Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1995:
Projet initial en génie informatique et travail en équipe

Travail pratique 8

# Makefile et production de librairie statique

Par l'équipe

No 27107

Noms:

Aurele Chanal 1885961
Paul Clas 1846912
Thomas McAdams 1904141
Daniel Nahum 1877872
Emmanuelle Roberge 1849263
Karima Salem 1896851

Date:
12 mars 2018

## Partie 1 : Description de la librairie

La librairie construite contient toutes les parties de code jugées utiles pour les prochaines étapes du projet. Cette librairie permet d'utiliser ces éléments sans avoir à réécrire le code. Ainsi, une librairie permet de sauver du temps et aide à l'optimisation du code, car elle évite les répétitions. La librairie que nous avons formée contient les éléments suivants :

CLASSES :

**CAN (can.cpp et can.h)**

Les fichiers de la classe can se trouvent dans notre librairie, car la conversion des signaux analogiques captés en signaux numériques est nécessaire pour que ces signaux puissent être traités par le processeur. Les méthodes retrouvées dans cette classe sont :

- **can()** et **~can()**
  Le constructeur et le destructeur
- **uint16_t lecture(uint8_t pos)**
  La méthode qui effectue la lecture d'une position et retourne la valeur lue sur 16 bits.

**UART (uart.cpp et uart.h)**

La classe uart permet la transmission de données entre le robot et l'ordinateur. De plus, elle permet d'afficher sur la console les valeurs enregistrées par le capteur en temps réel. Elle est utile pour l'établissement des seuils nécessaires aux changements d'états.

- **void initialisation_UART ();**
  Initialise le mode d'utilisation du uart
- **void transmit_UART(uint8_t byte);**
  Transmet la donnée enregistrée lorsque le data register empty flag et le status register ne sont pas égaux.
- **void transmit_UART_number(uint16_t n);**
  Permet l'affichage des données enregistrées par le capteur.

**MÉMOIRE (memoire_24.cpp et memoire_24.h)**

La mémoire est souvent utilisée au cours de ce projet, ce qui justifie l'utilisation fréquente des opérations sur cette dernière. Pour éviter de réécrire les mêmes bouts de code sans arrêt, la classe mémoire est placée dans notre librairie. Les méthodes retrouvées dans cette classe sont :

- **Memoire24CXXX()** et **~Memoire24CXXX()**
  Le constructeur et le destructeur
- **void init()**
  Une procédure d'initialisation lancée par le constructeur qui met le memory bank à 0.
- **static uint8_t choisir_banc(const uint8_t banc)**
  Cette méthode est appelée uniquement en cas de changement d'adresse.
- **uint8_t lecture(const uint16_t adresse, uint8_t *donnee)** et
- **uint8_t lecture(const uint16_t adresse, uint8_t *donnee, const uint8_t longueur)**

Elles servent toutes les deux à la lecture. L'unique différence entre elles est le fait que la deuxième méthode prend en paramètre la longueur d'un bloc de données, qui doit absolument être en dessous de 127.

- **uint8_t ecriture(const uint16_t adresse, const uint8_t donnee)** et
  **uint8_t ecriture(const uint16_t adresse, uint8_t *donnee, const uint8_t longueur)**
  Elles sont utilisées pour l'écriture et ici encore, l'unique différence entre elles est le fait que la deuxième méthode prend en paramètre la longueur d'un bloc de données, qui doit absolument être en dessous de 127.

**MINUTERIE (minuterie.cpp et minuterie.h)**

L'utilisation fréquente de la minuterie explique la présence de cette classe dans notre librairie. De plus, il est plus facile de la gérer de cette façon. La classe Minuterie présente les méthodes suivantes :

- **Minuterie()**
  Le constructeur
- **void partirMinuterie ( uint16_t duree )**
  Cette méthode sert à partir la minuterie pour une durée prise en paramètre.
- **void resetMinuterie ( uint16_t duree )**
  Elle sert à repartir la minuterie lorsque la minuterie atteint la valeur de la durée prise en paramètre.
- **uint8 getMinuterieExpiree()**
  Elle retourne 1 si la minuterie est expirée ou 0 si elle ne l'est pas.
- **uint8 getDixiemesSecondes()**
  Cette méthode retourne une valeur en dixième de seconde.

