Intruduction au langage C

Un processeur, comme par exemple ceux qui sont à l'intérieur des microcontrôleurs, comprennent des **instructions**, codées en binaire. C'est le **langage assembleu**r. Ces instructions sont simples, mais la puissance d'un processeur est liée à sa capacité d'exécuter très rapidement ces instructions (plusieurs millions d'instructions par seconde même sur un petit microcontrôleur).

L'écriture de programmes en assembleur est fastidieuse. Elle permet certes de tirer le meilleur parti d'un microcontrôleur, mais l'usage d'un langage de plus haut niveau s'est presque généralisé ces dernières années.

Le **langage C** est un langage ancien (début des années 1970). Il est simple et se prête très bien à la programmation des microcontrôleurs. Les **compilateurs** sont des programmes qui traduisent en assembleur les programmes écrits en C. Le compilateur s'exécute en général sur un PC. Il est souvent intégré dans un IDE (Integrated Devloppement Environnement = Environnnement de Développement Intégré). Comme les compilateurs progressent depuis 40 ans , ils sont devenus très performants !

Nous introduirons le langage C par petites étapes. En fait, nous avons déjà commencé à l'utiliser en écrivant nos premiers programmes Arduino.

Les procédures

En langage C, on écrit toujours des procédures. Nous expliquerons ce concept en détail plus tard, mais il suffit de savoir qu'une procédure est une suite d'instructions, que la procédure peut recevoir des données en entrée et qu'elle peut rendre des données en sortie .

Une procédure particulière est obligatoire dans tous les programmes en C. Elle s'appelle `main` (principal). Comme une procédure est un ensemble d'instructions, il faut signaler le début et la fin par des accolades. Voici donc un programme en C :

```
int main () {
}
```

Ce programme est vide. Le premier `int` indique que le programme va rendre un nombre entier (*integer*). On verra plus tard que ce n'est pas exactement le cas pour un programme écrit pour un microcontrôleurs (qui, en fait, ne se termine jamais), mais les compilateurs aiment bien cette manière d'écrire. Le mot réservé `main` indique que c'est la procédure principale : on parle plutôt de programme principal. Les parenthèses vides `()` indique que ce programme ne reçoit aucune donnée en entrée. Finalement, il nous reste à écrire un programme entre les accolades.

Un programme simple

Rappelons-nous quelques instructions en langage Arduino :

- `pinMode(no, mode)` no : numéro de la patte, mode : OUTPUT, INPUT, ...
- `digitalWrite(no, etat)` etat : LOW ou HIGH (0 ou 1)
- `int digitalRead(no)` int est la valeur rendue par la procédure : l'état de l'entrée
- `delay(duree)` durée : en ms

Voici un programme qui fait clignoter une LED. Il n'est pas écrit de la manière la plus élégante, le but étant plutôt de mettre en évidence la structure de contrôle :

```
#define Led 3
int main () {
    pinMode(Led, OUTPUT) ;
    while (1) {
        digitalWrite(Led, HIGH) ;
        delay(500) ;
        digitalWrite(Led, LOW) ;
        delay(500) ;
    } // fin du while
} // fin du main
```

Notez que le passage à la ligne n'est pas obligatoire en C. Il existe plusieurs styles de programmation. L'indentation rend plus lisible le programme. Le principe est le suivant : chaque fois qu'une accolage est ouverte, on décale les lignes suivantes jusqu'à la parenthèse fermée. La lisibilité en est grandement améliorée.

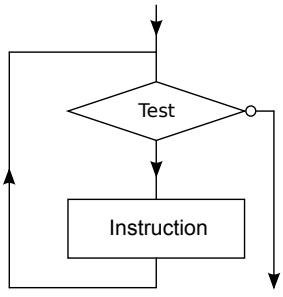
Vous remarquez certainement que l'instruction `pinMode(Led, OUTPUT)` correspond à ce qu'on écrit dans la procédure `void Setup()` de l'Arduino. De même, ce qui se trouve entre les accolades qui suivent le `while` correspond à la procédure `void Loop()`.

Même pour écrire un programme aussi simple qu'un clignotant, au moins une structure de contrôle du langage C a dû être utilisée : le `while`.

La structure while

Expliquons la structure de contrôle `while`. Ce mot se traduit par « *tant que* » en français. `while (condition) {instruction;}` peut donc se traduire par « tant que la condition est vraie, on exécute l'instruction ».

