino.

Décodeur BCD -> 7 segments

C’est le nom du circuit que nous allons utiliser. Son rôle est simple. Vous vous souvenez des conversions ? Pour passer du binaire au décimal ? Et bien c’est le moment de vous en servir, donc si vous ne vous rappelez plus de ça, allez revoir un peu le cours. Je disais donc que son rôle est simple. Et vous le constaterez par vous même, il va s’agir de convertir du binaire codé sur 4 bits vers un “code” utilisé pour afficher les chiffres. Ce code correspond en quelque sorte au tableau précédemment évoqué.

Principe du décodeur

Sur un afficheur 7 segments, on peut représenter aisément les chiffres de 0 à 9 (et en insistant un peu les lettres de A à F). En informatique, pour représenter ces chiffres, il nous faut au maximum 4 bits. Comme vous êtes des experts et que vous avez bien lu la partie sur le binaire, vous n’avez pas de mal à le comprendre. $(0000)_2$ fera $(0)_{10}$ et $(1111)_2$ fera $(15)_{10}$ ou $(F)_{16}$. Pour faire 9 par exemple on utilisera les bits 1001. En partant de se constat, des ingénieurs ont inventé un composant au doux nom de “décodeur” ou “driver” 7 segments. Il reçoit sur 4 broches les 4 bits de la valeur à afficher, et sur 7 autres broches ils pilotent les segments pour afficher ladite valeur. Ajouter à cela une broche d’alimentation et une broche de masse on obtient 13 broches ! Et ce n’est pas fini. La plupart des circuits intégrés de type décodeur possède aussi une broche d’activation et une broche pour tester si tous les segments fonctionnent.

Choix du décodeur

Nous allons utiliser le composant nommé MC14543B comme exemple. Tout d’abord, ouvrez ce lien dans un nouvel onglet, il vous menera directement vers le pdf du décodeur :

Datasheet du MC14543B

Les datasheets se composent souvent de la même manière. On trouve tout d’abord un résumé des fonctions du produit puis un schéma de son boîtier. Dans notre cas, on voit qu’il est monté sur un DIP 16 (DIP : Dual Inline Package, en gros “boîtier avec deux lignes de broches”). Si l’on continue, on voit la **table de vérité** faisant le lien entre les signaux d’entrées (INPUT) et les sorties (OUTPUT). On voit ainsi plusieurs choses :

- Si l’on met la broche BI (Blank, n°7) à un, toutes les sorties passent à zéro. En effet, comme son nom l’indique cette broche sert à effacer l’état de l’afficheur. Si vous ne voulez pas l’utiliser il faut donc la connecter à la masse pour la désactiver.
- Les entrées A, B, C et D (broches 5,3,2 et 4 respectivement) sont actives à l’état

HAUT. Les sorties elles sont actives à l'état BAS (pour piloter un afficheur à anode commune) **OU** HAUT selon l'état de la broche PH (6). C'est là un gros avantage de ce composant, il peut inverser la logique de la sortie, le rendant alors compatible avec des afficheurs à anode commune (broche PH à l'état 1) ou cathode commune (Ph = 0)

- La broche BI (Blank Input, n°7) sers à inhiber les entrées. On ne s'en servira pas et donc on la mettra à l'état HAUT (+5V)
- LD (n°1) sert à faire une mémoire de l'état des sorties, on ne s'en servira pas ici
- Enfin, les deux broches d'alimentation sont la 8 (GND/VSS, masse) et la 16 (VCC, +5V)

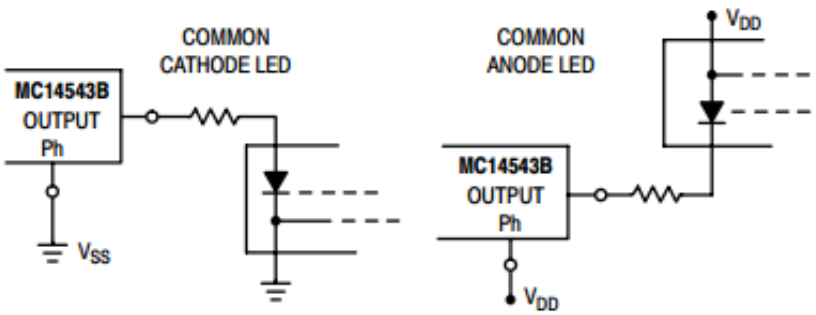
N'oubliez pas de mettre des résistances de limitations de courant entre chaque segment et la broche de signal du circuit!

Fonctionnement

C'est bien beau tout ça mais comment je lui dis au décodeur d'afficher le chiffre 5 par exemple ?

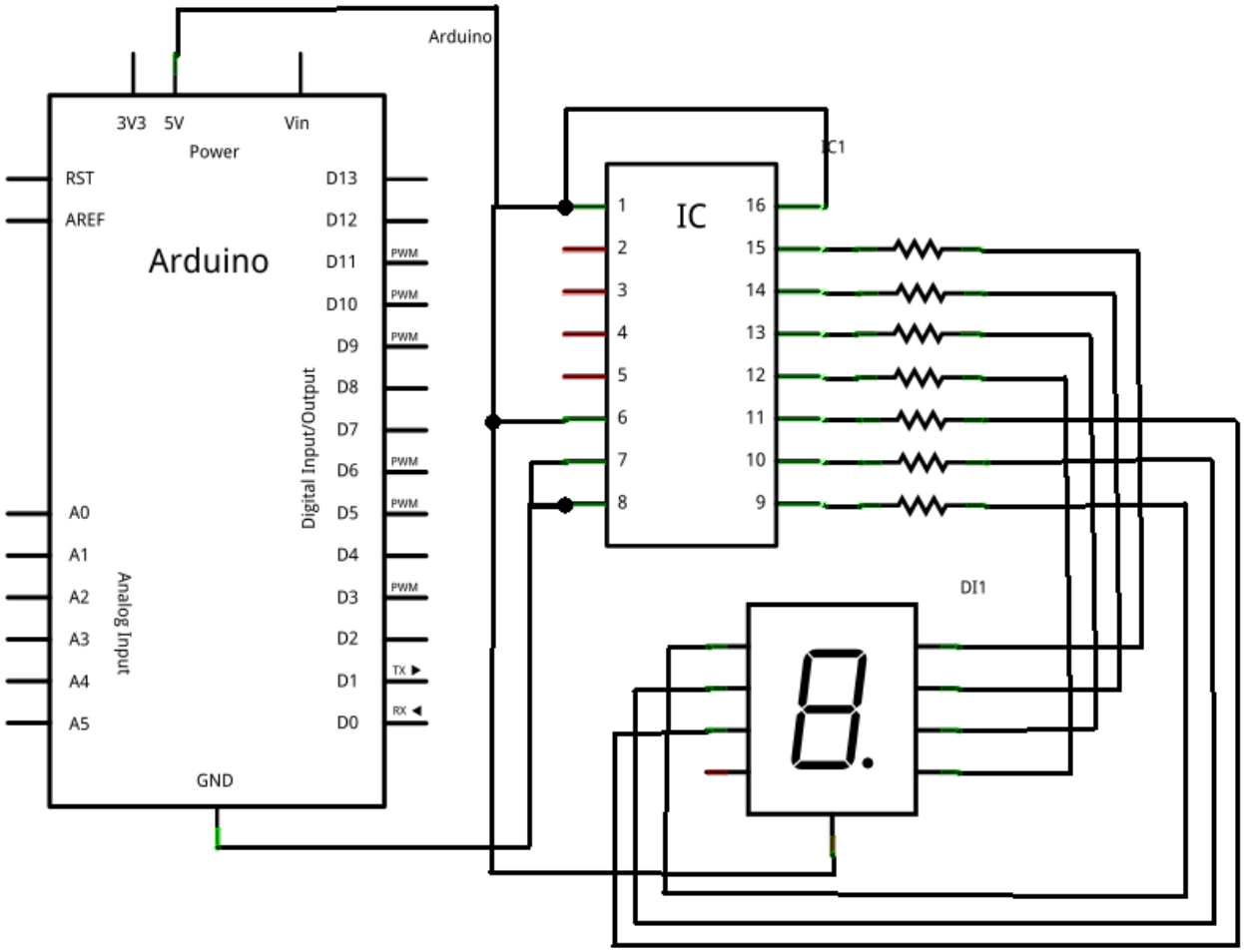
Il suffit de regarder le datasheet et sa table de vérité (c'est le tableau avec les entrées et les sorties). Ce que reçoit le décodeur sur ses entrées (A, B, C et D) défini les états de ses broches de sortie (a,b,c,d,e,f et g). C'est tout ! Donc, on va donner un code binaire sur 4 bits à notre décodeur et en fonction de ce code, le décodeur affichera le caractère voulu. En plus le fabricant est sympa, il met à disposition des notes d'applications à la page 6 pour bien brancher le composant :

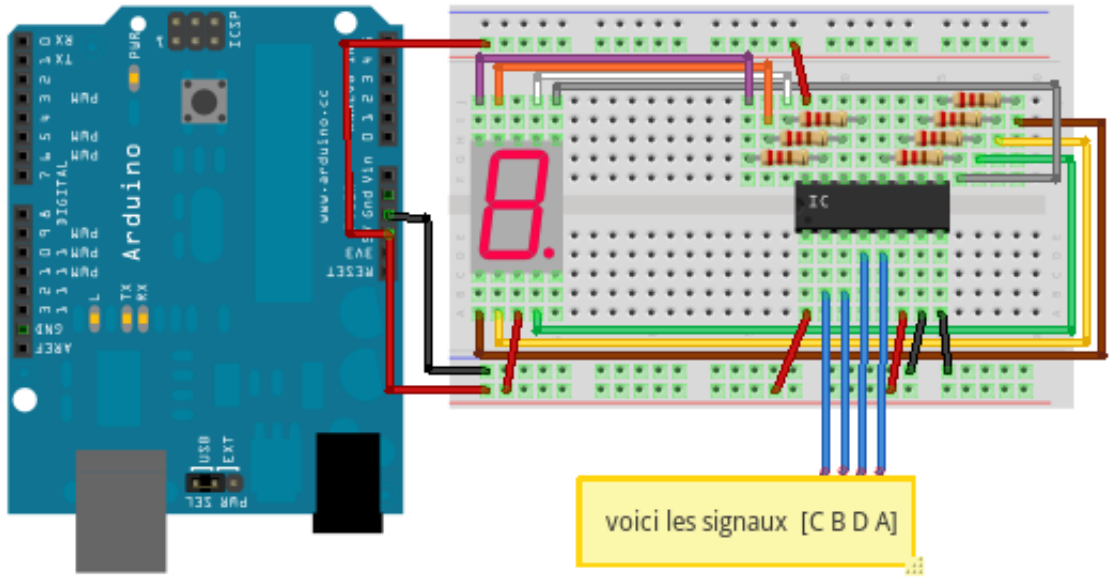
LIGHT EMITTING DIODE (LED) READOUT



NOTE: Bipolar transistors may be added for gain (for $V_{DD} \leq 10$ V or $I_{out} \geq 10$ mA).

