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

3991

Noms:

Louis Cormier
Dogbeba Georges Gnaga
Zouhair Chiguer
Wajiha Bissola Badirou

Date:
10 Mars 2017

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.

Pour la réalisation de ce TD, nous avons créé une librairie composée de classes. Ces dernières sont composées de fonctions; le tout permettant de regrouper les programmes rédigés jusqu'à présent, qui nous seront sûrement utile pour la suite du projet. Notre librairie contient les classes ci-dessous :

La classe TypePort : TypePort.h

La classe TypePort a été créé surtout pour être utilisée dans les autres classes de notre librairie. Elle est juste constituée de l'énumération des différents ports (A, B, C, D) du robot. Ainsi, nous n'avons pas à déclarer l'énumération dans chaque classe et pouvons juste faire appel à la classe TypePort pour cela.

La classe Del : del.cpp / del.h

Nous avons créé la classe Del qui nous permet de gérer les couleurs des DEL. Cette classe comporte quatre (4) fonctions principales : allumerRouge (), allumerVert (), allumerAmbre (), eteindreDel ().

Les fonctions allumerRouge (), allumerVert (), allumerAmbre () et eteindreDel (), permettent respectivement d'allumer la Del sur la couleur rouge, sur la couleur verte, sur le couleur ambre (qui est une succession de rouge et de vert à une fréquence adaptée) , puis pour finir d'éteindre la Del.

La classe Memoire24 : memoire_24.cpp/ memoire_24.h

Cette classe nous a été fourni, et étant donné son importance et la pertinence pour notre projet nous avons décidé de l'inclure dans notre librairie. La classe Memoire24 nous permet d'accéder à la mémoire et y mettre des informations.

La classe Moteur : moteur.cpp / moteur.h

La classe Moteur a été dans le seul but de contrôler le moteur. Elle comporte pour ce fait une seule fonction : tournerMoteur (float puissanceDroite, float puissanceGauche).
Cette fonction s'occupe principalement de faire tourner le moteur et par la même occasion le tourner de sort à faire avancer le robot.

La classe Controller : controlleur.cpp / controlleur.h

La classe Controller regroupe les autres fonctions permettant de contrôler le robot et qui sont nécessaires pour les autres classes. Cette classe comporte notamment une seule fonction principale qui est : estAppuye ().
Cette fonction permet entre autres de savoir si le bouton poussoir du robot est appuyé ou non, tout en tenant compte de l'anti-rebond.

La classe Can : can.cpp / can.h

Cette classe nous a aussi été fourni, compte tenu de son importance et pertinence nous avons décidé de l'inclure dans notre librairie. Cette classe permet de faire la conversion analogique – numérique des signaux.

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.*

En ce qui concerne nos Makefile, nous avons décidé d'en créer deux chacun ayant un but distinct. Nous avons le premier qui permet de créer la librairie, puis le deuxième qui sert à générer l'exécutable utilisant la librairie statique. Pour finir nous avons un makefile_common qui gèrent les deux premiers.

Makefile I – Création de la librairie

Afin de pouvoir créer notre librairie, nous avons dû modifier certaines parties de Makefile, Nous ferons par des modifications que nous avons eu à apporter. Tout d'abord, la création de notre Makefile, nécessite l'inclusion des répertoires comportant les fichiers ayant les extensions « .c », « .h » et « .cpp ». Ces fichiers sont obligatoirement nécessaires, et afin de simplifier leur inclusion dans notre makefile, nous les avons mises dans la variable **PRJSRC**. Par la suite, elles seront compilées à l'aide du compilateur **avr-gcc**. Le résultat de la compilation donnera des fichiers objets portant l'extension «. o ».

Dans le Makefile, les commandes :

```
$(TRG): $(OBJDEPS)
$(CCTRG) $(LDFLAGSTRG) -o $(TRG) $(OBJDEPS) \
-lm $(LIBS)
```

permettaient de convertir les fichiers objets en un fichier exécutable. Dans les commandes, nous distinguons les variables :

- **TRG** qui porte le nom de projet (**TRG=$(PROJECTNAME).out**)
- **OBJDEPS** qui contient les fichiers objets (**OBJDEPS=$(CFILES:.c=.o) \$ (CPPFILES:.cpp=.o)\$(BIGCFILES:.C=.o) \ $(CCFILES:.cc=.o) \ $(ASMFILES:.S=.o)**).
- **CCTRG** est le nom du compilateur que nous utilisons soit avr-gcc

Par la suite, afin de créer la librairie, **TRG** a dû être modifié en **TRG=$(PROJECTNAME). a** avec PROJECTNAME représentant la **libstatique**. L'extension «. a » spécifie que le fichier créé est une archive.
Étant donné que, "Généralement, on utilise les options **c** (pour créer l'archive), **r** (pour l'ajout des fichiers objets à l'archive) et **s** (pour produire en plus l'index des fichiers objets pour accélérer l'édition de liens) avec cette commande."[1], nous avons , la variable

- **LDFLAGSTRG** qui porte les options de compilation pour la création du fichier de la librairie.

Makefile II – Génération de l'exécutable

Le makefile de l'exécutable est assez simple et semblable aux makefile utilisés tout au long de la session. Ce dernier inclut la librairie crée, ainsi que les fichiers sources qui nous permettront de générer l'exécutable.
L'inclusion de la librairie se fait à l'aide la commande :

LIBS= -L$(LIBDIR) -lstatique

Dans cette commande, la variable LIBS sert à éditer les liens qui permettent de générer l'exécutable. À cette variable, nous avons affecté le chemin qui mène au dossier de la librairie afin qu'il puisse exécuter la nôtre. La commande montre que la librairie se trouve dans le répertoire *LIBDIR*. Nous relevons aussi l'utilisation de *lstatique* qui informe le compilateur qu'il doit chercher *libstatique.a* dans le répertoire.

Les includes additionnels se font à l'aide de la commande :

INC=-I$(LIBDIR)

Dans cette commande, la variable **INC** comporte le chemin menant vers la libraire statique. Cette variable sera utilisée lors de la compilation, et est nécessaire pour la construction de l'exécutable. Nous avons aussi la variable **LIBDIR** qui est le chemin vers le dossier contenants les fichiers pour la librairie. La commande associée à **INC** indique donc qu'il faut remonter jusqu'au dossier qui contient les fichiers de la librairie nécessaire afin de générer l'exécutable.

Makefile III – Makefile commun

Le makefile commun est un makefile servant à simplifier et synthétiser les deux premiers makefiles. Il regroupe les règles et commandes communes aux deux premiers makefile. Il pourra être ensuite inclus dans les autres makefile à l'aide de la commande « include ../Makefile_common ».

Dans ce makefile, nous avons gardé les instructions communes aux makefile puis les autres commandes tels que **PROJECTNAME, INC, LIBS, LIBDIR, TRG, DUMPTRG** qui sont propres aux autres makefile n'ont pas été inclus. De plus, dans notre makefile_common, nous avons inclus le clean plutôt que de le mettre dans les autres makefiles.

Nous avons alors vu dans ce rapport, d'une port les portions de code écrit suie aux différend travaux pratiques que nous avons choisis et évoquer la raison de nos choix. Puis nous avons décrit les différents makefile créer ainsi que les modifications que nous y avons apporter.

Références

[1]. Projet initial en ingénierie informatique et travail en équipe. [En ligne]. *http://www.groupes.polymtl.ca/inf1995/tp/tp8/*