Les **organigrammes** (ou plus exactement un algorigramme) sont souvent utilisé pour décrire une partie de programme. Voici l'organigramme de la structure de contrôle `while` :



`while (condition) {instruction;}`

Il ne faut jamais oublier, en regardant un organigramme, qu'un processeur ne s'arrête jamais ! Il exécute en permanence des instructions. On doit donc suivre sans cesse les flèches.

On trouve dans un organigramme des rectangles, qui représentent une instruction. Plus précisément, un rectangle peut représenter plusieurs instructions, y compris des procédures.

Les losanges représentent des tests. La condition est vraie ou fausse. Chaque losange a donc une entrée (là où la flèche entre) et deux sorties (là où les flèches sortent) . Celle qui a un petit rond correspond à la condition fausse.

La structure `while` est une boucle. Sa durée d'exécution peut être très variable, selon la condition, qui peut évoluer avec le temps.

Dans l'exemple du clignotant, on souhait simplement dire que le programme devait

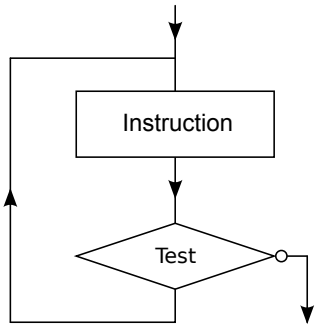
allumer et éteindre la LED en permanence, donc sans jamais s'arrêter. Pour pouvoir utiliser la structure de contrôle `while`, on a donc utilisé une condition qui est toujours vraie.

Certains langages de programmation connaissent explicitement les valeurs « vrai » (*true*) et « faux » (*false*). En C, la règle est la suivante :

- si la valeur est ***nulle***, la condition est considérée comme fausse
- si la valeur est ***non nulle***, la condition est considérée comme vraie.

L'expression `while (1)` correspond donc à une boucle infinie ! Et c'est bien ce qu'on veut avec un microcontrôleur. Le programme gravé dans sa mémoire morte ne doit jamais se terminer. Il s'arrêtera lors qu'on coupera l'alimentation...

La structure do...while



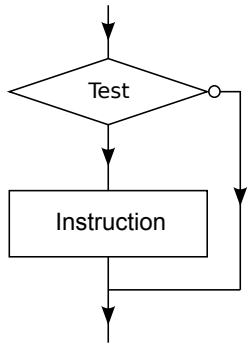
Une seconde structure de contrôle est très similaire au `while`. C'est la structure `do...while`.

En observant son organigramme, on remarque que le test et l'instruction sont inversés par rapport au `while`.

On peut traduire `do instruction; while (condition);` par « exécute l'instruction tant que la condition est vraie ».

Il faut noter que l'instruction va s'exécuter un moins une fois avec la structure `do...while`.

La structure if



La dernière structure que nous allons voir ici n'est pas une boucle. C'est une structure permettant d'effectuer un test.

`If (test) {instruction;}` signifie : « exécuter l'instruction si la condition est vrai, puis continuer dans tous les cas ».

Contrairement aux boucles, l'instruction `if` ne permet aucune répétition. Le test n'est exécuté une seule fois.

Exemple utilisant le if

Le programme suivant permet de commander une LED par un poussoir :

```
#define Led 3
#define Poussoir 4
int main () {
    pinMode(Led, OUTPUT);
    pinMode(poussoir, INPUT_PULLUP);
    while (1) {
        if (digitalRead(poussoir)) {
            digitalWrite(Led, HIGH);
        } else {
```

```
            digitalWrite(Led, LOW);
        }
    } // fin du while
} // fin du main
```

Dans ce programme, la condition est donnée par la procédure `digitalRead(poussoir)`, qui rend *vrai* si le bouton... n'est pas pressé (tout au moins avec le schéma classique, où une des extrémités du poussoir est connectée à la masse). Ce programme éteint donc la LED lorsque la poussoir est pressé.

Notez là aussi que ce programme peut être écrit de manière plus simple et plus élégante : la condition rendue par `digitalRead` pourrait être donnée directement en paramètre à la procédure `digitalWrite` :`digitalWrite(Led, digitalRead(poussoir));`

Autres structures

Il est possible d'écrire n'importe quel programme avec les trois structures de contrôle que nous venons de voir : `while, do...while et if.`

Le C propose encore trois autres structures :

- `if...else` : version plus complète du `if`, avec une instruction pour la condition fausse
- `switch...case` : c'est un test multiple, un peu compliqué... mais très puissant !
- `for` : une boucle, utilisée le plus souvent pour des itérations.

Nous expliquerons ces structures plus tard.