On voit alors qu'il suffit simplement de brancher la résistance entre le CI et les segments et s'assurer que PH à la bonne valeur et c'est tout ! En titre d'exercice afin de vous permettre de mieux comprendre, je vous propose de changer les états des entrées A, B, C et D du décodeur pour observer ce qu'il affiche. Après avoir réaliser votre schéma, regarder s'il correspond avec celui présent dans cette balise secrète. Cela vous évitera peut-être un mauvais branchement, qui sait ?





L’affichage par alternance

La seconde technique est utilisée dans le cas où l’on veut faire un affichage avec plusieurs afficheurs. Elle utilise le phénomène de persistance rétinienne. Pour faire simple, c’est grâce à cela que le cinéma vous parait fluide. On change une image toutes les 40 ms et votre œil n’a pas le temps de le voir, donc les images semble s’enchainer sans transition. Bref… Ici, la même stratégie sera utilisée. On va allumer un afficheur un certain temps, puis nous allumerons l’autre en éteignant le premier. Cette action est

assez simple à réaliser, mais nécessite l'emploi de deux broche supplémentaires, de quatre autres composants et d'un peu de code. Nous l'étudierons un petit peu plus tard, lorsque nous saurons géré un afficheur seul.

Utilisation du décodeur BCD

Nous y sommes, nous allons (enfin) utiliser la carte Arduino pour faire un affichage plus poussé qu'un unique afficheur. Pour cela, nous allons très simplement utiliser le montage précédent composé du décodeur BCD, de l'afficheur 7 segments et bien entendu des résistances de limitations de courant pour les LED de l'afficheur. Je vais vous montrer deux techniques qui peuvent être employées pour faire le programme.

Initialisation

Vous avez l'habitude maintenant, nous allons commencer par définir les différentes broches d'entrées/sorties. Pour débuter (et conformément au schéma), nous utiliserons seulement 4 broches, en sorties, correspondantes aux entrées du décodeur 7 segments. Voici le code pouvant traduire cette explication :

```
1  const int bit_A = 2;
2  const int bit_B = 3;
3  const int bit_C = 4;
4  const int bit_D = 5;
5
6  void setup()
7  {
8      //on met les broches en sorties
9      pinMode(bit_A, OUTPUT);
10     pinMode(bit_B, OUTPUT);
11     pinMode(bit_C, OUTPUT);
12     pinMode(bit_D, OUTPUT);
13
14     //on commence par écrire le chiffre 0, donc toutes les sorites à l'état bas
15     digitalWrite(bit_A, LOW);
16     digitalWrite(bit_B, LOW);
17     digitalWrite(bit_C, LOW);
18     digitalWrite(bit_D, LOW);
19 }
```

Ce code permet juste de déclarer les quatre broches à utiliser, puis les affectes en sorties. On les met ensuite toutes les quatre à zéro. Maintenant que l'afficheur est prêt, nous allons pouvoir commencer à afficher un chiffre !

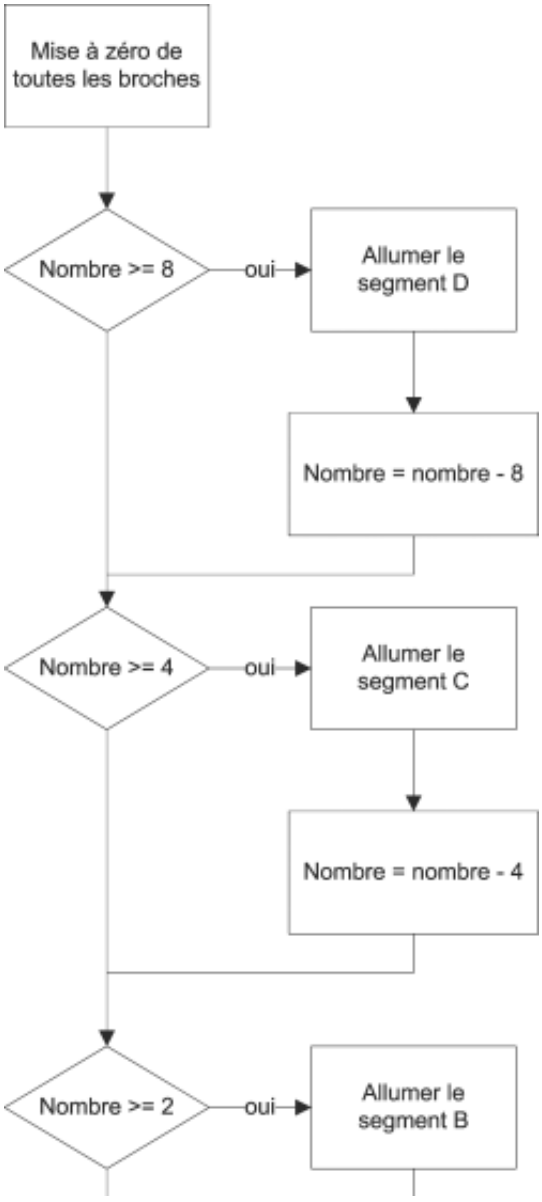
Programme principal

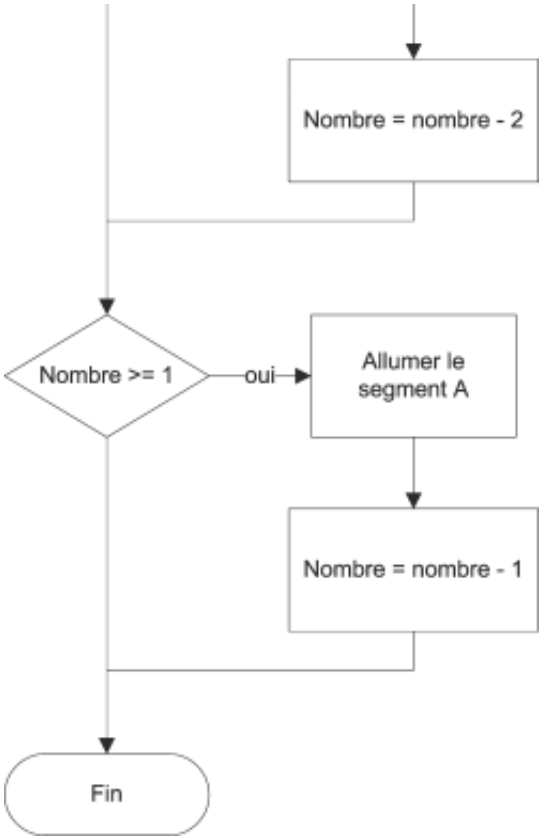
Si tout se passe bien, en ayant la boucle vide pour l'instant vous devriez voir un superbe 0 sur votre afficheur. Nous allons maintenant mettre en place un petit programme pour afficher les nombres de 0 à 9 en les incrémentant (à partir de 0) toutes les secondes. C'est donc un compteur. Pour cela, on va utiliser une boucle, qui comptera de 0 à 9. Dans cette boucle, on exécutera appellera la fonction `afficher()` qui s'occupera donc de l'affichage (belle démonstration de ce qui est une évidence  ).

```
void loop()
{
    char i=0; //variable "compteur"
    for(i=0; i<10; i++)
    {
        afficher(i); //on appel la fonction d'affichage
        delay(1000); //on attend 1 seconde
    }
}
```

Fonction d'affichage

Nous touchons maintenant au but ! Il ne nous reste plus qu'à réaliser la fonction d'affichage pour pouvoir convertir notre variable en chiffre sur l'afficheur. Pour cela, il existe différentes solutions. Nous allons en voir ici une qui est assez simple à mettre en œuvre mais qui nécessite de bien être comprise. Dans cette méthode, on va faire des opérations mathématiques (tout de suite c'est moins drôle 😅) successives pour déterminer quels bits mettre à l'état haut. Rappelez-vous, nous avons quatre broches à notre disposition, avec chacune un poids différent (8, 4, 2 et 1). En combinant ces différentes broches ont peu obtenir n'importe quel nombre de 0 à 15. Voici une démarche mathématique envisageable :





On peut coder cette méthode de manière assez simple et direct, en suivant cet organigramme :

```
1  //fonction écrivant sur un seul afficheur
2  void afficher(char chiffre)
3  {
4      //on met à zéro tout les bits du décodeur
5      digitalWrite(bit_A, LOW);
6      digitalWrite(bit_B, LOW);
7      digitalWrite(bit_C, LOW);
8      digitalWrite(bit_D, LOW);
9
10     //On allume les bits nécessaires
11     if(chiffre >= 8)
12     {
13         digitalWrite(bit_D, HIGH);
14         chiffre = chiffre - 8;
15     }
16     if(chiffre >= 4)
17     {
18         digitalWrite(bit_C, HIGH);
19         chiffre = chiffre - 4;
20     }
21     if(chiffre >= 2)
22     {
23         digitalWrite(bit_B, HIGH);
24         chiffre = chiffre - 2;
25     }
26     if(chiffre >= 1)
27     {
28         digitalWrite(bit_A, HIGH);
29         chiffre = chiffre - 1;
30     }
31 }
```

Quelques explications s'imposent… Le code gérant l'affichage réside sur les valeurs binaires des chiffres. Rappelons les valeurs binaires des chiffres :

Chiffre	DCBA
0	$(0000)_2$
1	$(0001)_2$
2	$(0010)_2$
3	$(0011)_2$
4	$(0100)_2$
5	$(0101)_2$
6	$(0110)_2$
7	$(0111)_2$
8	$(1000)_2$
9	$(1001)_2$

D'après ce tableau, si on veut le chiffre 8, on doit allumer le segment D, car 8 s'écrit $(1000)_2$ ayant pour segment respectif DCBA. Soit D=1, C=0, B=0 et A=0. En suivant cette logique, on arrive à déterminer les entrées du décodeur qui sont à mettre à l'état HAUT ou BAS. D'une manière plus lourde, on aurait pu écrire un code ressemblant à ça :

```
1  //fonction écrivant sur un seul afficheur
2  void afficher(char chiffre)
3  {
4      switch(chiffre)
5      {
6      case 0 :
7          digitalWrite(bit_A, LOW);
8          digitalWrite(bit_B, LOW);
9          digitalWrite(bit_C, LOW);
10         digitalWrite(bit_D, LOW);
11         break;
12     case 1 :
13         digitalWrite(bit_A, HIGH);
14         digitalWrite(bit_B, LOW);
15         digitalWrite(bit_C, LOW);
16         digitalWrite(bit_D, LOW);
17         break;
18     case 2 :
19         digitalWrite(bit_A, LOW);
20         digitalWrite(bit_B, HIGH);
21         digitalWrite(bit_C, LOW);
22         digitalWrite(bit_D, LOW);
23         break;
24     case 3 :
25         digitalWrite(bit_A, HIGH);
26         digitalWrite(bit_B, HIGH);
27         digitalWrite(bit_C, LOW);
28         digitalWrite(bit_D, LOW);
29         break;
30     case 4 :
31         digitalWrite(bit_A, LOW);
```

```
32          digitalWrite(bit_B, LOW);
33          digitalWrite(bit_C, HIGH);
34          digitalWrite(bit_D, LOW);
35          break;
36      case 5 :
37          digitalWrite(bit_A, HIGH);
38          digitalWrite(bit_B, LOW);
39          digitalWrite(bit_C, HIGH);
40          digitalWrite(bit_D, LOW);
41          break;
42      case 6 :
43          digitalWrite(bit_A, LOW);
44          digitalWrite(bit_B, HIGH);
45          digitalWrite(bit_C, HIGH);
46          digitalWrite(bit_D, LOW);
47          break;
48      case 7 :
49          digitalWrite(bit_A, HIGH);
50          digitalWrite(bit_B, HIGH);
51          digitalWrite(bit_C, HIGH);
52          digitalWrite(bit_D, LOW);
53          break;
54      case 8 :
55          digitalWrite(bit_A, LOW);
56          digitalWrite(bit_B, LOW);
57          digitalWrite(bit_C, LOW);
58          digitalWrite(bit_D, HIGH);
59          break;
60      case 9 :
61          digitalWrite(bit_A, HIGH);
62          digitalWrite(bit_B, LOW);
63          digitalWrite(bit_C, LOW);
64          digitalWrite(bit_D, HIGH);
65          break;
66      }
67  }
```

