Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1900:
Projet initial de système embarqué

Travail pratique 7

Makefile et production de librairie statique

Par l'équipe

No 4344

Noms:

Fecteau
Malouin
Poulin
Tran

Date:
3 Mars 2019

Partie 1 : Description de la librairie

Décrire la librairie construite et formée (définitions, fonctions ou classes, utilité, etc.) pour que cette partie du travail soient bien documentées pour la suite du projet pour le bénéfice de tous les membres de l'équipe.

can:
Les fichiers can.cpp et can.h sont les fichiers qui nous ont été fournis par Jérôme Collin et Matthew Khouzam. Ces derniers permettent de convertir les signaux analogiques des capteurs du robot en valeurs numériques. Les méthodes de cette classe sont les suivantes :

- uint16_t lecture(uint8_t pos)

Cette méthode analyse les signaux analogiques sur le PORTA afin de les convertir et de retourner une valeur numérique. L'entier à 8 bits « pos » permet de sp/cifier le bit que nous voulons analyser du PORTA. Un convertisseur analogique est un objet que nous devons instancier afin d'utiliser.

log:
Les fichiers log.cpp et log.h sont les fichiers qui nous permettent de faire apparaître à l'écran du texte lorsque nous en avons besoin. Les méthodes qui permettent son fonctionnement sont les suivantes :

- void initialisationUART()

Qui permet d'initialiser les fonctions du robot nécessaires à la transmission de données par cable USB.

- void transmissionUART(uint8_t donnee)

Qui permet de transférer un seul entier à 8 bits par cable USB, ce format étant celui d'un « unsigned char ». Cela permet donc de transmettre des symboles, du texte et des nombres par USB. Cette donnée doit être lue et traitée par le programme « serieviaUSB », fourni par les enseignants du cours.

- void log_uart(char mots[])

Qui permet d'appeler la fonction transmissionUART à répétition afin de tranférer un nombre plus grand de données en une seule commande, soit des phrases.

moteurs:
Les fichiers moteurs.cpp et moteurs.h permettent de gérer les signaux PWM nécessaires au bon fonctionnement des moteurs du robot. Les méthodes de ces fichiers sont les suivantes :

- void ajustementPWM(int pa, int pb)

Qui permet d'envoyer deux pourcentages sur 100, pourcentage a et pourcentage b,

afin de mettre à jour la vitesse voulue des moteurs en une seule commande.
wait:
Les fichiers wait.cpp et wait.h nous permettent de créer un délai dans le robot avec une valeur qui n'est pas une constante de compilation. La méthode de ce fichier est la suivante :

- void wait(double __ms)

__ms étant le nombre de milisecondes à attendre.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Décrire les quelques modifications apportées au Makefile de la librairie pour démontrer votre compréhension de la formation des fichiers. Faire de même pour les modifications apportées au Makefile du code (bidon) de test qui utilise cette librairie.

Makefile de la librairie:

Pour la création d'une librairie statique, nous devions modifier le Makefile utilisé précédemment pour les autres laboratoires. Puisque cette fois ci nous voulions créer une librairie et non seulement compiler le code, nous avons ajouté et modifié certaines commandes dans le Makefile pour la création et l'utilisation de celle-ci.

Premièrement, nous avons changé la valeur de PRJSRC, où on indique normalement les quelques fichiers sources(.cpp) à compiler pour le projet, par $ (wildcard *.cpp) qui permet d'aller chercher tous les fichiers cpp dans le répertoire au lieu de devoir tous les nommer individuellement.

Deuxièmement, nous avons créé la variable AR=avr-ar qui est l'équivalent de CC=avr-gcc utilisé auparavant, pour faciliter les commandes lors de l'implémentation de la cible. Nous avons aussi créé un nouveau flag pour l'archive (AR) qui est : ARFLAGS = -crs. Évidemment, l'option « c » est pour créer l'archive, l'option « r » est pour insérer les fichiers membres et l'option « s » est pour produire l'index des fichiers objets. On utilisera donc cette commande avec la liste des fichiers objet à créer, nommée OBJDEPS (qui est déterminé à l'aide de PRJSRC) et avec le nom de l'archive. Cela produira la librairie avec le nom choisi et avec l'extension « .a ».

Ensuite, il ne reste plus qu'à modifier l'implémentation de la cible. Comme il est dit dans le laboratoire, nous devons changer l'appel à avr-gcc, qui produit l'exécutable, par un appel à « ar » pour créer la librairie statique. C'est à cette étape que nous regroupons et utilisons la majorité des changements faits au Makefile. Nous utilisons donc les commandes :

```
$(TRG) : $(OBJDEPS)
        $(AR) $(ARFLAGS) $@ $^
```

Pour ce qui est de la première ligne, nous avons seulement modifié la variable

OBJDEPS indirectement en modifiant PRJSRC pour aller chercher tous les fichiers cpp. $(AR) est pour indiquer qu'il s'agit de la création d'une archive. $(ARFLAGS) est pour l'implémentation des options « c », « r » et « s ». Finalement $@ est le nom de la cible et $^ est la liste des dépendances (les fichiers cpp utilisés pour la librairie).

Cette commande permet donc la création d'une librairie qui nous donne accès aux fonctions des fichiers cpp fournis qui est produit avec l'extension « .a » dans le répertoire du Makefile.

Makefile du code:

Afin de tester la librairie créée, nous avons programmé un logiciel simple qui utilise la méthode « log_uart(char mots[]); » présente dans le fichier « log.h » de la librairie. Afin d'utiliser la librairie créée précédemment, nous avons modifié le Makefile permettant de compiler ce logiciel simple. Lors de la production des fichiers objets, nous avons inclue le chemin de la librairie avec l'option « -I ». Lors de l'implémentation de la cible, nous avons ajouté l'argument « -l » et « -L » afin de lier la librairie.

Le rapport total ne doit par dépasser 7 pages incluant la page couverture.

Barème: vous serez jugé sur:

- *La qualités et le choix de vos portions de code choisies (5 points sur 20)*
- *La qualités de vos modifications aux Makefiles (5 points sur 20)*
- *Le rapport (7 points sur 20)*
  - *Explications cohérentes par rapport au code retenu pour former la librairie (2 points)*
  - *Explications cohérentes par rapport aux Makefiles modifiés (2 points)*
  - *Explications claires avec un bon niveau de détails (2 points)*
  - *Bon français (1 point)*
- *Bonne soumission de l'ensemble du code (compilation sans erreurs, …) et du rapport selon le format demandé (3 points sur 20)*