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 6972

Noms :
Amar Ghaly,
Yacer Razouani,
Mazigh Ouanes
et Georges Louis

Date:
12 Mars 2018

## Partie 1 : Description de la librairie

Dans le but de créer une librairie possédant toutes les fonctions nécessaires, nous avons débuté par comparer les codes de nos travaux pratiques précédents. De cette façon, nous avons pu regroupé de manière logique les fonctions ayant un lien commun et les avons mises dans des classes. Notre librairie contient les définitions (.h) et les implémentations (.cpp) des classes que tous les membres de l'équipe jugent pertinent

La classe bouton contient les fonctions userPush() et userRelease(). Ces fonctions nous permettent respectivement de savoir si le bouton a été appuyé ou s'il a été relâché. Ces fonctions nous permettent de prévoir un mécanisme d'anti-rebond et d'économiser des lignes de codes.

La classe lumiereDel contient un constructeur par paramètre, un destructeur et les fonctions : activateRed(), activateGreen(), activateAmbre() et activateNeutral(). Cette classe contient aussi un attribut port_. Lorsque nous passons une lettre en paramètre dans le constructeur par paramètre, nous définissons quel port va sera connecté à la lumière DEL. Par exemple : lumiereDel(B), l'attribut port_ est maintenant égal à 'B' et tout appel à une fonction comme activateRed() va activer les 2 bits les moins significatifs sur le port B pour allumer la lumière DEL en rouge. Cette classe permet alors de rendre le code davantage lisible et propre.

La classe outin, qui tient son nom des « Outputs – Inputs », sert à définir les ports qui sont en sorties et ceux qui sont en entrées. Cette classe contient un constructeur par défaut qui initialise tous les ports en sorties puisque dans la majorité du temps les ports sont utilisés en sorties. Alors, quand il sera nécessaire nous déterminerons les ports qui doivent être en entrées et cela nous économisera du temps. Cette classe contient aussi un destructeur et deux fonctions : setOut (char D) et setIn (char D). La fonction setOut (char D) prend en paramètre une lettre correspondant à un port puis active ce port en sortie. La fonction setIn (char D) prend en paramètre une lettre correspondant à un port puis active ce port en entrée. L'utilité de cette classe réside dans l'économie de lignes de codes, puisqu'en une ligne on peut activer tous les ports en sorties (à l'aide du constructeur par défaut), puis s'il le faut on peut faire quelques ajustements avec les fonctions.

La classe roues contient les fonctions : activateReverse(), activateForward() et activateNeutral(). La fonction activateReverse() permet de faire reculer le robot. La fonction activateForward() fait tourner les roues pour faire avancer le robot. La fonction activateNeutral() permet d'immobiliser le robot. Cette classe regroupe les fonctions importantes de contrôle des roues et augmente la lisibilité et clarté du code.

Finalement, notre librairie contient les classes memoire_24 et can. Ces classes des ont été fourni dans les énoncés des travaux pratiques. Nous avons décidé de les inclure dans notre librairie en vue d'une possible utilisation future.

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

Notre travail pratique contient trois makefiles distincts. Tous ces makefiles sont basés sur celui fournit dans les travaux pratiques antérieurs. Le premier makefile correspond à celui contenu dans le répertoire de notre librairie nommé « lib ». Nous avons débuté par modifier le nom du projet, soit PROJECTNAME = mylib. Ensuite, nous avons modifié PRJSRC. Nous y retrouvons les noms des fichiers du code sources qui sont à compiler. Nous avons ajouté alors tous les fichiers d'implémentation de nos classes (.cpp), soit bouton.cpp, can.cpp, lumiereDel.cpp, memoire_24.cpp, outin.cpp et roues.cpp. Ici, il est inutile de modifier INC et LIBS car il n'y a aucune autre inclusion additionnelle nécessaire et aucune librairie à inclure car nous sommes dans la librairie même.

En ce qui concerne les variables du makefile, il est important d'ajouter le compilateur AR=avr-ar. Le suffixe ar signifie archive et est celui utilisé pour les librairies statiques. Il nous fallait aussi modifier l'extension de la variable TRG=$(PROJECTNAME).out pour TRG=$(PROJECTNAME).a. Lors de l'implémentation de la cible, nous avons modifié

```
$(TRG): $(OBJDEPS)
        $(CC) $(LDFLAGS) -o $(TRG) $(OBJDEPS) \
        -lm $(LIBS)
```

pour

```
 $(TRG): $(OBJDEPS)
        $(AR) crs -c $(TRG) $(OBJDEPS).
```

Cette modification nous permet de produire une librairie statique. L'usage de –c au lieu de –o permet de créer l'archive. Ensuite, le résultat produit par l'archive est le nom de ma librairie avec l'extension « .a ». Dans ce makefile, l'option Install est inutile car nous ne désirons pas écrire de programme en mémoire flash dans notre microcontrôleur. C'est pour cette même raison que nous avons retiré la production de fichiers en hexadécimale.

Notre second makefile est celui retrouvé dans le répertoire src. Nous avons

créé ce makefile dans le but de tester si notre librairie était bel et bien fonctionnelle. L'usage d'un fichier bidon est donc prescrit. Le fichier bidon contient des commandes se servant des fonctions retrouver dans nos différentes classes. Une modification apportée à ce makefile est PRJSRC= fichierbid.cpp car il s'agit du nom de notre fichier bidon à compiler. Ensuite, il est primordiale d'ajouter le chemin à emprunter pour se rendre à notre librairie, soit INC = -I../lib ainsi que le nom de la librairie à lier, soit LIBS= ../lib/mylib.a. Aucune autre modification n'a été apporté à ce makefile.

Finalement, le troisième makefile correspond au makefile générale. Celui-ci se trouve dans notre répertoire nommé TP8, à côté des répertoires lib et src. Les modifications apportées à ce makefile correspondent à des modifications que l'on retrouve dans le makefile de la librairie. Premièrement, nous avons renommé PROJECTNAME=monprojet. Ensuite, nous avons ajouté la variable de compilation AR = avr-ar et l'extension « .a ». En somme, nous avons été en mesure de créer une librairie statique fonctionnelles, de l'exécuter et de la tester à l'aide des différents makefiles.