Mais, c'est bien trop lourd à écrire. Enfin c'est vous qui voyez. 🙂

Utiliser plusieurs afficheurs

Maintenant que nous avons affiché un chiffre sur un seul afficheur, nous allons pouvoir apprendre à en utiliser plusieurs (avec un minimum de composants en plus !). Comme expliqué précédemment, la méthode employée ici va reposer sur le principe de la persistance rétinienne, qui donnera *l'impression* que les deux afficheurs fonctionnent en *même temps*.

Problématique

Nous souhaiterions utiliser deux afficheurs, mais nous ne disposons que de seulement 6 broches sur notre Arduino, le reste des broches étant utilisé pour une autre application. Pour réduire le nombre de broches, on peut d'ores et déjà utilisé un décodeur BCD, ce qui nous ferait 4 broches par afficheurs, soit 8 broches au total. Bon, ce n'est toujours pas ce que l'on veut. Et si on connectait les deux afficheurs ensemble, en parallèle, sur les sorties du décodeur ? Oui mais dans ce cas, on ne pourrait pas

afficher des chiffres différents sur chaque afficheur. Tout à l'heure, je vous ai parlé de *commutation*. Oui, la seule solution qui soit envisageable est d'allumer un afficheur et d'éteindre l'autre tout en les connectant ensemble sur le même décodeur. Ainsi un afficheur s'allume, il affiche le chiffre voulu, puis il s'éteint pour que l'autre puisse s'allumer à son tour. Cette opération est en fait un clignotement de chaque afficheur par alternance.

Un peu d'électronique…

Pour faire commuter nos deux afficheurs, vous allez avoir besoin d'un nouveau composant, j'ai nommé : le **transistor** !

> Transistor ? J'ai entendu dire qu'il y en avait plusieurs milliards dans nos ordinateurs ?

Et c'est tout à fait vrai. Des transistors, il en existe de différents types et pour différentes applications : amplification de courant/tension, commutation, etc. répartis dans plusieurs familles. Bon je ne vais pas faire trop de détails, si vous voulez en savoir plus, allez lire la première partie de ce chapitre (*lien à rajouter, en attente de la validation du chapitre en question*).

Le transistor bipolaire : présentation

Je le disais, je ne vais pas faire de détails. On va voir comment fonctionne un transistor bipolaire selon les besoins de notre application, à savoir, faire commuter les afficheurs. Un transistor, cela ressemble à ça :

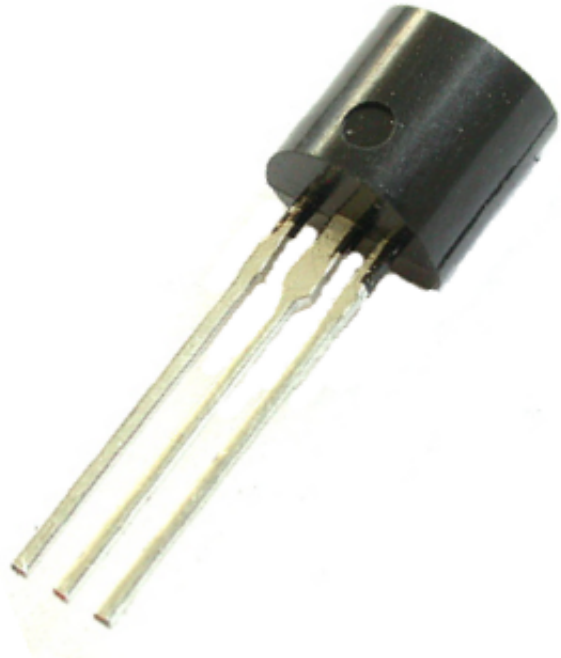
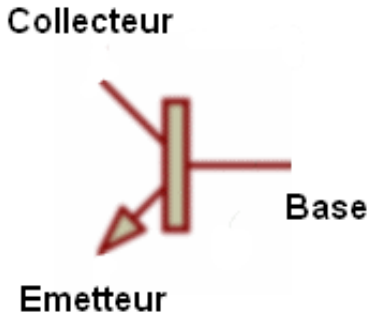


Photo d'un transistor

Pour notre application, nous allons utiliser des **transistors bipolaires**. Je vais vous expliquer comment cela fonctionne. Déjà, vous pouvez observer qu'un transistor possède trois pattes. Cela n'est pas de la moindre importance, au contraire il s'agit là d'une chose essentielle ! En fait, le transistor bipolaire à une broche d'entrée (**collecteur**), une broche de sortie (**émetteur**) et une broche de commande (**base**). Son

symbole est le suivant :



Ce symbole est celui d'un **transistor bipolaire de type NPN**. Il en existe qui sont de **type PNP**, mais ils sont beaucoup moins utilisés que les NPN. Quoi qu'il en soit, nous n'utiliserons que des transistors NPN dans ce chapitre.

Fonctionnement en commutation du transistor bipolaire

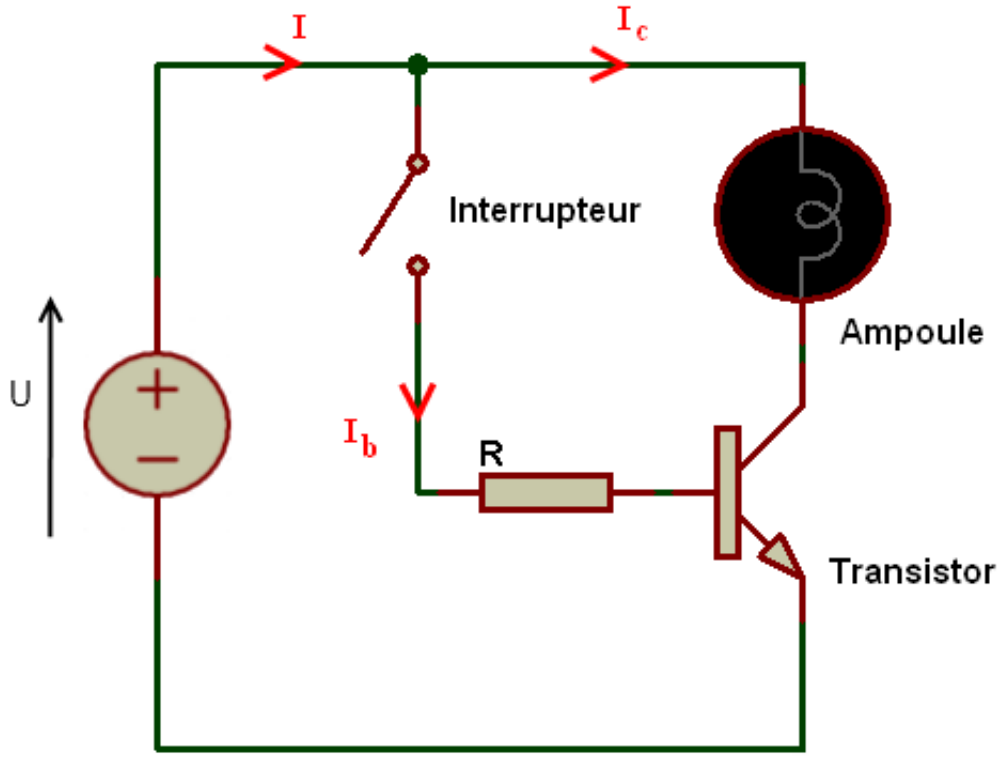
Pour faire simple, le transistor bipolaire NPN (c'est la dernière fois que je précise ce point) est un **interrupteur commandé en courant**.

Ceci est une présentation très vulgarisée et simplifiée sur le transistor pour l'utilisation que nous en ferons ici. Les usages et possibilités des transistors sont très nombreux et ils mériteraient un big-tuto à eux seuls ! Si vous voulez plus d'informations, rendez-vous sur le cours sur l'électronique ou approfondissez en cherchant des tutoriels sur le web. 😉

C'est tout ce qu'il faut savoir, pour ce qui est du fonctionnement. Après, on va voir ensemble comment l'utiliser et sans le faire griller ! 😁

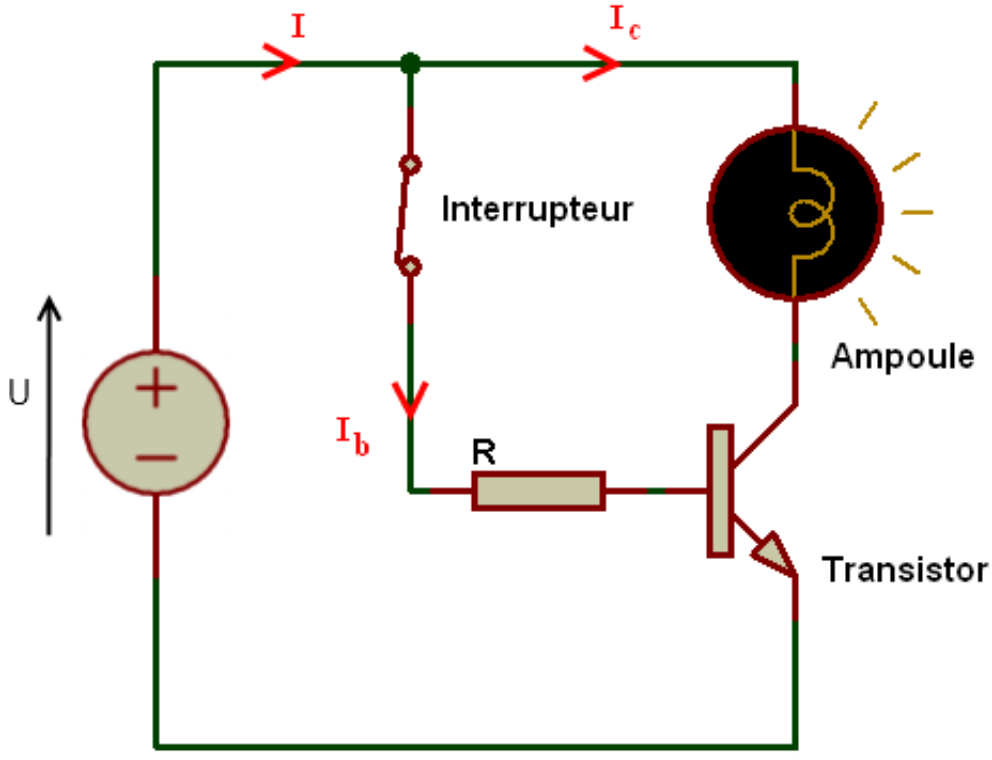
Utilisation générale

On peut utiliser notre transistor de deux manières différentes (pour notre application toujours, mais on peut bien évidemment utiliser le transistor avec beaucoup plus de flexibilités). A commencer par le câblage :



