UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

**Licence 3 Informatique**
**Info0601 – 2022/2023**
***Resp. :* Cyril Rabat**

# Makefile

## Exemple d'un `makefile` multi-répertoire

*Ce tutoriel présente un `makefile` qui peut être utilisé pour la création de projets en C. Il est présenté section par section.*

# 1 Configuration

Dans la partie configuration, il y a trois variables à renseigner. La première, nommée `EXEC`, correspond au nom de l'exécutable (pas d'extension sous *Linux*). S'il y a plusieurs exécutables, il faut spécifier les différents noms, séparés par un espace. La deuxième, nommée `OBJECTS`, correspond aux objets créés (ils doivent être indiqués avec l'extension .o et doivent être séparés par des espaces). Chaque objet correspond à un fichier source (.c) et éventuellement à un fichier d'en-tête (.h). La troisième, nommée `PROJECT_NAME`, est utilisée pour identifier le nom du projet en cours et pour l'archivage du projet.

Dans la variable `OBJECTS`, vous ne devez pas spécifier le nom de l'exécutable.

Les trois variables suivantes `OBJECTS_DIR`, `INCLUDE_DIR` et `BIN_DIR` indiquent respectivement les noms des répertoires pour les objets (.o), les includes (.h) et les exécutables.

Par exemple, supposons que nous ayons un programme nommé `test` (nous supposons donc l'existence du fichier `test.c` qui contient le `main`) qui utilise une méthode située dans `example.c` (avec un fichier `example.h` associé). La configuration est donc la suivante :

```
EXEC = test
OBJECTS = example.o
```

Autre exemple, supposons que nous ayons deux programmes `test1` et `test2` (correspondant respectivement aux fichiers C `test1.c` et `test2.c` qui contiennent chacun un `main`), la configuration est la suivante :

```
EXEC = test1 test2
OBJECTS =
```

La section `ARGUMENTS AND COMPILER` correspond au compilateur utilisé et aux options passées en paramètres. La variable `CC` est le nom du compilateur. Par défaut, il s'agit de `gcc`. La variable `CCFLAGS_STD` est la liste des options spécifiées au compilateur :

- `-Wall` : pour afficher tous les avertissements (*warning*).
- `-O3` : pour optimiser la compilation (la compilation est plus longue mais la vitesse d'exécution est très rapide).

- `-Werror` : pour considérer tous les avertissements comme des erreurs (dès qu'un avertissement est indiqué, la compilation échoue).

La variable `CCFLAGS_DEBUG` correspond aux options spécifiées au compilateur dans le cas du mode "DEBUG". La variable `CCLIBS` peut être utilisée pour lier des bibliothèques. Par exemple, avec la bibliothèque mathématiques (`math.h`), on peut écrire `CCLIBS = -lm`.

# 2 Compilation

Avant de compiler les programmes, il est nécessaire de renseigner au préalable les dépendances entre fichiers. Plutôt que de le faire manuellement, une règle spécifique est proposée : `depend`. Toutes les dépendances sont détectées et ajoutées à la fin du `makefile` dans la section `DEPENDANCIES`. Pour exécuter la règle, tapez simplement la commande suivante :

```
make depend
```

La règle `depend` doit être exécutée à chaque fois qu'un objet est ajoutée dans le `makefile`, ainsi qu'à chaque fois que de nouvelles inclusions sont ajoutées dans le projet (uniquement les inclusions de fichiers utilisateur, de fichiers d'en-tête).

Une fois les dépendances ajoutées, vous pouvez compiler les programmes en tapant la commande suivante :

```
make
```

# 3 Autres règles

Pour nettoyer le projet et supprimer tous les fichiers temporaires et les objets, vous pouvez utiliser la règle `clean`. Si le projet génère des fichiers spécifiques que l'on souhaite supprimer, il est possible de les ajouter à la suite. Pour exécuter la règle, tapez la commande suivante :

```
make clean
```

La règle `clean` a été proposée pour nettoyer tout le projet. Seuls les objets (fichiers .o) et les exécutables sont supprimés. Si les dépendances ont bien été faites (avec la règle `depend`), il ne faut pas utiliser cette règle systématiquement avant chaque compilation. Le cas échéant, le `makefile` ne servirait à rien.

Pour sauvegarder le projet, vous pouvez utiliser la règle `archive`. Une archive (au format .tar.gz) est créée contenant le contenu de tout le répertoire courant (à noter que cette règle appelle la règle `clean` au préalable). L'archive est placée dans le répertoire parent. Le nom de l'archive dépend du nom du projet et il est suivi de la date du jour. Pour exécuter la règle, tapez la commande suivante :

```
make archive
```

# 4 Débogage manuel

Lors du développement (en C), il est parfois nécessaire de mettre en place des parties qui devront disparaître dans la version finale du programme. Une solution consiste à définir une constante quelconque (ici, le choix a été fait d'utiliser `_DEBUG_`). Dans les sources, les parties à réaliser uniquement en mode débogage sont spécifiées comme suit :

```
#ifdef _DEBUG_
    /* Code à réaliser en mode débug */
#endif
```

Lors du développement, il est possible de définir la constante directement dans les sources (`#define _DEBUG_`), ou bien de la définir en argument du compilateur. Pour compiler en mode débogage à l'aide du `makefile`, il suffit de taper la commande suivante :

```
make debug
```

Les options de compilation définies dans la variable `CCFLAGS_DEBUG` du `makefile` seront ajoutées lors de la compilation. Avec `gcc`, il suffit d'ajouter `-D _DEBUG_`.

Si le programme a déjà été compilé et que vous souhaitez réaliser une compilation en mode débogage, il faudra d'abord faire appel à la règle `clean`.