# Makefile

# Shockwave

La compilation est une étape qui peut s'avérer fastidieuse, d'autant plus que l'on souhaite inclure plusieurs fichiers sources. Il est peu pratique d'avoir à répéter la commande de compilation pour chacun de ces fichiers. Pour pallier à cela, on utilise des **makefiles**, des fichiers spéciaux qui, combinés avec la commande **make**, rendent automatique la compilation d'un projet.

# make

Quand on lance make sur la console, ce programme va chercher un fichier au nom de makefile dans le dossier courant où on a placé tous nos fichiers sources.

## Chaîne de production

— Etape 1 : compilation
Le compilateur produit un fichier **objet** à partir d'un fichier source **.c**.

— Etape 2 : édition de liens
Il s'agit de produire un fichier **exécutable** à partir d'un fichier **objet**.

# Exemple

Je vais reprendre ici l'exemple du cours où l'on a un fichier source principal useComplexe.c nécessitant le fichier source complexe.c qui fait partie d'un module (complexe.h et complexe.c).

Voici les étapes à suivre sans makefile :

```
gcc -Wall -c complexe.c
gcc -Wall -c useComplexe.c
gcc -Wall useComplexe.o complexe.o -lm -o useComplexe
```

L'option **-c** nécessaire indique au compilateur de seulement compiler et de ne pas faire d'édition de lien. On compile donc les deux fichiers sources complexe.c et useComplexe.c suite à quoi, on obtient complexe.o et useComplexe.o. Puis dans la 3ème ligne, on crée un exécutable que l'on nomme useComplexe grâce à l'option **-o**. Remarquez qu'on écrit les deux fichiers objets dans la même ligne. L'option **-l** permet de spécifier qu'on utilise une bibliothèque qui n'est pas dans le dossier standar **lib**. Ici il s'agit de la bibliothèque mathématique **m**.

Lorsque le nombre de modules à inclure est plus grand, compiler sans makefile devient rapidement compliqué. Voyons maintenant comment utiliser un makefile.

# Makefile

On crée un fichier texte que l'on nomme makefile (pas d'extension .txt !). Voici la structure de base d'un fichier makefile :

cible principale : dépendances
<TAB> action
cible : dépendances
<TAB> action

> La cible principale est la première cible apparaissant dans le fichier makefile. Les cibles sont les fichier objets **.o** et l'exécutable que l'on souhaite créer.
>
> Ces cibles dépendent de fichiers sources **.c** et de fichiers d'en-tête **.h**.
>
> Après avoir spécifier une cible et ces dépendances, on saute une ligne et on tabule. Puis on écrit ce que le compilateur doit faire pour fabriquer la cible.
>
> Il est possible de rajouter des commentaires avec le symbole #.

Voici le fichier makefile de notre exemple :

```
useComplexe : useComplexe.o complexe.o
    gcc -Wall -lm useComplexe.o complexe.o -o useComplexe

useComplexe.o : useComplexe.c complexe.h
    gcc -Wall -c useComplexe.c

complexe.o : complexe.c complexe.h
    gcc -Wall -c complexe.c
```

Quand on tape make sur la console, le programme va chercher à construire la cible principale, c'est-à-dire l'exécutable useComplexe. Or ici, useComplexe dépend de useComplexe.o et complexe.o. Ainsi, make va d'abord construire les dépendances useComplexe.o et complexe.o. Il cherche alors dans le makefile des cibles du même nom.

L'ordre n'importe pas, mais la première cible est la cible principale que le make va chercher à construire.

Pour faire useComplexe.o, on a besoin de useComplexe.c auquel on a inclu le fichier d'en-tête complexe.h, d'où la nécessité de le préciser comme dépendance. La commande sur la ligne suivante s'applique et on produit notre fichier objet recherché.

De même pour complexe.o.

Finalement, la cible principale est produite.

# Variables

On peut utiliser des variables dans le makefile afin de spécifier une bonne fois pour toute la liste des fichiers objet, les options de compilation, etc. Pour définir une variable, on écrit son nom suivi du symbole **=** puis de la valeur. On utilise une variable ainsi : $(variable) ou ${variable}.

Voyons comment utiliser les variables avec l'exemple précédent. Le nouveau fichier makefile est :

```
# on place nos fichiers objets dans la variable OBJETS
OBJETS = useComplexe.o complexe.o

# on place les fichiers du module dans la variable MODULE
MODULE = complexe.c complexe.h

# on place le nom du compilateur dans la variable CC
CC = gcc

# on place les options de compilation dans la variable CFLAGS
CFLAGS = -Wall -c

useComplexe : $(OBJETS)
    $(CC) -Wall -lm $(OBJETS) -o useComplexe

useComplexe.o : useComplexe.c complexe.h
    $(CC) $(CFLAGS) useComplexe.c

complexe.o : $(MODULE)
    $(CC) $(CFLAGS) complexe.c
```