Câblage du transistor en commutation

Dans le cas présent, le collecteur (qui est l'entrée du transistor) se trouve être après l'ampoule, elle-même connectée à l'alimentation. L'émetteur (broche où il y a la flèche) est relié à la masse du montage. Cette disposition est "universelle", on ne peut pas inverser le sens de ces broches et mettre le collecteur à la place de l'émetteur et vice versa. Sans quoi, le montage ne fonctionnerait pas. Pour le moment, l'ampoule est éteinte car le transistor ne conduit pas. On dit qu'il est **bloqué** et empêche donc le courant I_C de circuler à travers l'ampoule. Soit $I_C = 0$ car $I_B = 0$. A présent, appuyons sur l'interrupteur :



L'ampoule est

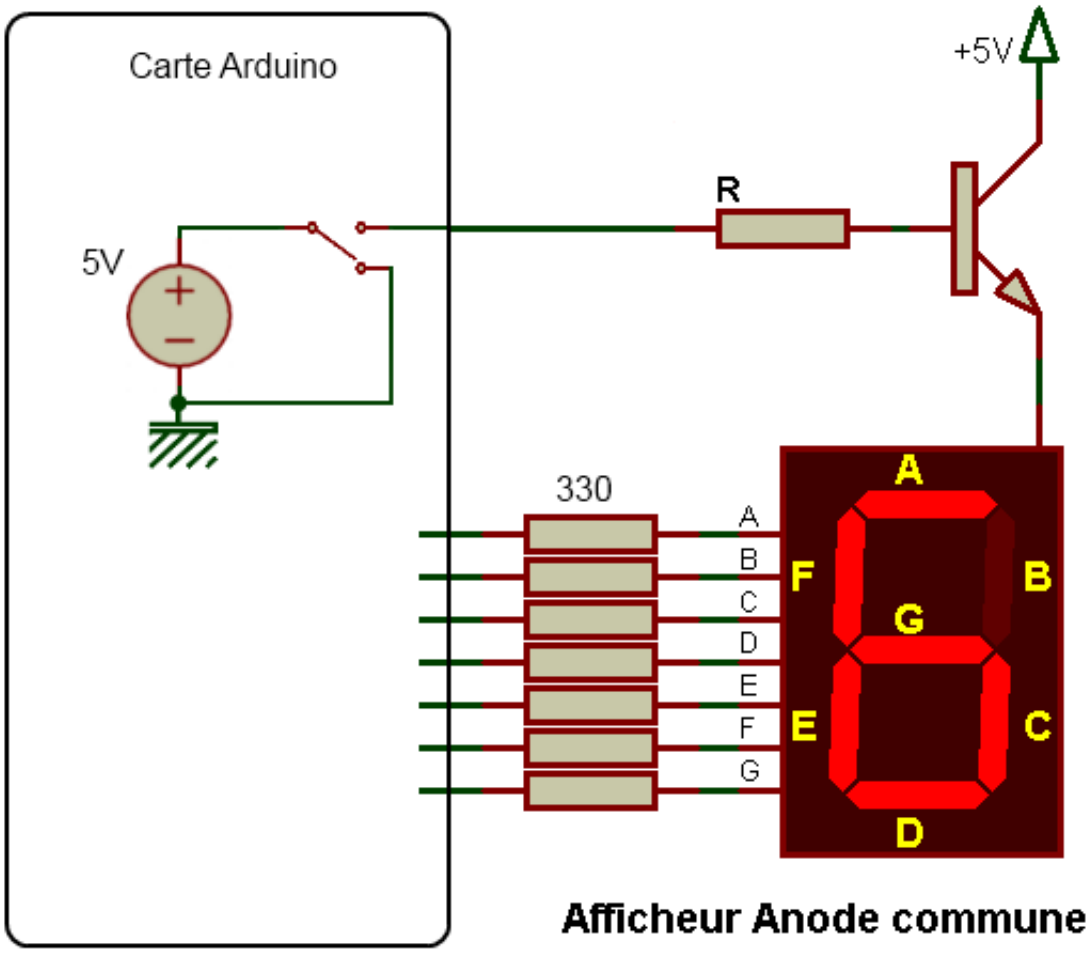
allumée

Que se passe-t-il ? Eh bien la base du transistor, qui était jusqu'à présent "en l'air", est parcourue par un courant électrique. Cette cause à pour conséquence de rendre le transistor **passant** ou **saturé** et permet au courant de s'établir à travers l'ampoule. Soit $I_C \neq 0$ car $I_B \neq 0$.

La résistance sur la base du transistor permet de le protéger des courants trop forts. Plus la résistance est de faible valeur, plus l'ampoule sera lumineuse. A l'inverse, une résistance trop forte sur la base du transistor pourra l'empêcher de conduire et de faire s'allumer l'ampoule. Rassurez_vous, je vous donnerais les valeurs de résistances à utiliser.

Utilisation avec nos afficheurs

Voyons un peu comment on va pouvoir utiliser ce transistor avec notre Arduino. La carte Arduino est en fait le générateur de tension (schéma précédent) du montage. Elle va définir si sa sortie est de 0V (transistor bloqué) ou de 5V (transistor saturé). Ainsi, on va pouvoir allumer ou éteindre les afficheurs. Voilà le modèle équivalent de la carte Arduino et de la commande de l'afficheur :

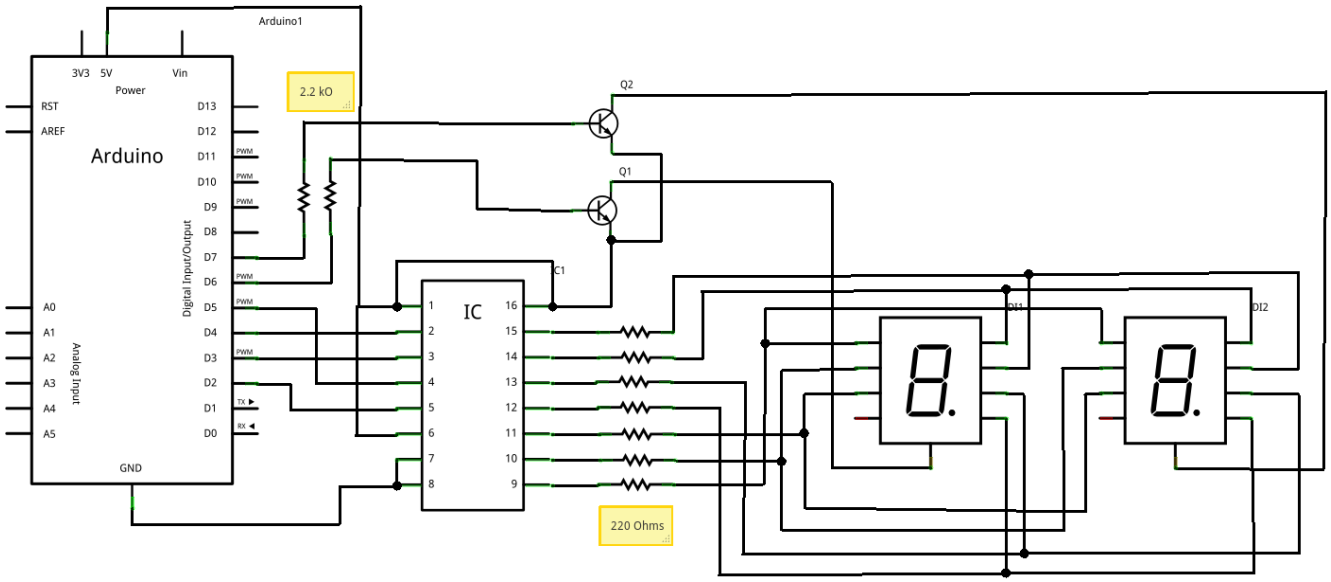


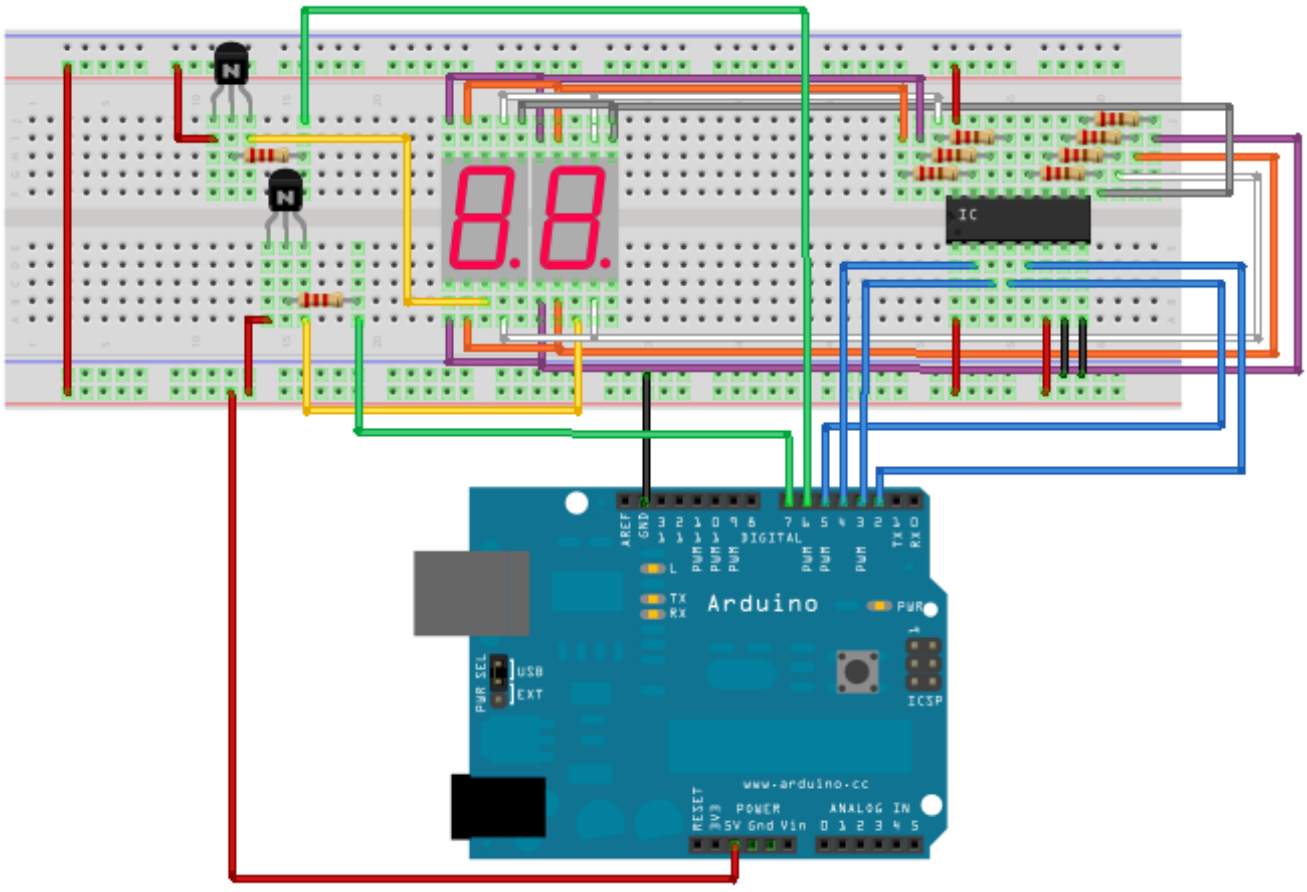
LA carte Arduino va soit mettre à la masse la base du transistor, soit la mettre à +5V. Dans le premier cas, il sera bloqué et l'afficheur sera éteint, dans le second il sera saturé et l'afficheur allumé. Il en est de même pour chaque broche de l'afficheur. Elles

seront au +5V ou à la masse selon la configuration que l'on aura définie dans le programme.

