

# NTYAM Kevin - ELOUECHRINE Amine.

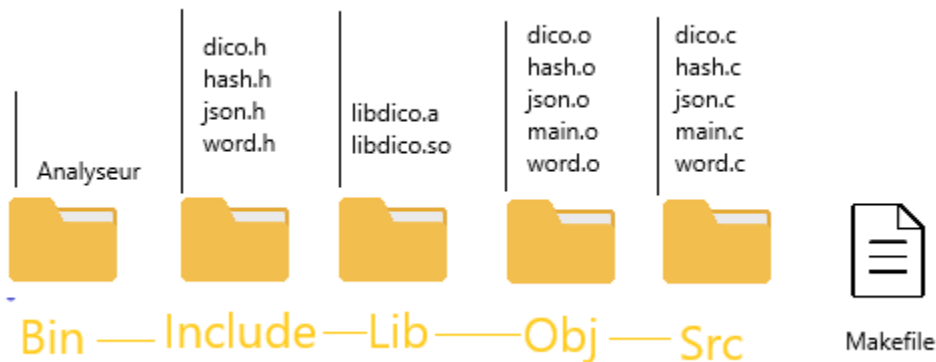
## Rapport MOCA - Semaine 2.

### Modifications effectuées

Actuellement, le projet est divisé en plusieurs sous-répertoires.

1. Le répertoire bin (court pour binary) contient l'exécutable en lui-même.
2. Le répertoire include contient l'ensemble des fichiers en-tête nécessaires aux fichiers source. Ce répertoire est référencé dans notre Makefile à l'aide de l'option "-I"
3. Le répertoire lib contient la bibliothèque utilisée par l'exécutable (statique et dynamique).
4. Le répertoire obj contient l'ensemble des fichiers objets générés par gcc.
5. Le répertoire src contient le code source de la bibliothèque mais également du programme principal. (A noter que le code source de la bibliothèque pourrait être supprimé dès que celles-ci sont générées mais cela dépasse le sujet.)

Cette structure est fortement inspirée de la hiérarchie de l'exercice donné lors du troisième cours.



# Création des bibliothèques

## Unique bibliothèque.

**Libdico:** Pour ce projet, nous avons choisi de créer une seule et unique bibliothèque. La raison est que l'ensemble des fonctions signatures contenues dans les fichiers sources se rapportent uniquement et principalement au programme principal. Ces fonctions ne pourraient pas être utilisées pour d'autres programmes, elles sont trop spécifiques. Par conséquent, une seule bibliothèque suffit.

## Annexe:

```
18 STATIC_LIB = $(LIB_DIR)/libdico.a
19 SHARED_LIB = $(LIB_DIR)/libdico.so
20
21 TARGET = analyseur
22
23 all: $(BIN_DIR)/$(TARGET)
24
25 # Options pour créer la bibliothèque statique et dynamique
26 $(STATIC_LIB): $(OBJJS)
27     @mkdir -p $(LIB_DIR)
28     ar r $@ $^
29
30 $(SHARED_LIB): $(OBJJS)
31     @mkdir -p $(LIB_DIR)
32     $(CC) -shared -o $@ $^
33
34 #Compilation avec option De la même manière qu'avec les options JSON, on définit une variable USE_STATIC qui quand elle vaut 1, utilise la
35     ifdef USE_STATIC première règle de compilation. La seconde sinon. Cette variables est définie dans le terminal au moment du make.
36 $(BIN_DIR)/$(TARGET): $(STATIC_LIB) $(SRC_DIR)/main.c
37     @mkdir -p $(BIN_DIR)
38     $(CC) $(CFLAGS) -o $@ $(SRC_DIR)/main.c -L$(LIB_DIR) -ldico $(LD_FLAGS)
39 else
40 $(BIN_DIR)/$(TARGET): $(SHARED_LIB) $(SRC_DIR)/main.c
41     @mkdir -p $(BIN_DIR)
42     $(CC) $(CFLAGS) -o $@ $(SRC_DIR)/main.c -L$(LIB_DIR) -ldico $(LD_FLAGS)
43 endif
44 (LD_FLAGS contient juste l'option "-lm" pour utiliser la bibliothèque mathématique.
45     [utile de préciser où elle se trouve avec -L car c'est une bibliothèque système.]
46 $(OBJ_DIR)/%.o:$(SRC_DIR)/%.c
47     @mkdir -p obj
48     $(CC) -c $(CFLAGS) $< -o $@
49
50 clean:
51     rm -f $(OBJJS) $(BIN_DIR)/$(TARGET) Un clean des plus classiques finalement.
```

Tout d'abord, on définit où vont se trouver nos librairies dans l'architecture du projet. (Ici, dans leur propre dossier lib)

Création de la bibliothèque statique. On se sert de la commande vue en cours pour la générer. (\$@: nom du but. \$^: toutes les options (ici fichiers objets))

Création de la bibliothèque dynamique. Ici, l'option "-fPIC" est donné en argument dans CFLAGS quand nous avons crée les objets. Il faut alors uniquement ajouter "-shared" pour créer la bibliothèque.

C'est ici que nous créons notre exécutable. Le but dépend de la valeur de USE\_STATIC. Nous créons notre dossier "bin". Puis nous executons gcc avec l'option "-L" qui permet d'indiquer où se trouve la bibliothèque et l'option "-l" qui doit avoir le nom de la bibliothèque et qui définit quelle bibliothèque doit être utilisée. Les bibliothèques sont toujours nommées "lib[nom].extension"

La création des objets reste identique. On crée notre dossier s'il n'existe pas déjà, puis on génère nos fichiers objets dans ce même dossier. (En utilisant OBJ\_DIR)

**Création de bibliothèque statique :** les commandes *ar* et *ranlib* permettent de créer l'archive voulue, par ex.

```
ar r libbindec.a utilitaires.o calculs.o
ranlib libbindec.a
```

**Création de bibliothèque partagée:** les fichiers doivent être compilés de façon à générer des fichiers objets de type *PIC* (Position Independent Code), puis la bibliothèque est produite par une compilation avec l'option *-shared*, par ex.

```
gcc -fPIC -c utilitaires.c
gcc -fPIC -c calculs.c
gcc -shared -o libbindec.so utilitaires.o calculs.o
```

Début du Makefile :

```
CC = gcc
CFLAGS = -Wall -Iinclude -fPIC
SRC_DIR=src
INC_DIR=include
OBJ_DIR=obj
BIN_DIR=bin
LIB_DIR=lib
LD_FLAGS=-lm
```

Ici, l'option *-fPIC* est nécessaire à la création d'une bibliothèque partagée.

L'option *-linclude* définit où le compilateur doit trouver les includes de tous les fichiers source. (Ceci nous permet d'éviter d'avoir des chemins relatifs qui peuvent être incorrects d'un système à un autre.)