TP1 : Construction de programmes

1 Objectifs

- ➢ Comprendre le fonctionnement d'un makefile explicite (cible, dépendance, mise à jour), puis des règles implicites.
- ➢ Créer un makefile par adaptation.
- ➢ Expérimenter quelques directives.
- ➢ Utiliser les différents types de variables.
- ➢ Créer un makefile avec des règles implicites et savoir créer ces règles.
- ➢ Ce TP devrait durer une séance.

2 Premier makefile explicite pour l'outil de conversion

Nous allons construire et modifier progressivement un makefile pour automatiser au mieux la construction de l'outil de conversion, implémenté en C et vu en cours.

- ➢ Créez un répertoire pour ce TP.
- ➢ Récupérez l'archive conversion.tgz et décompactez-la dans un répertoire créé pour cette série d'exercice (conversion semble un nom intéressant).
  http://deptinfo.unice.fr/twiki/bin/view/Linfo/GLEnvtProg0708TP2

2.1 L'invocation de `make`

① Etudiez le fichier `Makefile.1` et

② Faire construire l'application à l'aide de `make` sans modifier le makefile !

③ Que faudrait-il faire pour que, par défaut, l'application soit construite sans donner son nom de cible ?

Faites-le *proprement* dans un fichier `Makefile.2`.

④ Que faudrait-il faire pour que tout se construise simplement en tapant make (sans paramètres) ?

Faire cela avec un lien sur `Makefile.2` !

⑤ Améliorer la présentation du fichier `Makefile.2`.

Ajoutez des commentaires (vous le ferez systématiquement à partir de maintenant), par exemple pour expliquer les dépendances de compilation séparée et la reliure finale.

2.2 Gestion des fichiers d'en-têtes

① Modifiez le fichier `convertir.h` pour que le taux passe à 6.55957.

Relancez `make` : Que se passe-t-il ? Est-ce conforme au comportement espéré ?

② Créez un `Makefile.3` qui gère correctement les .h

Concernant ces nouvelles dépendances, vous les recherchez manuellement et ensuite vous vérifiez avec la "bonne" option de gcc.

③ **Sans faire exécuter les commandes de compilation**, tester votre makefile.

En modifiant la date des divers fichiers .h avec la commande `touch` , vérifier que l'approche "en faire le minimum" est satisfaite par `make` .

④ Il existe une option de `make` qui permet de faire un `touch` sur les cibles du makefile. Laquelle ?

Quelle est la différence avec la commande `touch` ?

A quoi cette option peut elle servir ?

⑤ Faire procéder à la compilation globale.

2.3 Faire le ménage

① Ajoutez une cible `clean` qui efface tous les fichiers inutiles avec l'action suivante `/bin/rm *.o *.bak *  core`

② Que se passe-t-il si vous lancez deux fois de suite la commande `make clean` ?

Rendez cette cible robuste : si le fichier est protégé en écriture et/ou si le fichier n'existe pas !

③ Ajoutez à ce makefile une règle `veryclean` qui, en plus d'éliminer les fichiers qui ne vous intéressent pas, efface aussi l'exécutable.

Utilisez la règle `clean` pour créer cette nouvelle cible.

3 Création d'un nouveau makefile

➢ Créez un répertoire `pacman` pour la suite de ce TP.

➢ Récupérez l'archive `leretourdepacman.tgz` et décompactez la.

① Si ca peut vous aider, recopiez la meilleure version de votre Makefile pour la *conversion* et adaptez la pour *pacman*.
La seule information dont vous avez besoin est que "pacman est une application graphique construite en utilisant la bibliothèque `libX11`".

② Testez que les dépendances (.c/.h) sont correctement gérées à l'aide de `touch` .

③ Ajoutez une cible `install` qui déplace l'exécutable créé dans votre répertoire bin : `$(PROJ)/bin`

④ Par défaut, le makefile construit et installe `pacman` !

3.1 Manipulation des variables d'environnement

➢ Créez un makefile dont la cible par défaut affiche la valeur des variables `SHELL` , `OSTYPE` et `USER` .

➢ Regardez l'effet produit en fonction du caractère d'exportation de ces variables (comparer le résultat à l'aide de set et export dans votre shell)

➢ Modifiez ces valeurs à l'appel du makefile (par exemple `make USER=pipo`).

➢ Regardez l'effet produit, à l'exécution de make, puis dans le shell qui a lancé make.

➢ Qu'en déduisez-vous ?