Schéma final

Et comme vous l'attendez surement depuis tout à l'heure, voici le schéma tant attendu (nous verrons juste après comment programmer ce nouveau montage) !





Quelques détails techniques

- Dans notre cas (et je vous passe les détails vraiment techniques et calculatoires),

la résistance sur la base du transistor sera de $2.2k\Omega$ (si vous n'avez pas cette valeur, elle pourra être de $3.3k\Omega$, ou encore de $3.9k\Omega$, voir même de $4.7k\Omega$).
- Les transistors seront des transistors bipolaires NPN de référence 2N2222, ou bien un équivalent qui est le BC547. Il en faudra deux donc.
- Le décodeur BCD est le même que précédemment (ou équivalent).

Et avec tout ça, on est prêt pour programmer ! 😀

…et de programmation

Nous utilisons deux nouvelles broches servant à piloter chacun des interrupteurs (transistors). Chacune de ces broches doivent donc être déclarées en global (pour son numéro) puis régler comme sortie. Ensuite, il ne vous restera plus qu'à alimenter chacun des transistors au bon moment pour allumer l'afficheur souhaité. En synchronisant l'allumage avec la valeur envoyé au décodeur, vous afficherez les nombres souhaités comme bon vous semble. Voici un exemple de code complet, de la fonction setup() jusqu'à la fonction d'affichage. Ce code est commenté et vous ne devriez donc avoir aucun mal à le comprendre ! Ce programme est un compteur sur 2 segments, il compte donc de 0 à 99 et recommence au début dès qu'il a atteint 99. La vidéo se trouve juste après ce code.

```
1  //définition des broches du décodeur 7 segments (vous pouvez changer les numéros si
2  const int bit_A = 2;
3  const int bit_B = 3;
4  const int bit_C = 4;
5  const int bit_D = 5;
6
7  //définitions des broches des transistors pour chaque afficheur (dizaines et unités
8  const int alim_dizaine = 6;
9  const int alim_unite = 7;
10
11 void setup()
12 {
13     //Les broches sont toutes des sorties
14     pinMode(bit_A, OUTPUT);
15     pinMode(bit_B, OUTPUT);
16     pinMode(bit_C, OUTPUT);
17     pinMode(bit_D, OUTPUT);
18     pinMode(alim_dizaine, OUTPUT);
19     pinMode(alim_unite, OUTPUT);
20
21     //Les broches sont toutes mises à l'état bas
22     digitalWrite(bit_A, LOW);
23     digitalWrite(bit_B, LOW);
24     digitalWrite(bit_C, LOW);
25     digitalWrite(bit_D, LOW);
26     digitalWrite(alim_dizaine, LOW);
27     digitalWrite(alim_unite, LOW);
28 }
29
30 void loop() //fonction principale
31 {
32     for(char i = 0; i<100; i++) //boucle qui permet de compter de 0 à 99 (= 100 val
33     {
34         afficher_nombre(i); //appel de la fonction affichage avec envoi du nombre à
35     }
36 }
```

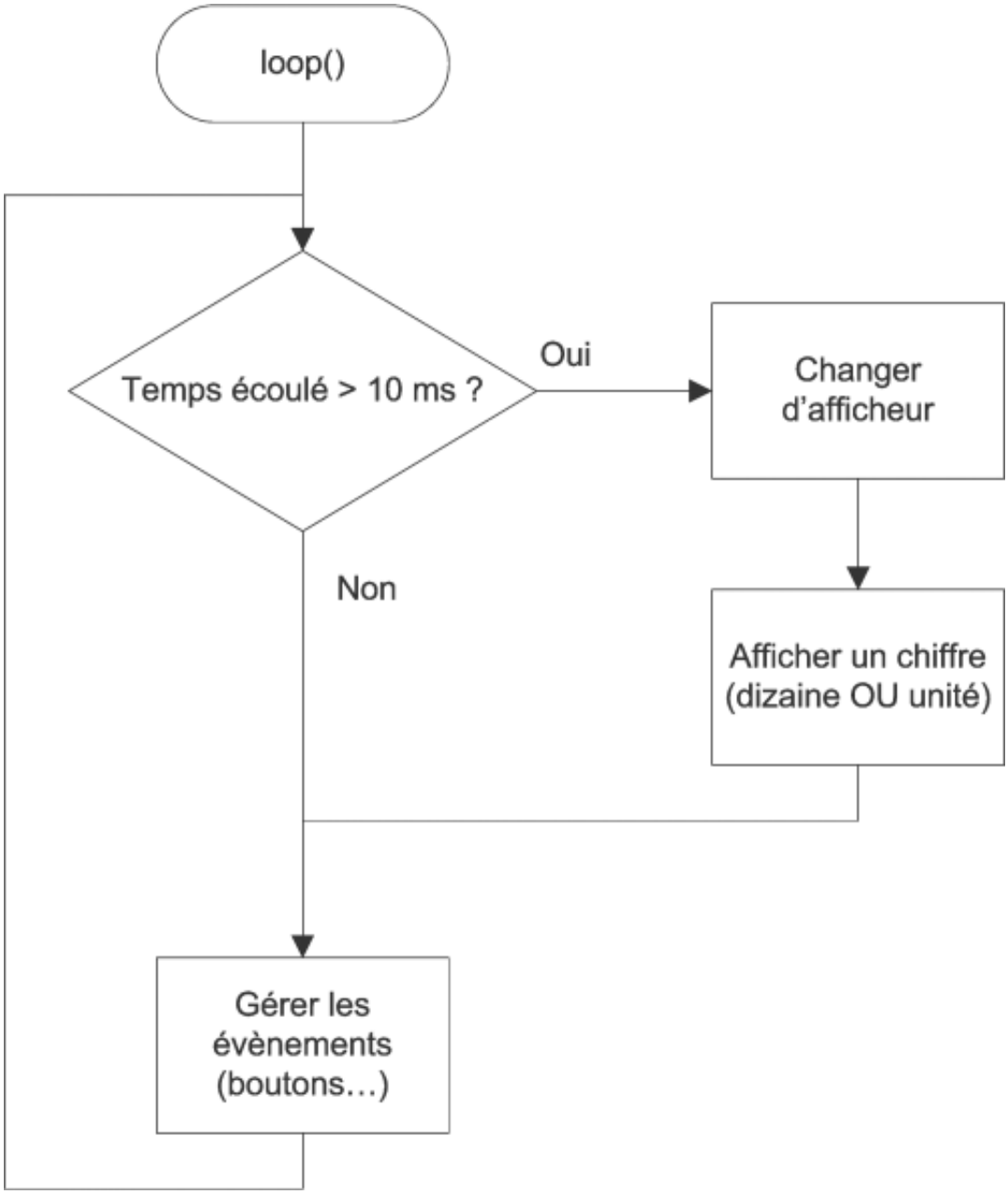
```
37
38  //fonction permettant d'afficher un nombre sur deux afficheurs
39  void afficher_nombre(char nombre)
40  {
41      long temps; //variable utilisée pour savoir le temps écoulé...
42      char unite = 0, dizaine = 0; //variable pour chaque afficheur
43
44      if(nombre > 9) //si le nombre reçu dépasse 9
45      {
46          dizaine = nombre / 10; //on récupère les dizaines
47      }
48
49      unite = nombre - (dizaine*10); //on récupère les unités
50
51      temps = millis(); //on récupère le temps courant
52
53      // tant qu'on a pas affiché ce chiffre pendant au moins 500 millisecondes
54      // permet donc de pouvoir lire le nombre affiché
55      while((millis()-temps) < 500)
56      {
57          //on affiche le nombre
58
59          //d'abord les dizaines pendant 10 ms
60          digitalWrite(alim_dizaine, HIGH); /* le transistor de l'afficheur des dizai
61  donc l'afficheur est allumé */
62          afficher(dizaine); //on appel la fonction qui permet d'afficher le chiffre
63          digitalWrite(alim_unite, LOW); // l'autre transistor est bloqué et l'affich
64          delay(10);
65
66          //puis les unités pendant 10 ms
67          digitalWrite(alim_dizaine, LOW); //on éteint le transistor allumé
68          afficher(unite); //on appel la fonction qui permet d'afficher le chiffre un
69          digitalWrite(alim_unite, HIGH); //et on allume l'autre
70          delay(10);
71      }
72  }
73
74  //fonction écrivant sur un seul afficheur
75  //on utilise le même principe que vu plus haut
76  void afficher(char chiffre)
77  {
78      digitalWrite(bit_A, LOW);
79      digitalWrite(bit_B, LOW);
80      digitalWrite(bit_C, LOW);
81      digitalWrite(bit_D, LOW);
82
83      if(chiffre >= 8)
84      {
85          digitalWrite(bit_D, HIGH);
86          chiffre = chiffre - 8;
87      }
88      if(chiffre >= 4)
89      {
90          digitalWrite(bit_C, HIGH);
91          chiffre = chiffre - 4;
92      }
93      if(chiffre >= 2)
94      {
95          digitalWrite(bit_B, HIGH);
96          chiffre = chiffre - 2;
97      }
```

```
98      if(chiffre >= 1)
99      {
100         digitalWrite(bit_A, HIGH);
101         chiffre = chiffre - 1;
102     }
103 }
104
105 //le code est terminé !
```

Voilà donc la vidéo présentant le résultat final :

