Projet shell : generateMakefile.sh
Système et programmation système

Julien BERNARD
Université de Franche-Comté

Licence 2 Informatique
2015 – 2016

Spécifications

Ce premier projet centré sur la programmation shell est prévu sur **deux séances**. N'hésitez pas à le compléter entre les deux séances.

Le but du projet est d'écrire un script `generateMakefile.sh` qui, comme son nom l'indique, génère un Makefile à partir de fichiers sources contenus dans un répertoire. Il prend un argument obligatoire, le nom du programme, suivi d'un argument optionnel, le répertoire qui contient les sources (par défaut, le répertoire courant). Voici les suppositions que vous pouvez faire :

— tous les fichiers `.c` et `.h` sont dans le même répertoire, et il n'y a aucun sous-répertoire ;
— tous les fichiers sources servent à construire un programme unique dont le nom est donné sur la ligne de commande ;
— les noms des fichiers sources ne contiennent pas d'espaces, mais uniquement des lettres et des chiffres.

Réalisation

Exercice 1 : Cas de tests

Avant toute chose, il faut préparer un ou deux cas de tests, c'est-à-dire d'une part des fichiers `.c` et `.h` qui serviront pour contruire le `Makefile` et d'autre part, le fichier `Makefile` qu'on cherche à construire de manière à voir si le fichier généré est identique (fonctionnellement) au fichier voulu.

Question 1.1 Compléter le TP «Interface et page de manuel» en écrivant un Makefile complet pour construire le programme `lcg_demo`.

Exercice 2 : Arguments de la ligne de commande

Dans cette partie, nous allons traiter la ligne de commande à travers les variables spéciales `$0`, `$1` et `$2`, ainsi que la variable spéciale `$#` qui contient le nombre d'arguments (sans le nom du script).

Question 2.1 Écrire une fonction `usage` qui affiche une aide sur l'utilisation de ce script. Par exemple :

```
Usage: generateMakefile.sh name [directory]
where:
        name is the name of the program
        directory is the source directory
```

Question 2.2 Afficher l'aide et quitter s'il n'y a aucun argument.

Question 2.3 S'il y a plus d'un arguments, mettre le premier argument dans une variable `PROGRAM` et s'il y a plus de deux arguments, mettre le second arguments dans une variable `ROOTDIR` (qui contiendra sinon le répertoire courant obtenu avec `pwd(1)`). Vérifier que ce second argument est bien un répertoire valide, sinon afficher l'aide et quitter.

Question 2.4 Changer de répertoire pour le répertoire `ROOTDIR`.

Exercice 3 : Récupération des en-têtes locaux d'un fichier

Dans cette partie, nous allons voir comment récupérer la liste des en-têtes locaux d'un fichier, c'est-à-dire les en-têtes qui sont inclus dans le fichier à l'aide de guillemets double (et pas avec des chevrons), le tout en une seule ligne de commande. Cette liste sera une chaîne de caractères des différents fichiers, séparés par des espaces.

Question 3.1 Sélectionner les lignes du fichier qui contiennent des directives d'inclusion.

Question 3.2 Enlever les directives d'inclusions avec des chevrons. Pour cela, on s'intéressera à l'option `-v` de `grep(1)`.

Question 3.3 Enlever tout ce qui n'est pas le fichier : la directive, les espaces, les guillemets double. Pour cela, on utilisera `sed(1)` en remplaçant les différentes parties par la chaîne vide.

Question 3.4 Transformer les passages à la ligne en espace grâce à la commande `tr(1)`.

Exercice 4 : Calcul récursif des dépendances

Dans cette partie, nous allons calculer les dépendances d'un fichier objet. Ces dépendances contiennent bien évidemment le fichier `.c` correspondant, mais aussi les en-têtes inclus par le fichier `.c`. Il faut également prendre en compte les en-têtes inclus eux-mêmes dans les en-têtes, et ainsi de suite. Il va donc falloir établir cette liste récursivement.

L'idée est de partir d'un ensemble de fichier A (qui contient au départ uniquement le fichier `.c`) puis de calculer $\text{dep}(A)$, les fichiers d'en-têtes inclus par

les fichiers de l'ensemble A. On a alors l'ensemble $B = A \cup \text{dep}(A)$. Si $B = A$ alors, on a fini, sinon on recommence avec B.

La difficulté est de calculer l'union de deux ensembles, c'est-à-dire que le résultat ne contienne pas deux fichiers identiques. Comme dans la partie précédente, une liste sera dans une chaîne de caractères.

Question 4.1 Pour un ensemble de fichiers contenu dans une chaîne, calculer toutes les dépendances dans une variable `DEPS`, sans se préoccuper si elles sont déjà dans l'ensemble ou pas.

Question 4.2 Éliminer les doublons de `DEPS`. Pour cela, nous allons utiliser `uniq(1)`, mais il faut d'abord transformer notre liste de fichiers séparés par des espaces en liste de fichiers séparée par des retour à la ligne grâce à `tr(1)`.

Question 4.3 Répéter cette opération tant que l'ensemble de fichiers n'est pas stable.

Exercice 5 : Génération du `Makefile`

Dans cette partie, il ne vous reste plus qu'à générer le `Makefile` proprement dit grâce à tout ce que vous avez fait précédemment.

Question 5.1 Vérifier qu'il y a bien des fichiers `.c` dans le répertoire donné à l'aide de `ls(1)`. Dans le cas contraire, afficher l'aide et quitter.

Question 5.2 Écrire un commentaire dans le `Makefile` pour prévenir qu'il a été généré. Par exemple :

```
# Generated by generateMakefile.sh
```

Question 5.3 Générer la règle `all` à l'aide de la variable `PROGRAM`.

Question 5.4 Générer les règles de compilation en fichier objet pour chaque fichier `.c`, en calculant les dépendances du fichier objet.

Question 5.5 Générer la règle de construction du programme final à partir de tous les fichiers objets.

Question 5.6 Générer la règle `clean` et la règle `mrproper`

Question 5.7 Ajouter les variables `CFLAGS` et `LDFLAGS` dans les commandes de compilation, sans les définir auparavant. Ceci permettra d'appeler votre script de la manière suivante :

```
$ CFLAGS="-Wall -O2" ./generateMakefile.sh
```

Exercice 6 : Bonus

Cette partie est facultative, elle ne doit être abordée qu'une fois les parties obligatoires terminées.

Question 6.1 Prendre en compte les répertoires avec des sous-répertoires.

Évaluation

Vous devrez déposer votre fichier dans le dépôt MOODLE prévu à cet effet avant la date indiquée.