# Production de librairie statique et stratégie de débogage

**Travaux pratiques 7 et 8**

Laurent Bourgon
Mehdi Benouhoud
Ihsane Majdoubi
Catalina Andrea Araya Figueroa

Lundi 13 mars 2023

# 1 Description de la librairie

## 1.1 Classes

### 1.1.1 Led

Depuis le début du cours, nous utilisons un diode électro-luminescente située sur le circuit imprimé du robot. Cette diode est dite « bi-colore », c'est-à-dire qu'elle peut s'allumer en rouge ou en vert, dépendament du sens du courant qu'on lui transmets. En faisant clignoter rapidement la diode en alternant la couleur, on obtient une troisième couleur : ambrée.

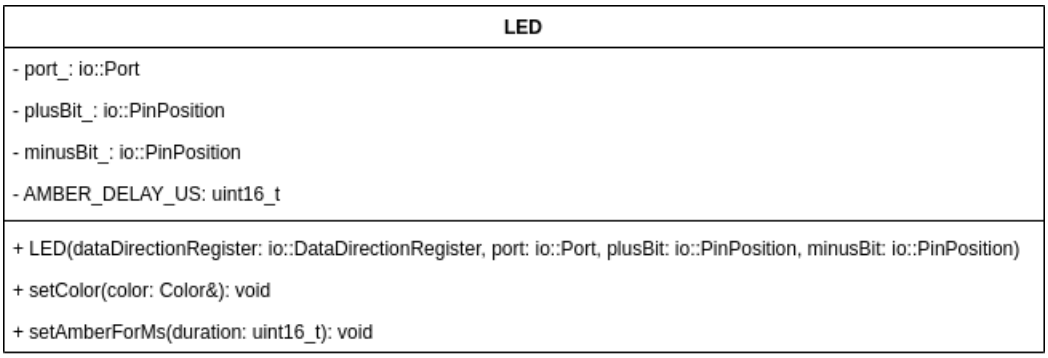


FIGURE 1 – Diagramme de classe UML de **Led**

La classe est doté d'un constructeur, nous permettant d'indiquer où est branchée la diode.

```
Led(io::DataDirectionRegister dataDirectionRegister, io::Port port,
    const io::PinPosition plus, const io::PinPosition minus)
```

`dataDirectionRegister` indique le registre de direction des données où la diode est branchée, `port` indique son port et finalement `plus` et `minus` indique les broches sur lesquelles sont branchés la respectivement cathode et l'anode de la diode. Il est important de respecter ce sens afin de permettre à la méthode `setColor(const Color &color)` de fonctionner adéquatement.

La méthode `setColor(const Color &color)` permet d'allumer la diode en rouge ou en vert, en passant en paramètre `Color::RED` ou `Color::GREEN`.

La méthode `setAmberForMs(const uint16_t durationMs)` permets d'allumer la

diode en ambrée. Nous devont passer une durée (en milisecondes) en paramètre. En effet, puisque la diode doit alterner rapidement entre le rouge et le vert, il est impossible d'effectuer cette opération indéfiniement.

# 2 Modifications apportées au Makefile de départ

Depuis le début du cours, nous utilisons le `Makefile` fourni afin de compiler notre code et l'installer sur le robot. Ayant créé une librairie, il a fallu faire quelques modifications.

## 2.1 Ajout d'une Makefile pour la librairie

La compilation et l'archivage d'une librairie en un fichier `.a` implique des variables et instructions supplémentaires par rapport à ce que le `Makefile` de départ nous permettait de faire.

Ce nouveau `Makefile`, situé dans le répertoire `lib`, permets de faire l'édition et l'archivage des liens des fichiers objets `.o` avec l'outil `avr-ar`. De plus il contient quelques variables supplémentaires :

- `LIBNAME` qui indique le nom de la librairie (lib1900)
- `SOURCES` et `OBJECTS` qui indique les fichiers sources `.cpp` et leurs fichiers objets homonymes en `.o`
- `ARTIFACTS` qui indique les fichiers artéfacts devant être supprimés dans le cas d'un `make clean`

La librairie utilise des variables et règles communes à tous les `Makefile`, comme `make clean`.

## 2.2 Encapsulation des variables et règles communes

C'est pour cela que nous avons créé un `Makefile.common`, situé à la racine du dépôt afin qu'il puisse être importer partout, qui regroupe les éléments communs aux `Makefile`s de l'exécution et de la librairie.

Ce fichier regroupe notament les variables et la logique relié à

- La supression d'artéfacts de compilation inutile, à l'aide de `make clean`
- La compilation des fichiers sources `.c` et `.cpp` en fichiers objets `.o`
- L'inclusion des fichiers de dépendances `.d`
- La création d'archives binaires `.hex` à partir des fichiers `.elf`
- L'installation (ou *flash*) du programme vers le microcontrôleur

Les deux `Makefile`s peuvemt importer leurs éléments commun à l'aide de l'instruction

```
include ../../../Makefile.common
```

## 2.3 Modifications au Makefile compilant le projet

Outre les retraits de certaines variables et règles, qui sont désormais contenus dans `Makefile.common`, nous avons procédé à de petites modifications pour que le compilateur inclut notre librairie lors de la compilation.