Contraintes des évènements

Comme vous l'avez vu juste avant, afficher de manière alternative n'est pas trop difficile. Cependant, vous avez surement remarqué, nous avons utilisé des fonctions bloquantes (delay). Si jamais un évènement devait arriver pendant ce temps, nous aurions beaucoup de chance de le rater car il pourrait arriver "pendant" un délai d'attente pour l'affichage. Pour parer à cela, je vais maintenant vous expliquer une autre méthode, préférable, pour faire de l'affichage. Elle s'appuiera sur l'utilisation de la fonction millis(), qui nous permettra de générer une boucle de rafraîchissement de l'affichage. Voici un organigramme qui explique le principe :



Comme vous pouvez le voir, il n'y a plus de fonction qui "attend". Tout se passe de manière continue, sans qu'il n'y ai jamais de pause. Ainsi, aucun évènement ne sera raté (en théorie, un évènement trèèèèèès rapide pourra toujours passer inaperçu). Voici un exemple de programmation de la boucle principal (suivi de ses fonctions annexes) :

```
1  bool afficheur = false; //variable pour le choix de l'afficheur
2
3  // --- setup() ---
4
5  void loop()
6  {
7      //gestion du rafraichissement
8      //si ça fait plus de 10 ms qu'on affiche, on change de 7 segments (alternance un
9      if((millis() - temps) > 10)
10     {
11         //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou diz
12         afficheur = !afficheur;
13         //on affiche la valeur sur l'afficheur
14         //afficheur : true->dizaines, false->unités
15         afficher_nombre(valeur, afficheur);
```

```
        temps = millis(); //on met à jour le temps
    }

    //ici, on peut traiter les évènements (bouton...)
}

//fonction permettant d'afficher un nombre
//elle affiche soit les dizaines soit les unités
void afficher_nombre(char nombre, bool afficheur)
{
    char unite = 0, dizaine = 0;
    if(nombre > 9)
        dizaine = nombre / 10; //on recupere les dizaines
    unite = nombre - (dizaine*10); //on recupere les unités

    //si "
    if(afficheur)
    {
        //on affiche les dizaines
        digitalWrite(alim_unite, LOW);
        afficher(dizaine);
        digitalWrite(alim_dizaine, HIGH);
    }
    else // égal à : else if(!afficheur)
    {
        //on affiche les unités
        digitalWrite(alim_dizaine, LOW);
        afficher(unite);
        digitalWrite(alim_unite, HIGH);
    }
}

//fonction écrivant sur un seul afficheur
void afficher(char chiffre)
{
    digitalWrite(bit_A, LOW);
    digitalWrite(bit_B, LOW);
    digitalWrite(bit_C, LOW);
    digitalWrite(bit_D, LOW);

    if(chiffre >= 8)
    {
        digitalWrite(bit_D, HIGH);
        chiffre = chiffre - 8;
    }
    if(chiffre >= 4)
    {
        digitalWrite(bit_C, HIGH);
        chiffre = chiffre - 4;
    }
    if(chiffre >= 2)
    {
        digitalWrite(bit_B, HIGH);
        chiffre = chiffre - 2;
    }
    if(chiffre >= 1)
    {
        digitalWrite(bit_A, HIGH);
        chiffre = chiffre - 1;
    }
}
```

Si vous voulez tester le phénomène de persistance rétinienne, vous pouvez changer le temps de la boucle de rafraichissement (ligne 9). Si vous l'augmenter, vous commencerez à vois les afficheurs clignoter. En mettant une valeur d'un peu moins de une seconde vous verrez les afficheurs s'illuminer l'un après l'autre.

Ce chapitre vous a appris à utiliser un nouveau moyen pour afficher des informations avec votre carte Arduino. L'afficheur peut sembler peu utilisé mais en fait de nombreuses applications existe ! (chronomètre, réveil, horloge, compteur de passage, afficheur de score, etc.). Par exemple, il pourra vous servir pour déboguer votre code et afficher la valeur des variables souhaitées…

[Arduino 206] [TP] Parking

Ça y est, une page se tourne avec l'acquisition de nombreuses connaissances de base. C'est donc l'occasion idéale pour faire un (gros 😈) TP qui utilisera l'ensemble de vos connaissances durement acquises. J'aime utiliser les situations de la vie réelle, je vais donc en prendre une pour ce sujet. Je vous propose de réaliser la gestion d'un parking souterrain… RDV aux consignes pour les détails.

Consigne

Après tant de connaissances chacune séparée dans son coin, nous allons pouvoir mettre en œuvre tout ce petit monde dans un TP traitant sur un sujet de la vie courante : les **parkings** !

Histoire

Le maire de zCity à décidé de rentabiliser le parking communal d'une capacité de 99 places (pas une de plus ni de moins). En effet, chaque jour des centaines de zTouristes viennent se promener en voiture et ont besoin de la garer quelque part. Le parking, n'étant pour le moment pas rentable, servira à financer l'entretien de la ville. Pour cela, il faut rajouter au parking existant un afficheur permettant de savoir le nombre de places disponibles en temps réel (le système de paiement du parking ne sera pas traité). Il dispose aussi dans la ville des lumières vertes et rouges signalant un parking complet ou non. Enfin, l'entrée du parking est équipée de deux barrières (une pour l'entrée et l'autre pour la sortie). Chaque entrée de voiture ou sortie génère un signal pour la gestion du nombre de places. Le maire vous a choisi pour vos compétences, votre esprit de créativité et il sait que vous aimez les défis. Vous acceptez évidemment en lui promettant de réussir dans les plus brefs délais !

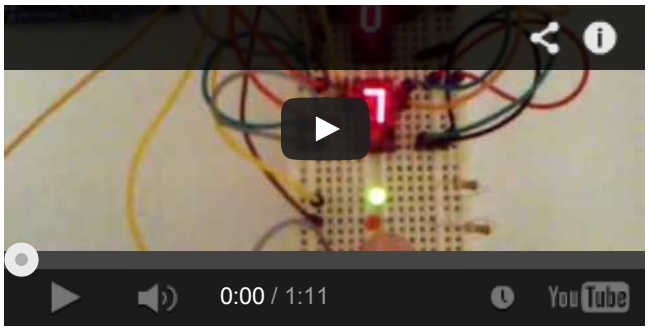
Matériel

Pour mener à bien ce TP voici la liste des courses conseillée :

- Une carte Arduino (évidemment)
- 2 LEDs avec leur résistance de limitations de courant (habituellement 330 Ohms) -> Elles symbolisent les témoins lumineux disposés dans la ville
- 2 boutons (avec 2 résistances de 10 kOhms et 2 condensateurs de 10 nF) -> Ce

sont les “capteurs” d’entrée et de sortie.
- 2 afficheurs 7 segments -> pour afficher le nombre de places disponibles
- 1 décodeur 4 bits vers 7 segments
- 7 résistances de 330 Ohms (pour les 7 segments)
- Une breadboard pour assembler le tout
- Un paquet de fils
- Votre cerveau et quelques doigts…

Voici une vidéo pour vous montrer le résultat attendu par le maire :



Bon courage !

Correction !

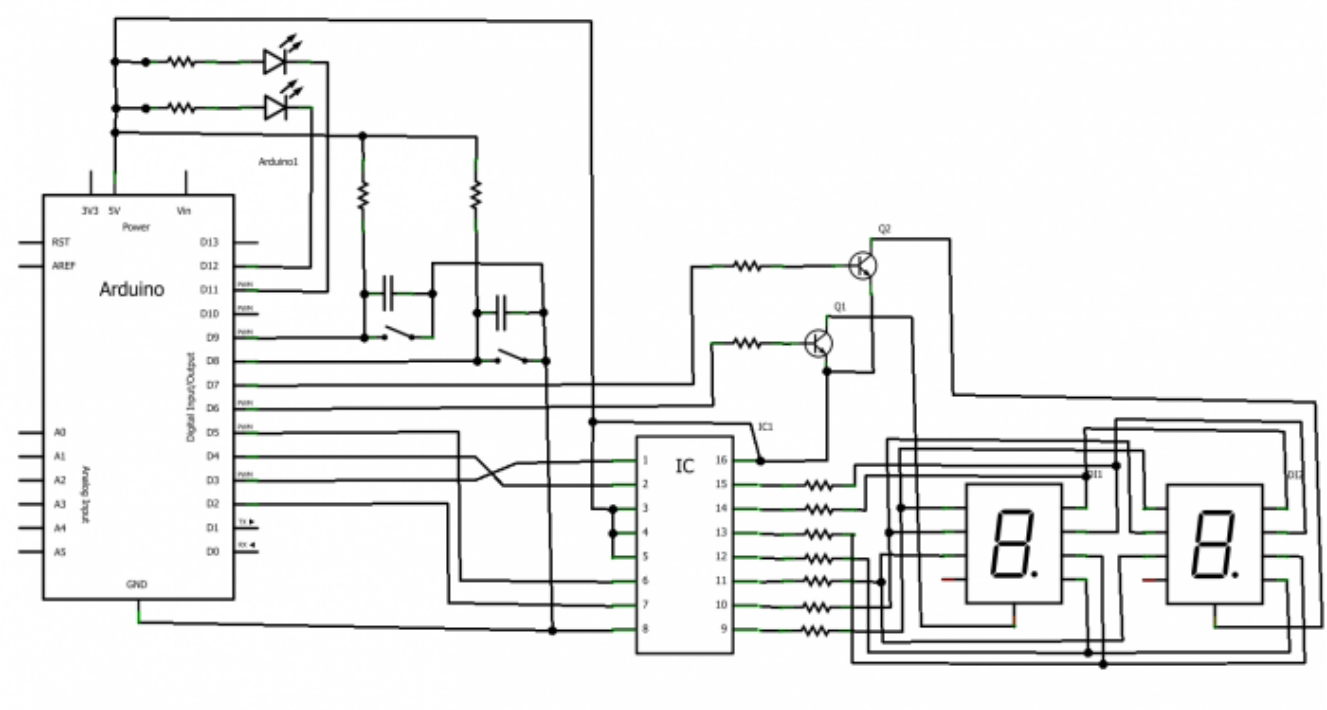
J’espère que tout s’est bien passé pour vous et que le maire sera content de votre travail. Voilà maintenant une correction (parmi tant d’autres, comme souvent en programmation et en électronique). Nous commencerons par voir le schéma électronique, puis ensuite nous rentrerons dans le code.

