Maintenance de Logiciels

Travaux Dirigés N° 01

Make

Préliminaires :

Make est un logiciel qui appartient à la famille des *moteurs de production*. Sa fonctionnalité principale consiste à rendre automatique un ensemble déterminé d'actions (compilation, édition des liens...) décrites dans un fichier de script appelé *Makefile*.
La différence principale entre Make et les autres interpréteurs de script classiques (sh, bash...) réside dans le fait que Make exécute les actions uniquement quand elles sont nécessaires. Pour ce faire, il analyse les dépendance entre les éléments à produire (les cibles) et les éléments utilisés pour la production (les dépendances) et ne produit les cibles que si ces dernières sont moins récentes que leurs dépendances.
Dans un projet logiciel, Make contrôle la génération d'exécutables et d'autres éléments comme les bibliothèques de fonctions. Néanmoins, il peut être utilisé pour des fins différentes du développement logiciel comme par exemple la production de documents PDF à partir de sources écrites en Latex.

L'objectif de ces travaux dirigés est de vous initier avec l'outil **GNU Make** qui est la variante de Make présente sur la grande majorité des systèmes GNU/Linux. Le TD contient 2 exercices.

Exercice 01 : Makefile pour un programme C

Cet exercice consiste à créer un fichier Makefile permettant la production automatisée d'un programme écrit dans le langage C et constitué de plusieurs fichiers sources (`.c`) et d'entêtes (`.h`).

L'application que l'on se propose de construire durant ce TD est un programme permettant l'utilisation de fonctionnalités arithmétiques sur les entiers. Ces fonctions seront distribuées sur deux bibliothèques :

- Fonctions arithmétiques basiques :
  - PGCD de deux entiers
  - PPCM de deux entiers
  - Ensemble des diviseurs d'un entier
- Fonctions arithmétiques sophistiqués : ce sont des fonctions qui utilisent (entre-autres) les fonctions de la bibliothèque basique pour réaliser des fonctions plus avancées :
  - Somme des diviseurs d'un entier
  - Test si un nombre est parfait ou non (un nombre parfait est un nombre égal à la somme de ses diviseurs).
  - Test si un entier est premier ou pas

L'application comporte aussi une fonction principale constituant le point d'entrée du programme. Cette fonction demande à l'utilisateur de choisir une fonction mathématique qu'il veux exécuter, ses

paramètres, ensuite affiche le résultat.

Notre programme est constitué donc des fichiers sources suivants :

- Bibliothèque basique : `fonctions_basiques.h` et `fonctions_basiques.c`
- Bibliothèque avancée : `fonctions_avancees.h` et `fonctions_avancees.c`
- Fonction principale : `arithm.c`

Le Makefile que l'on se propose de construire permet de réaliser les tâches suivantes :

1. La compilation des fichiers sources
2. L'édition de liens de l'application (exécutable `arithm`)
3. Le nettoyage des fichiers objets et exécutables
4. L'installation de l'exécutable dans un endroit bien déterminé
5. L'archivage pour distribution

Question 1 :

Identifier les différents modules du programme.

Créer les deux fichiers `fonctions_basiques.h` et `fonctions_avancees.h`. Chacun de ces deux fichiers contient les **signatures** des fonctions contenues dans la bibliothèque correspondante. Toutes les signatures doivent être précédées par le mot clé **`extern`** afin de permettre la compilation séparée des différents modules du programme. Exemple :

```
extern int pgcd (int, int);
```

Question 2 :

Comme l'objectif principal du TD n'est pas d'implanter les fonctions mathématiques mais plutôt de créer un fichier *Makefile*, les corps de ces fonctions seront, dans un premier temps, vides (contenant le minimum permettant une compilation correcte).

Créer des corps minimaux pour les fonctions des deux bibliothèques mathématiques dans les fichiers `fonctions_basiques.c`, `fonctions_avancees.c` et `arithm.c`.

Notes :

- Le fichier `fonctions_basiques.c` utilise les fonctions mathématiques de la bibliothèque standard (`math.h`)
- Le fichier `fonctions_avancees.c` utilise les fonction du fichier `fonctions_basiques.h`
- Le programme principal utilise les fonctions des deux bibliothèques.

Question 3 :

Identifier les dépendances qui existent entre les différents constituants du programme.

Question 4 :

En partant du *Makefile* incomplet donné dans l'annexe située à la fin de ce document, créer un fichier Makefile contenant les éléments suivants :

- Une règle d'inférence permettant de générer des fichiers objet (`.o`) à partir des fichiers sources (`.c`)
- Une cible permettant l'édition des liens de l'exécutable `arithm` à partir des fichiers objets
- L'expression des dépendances trouvées dans la question précédente

Tester la compilation du programme

Question 5 :

Ajouter les cibles (et leur éventuelles dépendances) permettant la réalisation du reste des fonctionnalités du *Makefile* :

1. Le nettoyage des fichiers objets et exécutables (cible `clean`)
2. L'installation de l'exécutable dans un endroit bien déterminé (cible `install`)
3. L'archivage pour distribution (cible `dist`)

Question 6 :

Tester le bon fonctionnement des différentes cibles et la bonne gestion des dépendances (en modifiant un fichier puis en vérifiant que tous le fichiers qui en dépendent sont recompilés).

Exercice 02 : Sources et objets dans des dossiers séparés.

En s'inspirant de l'exercice 1, créer un *Makefile* pour un programme C qui effectue les mêmes fonctionnalités du programme `arithm` avec la seule différence que les fichiers sources se trouvent dans un sous-dossier `src` et les fichiers objets sont créés dans un sous-dossier `obj`.

Note :

- Les fichiers `.class` doivent être produits dans un répertoire séparé nommé **`classes`**.

Annexe : Makefile incomplet pour le programme C

```
#
# Makefile
#

# Nom de l'application
APPLICATION = # TODO : Ajouter le nom de l'exétable

# Compilateur et options de compilation et d'édition des liens
CC = gcc
LD = $(CC)

CFLAGS = -c -g -Wall
LDFLAGS = -g -lm

# Paramètres d'installation
prefix = /tmp

# Constituants de l'application
OBJS = # TODO: liste des fichiers objets

# Règle d'inférence, construction des .o a partir des .c
# TODO: ajouter la règle d'inférence

# Cibles et leurs actions

all: $(APPLICATIONS)

$(APPLICATION): # TODO: à compléter

# TODO: ajouter les dépendances
```