**DEL (del.cpp et del.h)**

La classe DEL permet de contrôler les comportements de la DEL libre sur la carte mère. L'attribut de cette classe est de type PORTx, qui est une énumération des différents ports auxquels il est possible de brancher les fils pour allumer la DEL. Encore ici, la gestion de cette classe est simplifiée par le fait qu'elle est présente dans notre librairie et la fréquence d'utilisation des DEL explique pourquoi elle se trouve dans cette dernière. Les méthodes trouvées dans cette classe sont :

- **DEL(PORTx portx)**
  Le constructeur
- **void DEL_RED()**
  Cette méthode fait en sorte que la couleur de la DEL allumée est rouge.
- **void DEL_GREEN()**
  Cette méthode fait en sorte que la couleur de la DEL allumée est verte.
- **void DEL_OFF();**
  Cette méthode fait en sorte que la DEL s'éteigne.
- **void DEL_AMBRE().**
  Cette méthode fait en sorte que la couleur de la DEL est ambrée.

FONCTION :

**TOUCHBUTTON (touchButton.cpp et touchButton.h)**

Cette fonction détermine si le bouton a été pressé et retourne le booléen en conséquence.

INCLUSIONS ET DÉFINITIONS :

## Partie 2 : Décrire les modifications apportées au Makefile de départ

**Makefile de la librairie :**

Le makefile de la librairie a pour but de créer un fichier d'archive qui permettra au makefile de l'executable de compiler.

**PRJSRC = $(wildcard *.cpp)**
Cette commande permet de récupérer tous les fichiers (.cpp) dans le répertoire courant. La cible par défaut est le nom de la librairie avec l'extension (.a). Nous avons mis les dépendances objet qui remplacent les fichiers (.cpp) par les fichiers (.o). On a enlevé toute la production de fichiers hex à partir des fichiers elf.

**AR=avr-ar**
**ARFLAGS=-crs**
Déclaration de nouvelles variables

**vars:**
**@echo "PRJSRC = $(PRJSRC)"**
**@echo "OBJDEPS = $(OBJDEPS)"**
Lorsque l'instruction make vars est appelée, les valeurs des variables PRJSRC et OBJDEPS.

**TRG = librobotbaby.a**
Déclare le nom de la cible du fichier d'archive.

**OBJDEPS= $(PRJSRC:.cpp=.o)**
Conversion des fichiers .cpp en .o.

**$(TRG): $(OBJDEPS)**
**$(AR) $(ARFLAGS) $@ $^**
Ceci appelle les fichiers pour la compilation ainsi que les fichiers cibles.

**Makefile de l'exécutable :**

Pour ce Makefile, nous avons gardé le Makefile original fourni sur le site du cours INF1995 et nous y avons apporté les modifications suivantes :

**PRJSRC = $(wildcard *.cpp)**
Cette commande permet de récupérer tous les fichiers .cpp dans le répertoire courant.

**INC = -I ../lib_dir**

INC permet des inclusions additionnelles du répertoire lib_dir.

**LIBS= L ../lib_dir –l robotbaby**
LIBS est la librairie à lier elle refère à l'archive robotbaby.a pour créer l'executable.

**-DF_CPU=8000000**
Cette ligne de code permet de définir la vitesse en cycle par seconde du microcontroleur.

Nous gardons la création de fichier hexadécimal. On garde l'appel au microcontrôleur pour l'installation (éventuelle) de notre exécutable.

Pour exécuter notre programme, il faut compiler la librairie avant de compiler l'exécutable.