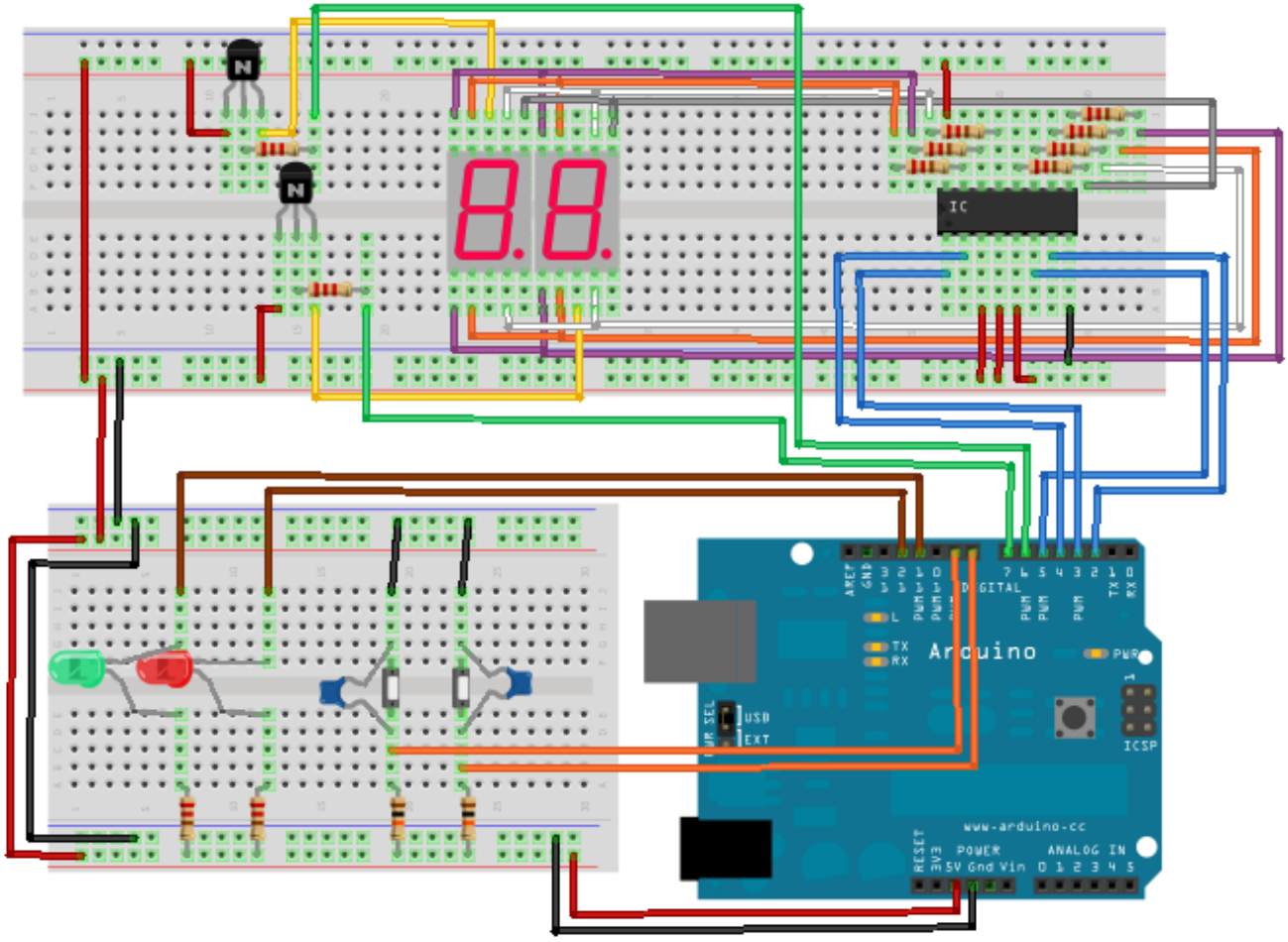
Montage

Le montage électronique est la base de ce qui va nous servir pour réaliser le système. Une fois qu’il est terminé on pourra l’utiliser grâce aux entrées/sorties de la carte Arduino et lui faire faire pleins de choses. Mais ça, vous le savez déjà. Alors ici pas de grand discours, il “suffit” de reprendre les différents blocs vus un par un dans les chapitres précédents et de faire le montage de façon simple.

Schéma

Je vous montre le schéma que j’ai réalisé, il n’est pas absolu et peut différer selon ce que vous avez fait, mais il reprend essentiellement tous les “blocs” (ou mini montages électroniques) que l’on a vus dans les précédents chapitres, en les assemblant de façon logique et ordonnée :





Procédure de montage

Voici l'ordre que j'ai suivi pour réaliser le montage :

- Débrancher la carte Arduino !
- Mettre les boutons
  - Mettre les résistances de pull-up

    - Puis les condensateurs de filtrage
    - Et tirez des fils de signaux jusqu'à la carte Arduino
    - Enfin, vérifiez la position des alimentations (+5V et masse)
- Mettre les LEDs rouge et verte avec leur résistance de limitation de courant et un fil vers Arduino
- Mettre les décodeurs
    - Relier les fils ABCD à Arduino
    - Mettre au +5V ou à la masse les signaux de commandes du décodeur
    - Mettre les résistances de limitations de courant des 7 segments
    - Enfin, vérifier la position des alimentations (+5V et masse)
- Puis mettre les afficheurs -> les relier entre le décodeur et leurs segments) -> les connecter au +5V
- Amener du +5V et la masse sur la breadboard

Ce étant terminé, la maquette est fin prête à être utilisée ! Évidemment, cela fait un montage (un peu) plus complet que les précédents !

Programme

Nous allons maintenant voir une solution de programme pour le problème de départ. La vôtre sera peut-être (voire surement) différente, et ce n'est pas grave, un problème n'exige pas une solution unique. Je n'ai peut-être même pas la meilleure solution ! (mais ça m'étonnerait :ninja:)

Les variables utiles et déclarations

Tout d'abord, nous allons voir les variables globales que nous allons utiliser ainsi que les déclarations utiles à faire. Pour ma part, j'utilise six variables globales. Vous reconnaîtrez la plupart d'entre elles car elles viennent des chapitres précédents.

- Deux pour stocker l'état des boutons un coup sur l'autre et une pour le stocker de manière courante
- Un char stockant le nombre de places disponibles dans le parking
- Un booléen désignant l'afficheur utilisé en dernier
- Un long stockant l'information de temps pour le rafraichissement de l'affichage

Voici ces différentes variables commentées.

```
1  //les broches du décodeur 7 segments
2  const int bit_A = 2;
3  const int bit_B = 3;
4  const int bit_C = 4;
5  const int bit_D = 5;
6  //les broches des transistors pour l'afficheur des dizaines et celui des unités
7  const int alim_dizaine = 6;
8  const int alim_unite = 7;
9  //les broches des boutons
10 const int btn_entree = 8;
11 const int btn_sortie = 9;
12 //les leds de signalements
13 const int led_rouge = 12;
14 const int led_verte = 11;
15 //les mémoires d'état des boutons
16 int mem_entree = HIGH;
```

```
int mem_sortie = HIGH;
int etat = HIGH; //variable stockant l'état courant d'un bouton

char place_dispo = 99; //contenu des places dispos
bool afficheur = false;
long temps;
```

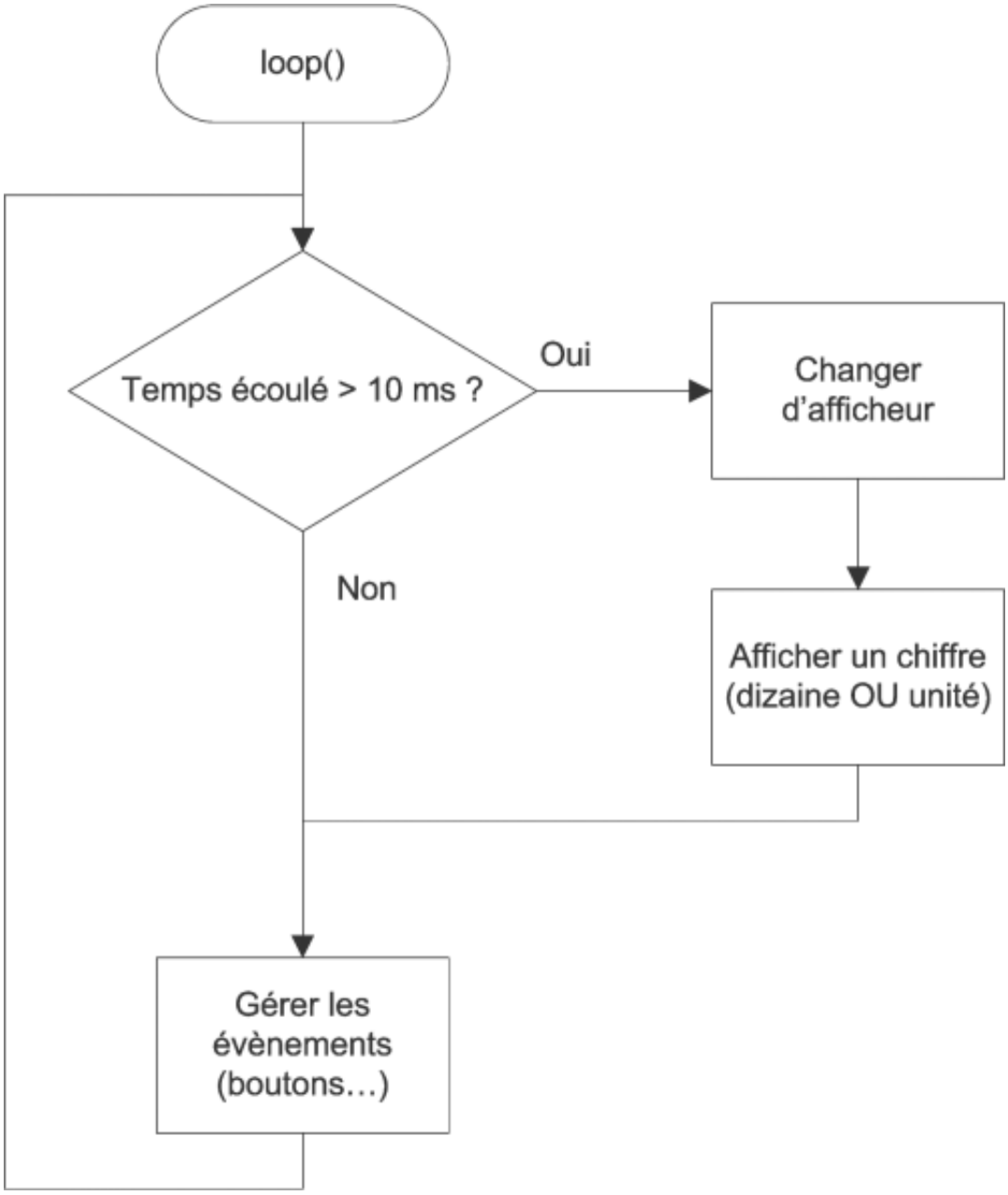
L'initialisation de la fonction setup()

Je ne vais pas faire un long baratin sur cette partie car je pense que vous serez en mesure de tout comprendre très facilement car il n'y a vraiment rien d'original par rapport à tout ce que l'on a fait avant (réglages des entrées/sorties et de leurs niveaux).

```
void setup()
{
    //Les broches sont toutes des sorties (sauf les boutons)
    pinMode(bit_A, OUTPUT);
    pinMode(bit_B, OUTPUT);
    pinMode(bit_C, OUTPUT);
    pinMode(bit_D, OUTPUT);
    pinMode(alim_dizaine, OUTPUT);
    pinMode(alim_unite, OUTPUT);
    pinMode(led_rouge, OUTPUT);
    pinMode(led_verte, OUTPUT);

    pinMode(btn_entree, INPUT);
    pinMode(btn_sortie, INPUT);

    //Les broches sont toutes mise à l'état bas (sauf led rouge éteinte)
    digitalWrite(bit_A, LOW);
    digitalWrite(bit_B, LOW);
    digitalWrite(bit_C, LOW);
    digitalWrite(bit_D, LOW);
    digitalWrite(alim_dizaine, LOW);
    digitalWrite(alim_unite, LOW);
    digitalWrite(led_rouge, HIGH); //rappelons que dans cette configuration, la LED
    digitalWrite(led_verte, LOW); //vert par défaut

    temps = millis(); //enregistre "l'heure"
}
```

La boucle principale (loop)

Ici se trouve la partie la plus compliquée du TP. En effet, elle doit s'occuper de gérer d'une part une boucle de rafraichissement de l'allumage des afficheurs 7 segments et d'autre part gérer les évènements. Rappelons-nous de l'organigramme vu dans la dernière partie sur les 7 segments :



Dans notre application, la gestion d'évènements sera "une voiture rentre-t/sort-elle du parking ?" qui sera symbolisée par un appui sur un bouton. Ensuite, il faudra aussi prendre en compte l'affichage de la disponibilité sur les LEDs selon si le parking est complet ou non… Voici une manière de coder tout cela :

```
1  void loop()
2  {
3      //si ca fait plus de 10 ms qu'on affiche, on change de 7 segments
4      if((millis() - temps) > 10)
5      {
6          //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou diz
7          afficheur = !afficheur;
8          //on affiche
9          afficher_nombre(place_dispo, afficheur);
10         temps = millis(); //on met à jour le temps
11     }
12
13     //on test maintenant si les boutons ont subi un appui (ou pas)
14     //d'abord le bouton plus puis le moins
15     etat = digitalRead(btn_entree);
```

```
16      if((etat != mem_entree) && (etat == LOW))
17          place_dispo += 1;
18      mem_entree = etat; //on enregistre l'état du bouton pour le tour suivant
19
20      //et maintenant pareil pour le bouton qui décrémente
21      etat = digitalRead(btn_sortie);
22      if((etat != mem_sortie) && (etat == LOW))
23          place_dispo -= 1;
24      mem_sortie = etat; //on enregistre l'état du bouton pour le tour suivant
25
26      //on applique des limites au nombre pour ne pas dépasser 99 ou 0
27      if(place_dispo > 99)
28          place_dispo = 99;
29      if(place_dispo < 0)
30          place_dispo = 0;
31
32      //on met à jour l'état des leds
33      //on commence par les éteindres
34      digitalWrite(led_verte, HIGH);
35      digitalWrite(led_rouge, HIGH);
36      if(place_dispo == 0) //s'il n'y a plus de place
37          digitalWrite(led_rouge, LOW);
38      else
39          digitalWrite(led_verte, LOW);
40  }
```

Dans les lignes 4 à 11, on retrouve la gestion du rafraichissement des 7 segments. Ensuite, on s'occupe de réceptionner les évènements en faisant un test par bouton pour savoir si son état a changé et s'il est à l'état bas. Enfin, on va borner le nombre de places et faire l'affichage sur les LED en conséquence. Vous voyez, ce n'était pas si difficile en fait ! Si, un peu quand même, non ? 😁 Il ne reste maintenant plus qu'à faire les fonctions d'affichages.

Les fonctions d'affichages

Là encore, je ne vais pas faire de grand discours puisque ces fonctions sont exactement les mêmes que celles réalisées dans la partie concernant l'affichage sur plusieurs afficheurs. Si elles ne vous semblent pas claires, je vous conseille de revenir sur le chapitre concernant les 7 segments.

```
1   //fonction permettant d'afficher un nombre
2   void afficher_nombre(char nombre, bool afficheur)
3   {
4       long temps;
5       char unite = 0, dizaine = 0;
6       if(nombre > 9)
7           dizaine = nombre / 10; //on recupere les dizaines
8       unite = nombre - (dizaine*10); //on recupere les unités
9
10      if(afficheur)
11      {
12          //on affiche les dizaines
13          digitalWrite(alim_unite, LOW);
14          digitalWrite(alim_dizaine, HIGH);
15          afficher(dizaine);
16      }
17      else
18      {
```

```
        //on affiche les unités
        digitalWrite(alim_dizaine, LOW);
        digitalWrite(alim_unite, HIGH);
        afficher(unite);
    }
}

//fonction écriveant sur un seul afficheur
void afficher(char chiffre)
{
    //on commence par écrire 0, donc tout à l'état bas
    digitalWrite(bit_A, LOW);
    digitalWrite(bit_B, LOW);
    digitalWrite(bit_C, LOW);
    digitalWrite(bit_D, LOW);

    if(chiffre >= 8)
    {
        digitalWrite(bit_D, HIGH);
        chiffre = chiffre - 8;
    }
    if(chiffre >= 4)
    {
        digitalWrite(bit_C, HIGH);
        chiffre = chiffre - 4;
    }
    if(chiffre >= 2)
    {
        digitalWrite(bit_B, HIGH);
        chiffre = chiffre - 2;
    }
    if(chiffre >= 1)
    {
        digitalWrite(bit_A, HIGH);
        chiffre = chiffre - 1;
    }
}
```

Et le code au complet

Si vous voulez tester l'ensemble de l'application sans faire d'erreurs de copier/coller, voici le code complet (qui doit fonctionner si on considère que vous avez branché chaque composant au même endroit que sur le schéma fourni au départ !)

```
//les broches du décodeur 7 segments
const int bit_A = 2;
const int bit_B = 3;
const int bit_C = 4;
const int bit_D = 5;
//les broches des transistors pour l'afficheur des dizaines et celui des unités
const int alim_dizaine = 6;
const int alim_unite = 7;
//les broches des boutons
const int btn_entree = 8;
const int btn_sortie = 9;
//les leds de signalements
const int led_rouge = 12;
const int led_verte = 11;
//les mémoires d'état des boutons
int mem_entree = HIGH;
```

```
int mem_sortie = HIGH;
int etat = HIGH; //variable stockant l'état courant d'un bouton

char place_dispo = 10; //contenu des places dispos
bool afficheur = false;
long temps;

void setup()
{
    //Les broches sont toutes des sorties (sauf les boutons)
    pinMode(bit_A, OUTPUT);
    pinMode(bit_B, OUTPUT);
    pinMode(bit_C, OUTPUT);
    pinMode(bit_D, OUTPUT);
    pinMode(alim_dizaine, OUTPUT);
    pinMode(alim_unite, OUTPUT);
    pinMode(btn_entree, INPUT);
    pinMode(btn_sortie, INPUT);
    pinMode(led_rouge, OUTPUT);
    pinMode(led_verte, OUTPUT);

    //Les broches sont toutes mises à l'état bas (sauf led rouge éteinte)
    digitalWrite(bit_A, LOW);
    digitalWrite(bit_B, LOW);
    digitalWrite(bit_C, LOW);
    digitalWrite(bit_D, LOW);
    digitalWrite(alim_dizaine, LOW);
    digitalWrite(alim_unite, LOW);
    digitalWrite(led_rouge, HIGH);
    digitalWrite(led_verte, LOW); //vert par défaut
    temps = millis(); //enregistre "l'heure"
}

void loop()
{
    //si ca fait plus de 10 ms qu'on affiche, on change de 7 segments
    if((millis() - temps) > 10)
    {
        //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou di
        afficheur = !afficheur;
        //on affiche
        afficher_nombre(place_dispo, afficheur);
        temps = millis(); //on met à jour le temps
    }

    //on test maintenant si les boutons ont subi un appui (ou pas)
    //d'abord le bouton plus puis le moins
    etat = digitalRead(btn_entree);
    if((etat != mem_entree) && (etat == LOW))
        place_dispo += 1;
    mem_entree = etat; //on enregistre l'état du bouton pour le tour suivant

    //et maintenant pareil pour le bouton qui décrémente
    etat = digitalRead(btn_sortie);
    if((etat != mem_sortie) && (etat == LOW))
        place_dispo -= 1;
    mem_sortie = etat; //on enregistre l'état du bouton pour le tour suivant

    //on applique des limites au nombre pour ne pas dépasser 99 ou 0
    if(place_dispo > 99)
        place_dispo = 99;
```

```
78      if(place_dispo < 0)
79          place_dispo = 0;
80
81      //on met à jour l'état des leds
82      //on commence par les éteindre
83      digitalWrite(led_verte, HIGH);
84      digitalWrite(led_rouge, HIGH);
85      if(place_dispo == 0) //s'il n'y a plus de place
86          digitalWrite(led_rouge, LOW);
87      else
88          digitalWrite(led_verte, LOW);
89  }
90
91  //fonction permettant d'afficher un nombre
92  void afficher_nombre(char nombre, bool afficheur)
93  {
94      long temps;
95      char unite = 0, dizaine = 0;
96      if(nombre > 9)
97          dizaine = nombre / 10; //on récupère les dizaines
98      unite = nombre - (dizaine*10); //on récupère les unités
99
100     if(afficheur)
101     {
102         //on affiche les dizaines
103         digitalWrite(alim_unite, LOW);
104         digitalWrite(alim_dizaine, HIGH);
105         afficher(dizaine);
106     }
107     else
108     {
109         //on affiche les unités
110         digitalWrite(alim_dizaine, LOW);
111         digitalWrite(alim_unite, HIGH);
112         afficher(unite);
113     }
114 }
115
116 //fonction écrivant sur un seul afficheur
117 void afficher(char chiffre)
118 {
119     //on commence par écrire 0, donc tout à l'état bas
120     digitalWrite(bit_A, LOW);
121     digitalWrite(bit_B, LOW);
122     digitalWrite(bit_C, LOW);
123     digitalWrite(bit_D, LOW);
124
125     if(chiffre >= 8)
126     {
127         digitalWrite(bit_D, HIGH);
128         chiffre = chiffre - 8;
129     }
130     if(chiffre >= 4)
131     {
132         digitalWrite(bit_C, HIGH);
133         chiffre = chiffre - 4;
134     }
135     if(chiffre >= 2)
136     {
137         digitalWrite(bit_B, HIGH);
138         chiffre = chiffre - 2;
```

```
139     }
140     if(chiffre >= 1)
141     {
142         digitalWrite(bit_A, HIGH);
143         chiffre = chiffre - 1;
144     }
145 }
146 //Fin du programme !
```

Conclusion

Bon, si vous ne comprenez pas tout du premier coup, c'est un petit peu normal, c'est en effet difficile de reprendre un programme que l'on a pas fait soi-même et ce pour diverses raisons. Le principal est que vous ayez cherché une solution par vous-même et que vous soyez arrivé à réaliser l'objectif final. Si vous n'avez pas réussi mais que vous pensiez y être presque, alors je vous invite à chercher profondément le pourquoi du comment votre programme ne fonctionne pas ou pas entièrement, cela vous aidera à trouver vos erreurs et à ne plus en refaire !

Il est pas magnifique ce parking ? J'espère que vous avez apprécié sa réalisation. Nous allons maintenant continuer à apprendre de nouvelles choses, toujours plus sympas les unes que les autres. Un conseil, gardez votre travail quelques part au chaud, vous pourriez l'améliorer avec vos connaissances futures !